Universidad del Biobío
Facultad de Ciencias Empresariales
Departamento de Sistemas de Información Profesor Guía: Claudio Gutiérrez S.

ALGORITMO DE CLUSTERING EN LÍNEA, PARA LA RECUPERACIÓN DE LA INFORMACIÓN.

"Informe Final de Proyecto de Título"

Martes, 09 de Mayo de 2017

Alumnos:
Frederick Lara Sánchez
Delia Moncada García

Ingeniería Civil en Informática

RESUMEN

La recuperación de la información tiene como propósito satisfacer una necesidad de información de un usuario.

En la recuperación de la información se ha incorporado el clustering de documentos con el propósito de mejorar la eficacia de los sistemas de recuperación de información. La hipótesis de clustering confirma que los documentos relevantes aparecen en los mismos grupos cuando éstos tienden a ser similares entre sí. Sin embargo, en investigaciones realizadas anteriormente no se ha podido concluir respecto a si el clustering dinámico de documentos trae mejoras.

Este trabajo tiene como objetivo principal mejorar la precisión de los documentos recuperados, aplicados en un contexto dinámico, con la motivación de investigar la efectividad del clustering de documentos. Es por esto, que se llevan a cabo algunos supuestos los cuales tratan de la forma dinámica en la que se realiza el clustering de documentos con respecto a una consulta.

En este proyecto de título se lleva a cabo la agrupación con el fin de investigar el comportamiento de los principales algoritmos de clustering, al realizar consultas con un grupo de documentos relevantes. Se realizan dos tipos de experimentos, los cuales se llevan a cabo con dos conjuntos de documentos, que se denominan como documentos antiguos y documentos nuevos. En primer lugar se realizaron los experimentos relacionados con la precisión de los documentos y en segundo lugar se analizó el comportamiento de tres algoritmos de clustering (Single Link, Complete Link y Average Link).

Se tiene como enfoque principal el clustering en línea, realizando una investigación analítica con una serie de cosas relacionadas con la eficacia de la recuperación en un contexto dinámico. En esta tesis la recuperación en línea se emplea sobre la base de su potencial para mejorar la eficacia de la agrupación de documentos y tener un mejor resultado en los sistemas de recuperación de información.

Los resultados obtenidos por medio de los experimentos con respecto a la precisión obtenida con la recuperación de documentos antiguos y la recuperación donde se unen documentos antiguos con documentos nuevos, demostró que existe un momento en el cual, la precisión siempre mejora, a medida que se van incorporando nuevos documentos.

En la segunda parte de los experimentos, se proporciona evidencia con respecto a la agrupación realizada, donde se toma en cuenta la cantidad de documentos relevantes al recorrer el clúster generado sólo con los documentos antiguos y la cantidad de documentos relevantes visitados al recorrer el clúster construido con la unión de los documentos antiguos y nuevos, dichos clústeres fueron generados con cada uno de los algoritmos de clustering, para así poder observar cuál de los tres algoritmos obtuvo los mejores resultados.

Índice General

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

Introducción

La Recuperación de la información (RI) envuelve tareas que tienen como propósito entregar información relevante a los usuarios. Entre las tareas que se efectúan en la recuperación de la información se puede encontrar la organización, la búsqueda, el análisis, y el almacenamiento de la información. En primer lugar, los usuarios presentan requerimientos de información que se llevan a cabo por medio de consultas, con estas consultas se busca encontrar la información necesaria para cumplir con las expectativas del usuario. Es factible encontrar diferentes fuentes de información tales como vídeo, audio, archivos XML, por mencionar algunos. Generalmente, la información se puede encontrar en un documento o en un conjunto de documentos. Por otro lado, la relevancia de un documento o de un conjunto de documentos es proporcionado por los usuarios. Dicha relevancia entregada por los usuarios es conocido como juicios de los usuarios, de este modo, dada una consulta y un conjunto de documentos relacionados a dicha consulta, cada uno de estos documentos puede ser clasificado por los usuarios como "relevante" o "no relevante" según la necesidad de información del usuario. En la comunidad de RI al conjunto de documentos, conjunto de consultas y sus juicios de usuarios se le denomina colección de pruebas. En la actualidad es posible encontrar la información plasmada en millones de documentos, es por esto, que se hace necesario automatizar las tareas mencionadas anteriormente. Los Sistemas de Recuperación de Información (SRI) brindan el apoyo necesario al usuario, con el fin de ayudar a localizar los documentos que tengan el potencial de satisfacer la necesidad de información del usuario. Un SRI proporciona un conjunto de documentos clasificados que están relacionados a la consulta enviada por un usuario. Comúnmente, la lista de documentos proporcionada por el SRI es clasificada por una puntuación decreciente. Esta puntuación representa la similaridad entre los documentos y la consulta calculado por el SRI(la similaridad puede ser aplicada entre las consultas, así como entre los documentos). Sin embargo, todos los documentos recuperados no satisfacen la necesidad de información del usuario. En palabras simples, no todos los documentos que aparecen en la lista son relevantes para el usuario. Además, la posición de los documentos relevantes en la lista es muy importante para el usuario. Por lo tanto, una buena situación es cuando los documentos relevantes aparecen juntos y al principio de la lista, es decir los documentos relevantes no

deben estar muy separados entre sí. Esta propiedad es denominada precisión en RI. Se puede ejemplificar esta situación con el siguiente caso: para una consulta *Q*, existe un subconjunto de documentos relevantes (*SDR)* para un conjunto de documentos. Un SRI R_1 proporciona una lista de documentos donde el *SDR* aparece en la parte superior de la lista. De manera similar, otro SRI R_2 entrega una lista de documentos donde el *SDR* aparece en la parte inferior de la lista. Al realizar una comparación entre los dos sistemas, R_1 otorga una mejor precisión que R_2, porque el mismo *SDR* aparece en la parte superior. Un desafío importante y difícil para un SRI es proporcionar la mejor precisión posible.

Es posible encontrar en la literatura una gran cantidad de contribuciones relacionadas con RI, donde se abordan diferentes perspectivas como funciones de matching[1], indexación[2], entre otros.

El clustering y los archivos invertidos son las estructuras de datos más utilizadas para construir los índices y el acceso a los documentos. Los archivos invertidos permiten una respuesta eficiente a las consultas de palabras claves, mientras que el clustering permite crear grupos de documentos similares. El clustering en RI se utiliza con el fin de mejorar la eficiencia (en tiempo) y efectividad (en precisión) de los SRI. Es posible identificar dos principales categorías de clustering: clustering estático y clustering post-recuperación (dinámico). El clustering estático es la aplicación tradicional del método clúster en una colección de documentos. En cambio, el clustering dinámico incluye información desde la consulta hacia el clustering de documentos. La similaridad entre los documentos está relacionada con la intersección entre los términos de los documentos. Normalmente, las funciones de similaridad como Jaccard o coseno se utilizan para medir la similaridad entre los documentos (Salton y McGill, 1983).

El clustering de documentos ha sido tradicionalmente aplicado de manera estática, a toda una colección de documentos, antes de realizar la consulta (clustering estático). Una aplicación alternativa del clustering es aplicar los algoritmos de clustering en línea o de manera dinámica, esto es debido a que el número de documentos puede aumentar o disminuir en cortos intervalos de tiempo, por lo que una inadecuada actualización o aplicación de los algoritmos provocaría una pérdida de información (por ejemplo documentos no

[1] La función de matching pretende estimar si un documento es relevante de acuerdo a la consulta enviada por un usuario.
[2] El proceso de indexación corresponde a la representación y el almacenamiento de documentos

considerados). Es por esto que nuestra investigación toma sentido, ya que se hace necesario conocer el momento en que es conveniente rehacer el clúster, para considerar la mayor cantidad de documentos al responder una consulta proporcionada por un usuario.

Para este proyecto de título, se pudieron identificar los siguientes problemas:

1. El proceso de clustering dinámico, incluye agregar documentos después del clustering inicial. Por lo tanto, cuando se agrega un nuevo documento a la lista, se hace necesario analizar detalladamente como varían los pesos de cada uno de los términos de los documentos relevantes, se realiza este proceso para generar los clústeres con los documentos que se incorporaron.
2. Cuando los documentos son relevantes, estos documentos proporcionan términos relevantes, que son utilizados para realizar el proceso de similaridad entre los documentos, por lo que se hace necesario realizar una lista donde se guardarán los términos relevantes, con el fin de utilizar esta lista cuando se incorpora un nuevo documento.
3. Cuando hablamos de clustering dinámico, nos referimos a que por cada incorporación de un documento nuevo, se realiza una nueva matriz de similaridad, debido a que los pesos de los términos varían con cada documento nuevo.

1.2 Objetivos del proyecto de título

El objetivo de este trabajo es mejorar la precisión de los documentos recuperados en un contexto dinámico. Para lograr este objetivo, este trabajo realiza un análisis profundo y una implementación de algoritmos de clustering, los cuales serán utilizados para realizar pruebas experimentales y analizar los resultados obtenidos.

Para cumplir con lo mencionado anteriormente, las principales contribuciones de este proyecto de título son:

1. Estudiar los conceptos envueltos en recuperación de la información e identificar la problemática de la recuperación de documentos utilizando algoritmos de clustering en un contexto dinámico.
2. El análisis de los principales algoritmos existentes de almacenamiento de clustering, con el fin de utilizar estos algoritmos a través de un algoritmo en línea.

3. Implementar cuatro algoritmos de clustering, los cuales serán utilizados con el fin de comparar y obtener resultados.
4. Analizar detalladamente los resultados empíricos.
5. Usar un algoritmo probabilístico con el fin de simular los juicios de usuarios para el conjunto de documentos nuevos (que se van incorporando constantemente en el tiempo).

1.1.1 Simulación de conjunto de documentos

Para realizar la simulación del conjunto de documentos, se utilizará un framework propuesto por Gutiérrez-Soto (2016), el cual permitirá simular un clúster de documentos, al cual se le pueden ir aumentando la cantidad de documentos para así simular un contexto dinámico.

1.1.2 Clustering de documentos

Generalmente, el clustering en el contexto de RI se utiliza para almacenar documentos (Tombros y van Rijsbergen (2004)). Sin embargo, el almacenamiento de los documentos relevantes puede proporcionar un contexto dinámico. Por ésto, en esta investigación se busca implementar y evaluar 4 algoritmos de clustering en un contexto dinámico para mejorar la precisión del clustering de documentos.

1.2 Descripción de capítulos

Este proyecto de título está compuesto de 5 capítulos (incluyendo este capítulo). A continuación se describe el contenido de los siguientes capítulos.

Capítulo 2: En este capítulo, se tratarán algunos conceptos relacionados con la recuperación de la información, estos conceptos serán utilizados en el trabajo experimental que se expondrán en este proyecto. El propósito es establecer una terminología básica de los temas que se utilizarán en los siguientes capítulos.

Capítulo 3: En este capítulo, se entregará una detallada revisión de trabajos realizados anteriormente relacionados con el clustering de documentos. Esta revisión tendrá como propósito analizar los pasos del proceso de clustering. Además se entregarán todos los antecedentes que sean necesarios para la aplicación del clustering jerárquico de documentos en RI y discutir los principales aspectos del proceso de clustering que sean relevantes.

Capítulo 4: En este capítulo, se realizarán y analizarán los escenarios experimentales y las pruebas necesarias para cumplir con los objetivos, además, se analizarán los resultados obtenidos en los escenarios experimentales realizando conclusiones respecto a estos.

Capítulo 5: En este capítulo, se presentarán las principales contribuciones de este proyecto, tomando en cuenta los objetivos, además de las contribuciones que se pueden llevar a cabo a futuro.

CAPÍTULO 2

Recuperación de la Información (RI)

La recuperación de la información es una disciplina que se ocupa de la organización, análisis, estructuración, almacenamiento y la búsqueda de la información. En la literatura se pueden encontrar diferentes definiciones respecto a la recuperación de la información. Una de estas definiciones es la realizada por Baeza-Yates y Ribeiro-Neto(1999): "... el sistema de RI debe de alguna manera interpretar el contenido de los elementos de información (documentos) de una colección y clasificarlos de acuerdo con un grado de relevancia para la consulta del usuario. Esta interpretación del contenido de un documento implica extraer información sintáctica y semántica del texto del documento... ". De la definición anterior, se pueden destacar tres aspectos importantes. El primer aspecto, indica que existe un usuario el cual tiene una necesidad de información, luego se tiene un conjunto de documentos los cuales son comparados, y en último lugar, una lista de documentos entregados por un SRI respondiendo a la consulta del usuario. De manera que, el objetivo que tiene un SRI es entregar un grupo de documentos que satisfaga la necesidad de información que tiene el usuario. El usuario por medio de una consulta expresa su necesidad de buscar información, y esta consulta es enviada al sistema. Luego, para procesar esta consulta, el SRI realiza una representación interna de la misma. Con la representación de la consulta, se realiza la comparación con el conjunto de documentos. Finalmente como resultado, al usuario se le entrega una lista con los documentos ordenados de manera decreciente por su similaridad con la consulta. Es por esto, que el usuario puede verificar si los documentos de la lista son realmente relevantes para él, si los documentos en la lista no son relevantes, se puede realizar una nueva consulta o reformular la consulta dada anteriormente.

En la actualidad existen diferentes modelos de RI, donde se puede encontrar la forma en que se realiza el proceso de matching entre consultas y documentos, así como también, la descripción de cómo representarlos.

Los modelos más difundidos en RI son el Booleano, Espacio Vectorial (Salton, 1971, Salton et al., 1975), Probabilística (Robertson et al., 1981) y Lógico (Van Rijsbergen, 1986).

El modelo Espacio Vectorial, representa documentos y consultas a través de vectores, basados en las frecuencias de términos en los documentos. La medida de similaridad entre una

consulta y un documento corresponde al coseno entre los términos de la consulta y los términos del documento.

El objetivo de los primeros SRI fue la búsqueda automatizada del material de bibliográfico por los usuarios, ésto con el fin de ser un apoyo a esta búsqueda. Con el gran crecimiento de la información disponible en formato electrónico, fue necesario ampliar los SRI en otros ámbitos. Es por esto, que en la literatura es posible encontrar una amplia gama de investigaciones que dependen de tipos heterogéneos de información.

Uno de los medios más populares que utilizan los usuarios para buscar información es la World Wide Web. En el internet podemos ver reflejados los SRI como motores de búsqueda web, que indexan una gran cantidad de información y promueven un acceso a la información de forma sencilla a los usuarios.

En la actualidad, la cantidad de trabajos enfocados en explotar las características que caracterizan las páginas web son múltiples, entre estos tenemos la estructura HTML, las estructuras de hipervínculos, entre otros. (Bharat y Henzinger, 1998; Kleinberg, 1999). Broder(2002), presenta una taxonomía que clasifica las consultas en la web en tres categorías primordiales: informativa, navegacional y transaccional. En la categoría informativa, se le proporciona al usuario información relacionada con un tema en particular, esto a través de las consultas de información. La categoría navegacional, es cuando los usuarios a través de la consulta de navegación acceden a una página o documento en particular. Finalmente, la categoría transaccional aborda los servicios de localización con los que el usuario interactúa.

2.1 Representación de consultas y documentos

Un SRI se encarga de transformar los documentos de su forma original a una representación interna, esto se denomina indexación. La indexación es un proceso, en el cual, se busca representar la información de la forma más exacta posible. Para lograr este objetivo se asigna un conjunto de características de indexación para cada documento. Las características más relevantes de un documento corresponden a una lista de palabras que permiten discriminar entre ellas. Estos son conocidos como términos. De hecho, un documento no sólo está representado por esta lista de términos, sino que también se accede por los términos que perteneces a su lista. Con el fin de obtener un mayor nivel de representación, que permita

recuperar las unidades frasales, o el uso de métodos lingüísticos, semánticos y basados en el conocimiento; Las características complejas deben estar involucradas en el proceso de indexación (Lewis y Jones, 1996).

Existe un proceso llamado normalización, el cual consiste, en proporcionar los términos relevantes antes de la indexación. Por ejemplo, las palabras con altas frecuencias en el documento (Stop-Words) no serán consideradas en la indexación (Rijsbergen, 1979). Las Stop-Words son palabras conocidas como artículos (por ejemplo, el, la, los, las) y preposiciones (por ejemplo, en por). La principal ventaja de este proceso es reducir el volumen del texto hasta un 50%.

Cuando se realiza la comparación de documentos con otros documentos se obtienen términos que son representativos, estos términos se obtienen, haciendo el contraste entre los términos que no son frecuentes con los términos que se hacen presente en todos los documentos. Se utiliza el concepto de ponderación inversa de frecuencia de documentos (IDF) para obtener los términos representativos. Si en un documento existe un término que aparece más frecuentemente, entonces el peso de ese término se incrementará. Al contrario, si un término de un documento aparece con menos frecuencia en otros documentos, entonces su peso disminuirá. Por lo tanto, para una colección de documentos de tamaño *N*, si el término *i* se encuentra en n_i documentos, entonces el peso *idf* de un término está dado por $\log\left(\frac{N}{n_i}\right)$. Una función de ponderación de términos corresponde a la combinación de los pesos *tf* e *idf*, que comúnmente se conoce como peso *tf-idf* (Salton (1971)):

$$W_{ij} = \frac{\log(Freq_{ij}+1)}{\log(length_j)} \log\left(\frac{N}{n_i}\right)$$

W_{ij}: $tf - idf$ *peso del término i en el documento j*

$freq_{ij}$: *frecuencia del término i en el documento j*

$length_j$: *longitud (en términos) del documento j*

N: *número de documentos en la colección*

n_i: *número de documentos a los que se asigna el término i*

Es importante mencionar que se ha omitido el proceso de derivación en los experimentos relacionados. Esto se omite porque los experimentos son construidos en un entorno ideal. Por lo tanto, el proceso de derivación no será necesario. Así mismo, las Stop-Word se descartan porque los términos usados son términos representativos. Resumiendo, puede ser visto como la aplicación general del mismo proceso de detención en todos los experimentos.

Los SRI tienen tres principales categorías de estructuras de datos utilizadas para almacenar términos, índices lexicográficos (índices que se ordenan), estructuras de archivos en clústeres e índices basados en el hashing. Sin embargo, las estructura de datos más utilizada por los motores de búsqueda web (MBW) corresponden al archivo invertido (Ouksel, 2002). Esta estructura corresponde a una lista que contiene los términos representativos de la colección de documentos, por lo que los términos que pertenecen a la consulta coinciden con palabras clave (términos) que están en la lista. Es factible localizar inmediatamente todos los documentos en la colección que contienen esta palabra clave (Rijsbergen, 1979).

2.2 Operaciones de consulta

El objetivo principal de un SRI es proporcionar apoyo al usuario satisfaciendo la necesidad de información de éste, en aquellos documentos que cumplan con esa necesidad. Para esto, el usuario expresa sus requisitos por medio de una "consulta".

En un SRI se puede realizar la formulación de la consulta mediante el uso de operadores booleanos. Una de las principales desventajas de los SRI booleanos es que los resultados son poco intuitivos para usuarios no experimentados, por lo que la formulación de una consulta ad-hoc puede ser ineficaz (Sparck Jones y Willett, 1997a). Como consecuencia, se busca sustituir estos sistemas por otros sistemas que proporcionan formular la consulta por medio de un lenguaje natural sin utilizar operadores específicos. Estos sistemas se basan en la mejor búsqueda de similaridad, calculada para cada consulta y cada documento.

Las consultas se procesan antes de ser indexadas, al igual que los documentos, esto quiere decir que, el procesamiento léxico y la ponderación de términos son ejecutados por el SRI.

2.3 Matching entre consultas y documentos

Un SRI, a través de un modelo booleano, encuentra un subconjunto de documentos del conjunto completo de documentos, este subconjunto contiene documentos que tienen al menos un término de la consulta enviada al SRI, que se presenta al usuario de una manera no clasificada. Por otra parte, resulta factible considerar sistemas que tengan mejores métodos de comparación y entreguen el resultado de acuerdo a la puntuación de relevancia sobre la pertinencia del documento con respecto a la consulta. En consecuencia, un SRI proporciona una lista de los documentos clasificados, ordenados de manera decreciente para el usuario.

En el modelo espacio vectorial (Salton y McGill, 1986), tanto las consultas como los documentos son representados como vectores en un espacio multidimensional. La representación espacial corresponde a los términos indexados de la colección de documentos.

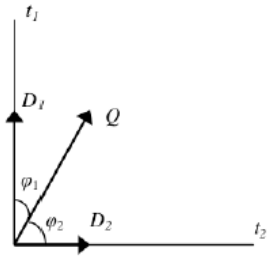

Figura 2.1: Representaciones de documentos y consultas en el modelo vectorial.

En la Figura 2.1 se puede observar un ejemplo del modelo vectorial. En el modelo se representa una consulta y dos documento. En el ejemplo, se puede observar que el espacio vectorial se muestra en dos dimensiones. Por lo tanto, los términos t_1 y t_2 corresponden a la dimensión. D_1 y D_2 son documentos que son representados en el espacio considerando cada uno de los pesos del documento como coordenadas. Se le denomina pesos a la frecuencia que tiene el término en el documento: $D_1=t_1$, t_1 y $D_2=t_2$. Sin embargo, la consulta Q se representa como $Q=t_1, t_1, t_2$. Los ángulos entre los vectores de $D_1 - Q$ y $D_2 - Q$ se presentan en la Figura 2.1

Este espacio es utilizado para definir las medidas que serán útiles para cuantificar la similaridad entre las consultas y los documentos. El mejor match entre una consulta particular y un conjunto de documentos corresponde al documento más cercano con respecto

a la consulta acorde a la medida de similaridad. En la literatura de RI se puede encontrar una amplia gama de fórmulas que son utilizadas para las medidas de distancia. Un detalle de la mayoría de ellos se puede encontrar en libros tales como (Rijsbergen, 1979, Salton y McGill, 1986).

Para realizar la comparación de forma sencilla de un documento con respecto a una consulta, se debe contar el número de términos que existen en común entre ellos. Así, se puede asumir que ambos son representados como vectores de longitud *n*(donde *n* es el número de términos en la colección). Por lo tanto, se tiene la siguiente medida:

$$sim(D, Q) = \sum_{i=1}^{n} D_i Q_i \qquad (2.1)$$

Esta medida se conoce como la función de matching de nivel de coordinación. En contraste, otras medidas de similaridad pueden normalizarse de acuerdo con la longitud del documento y la consulta.

La medida más utilizada por los SRI corresponde a la distancia del coseno. La razón de su popularidad se debe a la interpretación geométrica del modelo vectorial.

$$sim(D, Q) = \frac{\sum_{i=1}^{n} D_i Q_i}{\sqrt{\sum_{i=1}^{n} (D_i)^2 \sum_{i=1}^{n} (Q_i)^2}} \qquad (2.2)$$

La medida del coseno consiste en una función que proporciona los ángulos que se forman entre los vectores de la consulta y el documento (revisar la ecuación 2.2), donde los rangos de valores se encuentran entre 0 y 1, los documentos con las consulta que se encuentran en el valor 0 forman un ángulo de 90°, esto quiere decir que son diferentes, al contrario, los documentos con las consultas que se encuentran en valor 1 forman un ángulo de 0, esto quiere decir que son iguales. En la figura 2.1, los documentos D1 y D2 tienen una similaridad igual a 0, esto, debido a que el ángulo entre los dos vectores es 90°. Esto nos dice que no tienen ningún término en común.

2.4 Evaluación de SRI

En los SRI es común ver una gran cantidad de metodologías que se utilizan en la evaluación de éstos. Realizar una evaluación implica una gran complejidad, puesto que se deben considerar temas de diversas áreas de investigación, tales como estadística, diseño experimental, diseño de sistemas, entre otros. En esta sección se presenta un resumen de las evaluaciones con énfasis en los aspectos principales que constituyen este proyecto.

En un proceso de RI es posible evaluar varios aspectos. Los aspectos a evaluar pueden relacionarse con la velocidad de un SRI, el nivel de interacción del usuario final, las interfaces y el formato de la información presentada a los usuarios. La recuperación y la precisión corresponden a las medidas de efectividad más utilizadas. La recuperación es la proporción de los documentos relevantes que se han recuperado, mientras que la precisión corresponde a la fracción de documentos relevantes que fueron recuperados. La recuperación y la precisión se pueden definir como:

$$Precisión = \frac{número\ de\ documentos\ relevantes\ y\ documentos\ recuperados}{número\ de\ documentos\ recuperados}$$

$$Recuperación = \frac{número\ de\ documentos\ relevantes\ y\ documentos\ recuperados}{número\ de\ documentos\ relevantes}$$

Generalmente, podemos ver representadas estas medidas en un rango de 0 y 1, pero también se pueden expresar en porcentajes como en (Chowdhury, 2010).

Se hace necesario tener conocimiento del número total de documentos relevantes en un conjunto para calcular la recuperación. Sin embargo, esta medida implica una cantidad considerable de esfuerzo y de tiempo, por lo que lo hace una medida difícil de calcular. Además, los conjuntos de documentos que permiten calcular estas medidas pueden ser de libre disponibilidad o comerciales. Mediante el uso de estas colecciones, los investigadores de RI pueden evaluar enfoques con resultados empíricos y posteriormente utilizar los resultados para realizar comparaciones con otros sistemas u otros enfoques.

En la figura 2.2, se puede observar un equilibrio entre el grafico de recuperación-precisión(R-P). Esto quiere decir que la precisión y la recuperación están relacionados de manera inversa. Cuando se reduce la recuperación esto significa que aumenta la precisión y viceversa. Esto se ve reflejado en la mayoría de los textos y manuales de RI.

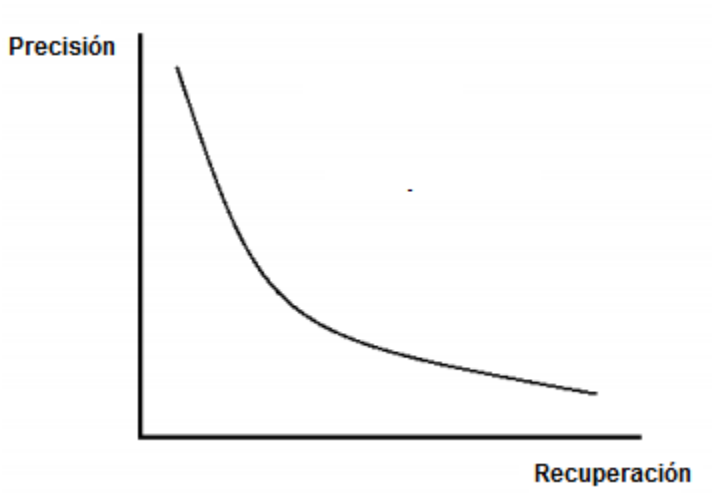

Figura 2.2: Gráfico de recuperación-precisión.

Cuando el conjunto de documentos aumenta constantemente, especialmente con la aparición de las Conferencias de Recuperación de Texto (TREC) (Harman, 1993), las cuales contienen miles de millones de documentos (implica el uso de muchos gigabytes de espacio en disco), se ha vuelto casi imposible proporcionar juicios exhaustivos sobre la pertinencia de los documentos (Tombros et al., 2002).

Para abordar este problema, se utiliza una técnica denominada pooling (Harman, 1993), que es una técnica ampliamente utilizada y muy conocida en estos casos. Esta técnica tiene como base la combinación de los documentos de mayor clasificación de varios SRI, en el cual una consulta se presenta en cada uno de ellos. De este modo, los juicios de los usuarios son un conjunto combinado de los SRI. Se puede deducir que, cuando los documentos relevantes recuperados son representativos de todos los documentos relevantes disponibles en los SRI, entonces esta técnica es efectiva (Harman, 1993).

El tipo de evaluación visto anteriormente se basa en juicios entregados por jueces expertos, basados en la relevancia actual. En (Schamber et al., 1990, Barry, 1994) la relevancia corresponde a un concepto multidimensional, que implica una sola dimensión. En este trabajo, se realizan metodologías de evaluación con respecto a la utilidad proporcionada por los SRI, tomando en cuenta otras dimensiones sobre el concepto de relevancia (Borlund e Ingwersen, 1997; Reid, 2000).

Es importante destacar que en las investigaciones de RI se extraen conclusiones de los resultados empíricos, esto involucra considerar una gran cantidad de factores como la

validación estadística de resultados empíricos, la elección de medidas adecuadas de desempeño, entre otros. Para abordar estos problemas, se ha propuesto una metodología para obtener argumentos científicos a partir de experimentos en RI (Keen, 1992).

2.5 Elección de una medida

Para el clustering de documentos, existe una gran cantidad de medidas disponibles, es por esto que surge la necesidad de elegir la medida más apropiada para el clustering. Van Rijsbergen (1979), advirtió contra el uso de cualquier medida que no esté normalizada por la longitud de los vectores del documento bajo comparación. También, Van Rijsbergen, Sneath y Sokal (1973), realizaron una observación la cual advierte que las diferentes medidas de asociación y distancia son monótonas unas con otras. En conclusión, un método de clustering que depende sólo del ordenamiento de los valores de similaridad daría resultados similares para todas estas medidas.

Willett (1983) verificó la necesidad de normalizar las medidas de similaridad por la longitud de los vectores de documentos. Willet, en este estudio, utiliza tres conjuntos de documentos, cuatro medidas de similaridad (producto interno, coeficiente de Tanimoto[3], coeficiente de coseno y coeficiente de superposición) y cinco esquemas de ponderación de término. Como resultado de los experimentos se pudo observar la poca efectividad de las medidas no normalizadas y también mostraron poca variación en la efectividad de las jerarquías obtenidas con medidas normalizadas. Finalmente se pudo observar que la medida que obtuvo una efectividad de recuperación mejor a las demás, fue la medida del coeficiente de coseno.

Griffiths et al. (1984) presentó algunas pruebas adicionales de la inadecuación de las medidas de similaridad no normalizadas (aunque fue de manera más bien accidental). Griffiths y sus colegas utilizaron la distancia de Hamming y el coeficiente de Dice para medir las relaciones entre documentos. Ambos casos fueron medidos en una serie de experimentos. Los resultados obtenidos con la distancia de Hamming fueron significativamente inferiores a los obtenidos en el coeficiente de Dice para todos los escenarios experimentales. A pesar de la naturaleza de los resultados, los autores no reconocieron significativamente el efecto de normalización.

[3] Es una de las métricas utilizadas para comparar la similaridad y diversidad de conjuntos de pruebas. Utiliza la razón del conjunto interceptante al conjunto de unión como la medida de similaridad. Es decir, es igual a cero si no hay elementos que intercepten e igual a uno si todos los elementos interceptan.

Kirriemuir y Willett (1995), aplicaron cuatro métodos de clustering y cinco medidas de similaridad y distancia, entre estos métodos se encontraban los coeficientes coseno y Jaccard y la distancia euclidiana normalizada. Los resultados demostraron que los coeficientes de coseno y Jaccard dieron clústeres más efectivos.

Rorvig (1999) hizo investigaciones las cuales tenían como objetivo comparar diferentes medidas de similaridad entre documentos. Rorvig tenía principalmente interés en investigar cómo un conjunto de medidas de similaridad actuaría como parte de una interfaz de recuperación de información visual. Se seleccionaron cinco temas del TREC y los conjuntos de documentos utilizados para cada tema comprendían entre 421 y 586 documentos TREC.

Para medir la calidad de las medidas de similaridad se definió separar visualmente los documentos relevantes y no relevantes para cada tema. En el estudio se incluyeron cinco medidas de similaridad (Dice, Jaccard, coseno, superposición y asimetría), lo que tuvo como resultado dos medidas con un mayor éxito y estas son la medida del coeficiente de coseno y superposición en la estructura de la recuperación. Las medidas que más se utilizan en la medición de proximidad entre documentos en un espacio vectorial de documentos, suele ser el coeficiente de coseno y la distancia euclidiana. Jones y Furnas (1987) realizaron un análisis a una gran cantidad de medidas de similaridad, entre estas, el coeficiente de coseno. Con el análisis realizado observaron que la comparación de documentos basada en su ángulo en un espacio vectorial se aproxima a una comparación que se basa en su contenido tópico, debido a que esto se expresa a través de relaciones de términos dentro del documento.

Dubin (1996) hizo una observación en donde las medidas angulares eran más sensibles a los pesos relativos de los atributos, en comparación con las medidas de distancia que son más sensibles a los pesos absolutos. Por otra parte, la distancia euclidiana para el clustering ha sido criticado por Willett (1988), señalando que con esta medida dos documentos pueden considerarse muy similares, incluso si no comparten términos en común.

Después de observar las medidas propuestas por distintos autores, se determinó que para este proyecto se utilizará la medida de distancia del coseno, ya que varios autores la consideran como una de las mejores medidas de similaridad de documentos. A continuación se dará a conocer más detalladamente esta medida.

2.5.1 Medida de distancia del coseno

La medida de distancia del coseno, es la medida más utilizada en la actualidad en los SRI. Para esta medida se realiza una matriz la cual está compuesta por todos los términos que se encuentran en todos los documentos sin repetirse. Para comprender de una mejor forma esta medida, se utilizará el ejemplo propuesto por Martínez(2004).

Si el SRI contiene los siguientes cuatro documentos:

D1: El río Danubio pasa por Viena, su color es azul.

D2: El caudal de un río asciende en invierno.

D3: El río Rhin y el río Danubio tienen mucho caudal.

D4: Si un río es navegable, es porque tiene mucho caudal

La matriz de términos y documentos dentro del modelo del Espacio Vectorial se puede observar a continuación en la tabla 2.1:

	Río	Danubio	Viena	Color	Azul	Caudal	Invierno	Rhin	Navegable
D1	1	1	1	1	1	0	0	0	0
D2	1	0	0	0	0	1	1	0	0
D3	2	1	0	0	0	1	0	1	0
D4	1	0	0	0	0	1	0	0	1

Tabla 2.1. Ejemplo matriz de términos y documentos en el espacio vectorial.

Como supuesto el SRI ha eliminado las determinantes, preposiciones y verbos (“el”, “pasa”, “por”, etc), que presentan los documentos.

Sparck-Jones, apreció la capacidad de discriminación de un término frente a otro. Esta generalidad de un término dentro del conjunto de documentos y no en un único documento, y se pensó en incentivar la presencia de aquellos términos que aparecen en menos documentos frente a los que aparecen en todos o casi todos, ya que realmente los términos muy frecuentes discriminan poco o nada a la hora de la representación del contenido de un documento. Para medir este valor de discriminación se utiliza la medida *idf* (frecuencia inversa de documento).

Así, para la construcción de la matriz de términos y documentos, se consideran las siguientes definiciones:

- $\boldsymbol{n}$ **=** Número de términos distintos en la colección de documentos.
- $\boldsymbol{tf_{ij}}$ **=** Número de ocurrencias de término *tj* en el documento D_i (Frecuencia del término o *tf*).
- $\boldsymbol{df_j}$ = Número de documentos que contienen el término *tj.*
- $\boldsymbol{idf_j}$ = el $log\left(\frac{d}{df_j}\right)$, donde *d* es el número total de documentos (frecuencia inversa del documento).

A los términos en cada documento de la matriz se le asignan pesos, los cuales se basan en la frecuencia con que ocurren en el conjunto de documentos y en la aparición de un término en un documento particular. Cuando un término aparece más a menudo en un documento, entonces, su peso aumenta, por otro lado si el término aparece más a menudo en todos los demás documentos, entonces su peso disminuye. Sólo si el término aparece en el documento, el peso es distinto de cero. Para un conjunto de documentos grande que contiene numerosos documentos pequeños, es probable que los vectores de los documentos contengan ceros principalmente.

Para calcular el factor de peso (d) de un término en un documento, se describe como la combinación de la frecuencia de término (*tf*), y la frecuencia inversa del documento (i*df*). Para calcular el valor de la *j*-esima entrada del vector que corresponde al documento *i*, se emplea la siguiente ecuación: $d_{ij} = tf_{ij}\ x\ idf_j$. El cálculo de las frecuencias inversas de los términos en los documentos y la posterior aplicación de esta fórmula sobre la matriz del ejemplo, proporcionaría la siguiente matriz de pesos.

<u>Calculo de frecuencias inversas:</u>

idf(Río) = Log (4/4) = log (1) = 0

idf(Danubio) = Log (4/2) = log (2) = 0,301

idf (Viena) = Log (4/1) = log 4 = 0,602

idf (color) = Log (4/1) = log 4 = 0,602

idf (azul) = Log (4/1) = log 4 = 0,602

idf (caudal) = Log (4/3) = log 1,33 = 0,124

idf (invierno) = Log (4/1) = log 4 = 0,602

idf (Rhin) = Log (4/1) = log 4 = 0,602

idf (navegable) = Log (4/1) = log 4 = 0,602

A continuación en la tabla 2.2 podemos observar un ejemplo de la matriz de peso de documentos con términos:

	Río	Danubio	Viena	Color	Azul	Caudal	Invierno	Rhin	Navegable
D1	0	0,301	0,602	0,602	0,602	0	0	0	0
D2	0	0	0	0	0	0,124	0,602	0	0
D3	0	0,301	0	0	0	0,124	0	0,602	0
D4	0	0	0	0	0	0,124	0	0	0,602

Tabla 2.2. Ejemplo matriz de peso documentos con términos.

Luego de calcular los pesos de cada término, se da paso a calcular la similaridad que existe entre cada uno de los documentos (D1, D2, D3, D4). Para calcular la similaridad entre los documentos se utiliza la medida de distancia del coseno. El modo más sencillo de obtener la similaridad es por medio del producto escalar de los vectores (es decir, multiplicando los componentes de cada vector y sumando los resultados). Continuando con el ejemplo anterior a continuación se realiza el cálculo de las similaridades:

Cálculo de similaridades:

Sim (D1,D2) = 0*0 + 0,301*0 + 0,602*0 + 0,602*0 + 0,602*0 + 0*0,124+0*0,602+ 0*0 + 0*0 = 0

Sim (D1,D3) = 0*0 + 0,301*0,301 + 0*0,602 + 0*0,602 + 0*0,602 +0*0,124 + 0*0 + 0*0,602 + 0*0 = 0,09

Sim (D1,D4) = 0*0 +0*0,301 + 0*0,602 + 0*0,602 + 0*0,602 + 0*0,124 +0*0+ 0*0 + 0*0,602 = 0

Sim (D2,D3) = 0*0 + 0*0,301 + 0*0 + 0*0 + 0*0 + 0,124*0,124 + 0*0,602 + 0*0,602 + 0*0 = 0,15376

Sim (D2,D4) = 0*0 + 0*0 + 0*0 + 0*0 + 0*0 + 0,124 * 0,124 + 0*0,602 + 0*0 + 0*0,602 = 0,15376

Sim (D3,D4) = 0*0 + 0*0,301 + 0*0 + 0*0 + 0*0 +0,124*0,124 + 0*0 +0*0,602 + 0*0,602 = 0,15376

En los cálculos de la similaridad entre documentos, se considera que la similaridad entre los mismos documentos es 1, por ejemplo entre D1 y D1, además la similaridad entre D1 y D2 es la misma que D2 y D1 por lo que no es necesario volver a calcular el resultado.

Con los valores de similaridad obtenidos entre todos los documentos, se realiza la construcción de la matriz de similaridad, la cual se muestra a continuación en la tabla 2.3:

	D1	D2	D3	D4
D1	1	0	0,09	0
D2	0	1	0,15376	0,15376
D3	0,09	0,15376	1	0,15376
D4	0	0,15376	0,15376	1

Tabla 2.3. Ejemplo matriz de similaridad entre documentos.

CAPÍTULO 3

Clustering de documentos para la RI

El análisis de clúster es una técnica estadística multivariable que permite identificar los grupos o clústeres que sean similares entre sí, ésto en un espacio que es típicamente multidimensional. Los objetos son agrupados de tal manera que los objetos del mismo grupo son similares entre sí y disímiles a los objetos de otros grupos (Gordon, 1987). Agrupar objetos es una tarea realizada por los seres humanos durante miles de años (Willett, 1988; Kural, 1999), y en las últimas décadas se ha ido automatizando debido a los avances de la tecnología (Willett, 1988).

Es común que exista una confusión con respecto al uso de los términos en análisis de clúster y clasificación. Watanabe (1969) y Willett (1988), entre otros autores, han hecho distinción entre estos dos conceptos. Para la tarea de clustering es necesario agrupar los objetos, definiendo un conjunto de propiedades en clases de acuerdo con la intensidad de relaciones que existe entre los objetos (esto quiere decir, que las clases tienen que ser descubiertas). Por otro lado, la tarea de clasificación, primero ubica un conjunto de objetos de muestra en algunas clases (generalmente de forma manual), luego se debe esperar que las nuevas muestras se ubiquen en las clases existentes imitando la clasificación demostrada en la etapa de entrenamiento. Sin embargo, debe tenerse en cuenta que en la literatura inicial, cuando se utilizaba la tarea de clasificación se debía asignar objetos a los clústeres, y el diagnóstico implicaba la tarea de asignar un nuevo objeto entrante a uno de los clústeres existentes (Jardine y Sibson, 1971). Durante los años la terminología ha ido cambiando, en especial en el área de RI. Es por esto que en este proyecto no se considera el término de clasificación como una alternativa del clustering.

Las técnicas de análisis de clustering han sido aplicadas durante mucho tiempo a campos científicos como la biología, ciencias médicas (psiquiatría, patología), ciencias sociales (arqueología, sociología, criminología), ciencias de la tierra (geografía, geología) y ciencias de la ingeniería (reconocimiento de patrones, cibernética) (Anderberg, M.R., 1973).

Las aplicaciones específicas de análisis de clustering cuentan desde clustering de estructuras de ADN para el análisis de expresión genética (Hartuv et al., 1999), hasta el clustering de un

simple whisky de malta basado en características como fermentación, dulzura, entre otros (Wishart, D., 1998). En RI, se ha utilizado el análisis de clúster en el clustering de documentos y en clustering de términos. El clustering de términos, se basa en la idea de realizar la agrupación con los documentos en que los términos se repiten, y permite que cada término en un documento o consulta sea reemplazado por la representación (es decir, colección de términos de indexación) que determina a que grupo pertenece el término. El clustering de términos se aplica en áreas que incluyen la expansión de la consulta (Sparck Jones, 1971, Minker et al., 1972, Van Rijsbergen et al., 1981), la construcción automática del tesauro[4] (Crouch & Yang, 1992) y la unión del tesauro (Amba et al. 1996). Peat y Willett (1991) plantea preguntas sobre el uso de datos efectivo de la co-ocurrencia de palabras claves.

El clustering de documentos opera principalmente sobre la base de la similaridad entre los documentos. La cantidad de términos que existen en común entre un par de documentos es lo que indica típicamente la similaridad entre este par de documentos. Según Van Rijsbergen (1979), uno de los investigadores que por primera vez sugirieron el uso del clustering automático para RI fue Good (1958). La forma tradicional de aplicar el clúster de documentos es en forma estática, con una colección completa de documentos, antes de consultar (clustering estático). Una alternativa para agrupar documentos, es responder a una consulta, agrupando sólo los documentos que han sido recuperados por el SRI respondiendo a esta consulta (clustering dinámico) (Preece, 1973). En clustering dinámico, puede ocurrir que los grupos de documentos resultantes sean diferentes para diferentes consultas. En el clustering de documentos existen dos grandes tipos de clustering que se utilizan principalmente en RI, estos son aglomerativos y jerárquicos.

En el método aglomerativo se organizan K clústeres los cuales contienen un conjunto de N documentos en total, donde K se especifica a priori o se determina como parte del método de clustering. Los métodos aglomerativos cuentan con requisitos computacionales bajos, típicamente en el orden de $O(N)$ a $O(NlogN)$(en tiempo) para el clustering de N documentos (Willett, 1988). Esto tuvo como consecuencia que en el primer período de investigación en el clustering de documentos, los métodos aglomerativos fueron favorecidos, ya que ofrecían el potencial de aumentar la eficiencia de un SRI (Rocchio, 1966; Salton, 1971).

[4] El tesauro es una clase de diccionario que contiene relaciones entre una gran cantidad de palabras que se vinculan de manera directa con un término que se consulta.

En el método aglomerativo la idea principal es elegir una partición inicial de los documentos, para posteriormente, modificar las pertenencias de los clústeres y así obtener una mejor partición[5] (Anderberg, 1973). El número de particiones posibles de N documentos en K clústeres (especialmente para grandes valores de N) hace que una enumeración completa sea difícil (Willett, 1988), por lo tanto una solución óptima es imposible. Es por esto que para encontrar una solución aproximada se utilizan métodos heurísticos. En resumen, los métodos aglomerativos sufren en una base teórica, ésto debido a que en la mayoría de los casos requieren para su experimentación un gran número de parámetros determinados arbitrariamente, además éstos pueden depender del orden en que se procesan los documentos (Salton & Amp; Wong, 1978, Willett, 1988).

En este proyecto se utilizará el método de clustering jerárquico, uno de los métodos más utilizado en la agrupación en RI (Willet, 1988). Jardine y Sibson (1971), Salton y Wong (1978) y Van Rijsbergen (1979) identificaron tres principales fortalezas de los métodos jerárquicos. En principio, teóricamente estos métodos son más atractivos, porque estos no dependen del orden en que son procesados los documentos. En segundo lugar, del conjunto de documentos se derivará una sola clasificación. Y por último, estos métodos pueden presentar pequeños cambios en los vectores originales de los documentos y esto tendrá como resultado pequeños cambios en las jerarquías resultantes, lo que lo hace un método estable.

3.1 Representación de documentos

Para comenzar con el proceso de clustering, es necesario determinar el número y el tipo de variables que describen los documentos. En un espacio N-dimensional se puede ver representados los documentos por un vector, donde el número de términos corresponden a N, estos términos forman el vocabulario de indexación de la base de datos. En la literatura es común ver que los términos se han utilizado como características de indexación para clutering. Recientemente, Hatzivassiloglou et al. (2000) y Maarek et al. (2000), han aumentado las representaciones de documentos con unidades frasales empleando diferentes niveles de análisis lingüístico. Tomando en cuenta lo anterior, surgen dos cosas importantes, que términos se eligen para representar un documento, y cómo se evalúa la importancia relativa de estos términos para propósitos de clustering.

[5] Una forma típica de decidir la pertenencia a un clúster es minimizando una función de costo. Por ejemplo, el método popular K-means (Kaufman & Rousseeuw, 1990) se basa en minimizar la función de suma de cuadrados.

3.2 Medición de la relación entre documentos

Cuando se obtiene un conjunto de documentos, es necesario establecer una representación adecuada para éstos, para posteriormente medir la similaridad existente entre todos los pares de documentos pertenecientes a este conjunto. Para lograr este objetivo, existen una gran cantidad de medidas que calculan la similaridad entre los objetos. Sneath y Sokal (1973) clasifican dichas medidas en cuatro clases principales: asociación, disimilaridad, probabilística y coeficientes de correlación. Debido a que las dos últimas clases de medida en clustering han sido limitadas, la mayoría de la literatura se refiere a la asociación (o similaridad) y disimilaridad.

En el análisis de clustering se utiliza una gran cantidad de medidas de similaridad. Cormack (1971), Anderberg (1973), Sneath y Sokal (1973), Hubálek (1982), Kaufman y Rousseeuw (1990), entre otros proporcionan discusiones detalladas sobre el uso de tales medidas en el análisis de clústeres. En particular, para el contexto de clustering de documentos, los coeficientes más utilizados se pueden encontrar en (Van Rijsbergen, 1979, Salton y McGill, 1983, Willett, 1988, Ellis et al., 1993). Para poder discutir respecto al efecto que puede tener alguna de estas medidas en particular el clustering de documentos, primero será necesaria una definición formal de estas medidas.

3.2.1 Definición Formal

Para la siguiente definición se asumirá que los documentos corresponderán a vectores en un espacio *N*-dimensional, *N* corresponde a la cantidad de términos de indexación. Suponemos que un documento x_i contiene *N* términos de indexación que se representan por binarios o por pesos reales: $x_i = (x_1, x_2, \dots, x_n)$.

Si la cantidad de documentos para agrupar es *X*, entonces un coeficiente de distancia es una función $d: X\ x\ X \rightarrow \mathbb{R}$, donde el conjunto de números reales no negativos corresponde a $\mathbb{R}$. Tal función *d,* en general, satisface los siguientes axiomas:

Reflexividad: d (x, x) = 0

Simetría: d (x, y) = d (y, x)

Desigualdad triangular: d (x, y) ≤ d (x, z) + d (z, y)

Donde los documentos *x, y, z* pertenecen al conjunto *X*. Una función de distancia que satisface estos tres axiomas es una función métrica. Una función ultramétrica δ es aquella que satisface los dos primeros axiomas y: Δ (x, y) ≤ max [d (x, z), d (z, y)] (Diday & Simon, 1976).

Una medida de similaridad *s* también puede definirse como una función s: X x X → ℝ. Una función de similaridad *s* es métrica si satisface los axiomas de reflexividad y simetría, y además:

s(x, y)s(y, z) ≤ [s(x, y) + s(y, z)]s(x, z) (Theodoridis & Koutroumbas, 1999)

Generalmente, los valores de similaridad y distancia caen en el rango entre 0 y 1. Cuando el valor sea cercano a 1 entonces la similaridad entre los documentos será más alta. Para la función de distancia, mientras más pequeño sea el valor de la función, entonces los documentos son menos similares entre sí. En esta sección consideraremos la medida de similaridad.

	x_1	x_2	x_3	x_4	x_5
x_1	1				
x_2	0.6	1			
x_3	0.4	0.8	1		
x_4	0.1	0.5	0.7	1	
x_5	0.1	0.2	0.2	0.3	1

Figura 3.1. Una matriz de similaridad.

Si se tiene *N* documentos, entonces la matriz será de *NxN* para almacenar los valores de similaridad entre los documentos, esta matriz corresponde a *S(X).* La matriz resulta ser una matriz triangular, donde S_{ij} es la medida de similaridad que existe entre el documento x_i y el documento x_j. En la diagonal de la matriz se pueden observar los valores máximos de similaridad que existen entre los documentos, esto debido a que la intersección corresponde a los mismos documentos. También, es importante notar que las funciones de similaridad son

simétricas ($S_{ij} = S_{ji}$), para calcular todos los pares de asociaciones de documentos se requiere de $\frac{Nx(N-1)}{2}$ operaciones, como consecuencia el cálculo de similaridad es independiente y computacionalmente caro para un gran conjunto de datos $O(N^2)$ (en tiempo). En la figura 3.1 se puede observar una matriz de 5x5. Debido a que la matriz es simétrica sólo se consideran los valores que se encuentran por debajo de la diagonal.

Con el fin de hacer cálculos más eficaces de similaridad para una gran cantidad de documentos Croft (1977), propuso un método para el cálculo de coeficientes basado en una estructura de indices invertidos. Este tipo de estructura sirve para determinar qué documentos no tienen términos en común con otro documento, y así, evitar los coeficientes entre ese par de documentos, ya que, su similaridad sería igual a cero.

3.3 Método de clustering jerárquico

Un método de clustering jerárquico es una descomposición jerárquica de una colección de datos, lo cual forma un dendograma o árbol, que divide el conjunto de datos en conjuntos cada vez más pequeños.

Si se tiene un conjunto de documentos el cual está representado por *X*, que se agrupan $X = \{x_1, x_2, \dots, x_n\}$. Los documentos son representados por vectores *n*-dimensional, donde cada dimensión es un término de indexación.

Un clustering de *X* en *m* conjuntos de documentos se puede definir como $R = \{C_1, C_2, \dots, C_m\}$, de tal forma que se cumplan las siguientes condiciones:

- Cada cluster C_i debe contener al menos un documento: $C_i \neq \emptyset$, i = 1,…,m
- Todos los clústeres unidos corresponden al conjunto $X = \cup_{i=1}^{m} C_i = X$
- Dos clústeres no tienen documentos en común: $C_i \cap C_j = \emptyset, i \neq j,\ i,j = 1, \dots, m$

Se dice que un agrupamiento R_1 que contiene *k* clústeres está anidado en el agrupamiento R_2, que contiene clústeres *r<k*, si cada clúster en R_1 es un subconjunto de un clúster en R_2, y al menos un clúster de R_1 es un subconjunto propio de R_2 (Theodoridis y Koutroumbas, 1999). Por ejemplo, el clúster $R_1 = \{\{x_1, x_3\}, \{x_4\}, \{x_2, x_5\}\}$ está anidado en $R_2 = \{\{x_1, x_3, x_4\}, \{x_2, x_5\}\}$. Por

otro lado, R_1 no está anidado dentro de $R_3 = \{\{x_1, x_4\}, \{x_3\}, \{x_2, x_5\}\}$ (ejemplos tomados de Theodoridis y Koutroumbas (1999)).

Los métodos jerárquicos pueden distinguirse si son basados en conceptos de teoría de matrices o en conceptos de teoría de grafos (Anderberg, 1973; Theodoridis & Koutroumbas, 1999). Esta sección trata de métodos que son basados en la teoría de matriz, esto debido a que son los métodos más utilizados en la RI (Willet, 1988).

Generalmente el procedimiento que siguen los métodos jerárquicos es el siguiente:

1. Determinar las similaridades entre los documentos.
2. Formar un clúster de los dos objetos más cercanos.
3. Redefinir la similaridad entre el clúster nuevo con los demás objetos, sin alterar las demás similaridades.
4. Repetir los pasos 2 y 3 hasta que todos los objetos se encuentren en un clúster.

En general los diferentes métodos aglomerativos que existen, se diferencian en la forma en que se implementa el proceso número 3. En cada paso *t* del proceso de clustering, la matriz de similaridad que inicialmente es de tamaño *NxN*, se convierte en (*N*-*t*) x (*N*-*t*). La matriz $S_t(X)$ de la etapa *t* del proceso se deriva de la matriz $S_{t-1}(X)$, anulando las dos filas y columnas que corresponden a los nuevos documentos fusionados, y se añade una nueva fila y columna que tiene las nuevas similaridades que se formaron a partir de los documentos fusionados y con todos los documentos que no fueron afectados.

Al aplicar un método de clustering jerárquico, se pueden representar sus resultados en forma de un dendograma (Jardine & Sibson, 1971) (Figura 3.2). Un dendograma corresponde a un árbol con niveles numéricos que están asociados a sus ramas. Los valores numéricos que se observan en la figura 3.2 corresponden al nivel de similaridad en los que se forman los clústeres. En cualquier nivel de similaridad, se puede trazar una línea perpendicular al eje de similaridad. De esta manera, cada rama del árbol que se corta por la línea, representa un clúster que consiste en elementos en el subárbol con raíz en esa rama. En el nivel más bajo de similaridad, todos los documentos están en un solo clúster.

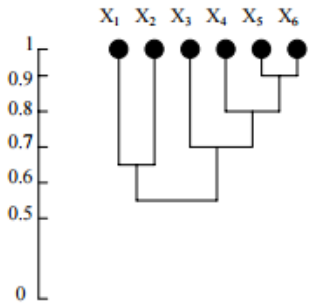

Figura 3.2. Dendograma de similaridad.

Existe una gran cantidad de lecturas, las cuales ofrecen una visión detallista sobre los aspectos de eficiencia de los diversos métodos de clustering. Es importante señalar que la eficiencia realmente es una propiedad del algoritmo que implementa el método de clustering (Jardine & Sibson, 1971). Van Rijsbergen (1979) señaló que en ocasiones es útil distinguir el método clúster de su algoritmo, pero también señaló que en el contexto de RI esta distinción es menos importante.

A continuación en los siguientes párrafos se presentarán cuatro algoritmos de clustering jerárquicos los cuales han sido investigados más ampliamente en investigaciones de RI. Estos cuatro algoritmos corresponden a Single link, Complete link, Average link y Ward's.

3.3.1 Single Link

En el algoritmo Single Link, dos clústeres son similares si la similaridad entre todos los pares de documentos es alta, de tal manera que un documento se encuentra en un clúster y el otro documento en otro clúster (Voorhees, 1985a). Como ejemplo, si se tiene un clúster *i* y un clúster *j* que se han unido, entonces la similaridad entre un nuevo clúster (denotado por *p*) y otro clúster (denotado por *r)* se determina de la siguiente forma: $S_{pr} = \max(S_{ir}S_{jr})$.

Single link es denominado de esta manera, puesto que los clústeres son unidos en cada etapa por el par de objetos más cercanos entre ellos. Para cualquier clúster formado por este algoritmo, cada objeto del clúster es más similar a otro objeto del mismo clúster. Así mismo, cualquier objeto que no se encuentra en el mismo clúster, es menos similar.

Consecuentemente, cada documento debe estar en el mismo clúster con el documento más cercano (más similar).

El algoritmo Single Link posee una característica que tiende a formar clústeres alargados con poca cohesión interna, un efecto denominado encadenamiento (Jardine & Sibson, 1968). Jardin y Sibson ven el efecto de encadenamiento como una descripción de lo que hace el método en términos de teoría de grafos. Hartigan (1975) describe que los clústeres producidos por el algoritmo Single Link "son famosamente encadenados en formas alargadas, donde objetos muy separados están unidos entre sí por una cadena de objetos cercanos". Además, si los clústeres son largos con altas densidades de objetos dentro de cada grupo, entonces el algoritmo Single Link será mejor que el resto de los algoritmos jerárquicos.

Van Rijsbergen (1971) propuso una implementación del algoritmo de Single Link que tiene O(N2) (en tiempo) de requisitos de almacenamiento y espacio.

3.3.1.1 Ejemplo algoritmo Single Link

Para comprender de mejor manera el método Single link se mostrará a través de un ejemplo. En la figura 3.1 de este capítulo se observa una matriz de similaridad, la cual es utilizada como base para el siguiente ejemplo.

En la figura 3.3a se observa que los documentos que poseen una mayor similaridad son $\{x_3, x_2\}$ con un puntuación de 0.8. Por lo tanto, se convierte en el primer par de documentos en juntarse formando el clúster $x_{2,3}$. Luego, en la figura 3.3b se observa una matriz la cual es el resultado de la matriz 3.3a, donde se unieron las filas y columnas de los documentos x_2 y x_3, y se insertó un fila y columna del clúster formado recientemente $x_{2,3}$. Debido a la unión de los documentos más similares, los valores de similaridad en la matriz 3.3b fueron actualizados, de tal forma que la similaridad entre el nuevo clúster $x_{2,3}$ y cada uno de los documentos x_1, x_4, x_5, es el máximo de las similaridades.

En la etapa que sigue la similaridad más alta que se observa es entre el documento x_4 y el clúster $x_{2,3}$ con una puntuación de 0.7 en la matriz 3.3b. Al igual que en la iteración anterior, la matriz 3.3c deriva de la matriz 3.3b, y el algoritmo continúa uniendo el par de documentos,

o clústeres, que contenga la similaridad más alta hasta que finalmente queda un solo clúster. Finalmente la matriz de similaridad queda con solo dos objetos (x_1 y $x_{2,3,4,5}$).

	x_1	x_2	x_3	x_4	x_5
x_1	1				
x_2	0.6	1			
x_3	0.4	0.8	1		
x_4	0.1	0.5	0.7	1	
x_5	0.1	0.2	0.2	0.3	1

(3.3a)

	x_1	$x_{2,3}$	x_4	x_5
x_1	1			
$x_{2,3}$	0.6	1		
x_4	0.1	0.7	1	
x_5	0.1	0.2	0.3	1

(3.3b)

	x_1	$x_{2,3,4}$	x_5
x_1	1		
$x_{2,3,4}$	0.6	1	
x_5	0.1	0.3	1

(3.3c)

Figura 3.3. Transformación de la matriz de similaridad mediante la aplicación del algoritmo Single Link.

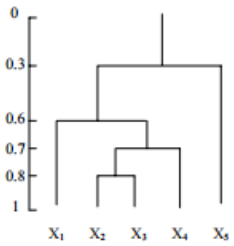

Figura 3.4. Dendrograma de similaridad para el ejemplo.

Se puede observar en la figura 3.4 el dendograma de similaridad que resulta luego de aplicar el algoritmo de Single Link a la matriz de similaridad de entrada. Sólo los niveles de similaridad en los que se forman los grupos en el ejemplo se muestran en la Figura 3.4. En el nivel de similaridad 0 sólo hay un clúster presente, todo el conjunto de documentos. Los otros tres algoritmos de clustering se pueden aplicar a través de una estrategia de actualización diferente de la matriz de similaridad.

3.3.2 Complete Link

El algoritmo Complete Link es una definición completamente opuesta al algoritmo de Single Link, puesto que la similaridad entre dos clústeres es el mínimo de las similaridades entre todos los pares de documentos, de tal forma que un documento del par está en un clúster y el

otro documento en otro clúster. Como ejemplo, si se tiene agrupado el clúster *i* y el clúster *j,* entonces la similaridad entre el nuevo clúster p y otro clúster *r* se determina de la siguiente forma: $S_{pr} = \min(S_{ir}, S_{jr})$.

En el algoritmo Complete Link, los clústeres que se forman tienden a ser más pequeños y estrechamente unidos, ésto se debe a la forma en que son unidos, todo lo contrario al algoritmo Single Link. Este algoritmo se diferencia del Single Link, debido que la similaridad mínima que existe entre los documentos que pertenecen a un mismo clúster nunca puede llegar a ser cero. En el algoritmo complete link puede ocurrir que el vecino más cercano de un documento se encuentre en un clúster diferente, sin embargo, los vecinos más cercanos mutuos estarán siempre en el mismo clúster (Voorhees, 1985).

3.3.3. Average Link

En el algoritmo Average Link la similaridad entre dos clústeres es la media de las similaridades entre todos los pares de documentos, de esta forma un documento del par pertenece a un clúster y el otro documento a otro clúster.

Los clústeres formados por el algoritmo Average Link no son tan alargados como los del algoritmo Single Link, ni tan ajustados como en el algoritmo de Complete Link. Las similaridades en el algoritmo Average link se basan en un promedio. Es debido a esto que no se pueden deducir las similaridades mínimas o máximas entre los documentos de un clúster (Voorhees, 1985).

Sneath y Sokal (1973), afirmaron que el algoritmo Average Link es el mejor de los métodos jerárquicos.

3.3.4. Ward's

De acuerdo al algoritmo propuesto por Ward's (1963), la unión entre clústeres se elige para minimizar una función objetivo la cual demuestra el interés que tiene el investigador en un problema en particular.

Este algoritmo fue ilustrado como una función objetivo de suma de errores de cuadrados. Por otro lado Wishart (1969) demostró que el algoritmo de Ward puede implementarse por medio de la actualización de una matriz de distancias euclidianas cuadradas entre los centroides de los grupos.

En el algoritmo Ward se define un clúster como un grupo de documentos de tal forma que la suma de errores de cuadrados de distancias euclidianas entre documentos de cada clúster es mínima. Los clústeres producidos por este algoritmo tienden a ser aproximadamente del mismo tamaño (Milligan et al., 1983).

CAPÍTULO 4

4.1 Algoritmo en línea

Para comprender de una mejor manera los experimentos que se realizarán en la primera parte de los escenarios experimentales con respecto a la precisión, es necesario comprender el funcionamiento de un algoritmo en línea.

Un algoritmo en línea es capaz de ejecutarse sin necesidad de disponer de todos los datos de entrada al inicio de su ejecución, esto quiere decir que puede trabajar a medida que va recibiendo los datos de entrada. Sin embargo, como un algoritmo en línea no conoce toda la información al inicio de su ejecución, encontrar un óptimo resulta imposible. Es por esto, que para saber si un algoritmo en línea es competitivo se utiliza una constante *c*, donde la *competitividad = c * Óptimo*. Un ejemplo conocido de un algoritmo en línea es el ejemplo de aprender a esquiar.

Supongamos que aprender/arrendar el equipamiento cuesta *$x* al día y comprarlo cuesta *$y*. Además, $y = cx$ para un entero $c \geq 1$ (para simplificar). La persona no sabe si le gustará el nuevo deporte y al final de cada día decidirá si arrienda o compra.

El óptimo es cuando se sabe cuántos días se usará el equipamiento. Supongamos que son t días

Si $tx < y$, conviene arrendar

Si no comprar ($tx \geq y$)

(Si $tx = y => t = c$, cuesta lo mismo arrendar o comprar)

A continuación se puede observar por medio de la figura 4.1 un gráfico del ejemplo del óptimo de aprender a esquiar:

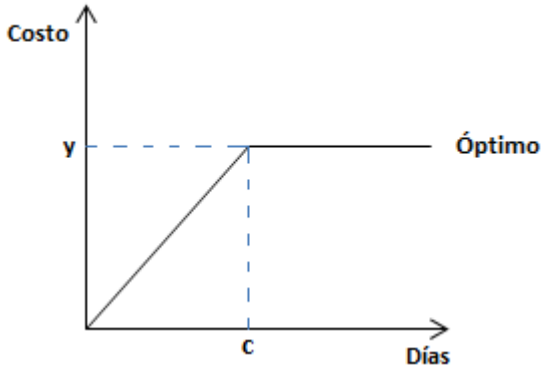

Figura 4.1. Gráfico ejemplo aprender a esquiar.

Supongamos que arrendamos hasta que $c = \frac{y}{x}$ arriendos, y luego compramos si decidimos seguir aprendiendo a esquiar (día c+1).

A continuación se puede observar por medio de la figura 4.2 un gráfico del ejemplo del óptimo v/s en línea de aprender a esquiar:

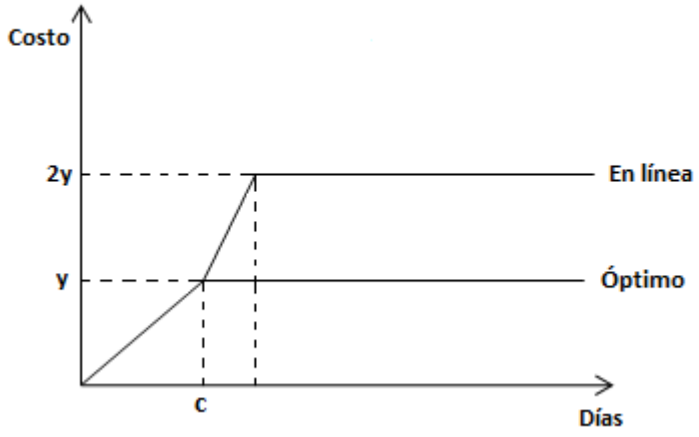

Figura 4.2. Gráfico ejemplo óptimo v/s en línea.

$Competitividad = \begin{cases} 1, & t \leq c \\ 2, & t > c \end{cases}$, con t = Días.

Supongamos que ahora arrendamos hasta *K* días y luego compramos (antes teníamos

$K = c$).

Luego en la figura 4.3 se puede observar el gráfico del peor caso del ejemplo aprender a esquiar:

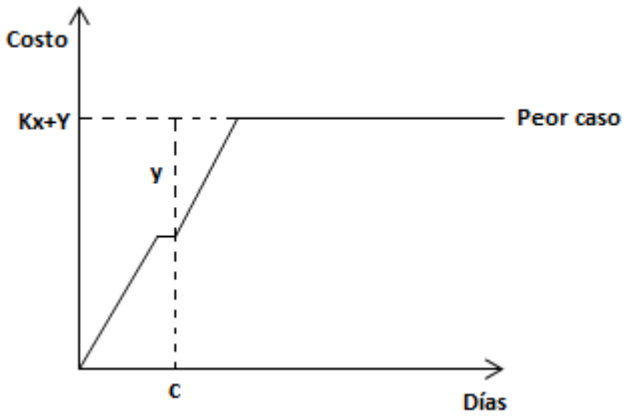

Figura 4.3. Gráfico ejemplo peor caso aprender a esquiar.

Ningún valor de *K* puede tener competitividad 1. (Basta con escoger $t = k+1$)

$$Competitividad = Max\left(\frac{kx+y}{tx}, \frac{kx+y}{y}\right)$$

$$k = 0, t = 1 => t = c \geq 2$$

$$sino\ si\ kx \leq y\ tenemos\ que$$

$$\frac{kx+y}{kx} \geq 2 \quad (k = t)$$

$$si\ kx\ > y\ entonces\ \frac{kx+y}{y} \geq 2$$

Entonces ningún algoritmo determinístico tiene $c < 2$ => usar kc es óptimo. Pero un deportista "extremo" no es tan conservador, así que decide usar un algoritmo aleatorio y en ese caso la competitividad estará basada en el promedio sobre los algoritmos y no el peor caso.

Sea A_i el algoritmo que arrienda *i-1* días y compra e *i*-ésimo día, sea π_i la probabilidad de usar A_i $(i \geq 1)$.

Supongamos que la competitividad promedio de todas las A_i sea a lo más α. Entonces, tendremos que escoger la distribución π tal que:

$$Costo\ (en\ línea) \leq\ \alpha t x \ldots t \leq c$$

$$Costo\ (en\ línea) \leq \alpha y \ldots t > c$$

$$Costo\ (en\ línea) = \sum_{i \geq 1} \pi_i\ costo(A_i) = \sum_{i \geq 1} \pi_i((i-1)x + y)$$

Resolviendo el sistema de ecuaciones para la igualdad tenemos

$$\pi_i = \begin{cases} \frac{\alpha - 1}{c}\left(\frac{c}{c-1}\right)^i, & i = 1 \ldots c\ (Geométrica) \\ 0, & i > 0 \end{cases}$$

Pero

$$\sum_{i \geq 1} \pi_i = 1 => \ \alpha = \frac{1}{\left(1 + \left(\frac{1}{c-1}\right)\right)^c - 1} + 1\ (este\ es\ lo\ mejor\ caso\ posible)$$

(En este caso *c* puede ser real y el tiempo es continuo)

Si $c = 2 => \ \alpha = \frac{4}{3} < 2\ (Mejor\ que\ el\ caso\ deterministico)$

Si $c \to \infty, \left(1 + \frac{1}{c-1}\right)^c \to e$

$$\alpha = \frac{e}{e-1} \approx 1.58$$

Así que $\alpha < 2\ \forall\ c$

4.1.1 Pseudocódigo algoritmo en línea

En este proyecto se implementó un algoritmo en línea, el cual tiene como propósito mejorar la precisión de un conjunto de documentos. Con el algoritmo implementado se logra determinar la cantidad de documentos nuevos necesarios para que la precisión aumente y se determine cuando rehacer el clúster. Para comprender de una forma más clara el funcionamiento de este algoritmo se implementó un pseudocódigo que se mostrará a continuación:

Definición de funciones:

- generarDocumentosAntiguos: Esta función recibe como parámetro de entrada el conjunto de términos antiguos, a partir del cual se genera el conjunto de documentos antiguos.
- generarDocumentosNuevos: Esta función recibe como parámetro de entrada el conjunto de términos antiguos y el conjunto de términos nuevos. Cada documento nuevo que genera la funcion está compuesto por un 66% de términos antiguos y un 33% de términos nuevos.
- generarConsultas: Esta función recibe como parámetro de entrada el conjunto de documentos antiguos. La función retorna las consultas construidas a partir de este conjunto.
- aplicarConsultaADocumentos: Esta función recibe como parámetro una consulta y un conjunto de documentos. La función retorna los documentos más similares a la consulta ordenados del más similar al menos similar.
- distribucionZeta: Esta función recibe como parámetro de entrada los documentos recuperados al aplicar una consulta. La función retorna la relevancia para cada uno de los documentos antiguos recuperados, retornando 1 si el documento es relevante y 0 si no lo es.
- agregarDoc: Esta función recibe como parámetro un documento relevante recuperado al aplicar una consulta y una lista de documentos relevantes. La función verifica si el documento relevante está dentro de la lista de documentos relevantes, si no se encuentra en la lista, entonces lo agrega.
- calcularPrecision: Esta función recibe como parámetro de entrada la relevancia de los documentos recuperados al aplicar una consulta sobre un conjunto de documentos. La función retorna la precisión calculada.

- extraerTerminos: Esta función recibe como parámetro de entrada los documentos antiguos relevantes. La función retorna los términos que se encuentran en los documentos relevantes.
- porcentajeTerminosRelevantes: Esta función recibe como parámetro de entrada un documento nuevo y la lista de términos relevantes. La función retorna el porcentaje de términos relevantes presentes en el documento nuevo.
- algoritmoProbabilistico: Esta función recibe como parámetro de entrada los documentos recuperados al aplicar una consulta sobre la lista de documentos unidos. La función retorna la relevancia para los documentos recuperados, retornando un 1 si el documento es relevante y un 0 si no lo es. Este algoritmo mantiene la relevancia de los documentos antiguos (obtenida por medio de la distribución Zeta) y aplica sobre los documentos nuevos un algoritmo probabilístico con el fin de simular los juicios de usuarios.

Definición de conjuntos y variables:

Sea terminosAntiguos un conjunto de términos antiguos.

Sea terminosNuevos un conjunto de términos nuevos.

Sea documentosAntiguos un conjunto donde se almacenan documentos antiguos, donde inicialmente documentosAntiguos = ∅.

Sea documentosNuevos un conjunto donde se almacenan documentos nuevos, donde inicialmente documentosNuevos = ∅.

Sea consultas un conjunto donde se almacenan consultas, donde inicialmente consultas = ∅.

Sea documentosRecuperados un conjunto donde se almacenan documentos antiguos recuperados al aplicar una consulta sobre el conjunto documentosAntiguos, donde inicialmente documentosRecuperados = ∅.

Sea relevanciaDocumentosAntiguos un vector donde se almacena la relevancia de los documentos recuperados al aplicar una consulta sobre el conjunto documentosAntiguos, donde inicialmente relevanciaDocumentosAntiguos = ∅.

Sea documentosRelevantes un subconjunto de documentosAntiguos, donde inicialmente documentosRelevantes = ∅.

Sea precisionAntiguaParcial un vector donde se almacena la precisión de los documentos antiguos para cada consulta, donde inicialmente precisionAntiguaParcial = ∅.

Sea listaTerminosRelevantes un conjunto donde se almacenan términos relevantes, donde inicialmente listaTerminosRelevantes = ∅.

Sea subconjuntoDocumentosNuevos un subconjunto donde se almacenan documentos nuevos que tienen una cantidad de términos relevantes >= 50%, donde inicialmente subconjuntoDocumentosNuevos = ∅.

Sea listaDocumentosUnidos un conjunto donde se almacenan documentos antiguos y documentos nuevos, donde inicialmente listaDocumentosUnidos = ∅.

Sea documentosRecuperadosUnidos un conjunto donde se almacenan documentos antiguos y nuevos recuperados al aplicar una consulta sobre el conjunto listaDocumentosUnidos, donde inicialmente documentosRecuperadosUnidos = ∅.

Sea relevanciaDocumentosUnidos un vector donde se almacena la relevancia de los documentos recuperados al aplicar una consulta sobre el conjunto listaDocumentosUnidos, donde inicialmente relevanciaDocumentosUnidos = ∅.

Sea precisionNuevaParcial un vector donde se almacena la precisión de los documentos unidos para cada consulta, donde inicialmente precisionNuevaParcial = ∅.

Sean i, j variables enteras.

Sean porcentaje, precisionAntigua, precisionNueva variables Reales.

```
1 terminosAntiguos;
2 terminosNuevos;
3 documentosAntiguos <- generarDocumentosAntiguos(terminosAntiguos);
4 documentosNuevos <- generarDocumentosNuevos(terminosAntiguos, terminosNuevos);
5 consultas <- generarConsultas(documentosAntiguos);
6 for i <- 1 to i <= consultas.length do
7       documentosRecuperados <- aplicarConsultaADocumentos(consultas[i],documentosAntiguos);
8        relevanciaDocumentosAntiguos <- distribucionZeta(documentosRecuperados);
9       for j <- 1 to j <= documentosRecuperados.length do
10              if relevanciaDocumentosAntiguos[i] = 1 then
11                      documentosRelevantes <- agregarDoc(documentosRecuperados[i],
                        documentosRelevantes);
12              ifEnd
13      forEnd
14      precisionAntiguaParcial <- calcularPrecision(relevanciaDocumentosAntiguos);
15 forEnd
16 listaTerminosRelevantes <- extraerTerminos(documentosRelevantes);
17 for i <- 1 to i <= documentosNuevos.length do
18      porcentaje <- porcentajeTerminosRelevantes(documentosNuevos[i], listaTerminosRelevantes);
19       if porcentaje >= 0,5 then
20              subconjuntoDocumentosNuevos <- subconjuntoDocumentosNuevos ∪ documentosNuevos[i];
21      ifEnd
```

```
22 forEnd
23 listaDocumentosUnidos <- documentosAntiguos ∪ subconjuntoDocumentosNuevos;
24 for i <- 1 to i <= consultas.length do
25        documentosRecuperadosUnidos <- aplicarConsultaADocumentos(consultas[i], listaDocumentosUnidos);
26        relevanciaDocumentosUnidos <- algoritmoProbabilistico(documentosRecuperadosUnidos);
27         precisionNuevaParcial <- calcularPrecision(relevanciaDocumentosUnidos);
28 forEnd
29 precisionAntigua <- 0;
30 precisionNueva <- 0;
31 for i <- 1 to i <= precisionAntiguaParcial.length do
32        precisionAntigua <- precisionAntigua + precisionAntiguaParcial[i];
33        precisionNueva <- precisionNueva + precisionNuevaParcial[i];
34 forEnd
35 precisionAntigua <- precisionAntigua / precisionAntiguaParcial.length;
36 precisionNueva <- precisionNueva / precisionNuevaParcial.length;
37 if precisionAntigua > precisionNueva then
38        //conviene rehacer el cluster
39 else
40        //no conviene rehacer el cluster
41 ifElseEnd
```

4.2 Framework de simulación

Desde nuestro punto de vista, existen documentos relevantes para una consulta dada por un usuario que no están siendo considerados. Nuestra suposición es que para una misma consulta existen nuevos documentos que podrían ser relevantes.

En esta sección se presenta un framework implementado por Gutiérrez-Soto (2016), el cual permite simular colecciones de RI tradicionales, las cuales están formadas por un conjunto de documentos, un conjunto de consultas y juicios de los usuarios sobre los documentos. Por lo tanto, uno de los principales objetivos de este framework es proporcionar un entorno ideal para evaluar enfoques basados en un conjunto documentos antiguos y documentos nuevos, donde para ambos conjuntos se aplica una misma consulta. Este framework será utilizado para la creación de documentos y consultas, recuperación de documentos potencialmente relevantes y simulación de juicios de usuarios. Este framework se basa en las leyes bien conocidas de la ciencia de la información, como las leyes de Zipf y Bradford, utilizadas en diversos campos de investigación como las Bibliotecas Digitales y la Recuperación de Información (Sparck Jones y Willett, 1997a, Schaer, 2013).

4.2.1 Creación de consultas y documentos

Para la creación de consultas y documentos se utilizó el framework propuesto por Gutiérrez-Soto (2016). Como en este proyecto se busca simular los documentos en línea, se hace necesaria la construcción de dos conjuntos de documentos. El primer conjunto de documentos corresponde al conjunto de documentos antiguos, el cual busca simular los documentos que se encuentran en el clúster, y el segundo conjunto corresponde al conjunto de documentos nuevos, que será utilizado para simular los documentos que se van incorporando constantemente en un contexto dinámico.

Para construir el conjunto de documentos antiguos y el conjunto de documentos nuevos, es necesario contar con dos listas de términos, una lista de términos antiguos y una lista de términos nuevos. Cada término está formado por entre 3 a 7 letras y cada letra se elige utilizando una distribución uniforme. Además, cada término es único y la intersección entre la

lista de términos antiguos y la lista de términos nuevos es vacía. El conjunto de documentos antiguos es construido con la lista de términos antiguos, cada documento del conjunto está formado por una cantidad de términos que puede variar entre 15 a 30. En cambio, el conjunto de documentos nuevos es construido con la lista de términos antiguos y la lista de términos nuevos, cada documento nuevo contiene un 66% de términos antiguos y un 33% de términos nuevos y al igual que en los documentos antiguos, la cantidad de términos puede variar entre 15 a 30. Así, por ejemplo si un documento tiene 21 términos, 14 de éstos términos corresponderán a términos de la lista de términos antiguos y 7 de ellos a la lista de términos nuevos.

Para crear la lista de consultas, se elige un documento antiguo para cada consulta utilizando una distribución uniforme. Los términos que constituyen la consulta se eligen del documento antiguo bajo la misma distribución, además una consulta está compuesta por una cantidad de 8 términos.

4.2.2 Simulación de juicios relevantes

Para la simulación de los juicios de usuario respecto de si un documento puede ser relevante para una consulta determinada, se ha utilizado la distribución Zeta. Esta distribución da un enfoque discreto de la ley de Bradford. La ley de Bradford dice que, entre la producción de artículos de revistas, hay un número heterogéneo de artículos donde los artículos más relevantes están en pocas revistas, mientras que un número reducido de artículos relevantes se difunden en una gran cantidad de revistas. En nuestro caso, para una consulta dada, significa que los documentos más relevantes deben estar al principio de la lista porque su similaridad con la consulta es más alta, mientras que, unos pocos documentos relevantes deben estar distribuidos en la parte inferior de la lista de documentos. Con el objetivo de simular este escenario, la distribución Zeta se utiliza para determinar los documentos relevantes para un conjunto de documentos antiguos. Sin embargo en este proyecto, además del conjunto de documentos antiguos se hace necesario contar con un conjunto de documentos nuevos que simulen un entorno en línea en donde se van agregando nuevos documentos constantemente. Así, si al conjunto de documentos antiguos se le agregan documentos del conjunto de documentos nuevos, se genera un nuevo conjunto que contiene

documentos antiguos y documentos nuevos, al aplicar una consulta dada al nuevo conjunto de documentos (conjunto con documentos antiguos y nuevos), se recuperan los documentos que hicieron match con la consulta. Para los documentos recuperados que pertenecen al conjunto de documentos antiguos se mantiene la relevancia calculada anteriormente mediante la distribución Zeta. Por otro lado, para los documentos recuperados que pertenecen al conjunto de documentos nuevos se hace necesario diseñar e implementar un nuevo algoritmo probabilístico, ya que la distribución Zeta sólo considera que los documentos más relevantes están ubicados en las primeras posiciones de la lista, pero no considera la cantidad de términos relevantes presentes en los documentos nuevos. Para la implementación de este algoritmo se consideraron dos criterios. En primer lugar, se tomó en cuenta la posición en la cual se encuentran los documentos nuevos en la lista de documentos recuperados, esto debido a que los primeros lugares de la lista contienen los documentos que realizan más match con la consulta, así un documento en la posición *i* tiene mayor probabilidad de ser relevante que un documento en la posición *i + 1*. En segundo lugar, se consideró la cantidad de términos relevantes presentes en el documento, ya que entre más términos relevantes tenga el documento nuevo, mayor probabilidad existe que el documento sea relevante. En la sección 4.2.2.1 se encuentra la implementación en pseudolenguaje c del algoritmo probabilístico.

Para ilustrar de mejor forma lo descrito anteriormente a continuación se mostrará un ejemplo. En la figura 4.4, donde se pueden observar dos columnas, en la primera de ellas se encuentran los documentos recuperados al aplicar una consulta al conjunto de documentos antiguos y en la segunda columna se pueden observar los documentos recuperados al aplicar la misma consulta al conjunto de documentos que contiene documentos antiguos y nuevos. El número que se encuentra a la derecha de los documentos indica la relevancia de los documentos con respecto a la consulta (1 indica que el documento es relevante y 0 indica que no lo es). Al interior del rectángulo de la figura 4.4 se encuentran los documentos que pertenecen al conjunto de documentos antiguos (d_3, d_5, d_6), por lo cual su relevancia se mantiene en ambas columnas, ésta relevancia fue determinada con la distribución Zeta al igual que la relevancia de los documentos que se encuentran en la primera columna y al exterior del rectángulo (d_1, d_8, d_9), en cambio los documentos que se encuentran en la segunda columna y al exterior del rectángulo (d_2, d_7, d_{10}), corresponden al conjunto de documentos nuevos, y su relevancia fue calculada con el algoritmo probabilístico mencionado al final del párrafo anterior.

Q (primera columna): Corresponde a los documentos recuperados al aplicar una consulta al conjunto de documentos antiguos.

Q (segunda columna): Corresponde a los documentos recuperados al aplicar la consulta al conjunto de documentos antiguos y nuevos.

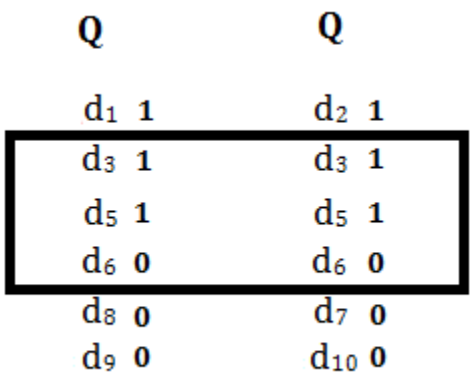

Figura 4.4. Obtención de juicios de usuarios.

4.2.2.1 Algoritmo probabilístico

Al aplicar una consulta a la unión del conjunto de documentos antiguos con el conjunto de documentos nuevos, se obtiene una lista de documentos recuperados que hicieron match con la consulta. En esta lista se pueden encontrar documentos antiguos y documentos nuevos. Para obtener la relevancia de los documentos antiguos, se mantiene la relevancia obtenida anteriormente al aplicar la distribución Zeta solo al conjunto de documentos antiguos, esta distribución no fue implementada en este proyecto, pues ya se encontraba implementada en el framework desarrollado por Gutiérrez-Soto (2016). Por otro lado, para obtener la relevancia de los documentos nuevos, es necesario considerar dos criterios. El primero de éstos considera la posición en la cual se encuentra el documento nuevo, ya que los documentos se encuentran ordenados desde el más similar a la consulta hasta el menos similar. El segundo criterio considera la cantidad de términos relevantes presentes en el documento nuevo, ya que entre más términos relevantes tenga el documento, mayor será la probabilidad de que el documento sea relevante. Es por esto, que para obtener la relevancia de los documentos nuevos se implementó un algoritmo probabilístico en pseudolenguaje c, el cual mostraremos a continuación:

```
void algoritmoProbabilistico(int AuxOld[], int Arroba30, DOCUMENT headD, DOCUMENT
listaDocRel ,int NumberDocuments, short unsigned int soloDRel[]){

/*Esta función simula los juicios de usuarios para los documentos nuevos y mantiene la
relevancia de los documentos antiguos.

 El parámetro AuxOld contiene las id de los documentos recuperados por la consulta
(documentos nuevos y antiguos), Arroba30 indica el tamaño de AuxOld.

 headD es el conjunto de documentos antiguos y la listaDocRel son los documentos nuevos que
tienen más de un 50% de términos relevantes.

 NumberDocuments corresponde a la última id del conjunto de documentos antiguos(después de
este número empiezan las id de los documentos nuevos).

 soloDRel contiene las id de los documentos antiguos relevantes recuperados al aplicar la
consulta solo al conjunto de documentos antiguos. */

 int Aux[30],num,random,i,stop;

 float p,p2;

 DOCUMENT d;

 for(i = 0; i < 30 && AuxOld[i] != 0; ++i)

 {

  stop = 0;

  if(AuxOld[i] <= NumberDocuments)//si se entra a este if es porque el i-esimo documento es
un documento antiguo

  {

   if(estaIdDoc(soloDRel,AuxOld[i]) == 1)// la funcion estaIdDoc retorna un 1 si la id del
documento (AuxOld[i]) esta en la lista soloDRel

   {                    // y retorna un 0 si no esta

    Aux[i] = 1;
```

```
      stop = 1;
      continue;
    }
    else{//deja en 0 los documentos antiguos que no son relevantes
      Aux[i] = 0;
      stop = 1;
      continue;
    }
    d = buscarDocPorId(headD,AuxOld[i]);//la funcion buscarDocPorId utiliza la id AuxOld[i] para retornar el documento representado por esa id
  }
  else{
   d = buscarDocPorId(listaDocRel,AuxOld[i]);
  }
  if(stop == 0)
  {
    p = d->porcentajeTerRelevantes;//porcentaje de terminos relevantes presente en un documento nuevo
    p2 = 51.0 - ((float)i) * 1.7;// este porcentaje dependera de la posicion del documento en la lista
    p2 = p2 / 100.0;
    num = ceil(((p * 100.0) + (p2 * 100.0)));
    random = (rand() % 100) + 1;
```

```
    if (random <= num)
    {
      Aux[i] = 1;
    }
    else{
      Aux[i] = 0;
    }
   }
 }
 for(i=0;i<30;i++)
   AuxOld[i]=Aux[i];
} // Fin del algoritmo
```

4.3 Experimentos

4.3.1 Entorno experimental

Para llevar a cabo el entorno experimental en un contexto dinámico, se deben considerar dos conjuntos de documentos, el primero de ellos es el conjunto de documentos antiguos el cual se encuentra agrupado en un clúster y el segundo conjunto es el conjunto de documentos nuevos, que corresponde a documentos que se van incorporando constantemente en el tiempo, esto con el fin de evitar la pérdida de información (por ejemplo documentos no considerados). Para simular esta problemática, se utilizó el framework de simulación implementado por Gutiérrez-Soto (2016) con el cual se construyeron la lista de términos antiguos (1400 términos) y la lista de términos nuevos (1400 términos). También, fue posible construir el conjunto de documentos antiguos (con la lista de términos antiguos) y el conjunto de documentos nuevos (con la lista de términos antiguos y nuevos). Las consultas fueron

construidas a partir del conjunto de documentos antiguos. El proceso de construcción de términos, documentos y consultas esta explicado más detalladamente en la sección 4.2.1.

4.3.2 Resultados Empíricos

Esta sección se divide en dos partes, primero que nada se realizarán experimentos relacionados a la precisión de dos conjuntos de documentos, el primer conjunto de documentos corresponde a documentos antiguos y el segundo conjunto de documentos corresponde a documentos antiguos y nuevos. Esta primera parte de experimentos se realiza para determinar cuándo es conveniente rehacer el clúster. La segunda parte tiene por objetivo analizar con cuál de los principales algoritmos de clustering jerárquico resulta más conveniente rehacer el clúster de documentos.

4.3.3 Primera parte experimentos (precisión):

Inicialmente al conjunto de documentos antiguos se aplica una consulta obtenida de la lista de consultas. Como resultado de aplicar dicha consulta se obtiene una lista de documentos recuperados que hicieron la mayor cantidad de match con la consulta. Para obtener los documentos relevantes de la lista recuperada, se hace uso de la distribución Zeta, esta distribución esta explicada detalladamente en la sección 4.2.2.

Una vez obtenida la relevancia para los documentos recuperados (documentos antiguos) es posible construir una lista con todos los términos relevantes (se consideran relevantes todos los términos que pertenecen a un documentos relevante). Esta lista de términos relevantes será utilizada para construir el conjunto de documentos nuevos, pues como cada documento de este conjunto contiene un 66% de términos antiguos y un 33% de términos nuevos, podrá utilizarse la lista de términos relevantes (antiguos) para saber cuántos de los términos antiguos de cada documento nuevo son relevantes.

Como para cada documento del conjunto de documentos nuevos, se conoce la cantidad de términos relevantes que contiene, es posible generar una lista con todos los documentos nuevos que poseen más de un 50% de términos relevantes. Luego, dicha lista se une con el

conjunto de documentos antiguos, esta unión que contiene todo el conjunto de documentos antiguos junto a la lista de documentos nuevos que tienen más de un 50% de términos relevantes, la denominaremos "lista de documentos unidos".

Una vez obtenida la lista de documentos unidos, se debe aplicar a esta lista la misma consulta que se aplicó al conjunto de documentos antiguos, como resultado de aplicar esta consulta, se obtiene una nueva lista de documentos recuperados que hicieron la mayor cantidad de match con la consulta, esta nueva lista puede contener tanto documentos antiguos como documentos nuevos. Para los documentos antiguos que se encuentran en la lista de documentos unidos, se mantiene la relevancia obtenida anteriormente con la distribución Zeta, en cambio para obtener la relevancia de los documentos nuevos se hizo necesario implementar un nuevo algoritmo probabilístico, ya que la distribución Zeta sólo considera que los documentos más relevantes están ubicados en las primeras posiciones de la lista, pero no considera la cantidad de términos relevantes presentes en los documentos nuevos. Por lo tanto, para obtener la relevancia de los documentos nuevos, se implementó el algoritmo probabilístico considerando dos criterios. El primero de éstos considera la posición en la cual se encuentra el documento nuevo, ya que los documentos se encuentran ordenados desde el más similar a la consulta hasta el menos similar. El segundo criterio considera la cantidad de términos relevantes presentes en el documento nuevo, ya que entre más términos relevantes tenga el documento, mayor será la probabilidad de que el documento sea relevante.

Después de obtener la relevancia para el conjunto de documento antiguos y la lista de documentos unidos, se hace necesario calcular la precisión para ambos conjuntos de documentos. Con el fin de analizar, si al añadir documentos nuevos al conjunto de documentos antiguos la precisión mejora. A continuación, en la tabla 4.1 mostraremos por medio de un ejemplo el proceso necesario para calcular la precisión.

Documentos: Corresponde a los documentos más similares (usando medida del coseno) con la consulta. En la tabla 4.1 el documento 1(D1) es el más similar a la consulta.

Z: Indica si los documentos son o no relevantes.

X: Es la cantidad de documentos relevantes parciales dividido por la cantidad de documentos parciales.

n: Corresponde a la cantidad de documentos relevantes acumulados.

m: Corresponde a la cantidad de documentos (30).

Documentos	Z	X
D1	1	1/1
D2	0	1/2
D3	1	2/3
D4	0	2/4
D5	0	2/5
D6	1	3/6
. . . .		. . . .
Dm	0	n/m

Tabla 4.1. Procedimiento para calcular la precisión.

$$\text{Precisión} = \frac{\sum_{i=1}^{30} X_i}{m}$$

Una vez calculada la precisión para ambos conjuntos de documentos (conjunto de documentos antiguos, lista de documentos unidos), se debe analizar si es que la precisión obtenida para la lista de documentos unidos mejoró o empeoró con respecto a la precisión obtenida para el conjunto de documentos antiguos. Si se da el caso en que la precisión mejora, se deben almacenar dos listas de documentos relevantes. En la primera lista deben almacenarse todos los documentos relevantes del conjunto de documentos antiguos y en la segunda lista deben almacenarse todos los documentos relevantes de la lista de documentos unidos. Este proceso de almacenamiento es necesario, puesto que los documentos relevantes de ambos conjuntos se utilizaran en la segunda parte de los experimentos (sección 4.3.4).

4.3.3.1 Experimentos de precisión

Experimento 1: En el primer experimento se utilizaron dos conjuntos de documentos, el primer conjunto de documentos corresponde al conjunto de documentos antiguos y el segundo conjunto corresponde al conjunto de documentos nuevos. La cantidad de documentos antiguos se mantuvo en 700. Por otro lado, la cantidad de documentos nuevos varía en 700, 1400 y 3500. A continuación se muestra la tabla 4.2 con los resultados obtenidos con las distintas

cantidades de documentos, además se pueden observar gráficamente los resultados mediante la figura 4.2:

Precisión Antigua: Corresponde a la precisión calculada con la relevancia de los documentos recuperados al aplicar la consulta al conjunto de documentos antiguos.

Precisión Nueva: Corresponde a la precisión calculada con la relevancia de los documentos recuperados al aplicar la consulta a la "lista de documentos unidos".

Doc. Antiguos	Doc. Nuevos	Precisión Antigua	Precisión Nueva
700	700	0,350806	0,589211
700	1.400	0,355886	0,689694
700	3.500	0,322908	0,744821

Tabla 4.2. Resultados primer experimento de precisión antes de aplicar algoritmos de clustering.

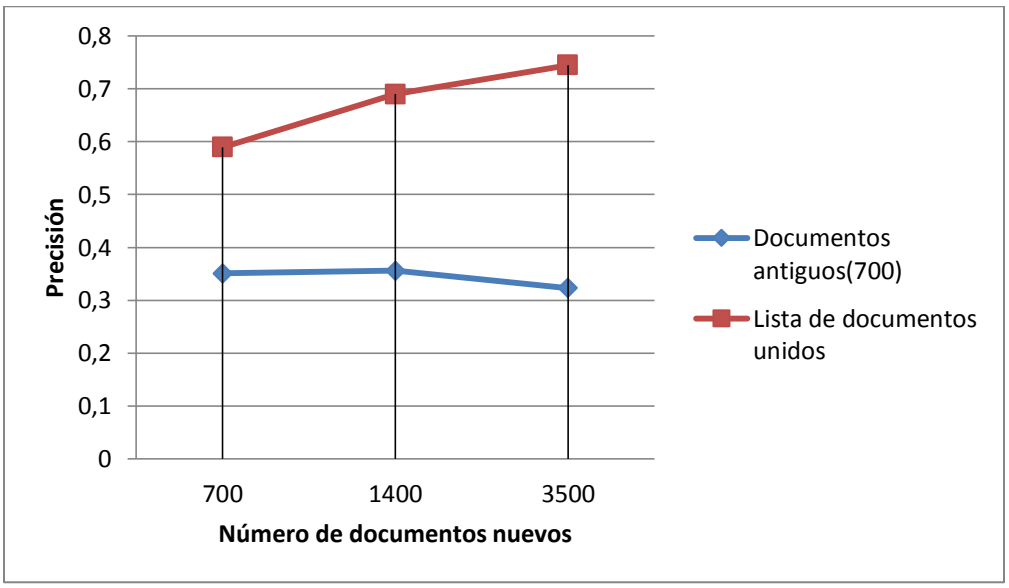

Figura 4.5. Gráfico resultados primer experimento precisión, donde las abscisas representan la mejora de precisión al aumentar la cantidad de documentos nuevos que se agregan al conjunto de documentos antiguos (que se mantiene en 700 en los tres casos).

Experimento 2:

En el segundo experimento se utilizaron dos conjuntos de documentos, el primer conjunto de documentos corresponde al conjunto de documentos antiguos y el segundo conjunto corresponde al conjunto de documentos nuevos. La cantidad de documentos antiguos se mantuvo en 1.400. Por otro lado, la cantidad de documentos nuevos varía en 700, 1400 y 3500. A continuación se muestra la tabla 4.3 con los resultados obtenidos con las distintas cantidades de documentos, además se pueden observar gráficamente los resultados mediante la figura 4.3:

Doc. Antiguos	**Doc. Nuevos**	**Precisión antigua**	**Precisión nueva**
1.400	700	0,351264	0,589211
1.400	1.400	0,342910	0,448431
1.400	3.500	0,331102	0,534738

Tabla 4.3. Resultados segundo experimento de precisión antes de aplicar algoritmos de clustering.

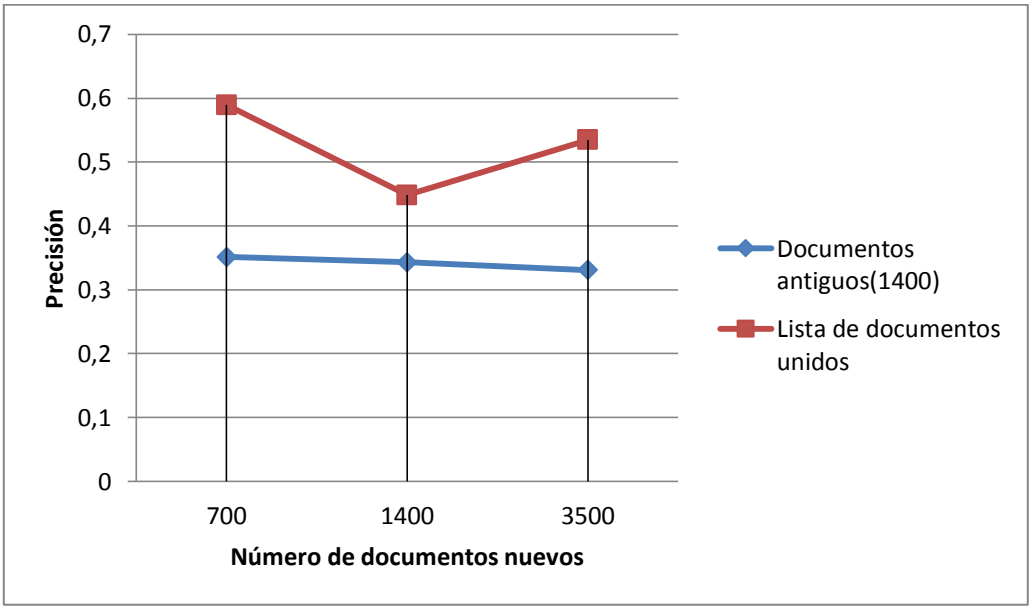

Figura 4.6. Gráfico resultados segundo experimento precisión, donde las abscisas representan la mejora de precisión al aumentar la cantidad de documentos nuevos que se agregan al conjunto de documentos antiguos (que se mantiene en 1.400 en los tres casos).

Experimento 3:

En el tercer experimento se utilizaron dos conjuntos de documentos, el primer conjunto de documentos corresponde al conjunto de documentos antiguos y el segundo conjunto corresponde al conjunto de documentos nuevos. La cantidad de documentos antiguos se mantuvo en 3.500. Por otro lado, la cantidad de documentos nuevos varía en 700, 1400 y 3500. A continuación se muestra la tabla 4.4 con los resultados obtenidos con las distintas cantidades de documentos, además se pueden observar gráficamente los resultados mediante la figura 4.4:

Doc. Antiguos	**Doc. Nuevos**	**Precisión antigua**	**Precisión nueva**
3.500	700	0,327009	0,344603
3.500	1.400	0,342851	0,376291
3.500	3.500	0,328145	0,378947

Tabla 4.4. Resultados tercer experimento de precisión antes de aplicar algoritmos de clustering.

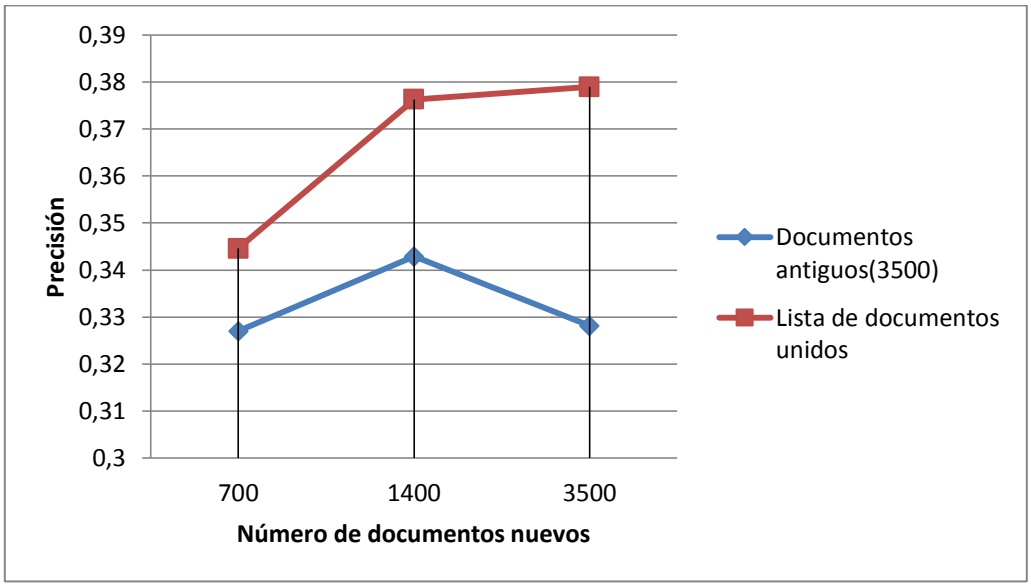

Figura 4.7. Gráfico resultados tercer experimento precisión, donde las abscisas representan la mejora de precisión al aumentar la cantidad de documentos nuevos que se agregan al conjunto de documentos antiguos (que se mantiene en 3.500 en los tres casos).

4.3.4 Segunda Parte experimentos (clustering):

Esta segunda parte de experimentos es posible llevarla a cabo debido a que en la primera parte de los experimentos, desarrollados en la sección 4.3.3, se pudo observar que al agregar documentos nuevos al clúster de documentos antiguos, la precisión mejoraba cada vez más a medida que se iban incorporando nuevos documentos. Como en los experimentos realizados, la precisión siempre mejoró cuando al menos se agregaban 700 documentos nuevos, sabemos que resulta conveniente rehacer el clúster cuando los documentos nuevos igualan o superan dicha cantidad (para una cantidad menor a 700 documentos nuevos no se realizaron pruebas, por lo que no se puede asegurar si la precisión mejora o empeora para esos casos).

Como ya se sabe que resulta conveniente rehacer el clúster, surge ahora la interrogante de cuál algoritmo de clustering jerárquico, implementado en este proyecto (Single Link, Complete Link, Average Link o Ward's), obtendrá mejores resultados al rehacer el clúster. Esta segunda parte de experimentos busca responder dicha interrogante rehaciendo los clústeres con los algoritmos de clustering implementados, analizando cuantos documentos relevantes son visitados al recorrer los nuevos clústeres.

Antes de rehacer el clúster es necesario conocer cuáles son los documentos relevantes tanto del conjunto de documentos antiguos como también de la lista de documentos unidos, pues éstos serán necesarios para evaluar cuantos documentos relevantes son visitados al recorrer ambos clústeres. Para saber cuáles son los documentos relevantes del conjunto de documentos antiguos y lista de documentos unidos, al final de la primera parte de los experimentos (sección 4.3.3), se almacenaron dos listas de documentos relevantes, una para el conjunto de documentos antiguos y otra para la lista de documentos unidos.

Para rehacer el clúster, además de contar con los documentos relevantes se debe construir una matriz de similaridad entre documentos, para el conjunto de documentos antiguos y otra matriz de similaridad entre documentos, para la lista de documentos unidos, éstas matrices se utilizarán como entrada para los algoritmos de clustering implementados. Para construir la matriz de similaridad entre documentos se utiliza la medida de distancia del coseno. Un ejemplo de cómo se construyó la matriz de similaridad, se puede encontrar en el capítulo 2, en la sección 2.5.1.

Para generar los clústeres se utilizaron diferentes algoritmos de clustering (Single Link, Complete Link, Average Link o Ward's), con la finalidad de observar en base a los resultados obtenidos, cuál de estos algoritmos logra rehacer de mejor forma el clúster, es decir que algoritmo forma el clúster que considera la mayor cantidad de documentos relevantes al ser visitado. A continuación se muestran los resultados obtenidos por medio de los experimentos:

4.3.4.1 Experimentos con el algoritmo complete link

Experimento 1: Para el primer experimento se realizó el proceso de clustering con el algoritmo jerárquico Complete Link. La cantidad de documentos antiguos con los que se realizaron los experimentos se mantuvo en 700, mientras que la cantidad de documentos en la lista de documentos unidos fue 700, 1.400 y 3.500. A continuación podemos observar la tabla 4.5 la cual contiene los resultados obtenidos en el experimento:

CDRCA: Cantidad de documentos relevantes visitados al recorrer el clúster generado con los documentos antiguos.

CDRCU: Cantidad de documentos relevantes visitados al recorrer el clúster generado con la lista de documentos unidos.

	Doc. Antiguos	**Doc. Nuevos**	**CDRCA**	**CDRCU**
Complete Link	700	700	231	236
	700	1.400	208	220
	700	3.500	202	240

Tabla 4.5. Resultados primer experimento de clustering con el algoritmo Complete Link de la cantidad de documentos relevantes visitados al recorrer el clúster generado.

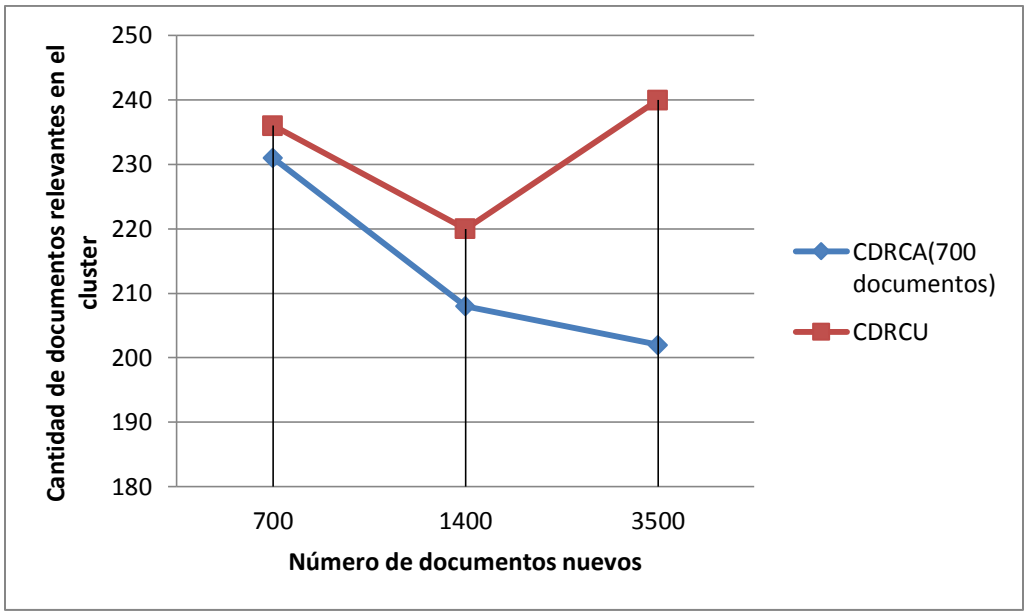

Figura 4.8. Resultados primer experimento de clustering usando el algoritmo Complete Link, donde las abscisas representan como varía la cantidad de documentos relevantes visitados al recorrer el clúster generado, aumentando la cantidad de documentos nuevos que se agregan al conjunto de documentos antiguos (que se mantiene en 700 en los tres casos).

Experimento 2: Para el segundo experimento se realizó el proceso de clustering con el algoritmo jerárquico Complete Link. La cantidad de documentos antiguos con los que se realizaron los experimentos se mantuvo en 1.400, mientras que la cantidad de documentos en la lista de documentos unidos fue 700, 1.400 y 3.500. A continuación podemos observar la tabla 4.6 la cual contiene los resultados obtenidos en el experimento:

	Doc. Antiguos	**Doc. Nuevos**	**CDRCA**	**CDRCU**
Complete Link	1.400	700	242	248
	1.400	1.400	229	226
	1.400	3.500	256	253

Tabla 4.6. Resultados segundo experimento de clustering con el algoritmo Complete Link de la cantidad de documentos relevantes visitados al recorrer el clúster generado.

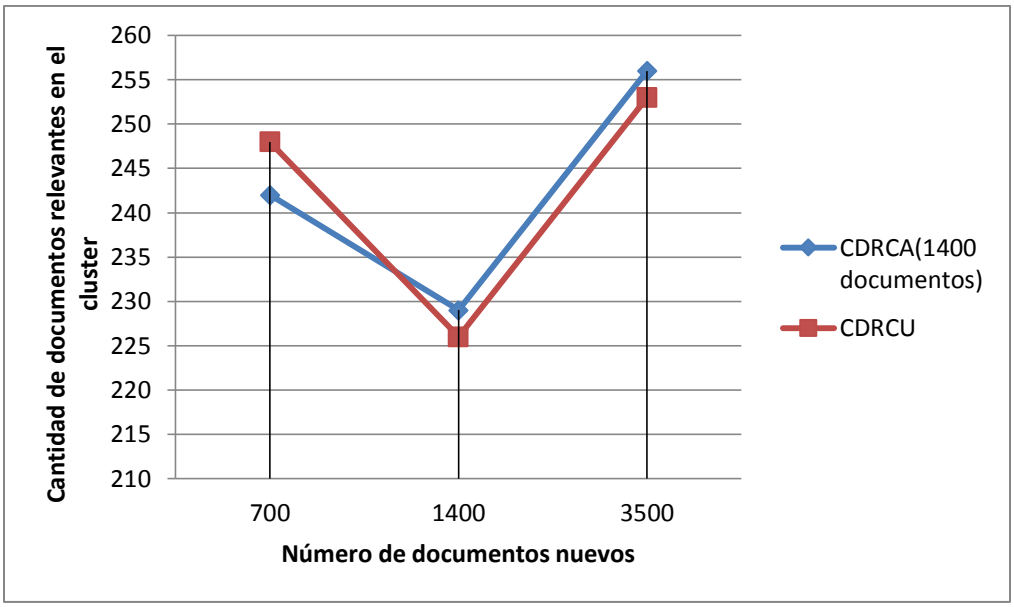

Figura 4.9. Resultados segundo experimento de clustering usando el algoritmo Complete Link, donde las abscisas representan como varía la cantidad de documentos relevantes visitados al recorrer el clúster generado, aumentando la cantidad de documentos nuevos que se agregan al conjunto de documentos antiguos (que se mantiene en 1.400 en los tres casos).

Experimento 3: Para el tercer experimento se realizó el proceso de clustering con el algoritmo jerárquico Complete Link. La cantidad de documentos antiguos con los que se realizaron los experimentos se mantuvo en 3.500, mientras que la cantidad de documentos en la lista de documentos unidos fue 700, 1.400 y 3.500. A continuación podemos observar la tabla 4.7 la cual contiene los resultados obtenidos en el experimento:

	Doc. Antiguos	**Doc. Nuevos**	**CDRCA**	**CDRCU**
Complete Link	3.500	700	247	236
	3.500	1.400	232	225
	3.500	3.500	272	250

Tabla 4.7. Resultados tercer experimento de clustering con el algoritmo Complete Link de la cantidad de documentos relevantes visitados al recorrer el clúster generado.

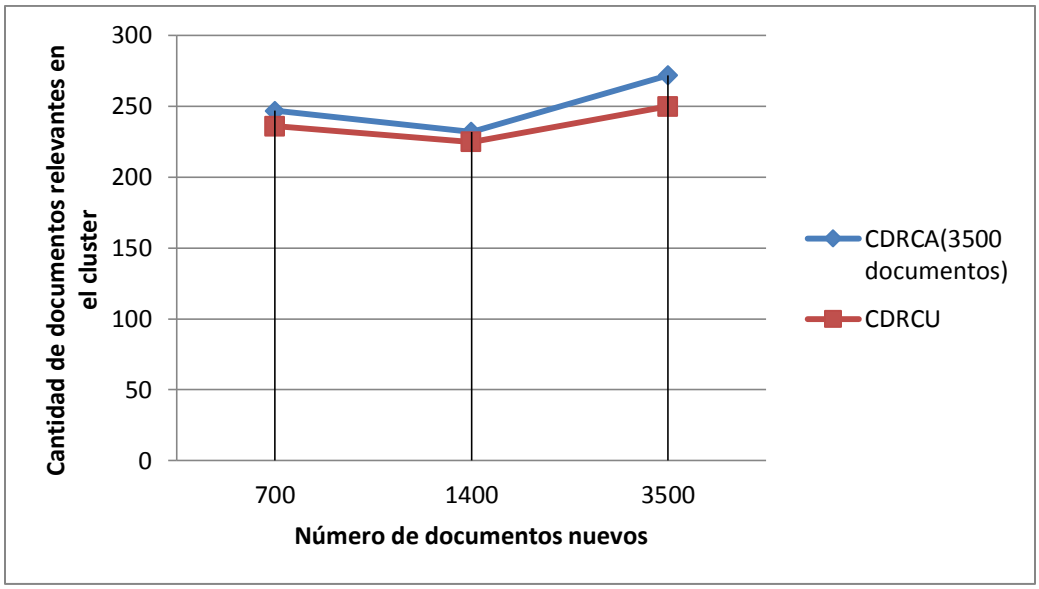

Figura 4.10. Resultados tercer experimento de clustering usando el algoritmo Complete Link, donde las abscisas representan como varía la cantidad de documentos relevantes visitados al recorrer el clúster generado, aumentando la cantidad de documentos nuevos que se agregan al conjunto de documentos antiguos (que se mantiene en 3.500 en los tres casos).

4.3.4.2 Experimentos con el algoritmo Average Link

Experimento 4: Para el cuarto experimento se realizó el proceso de clustering con el algoritmo jerárquico Average Link. La cantidad de documentos antiguos con los que se realizaron los experimentos se mantuvo en 700, mientras que la cantidad de documentos en la lista de documentos unidos fue 700, 1.400 y 3.500. A continuación podemos observar la tabla 4.8 la cual contiene los resultados obtenidos en el experimento:

	Doc. Antiguos	**Doc. Nuevos**	**CDRCA**	**CDRCU**
Average Link	700	700	2.039	2.403
	700	1.400	1.944	2.242
	700	3.500	1.870	2.905

Tabla 4.8. Resultados cuarto experimento de clustering con el algoritmo Average Link de la cantidad de documentos relevantes visitados al recorrer el clúster generado.

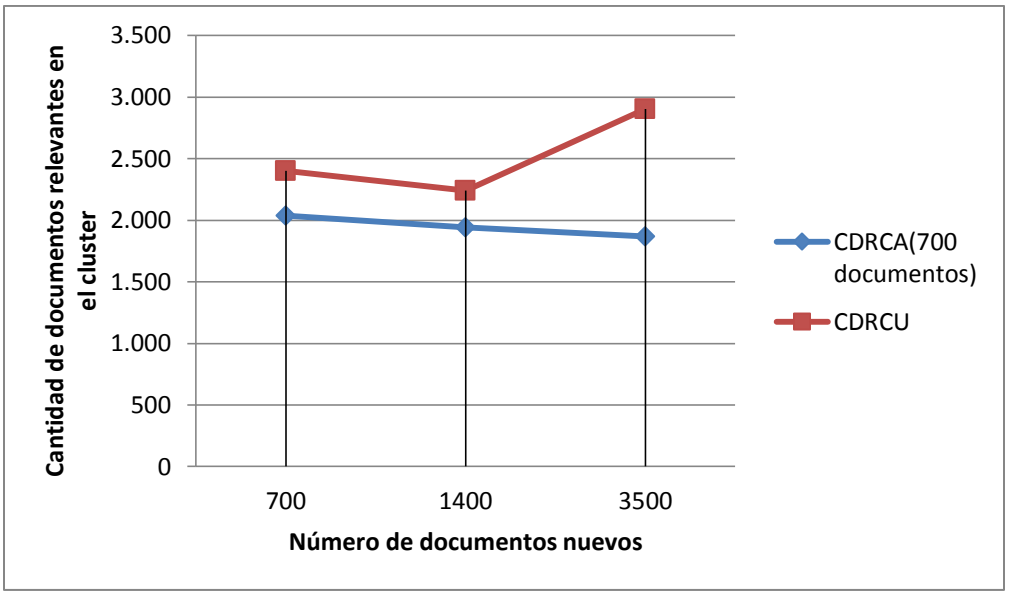

Figura 4.11. Resultados cuarto experimento de clustering usando el algoritmo Average Link, donde las abscisas representan como varía la cantidad de documentos relevantes visitados al recorrer el clúster generado, aumentando la cantidad de documentos nuevos que se agregan al conjunto de documentos antiguos (que se mantiene en 700 en los tres casos).

Experimento 5: Para el quinto experimento se realizó el proceso de clustering con el algoritmo jerárquico Average Link. La cantidad de documentos antiguos con los que se realizaron los experimentos se mantuvo en 1.400, mientras que la cantidad de documentos en la lista de documentos unidos fue 700, 1.400 y 3.500. A continuación podemos observar la tabla 4.9 la cual contiene los resultados obtenidos en el experimento:

	Doc. Antiguos	**Doc. Nuevos**	**CDRCA**	**CDRCU**
Average Link	1.400	700	2.067	2.560
	1.400	1.400	2.013	2.808
	1.400	3.500	2.594	3.066

Tabla 4.9. Resultados quinto experimento de clustering con el algoritmo Average Link de la cantidad de documentos relevantes visitados al recorrer el clúster generado.

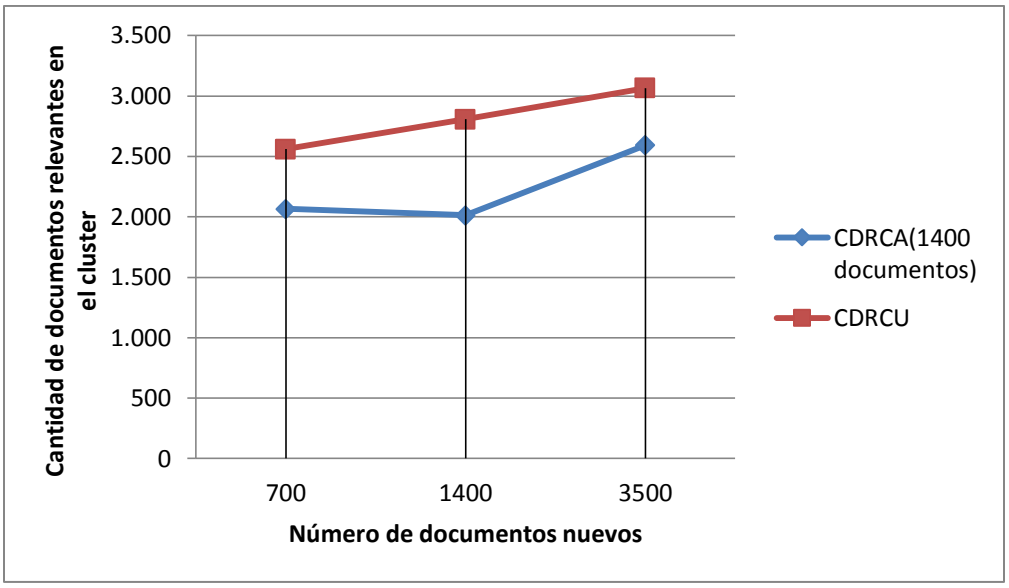

Figura 4.12. Resultados quinto experimento de clustering usando el algoritmo Average Link, donde las abscisas representan como varía la cantidad de documentos relevantes visitados al recorrer el clúster generado, aumentando la cantidad de documentos nuevos que se agregan al conjunto de documentos antiguos (que se mantiene en 1.400 en los tres casos).

Experimento 6: Para el sexto experimento se realizó el proceso de clustering con el algoritmo jerárquico Average Link. La cantidad de documentos antiguos con los que se realizaron los experimentos se mantuvo en 3.500, mientras que la cantidad de documentos en la lista de documentos unidos fue 700, 1.400 y 3.500. A continuación podemos observar la tabla 4.10 la cual contiene los resultados obtenidos en el experimento:

	Doc. Antiguos	**Doc. Nuevos**	**CDRCA**	**CDRCU**
Average Link	3.500	700	2.778	3.169
	3.500	1.400	2.822	3.335
	3.500	3.500	2.674	3.218

Tabla 4.10. Resultados sexto experimento de clustering con el algoritmo Average Link de la cantidad de documentos relevantes visitados al recorrer el clúster generado.

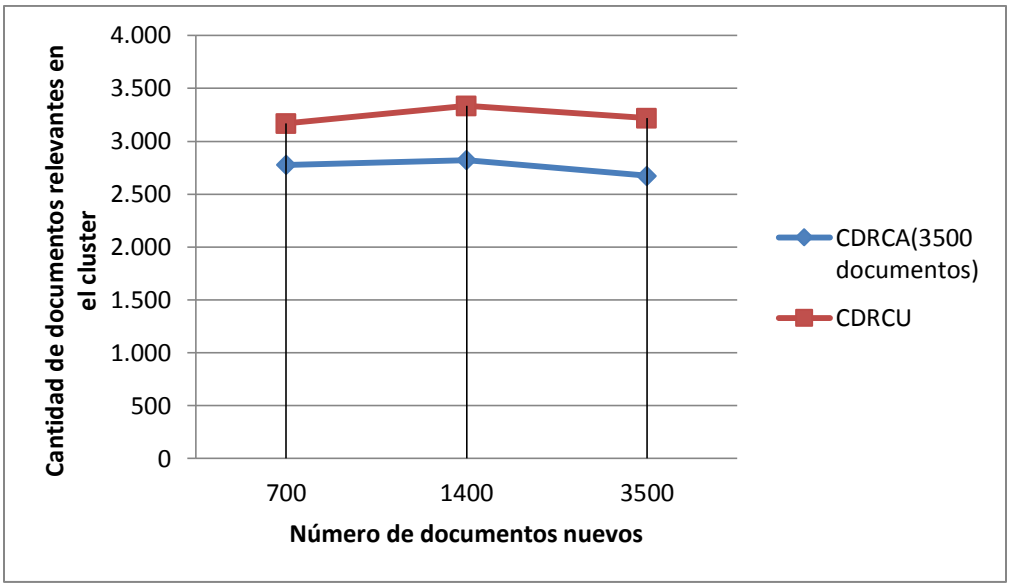

Figura 4.13. Resultados sexto experimento de clustering usando el algoritmo Average Link, donde las abscisas representan como varía la cantidad de documentos relevantes visitados al recorrer el clúster generado, aumentando la cantidad de documentos nuevos que se agregan al conjunto de documentos antiguos (que se mantiene en 3.500 en los tres casos).

4.3.4.3 Experimentos con el algoritmo Single Link

Experimento 7: Para el séptimo experimento se realizó el proceso de clustering con el algoritmo jerárquico Single Link. La cantidad de documentos antiguos con los que se realizaron los experimentos se mantuvo en 700, mientras que la cantidad de documentos en la lista de documentos unidos fue 700, 1.400 y 3.500. A continuación podemos observar la tabla 4.11 la cual contiene los resultados obtenidos en el experimento:

	Doc. Antiguos	**Doc. Nuevos**	**CDRCA**	**CDRCU**
Single Link	700	700	221	243
	700	1.400	234	230
	700	3.500	234	224

Tabla 4.11. Resultados séptimo experimento de clustering con el algoritmo Single Link de la cantidad de documentos relevantes visitados al recorrer el clúster generado.

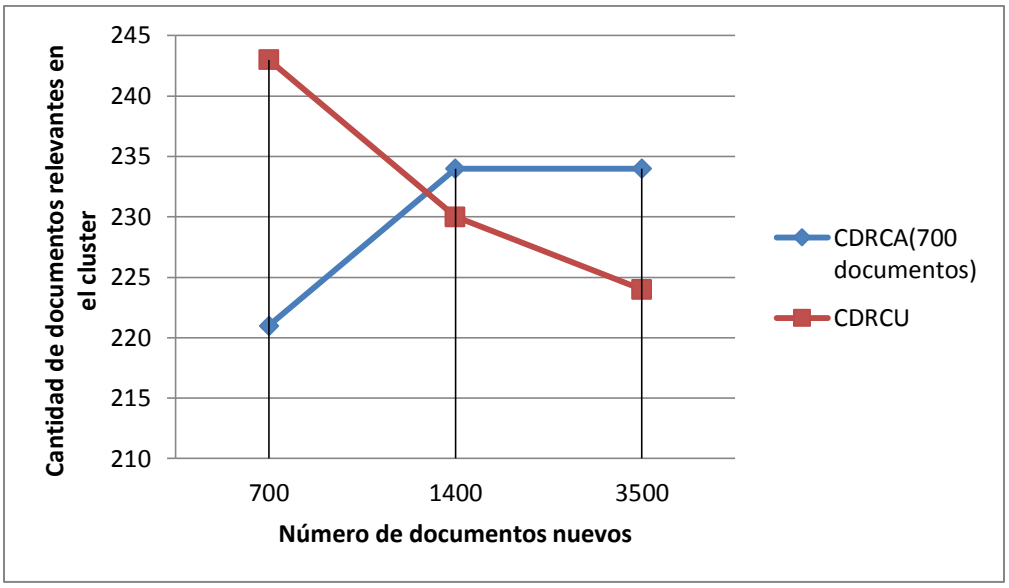

Figura 4.14. Resultados séptimo experimento de clustering usando el algoritmo Single Link, donde las abscisas representan como varía la cantidad de documentos relevantes visitados al recorrer el clúster generado, aumentando la cantidad de documentos nuevos que se agregan al conjunto de documentos antiguos (que se mantiene en 700 en los tres casos).

Experimento 8: Para el octavo experimento se realizó el proceso de clustering con el algoritmo jerárquico Single Link. La cantidad de documentos antiguos con los que se realizaron los experimentos se mantuvo en 1.400, mientras que la cantidad de documentos en la lista de documentos unidos fue 700, 1.400 y 3.500. A continuación podemos observar la tabla 4.12 la cual contiene los resultados obtenidos en el experimento:

	Doc. Antiguos	**Doc. Nuevos**	**CDRCA**	**CDRCU**
Single Link	1.400	700	247	240
	1.400	1.400	229	231
	1.400	3.500	234	212

Tabla 4.12. Resultados octavo experimento de clustering con el algoritmo Single Link de la cantidad de documentos relevantes visitados al recorrer el clúster generado.

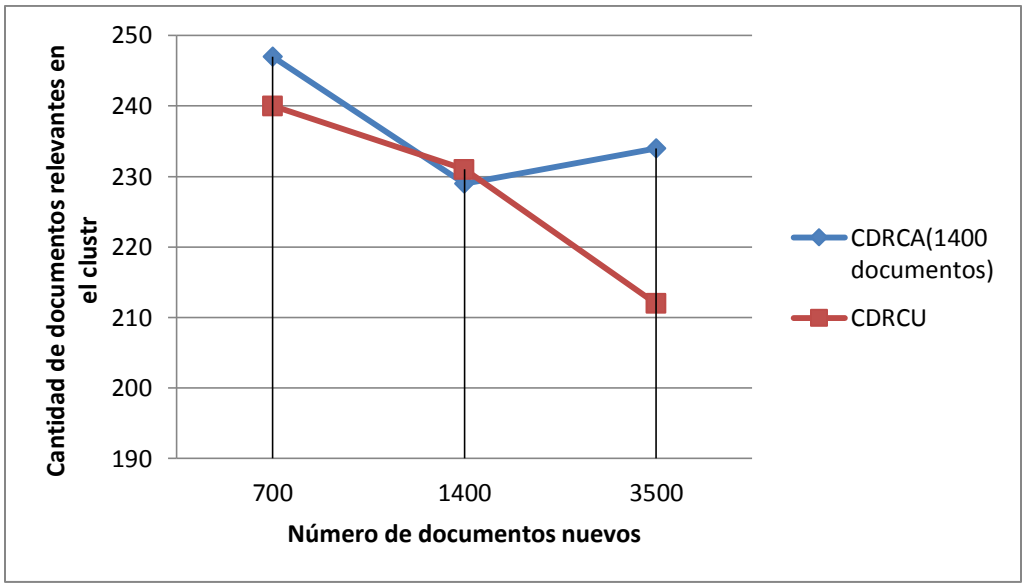

Figura 4.15. Resultados octavo experimento de clustering usando el algoritmo Single Link, donde las abscisas representan como varía la cantidad de documentos relevantes visitados al recorrer el clúster generado, aumentando la cantidad de documentos nuevos que se agregan al conjunto de documentos antiguos (que se mantiene en 1.400 en los tres casos).

Experimento 9: Para el noveno experimento se realizó el proceso de clustering con el algoritmo jerárquico Single Link. La cantidad de documentos antiguos con los que se realizaron los experimentos se mantuvo en 3.500, mientras que la cantidad de documentos en la lista de documentos unidos fue 700, 1.400 y 3.500. A continuación podemos observar la tabla 4.13 la cual contiene los resultados obtenidos en el experimento.

	Doc. Antiguos	**Doc. Nuevos**	**CDRCA**	**CDRCU**
Single Link	3.500	700	222	227
	3.500	1.400	225	223
	3.500	3.500	240	212

Tabla 4.13. Resultados noveno experimento de clustering con el algoritmo Single Link de la cantidad de documentos relevantes visitados al recorrer el clúster generado.

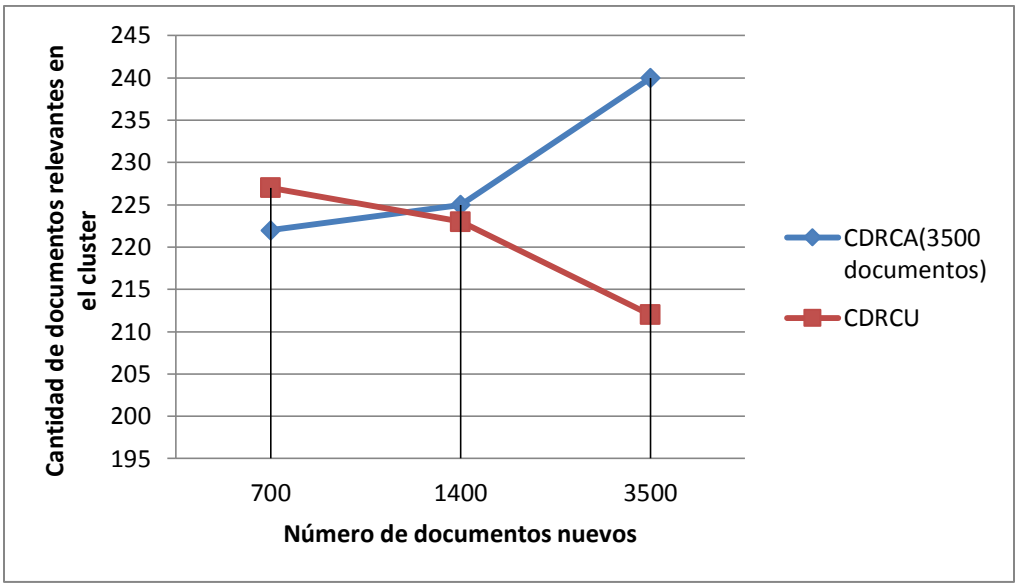

Figura 4.16. Resultados noveno experimento de clustering usando el algoritmo Single Link, donde las abscisas representan como varía la cantidad de documentos relevantes visitados al recorrer el clúster generado, aumentando la cantidad de documentos nuevos que se agregan al conjunto de documentos antiguos (que se mantiene en 3.500 en los tres casos).

4.3.4.4 Experimentos con el algoritmo Ward's

Para los experimentos del algoritmo de clustering Ward's es necesario utilizar la distancia euclidiana a diferencia de los otros algoritmos de clustering utilizados en este trabajo, los cuales utilizan la distancia del coseno como entrada para generar los clústeres. Es por esto, que para poder realizar correctamente los experimentos del algoritmo Ward's es necesario realizar una transformación para todos los experimentos realizados anteriormente. A continuación explicaremos más detalladamente la transformación que se debería llevar a cabo.

Lance y Williams (1967) han demostrado que existe una ecuación combinatoria general que puede usarse para describir cómo los diferentes métodos de clustering jerárquico actualizan la matriz de similitud después de una fusión de dos objetos cualquiera. La ecuación es:

$$s_{hk} = \alpha_i s_{hi} + \alpha_j s_{hj} + \beta s_{ij} + \gamma |s_{hi} - s_{hj}|$$

En esta ecuación, s_{ij} se refiere a la similitud entre los objetos *i* y *j* que se han fusionado para formar el nuevo clúster *k*. La nueva similitud entre el clúster *k* y cualquier objeto *h* está dada por s_{hk} y α_i, α_j, β, y γ son parámetros cuyos valores son especificados por el procedimiento de clustering jerárquico. En la Tabla 4.14, se presentan los valores de los parámetros para cada uno de los cuatro algoritmos de clustering previamente analizados. Para las fórmulas de esta tabla, n_r corresponde al número de documentos contenidos en el clúster *r*, *donde r = i, j, h.*

Cabe señalar que Wishart (1969) sugiere que el método de Ward es compatible con la fórmula de Lance y Williams, y diseñó un algoritmo eficiente para su implementación. También, sugirió que todos los otros algoritmos de clustering podrían implementarse a través del mismo algoritmo haciendo uso de la fórmula de transformación dada por Lance y Williams.

	α_i	α_j	β	γ
Single Link	*1/2*	*1/2*	*0*	*-1/2*
Complete Link	*1/2*	*1/2*	*0*	*1/2*
Average Link	$\frac{n_i}{n_i + n_j}$	$\frac{n_j}{n_i + n_j}$	*0*	*0*
Ward's	$\frac{n_h + n_i}{n_h + n_i + n_j}$	$\frac{n_h + n_j}{n_h + n_i + n_j}$	$\frac{-n_j}{n_h + n_i + n_j}$	*0*

Tabla 4.14. Valores para los parámetros de la ecuación combinatoria de Lance y Williams.

Debido a que para poder llevar a cabo una correcta comparación entre los cuatros algoritmo de clustering implementados se hace necesaria la transformación explicada anteriormente, no fue posible realizar los experimentos para el algoritmo Ward's, ya que nos hubiera tomado mucho más tiempo transformar todos los algoritmos implementados para adecuarlos a la ecuación combinatoria de Lance y Williams.

4.4 Conclusiones de los experimentos

Para la primera parte de los experimentos relacionados con la precisión de documentos en línea, se contaba con dos conjuntos de documentos, el primero de ellos es el conjunto de documentos antiguos el cuál se encuentra agrupado en un clúster y el segundo conjunto es el conjunto de documentos nuevos, que representan a documentos que se van incorporando constantemente en el tiempo. Con estos dos conjuntos de documentos se simuló un contexto dinámico, en el cuál surge la problemática de saber cuándo es conveniente rehacer el clúster. Para esto se realizaron experimentos donde se evaluó la precisión obtenida con el conjunto de documentos antiguos y la precisión obtenida al unir el conjunto de documentos antiguos con el conjunto de documentos nuevos. Los experimentos realizados se llevaron a cabo con distintas cantidades de documentos antiguos y nuevos (700, 1.400 y 3.500). En cada uno de los experimentos realizados se observó que la precisión mejoraba, es por esto que resulta conveniente en todos los casos rehacer el clúster. Sin embargo, a pesar de que la precisión mejoró en cada uno de los experimentos realizados, el mejor de los casos que se pudo observar en los experimentos fue cuando la cantidad de documentos antiguos fue de 700 y la cantidad de documentos nuevos fue de 3.500, esto se debe a que la cantidad de documentos nuevos que se van incorporando es considerablemente alta con respecto al conjunto de documentos antiguos.

Debido a que en la primera parte de los experimentos se determinó que era conveniente rehacer el clúster, surgió la interrogante de cuál algoritmo de clustering jerárquico, implementado en este proyecto (Single Link, Complete Link y Average Link), obtuvo mejores resultados al rehacer el clúster. Para esto se llevó a cabo la segunda parte de los experimentos donde se observaron los resultados obtenidos al rehacer el clúster con cada uno de los algoritmos de clustering. Para determinar cuál algoritmo entregó mejores resultados se contabilizó la cantidad de documentos relevantes visitados al recorrer los clústeres generados con cada uno de los algoritmos.

Analizando los resultados obtenidos por los tres algoritmos de clustering, se pudo observar que el algoritmo que arrojó mejores resultados fue el algoritmo Average Link. En el experimento número 4 donde la cantidad de documentos antiguos fue de 700 y la cantidad de documentos nuevos fue de 3.500, se pueden observar resultados considerablemente mejores, esto se debe a que la cantidad de documentos nuevos es mucho más alta en relación a los

documentos antiguos. Por otro lado en el experimento número 6 donde la cantidad de documentos antiguos fue de 3.500 y la cantidad de documentos nuevos fue de 700, a pesar de que los resultados son favorables, al tener una cantidad de documentos nuevos más reducida, los resultados obtenidos son más acotados.

Para los algoritmos Complete Link y Single Link los resultados obtenidos por medio de los experimentos fueron bastante similares. Para ambos casos la variación de mejora fue mínima pudiendose notar en el clúster construido con el algoritmo Complete Link, resultados levemente mejores. También es importante mencionar que se dieron casos, donde a pesar de que se agregaron documentos nuevos para reconstruir el clúster, la cantidad de documentos relevantes al recorrer el clúster disminuyó. Estos casos se dieron debido a que, al rehacer el clúster con nuevos documentos sus jerarquías internas cambian, por lo que al recorrer el clúster los documentos visitados pueden ser distintos a los visitados en el clúster inicial (clúster construido solo con el conjunto de documentos antiguos), lo que explicaría la disminución de la cantidad de documentos relevantes al recorrer el clúster reconstruido.

CAPITULO 5

Conclusiones generales

5.1 Contribuciones

Este proyecto tuvo como propósito principal investigar y evaluar la eficacia del clustering jerárquico basado en la recuperación de la información en línea. Se investigaron y describieron los 4 algoritmos de clustering utilizados en la recuperación de la información. Para analizar de forma más detallada las contribuciones de este proyecto, se utilizaron los siguientes objetivos:

- Estudiar los conceptos envueltos en recuperación de la información e identificar la problemática de la recuperación de documentos utilizando algoritmos de clustering en un contexto dinámico.
- El análisis de los principales algoritmos existentes de almacenamiento de clustering, con el fin de utilizar estos algoritmos a través de un algoritmo en línea.
- Implementar cuatro algoritmos de clustering, los cuales serán utilizados con el fin de comparar y obtener resultados.
- Analizar detalladamente los resultados empíricos.
- Usar un algoritmo probabilístico con el fin de simular los juicios de usuarios para el conjunto de documentos nuevos (que se van incorporando constantemente en el tiempo).

Para cada uno de los objetivos señalados anteriormente, se explicaran las actividades que se realizaron para cumplir con cada uno de ellos.

1) Estudiar los conceptos envueltos en recuperación de la información e identificar la problemática de recuperación de documentos utilizando algoritmos de clustering en un contexto dinámico.

En el capítulo 2, se estudiaron las diferentes definiciones existentes en la literatura respecto a la recuperación de la información con el fin de estudiar y comprender las características de un

SRI. Se definió cuál es el objetivo principal de un SRI, así como también, se explicó detalladamente el proceso que se lleva a cabo y los aspectos que se consideran en un SRI.

En las diferentes secciones del capítulo 2 se detalla la representación interna de las consultas y documentos que se lleva a cabo para realizar el proceso. También, en la sección 2.3, se describe el proceso de matching entre las consultas y documentos. Por otro lado, se habla de los métodos de comparación que existen donde se explica los dos principales métodos del espacio vectorial y medida del coseno.
Finalmente en la última sección del capítulo se habló de las diferentes áreas de investigación, en las cuales se puede enfocar la evaluación de un SRI. También, se detalla las medidas de efectividad más utilizadas las cuales corresponden a la recuperación y la precisión.

2) El análisis de los principales algoritmos existentes de almacenamiento de clustering, con el fin de utilizar estos algoritmos a través de un algoritmo en línea.

En el capítulo 3 hablamos de los algoritmos de clustering jerárquicos, donde para comprenderlos de mejor forma damos algunos ejemplos de estos algoritmos. Para esto, detallamos el procedimiento que se sigue paso a paso para realizar un algoritmo de clustering jerárquico. En las siguientes secciones del capítulo 3 se describieron detalladamente los cuatro principales algoritmos de clustering, los cuales fueron investigados más ampliamente en RI. Los cuatro algoritmos descritos corresponden a Single Link, Complete Link, Average Link y Ward's. Estos algoritmos de clustering se utilizaron después de demostrar que era conveniente rehacer el clúster, esto se demostró empíricamente mediante el aumento de la precisión obtenida con la unión de documentos antiguos y nuevos.

3) Implementar cuatro algoritmos de clustering, los cuales serán utilizados con el fin de comparar y obtener resultados.

Para la realización de esta tesis, era fundamental contar con algoritmos de clustering, que permitieran agrupar de la mejor forma posible un conjunto de documentos, para así, poder analizar con cuál de ellos se obtenían mejores resultados. Para esto, se implementaron cuatro algoritmos de clustering (Single Link, Complete Link, Average Link y Ward's), que fueron programados en lenguaje de programación C y que se encuentran en el anexo X.

4) Analizar detalladamente los resultados empíricos.

En el capítulo 4 se realizaron diversos experimentos que se separaron en dos partes, estas se encuentran en el apartado 4.4.1 y 4.4.2. Para la primera parte que corresponde a los experimentos de precisión fue necesario analizar detalladamente un framework programado en lenguaje C, que contaba con aproximadamente 4,000 líneas de código y se encargaba de simular el escenario experimental permitiendo generar conjuntos de términos, conjuntos de documentos y conjuntos de consultas, además, contaba con la posibilidad de utilizar algunas distribuciones de probabilidad que fueron utilizadas para simular juicios de usuarios para los documentos antiguos. Para la segunda parte que corresponde a los experimentos de clustering, además del framework mencionado anteriormente, se utilizaron los algoritmos de clustering implementados en lenguaje de programacion C mencionados en el objetivo 3). Una vez realizados los experimentos se analizan los resultados empíricos en la sección 4.5 donde se puede observar como mejora la precisión con la aparición de nuevos documentos, así como también, cuál de los algoritmos de clustering obtuvo los mejores resultados en los experimentos.

5) Usar un algoritmo probabilístico con el fin de simular los juicios de usuarios para el conjunto de documentos nuevos (que se van incorporando constantemente en el tiempo)

En el capítulo 4 en la sección 4.2.2 se explicó porque fue necesario implementar un nuevo algoritmo probabilístico para simular los juicios de usuarios para el conjunto de documentos nuevos. Luego, en la sección 4.2.2.1 se agregó el algoritmo probabilístico implementado en lenguaje C.

5.2 Trabajos Futuros

En esta sección se mostrarán ideas para posibles trabajos futuros. Estas ideas describen aspectos del trabajo de este proyecto que podrían ser merecedores de más investigación, o que derivan como consecuencia de los resultados de esta tesis.

1) Expansión de la evaluación de escenarios experimentales para obtener el momento exacto en el cual mejora la precisión.

Con todos los resultados empíricos obtenidos se logró implementar un algoritmo en línea con el cual se pudo observar cuando era conveniente rehacer el clúster. La decisión de rehacer el clúster es desencadenada en el momento exacto en que aumenta la precisión. Es por esto que se propone como trabajo futuro realizar experimentos donde se consideren intervalos más pequeños de documentos, donde sea posible saber la cantidad exacta de documentos nuevos necesarios en la que la precisión comienza a aumentar.

2) Evaluar con escenarios experimentales el algoritmo de clustering Ward's.

Como se vio en el capítulo 4 en la sección 4.3.2.4 no fue posible evaluar con experimentos el algoritmo ward's, debido que, este algoritmo utiliza la distancia euclidiana para generar los clústeres, a diferencia de los otros algoritmos de clustering, que utilizan la medida del coseno para realizar la agrupación. Debido a esto, no fue posible realizar una comparación donde se considera el algoritmo de clustering Ward's. Por esta razón, sería conveniente poder realizar a futuro una transformación que normalice los documentos, como la que propone Wishart (1969), y así poder crear un escenario experimental donde sea posible evaluar el algoritmo Ward's. A pesar de que no fue posible realizar experimentos con el algoritmo de clustering Ward´s, en este proyecto se logró implementar dicho algoritmo en lenguaje C, el cual se encuentra en el anexo X

Bibliografía

- Amba, S., Narasimhamurthi, N., O'Kane, K.C., Turner, P.M. (1996). Automatic linking of thesauri. In Proceedings of the 19th Annual ACM SIGIR Conference, pp. 181-186. Zurich, Switzerland.
- Anderberg, M.R. (1973). Cluster Analysis for Applications. New York: Academic Press. Anick, P.G. and Vaithyanathan, S. (1997). Exploiting clustering and phrases for context-based information retrieval. In Proceedings of the 20th Annual ACM SIGIR Conference, pp. 314-323. Philadelphia, PA.
- Baeza-Yates, R. A. and Ribeiro-Neto, B. (1999). Modern Information Retrieval. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Barry, C.L. (1994). User defined relevance criteria: An exploratory study. Journal of the American Society for Information Science, 45(3).
- Bharat, K. and Henzinger, M. R. (1998). Improved algorithms for topic distillation in a hyperlinked environment. In In SIGIR Conference on Research and Development in Information Retrieval.
- Borlund, P. and Ingwersen, P. (1997). The development of a method for the evaluation of interactive information retrieval systems. Journal of Documentation, 53.
- Broder, A. (2002). A taxonomy of web search. SIGIR Forum, 36(2).
- Chowdhury, G. (2010). Introduction to Modern Information Retrieval, Third Edition. Facet Publishing, 3rd edition.
- Cormack, R.M. (1971). A review of classification. Journal of the Royal Statistical Society, Series A, 134.
- Croft, W.B. (1977). Clustering large files of documents using the single-link method. Journal of the American Society for Information Science, 28.
- Croft, W.B. (1978). Organizing and searching large files of document descriptions. Ph.D. Thesis, Churchill College, University of Cambridge.
- Croft, W.B. and Harper, D.J. (1979). Using probabilistic models of document retrieval without relevance information. Journal of Documentation, 35.
- Crouch, C.J. and Yang, B. (1992). Experiments in automatic statistical thesaurus construction. In Proceedings of the 15th Annual ACM SIGIR Conference, Copenhagen, Denmark.

- Diday, E. and Simon, J.C. (1976). Clustering Analysis. In Fu, K.S., Keidel, W.D., Wolter, H. (eds.) Digital Pattern Recognition. Berlin: Springer-Verlag.
- Dubin, D.S. (1996). Structure in document browsing spaces. Ph.D. Thesis, School of Information Sciences, University of Pittsburgh.
- Ellis, D., Furner-Hines, J., Willett, P. (1993). Measuring the degree of similarity between objects in text retrieval systems. Perspectives in Information Management, 3(2).
- Good, I.J. (1958). Speculations concerning information retrieval. Research report PC-78, IBM Research Centre, Yorktown Heights, New York.
- Gordon, A.D. (1987). A review of hierarchical classification. Journal of the Royal Statistical Society, Series A, 150(2).
- Gray, P.,Watson, H.J.: Present and future directions in data warehousing. SIGMIS Database 29(3) (June 1998).
- Griffiths, A., Robinson, L.A., Willett, P. (1984). Hierarchic agglomerative clustering methods for automatic document classification. Journal of Documentation, 40(3).
- Gutiérrez-Soto, C. (2016). Exploring the Reuse of Past Search Results in Information Retrieval.
- Harman, D. (1993). Overview of the first trec conference. In Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '93, pages 36–47, New York, NY, USA. ACM.
- Hartigan, J.A. (1975). Clustering algorithms. New York: Wiley.
- Hartuv, E., Schmitt, A., Lange, J., Meier-Ewert, S., Lehrach, H., Shamir, R. (1999). An algorithm for clustering cDNAs for gene expression analysis. In Proceedings of the 3rd Annual International Conference on Computational Molecular Biology, pp. Lyon, France.
- Hatzivassiloglou, V., Gravano, L., Maganti, A. (2000). An investigation of linguistic features and clustering algorithms for topical document clustering. In Proceedings of the 23rd Annual ACM SIGIR Conference, pp. 224-231. Athens, Greece.
- Hubálek, Z. (1982). Coefficients of association and similarity, based on binary (presence-absence) data: an evaluation. Biological Reviews of the Cambridge Philosophical Society, 57(4).
- Jardine, N. and Sibson, R. (1968). The construction of hierarchic and non-hierarchic classifications. Computer Journal, 11(2):177-184.

- Jardine, N. and Sibson, R. (1971). Mathematical Taxonomy. New York: Wiley.
- Jardine, N. and van Rijsbergen, C.J. (1971). The use of hierarchical clustering in information retrieval. Information Storage and Retrieval, 7:217-240.
- Jones, W.P. and Furnas, G.W. (1987). Pictures of relevance: A geometric analysis of similarity measures. Journal of the American Society for Information Science, 38(6):420-442.
- Kaufman, L. and Rousseeuw, P.J. (1990). Finding groups in data: an introduction to cluster analysis. New York: Wiley.
- Keen, E. M. (1992). Presenting results of experimental retrieval comparisons. Inf. Process. Manage., 28(4) :491–502.
- Kirriemuir, J.W. and Willett, P. (1995). Identification of duplicate and near-duplicate full-text records in database search-outputs using hierarchic cluster analysis. Program, 29(3):241-256.
- Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment.J. ACM, 46(5) :604–632.
- Kural, Y. (1999). Clustering information retrieval search outputs. Ph.D. Thesis, City University, London.
- Lewis, D. D. and Jones, K. S. (1996). Natural language processing for information retrieval. Commun. ACM, 39(1):92–101.
- Maarek, Y.S., Fagin, R., Ben-Shaul, I.Z., Pelleg, D. (2000). Ephemeral document clustering for web applications. IBM Research Report RJ 10186, IBM Research, Haifa, Israel.
- Milligan, G.W., Soon, S.C., Sokol, L.M. (1983). The effect of cluster size, dimensionality, and the number of cluster on recovery of true cluster structure. IEEE Transactions on Patter Recognition and Machine Intelligence, 5(1).
- Minker, J., Wilson, G.A., Zimmerman, B.H. (1972). An evaluation of query expansion by the addition of clustered terms for a document retrieval system. Information Storage & Retrieval, 8:329-348.
- Ouksel, A. (2002). Mining the world wide web : An information search approach by george chang, marcus j. healey (editor), james a. m. mchugh, jason t. l. wang. SIGMOD Rec., 31(2) :69–70.

- Peat, H.J. and Willett, P. (1991). The limitations of term co-occurrence data for query expansion in document retrieval systems. Journal of the American Society for Information Science, 42(5):378-383.
- Preece, S.E. (1973). Clustering as an output option. Proceedings of the American Society for Information Science, 10.
- Reid, J. (2000). A task-oriented non-interactive evaluation methodology for. Information Retrieval Systems, Information Retrieval, 2 :115–129.
- Robertson, S. E., van Rijsbergen, C. J., and Porter, M. F. (1981). Probabilistic models of indexing and searching. In Proceedings of the 3rd Annual ACM Conference on Research and Development in Information Retrieval, SIGIR '80, pages 35–56, Kent, UK, UK. Butterworth & Co.
- Rocchio, J.J. (1966). Document retrieval systems - Optimization and evaluation. PhD Thesis, Report ISR-10 to the National Science Foundation, Harvard Computation Laboratory.
- Rorvig, M. (1999). Images of similarity: a visual exploration of optimal similarity metrics and scaling properties of TREC topic-document sets. Journal of the American Society for Information Science, 50(8):639-651.
- Salton, G. (1971). The SMART Retrieval System and ;Experiments in Automatic Document Processing. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- Salton, G., (ed.) (1971). The SMART Retrieval System - Experiments in Automatic Document Retrieval. New Jersey, Englewood Cliffs: Prentice Hall Inc.
- Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. Commun. ACM, 18(11) :613–620.
- Salton, G. and Wong, A. (1978). Generation and search of clustered files. ACM Transactions on Database Systems, 3(4):321-346.
- Salton, G. and McGill, M.J. (1983). Introduction to modern information retrieval. New York: McGraw-Hill.
- Salton, G. and McGill, M. J. (1986). Introduction to Modern Information Retrieval. McGraw-Hill, Inc., New York, NY, USA.
- Salton, G., Buckley, C.: Readings in information retrieval. In Sparck Jones, K., Willett, P., eds.: Readings in information retrieval. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1997) 355-364.

- Schaer, P. (2013). Applied informetrics for digital libraries : an overview of foundations, problems and current approaches. Historical Social Research, 38(3):267–281.
- Schamber, L., Eisenberg, M.B., Nilan, M.S. (1990). A re-examination of relevance: Toward a dynamic, situational definition. Information Processing & Management, 26(6):755-776.
- Sibson, R. (1973). SLINK: an optimally efficient algorithm for the single link cluster method. Computer Journal, 16:30-34.
- Small, H. (1999). Visualizing science by citation mapping. Journal of the American Society for Information Science, 50(9):799-813.
- Sneath, P.H.A. and Sokal, R.R. (1973). Numerical taxonomy: the principles and practice of numerical classification. San Francisco: W.H. Freeman.
- Sparck Jones, K. (1971). Automatic keyword classification for information retrieval. London: Butterworths.
- Sparck Jones, K. and Willett, P., editors (1997a). Readings in Information Retrieval. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Späth, H. (1980). Cluster Analysis Algorithms For Data Reduction and Classification of Objects. England: John Ellis Horwood Limited.
- Theodoridis, S. and Koutroumbas, K. (1999). Pattern Recognition. San Diego: Academic Press.
- Tombros, A., Villa, R., and Rijsbergen, C. J. V. (2002). The effectiveness of query-specific hierarchic clustering. In in information retrieval. Information Processing and Management.
- Tombros, A., Rijsbergen, C.J.V.: Query-sensitive similarity measures for informa-tion retrieval. Knowledge and Information Systems 6 2004.
- van Rijsbergen, C.J. and Sparck Jones, K. (1973). A test for the separation of relevant and nonrelevant documents in experimental retrieval collections. Journal of Documentation, 29(3).
- Van Rijsbergen, C.J. (1979). Information Retrieval. London: Butterworths, 2nd Edition.
- van Rijsbergen, C.J, Harper D.J., Porter, M.F. (1981). The selection of good search terms. Information Processing & Management, 17.
- Van Rijsbergen, C.J. (1986). A new theoretical framework for information retrieval. In Proceedings of the 9th Annual ACM SIGIR Conference, pp. 194-200. Pisa, Italy.

- Van Ryzin, J. (ed.) (1977). Classification and Clustering: Proceedings of the Advanced Seminar Conducted by the Mathematics Research Center, the University of Wisconsin at Madison. New York, London: Academic Press.
- Voorhees, E.M. (1985a). The effectiveness and efficiency of agglomerative hierarchic clustering in document retrieval. Ph.D. Thesis, Technical Report TR 85-705 of the Department of Computing Science, Cornell University.
- Voorhees, E.M. (1985b). The cluster hypothesis revisited. In Proceedings of the 8th Annual ACM SIGIR Conference, pp. 188-196. Montreal, Canada.
- Ward, J.H. (1963). Hierarchical grouping to minimize an objective function. Journal of the American Statistical Association, 58:236-244.
- Watanabe, S. (1969). Knowing and guessing: a quantitative study of inference and information. New York: Wiley.
- Willett, P. (1983). Similarity coefficients and weighting functions for automatic document classification: an empirical comparison. International Classification, 3:138-142.
- Willett, P. (1988). Recent trends in hierarchic document clustering: A critical review. Information Processing & Management, 24(5).
- Wishart, D. (1969). An algorithm for hierarchical classification. Biometrics, 25.

Anexo

Codigo fuente de algoritmo Single Link

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<malloc.h>
/*****************************************************************************
 Autores: Delia Moncada - Frederick Lara.
 La forma de compilar este programa es: gcc SingleLink.c -o singleLink -lm
 Para poder ejecutar correctamente el programa se debe considerar lo siguiente:
        -debe existir un archivo de texto de nombre "documentos.txt" sin comillas, que
         contenga los documentos con los que se construira la matriz de similaridad.
        -se debe cambiar la cantidad de la variable LARGOM,por la cantidad de documentos
         que contenga el archivo de texto de nombre "documentos.txt".
        -debe existir un archivo de texto de nombre "documentosR.txt" sin comillas, que
         contenga las id de los documentos relevantes.
        -ejecutar con el comando: ./singleLink
*****************************************************************************/
int LARGOM=700;// LARGOM es la cantidad de documentos que contiene el File archivo
int NTERMINOS;
```

```
struct StopWords{
   char stopW[100];
   struct StopWords *sgte;
};
typedef struct StopWords STOPW;
typedef STOPW *SP;
struct SQUERY2{
        int Id;
        char FQuery[1000];
        struct SQUERY2 *sgte;
};
typedef struct SQUERY2 SQ2;
typedef SQ2 *PQ2;
struct Terminos{
        char Termino[30];
        int esta;
        float log;
        int ID;
};
typedef struct Terminos TER;
typedef TER *TER2;
struct lista_Doc_Query{
```

```
        int IDQuery;
        int IDDocumento;
        struct lista_Doc_Query *sgte;
};
struct Respuestas{
        int Id;
        float sim;
        struct Respuestas *sgte;
};
struct GlobalRespuestas{
        int Id;
        struct Respuestas *dato;
        struct GlobalRespuestas *sgte;
};
struct Respuestas_{
        int Id;
        float sim;
};
struct Documentos{
        int Id;
        char Documento[1500];
        struct Documentos *sgte;
```

```
};
typedef struct Documentos Docu;
typedef Docu *DOC;
struct GlobalRespuestas *InsertarGR(struct GlobalRespuestas *headGR, struct Respuestas
*dato)
{
        if(headGR==NULL)
        {
                headGR=(struct GlobalRespuestas *)malloc(sizeof(struct GlobalRespuestas));
                headGR->dato=dato;
                headGR->sgte=NULL;
        }
        else headGR->sgte=InsertarGR(headGR->sgte,dato);
        return(headGR);
}
DOC InsertarD(DOC head, char Documento[], int Id)
{
        if(head==NULL)
        {
                head=(DOC)malloc(sizeof(Docu));
                head->Id=Id;
                strcpy(head->Documento,Documento);
                head->sgte=NULL;
```

```
        }
        else head->sgte=InsertarD(head->sgte,Documento,Id);
        return(head);
}
struct Respuestas *Insertar(struct Respuestas *head, int Id, float sim)
{
        if(head==NULL)
        {
                head=(struct Respuestas *)malloc(sizeof(struct Respuestas));
                head->Id=Id;
                head->sim=sim;
                head->sgte=NULL;
        }
        else head->sgte=Insertar(head->sgte,Id,sim);
        return(head);
}
int ContarPalabras(char cadena[])
{
        int i,cont,pal;
        for(i=0,cont=0,pal=0;cadena[i]!='\0';i++)
                if(cadena[i]!=' ')
                        pal++;
```

```
                else if(pal!=0)
                {
                        cont++;
                        pal=0;
                }
                else continue;
                if(pal!=0)
                        return(cont+1);
                else return(cont);
}
void Copiar_iesimo(int inicio,int largo,char original[],char destino[])
{
        int i,j;
        for(i=1,j=0;i<=largo;inicio++,i++,j++)
                destino[j]=original[inicio];
        destino[j]='\0';
}
void SacarCaracteresNoValidos(char cadena[])
{
        int i;
        for(i=0; cadena[i]!='\0';i++)
```

```
if(cadena[i]=='(' || cadena[i]==')' || cadena[i]=='?' || cadena[i]=='-' || cadena[i]==';' ||
cadena[i]==',' || cadena[i]=='"' || cadena[i]==':' || cadena[i]=='*' || cadena[i]=='+' ||
cadena[i]=='/' || cadena[i]==9)

  cadena[i]=' ';

  else if(cadena[i]=='.')

cadena[i]=' ';

  else continue;

    for(i=0; cadena[i]!='\0';i++)

      if(cadena[i]=='A')

          cadena[i]='a';

          else if(cadena[i]=='B')

          cadena[i]='b';

        else if(cadena[i]=='C')

                cadena[i]='c';

         else if(cadena[i]=='D')

          cadena[i]='d';

            else if(cadena[i]=='E')

                cadena[i]='e';

        else if(cadena[i]=='F')

                cadena[i]='f';

                else if(cadena[i]=='G')

                cadena[i]='g';

              else if(cadena[i]=='H')
```

```
            cadena[i]='h';
        else if(cadena[i]=='I')
                cadena[i]='i';
            else if(cadena[i]=='J')
                cadena[i]='j';
              else continue;
for(i=0; cadena[i]!='\0';i++)
  if(cadena[i]=='K')
    cadena[i]='k';
  else if(cadena[i]=='L')
      cadena[i]='l';
    else if(cadena[i]=='M')
        cadena[i]='m';
      else if(cadena[i]=='N')
cadena[i]='n';
   else if(cadena[i]=='O')
        cadena[i]='o';
else if(cadena[i]=='P')
        cadena[i]='p';
  else if(cadena[i]=='Q')
        cadena[i]='q';
        else if(cadena[i]=='R')
```

```
                cadena[i]='r';
                else if(cadena[i]=='T')
                        cadena[i]='t';
                else if(cadena[i]=='S')
                        cadena[i]='s';
                        else if(cadena[i]=='U')
                        cadena[i]='u';
                        else if(cadena[i]=='V')
                                cadena[i]='v';
                        else if(cadena[i]=='W')
                                cadena[i]='w';
                                else if(cadena[i]=='X')
                            cadena[i]='x';
                                else if(cadena[i]=='Y')
                               cadena[i]='y';
                            else if(cadena[i]=='Z')
                                        cadena[i]='z';
                               else continue;
}
int CaracterValido(char cadena[],int i)
{
        while(cadena[i]!='\0')
```

```
        {
                if(cadena[i]!=' ')
                        return(i);
                else i=i+1;
        }
   return(-1); // son puros espacios en blancos
}
void SacarEspaciosS(char cadena[])
{
        int i,j,k;
        i=0;
        while(cadena[i]!='\0')
        {
                if(cadena[i]==' ' || cadena[i]==9)
                {
                        if(cadena[i]==' ' && cadena[i+1]==' ' && cadena[i+1]!='\0')
                        {
        k=CaracterValido(cadena,i); // si es -1 entonces son puros espacios los faltantes
        if(k!=-1)
        {
                for(j=i+1;cadena[k]!='\0';j++,k++)
                {
```

```
                        cadena[j]=cadena[k];
                }
                cadena[j]='\0';
                i=i+1;
          }
          else cadena[i]='\0';
        }
        else  i=i+1;
      }
      else  i=i+1;
   }
}
PQ2 Insertar_Q(PQ2 head,char stopW[], int Id)
{
          if(head==NULL)
          {
                head=(PQ2)malloc(sizeof(SQ2));
                head->sgte=NULL;
                head->Id=Id;
                strcpy(head->FQuery,stopW);
          }
          else head->sgte= Insertar_Q(head->sgte,stopW,Id);
```

```
        return(head);
}
SP Insertar_SW(SP headSW,char stopW[])
{
        if(headSW==NULL)
        {
                headSW=(SP)malloc(sizeof(STOPW));
                headSW->sgte=NULL;
                strcpy(headSW->stopW,stopW);
        }
        else headSW->sgte= Insertar_SW(headSW->sgte,stopW);
        return(headSW);
}
int esSubcadena(char Query[],int i,int largo, SP headSW)
{
        int LARGO;
        LARGO=strlen(Query);
        if(i!=0 && Query[i-1]!=' ')
                return(1);
        else if(Query[i+largo]!=' ' && i+largo<LARGO && Query[i+largo]!='\0')
                return(1);
        else return(0);
```

```
}
SP Sacar(char Query[],SP headSW)
{
        char Aux[50];
        int largo,i,j,k;
        if(headSW!=NULL)
        {
                largo=strlen(headSW->stopW);
                for(i=0; Query[i]!='\0';i++)
                        if(Query[i]==headSW->stopW[0] &&
!esSubcadena(Query,i,largo,headSW))
                        {
                                Copiar_iesimo(i,largo,Query,Aux);
                                if(!strcmp(headSW->stopW,Aux))
                                {
                                        if(Query[i+largo]!='\0')
                                        {
                                                if(Query[i+largo]=='0')
                                                {
                                                        k=i+largo+1;
                                                        j=i;
                                                        while(Query[k]!='\0')
                                                        {
```

```
                                        Query[j]=Query[k];
                                        j++;
                                        k++;
                                }
                                Query[j]='\0';
                        }
                        else{
                                k=i+largo;
                                j=i;
                                while(Query[k]!='\0')
                                {
                                        Query[j]=Query[k];
                                        j++;
                                        k++;
                                }
                                Query[j]='\0';
                        }
                }
                else Query[i]='\0';
        }
        else continue;
}
```

```
                else continue;
                headSW->sgte=Sacar(Query,headSW->sgte);
        }
        else return(NULL);
        return(headSW);
}
DOC SacarStopWords_D(DOC head, SP headSW)
{
        char Documento[1500];
        int i,j;
        if(head!=NULL)
        {
                strcpy(Documento, head->Documento);
                headSW=Sacar(Documento, headSW);
                SacarEspaciosS(Documento);
                if(Documento[0]==' ')
                {
                        for(i=0, j=i+1;Documento[j]!='\0';i++,j++)
                                Documento[i]=Documento[j];
                        Documento[i]='\0';
                }
                strcpy( head->Documento, Documento);
```

```
                head->sgte=SacarStopWords_D(head->sgte, headSW);
        }
        else return(NULL);
        return(head);
}
PQ2 SacarStopWords(PQ2 head, SP headSW)
{
        char Query[1000];
        int i,j;
        if(head!=NULL)
        {
                strcpy(Query, head->FQuery);
                headSW=Sacar(Query, headSW);
                SacarEspaciosS(Query);
                if(Query[0]==' ')
                {
                        for(i=0, j=i+1;Query[j]!='\0';i++,j++)
                                Query[i]=Query[j];
                        Query[i]='\0';
                }
                strcpy( head->FQuery, Query);
                head->sgte=SacarStopWords(head->sgte, headSW);
```

```
        }
        else return(NULL);
        return(head);
}
void  LLenarQuery(char cadena[],char Query[])
{
        int i,stop=0,j;
        for(i=0; Query[i]!='\0' && stop==0;i++)
                if(Query[i]=='0')
                {
                        j=i;
                        stop=1;
                }
                else continue;
                Query[j]=' ';
                j=i;
                for(i=0; cadena[i]!='\0' && Query[j]!='\0'; i++, j++)
                        Query[j]=cadena[i];
}
int CompararCadenas(char cad1[],char cad2[])
{
        int i,largo1,largo2,stop;
```

```
        largo1=strlen(cad1);
        largo2=strlen(cad2);
        if(largo1==largo2)
                for(i=0, stop=1; cad1[i]!='\0' && stop==1;i++)
                        if(cad1[i]!=cad2[i])
                                stop=0;
                        else continue;
        else return(0);
        return(stop);
}
int Buscar(char cadena[],char frase[])
{
        char Aux[30];
        int i,largo,largo2,total;
        largo=strlen(cadena);
        largo2=strlen(frase);
        for(i=0;frase[i]!='\0' && ((i+largo)<=largo2);i++)//aqui estaba el error, que no
consideraba el ultimo termino del documento
        if(frase[i]==cadena[0])
        {
                Copiar_iesimo(i,largo,frase,Aux);
                total=i+largo;
                if(CompararCadenas(Aux,cadena) == 1)
```

```
                {
                        return(1);
                }
                else continue;
        }
        else continue;
        return(0);
}
int EsCadenaVacia(char cadena[], int largo)
{
        int i;
        for(i=0; i<largo;i++)
                if(cadena[i]!=' ')
                        return(1);
                else continue;
        return(0);
}
int ContarSW(SP headSW, int i)
{
        if(headSW!=NULL)
                return(ContarSW(headSW->sgte,i+1));
        else return(i);
```

```
}
int NQueries(PQ2 headQ, int cont)
{
	if(headQ!=NULL)
		return(NQueries(headQ->sgte, cont+1));
	else return(cont);
}
void Copiar(struct Respuestas *head,struct Respuestas_ *Arr,int inic,int largo)
{
	if(head!=NULL && inic<largo)
	{
		Arr[inic].Id=head->Id;
		Arr[inic].sim=head->sim;
		Copiar(head->sgte,Arr,inic+1,largo);
	}
}
int LargoRespuestas(struct Respuestas *head, int largo)
{
	if(head!=NULL)
		return(LargoRespuestas(head->sgte, largo+1));
	else return(largo);
}
```

```
struct Respuestas *Ordenar(struct Respuestas *head)
{
	int largo,i,j,k;
	struct Respuestas_ *Arr,Aux;
	struct Respuestas *head2=NULL;
	largo=LargoRespuestas(head,0);
	Arr=(struct Respuestas_ *)malloc(largo*sizeof(struct Respuestas_ ));
	Copiar(head,Arr,0,largo);
	for(j=0;j<largo;j++)
	{
		Aux=Arr[j];
		for(i=j+1,k=j;i<largo;i++)
		{
			if(Arr[i].sim > Aux.sim)
			{
				Aux=Arr[i];
				k=i;
			}
			else continue;
		}
		Arr[k]=Arr[j];
		Arr[j]=Aux;
```

```
        }
        for(i=0;i<largo;i++)
                head2=Insertar(head2,Arr[i].Id,Arr[i].sim);
        return(head2);
}
int ContarQDValidos(struct Terminos **MatrizT, int fil, int col)
{
        int i,cont;
        for(i=1,cont=0;i<fil;i++)
                if(MatrizT[i][0].esta==1)
                {
                        cont++;
                }
                else continue;
        return(cont);
}
float Similaridad(struct Terminos **MatrizT, int NT, int largo,int ID1,int ID2)
{
        float s,s2,s3,sim;
        int j;
        for(j=1,s=0.0,s2=0.0,s3=0.0;j<NT;j++)
        {
```

```
                s=s+MatrizT[ID1][j].log*MatrizT[ID2][j].log;
                s2=s2+MatrizT[ID1][j].log*MatrizT[ID1][j].log;
                s3=s3+MatrizT[ID2][j].log*MatrizT[ID2][j].log;
        }
        if(s2!=0.0 && s3!=0.0)
                sim=s/sqrt(s2*s3);
        else sim=0.0;
        return(sim);
}
int ContarTerminosCol(struct Terminos **MatrizT,int col, int NT, int largo)
{
        int i,cont;
        for(cont=0,i=1;i<=largo;i++)
        {
                if(MatrizT[i][col].esta>=1)
                        cont++;
                else continue;
        }
        return(cont);
}
void LLenarLDF(struct Terminos **MatrizT,int col,int NT, int largo,float ldf)
{
```

```
        int i;
        for(i=1;i<largo;i++)
        {
                if(MatrizT[i][col].esta>=1)
                {
                        MatrizT[i][col].log=ldf*(float) MatrizT[i][col].esta;
                }
                else {
                        MatrizT[i][col].log=0;
                }
        }
}
PQ2 Esta4(struct Terminos **MatrizT,int Nterm, PQ2 headQ, int actual)
{
        int j,i,cont,largo;
        char Aux[100];
        if(headQ!=NULL)
        {
                for(j=1;j< Nterm;j++)
                {
                        if(Buscar(MatrizT[0][j].Termino,headQ->FQuery)==1)
                        {
```

```
                        largo=strlen(MatrizT[0][j].Termino);
                        for(i=0,cont=0;headQ->FQuery[i]!='\0';i++)
                                if(MatrizT[0][j].Termino[0]==headQ->FQuery[i])
                                {
                                        Copiar_iesimo(i,largo,headQ->FQuery,Aux);

        if(CompararCadenas(Aux,MatrizT[0][j].Termino) == 1)
                                                cont++;
                                        else continue;
                                }
                                else continue;
                        MatrizT[actual][j].esta=cont;
                        MatrizT[actual][0].esta=1;
                }
                else continue;
        }
        headQ->sgte=Esta4(MatrizT,Nterm,headQ->sgte,actual+1);
    }
    return(headQ);
}
int NDocumentos(DOC headD, int cont)
{
    if(headD!=NULL)
```

```
                return(NDocumentos(headD->sgte, cont+1));
        else return(cont);
}
int ObtenerIDDocumentos(DOC headD,int cont, int fin,int largo)
{
        if(cont==fin && fin<=largo && headD!=NULL)
                return(headD->Id);
        else return(ObtenerIDDocumentos(headD->sgte,cont+1, fin ,largo));
}
int ObtenerIDQueries(PQ2 headQ,int cont, int fin,int largo)
{
        if(cont==fin && fin<=largo && headQ!=NULL)
                return(headQ->Id);
        else return(ObtenerIDQueries(headQ->sgte,cont+1, fin ,largo));
}
int EsCadenaVacia_D(char cadena[], int largo)
{
        int i,stop=1;
        if(largo==1 && cadena[0]==10)
                return(1);
        else for(i=0;i<largo && stop==1; i++)
        if(cadena[i]!=10)
```

```
                stop=0;
        else continue;
        return(stop);
}
void  LLenarDocumento(char cadena[],char Documento[])
{
        int i,stop=0,j;
        for(i=0; Documento[i]!='\0' && stop==0;i++)
                if(Documento[i]=='\0')
                {
                        j=i;
                        stop=1;
                }
                else continue;
        if(j!=0)
        {
                Documento[j]=' ';
                j=i;
        }
        for(i=0,stop=0; cadena[i]!='\0'  && Documento[j]!='\0' && stop==0; i++, j++)
        {
                if(cadena[i]!=10)
```

```
                {
                        Documento[j]=cadena[i];
                }
                else stop=1;
        }
        j=j-1;
        Documento[j]=' ';
}
void ObtenerQuery(int inic,int actual,int largo, char QueryAux[],PQ2 head)
{
        if(head!=NULL && inic<=largo)
        {
                if(inic==actual)
                {
                        strcpy(QueryAux, head->FQuery);
                }
                else ObtenerQuery(inic+1,actual,largo,QueryAux,head->sgte);
        }
}
void ObtenerT(int fin,int inic,SP head,char Termino[])
{
        if(head!=NULL && inic== fin)
```

```
                strcpy(Termino,head->stopW);
        else  ObtenerT(fin,inic+1, head->sgte, Termino);
}
void ColocarTerminos3(int col,struct Terminos **M,int Nterminos,SP head)
{
        char Termino[30];
        ObtenerT(col,1,head,Termino);
        strcpy(M[0][col].Termino,Termino);
}
int ContarTerminos(SP head,int cont)
{
        if(head!=NULL)
                return(ContarTerminos(head->sgte,cont+1));
        else return(cont);
}
int EstaT(SP head,char Term[])
{
        if(head!=NULL)
        {
                if(CompararCadenas(head->stopW,Term)==1)
                {
                        return(1);
```

```
                }
                else return(EstaT(head->sgte,Term));
        }
        else return(0);
}
SP ColocarTerminosD(SP head,DOC headD)
{
        int i,j,largo,k;
        char Term[100],Documento[1500];
        if(headD!=NULL)
        {
                strcpy(Documento,headD->Documento);
                largo=strlen(Documento);
                for(k=0;k<=largo+3; k++)
                        i=0;
                while( Documento[i]!='\0' && i<largo)
                {
                        j=0;
                        while(Documento[i]!='\0' &&  Documento[i]!=' ' && i<largo)
                        {
                                Term[j]=Documento[i];
                                j++;
```

```
                        i++;
                }
                Term[j]='\0';
                if(EstaT(head,Term)==0)
                {
                        head=Insertar_SW(head,Term);
                }
                i=i+1;
        }
        head= ColocarTerminosD(head,headD->sgte);
    }
    return(head);
}
SP ColocarTerminos(SP head,PQ2 headQ)
{
    int i,j,largo,k;
    char Term[100],FQueryA[1000];
    if(headQ!=NULL)
    {
        strcpy(FQueryA,headQ->FQuery);
        largo=strlen(FQueryA);
        for(k=0;k<=largo+3; k++)
```

```
                i=0;
            while(FQueryA[i]!='\0' && i<largo)
            {
                j=0;
                while(FQueryA[i]!='\0' &&  FQueryA[i]!=' ' && i<largo)
                {
                    Term[j]=FQueryA[i];
                    j++;
                    i++;
                }
                Term[j]='\0';
                if(EstaT(head,Term)==0)
                {
                    head=Insertar_SW(head,Term);
                }
                i=i+1;
            }
            head= ColocarTerminos(head,headQ->sgte);
        }
        return(head);
}
struct Terminos **Obtener3( PQ2 headQ)
```

```
{
        struct Terminos **MatrizT;
        int largo,Nterminos,i,j,LAux,ID;
        float ldf,sim;
        struct Respuestas **Tabla;
        SP T=NULL;
        T=ColocarTerminos(T,headQ);
        Nterminos=ContarTerminos(T,0);
        largo=LARGOM;
        largo=largo+1;
        Nterminos=Nterminos+1;
        MatrizT=(struct Terminos **)malloc(largo*sizeof(struct Terminos *));
        for(i=0;i<largo;i++)
                MatrizT[i]=(struct Terminos *)malloc(Nterminos*sizeof(struct Terminos));
        for(i=0;i<largo;i++)
          for(j=0;j<Nterminos;j++) // el 0 es por si es que no tiene ningun termino en comun
          {
                MatrizT[i][j].esta=0;
                MatrizT[i][0].ID=i+1;
                MatrizT[i][0].log=0.0;
          }
        for(j=1; j<Nterminos ;j++)// esto coloca todos los terminos de todas las queries en la
lista T
```

```
        ColocarTerminos3(j,MatrizT,Nterminos,T);
headQ=Esta4(MatrizT,Nterminos,headQ,1);
NTERMINOS=Nterminos;
for(j=1;j<Nterminos;j++)
{
        LAux=ContarTerminosCol(MatrizT,j,Nterminos,largo-1);
        if(LAux!=0)
        {
                ldf=log10((largo-1)/(float)LAux);
        }
        else ldf=0.0;
        LLenarLDF(MatrizT,j,Nterminos,largo,ldf);
}
ID=1;
Tabla=(struct Respuestas **)malloc(LARGOM*sizeof(struct Respuestas));
for(i=0;i<LARGOM;i++)
        Tabla[i]=NULL;
for(ID=0;ID<LARGOM;ID++)
{
        for(i=1; i<largo;i++)
        {
                if(i!=(ID+1))
```

```
                        {
                                sim=Similaridad(MatrizT,Nterminos,largo,i,(ID+1));
                        }
                        else continue;
                }
                printf("dentrofor:%d\n",ID );
        }
        return MatrizT;
}
/****************************************************** Codigo SingleLink
*********************************************************/
int N;
struct Elem{
        int indice;
        int idDoc;
        int Topic;
  float TermDoc[2]; // char TermDoc[401]
  int MarcadoFil;
  int MarcadoCol;
        // Matriz
  float valorMatriz; //calculo de distancia
  int Marcado;
  int t;
```

```
};
struct Elem *Arr; // son los documentos, de aqu¡ formo la matriz
struct Elem **Matriz; // esta es la matriz de distancias
struct Lista{ // esta lista contiene los de un mismo cluster elementos
        int idDoc[20];
        int t;
        float min;
        int n;
        int i;
        int j;
        struct Lista *sgte;
};
struct Cluster{
   struct Lista *L; //esta es la lista de los cluster
   struct Cluster *sgte;
   int n;
};
struct Lista *InsertarL(struct Lista *H, int idDoc[], int N, int n, float min,int t)
{
        int i;
        if(H==NULL)
        {
```

```
                H=(struct Lista *)malloc(sizeof(struct Lista));
                H->sgte=NULL;
                H->min=min;
                H->t=t;
                H->n=n;
                for(i=0; i<N;i++)
                        if(i<n)
                                H->idDoc[i]=idDoc[i];
                        else H->idDoc[i]=-1;
        }
        else H->sgte=InsertarL(H->sgte,idDoc,N,n,min,t);
        return(H);
}
int LargoL(struct Lista *H,int i)
{
        if(H!=NULL)
                return(LargoL(H->sgte,i+1));
        else return(i);
}
int NC=0;
struct Cluster *InsertarC(struct Cluster *C, struct Lista *H, int n)
{
```

```
        NC++;
        if(C==NULL)
        {
                C=(struct Cluster *)malloc(sizeof(struct Cluster));
                C->L=H;
                C->n=n;
                C->sgte=NULL;
        }
        else C->sgte=InsertarC(C->sgte,H,n);
        return(C);
}
int NumeroCluster(struct Cluster *C)
{
        if(C!=NULL && C->sgte==NULL)
                return(C->n);
        else return(NumeroCluster(C->sgte));
}
struct Lista *CrearLista2(struct Elem **M,struct Cluster *C,struct Lista *H,int N,int t, float min)
{
        for (int i = 0; i < LARGOM; ++i)
        {
                for (int j = 0; j < LARGOM && i>j ; ++j)
```

```
{
        if (M[i][j].Marcado == 1 && H == NULL)
        {
                H=(struct Lista *)malloc(sizeof(struct Lista));
                H->t=t;
                H->min=M[i][j].valorMatriz;
                H->sgte=NULL;
                H->i=i;
                H->j=j;
                M[i][j].Marcado = 2;
        }
        else
        {
                if (M[i][j].Marcado == 1 && H != NULL)
                {
                        C=InsertarC(C,H,1);
                        H=NULL;
                        H=(struct Lista *)malloc(sizeof(struct Lista));
                        H->t=t;
                        H->min=M[i][j].valorMatriz;
                        H->i=i;
                        H->j=j;
```

```
                                        H->sgte=NULL;
                                        M[i][j].Marcado = 2;
                                    }
                                }
                            }
                        }
                        return(H);
}
int OtroMin(struct Elem **M,int N,struct Elem *A,float min)
{
        int i,j;
        for(i=0;i<N;i++)
    for(j=0;j<N && i>j ;j++)// asumo que la distancia minima es 0
    {
        if(A[i].MarcadoFil==0 && A[j].MarcadoCol==0)
                if(M[i][j].valorMatriz==min && M[i][j].Marcado==0 && !(A[i].MarcadoFil==1
&& A[i].MarcadoCol==1) && !(A[j].MarcadoFil==1 && A[j].MarcadoCol==1))
                {
                        return(1);
                }
                else continue;
        else continue;
    }
```

```
   return(0);
}
void MarcarArr(struct Elem **M,struct Elem *A,int N,float min,int t)
{
        int i,j;
        for(i=0;i<N;i++)
    for(j=0;j<N && i>j ;j++)// asumo que la distancia minima es 0
    {
        if(A[i].MarcadoFil==0 && A[j].MarcadoCol==0)
                if(M[i][j].valorMatriz==min && M[i][j].Marcado==0)
                {
                        A[i].MarcadoFil=1;
                        A[j].MarcadoCol=1;
                        M[i][j].Marcado=1;
                        M[i][j].t=t;
                        A[i].t=t;
                        A[j].t=t;
                }
                else continue;
        else continue;
                }
}
```

```
int contarDocRelEnCluster(struct Cluster *CHead, int id){
        int cont = 0;
        struct Lista *l = NULL;
        while(CHead != NULL){
                l = CHead->L;
                while(l != NULL){
                if ((l->i + 1) == id || (l->j+1) == id)// el +1 es porque para los cluster la matriz
la comence desde la posicion 0 y no del 1
                {
                        cont++;
                }
                l=l->sgte;
                }
        CHead=CHead->sgte;
        }
return(cont);
}
/****************************************************** Codigo SingleLink Fin
******************************************************/
int main(int argc, char *argv[])
{
        FILE *archivo,*archivo2;
        char cadena[1000],Query[1000],Query2[1000],IDQuery[6],Band[3],Aux[5],Band2[3];
```

```
char stopW[25];
int i, parar,largo,k,stop,j,cont=1;
SP headSW=NULL;
PQ2 headQ=NULL;
// estas definiciones son de los documentos
char Documento[1500],IDDocument[8],Band3[3],Band4[3],ID[8];
struct Terminos **MatrizF=NULL;
for(i=0;i<999;i++)
        Query[i]='0';
Query[i]='\0';
for(i=0;i<1499;i++)
        Documento[i]='0';
Documento[i]='\0';
Band[0]='.',Band[1]='I', Band[2]='\0';
Band2[0]='.',Band2[1]='T', Band[2]='\0';
Band3[0]='.', Band3[1]='W',Band3[2]='\0';
Band4[0]='.', Band4[1]='B',Band4[2]='\0';
archivo=fopen("documentos.txt","r");//nombre del archivo de texto donde estan los
documentos
if(archivo!=NULL)
{
        while (!feof(archivo))
        {
```

```
parar=0;
fgets(IDQuery,6,archivo);
Copiar_iesimo(0,2,IDQuery,Aux);
if(!strcmp(Band,Aux))
{
        largo=strlen(IDDocument);
        Copiar_iesimo(3,largo-1,IDDocument,ID);
        fgets(cadena,1000,archivo);
        if (EsCadenaVacia_D(cadena,largo)==1)
                fgets(cadena,1000,archivo);
        if((cadena[0]!='.' && cadena[1]!='W'))
        {
                largo=strlen(cadena);
                if(cadena[largo-2]!='.' && !(cadena[0]=='.' &&
cadena[1]=='W'))
                        LLenarQuery(cadena,Query);
                else if((cadena[largo-2]=='.'))
                {
                        LLenarQuery(cadena,Query);
                        parar=1;
                }
                else continue;
        }
```

```
                while(!feof(archivo) && parar==0)
                {
                        fgets(cadena,1000,archivo);
                        largo=strlen(cadena);
                        if(cadena[0]=='.' && cadena[1]=='N' && largo>=2)
                                parar=1;
                        else{
                                if(cadena[largo-2]!='.' && !(cadena[0]=='.' &&
cadena[1]=='W'))
                                        LLenarQuery(cadena,Query);
                                else if((cadena[largo-2]=='.'))
                                {
                                        LLenarQuery(cadena,Query);
                                        parar=1;
                                }
                                else continue;
                        }
        }// while
        for(i=1,stop=0;i<999 && stop==0;i++)
                if (Query[i]=='0')
                {
                        Query[i]='\0';
                        stop=1;
```

```
                }
                else continue;
        for(i=1,j=0; Query[i]!='\0';i++,j++)
                if(Query[i]==10)
                        Query2[j]=' ';
                else Query2[j]=Query[i];
                Query2[j]='\0';
        SacarCaracteresNoValidos(Query2);
                headQ=Insertar_Q(headQ,Query2,cont);
                cont=cont+1;
                        for(i=0;i<999;i++)
                                Query[i]='0';
                        Query[i]='\0';
        }// if strcmp
        else continue;
        }// while
        }// if archivo
        fclose(archivo);
        printf("Comenzando a leer los stopwords\n");
        archivo2=fopen("stopwords.txt","r");//archivo de stopwords en nuestro caso no son
necesarias, pues los terminos son creados con letras aleatorias
        if(archivo2!=NULL)
        {
```

```
                while (!feof(archivo2))
                {
                        fgets(stopW,25,archivo2);
                        largo=strlen(stopW);
                        stopW[largo-2]='\0';
                        headSW=Insertar_SW(headSW,stopW);
        }
        fclose(archivo2);
        }
        headQ=SacarStopWords(headQ, headSW);
        MatrizF = Obtener3(headQ);
/****************************************codigo del main Single
Link*********************************************/
        int N,l,n,t,Largo;
        float min,temporal;
        N=LARGOM;
        temporal=0;
        struct Elem **MatrizSim;
        MatrizSim=(struct Elem **)malloc((N+1)*sizeof(struct Elem));
        for(i=0;i<(N+1) ;i++)
                MatrizSim[i]=(struct Elem*)malloc((N+1)*sizeof(struct Elem));
        for ( int i1 = 1; i1 <=N; ++i1)
        {
```

```
            for ( int i2 = 1; i2 <=N; ++i2)
            {
                if ( i1==i2)
                {
                    MatrizSim[i1][i2].valorMatriz=1;
                }
                else{
                    printf("dentro de Single link\n");
                    for (int j = 1; j <= NTERMINOS; ++j)
                    {
                        temporal+=(((struct Terminos **)MatrizF)[i1][j].log *
((struct Terminos **)MatrizF)[i2][j].log);
                    }
                    MatrizSim[i1][i2].valorMatriz=temporal;
                    temporal=0;
                }
            }
            printf("dentro de Single link 2\n");
        }
        free(MatrizF);
        printf("dentro de Single link \n");//estos printf(dentro de single link) los dejo porque,
si no imprimo nada al correr en el servidor a veces se desconecta
        struct Lista *LHead=NULL;
```

```
        struct Cluster *CHead=NULL;
        Arr=(struct Elem*)malloc(N*sizeof(struct Elem));
  // Inicializo los valores del vector Fila Columna en 0
        for(i=0;i<N;i++)
        {
                Arr[i].MarcadoFil=0;
                Arr[i].MarcadoCol=0;
                Arr[i].t=0;
        }
        // memoria para la matriz
        Matriz=(struct Elem **)malloc(N*sizeof(struct Elem));
        for(i=0;i<N ;i++)
                Matriz[i]=(struct Elem*)malloc(N*sizeof(struct Elem));
        // Inicializo los valores de la matriz
        for(i=0;i<N;i++)
                for(j=0;j<N && i>j; j++)
                {
                        Matriz[i][j].valorMatriz = MatrizSim[i+1][j+1].valorMatriz;// paso los
valores de la matriz de similaridad(calculada con distancia del coseno)
                        Matriz[i][j].Marcado=0;
                }
        free(MatrizSim);
        n=N;
```

```
l=-1;
k=-1;
t=1;
min=-1;
while(1)
{
    l=-1;
    min=-1;
    for(i=0;i<N;i++)
  for(j=0;j<N && i>j ;j++)// asumo que la distancia minima es 0
  {
    if(Arr[i].MarcadoFil==0 && Arr[j].MarcadoCol==0)
        if(Matriz[i][j].valorMatriz > min && Matriz[i][j].Marcado==0 &&
Matriz[i][j].valorMatriz > 0)//busco el valor mayor de similaridad
        {
            l=i;
            k=j;
            min=Matriz[i][j].valorMatriz;
        }
        else continue;
    else continue;
  }
```

```
        if(l==-1){//si l es igual a -1 quiere decir que no encontro ningun valor > 0 valido en la matriz para seguir con el algoritmo
            break;
        }
    else if(l!=-1)
        {
            if(Arr[l].MarcadoFil==0)
            {
                Arr[l].MarcadoFil=1;
                Arr[l].t=t;
            }
            if(Arr[k].MarcadoCol==0)
            {
                Arr[k].MarcadoCol=1;
                Arr[k].t=t;
            }
            Matriz[l][k].Marcado=1;
            Matriz[l][k].t=t;
            if(OtroMin(Matriz,N,Arr,min)==1)
            {
                MarcarArr(Matriz,Arr,N,min,t);
                LHead=NULL;
```

```
                    LHead=CrearLista2(Matriz,CHead,LHead, N, t, min);//agrego el min a la
lista
                    if (LHead != NULL)
                    {
                        Largo=LargoL(LHead,0);
                        CHead=InsertarC(CHead, LHead,Largo);// agrego el cluster a la
lista de clusteres
                    }
                }
                else
                {
                    LHead=NULL;
                    LHead=CrearLista2(Matriz,CHead,LHead, N, t, min);
                    if (LHead != NULL)
                    {
                        Largo=LargoL(LHead,0);
                        CHead=InsertarC(CHead, LHead,Largo);
                    }
                }
                ++t;
                n--;
    }
  }//while
```

```
/**************************************codigo del main Single Link Fin*****************************************/
   struct Respuestas *docRAntiguos = NULL;
   struct Respuestas *docRAntiguosAux = NULL;
   FILE *file;
   file = fopen("documentosR.txt","r");// archivo que contiene las id de los documentos relevantes
   if( file == NULL)
   {
         printf("Error al abrir archivo");
   }
   while( !feof(file))//proceso de lectura del archivo file
   {
         if (docRAntiguos == NULL)
         {
                docRAntiguos = (struct Respuestas*)malloc(sizeof(struct Respuestas));
                docRAntiguos->sgte = NULL;
                fscanf(file, "%d ",&docRAntiguos->Id);//almaceno la id de los documentos relevantes
         }
         else{
                docRAntiguosAux = docRAntiguos;
                while(docRAntiguosAux->sgte != NULL){
```

```
                    docRAntiguosAux = docRAntiguosAux->sgte;
                }
                docRAntiguosAux->sgte = (struct Respuestas*)malloc(sizeof(struct
Respuestas));
                docRAntiguosAux->sgte->sgte = NULL;
                fscanf(file, "%d ",&docRAntiguosAux->sgte->Id);
        }
  }
  fclose(file);
  struct Cluster *CHeadI=NULL;
  CHeadI = CHead;
  int contRel = 0;
  while(docRAntiguos != NULL){//aqui cuenta los documentos relevantes visitados al
recorrer el cluster construido con el algoritmo Single Link
        contRel = contRel + contarDocRelEnCluster(CHeadI,docRAntiguos->Id);
        docRAntiguos = docRAntiguos->sgte;
  }
        /*while(CHead != NULL){// este while muestra los clusters creados
          printf("tamano cluster=%d\n",CHead->n );
          while(CHead->L != NULL){
                printf("lista_min=%f\n",CHead->L->min );
                printf("lista_t=%d\n",CHead->L->t );
                printf("posicion i:%d\n",CHead->L->i);
```

```
                printf("posicion j:%d\n",CHead->L->j);

                CHead->L=CHead->L->sgte;

            }

            CHead=CHead->sgte;

        }

        printf("numero de cluster llamados%d\n",NC );*/

    printf("La cantidad de documentos relevantes visitados al recorrer los clusteres
son:%d\n",contRel);

    return 0;

}
```

Codigo fuente de algoritmo algoritmo Complete Link

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<malloc.h>
/*******************************************************************************
 Autores: Delia Moncada - Frederick Lara.
 La forma de compilar este programa es: gcc CompleteLink.c -o completeLink -lm
 Para poder ejecutar correctamente el programa se debe considerar lo siguiente:
  -debe existir un archivo de texto de nombre "documentos.txt" sin comillas, que
   contenga los documentos con los que se construira la matriz de similaridad.
  -se debe cambiar la cantidad de la variable LARGOM,por la cantidad de documentos
   que contenga el archivo de texto de nombre "documentos.txt".
  -debe existir un archivo de texto de nombre "documentosR.txt" sin comillas, que
   contenga las id de los documentos relevantes.
  -ejecutar con el comando: ./completeLink
*******************************************************************************/
int LARGOM=700;
int NTERMINOS;
struct StopWords{
   char stopW[100];
```

```
   struct StopWords *sgte;
};
typedef struct StopWords STOPW;
typedef STOPW *SP;
struct SQUERY2{
 int Id;
 char FQuery[1000];
 struct SQUERY2 *sgte;
};
typedef struct SQUERY2 SQ2;
typedef SQ2 *PQ2;
struct Terminos{
 char Termino[30];
 int esta;
 float log;
 int ID;
};
typedef struct Terminos TER;
typedef TER *TER2;
struct lista_Doc_Query{
 int IDQuery;
 int IDDocumento;
```

```
  struct lista_Doc_Query *sgte;
};
struct Respuestas{
  int Id;
  float sim;
  struct Respuestas *sgte;
};
struct GlobalRespuestas{
  int Id;
  struct Respuestas *dato;
  struct GlobalRespuestas *sgte;
};
struct Respuestas_{
  int Id;
  float sim;
};
struct Documentos{
  int Id;
  char Documento[1500];
  struct Documentos *sgte;
};
typedef struct Documentos Docu;
```

```
typedef Docu *DOC;

struct GlobalRespuestas *InsertarGR(struct GlobalRespuestas *headGR, struct Respuestas
*dato)

{

 if(headGR==NULL)

 {

   headGR=(struct GlobalRespuestas *)malloc(sizeof(struct GlobalRespuestas));

   headGR->dato=dato;

   headGR->sgte=NULL;

 }

 else headGR->sgte=InsertarGR(headGR->sgte,dato);

 return(headGR);

}

DOC InsertarD(DOC head, char Documento[], int Id)

{

 if(head==NULL)

 {

   head=(DOC)malloc(sizeof(Docu));

   head->Id=Id;

   strcpy(head->Documento,Documento);

   head->sgte=NULL;

 }

 else head->sgte=InsertarD(head->sgte,Documento,Id);
```

```
 return(head);
}
struct Respuestas *Insertar(struct Respuestas *head, int Id, float sim)
{
 if(head==NULL)
 {
  head=(struct Respuestas *)malloc(sizeof(struct Respuestas));
  head->Id=Id;
  head->sim=sim;
  head->sgte=NULL;
 }
 else head->sgte=Insertar(head->sgte,Id,sim);
 return(head);
}
int ContarPalabras(char cadena[])
{
 int i,cont,pal;
 for(i=0,cont=0,pal=0;cadena[i]!='\0';i++)
  if(cadena[i]!=' ')
    pal++;
  else if(pal!=0)
  {
```

```
    cont++;
    pal=0;
   }
   else continue;
   if(pal!=0)
    return(cont+1);
   else return(cont);
}
void Copiar_iesimo(int inicio,int largo,char original[],char destino[])
{
 int i,j;
 for(i=1,j=0;i<=largo;inicio++,i++,j++)
   destino[j]=original[inicio];
 destino[j]='\0';
}
void SacarCaracteresNoValidos(char cadena[])
{
   int i;
   for(i=0; cadena[i]!='\0';i++)
      if(cadena[i]=='(' || cadena[i]==')' || cadena[i]=='?' || cadena[i]=='-' || cadena[i]==';' ||
cadena[i]==',' || cadena[i]=='"' || cadena[i]==':' || cadena[i]=='*' || cadena[i]=='+' ||
cadena[i]=='/' || cadena[i]==9)
        cadena[i]=' ';
```

```
 else if(cadena[i]=='.')
    cadena[i]=' ';
    else continue;
for(i=0; cadena[i]!='\0';i++)
  if(cadena[i]=='A')
     cadena[i]='a';
  else if(cadena[i]=='B')
        cadena[i]='b';
     else if(cadena[i]=='C')
           cadena[i]='c';
        else if(cadena[i]=='D')
              cadena[i]='d';
           else if(cadena[i]=='E')
                 cadena[i]='e';
              else if(cadena[i]=='F')
                    cadena[i]='f';
                 else if(cadena[i]=='G')
                       cadena[i]='g';
                    else if(cadena[i]=='H')
                          cadena[i]='h';
                       else if(cadena[i]=='I')
                             cadena[i]='i';
```

```
                else if(cadena[i]=='J')
                    cadena[i]='j';
                else continue;
for(i=0; cadena[i]!='\0';i++)
 if(cadena[i]=='K')
    cadena[i]='k';
 else if(cadena[i]=='L')
      cadena[i]='l';
    else if(cadena[i]=='M')
       cadena[i]='m';
      else if(cadena[i]=='N')
          cadena[i]='n';
        else if(cadena[i]=='O')
            cadena[i]='o';
          else if(cadena[i]=='P')
              cadena[i]='p';
            else if(cadena[i]=='Q')
                cadena[i]='q';
              else if(cadena[i]=='R')
                  cadena[i]='r';
                else if(cadena[i]=='T')
                    cadena[i]='t';
```

```
                else if(cadena[i]=='S')
                    cadena[i]='s';
                else if(cadena[i]=='U')
                    cadena[i]='u';
                else if(cadena[i]=='V')
                    cadena[i]='v';
                else if(cadena[i]=='W')
                    cadena[i]='w';
                else if(cadena[i]=='X')
                    cadena[i]='x';
                else if(cadena[i]=='Y')
                    cadena[i]='y';
                else if(cadena[i]=='Z')
                    cadena[i]='z';
                else continue;
}
int CaracterValido(char cadena[],int i)
{
 while(cadena[i]!='\0')
 {
  if(cadena[i]!=' ')
   return(i);
```

```
   else i=i+1;
  }
   return(-1); // son puros espacios en blancos
}
void SacarEspaciosS(char cadena[])
{
 int i,j,k;
 i=0;
 while(cadena[i]!='\0')
 {
  if(cadena[i]==' ' || cadena[i]==9)
  {
   if(cadena[i]==' ' && cadena[i+1]==' ' && cadena[i+1]!='\0')
   {
        k=CaracterValido(cadena,i); // si es -1 entonces son puros espacios los faltantes
        if(k!=-1)
        {
         for(j=i+1;cadena[k]!='\0';j++,k++)
         {
          cadena[j]=cadena[k];
         }
         cadena[j]='\0';
```

```
            i=i+1;
           }
           else cadena[i]='\0';
          }
          else  i=i+1;
      }
      else  i=i+1;
   }
}
int CompararCadenas(char cad1[],char cad2[])
{
 int i,largo1,largo2,stop;
 largo1=strlen(cad1);
 largo2=strlen(cad2);
 if(largo1==largo2)
   for(i=0, stop=1; cad1[i]!='\0' && stop==1;i++)
     if(cad1[i]!=cad2[i])
       stop=0;
     else continue;
 else return(0);
 return(stop);
}
```

```
int Buscar(char cadena[],char frase[])
{
 char Aux[30];
 int i,largo,largo2,total;
 largo=strlen(cadena);
 largo2=strlen(frase);
  for(i=0;frase[i]!='\0' && ((i+largo)<=largo2);i++)//aqui estaba el error, que no consideraba el ultimo termino del documento
  if(frase[i]==cadena[0])
  {
   Copiar_iesimo(i,largo,frase,Aux);
   total=i+largo;
   if(CompararCadenas(Aux,cadena)==1)
   {
    return(1);
   }
   else continue;
  }
  else continue;
  return(0);
}
PQ2 Insertar_Q(PQ2 head,char stopW[], int Id)
{
```

```
 if(head==NULL)
 {
  head=(PQ2)malloc(sizeof(SQ2));
  head->sgte=NULL;
  head->Id=Id;
  strcpy(head->FQuery,stopW);
 }
 else head->sgte= Insertar_Q(head->sgte,stopW,Id);
 return(head);
}
SP Insertar_SW(SP headSW,char stopW[])
{
 if(headSW==NULL)
 {
  headSW=(SP)malloc(sizeof(STOPW));
  headSW->sgte=NULL;
  strcpy(headSW->stopW,stopW);
 }
 else headSW->sgte= Insertar_SW(headSW->sgte,stopW);
 return(headSW);
}
int esSubcadena(char Query[],int i,int largo, SP headSW)
```

```
{
 int LARGO;
 LARGO=strlen(Query);
 if(i!=0 && Query[i-1]!=' ')
  return(1);
 else if(Query[i+largo]!=' ' && i+largo<LARGO && Query[i+largo]!='\0')
  return(1);
 else return(0);
}
SP Sacar(char Query[],SP headSW)
{
 char Aux[50];
 int largo,i,j,k;
 if(headSW!=NULL)
 {
  largo=strlen(headSW->stopW);
  for(i=0; Query[i]!='\0';i++)
    if(Query[i]==headSW->stopW[0] && !esSubcadena(Query,i,largo,headSW))
    {
      Copiar_iesimo(i,largo,Query,Aux);
      if(!strcmp(headSW->stopW,Aux))
      {
```

```
if(Query[i+largo]!='\0')
{
 if(Query[i+largo]=='0')
 {
  k=i+largo+1;
  j=i;
  while(Query[k]!='\0')
  {
   Query[j]=Query[k];
   j++;
   k++;
  }
  Query[j]='\0';
 }
 else{
  k=i+largo;
  j=i;
  while(Query[k]!='\0')
  {
   Query[j]=Query[k];
   j++;
   k++;
```

```
            }
            Query[j]='\0';
          }
        }
        else Query[i]='\0';
      }
      else continue;
    }
    else continue;
    headSW->sgte=Sacar(Query,headSW->sgte);
  }
  else return(NULL);
  return(headSW);
}
DOC SacarStopWords_D(DOC head, SP headSW)
{
  char Documento[1500];
  int i,j;
  if(head!=NULL)
  {
   strcpy(Documento, head->Documento);
   headSW=Sacar(Documento, headSW);
```

```
   SacarEspaciosS(Documento);
   if(Documento[0]==' ')
   {
    for(i=0, j=i+1;Documento[j]!='\0';i++,j++)
      Documento[i]=Documento[j];
    Documento[i]='\0';
   }
   strcpy( head->Documento, Documento);
   head->sgte=SacarStopWords_D(head->sgte, headSW);
  }
  else return(NULL);
  return(head);
}
PQ2 SacarStopWords(PQ2 head, SP headSW)
{
  char Query[1000];
  int i,j;
  if(head!=NULL)
  {
   strcpy(Query, head->FQuery);
   headSW=Sacar(Query, headSW);
   SacarEspaciosS(Query);
```

```
   if(Query[0]==' ')
   {
    for(i=0, j=i+1;Query[j]!='\0';i++,j++)
      Query[i]=Query[j];
    Query[i]='\0';
   }
   strcpy( head->FQuery, Query);
   head->sgte=SacarStopWords(head->sgte, headSW);
 }
 else return(NULL);
 return(head);
}
void  LLenarQuery(char cadena[],char Query[])
{
 int i,stop=0,j;
 for(i=0; Query[i]!='\0' && stop==0;i++)
   if(Query[i]=='0')
   {
    j=i;
    stop=1;
   }
   else continue;
```

```
   Query[j]=' ';
   j=i;
   for(i=0; cadena[i]!='\0' && Query[j]!='\0'; i++, j++)
     Query[j]=cadena[i];
}
int EsCadenaVacia(char cadena[], int largo)
{
 int i;
 for(i=0; i<largo;i++)
   if(cadena[i]!=' ')
     return(1);
   else continue;
 return(0);
}
int ContarSW(SP headSW, int i)
{
 if(headSW!=NULL)
   return(ContarSW(headSW->sgte,i+1));
 else return(i);
}
int NQueries(PQ2 headQ, int cont)
{
```

```
 if(headQ!=NULL)
   return(NQueries(headQ->sgte, cont+1));
 else return(cont);
}
void Copiar(struct Respuestas *head,struct Respuestas_ *Arr,int inic,int largo)
{
 if(head!=NULL && inic<largo)
 {
   Arr[inic].Id=head->Id;
   Arr[inic].sim=head->sim;
   Copiar(head->sgte,Arr,inic+1,largo);
 }
}
int LargoRespuestas(struct Respuestas *head, int largo)
{
 if(head!=NULL)
   return(LargoRespuestas(head->sgte, largo+1));
 else return(largo);
}
struct Respuestas *Ordenar(struct Respuestas *head)
{
 int largo,i,j,k;
```

```
struct Respuestas_ *Arr,Aux;
struct Respuestas *head2=NULL;
largo=LargoRespuestas(head,0);
Arr=(struct Respuestas_ *)malloc(largo*sizeof(struct Respuestas_ ));
Copiar(head,Arr,0,largo);
for(j=0;j<largo;j++)
{
 Aux=Arr[j];
 for(i=j+1,k=j;i<largo;i++)
 {
  if(Arr[i].sim > Aux.sim)
  {
   Aux=Arr[i];
   k=i;
  }
  else continue;
 }
 Arr[k]=Arr[j];
 Arr[j]=Aux;
}
for(i=0;i<largo;i++)
  head2=Insertar(head2,Arr[i].Id,Arr[i].sim);
```

```
 return(head2);
}
int ContarQDValidos(struct Terminos **MatrizT, int fil, int col)
{
 int i,cont;
 for(i=1,cont=0;i<fil;i++)
   if(MatrizT[i][0].esta==1)
   {
    cont++;
   }
   else continue;
 return(cont);
}
float Similaridad(struct Terminos **MatrizT, int NT, int largo,int ID1,int ID2)
{
 float s,s2,s3,sim;
 int j;
 for(j=1,s=0.0,s2=0.0,s3=0.0;j<NT;j++)
 {
   s=s+MatrizT[ID1][j].log*MatrizT[ID2][j].log;
   s2=s2+MatrizT[ID1][j].log*MatrizT[ID1][j].log;
   s3=s3+MatrizT[ID2][j].log*MatrizT[ID2][j].log;
```

```
 }
 if(s2!=0.0 && s3!=0.0)
   sim=s/sqrt(s2*s3);
 else sim=0.0;
 return(sim);
}
int ContarTerminosCol(struct Terminos **MatrizT,int col, int NT, int largo)
{
 int i,cont;
 for(cont=0,i=1;i<=largo;i++)
 {
   if(MatrizT[i][col].esta>=1)
     cont++;
   else continue;
 }
 return(cont);
}
void LLenarLDF(struct Terminos **MatrizT,int col,int NT, int largo,float ldf)
{
 int i;
 for(i=1;i<largo;i++)
 {
```

```
   if(MatrizT[i][col].esta>=1)
   {
     MatrizT[i][col].log=ldf*(float) MatrizT[i][col].esta;
   }
   else {
     MatrizT[i][col].log=0;
   }
  }
}
PQ2 Esta4(struct Terminos **MatrizT,int Nterm, PQ2 headQ, int actual)
{
  int j,i,cont,largo;
  char Aux[100];
  if(headQ!=NULL)
  {
   for(j=1;j< Nterm;j++)
   {
    if(Buscar(MatrizT[0][j].Termino,headQ->FQuery)==1)
    {
      largo=strlen(MatrizT[0][j].Termino);
      for(i=0,cont=0;headQ->FQuery[i]!='\0';i++)
        if(MatrizT[0][j].Termino[0]==headQ->FQuery[i])
```

```
        {
          Copiar_iesimo(i,largo,headQ->FQuery,Aux);
          if(CompararCadenas(Aux,MatrizT[0][j].Termino)==1)
            cont++;
          else continue;
        }
        else continue;
       MatrizT[actual][j].esta=cont;
       MatrizT[actual][0].esta=1;
     }
     else continue;
   }
   headQ->sgte=Esta4(MatrizT,Nterm,headQ->sgte,actual+1);
 }
 return(headQ);
}
int NDocumentos(DOC headD, int cont)
{
 if(headD!=NULL)
   return(NDocumentos(headD->sgte, cont+1));
 else return(cont);
}
```

```
int ObtenerIDDocumentos(DOC headD,int cont, int fin,int largo)
{
 if(cont==fin && fin<=largo && headD!=NULL)
   return(headD->Id);
 else return(ObtenerIDDocumentos(headD->sgte,cont+1, fin ,largo));
}
int ObtenerIDQueries(PQ2 headQ,int cont, int fin,int largo)
{
 if(cont==fin && fin<=largo && headQ!=NULL)
   return(headQ->Id);
 else return(ObtenerIDQueries(headQ->sgte,cont+1, fin ,largo));
}
int EsCadenaVacia_D(char cadena[], int largo)
{
 int i,stop=1;
 if(largo==1 && cadena[0]==10)
   return(1);
 else for(i=0;i<largo && stop==1; i++)
 if(cadena[i]!=10)
   stop=0;
 else continue;
 return(stop);
```

```
}
void  LLenarDocumento(char cadena[],char Documento[])
{
 int i,stop=0,j;
 for(i=0; Documento[i]!='\0' && stop==0;i++)
  if(Documento[i]=='\0')
  {
   j=i;
   stop=1;
  }
  else continue;
 if(j!=0)
 {
  Documento[j]=' ';
  j=i;
 }
 for(i=0,stop=0; cadena[i]!='\0'  && Documento[j]!='\0' && stop==0; i++, j++)
 {
  if(cadena[i]!=10)
  {
   Documento[j]=cadena[i];
  }
```

```
   else stop=1;
  }
  j=j-1;
  Documento[j]=' ';
}
void ObtenerQuery(int inic,int actual,int largo, char QueryAux[],PQ2 head)
{
  if(head!=NULL && inic<=largo)
  {
    if(inic==actual)
    {
      strcpy(QueryAux, head->FQuery);
    }
    else ObtenerQuery(inic+1,actual,largo,QueryAux,head->sgte);
  }
}
void ObtenerT(int fin,int inic,SP head,char Termino[])
{
  if(head!=NULL && inic== fin)
    strcpy(Termino,head->stopW);
  else  ObtenerT(fin,inic+1, head->sgte, Termino);
}
```

```
void ColocarTerminos3(int col,struct Terminos **M,int Nterminos,SP head)
{
 char Termino[30];
 ObtenerT(col,1,head,Termino);
 strcpy(M[0][col].Termino,Termino);
}
int ContarTerminos(SP head,int cont)
{
 if(head!=NULL)
  return(ContarTerminos(head->sgte,cont+1));
 else return(cont);
}
int EstaT(SP head,char Term[])
{
 if(head!=NULL)
 {
  if(CompararCadenas(head->stopW,Term)==1)
  {
   return(1);
  }
  else return(EstaT(head->sgte,Term));
 }
```

```
 else return(0);
}
SP ColocarTerminosD(SP head,DOC headD)
{
 int i,j,largo,k;
 char Term[100],Documento[1500];
 if(headD!=NULL)
 {
  strcpy(Documento,headD->Documento);
  largo=strlen(Documento);
  for(k=0;k<=largo+3; k++)
   i=0;
  while( Documento[i]!='\0' && i<largo)
  {
   j=0;
   while(Documento[i]!='\0' &&  Documento[i]!=' ' && i<largo)
   {
    Term[j]=Documento[i];
    j++;
    i++;
   }
   Term[j]='\0';
```

```
    if(EstaT(head,Term)==0)
    {
      head=Insertar_SW(head,Term);
    }
    i=i+1;
  }
  head= ColocarTerminosD(head,headD->sgte);
 }
 return(head);
}
SP ColocarTerminos(SP head,PQ2 headQ)
{
 int i,j,largo,k;
 char Term[100],FQueryA[1000];
 if(headQ!=NULL)
 {
  strcpy(FQueryA,headQ->FQuery);
  largo=strlen(FQueryA);
  for(k=0;k<=largo+3; k++)
   i=0;
  while(FQueryA[i]!='\0' && i<largo)
  {
```

```
    j=0;
    while(FQueryA[i]!='\0' &&  FQueryA[i]!=' ' && i<largo)
    {
     Term[j]=FQueryA[i];
     j++;
     i++;
    }
    Term[j]='\0';
    if(EstaT(head,Term)==0)
    {
     head=Insertar_SW(head,Term);
    }
    i=i+1;
   }
   head= ColocarTerminos(head,headQ->sgte);
 }
 return(head);
}
struct Terminos **Obtener3( PQ2 headQ)
{
 struct Terminos **MatrizT;
 int largo,Nterminos,i,j,LAux,ID;
```

```
float ldf,sim;
struct Respuestas **Tabla;
SP T=NULL;
T=ColocarTerminos(T,headQ);
Nterminos=ContarTerminos(T,0);
largo=LARGOM;
largo=largo+1;
Nterminos=Nterminos+1;
MatrizT=(struct Terminos **)malloc(largo*sizeof(struct Terminos *));
for(i=0;i<largo;i++)
 MatrizT[i]=(struct Terminos *)malloc(Nterminos*sizeof(struct Terminos));
for(i=0;i<largo;i++)
 for(j=0;j<Nterminos;j++) // el 0 es por si es que no tiene ningun termino en comun
 {
  MatrizT[i][j].esta=0;
  MatrizT[i][0].ID=i+1;
  MatrizT[i][0].log=0.0;
 }
 for(j=1; j<Nterminos ;j++)// esto coloca todos los terminos de todas las queries en la lista T
  ColocarTerminos3(j,MatrizT,Nterminos,T);
 headQ=Esta4(MatrizT,Nterminos,headQ,1);
 NTERMINOS=Nterminos;
```

```
for(j=1;j<Nterminos;j++)
{
 LAux=ContarTerminosCol(MatrizT,j,Nterminos,largo-1);
 if(LAux!=0)
 {
  ldf=log10((largo-1)/(float)LAux);
 }
 else ldf=0.0;
 LLenarLDF(MatrizT,j,Nterminos,largo,ldf);
}
ID=1;
Tabla=(struct Respuestas **)malloc(LARGOM*sizeof(struct Respuestas));
for(i=0;i<LARGOM;i++)
 Tabla[i]=NULL;
for(ID=0;ID<LARGOM;ID++)
{
 for(i=1; i<largo;i++)
 {
  if(i!=(ID+1))
  {
   sim=Similaridad(MatrizT,Nterminos,largo,i,(ID+1));
  }
```

```
      else continue;
    }
    printf("dentrofor:%d\n",ID );
  }
 return MatrizT;
}
/******************************************************* Codigo CompleteLink
*************************************************************/
int N;
struct Elem{
  // Vector
 int indice;
 int idDoc;
 int Topic;
 float TermDoc[2]; // char TermDoc[401]
 int MarcadoFil;
 int MarcadoCol;
 // Matriz
 float valorMatriz; //calculo de distancia
 int Marcado;
 int t;
};
struct Elem *Arr; // son los documentos, de aqu¡ formo la matriz
```

```
struct Elem **Matriz; // esta es la matriz de distancias
struct Lista{ // esta lista contiene los de un mismo cluster elementos
 int idDoc[20];
 int t;
 float min;
 int n;
 int i;
 int j;
 struct Lista *sgte;
};
struct Cluster{
 struct Lista *L; //esta es la lista de los cluster
 struct Cluster *sgte;
 int n;
};
struct Lista *InsertarL(struct Lista *H, int idDoc[], int N, int n, float min,int t)
{
 int i;
 if(H==NULL)
 {
   H=(struct Lista *)malloc(sizeof(struct Lista));
   H->sgte=NULL;
```

```
   H->min=min;
   H->t=t;
   H->n=n;
   for(i=0; i<N;i++)
    if(i<n)
      H->idDoc[i]=idDoc[i];
    else H->idDoc[i]=-1;
 }
 else H->sgte=InsertarL(H->sgte,idDoc,N,n,min,t);
 return(H);
}
int LargoL(struct Lista *H,int i)
{
 if(H!=NULL)
   return(LargoL(H->sgte,i+1));
 else return(i);
}
int NC=0;
struct Cluster *InsertarC(struct Cluster *C, struct Lista *H, int n)
{
 NC++;
 if(C==NULL)
```

```
 {
   C=(struct Cluster *)malloc(sizeof(struct Cluster));
   C->L=H;
   C->n=n;
   C->sgte=NULL;
 }
 else C->sgte=InsertarC(C->sgte,H,n);
 return(C);
}
struct Lista *CrearLista2(struct Elem **M,struct Cluster *C,struct Lista *H,int N,int t, float min)
{
 for (int i = 0; i < LARGOM; ++i)
 {
   for (int j = 0; j < LARGOM && i>j ; ++j)
   {
     if (M[i][j].Marcado == 1 && H == NULL)
     {
       H=(struct Lista *)malloc(sizeof(struct Lista));
       H->t=t;
       H->min=M[i][j].valorMatriz;
       H->i=i;
       H->j=j;
```

```
      H->sgte=NULL;
      M[i][j].Marcado = 2;
    }
    else{
     if (M[i][j].Marcado == 1 && H != NULL)
     {
       C=InsertarC(C,H,1);
       H=NULL;
       H=(struct Lista *)malloc(sizeof(struct Lista));
       H->t=t;
       H->min=M[i][j].valorMatriz;
       H->i=i;
       H->j=j;
       H->sgte=NULL;
       M[i][j].Marcado = 2;
     }
    }
   }
  }
  return(H);
}
int OtroMin(struct Elem **M,int N,struct Elem *A,float min)
```

```
{
 int i,j;
 for(i=0;i<N;i++)
   for(j=0;j<N && i>j ;j++)// asumo que la distancia minima es 0
   {
    if(A[i].MarcadoFil==0 && A[j].MarcadoCol==0)
     if(M[i][j].valorMatriz==min && M[i][j].Marcado==0 && !(A[i].MarcadoFil==1 &&
A[i].MarcadoCol==1) && !(A[j].MarcadoFil==1 && A[j].MarcadoCol==1))
     {
      return(1);
     }
     else continue;
    else continue;
   }
 return(0);
}
void MarcarArr(struct Elem **M,struct Elem *A,int N,float min,int t)
{
 int i,j;
 for(i=0;i<N;i++)
   for(j=0;j<N && i>j ;j++)// asumo que la distancia minima es 0
   {
    if(A[i].MarcadoFil==0 && A[j].MarcadoCol==0)
```

```
      if(M[i][j].valorMatriz==min && M[i][j].Marcado==0)
      {
        A[i].MarcadoFil=1;
        A[j].MarcadoCol=1;
        M[i][j].Marcado=1;
        M[i][j].t=t;
        A[i].t=t;
        A[j].t=t;
      }
      else continue;
    else continue;
  }
}
int contarDocRelEnCluster(struct Cluster *CHead, int id){
 int cont = 0;
 struct Lista *l = NULL;
 while(CHead != NULL){
  l = CHead->L;
  while(l != NULL){
    if ((l->i + 1) == id || (l->j+1) == id)// el +1 es porque para los cluster la matriz la comence desde la posicion 0 y no del 1
    {
      cont++;
```

```
    }
    l=l->sgte;
   }
   CHead=CHead->sgte;
 }
 return(cont);
}
/****************************************************** Codigo CompleteLink Fin
******************************************************/
int main(int argc, char *argv[])
{
 FILE *archivo,*archivo2;
 char cadena[1000],Query[1000],Query2[1000],IDQuery[6],Band[3],Aux[5],Band2[3];
 char stopW[25];
 int i, parar,largo,k,stop,j,cont=1;
 SP headSW=NULL;
 PQ2 headQ=NULL;
// estas definiciones son de los documentos
 char Documento[1500],IDDocument[8],Band3[3],Band4[3],ID[8];
 struct Terminos **MatrizF=NULL;
 for(i=0;i<999;i++)
   Query[i]='0';
 Query[i]='\0';
```

```
 for(i=0;i<1499;i++)
   Documento[i]='0';
 Documento[i]='\0';
 Band[0]='.',Band[1]='I', Band[2]='\0';
 Band2[0]='.',Band2[1]='T', Band[2]='\0';
 Band3[0]='.', Band3[1]='W',Band3[2]='\0';
 Band4[0]='.', Band4[1]='B',Band4[2]='\0';
 archivo=fopen("documentos.txt","r");//nombre del archivo de texto donde estan los
documentos
 if(archivo!=NULL)
 {
  while (!feof(archivo))
  {
    parar=0;
    fgets(IDQuery,6,archivo);
    Copiar_iesimo(0,2,IDQuery,Aux);
    if(!strcmp(Band,Aux))
    {
     largo=strlen(IDDocument);
     Copiar_iesimo(3,largo-1,IDDocument,ID);
     fgets(cadena,1000,archivo);
     if(EsCadenaVacia_D(cadena,largo)==1)
      fgets(cadena,1000,archivo);
```

```
if((cadena[0]!='.' && cadena[1]!='W'))
{
 largo=strlen(cadena);
 if(cadena[largo-2]!='.' && !(cadena[0]=='.' && cadena[1]=='W'))
  LLenarQuery(cadena,Query);
 else if((cadena[largo-2]=='.'))
 {
  LLenarQuery(cadena,Query);
  parar=1;
 }
 else continue;
}
while(!feof(archivo) && parar==0)
{
 fgets(cadena,1000,archivo);
 largo=strlen(cadena);
 if(cadena[0]=='.' && cadena[1]=='N' && largo>=2)
  parar=1;
 else{
  if(cadena[largo-2]!='.' && !(cadena[0]=='.' && cadena[1]=='W'))
    LLenarQuery(cadena,Query);
  else if((cadena[largo-2]=='.'))
```

```
  {
    LLenarQuery(cadena,Query);
    parar=1;
  }
  else continue;
 }
}// while
for(i=1,stop=0;i<999 && stop==0;i++)
 if (Query[i]=='0')
 {
  Query[i]='\0';
  stop=1;
 }
 else continue;
for(i=1,j=0; Query[i]!='\0';i++,j++)
 if(Query[i]==10)
  Query2[j]=' ';
 else Query2[j]=Query[i];
 Query2[j]='\0';
 SacarCaracteresNoValidos(Query2);
 headQ=Insertar_Q(headQ,Query2,cont);
 cont=cont+1;
```

```
        for(i=0;i<999;i++)
          Query[i]='0';
        Query[i]='\0';
     }// if strcmp
     else continue;
   }// while
 }// if archivo
 fclose(archivo);
 printf("Comenzando a leer los stopwords\n");
 archivo2=fopen("stopwords.txt","r");//archivo de stopwords en nuestro caso no son necesarias, pues los terminos son creados con letras aleatorias
 if(archivo2!=NULL)
 {
   while (!feof(archivo2))
   {
     fgets(stopW,25,archivo2);
     largo=strlen(stopW);
     stopW[largo-2]='\0';
     headSW=Insertar_SW(headSW,stopW);
   }
   fclose(archivo2);
 }// if archivo2
 headQ=SacarStopWords(headQ, headSW);
```

```
 MatrizF = Obtener3(headQ);

/****************************************codigo del main Complete
Link*********************************************/

 int N,l,n,t,Largo;

 float min,temporal;

 N=LARGOM;

 temporal=0;

 struct Elem **MatrizSim;

 MatrizSim=(struct Elem **)malloc((N+1)*sizeof(struct Elem));

 for(i=0;i<(N+1) ;i++)

  MatrizSim[i]=(struct Elem*)malloc((N+1)*sizeof(struct Elem));

 int i1,i2;

 for (i1 = 1; i1 <=N; ++i1)

 {

  for (i2 = 1; i2 <=N; ++i2)

  {

   if ( i1==i2)

   {

    MatrizSim[i1][i2].valorMatriz=1;

   }

   else{

    for (int j = 1; j <= NTERMINOS; ++j)

    {
```

```
        temporal+=(((struct Terminos **)MatrizF)[i1][j].log * ((struct Terminos
**)MatrizF)[i2][j].log);
       }
       MatrizSim[i1][i2].valorMatriz=temporal;
       temporal=0;
     }
   }
   printf("dentro de Complete Link 2\n");
 }
 free(MatrizF);
 printf("dentro de Complete Link \n");//estos printf(dentro de Complete Link los dejo
porque, si no imprimo nada al correr en el servidor a veces se desconecta
 struct Lista *LHead=NULL;
 struct Cluster *CHead=NULL;
 Arr=(struct Elem*)malloc(N*sizeof(struct Elem));
   // Inicializo los valores del vector Fila Columna en 0
 for(i=0;i<N;i++)
 {
   Arr[i].MarcadoFil=0;
   Arr[i].MarcadoCol=0;
   Arr[i].t=0;
 }
  // memoria para la matriz
```

```
 Matriz=(struct Elem **)malloc(N*sizeof(struct Elem));
 for(i=0;i<N ;i++)
  Matriz[i]=(struct Elem*)malloc(N*sizeof(struct Elem));
 // Inicializo los valores de la matriz
 for(i=0;i<N;i++)
  for(j=0;j<N && i>j; j++)
  {
   Matriz[i][j].valorMatriz = MatrizSim[i+1][j+1].valorMatriz;//ingresando los valores de la matriz de similaridad
   Matriz[i][j].Marcado=0;
  }
 free(MatrizSim);
 n=N;
 l=-1;
 k=-1;
 t=1;
 min=100000;
 while(1)
 {
  l=-1;
  min=100000;
  for(i=0;i<N;i++)
   for(j=0;j<N && i>j ;j++)// asumo que la distancia minima es 0
```

```
    {
      if(Arr[i].MarcadoFil==0 && Arr[j].MarcadoCol==0)
       if(Matriz[i][j].valorMatriz < min && Matriz[i][j].Marcado==0 && Matriz[i][j].valorMatriz
> 0)//busco el menor valor de similaridad
       {
        l=i;
        k=j;
        min=Matriz[i][j].valorMatriz;
       }
       else continue;
      else continue;
    }
  if(l==-1)//si l es igual a -1 quiere decir que no encontro ningun valor > 0 valido en la matriz
para seguir con el algoritmo
  {
   break;
  }
  else if(l!=-1 && min != 0)
  {
   if(Arr[l].MarcadoFil==0)
   {
    Arr[l].MarcadoFil=1;
    Arr[l].t=t;
```

```
}
if(Arr[k].MarcadoCol==0)
{
 Arr[k].MarcadoCol=1;
 Arr[k].t=t;
}
Matriz[l][k].Marcado=1;
Matriz[l][k].t=t;
if(OtroMin(Matriz,N,Arr,min)==1)
{
 MarcarArr(Matriz,Arr,N,min,t);
 LHead=NULL;
 LHead=CrearLista2(Matriz,CHead,LHead, N, t, min);//agrego el min a la lista
 if (LHead != NULL)
 {
  Largo=LargoL(LHead,0);
  CHead=InsertarC(CHead, LHead,Largo);// agrego el cluster a la lista de clusteres
 }
}
else{
 LHead=NULL;
 LHead=CrearLista2(Matriz,CHead,LHead, N, t, min);
```

```
      printf("despues lista2\n");
      if (LHead != NULL)
      {
        Largo=LargoL(LHead,0);
        CHead=InsertarC(CHead, LHead,Largo);
      }
    }
    ++t;
    n--;
  }
 }//while
/*****************************************codigo del main Single Link
Fin*********************************************/
 struct Respuestas *docRAntiguos = NULL;
 struct Respuestas *docRAntiguosAux = NULL;
 FILE *file;
 file = fopen("documentosR.txt","r");// archivo que contiene las id de los documentos
relevantes
 if( file == NULL)
 {
  printf("Error al abrir archivo");
 }
 while( !feof(file))//proceso de lectura del archivo file
```

```
 {
  if (docRAntiguos == NULL)
  {
   docRAntiguos = (struct Respuestas*)malloc(sizeof(struct Respuestas));
   docRAntiguos->sgte = NULL;
   fscanf(file, "%d ",&docRAntiguos->Id);
  }
  else{
   docRAntiguosAux = docRAntiguos;
   while(docRAntiguosAux->sgte != NULL){
    docRAntiguosAux = docRAntiguosAux->sgte;
   }
   docRAntiguosAux->sgte = (struct Respuestas*)malloc(sizeof(struct Respuestas));
   docRAntiguosAux->sgte->sgte = NULL;
   fscanf(file, "%d ",&docRAntiguosAux->sgte->Id);//almaceno la id de los documentos
relevantes
  }
 }
 fclose(file);
 struct Cluster *CHeadI=NULL;
 CHeadI = CHead;
 int contRel = 0;
```

```
 while(docRAntiguos != NULL){//aqui cuenta los documentos relevantes visitados al recorrer el cluster construido con el algoritmo Complete Link
   contRel = contRel + contarDocRelEnCluster(CHeadI,docRAntiguos->Id);
   docRAntiguos = docRAntiguos->sgte;
 }
 /*while(CHead != NULL){
   printf("tamano cluster=%d\n",CHead->n );
   while(CHead->L != NULL){
     printf("lista_min=%f\n",CHead->L->min );
     printf("lista_t=%d\n",CHead->L->t );
     printf("posicion i:%d\n",CHead->L->i);
     printf("posicion j:%d\n",CHead->L->j);
     CHead->L=CHead->L->sgte;
   }
   CHead=CHead->sgte;
 }
 printf("numero de cluster llamados%d\n",NC );*/
 printf("La cantidad de documentos relevantes visitados al recorrer los clusteres son:%d\n",contRel);
 return 0;
}
```

Codigo fuente de algoritmo Average Link

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<malloc.h>
#include<stdbool.h>
/*****************************************************************************
 Autores: Delia Moncada - Frederick Lara.
 La forma de compilar este programa es: gcc AverageLink.c -o averageLink -lm
 Para poder ejecutar correctamente el programa se debe considerar lo siguiente:
  -debe existir un archivo de texto de nombre "documentos.txt" sin comillas, que
   contenga los documentos con los que se construira la matriz de similaridad.
  -se debe cambiar la cantidad de la variable LARGOM,por la cantidad de documentos
   que contenga el archivo de texto de nombre "documentos.txt".
  -debe existir un archivo de texto de nombre "documentosR.txt" sin comillas, que
   contenga las id de los documentos relevantes.
  -ejecutar con el comando: ./averageLink
*****************************************************************************/
int LARGOM=700;// LARGOM es la cantidad de documentos que contiene el File archivo
int NTERMINOS;
struct StopWords{
```

```
   char stopW[100];
   struct StopWords *sgte;
};
typedef struct StopWords STOPW;
typedef STOPW *SP;
struct SQUERY2{
 int Id;
 char FQuery[1000];
 struct SQUERY2 *sgte;
};
typedef struct SQUERY2 SQ2;
typedef SQ2 *PQ2;
struct Terminos{
 char Termino[30];
 int esta;
 float log;
 int ID;
};
typedef struct Terminos TER;
typedef TER *TER2;
struct lista_Doc_Query{
 int IDQuery;
```

```
  int IDDocumento;
  struct lista_Doc_Query *sgte;
};
struct Respuestas{
  int Id;
  float sim;
  struct Respuestas *sgte;
};
struct GlobalRespuestas{
  int Id;
  struct Respuestas *dato;
  struct GlobalRespuestas *sgte;
};
struct Respuestas_{
  int Id;
  float sim;
};
struct Documentos{
  int Id;
  char Documento[1500];
  struct Documentos *sgte;
};
```

```
typedef struct Documentos Docu;

typedef Docu *DOC;

struct GlobalRespuestas *InsertarGR(struct GlobalRespuestas *headGR, struct Respuestas
*dato)

{

 if(headGR==NULL)

 {

   headGR=(struct GlobalRespuestas *)malloc(sizeof(struct GlobalRespuestas));

   headGR->dato=dato;

   headGR->sgte=NULL;

 }

 else headGR->sgte=InsertarGR(headGR->sgte,dato);

 return(headGR);

}

DOC InsertarD(DOC head, char Documento[], int Id)

{

 if(head==NULL)

 {

   head=(DOC)malloc(sizeof(Docu));

   head->Id=Id;

   strcpy(head->Documento,Documento);

   head->sgte=NULL;

 }
```

```
 else head->sgte=InsertarD(head->sgte,Documento,Id);
 return(head);
}
struct Respuestas *Insertar(struct Respuestas *head, int Id, float sim)
{
 if(head==NULL)
 {
  head=(struct Respuestas *)malloc(sizeof(struct Respuestas));
  head->Id=Id;
  head->sim=sim;
  head->sgte=NULL;
 }
 else head->sgte=Insertar(head->sgte,Id,sim);
 return(head);
}
int ContarPalabras(char cadena[])
{
 int i,cont,pal;
 for(i=0,cont=0,pal=0;cadena[i]!='\0';i++)
  if(cadena[i]!=' ')
    pal++;
  else if(pal!=0)
```

```
   {
    cont++;
    pal=0;
   }
   else continue;
   if(pal!=0)
    return(cont+1);
   else return(cont);
}
void Copiar_iesimo(int inicio,int largo,char original[],char destino[])
{
 int i,j;
 for(i=1,j=0;i<=largo;inicio++,i++,j++)
   destino[j]=original[inicio];
 destino[j]='\0';
}
void SacarCaracteresNoValidos(char cadena[])
{
   int i;
   for(i=0; cadena[i]!='\0';i++)
      if(cadena[i]=='(' || cadena[i]==')' || cadena[i]=='?' || cadena[i]=='-' || cadena[i]==';' ||
cadena[i]==',' || cadena[i]=='"' || cadena[i]==':' || cadena[i]=='*' || cadena[i]=='+' ||
cadena[i]=='/' || cadena[i]==9)
```

```
    cadena[i]=' ';
  else if(cadena[i]=='.')
    cadena[i]=' ';
    else continue;
for(i=0; cadena[i]!='\0';i++)
  if(cadena[i]=='A')
    cadena[i]='a';
  else if(cadena[i]=='B')
      cadena[i]='b';
    else if(cadena[i]=='C')
        cadena[i]='c';
      else if(cadena[i]=='D')
          cadena[i]='d';
        else if(cadena[i]=='E')
            cadena[i]='e';
          else if(cadena[i]=='F')
              cadena[i]='f';
            else if(cadena[i]=='G')
                cadena[i]='g';
              else if(cadena[i]=='H')
                  cadena[i]='h';
                else if(cadena[i]=='I')
```

```
                    cadena[i]='i';
                else if(cadena[i]=='J')
                    cadena[i]='j';
                else continue;
for(i=0; cadena[i]!='\0';i++)
 if(cadena[i]=='K')
   cadena[i]='k';
 else if(cadena[i]=='L')
     cadena[i]='l';
   else if(cadena[i]=='M')
      cadena[i]='m';
     else if(cadena[i]=='N')
         cadena[i]='n';
       else if(cadena[i]=='O')
           cadena[i]='o';
         else if(cadena[i]=='P')
             cadena[i]='p';
           else if(cadena[i]=='Q')
               cadena[i]='q';
           else if(cadena[i]=='R')
               cadena[i]='r';
             else if(cadena[i]=='T')
```

```
                cadena[i]='t';
              else if(cadena[i]=='S')
                  cadena[i]='s';
               else if(cadena[i]=='U')
                   cadena[i]='u';
                  else if(cadena[i]=='V')
                      cadena[i]='v';
                     else if(cadena[i]=='W')
                         cadena[i]='w';
                        else if(cadena[i]=='X')
                            cadena[i]='x';
                           else if(cadena[i]=='Y')
                               cadena[i]='y';
                              else if(cadena[i]=='Z')
                                  cadena[i]='z';
                                 else continue;
}
int CaracterValido(char cadena[],int i)
{
 while(cadena[i]!='\0')
 {
  if(cadena[i]!=' ')
```

```
    return(i);
   else i=i+1;
 }
   return(-1); // son puros espacios en blancos
}
void SacarEspaciosS(char cadena[])
{
 int i,j,k;
 i=0;
 while(cadena[i]!='\0')
 {
  if(cadena[i]==' ' || cadena[i]==9)
  {
   if(cadena[i]==' ' && cadena[i+1]==' ' && cadena[i+1]!='\0')
   {
        k=CaracterValido(cadena,i); // si es -1 entonces son puros espacios los faltantes
        if(k!=-1)
        {
         for(j=i+1;cadena[k]!='\0';j++,k++)
         {
          cadena[j]=cadena[k];
         }
```

```
            cadena[j]='\0';
            i=i+1;
          }
          else cadena[i]='\0';
        }
        else  i=i+1;
      }
      else  i=i+1;
   }
}
int CompararCadenas(char cad1[],char cad2[])
{
 int i,largo1,largo2,stop;
 largo1=strlen(cad1);
 largo2=strlen(cad2);
 if(largo1==largo2)
   for(i=0, stop=1; cad1[i]!='\0' && stop==1;i++)
     if(cad1[i]!=cad2[i])
       stop=0;
     else continue;
 else return(0);
 return(stop);
```

```
}
int Buscar(char cadena[],char frase[])
{
 char Aux[30];
 int i,largo,largo2,total;
 largo=strlen(cadena);
 largo2=strlen(frase);
  for(i=0;frase[i]!='\0' && ((i+largo)<=largo2);i++)//aqui estaba el error, que no consideraba el ultimo termino del documento
  if(frase[i]==cadena[0])
  {
   Copiar_iesimo(i,largo,frase,Aux);
   total=i+largo;
   if(CompararCadenas(Aux,cadena)==1)
   {
    return(1);
   }
   else continue;
  }
  else continue;
  return(0);
}
PQ2 Insertar_Q(PQ2 head,char stopW[], int Id)
```

```
{
 if(head==NULL)
 {
  head=(PQ2)malloc(sizeof(SQ2));
  head->sgte=NULL;
  head->Id=Id;
  strcpy(head->FQuery,stopW);
 }
 else head->sgte= Insertar_Q(head->sgte,stopW,Id);
 return(head);
}
SP Insertar_SW(SP headSW,char stopW[])
{
 if(headSW==NULL)
 {
  headSW=(SP)malloc(sizeof(STOPW));
  headSW->sgte=NULL;
  strcpy(headSW->stopW,stopW);
 }
 else headSW->sgte= Insertar_SW(headSW->sgte,stopW);
 return(headSW);
}
```

```
int esSubcadena(char Query[],int i,int largo, SP headSW)
{
 int LARGO;
 LARGO=strlen(Query);
 if(i!=0 && Query[i-1]!=' ')
  return(1);
 else if(Query[i+largo]!=' ' && i+largo<LARGO && Query[i+largo]!='\0')
  return(1);
 else return(0);
}
SP Sacar(char Query[],SP headSW)
{
 char Aux[50];
 int largo,i,j,k;
 if(headSW!=NULL)
 {
  largo=strlen(headSW->stopW);
  for(i=0; Query[i]!='\0';i++)
    if(Query[i]==headSW->stopW[0] && !esSubcadena(Query,i,largo,headSW))
    {
      Copiar_iesimo(i,largo,Query,Aux);
      if(!strcmp(headSW->stopW,Aux))
```

```
{
 if(Query[i+largo]!='\0')
 {
  if(Query[i+largo]=='0')
  {
   k=i+largo+1;
   j=i;
   while(Query[k]!='\0')
   {
    Query[j]=Query[k];
    j++;
    k++;
   }
   Query[j]='\0';
  }
  else{
   k=i+largo;
   j=i;
   while(Query[k]!='\0')
   {
    Query[j]=Query[k];
    j++;
```

```
            k++;
           }
           Query[j]='\0';
          }
        }
         else Query[i]='\0';
       }
       else continue;
     }
     else continue;
     headSW->sgte=Sacar(Query,headSW->sgte);
  }
  else return(NULL);
  return(headSW);
}
DOC SacarStopWords_D(DOC head, SP headSW)
{
  char Documento[1500];
  int i,j;
  if(head!=NULL)
  {
    strcpy(Documento, head->Documento);
```

```
   headSW=Sacar(Documento, headSW);
   SacarEspaciosS(Documento);
   if(Documento[0]==' ')
   {
    for(i=0, j=i+1;Documento[j]!='\0';i++,j++)
      Documento[i]=Documento[j];
    Documento[i]='\0';
   }
   strcpy( head->Documento, Documento);
   head->sgte=SacarStopWords_D(head->sgte, headSW);
  }
  else return(NULL);
  return(head);
}
PQ2 SacarStopWords(PQ2 head, SP headSW)
{
  char Query[1000];
  int i,j;
  if(head!=NULL)
  {
   strcpy(Query, head->FQuery);
   headSW=Sacar(Query, headSW);
```

```
   SacarEspaciosS(Query);
   if(Query[0]==' ')
   {
    for(i=0, j=i+1;Query[j]!='\0';i++,j++)
      Query[i]=Query[j];
    Query[i]='\0';
   }
   strcpy( head->FQuery, Query);
   head->sgte=SacarStopWords(head->sgte, headSW);
  }
  else return(NULL);
  return(head);
}
void  LLenarQuery(char cadena[],char Query[])
{
  int i,stop=0,j;
  for(i=0; Query[i]!='\0' && stop==0;i++)
   if(Query[i]=='0')
   {
    j=i;
    stop=1;
   }
```

```
   else continue;
   Query[j]=' ';
   j=i;
   for(i=0; cadena[i]!='\0' && Query[j]!='\0'; i++, j++)
    Query[j]=cadena[i];
}
int EsCadenaVacia(char cadena[], int largo)
{
 int i;
 for(i=0; i<largo;i++)
   if(cadena[i]!=' ')
    return(1);
   else continue;
 return(0);
}
void ImprimirQ2(PQ2 head)
{
 if(head!=NULL)
 {
   printf("Query->FQuery:%s, Query->Id:%d\n",head->FQuery,head->Id);
   printf("\n");
   ImprimirQ2(head->sgte);
```

```
 }
 else printf("NULL\n");
}
void imprimirSW(SP headSW)
{
 if(headSW!=NULL)
 {
   printf("headSW->stopW:%s \n", headSW->stopW);
   imprimirSW(headSW->sgte);
 }
 else printf("headSW->stopW: NULL \n");
}
int ContarSW(SP headSW, int i)
{
 if(headSW!=NULL)
   return(ContarSW(headSW->sgte,i+1));
 else return(i);
}
int NQueries(PQ2 headQ, int cont)
{
 if(headQ!=NULL)
   return(NQueries(headQ->sgte, cont+1));
```

```
 else return(cont);
}
void Copiar(struct Respuestas *head,struct Respuestas_ *Arr,int inic,int largo)
{
 if(head!=NULL && inic<largo)
 {
  Arr[inic].Id=head->Id;
  Arr[inic].sim=head->sim;
  Copiar(head->sgte,Arr,inic+1,largo);
 }
}
int LargoRespuestas(struct Respuestas *head, int largo)
{
 if(head!=NULL)
  return(LargoRespuestas(head->sgte, largo+1));
 else return(largo);
}
struct Respuestas *Ordenar(struct Respuestas *head)
{
 int largo,i,j,k;
 struct Respuestas_ *Arr,Aux;
 struct Respuestas *head2=NULL;
```

```
 largo=LargoRespuestas(head,0);
 Arr=(struct Respuestas_ *)malloc(largo*sizeof(struct Respuestas_ ));
 Copiar(head,Arr,0,largo);
 for(j=0;j<largo;j++)
 {
  Aux=Arr[j];
  for(i=j+1,k=j;i<largo;i++)
  {
   if(Arr[i].sim > Aux.sim)
   {
    Aux=Arr[i];
    k=i;
   }
   else continue;
  }
  Arr[k]=Arr[j];
  Arr[j]=Aux;
 }
 for(i=0;i<largo;i++)
  head2=Insertar(head2,Arr[i].Id,Arr[i].sim);
 return(head2);
}
```

```
int ContarQDValidos(struct Terminos **MatrizT, int fil, int col)
{
 int i,cont;
 for(i=1,cont=0;i<fil;i++)
  if(MatrizT[i][0].esta==1)
  {
   cont++;
  }
  else continue;
 return(cont);
}
float Similaridad(struct Terminos **MatrizT, int NT, int largo,int ID1,int ID2)
{
 float s,s2,s3,sim;
 int j;
 for(j=1,s=0.0,s2=0.0,s3=0.0;j<NT;j++)
 {
  s=s+MatrizT[ID1][j].log*MatrizT[ID2][j].log;
  s2=s2+MatrizT[ID1][j].log*MatrizT[ID1][j].log;
  s3=s3+MatrizT[ID2][j].log*MatrizT[ID2][j].log;
 }
 if(s2!=0.0 && s3!=0.0)
```

```
   sim=s/sqrt(s2*s3);
  else sim=0.0;
  return(sim);
}
int ContarTerminosCol(struct Terminos **MatrizT,int col, int NT, int largo)
{
  int i,cont;
  for(cont=0,i=1;i<=largo;i++)
  {
    if(MatrizT[i][col].esta>=1)
      cont++;
    else continue;
  }
  return(cont);
}
void LLenarLDF(struct Terminos **MatrizT,int col,int NT, int largo,float ldf)
{
  int i;
  for(i=1;i<largo;i++)
  {
    if(MatrizT[i][col].esta>=1)
    {
```

```
    MatrizT[i][col].log=ldf*(float) MatrizT[i][col].esta;
   }
   else {
    MatrizT[i][col].log=0;
   }
  }
}
PQ2 Esta4(struct Terminos **MatrizT,int Nterm, PQ2 headQ, int actual)
{
 int j,i,cont,largo;
 char Aux[100];
 if(headQ!=NULL)
 {
  for(j=1;j< Nterm;j++)
  {
   if(Buscar(MatrizT[0][j].Termino,headQ->FQuery)==1)
   {
    largo=strlen(MatrizT[0][j].Termino);
    for(i=0,cont=0;headQ->FQuery[i]!='\0';i++)
     if(MatrizT[0][j].Termino[0]==headQ->FQuery[i])
     {
      Copiar_iesimo(i,largo,headQ->FQuery,Aux);
```

```
          if(CompararCadenas(Aux,MatrizT[0][j].Termino)==1)
            cont++;
          else continue;
        }
        else continue;
       MatrizT[actual][j].esta=cont;
       MatrizT[actual][0].esta=1;
     }
     else continue;
   }
   headQ->sgte=Esta4(MatrizT,Nterm,headQ->sgte,actual+1);
 }
 return(headQ);
}
int NDocumentos(DOC headD, int cont)
{
 if(headD!=NULL)
   return(NDocumentos(headD->sgte, cont+1));
 else return(cont);
}
int ObtenerIDDocumentos(DOC headD,int cont, int fin,int largo)
{
```

```
 if(cont==fin && fin<=largo && headD!=NULL)
   return(headD->Id);
 else return(ObtenerIDDocumentos(headD->sgte,cont+1, fin ,largo));
}
int ObtenerIDQueries(PQ2 headQ,int cont, int fin,int largo)
{
 if(cont==fin && fin<=largo && headQ!=NULL)
   return(headQ->Id);
 else return(ObtenerIDQueries(headQ->sgte,cont+1, fin ,largo));
}
int EsCadenaVacia_D(char cadena[], int largo)
{
 int i,stop=1;
 if(largo==1 && cadena[0]==10)
   return(1);
 else for(i=0;i<largo && stop==1; i++)
 if(cadena[i]!=10)
   stop=0;
 else continue;
 return(stop);
}
void  LLenarDocumento(char cadena[],char Documento[])
```

```
{
 int i,stop=0,j;
 for(i=0; Documento[i]!='\0' && stop==0;i++)
  if(Documento[i]=='\0')
  {
   j=i;
   stop=1;
  }
  else continue;
 if(j!=0)
 {
  Documento[j]=' ';
  j=i;
 }
 for(i=0,stop=0; cadena[i]!='\0'  && Documento[j]!='\0' && stop==0; i++, j++)
 {
  if(cadena[i]!=10)
  {
   Documento[j]=cadena[i];
  }
  else stop=1;
 }
```

```
 j=j-1;
 Documento[j]=' ';
}
void ObtenerQuery(int inic,int actual,int largo, char QueryAux[],PQ2 head)
{
 if(head!=NULL && inic<=largo)
 {
  if(inic==actual)
  {
   strcpy(QueryAux, head->FQuery);
  }
  else ObtenerQuery(inic+1,actual,largo,QueryAux,head->sgte);
 }
}
void ObtenerT(int fin,int inic,SP head,char Termino[])
{
 if(head!=NULL && inic== fin)
  strcpy(Termino,head->stopW);
 else  ObtenerT(fin,inic+1, head->sgte, Termino);
}
void ColocarTerminos3(int col,struct Terminos **M,int Nterminos,SP head)
{
```

```
 char Termino[30];
 ObtenerT(col,1,head,Termino);
 strcpy(M[0][col].Termino,Termino);
}
int ContarTerminos(SP head,int cont)
{
 if(head!=NULL)
   return(ContarTerminos(head->sgte,cont+1));
 else return(cont);
}
int EstaT(SP head,char Term[])
{
 if(head!=NULL)
 {
   if(CompararCadenas(head->stopW,Term)==1)
   {
     return(1);
   }
   else return(EstaT(head->sgte,Term));
 }
 else return(0);
}
```

```
SP ColocarTerminosD(SP head,DOC headD)
{
 int i,j,largo,k;
 char Term[100],Documento[1500];
 if(headD!=NULL)
 {
  strcpy(Documento,headD->Documento);
  largo=strlen(Documento);
  for(k=0;k<=largo+3; k++)
   i=0;
  while( Documento[i]!='\0' && i<largo)
  {
   j=0;
   while(Documento[i]!='\0' &&  Documento[i]!=' ' && i<largo)
   {
    Term[j]=Documento[i];
    j++;
    i++;
   }
   Term[j]='\0';
   if(EstaT(head,Term)==0)
   {
```

```
      head=Insertar_SW(head,Term);
     }
     i=i+1;
   }
   head= ColocarTerminosD(head,headD->sgte);
 }
 return(head);
}
SP ColocarTerminos(SP head,PQ2 headQ)
{
 int i,j,largo,k;
 char Term[100],FQueryA[1000];
 if(headQ!=NULL)
 {
   strcpy(FQueryA,headQ->FQuery);
   largo=strlen(FQueryA);
   for(k=0;k<=largo+3; k++)
     i=0;
   while(FQueryA[i]!='\0' && i<largo)
   {
     j=0;
     while(FQueryA[i]!='\0' &&  FQueryA[i]!=' ' && i<largo)
```

```
    {
      Term[j]=FQueryA[i];
      j++;
      i++;
    }
    Term[j]='\0';
    if(EstaT(head,Term)==0)
    {
      head=Insertar_SW(head,Term);
    }
    i=i+1;
   }
   head= ColocarTerminos(head,headQ->sgte);
 }
 return(head);
}
struct Terminos **Obtener3( PQ2 headQ)
{
 struct Terminos **MatrizT;
 int largo,Nterminos,i,j,LAux,ID;
 float ldf,sim;
 struct Respuestas **Tabla;
```

```
SP T=NULL;
T=ColocarTerminos(T,headQ);
Nterminos=ContarTerminos(T,0);
largo=LARGOM;
largo=largo+1;
Nterminos=Nterminos+1;
MatrizT=(struct Terminos **)malloc(largo*sizeof(struct Terminos *));
for(i=0;i<largo;i++)
 MatrizT[i]=(struct Terminos *)malloc(Nterminos*sizeof(struct Terminos));
for(i=0;i<largo;i++)
 for(j=0;j<Nterminos;j++) // el 0 es por si es que no tiene ningun termino en comun
 {
  MatrizT[i][j].esta=0;
  MatrizT[i][0].ID=i+1;
  MatrizT[i][0].log=0.0;
 }
 for(j=1; j<Nterminos ;j++)// esto coloca todos los terminos de todas las queries en la lista T
  ColocarTerminos3(j,MatrizT,Nterminos,T);
 headQ=Esta4(MatrizT,Nterminos,headQ,1);
 NTERMINOS=Nterminos;
 for(j=1;j<Nterminos;j++)
 {
```

```
  LAux=ContarTerminosCol(MatrizT,j,Nterminos,largo-1);
  if(LAux!=0)
  {
   ldf=log10((largo-1)/(float)LAux);
  }
  else ldf=0.0;
  LLenarLDF(MatrizT,j,Nterminos,largo,ldf);
 }
 ID=1;
 Tabla=(struct Respuestas **)malloc(LARGOM*sizeof(struct Respuestas));
 for(i=0;i<LARGOM;i++)
  Tabla[i]=NULL;
 for(ID=0;ID<LARGOM;ID++)
 {
  for(i=1; i<largo;i++)
  {
   if(i!=(ID+1))
   {
    sim=Similaridad(MatrizT,Nterminos,largo,i,(ID+1));
   }
   else continue;
  }
```

```
    printf("dentrofor:%d\n",ID );
   }
 return MatrizT;
}
/******************************************************* Codigo AverageLink
*********************************************************/
int N;
struct iInicial{
 int i;
 struct iInicial *sgte;
};
struct jInicial{
 int j;
 struct jInicial *sgte;
};
struct Elem{
 // Vector
 int indice;
 int idDoc;
 int Topic;
 float TermDoc[2]; // char TermDoc[401]
 int MarcadoFil;
 int MarcadoCol;
```

```
  // Matriz
  float valorMatriz; //calculo de distancia
  int Marcado;
  int t;
  struct iInicial *IINICIAL;
  struct jInicial *JINICIAL;
};
struct Elem *Arr; // son los documentos, de aqu¡ formo la matriz
struct Elem **Matriz; // esta es la matriz de distancias
struct Elem **Matriz2;
struct Lista{ // esta lista contiene los de un mismo cluster elementos
  int idDoc[20];
  int t;
  float min;
  int n;
  struct iInicial *IINICIAL;
  struct jInicial *JINICIAL;
  struct Lista *sgte;
};
struct Cluster{
  struct Lista *L; //esta es la lista de los cluster
  struct Cluster *sgte;
```

```
 int n;
};
struct Lista *InsertarL(struct Lista *H, int idDoc[], int N, int n, float min,int t)
{
 int i;
 if(H==NULL)
 {
   H=(struct Lista *)malloc(sizeof(struct Lista));
   H->sgte=NULL;
   H->min=min;
   H->t=t;
   H->n=n;
   for(i=0; i<N;i++)
    if(i<n)
      H->idDoc[i]=idDoc[i];
    else H->idDoc[i]=-1;
 }
 else H->sgte=InsertarL(H->sgte,idDoc,N,n,min,t);
 return(H);
}
int LargoL(struct Lista *H,int i)
{
```

```
 if(H!=NULL)
   return(LargoL(H->sgte,i+1));
 else return(i);
}
int NC=0;
struct Cluster *InsertarC(struct Cluster *C, struct Lista *H, int n)
{
 NC++;
 if(C==NULL)
 {
   C=(struct Cluster *)malloc(sizeof(struct Cluster));
   C->L=H;
   C->n=n;
   C->sgte=NULL;
 }
 else C->sgte=InsertarC(C->sgte,H,n);
 return(C);
}
int NumeroCluster(struct Cluster *C)
{
 if(C!=NULL && C->sgte==NULL)
   return(C->n);
```

```
  else return(NumeroCluster(C->sgte));
}

int ContarLista_t(struct Lista *H,int t, int i)
{
  if(H!=NULL)
    if(H->t==t)
      return(ContarLista_t(H->sgte,t,i+1));
    else
      return(ContarLista_t(H->sgte,t,i));
  else return(i);
}
void Insertar_idDocL(struct Lista *H,int IdDoc[],int k,int t,int i)
{
  int j,l,stop=0;
  if(H!=NULL && i<=k)
  if(H->t==t )
  {
    j=0;
    while(j<20 && !stop)
    {
      if(IdDoc[j]==-1)
```

```
      stop=1;
     else j++;
   }
   for(l=0;l<20 && j<20;l++)
     if(H->idDoc[l]!=-1)
       IdDoc[j++]=H->idDoc[l];
   Insertar_idDocL(H->sgte,IdDoc,k,t,i+1);
 }
 else
   Insertar_idDocL(H->sgte,IdDoc,k,t,i);
}
int Interseccion(int H_idDoc[],int A_idDoc[],int Hn,int An)
{
 int i,j,stop=0;
 for(i=0;i<Hn && stop==0;i++)
   for(j=0;j<An && stop==0;j++)
     if(H_idDoc[i]==A_idDoc[j])
       stop=1;
     else continue;
 return(stop);
}
struct Lista *ReacomodarLista(struct Lista *Aux,struct Lista *H, int t)
```

```
{
 if(H!=NULL)
 {
   Aux= InsertarL(Aux, H->idDoc,20,H-> n, H-> min,H->t);
   return(ReacomodarLista(Aux,H->sgte,t));
 }
 else return(Aux);
}
int Esta(int IdDoc[],int idDoc,int E)
{
 int i;
 for(i=0;i<E;i++)
   if(IdDoc[i]==idDoc)
     return(1);
   else continue;
 return(0);
}
struct Lista *CrearLista2(struct Elem **M,struct Cluster *C,struct Lista *H, int N,int t, float
min)
{
 for (int i = 0; i < N; ++i)
 {
   for (int j = 0; j < N && i>j ; ++j)
```

```
{
 if (M[i][j].Marcado == 1 && H == NULL)
 {
  H=(struct Lista *)malloc(sizeof(struct Lista));
  H->t=t;
  H->min=M[i][j].valorMatriz;
  H->sgte=NULL;
  H->IINICIAL=M[i][j].IINICIAL;
  H->JINICIAL=M[i][j].JINICIAL;
  M[i][j].Marcado = 2;
 }
 else{
  if (M[i][j].Marcado == 1 && H != NULL)
  {
   C=InsertarC(C,H,1);
   H=NULL;
   H=(struct Lista *)malloc(sizeof(struct Lista));
   H->t=t;
   H->min=M[i][j].valorMatriz;
   H->IINICIAL=M[i][j].IINICIAL;
   H->JINICIAL=M[i][j].JINICIAL;
   H->sgte=NULL;
```

```
        M[i][j].Marcado = 2;
      }
    }
  }
 }
 return(H);
}

int OtroMin(struct Elem **M,int N,struct Elem *A,float min)
{
 int i,j;
 for(i=0;i<N;i++)
  for(j=0;j<N && i>j ;j++)// asumo que la distancia minima es 0
  {
    if(A[i].MarcadoFil==0 && A[j].MarcadoCol==0)
     if(M[i][j].valorMatriz==min && M[i][j].Marcado==0 && !(A[i].MarcadoFil==1 &&
A[i].MarcadoCol==1) && !(A[j].MarcadoFil==1 && A[j].MarcadoCol==1))
     {
       return(1);
     }
     else continue;
    else continue;
  }
```

```
 return(0);
}
void CambiarTL(struct Lista *H,int IdDoc,int t,int stop)
{
 int i;
 if(H!=NULL && stop==0)
 {
  for(i=0;i<20 && stop==0;i++)
    if(H->idDoc[i]!=-1 && H->idDoc[i]==IdDoc)
    {
      H->t=t;
      stop=1;
    }
    else continue;
 }
 else CambiarTL(H->sgte,IdDoc,t,stop);
}
void CambiarT(struct Cluster *C,int IdDoc,int t)
{
 if(C->sgte==NULL)
   CambiarTL(C->L, IdDoc,t,0);
 else CambiarT(C->sgte,IdDoc,t);
```

```
}
void MarcarArr(struct Elem **M,struct Elem *A,int N,float min,int t)
{
 int i,j;
 for(i=0;i<N;i++)
  for(j=0;j<N && i>j ;j++)// asumo que la distancia minima es 0
  {
   if(A[i].MarcadoFil==0 && A[j].MarcadoCol==0)
    if(M[i][j].valorMatriz==min && M[i][j].Marcado==0)
    {
     A[i].MarcadoFil=1;
     A[j].MarcadoCol=1;
     M[i][j].Marcado=1;
     M[i][j].t=t;
     A[i].t=t;
     A[j].t=t;
    }
    else continue;
   else continue;
  }
}
void juntarMatriz(struct Elem **M, struct Elem **M2,int iE,int jE,int n){
```

```
int K;
int L;
for (int i = 0; i < n; ++i)
{
 for (int j = 0; j < n && i>j; ++j)
 {
  K=i;
  L=j;
  if (i == jE+1)
  {
   K=iE;
  }
  if (i > jE+1 && i <= iE)
  {
   K=i-1;
  }
  if (j == jE+1)
  {
   L=iE;
  }
  if (j > jE+1 && j <= iE)
  {
```

```
      L=j-1;
    }
    if (K>L)
    {
      M2[i][j].valorMatriz=M[K][L].valorMatriz;
      M2[i][j].IINICIAL=M[K][L].IINICIAL;//iInicial y jInicial son para no perder la referencia de los documentos
      M2[i][j].JINICIAL=M[K][L].JINICIAL;
    }
    else{
      M2[i][j].valorMatriz=M[L][K].valorMatriz;
      M2[i][j].IINICIAL=(struct iInicial *)M[L][K].JINICIAL;//iInicial y jInicial son para no perder la referencia de los documentos
      M2[i][j].JINICIAL=(struct jInicial *)M[L][K].IINICIAL;//el i y el j estan cambiados para arreglar el problema que nos surgía al final
    }
   }
  }
}
void colocarNoMarcadosYMarcados(struct Elem **M, struct Elem **M2,int iE,int jE,int n){
  for (int i = 0; i < n; ++i)
  {
    for (int j = 0; j < n && i>j; ++j)
```

```
{
 if (i != iE && i != jE && j != jE && j != iE)//no marcados
 {
  if (i > iE && j < jE)//estos 4 if son los 4 casos en donde se podria ubicar un no marcado
  {
   M2[i-1][j].valorMatriz=M[i][j].valorMatriz;
   M2[i-1][j].IINICIAL=M[i][j].IINICIAL;
   M2[i-1][j].JINICIAL=M[i][j].JINICIAL;
  }
  if (i > iE && j > jE)
  {
   M2[i-1][j-1].valorMatriz=M[i][j].valorMatriz;
   M2[i-1][j-1].IINICIAL=M[i][j].IINICIAL;
   M2[i-1][j-1].JINICIAL=M[i][j].JINICIAL;
  }
  if (i < iE && j > jE)
  {
   M2[i][j-1].valorMatriz=M[i][j].valorMatriz;
   M2[i][j-1].IINICIAL=M[i][j].IINICIAL;
   M2[i][j-1].JINICIAL=M[i][j].JINICIAL;
  }
  if (i < iE && j < jE)
```

```
 {
   M2[i][j].valorMatriz=M[i][j].valorMatriz;
   M2[i][j].IINICIAL=M[i][j].IINICIAL;
   M2[i][j].JINICIAL=M[i][j].JINICIAL;
 }
}
else//Marcados
{
 if (!((i == jE || i == iE ) && (j == jE || j == iE)))
 {
  if (i == jE)
  {
   struct iInicial *iAux=M[i][j].IINICIAL;
   while(iAux->sgte != NULL)
   {
    iAux=iAux->sgte;
   }
   iAux->sgte=M[i+1][j].IINICIAL;
   M2[i][j].valorMatriz=(M[i][j].valorMatriz + M[i+1][j].valorMatriz) / 2;
   M2[i][j].IINICIAL=M[i][j].IINICIAL;
   M2[i][j].JINICIAL=M[i][j].JINICIAL;
  }
```

```
        if (j == jE)
        {
         struct jInicial *jAux=M[i][j].JINICIAL;
         while(jAux->sgte != NULL)
         {
          jAux=jAux->sgte;
         }
         jAux->sgte=M[i][j+1].JINICIAL;
         M2[i-1][j].valorMatriz=(M[i][j].valorMatriz + M[i][j+1].valorMatriz) / 2;
         M2[i-1][j].IINICIAL=M[i][j].IINICIAL;
         M2[i-1][j].JINICIAL=M[i][j].JINICIAL;
        }
      }
    }
  }
 }
}
int contarDocRelEnCluster(struct Cluster *CHead, int id){
 int cont = 0;
 struct Lista *l = NULL;
 while(CHead != NULL){
  l = CHead->L;
```

```
  while(l != NULL){
   struct iInicial *auxI=l->IINICIAL;
   struct jInicial *auxJ=l->JINICIAL;
   while(auxI != NULL){
    if ((auxI->i + 1) == id)
    {
     cont++;
    }
    auxI=auxI->sgte;
   }
   while(auxJ != NULL){
    if ((auxJ->j + 1) == id)
    {
     cont++;
    }
    auxJ=auxJ->sgte;
   }
   l=l->sgte;
  }
  CHead=CHead->sgte;
 }
 return(cont);
```

```
}
/****************************************************** Codigo AverageLink Fin
*******************************************************/
int main(int argc, char *argv[])
{
 FILE *archivo,*archivo2;
 char cadena[1000],Query[1000],Query2[1000],IDQuery[6],Band[3],Aux[5],Band2[3];
 char stopW[25];
 int i, parar,largo,k,stop,j,cont=1;
 SP headSW=NULL;
 PQ2 headQ=NULL;
 // estas definiciones son de los documentos
 char Documento[1500],IDDocument[8],Band3[3],Band4[3],ID[8];
 struct Terminos **MatrizF=NULL;
 for(i=0;i<999;i++)
  Query[i]='0';
 Query[i]='\0';
 for(i=0;i<1499;i++)
  Documento[i]='0';
 Documento[i]='\0';
 Band[0]='.',Band[1]='I', Band[2]='\0';
 Band2[0]='.',Band2[1]='T', Band[2]='\0';
 Band3[0]='.', Band3[1]='W',Band3[2]='\0';
```

```
 Band4[0]='.', Band4[1]='B',Band4[2]='\0';
 archivo=fopen("documentos.txt","r");//nombre del archivo de texto donde estan los documentos
 if(archivo!=NULL)
 {
  while (!feof(archivo))
  {
   parar=0;
   fgets(IDQuery,6,archivo);
   Copiar_iesimo(0,2,IDQuery,Aux);
   if(!strcmp(Band,Aux))
   {
    largo=strlen(IDDocument);
    Copiar_iesimo(3,largo-1,IDDocument,ID);
    fgets(cadena,1000,archivo);
    if(EsCadenaVacia_D(cadena,largo)==1)
     fgets(cadena,1000,archivo);
    if((cadena[0]!='.' && cadena[1]!='W'))
    {
     largo=strlen(cadena);
     if(cadena[largo-2]!='.' && !(cadena[0]=='.' && cadena[1]=='W'))
      LLenarQuery(cadena,Query);
     else if((cadena[largo-2]=='.'))
```

```
 {
  LLenarQuery(cadena,Query);
  parar=1;
 }
 else continue;
}
while(!feof(archivo) && parar==0)
{
 fgets(cadena,1000,archivo);
 largo=strlen(cadena);
 if(cadena[0]=='.' && cadena[1]=='N' && largo>=2)
  parar=1;
 else{
  if(cadena[largo-2]!='.' && !(cadena[0]=='.' && cadena[1]=='W'))
   LLenarQuery(cadena,Query);
  else if((cadena[largo-2]=='.'))
  {
   LLenarQuery(cadena,Query);
   parar=1;
  }
  else continue;
 }
```

```
  }// while
  for(i=1,stop=0;i<999 && stop==0;i++)
   if (Query[i]=='0')
   {
    Query[i]='\0';
    stop=1;
   }
   else continue;
   for(i=1,j=0; Query[i]!='\0';i++,j++)
    if(Query[i]==10)
      Query2[j]=' ';
    else Query2[j]=Query[i];
   Query2[j]='\0';
   SacarCaracteresNoValidos(Query2);
   headQ=Insertar_Q(headQ,Query2,cont);
   cont=cont+1;
   for(i=0;i<999;i++)
    Query[i]='0';
   Query[i]='\0';
 }// if strcmp
 else continue;
}// while
```

```
 }// if archivo
 fclose(archivo);
 printf("Comenzando a leer los stopwords\n");
 archivo2=fopen("stopwords.txt","r");//archivo de stopwords en nuestro caso no son
necesarias, pues los terminos son creados con letras aleatorias
 if(archivo2!=NULL)
 {
  while (!feof(archivo2))
  {
   fgets(stopW,25,archivo2);
   largo=strlen(stopW);
   stopW[largo-2]='\0';
   headSW=Insertar_SW(headSW,stopW);
  }// while archivo2
  fclose(archivo2);
 }  // if archivo2
 headQ=SacarStopWords(headQ, headSW);
 MatrizF = Obtener3(headQ);
/****************************************codigo del main Average
Link*********************************************/
 int N,l,n,t;
 float min,temporal;
 N=LARGOM;
```

```
 temporal=0;
 struct Elem **MatrizSim;
 MatrizSim=(struct Elem **)malloc((N+1)*sizeof(struct Elem));
 for(i=0;i<(N+1) ;i++)
  MatrizSim[i]=(struct Elem*)malloc((N+1)*sizeof(struct Elem));
 for ( int i1 = 1; i1 <=N; ++i1)
 {
  for ( int i2 = 1; i2 <=N; ++i2)
  {
   if ( i1==i2)
   {
    MatrizSim[i1][i2].valorMatriz=1;
   }
   else{
    for (int j = 1; j <= NTERMINOS; ++j)
    {
     temporal+=(((struct Terminos **)MatrizF)[i1][j].log * ((struct Terminos
**)MatrizF)[i2][j].log);
    }
    MatrizSim[i1][i2].valorMatriz=temporal;
    temporal=0;
   }
  }
```

```
  printf("dentro de Average Link 2\n");
 }
 free(MatrizF);
 printf("dentro de Average Link \n");//estos printf(dentro de Complete Link los dejo porque,
si no imprimo nada al correr en el servidor a veces se desconecta
 struct Lista *LHead=NULL;
 struct Cluster *CHead=NULL;
 Arr=(struct Elem*)malloc(N*sizeof(struct Elem));
 // Inicializo los valores del vector Fila Columna en 0
 for(i=0;i<N;i++)
 {
  Arr[i].MarcadoFil=0;
  Arr[i].MarcadoCol=0;
  Arr[i].t=0;
 }
 // memoria para la matriz
 Matriz=(struct Elem **)malloc(N*sizeof(struct Elem));
 for(i=0;i<N ;i++)
  Matriz[i]=(struct Elem*)malloc(N*sizeof(struct Elem));
 Matriz2=(struct Elem **)malloc(N*sizeof(struct Elem));
 for(i=0;i<N ;i++)
  Matriz2[i]=(struct Elem*)malloc(N*sizeof(struct Elem));
 for(i=0;i<N;i++)
```

```
  for(j=0;j<N && i>j; j++)
  {
   struct iInicial *iAux=NULL;
   struct jInicial *jAux=NULL;
   iAux=(struct iInicial *)malloc(sizeof(struct iInicial));
   jAux=(struct jInicial *)malloc(sizeof(struct jInicial));
   iAux->i = i;
   iAux->sgte = NULL;
   jAux->j = j;
   jAux->sgte = NULL;
   Matriz[i][j].valorMatriz =MatrizSim[i+1][j+1].valorMatriz; //asignando valores de
similaridad a la matriz
   Matriz[i][j].Marcado=0;
   Matriz[i][j].IINICIAL=iAux;
   Matriz[i][j].JINICIAL=jAux;
  }
 free(MatrizSim);
 for(i=0;i<N;i++)//inicializando matriz2
  for(j=0;j<N && i>j; j++)
  {
   Matriz2[i][j].valorMatriz=0;
   Matriz2[i][j].Marcado=0;
  }
```

```
 n=N;
 l=-1;
 k=-1;
 t=1;
 min=-1;
 while(1)
 {
  l=-1;
  min=-1;
  for(i=0;i<n;i++)
   for(j=0;j<n && i>j ;j++)// asumo que la distancia minima es 0
   {
    if(Matriz[i][j].valorMatriz > min && Matriz[i][j].valorMatriz > 0)//busco el mayor valor de similaridad
     {
      l=i;
      k=j;
      min=Matriz[i][j].valorMatriz;
     }
     else continue;
   }
  if(l==-1){//si l es igual a -1 quiere decir que no encontro ningun valor > 0 valido en la matriz para seguir con el algoritmo
```

```
    break;
   }
   else if(l!=-1)
   {
    if(k-l != 1 && k-l != -1){//si esto ocurre tengo que juntar la j con la i de la matriz
     juntarMatriz(Matriz,Matriz2,l,k,n);
     for (i = 0; i < n; ++i)//copiando archivos de la matriz2 a la matriz principal
     {
      for (j = 0; j < n && i>j; ++j)
      {
       Matriz[i][j].valorMatriz=Matriz2[i][j].valorMatriz;
       Matriz[i][j].IINICIAL=Matriz2[i][j].IINICIAL;//iInicial y jInicial son para no perder la
referencia de los documentos
       Matriz[i][j].JINICIAL=Matriz2[i][j].JINICIAL;
      }
     }
     continue;
    }
    if(Arr[l].MarcadoFil==0)
    {
     Arr[l].MarcadoFil=1;
     Arr[l].t=t;
    }
```

```
    if(Arr[k].MarcadoCol==0)
    {
     Arr[k].MarcadoCol=1;
     Arr[k].t=t;
    }
    Matriz[l][k].Marcado=1;
    Matriz[l][k].t=t;
    LHead=NULL;
    LHead=CrearLista2(Matriz,CHead,LHead, n, t, min);//agrego el min a la lista
    if (LHead != NULL)
    {
     CHead=InsertarC(CHead, LHead,1);// agrego el cluster a la lista de clusteres
    }
    colocarNoMarcadosYMarcados(Matriz,Matriz2,l,k,n);
    for(i = 0; i < n-1; ++i)//copiando archivos de la matriz2 a la matriz principal
    {
     for(j = 0; j < n-1 && i>j; ++j)
     {
       Matriz[i][j].valorMatriz=Matriz2[i][j].valorMatriz;
       Matriz[i][j].IINICIAL=Matriz2[i][j].IINICIAL;//iInicial y jInicial son para no perder la
referencia de los documentos
       Matriz[i][j].JINICIAL=Matriz2[i][j].JINICIAL;
     }
```

```
    }
    n--;
    for(i = 0; i < n; ++i)
    {
     for(j = 0; j < n && i>j; ++j)
     {
       struct iInicial *auxIA=Matriz[i][j].IINICIAL;
       struct jInicial *auxJA=Matriz[i][j].JINICIAL;
     }
    }
   }
 }//while
/****************************************codigo del main Average Link
Fin*****************************************/
 struct Respuestas *docRAntiguos = NULL;
 struct Respuestas *docRAntiguosAux = NULL;
 FILE *file;
 file = fopen("documentosR.txt","r");// archivo que contiene las id de los documentos
relevantes
 if( file == NULL)
 {
  printf("Error al abrir archivo");
 }
```

```
 while( !feof(file))//proceso de lectura del archivo
 {
  if (docRAntiguos == NULL)
  {
   docRAntiguos = (struct Respuestas*)malloc(sizeof(struct Respuestas));
   docRAntiguos->sgte = NULL;
   fscanf(file, "%d ",&docRAntiguos->Id);
  }
  else{
   docRAntiguosAux = docRAntiguos;
   while(docRAntiguosAux->sgte != NULL){
    docRAntiguosAux = docRAntiguosAux->sgte;
   }
   docRAntiguosAux->sgte = (struct Respuestas*)malloc(sizeof(struct Respuestas));
   docRAntiguosAux->sgte->sgte = NULL;
   fscanf(file, "%d ",&docRAntiguosAux->sgte->Id);//almaceno la id de los documentos
relevantes
  }
 }
 fclose(file);
 struct Cluster *CHeadI=NULL;
 CHeadI = CHead;
 int contRel = 0;
```

```
 while(docRAntiguos != NULL){//aqui cuenta los documentos relevantes visitados al recorrer el cluster construido con el algoritmo Average Link
   contRel = contRel + contarDocRelEnCluster(CHeadI,docRAntiguos->Id);
   docRAntiguos = docRAntiguos->sgte;
 }

/*while(CHead != NULL){
 printf("tamano cluster=%d\n",CHead->n );
 while(CHead->L != NULL){
   printf("lista_min=%f\n",CHead->L->min );
   printf("lista_t=%d\n",CHead->L->t );
   struct iInicial *auxI=CHead->L->IINICIAL;
   struct jInicial *auxJ=CHead->L->JINICIAL;
   printf("\nposicion i:");
   while(auxI != NULL){
    printf("%d->",auxI->i);
    auxI=auxI->sgte;
   }
   printf("\nposicion j:");
   while(auxJ != NULL){
    printf("%d->",auxJ->j);
    auxJ=auxJ->sgte;
   }
```

```
   printf("\n");
   CHead->L=CHead->L->sgte;
  }

  CHead=CHead->sgte;
  }*/
  printf("La cantidad de documentos relevantes visitados al recorrer los clusteres
son:%d\n",contRel);
  return 0;
}
```

Codigo fuente de algoritmo algoritmo Ward

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<malloc.h>
#include<stdbool.h>
#include<string.h>
/*****************************************************************************
 Autores: Delia Moncada - Frederick Lara.
 La forma de compilar este programa es: gcc Ward.c -o ward -lm
 La forma de ejecutar este programa es: ./ward
*****************************************************************************/
struct Distancia{
        int id;
        float distancia;
        bool activo;
        bool considerado;
};
struct Lista{ // esta lista contiene los de un mismo cluster elementos
        struct Distancia dist;
        struct Lista *sgte;
```

```
};
struct Cluster{
   struct Lista *L; //esta es la lista de los cluster
   struct Cluster *sgte;
   int n;
 };
struct Cluster *InsertarC(struct Cluster *C, struct Lista *H)
{
   if(C==NULL)
   {
      C=(struct Cluster *)malloc(sizeof(struct Cluster));
      C->L=H;
      C->sgte=NULL;
   }
   else C->sgte=InsertarC(C->sgte,H);
            return(C);
}
int N=5;
int main(int argc, char *argv[])
{
            int k=N;
            float z,zAux;
```

```
int i=0,j=0,iMenor=k,jMenor=k;
struct Cluster *cluster = NULL;
struct Distancia distancias[N];
distancias[0].id=1;
distancias[1].id=2;
distancias[2].id=3;
distancias[3].id=4;
distancias[4].id=5;
distancias[0].distancia=1;//estos valores de distancias son los que aparecen en el paper de Ward
distancias[1].distancia=7;
distancias[2].distancia=2;
distancias[3].distancia=9;
distancias[4].distancia=12;
distancias[0].activo=true;
distancias[1].activo=true;
distancias[2].activo=true;
distancias[3].activo=true;
distancias[4].activo=true;
distancias[0].considerado=false;
distancias[1].considerado=false;
distancias[2].considerado=false;
distancias[3].considerado=false;
```

```
distancias[4].considerado=false;
k=N;
while(k>3){
	z=1000;
	i=-1;
	for (int l = 0; l < k; ++l)//elegir j > i
	{
		if (distancias[l].activo == true && i == -1)
		{
			i=distancias[l].id;
			continue;
		}
		if (distancias[l].activo == true && distancias[l].id > i && distancias[l].id
!= i)
		{
			j=distancias[l].id;
			break;
		}
	}
	printf("inicial i=%d,j=%d\n",i,j );
	while(true){
		printf("i=%d,j=%d\n",i,j );
```

```
                zAux = (distancias[i-1].distancia * distancias[i-1].distancia) +
(distancias[j-1].distancia * distancias[j-1].distancia) -

                (0.5 * ((distancias[i-1].distancia + distancias[j-
1].distancia)*(distancias[i-1].distancia + distancias[j-1].distancia)));

                if (zAux < z && zAux != 0)// el distinto de 0 es por si se calcula z con i y
j iguales ej: z(5,5)

                {

                        z=zAux;

                        iMenor=i;

                        jMenor=j;

                }

                if (j == N)

                {

                        if (i == N)//block 9,block 11

                        {

                                struct Lista *lista = (struct Lista *)malloc(sizeof(struct
Lista));

                                struct Lista *listaAux = (struct Lista
*)malloc(sizeof(struct Lista));

                                lista->dist=distancias[iMenor-1];

                                listaAux->dist=distancias[jMenor-1];

                                lista->sgte = listaAux;

                                listaAux->sgte = NULL;

                                cluster = InsertarC(cluster,lista);
```

```
            printf("iMenor= %d,jMenor=%d\n",iMenor,jMenor);
            distancias[jMenor-1].activo = false;
            distancias[jMenor-1].considerado = true;
            distancias[iMenor-1].considerado = true;
            k=k-1;
            break;
        }
        else{//block 10
            bool desicion=false;
            while(desicion == false){
                i++;
                if (distancias[i-1].activo == true)
                {
                    desicion = true;
                }
            }
            if (i<k)// reinicia el j cuando aumenta i
            {
                j=1;
                while(j <= i || distancias[j-1].activo == false){
                    j++;
                }
```

```
                    }
                }
            }
            else{
                bool desicion=false;
                while(desicion == false){
                    j++;
                    if (distancias[j-1].activo == true && j<=N)
                    {
                        desicion = true;
                    }
                    else{
            if (j>=N)
            {
            k=2;
            break;
            }
         }
                    }
            }
        }
    }
```

```
        for (int l = 0; l < N; ++l)
        {
                if (distancias[l].considerado == false)
                {
                        struct Lista *listaSolo = (struct Lista *)malloc(sizeof(struct Lista));
                        listaSolo->dist=distancias[l];
                        listaSolo->sgte = NULL;
                        cluster = InsertarC(cluster,listaSolo);
                }
        }
        struct Cluster *clusterImprimir = cluster;
        while(clusterImprimir != NULL){
    printf("id=");
    while(clusterImprimir->L != NULL){
      printf("%d,",clusterImprimir->L->dist.id);
      clusterImprimir->L=clusterImprimir->L->sgte;
    }
    printf("\n");
    clusterImprimir=clusterImprimir->sgte;
   }
}
```

Codigo fuente del framework

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<malloc.h>
#include<time.h>
/*******************************************************************************
 Autor: Gutierrez-Soto(2016).
 Modificado Por: Delia Moncada - Frederick Lara.
 Es importante mencionar que este framework tenia muchas mas funciones que fueron
 extraidas para que el codigo no quedara tan extenso.
 La forma de compilar este framework es: gcc Framework.c -o framework -lm
 La forma de ejecutar este framework es: ./framework
*******************************************************************************/
char
Alphabet[27]={'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z','\0'};
struct Terms{
  char term[8];
  int id;
  struct Terms *sgte;
  int topico;
```

```
 unsigned short int Marcado;
};
typedef struct Terms TERM;
typedef  TERM *TERMS;
struct TerRelevantes{
 char term[8];
 int id;
 struct TerRelevantes *sgte;
 int topico;
 unsigned short int Marcado;
};
typedef struct TerRelevantes TERMR;
typedef  TERMR *TERMSR;
struct Doc{
 signed short int TermDoc[30];
 unsigned short int idDoc;
 unsigned short int Topico;
 unsigned short int Marcado;
 int cantidadTerRelevantes;
 float porcentajeTerRelevantes;
 struct Doc *sgte;
};
```

```
typedef struct Doc DOC;
typedef DOC *DOCUMENT;
struct Query{
 unsigned short int TermDoc[8];
 int idQuery;
 int Topic;
 int Marcado;
 struct Query *sgte;
};
typedef Query LISTAQ;
typedef LISTAQ *ENLACEQ;
ENLACEQ InsertQ(ENLACEQ headQ,unsigned short int TermDoc[],int idQuery, int Topic)
{
 int i;
 if(headQ==NULL)
 {
  headQ=(ENLACEQ)malloc(sizeof(LISTAQ));
  headQ->idQuery=idQuery;
  headQ->sgte=NULL;
  headQ->Topic=Topic;
  for(i=0;i<8;i++)
   headQ->TermDoc[i]=TermDoc[i];
```

```
 }
 else headQ->sgte=InsertQ(headQ->sgte,TermDoc,idQuery,Topic);
 return(headQ);
}
struct Grafo{
 int idQuery;
 int fil;
 float Distancia;
 struct Grafo *sgte;
};
struct Past_Query{
 unsigned short int idDoc[30][2]; // la primera dimension es el idDoc, la segunda es el Topico del documento
        unsigned short int PyR_idDoc[30];// esto es lo que dice si son relevantes @30, pero se consideran los @10 primeros
 signed short int vectorZ[30];//aqui guardo lo que retorna la funcion de distribucion Z
 unsigned short int soloDRel[30];//solo documentos Relevantes
 int cantTermIgualesPorDocumento[30];
 int cantTermIguales;
 float precision;
 int contador;
        float Dis[30][2]; // las distancias de la query a los documentos, la otra dimension es si esta Marcado
```

```
 int NDoc; // esto contiene el numero de documentos cuya Dis >0, el numero es <=30
 int TipoQ;
 int Rankeado;
 int idQuery;
 int Topic;
 int Marcado;
 int NVertices;
 float Precision;
 struct Past_Query *sgte;
 struct Grafo *Lista; // esto es de las queries mas cercanas
};
typedef Past_Query PLISTAQ;
typedef PLISTAQ *PENLACEQ;
PENLACEQ InsertPQ(PENLACEQ headQ, unsigned short int idDoc[][2],unsigned short int
PyR_idDoc[],int idQuery, int Topic)
{
 int i,j;
 if(headQ==NULL)
 {
  headQ=(PENLACEQ)malloc(sizeof(PLISTAQ));
  headQ->idQuery=idQuery;
  headQ->sgte=NULL;
  headQ->Topic=Topic;
```

```
   headQ->Marcado=0;
   headQ->NVertices=0;
   headQ->Precision=0.0;
   headQ->Lista=NULL;
   headQ->NDoc=0;
   for(i=0;i<30;i++)
    for(j=0;j<2;j++)
     headQ->idDoc[i][j]=idDoc[i][j];
   for(i=0;i<30;i++)
    headQ->PyR_idDoc[i]=PyR_idDoc[i];
  }
  else headQ->sgte=InsertPQ(headQ->sgte,idDoc,PyR_idDoc, idQuery,Topic);
  return(headQ);
}
int ObtenerMarcaQ(ENLACEQ headQ,unsigned short int i,unsigned short int j)
{
  if(headQ!=NULL && i==j)
  {
   printf("Dentro de ObtenerMarcaQ , i=%d, j=%d, Marcado=%d \n",i,j,headQ->Marcado);
   return(headQ->Marcado);
  }
  else if(headQ!=NULL)
```

```
   return(ObtenerMarcaQ(headQ->sgte,i,(j+1)));
  else return(10);
}
void Marcar_Query(ENLACEQ headQ, int aux, unsigned short int TermDoc_[], int i)
{
  int j;
  if(headQ!=NULL && aux==i)
  {
    printf("Dentro de Marcar_Query \n");
    for(j=0;j<8;j++)
    TermDoc_[j]=headQ->TermDoc[j];
    headQ->Marcado=1;
  }
  else if(headQ!=NULL)
  Marcar_Query(headQ->sgte,aux,TermDoc_,(i+1));
}
void ImprimirQueries(ENLACEQ headQ)
{
  int i,stop;
  if(headQ!=NULL)
  {
    for(i=0,stop=0;i<8 && stop==0;i++)
```

```
   {
    if(headQ->TermDoc[i] !=0)
      printf(" headQ->d=%d ",headQ->TermDoc[i]);
    else stop=1;
   }
   printf("headQ->idQuery:%d ",headQ->idQuery);
   printf("headQ->Topic:%d ",headQ->Topic);
   ImprimirQueries(headQ->sgte);
 }
 else{
   printf("\n");
   printf("\n");
   printf("NULL \n");
 }
}
int ObteneridQuery(PENLACEQ headPQ,int i,int j)
{
 if(headPQ!=NULL && i==j)
 {
   return(headPQ->idQuery);
 }
 else return(ObteneridQuery(headPQ->sgte,i,(j+1)));
```

```
}
int EstaMarcadoTerm(int IdTerm,TERMS headT)
{
 if(headT!=NULL && headT->id==IdTerm)
   if(headT->Marcado==1)
     return(1);
   else return(0);
 else if(headT->sgte!=NULL)
   EstaMarcadoTerm(IdTerm, headT->sgte);
 else return(0);
}
void PonerTerminos(char Todos[8][8], char TermQuery[])
{
 int i,j,k,l=0;
 char Aux[8];
 for(i=0,j=0,k=0;TermQuery[i]!='\0';i++)
   if(TermQuery[i]!='#')
     Aux[j++]=TermQuery[i];
   else if(TermQuery[i]=='#' && j!=0)
   {
     Aux[j]='\0';
     strcpy(Todos[k++],Aux);
```

```
    l++;
    j=0;
   }
   else continue;
 for(i=l;i<8;i++)
   Todos[l][0]='\0';
}
TERMS InsertT(TERMS head, char term[],int id)
{
 if(head==NULL)
 {
   head=(TERMS)malloc(sizeof(TERM));
   head->id=id;
   head->topico=-1;
   strcpy(head->term,term);
   head->sgte=NULL;
   head->Marcado=0;
 }
 else head->sgte=InsertT(head->sgte, term,id++);
 return(head);
}
DOCUMENT InsertD(DOCUMENT head, unsigned short int TermDoc[],unsigned short int
Topico, unsigned short int idDoc)
```

```
{
 int i;
 if(head==NULL)
 {
  head=(DOCUMENT)malloc(sizeof(DOC));
  head->idDoc=idDoc;
  for(i=0;i<30;i++)
    head->TermDoc[i]=TermDoc[i];
  head->Topico=Topico;
  head->Marcado=0;
  head->sgte=NULL;
 }
 else head->sgte=InsertD(head->sgte,TermDoc,Topico,idDoc);
 return(head);
}
```

//funcion creada para crear documentos con su cantidad de terminos relevantes, es decir los coincidentes con la lista de terminos relevantes

```
DOCUMENT InsertDConTerRelevantes(DOCUMENT head, unsigned short int
TermDoc[],unsigned short int Topico, unsigned short int idDoc, int cantidadTerRelevantes, int
NTermsbyDoc)
{
 int i;
 if(head==NULL)
```

```
 {
   head=(DOCUMENT)malloc(sizeof(DOC));
   head->idDoc=idDoc;
   for(i=0;i<30;i++)
    head->TermDoc[i]=TermDoc[i];
   head->Topico=Topico;
   head->Marcado=0;
   head->cantidadTerRelevantes = cantidadTerRelevantes;
   head->porcentajeTerRelevantes = ((float)((float)cantidadTerRelevantes) /
((float)NTermsbyDoc));
   head->sgte=NULL;
 }
 else head->sgte=InsertDConTerRelevantes(head-
>sgte,TermDoc,Topico,idDoc,cantidadTerRelevantes,NTermsbyDoc);
 return(head);
}
int BuscarTERMS(TERMS head, char term[])
{
 if(head!=NULL)
   if(!strcmp(head->term,term))
            return(1);
   else return(BuscarTERMS(head->sgte,term));
 else return(0);
```

```
}
void Colocar(TERMS headT, int topico,int inic, int fin,int i)
{
 if(headT!=NULL && i>=inic && i<=fin)
 {
   headT->topico=topico;
   Colocar(headT->sgte,topico,inic,fin,(i+1));
 }
 else if(headT!=NULL)
   Colocar(headT->sgte,topico,inic,fin,(i+1));
}

TERMS ColocarTopico(int inic,int fin,TERMS headT,int topico)
{
 int i=0;
 if(headT!=NULL)
 {
   while(i<=fin)
   {
     if(i>=inic && i<=fin)
     {
       Colocar(headT,topico,inic,fin,0);
```

```
      return(headT);
    }
    i++;
   }
  }
  else return(headT);
}
TERMS GenerarTopics_(int NTermst,int NTopic,TERMS headT)
{
  int inic, fin, inter,topico;
  inter=NTermst/NTopic;
  printf(" Dentro de Generar Topicos \n");
  printf("NTermst: %d, NTopic:%d, intervalo:%d \n",NTermst,NTopic,inter);
  topico=1;
  inic=0;
  fin=inter-1;
  while( fin<=NTermst)
  {
   printf("inic: %d, fin:%d, topico:%d \n",inic,fin,topico);
   headT=ColocarTopico(inic,fin,headT,topico);
   inic=fin+1;
   fin=fin+inter;
```

```
   topico++;
  }
  return(headT);
}
TERMS GenerarTERMS(int NTermsT,TERMS headT)
{
  char Aux[8];
  int i,j,id=1,index,largo,k;
  for(i=1;i<=NTermsT;i++)
  {
    //Here, I generate the lenght of every word 3-7
    largo=(rand()%5)+3;
    for(j=0;j<largo;j++)
    {
      index=(rand()%26); // in this case is 26, because of number english letters
      Aux[j]=Alphabet[index];
    }
    Aux[j]='\0';
    if(j!=8)
    {
      for(k=j; k<8;k++)
        Aux[k]='\0';
```

```
   }
   if(BuscarTERMS(headT,Aux)==0)
   {
    headT=InsertT(headT,Aux,id++);
    for(k=0; k<8;k++)
      Aux[k]='\0';
   }
   else {i=i-1;continue;}
  }
  return(headT);
}
TERMS GenerarTERMS2(int NTermsT,TERMS headT,TERMS headT2)
{
  char Aux[8];
  int i,j,id=1,index,largo,k;
  for(i=1;i<=NTermsT;i++)
  {
   //Here, I generate the lenght of every word 3-7
   largo=(rand()%5)+3;
   for(j=0;j<largo;j++)
   {
    index=(rand()%26); // in this case is 26, because of number english letters
```

```
    Aux[j]=Alphabet[index];
   }
   Aux[j]='\0';
   if(j!=8)
   {
    for(k=j; k<8;k++)
      Aux[k]='\0';
   }
   if(BuscarTERMS(headT,Aux)==0 && BuscarTERMS(headT2,Aux)==0)
   {
    headT=InsertT(headT,Aux,id++);
    for(k=0; k<8;k++)
      Aux[k]='\0';
   }
   else{
    i=i-1;
    continue;
   }
 }
 return(headT);
}
TERMS Copiar_Lista_De(TERMS head_T,int inic, int fin,TERMS head_F, int j)
```

```
{
 if(j<=fin && head_T!=NULL)
 {
  if(j>=inic && j<fin)
  {
   head_F=InsertT(head_F,head_T->term,j+1);
   return( Copiar_Lista_De(head_T->sgte,inic,fin,head_F,j+1));
  }
 else if(j==fin)
 {
  head_F=InsertT(head_F,head_T->term,j+1);
  return(head_F);
 }
 else return( Copiar_Lista_De(head_T->sgte,inic,fin,head_F,j+1));
 }
}
int ContarDocument(DOCUMENT head_D,int i)
{
 if(head_D!=NULL)
  ContarDocument(head_D->sgte,(i+1));
 else return(i);
}
```

```
int BuscarT_D(unsigned short int TermDoc[],unsigned short int TermAux)
{
 int i,stop=0;
 for(i=0,stop=1; TermDoc[i]!=0 && stop ; i++)
 {
  if(TermDoc[i]==TermAux)
    return(1);
  else  continue;
 }
 return(0);
}
int BuscarT_D2(unsigned short int TermDoc[],unsigned short int TermAux,int cantOldTerms)
{
 int i,stop=0;
 for(i=cantOldTerms+1,stop=1; TermDoc[i]!=0 && stop ; i++)
 {
  if(TermDoc[i]==TermAux)
    return(1);
  else  continue;
 }
 return(0);
}
```

```
void AgregarTerm_Doc(unsigned short int TermDoc[],unsigned short int TermAux)
{
 int i=0;
 for(i=0; TermDoc[i]!=0 && i<40;i++);
   TermDoc[i]=TermAux;
}
float PromedioLetras(char TermDoc[], int total)
{
 int i,stop=1, Nl=0;
 for(i=0; TermDoc[i]!='\0' && stop ;i++)
            if(TermDoc[i]!='#')
              Nl++;
            else if( TermDoc[i+1]!='\0' && TermDoc[i]=='#' && TermDoc[i+1]=='#')
                        stop=0;
            else continue;
 return((float)(Nl/total));
}
/*************************** Fin Generar los documentos
************************************************/
struct Lista_{ // esta lista es un arreglo, pero que contiene los terminos
           // la primera dimension es el IdTerm,
           // la segunda cantidad de documentos que tienen el termino en el clus
 unsigned short int IdTerm[2];
```

```
 unsigned short int NDocClus; // esto contiene el numero de documentos en el cluster
 struct Lista_ *sgte;
};
struct Topicos{
 int IdTopico;
 int IdVar;
};
struct Doc_{
 unsigned short int idDoc;
 unsigned short int Topico;
 int Marcado; // Esto lo agregue al ultimo, hay que arreglar el insertar
};
struct Lista{ // esta lista contiene los k cluster
        struct Doc_ Punto; // lista de puntos
 struct Doc_ Centroide;
 struct Lista *sgte;
};
int ObtenerTopicoD(unsigned short int idDoc_, DOCUMENT headD)
{
 if(headD!=NULL && headD->idDoc==idDoc_)
   return(headD->Topico);
 else if(headD!=NULL)
```

```
   return(ObtenerTopicoD(idDoc_, headD->sgte));
  else return(0);
}
int ObtenerNumeroDocumentos_(struct Lista *Clus,int i)
{
  if(Clus!=NULL)
   return(ObtenerNumeroDocumentos_(Clus->sgte,(i+1)));
  else return(i);
}
int TermP_Doc(struct Doc_ P,unsigned short int IdTerm_,DOCUMENT headD)
{
  int i;
  if(headD!=NULL && headD->idDoc==P.idDoc)
   for(i=0; i<30 ;i++)
     if(headD->TermDoc[i]==IdTerm_)
       return(1);
     else continue;
  else if( headD!=NULL)
   return(TermP_Doc(P,IdTerm_,headD->sgte));
  else return(0);
}
struct Lista_ *InsertarTC(struct Lista_ *head,unsigned short int IdTerm_,unsigned short int
NTerm,unsigned short int NDoc)
```

```
{
 if(head==NULL)
 {
   head=(struct Lista_*)malloc(sizeof(struct Lista_));
   head->IdTerm[0]=IdTerm_;
   head->IdTerm[1]=NTerm; // la cantidad de veces que aparece el termino en los documentos
   head->NDocClus=NDoc;
   head->sgte=NULL;
 }
 else head->sgte=InsertarTC(head->sgte, IdTerm_, NTerm, NDoc);
   return(head);
}
int EstaTermino(struct Lista_ *head,unsigned short int IdTerm_)
{
 if(head!=NULL)
   if(head->IdTerm[0]==IdTerm_)
     return(1);
   else return(EstaTermino(head->sgte,IdTerm_));
 else if(head==NULL)
 return(0);
}
void ImprimirTerminoCentroide(struct Lista_ *head)
```

```
{
 if(head!=NULL)
 {
  printf(" TerminoCLuster:%d,",head->IdTerm[0]);
  ImprimirTerminoCentroide(head->sgte);
 }
 else printf("FinTermino NULL \n");
}
struct Lista_ *ActualizarTC(struct Lista_ *head,unsigned short int NTerms,unsigned short int IdTerm_,unsigned short int NDoc)
{
 if(head!=NULL && head->IdTerm[0]==IdTerm_)
 {
  head->IdTerm[1]=NTerms;
  head->NDocClus=NDoc;
 }
 else if (head!=NULL)
  head->sgte=ActualizarTC(head->sgte,NTerms,IdTerm_,NDoc);
 else return(NULL);
 return(head);
}

unsigned short int NumeroTerminos(struct Lista_ *headAux,unsigned short int TermDoc1)
```

```
{
 if(headAux!=NULL && headAux->IdTerm[0]==TermDoc1)
   return(headAux->IdTerm[1]);
 else if(headAux!=NULL)
   return(NumeroTerminos(headAux->sgte,TermDoc1));
 else return(0);
}
struct Lista_ *Actualizar(struct Lista_ *headAux,unsigned short int TermDoc1,unsigned short int s)
{
 if(headAux!=NULL && headAux->IdTerm[0]==TermDoc1)
   headAux->IdTerm[1]=s;
 else if(headAux!=NULL)
   headAux->sgte=Actualizar(headAux->sgte,TermDoc1,s);
 return(headAux);
}
void Marcar_Documento(DOCUMENT headD,unsigned short int aux, unsigned short int TermDoc_[], int i)  // debo de agregar esta funcion
{
 int j;
 if(headD!=NULL && aux==i)
 {
   for(j=0;j<30;j++)
```

```
    TermDoc_[j]=headD->TermDoc[j];
   headD->Marcado=1;
 }
 else if(headD!=NULL)
   Marcar_Documento(headD->sgte,aux,TermDoc_,(i+1));
}
void Marcar_Documento2(DOCUMENT headD,unsigned short int aux, unsigned short int
TermDoc_[], int i)  // debo de agregar esta funcion
{
 if(headD!=NULL && aux==i)
   headD->Marcado=1;
 else if(headD!=NULL)
   Marcar_Documento2(headD->sgte,aux,TermDoc_,(i+1));
}
int ObtenerIdDocClus(struct Lista *Clus, int i, int k)
{
 if(Clus!=NULL && i==k)
   return(Clus->Punto.idDoc);
 else if(Clus!=NULL)
   return(ObtenerIdDocClus(Clus->sgte, i, (k+1)));
 else return(0);
}
```

```
void Poner_T ( struct Doc_ Punto,DOCUMENT headD,int i, int k) // debo de agregar esta
funcion
{
 if(headD!=NULL && i==k)
 {
   Punto.idDoc=headD->idDoc;
   printf("Punto , idDoc:%d \n", Punto.idDoc);
   printf("headD->idDoc , idDoc:%d \n", headD->idDoc);
   Punto.Marcado=0;
 }
 else if(headD!=NULL)
   Poner_T(Punto,headD,i,k+1);
}
struct Lista *AgregarDocumento(struct Lista *head,unsigned short int IdDoc)
{
 if(head==NULL)
 {
   head=(struct Lista *)malloc(sizeof(struct Lista));
   head->Punto.idDoc=IdDoc;
   head->sgte=NULL;
 }
 else head->sgte=AgregarDocumento(head->sgte,IdDoc);
 return(head);
```

```
}
void ImprimirCentroides(struct Lista_ *Clus)
{
 if(Clus!=NULL)
 {
   printf("IdTerm: %d  ,",Clus->IdTerm[0]);
   ImprimirCentroides(Clus->sgte);
 }
 else printf("IdTerm NULL \n");
}
int Marcado_Documento(DOCUMENT headD,int  i,int j)
{
 if(headD!=NULL && i==j)
   return(headD->Marcado);
 else if(headD!=NULL)
   return(Marcado_Documento(headD->sgte,i,(j+1)));
 else return(0);
}
void ImprimirMarcaDocumento(DOCUMENT headD)
{
 if(headD!=NULL)
 {
```

```
   printf("idDoc:%d, Marca:%d \n", headD->idDoc,headD->Marcado);
   ImprimirMarcaDocumento(headD->sgte);
  }
 else printf("NULL");
}
void Imprimir_idDoc(struct Lista *Clus)
{
  if(Clus!=NULL)
  {
    printf("IdDoc: %d  ,",Clus->Punto.idDoc);
    Imprimir_idDoc(Clus->sgte);
  }
  else printf("idDoc NULL \n");
}
int ObtenerNumeroDocLista(struct Lista *head_D,int i)
{
  if(head_D!=NULL)
    return(ObtenerNumeroDocLista(head_D->sgte,(i+1)));
  else return(i);
}
void Obtener_TQuery(ENLACEQ headQ,int aux,unsigned short int TermQuery[],int i)
{
```

```
 int j;
 if(headQ!=NULL && aux==i)
 {
  for(j=0;j<8;j++)
    TermQuery[j]=headQ->TermDoc[j];

 }
 else if(headQ!=NULL)
  Obtener_TQuery(headQ->sgte,aux,TermQuery,(i+1));
}
void Obtener_TDocumento(DOCUMENT headD, int aux, unsigned short int TermDoc_[], int i)
{
 int j;
 if(headD!=NULL && aux==i)
 {
  for(j=0;j<30;j++)
    TermDoc_[j]=headD->TermDoc[j];
 }
 else if(headD!=NULL)
  Obtener_TDocumento(headD->sgte,aux,TermDoc_,(i+1));
}
int NTermCluster(struct Lista_ *head, int i)
```

```
{
  if(head!=NULL)
    return(NTermCluster(head->sgte, (i+1)));
  else return(i);
}
struct Query_Doc{
  int idQuery;
  int idDoc;
  float Dis;
  int Marcado;
  struct Query_Doc *sgte;
};
struct idQueriesDoc{
  int idQuery;
  struct Query_Doc *QD;
  struct idQueriesDoc *sgte;
};
struct Zeta{
  // este es un caso particular de la distribucion de Pareto y Skewed
  float L_izq;
  float L_der;
        float ValorDis; // este valor es un real entre 0-1
```

```
 int Posicion;
 float Uno_div_P;
};
struct Query_Doc *InsertarQD(struct Query_Doc * head,int idQuery,int idDoc,float dis)
{
 if(head==NULL)
 {
  head=(struct Query_Doc *)malloc(sizeof(struct Query_Doc));
  head->idQuery=idQuery;
  head->idDoc=idDoc;
  head->Dis=dis;
  head->Marcado=0;
  head->sgte=NULL;
 }
 else head->sgte=InsertarQD(head->sgte,idQuery,idDoc,dis);
  return(head);
}
float DISTANCIA(int idQuery,ENLACEQ headQ, int idDoc,DOCUMENT headD,struct Lista_
*head,struct Lista *D)
{
 int stop,k,l;
 unsigned short int TermDoc1[30],TermQuery[8],NTermC;
 float suma=0.0,f;
```

```
for(k=0;k<30;k++)
  TermDoc1[k]=0;
for(k=0;k<8;k++)
  TermQuery[k]=0;
Obtener_TDocumento(headD,idDoc,TermDoc1,1) ;
Obtener_TQuery(headQ,idQuery,TermQuery,1);
for(k=0,stop=0;k<30 && stop==0;k++)
  for(l=0;l<8;l++)
    if(TermDoc1[k]==TermQuery[l] && TermQuery[l]!=0 && TermDoc1[k]!=0)
      stop=1;
    else continue;
if(stop==1)
{
  NTermC=NTermCluster(head, 0);
  for(k=0,stop=0,f=1.0;k<30 ;k++)
    if(EstaTermino(head,TermDoc1[k])==0)
      continue;
    else
      suma=suma+1.0;
  return(suma/sqrt(NTermC*NTermC));
}
else return(0.0);
```

```
}
void Imprimir_QueriesDoc( struct Query_Doc *head)
{
 if(head!=NULL)
 {
   printf(" idQuery: %d, idDoc:%d, Dis:%f \n",head->idQuery,head->idDoc,head->Dis);
   Imprimir_QueriesDoc(head->sgte);
 }
 else printf("NULL");
}
int ObteneridDoc(struct Query_Doc *head,int i,int j)
{
 if(head!=NULL && (i==j))
   return(head->idDoc);
 else return(ObteneridDoc(head->sgte,i,(j+1)));
}
float ObtenerDistancia(struct Query_Doc *head,int idDocF)
{
 if(head!=NULL && head->idDoc==idDocF)
   return(head->Dis);
 else return(ObtenerDistancia(head->sgte,idDocF));
}
```

```
struct Query_Doc *MarcarQD(struct Query_Doc *head,int idQuery,int idDocF)
{
 if(head!=NULL && head->idQuery==idQuery && head->idDoc==idDocF)
   head->Marcado=1;
 else if(head!=NULL)
   head->sgte=MarcarQD(head->sgte,idQuery,idDocF);
 return(head);
}
int ObtenerMayorDoc(Query_Doc *head,float max, int idDoc)
{
 if(head!=NULL && head->Marcado==0 && head->Dis>max)
   return(ObtenerMayorDoc(head->sgte,head->Dis,head->idDoc) );
 else if(head!=NULL)
   return(ObtenerMayorDoc(head->sgte,max,idDoc));
 else return(idDoc);
}
struct idQueriesDoc *InsertarGlobalQD(struct idQueriesDoc *headQD,int idQuery,DOCUMENT
headD, struct Lista_ *T,struct Lista *D,ENLACEQ headQ)
{
 int NDoc,i,idDoc,cont,idDocAux;
 float dis,disAux;
 struct Query_Doc *head=NULL;
 struct Query_Doc *headF=NULL;
```

```
 if(headQD==NULL)
 {
  NDoc=ObtenerNumeroDocumentos_(D,0);
  for(i=1,cont=0;i<=NDoc;i++)
  {
   idDoc= ObtenerIdDocClus(D, i, 1);
   dis=DISTANCIA(idQuery,headQ,idDoc,headD,T,D);
   if(dis>0)
   {
    head=InsertarQD(head,idQuery,idDoc, dis);
    cont++;
   }
   else continue;
  }// for i<=NDoc
  printf(" El numero de documentos que hicieron match con la idQuery:%d, son
:%d\n",idQuery, cont);
  // Elegir los 50 más relevantes, crear una nueva lista
  for(i=1;i<=cont;i++)
  {
   idDocAux=ObtenerMayorDoc(head,0,0);
   disAux=ObtenerDistancia(head,idDocAux);
   head=MarcarQD(head,idQuery,idDocAux);
   headF=InsertarQD(headF,idQuery,idDocAux, disAux);
```

```
   }
   headQD=(struct idQueriesDoc *)malloc(sizeof(struct idQueriesDoc));
   headQD->QD=headF;
   headQD->idQuery=idQuery;
   headQD->sgte=NULL;
  }
  else headQD->sgte=InsertarGlobalQD(headQD->sgte,idQuery,headD,T,D,headQ);
   return(headQD);
}
float DISTANCIA_Q_CLUSTER(unsigned short int idQuery,struct Lista_ *head,ENLACEQ headQ)
{
  int stop,k;
  unsigned short int TermQuery[8],NTermC;
  float suma=0.0;
  for(k=0;k<8;k++)
   TermQuery[k]=0;
  Obtener_TQuery(headQ,idQuery,TermQuery,1) ;
  for(k=0,stop=0;k<8 && stop==0;k++)
   if(EstaTermino(head,TermQuery[k])==1 && TermQuery[k]!=0)
    stop=1;
   else continue;
  if(stop==1)
```

```
 {
  NTermC=NTermCluster(head, 0);
  for(k=0,stop=0;k<8 ;k++)
   if(EstaTermino(head,TermQuery[k])==0)
    continue;
   else
  suma=suma+1.0;
  return(suma/sqrt(NTermC*NTermC));
 }
 else return(0.0);
}
float DISTANCIACLUSTER(unsigned short int idDoc,struct Lista_ *head, struct Lista *head_D,
DOCUMENT headD)
{
 int stop,k;
 unsigned short int TermDoc1[30],NTermC;
 float suma=0.0;
 for(k=0;k<30;k++)
  TermDoc1[k]=0;
 Obtener_TDocumento(headD,idDoc,TermDoc1,1);
 for(k=0,stop=0;k<30 && stop==0;k++)
  if(EstaTermino(head,TermDoc1[k])==1 && TermDoc1[k]!=0)
  {
```

```
    stop=1;
   }
   else continue;
 if(stop==1)
 {
   NTermC=NTermCluster(head, 0);
   for(k=0,stop=0;k<30 ;k++)
     if(EstaTermino(head,TermDoc1[k])==0)
       continue;
     else
       suma=suma+1.0;
   return(suma/sqrt(NTermC*NTermC));
 }
 else return(0.0);
}
int ObtenerLargo(struct Query_Doc *head,int i)
{
 if(head!=NULL)
   return(ObtenerLargo(head->sgte,(i+1)));
 else return(i);
}
struct Query_Doc ObtenerDatosQ(struct Query_Doc *head,int i,int j)
```

```
{
 struct Query_Doc Aux;
 if(head!=NULL && i==j)
 {
  Aux.idQuery=head->idQuery;
  Aux.idDoc=head->idDoc;
  Aux.Dis=head->Dis;
  Aux.Marcado=head->Marcado;
  Aux.sgte=NULL;
 }
 else if(head!=NULL)
  return(ObtenerDatosQ(head->sgte,i,(j+1)));
 return(Aux);
}
struct Query_Doc *MarcadoQD(struct Query_Doc *head,int idQuery)
{
 if(head!=NULL && head->idQuery==idQuery)
  head->Marcado=1;
 else if(head!=NULL)
  head->sgte=MarcadoQD(head->sgte,idQuery);
 return(head);
}
```

```
PENLACEQ Rankear2(PENLACEQ HEAD, int Arroba30, int Arroba10, float S)
{
 //S=2,3,4 para calcular la funcion Zeta=Zifp=Pareto=Skewed
 unsigned short int i,j,l,y;
 float suma,u_,e,b,div,J,L;
 struct Zeta *P;
 int RelDoc[30];
 if(HEAD!=NULL)
 {
  P=(struct Zeta *)malloc(Arroba30*sizeof(struct Zeta));
  for(i=0,suma=0.0;i<Arroba30;i++)
  {
   P[i].Posicion=(i+1);
   div=(pow((float)(i+1),S));
   P[i].Uno_div_P=1/div;
   suma=suma+P[i].Uno_div_P;
  }
  //Esto deberia de hacer elegir un termino del documento bajo la distribucion Skewed
  for(j=0;j<30;j++)
  {
   u_=rand()%(Arroba30+1);
   u_=u_/100;
```

```
  b=(u_*suma);
  e=1/pow(b,(1/S));
  y=ceil(e);
  RelDoc[j]=y;
  printf("j=%d, e=%f, y=%d \n",j,e,y);
  if( y<1 || y>Arroba30)
    printf("Error en la distribucion Skewed \n");
 }
 for(i=0;i<30;i++)
  for(j=i+1;j<30;j++)
    if(RelDoc[i]!=0 && RelDoc[i]==RelDoc[j])
      RelDoc[j]=0;
    else continue;
 for(i=0;i<30;i++)
  if(RelDoc[i]!=0)
    RelDoc[i]=1;
  else continue;
 for(l=0;l<30;l++)
 {
  HEAD->PyR_idDoc[l]=RelDoc[l];
  printf(" HEAD->PyR_idDoc[l:%d]:%d, idDoc:%d\n",l,RelDoc[l],HEAD->idDoc[l][0]);
 }
```

```
   for(l=0,j=0,suma=0.0;l<Arroba10;l++)
   {
    if(RelDoc[l]==1)
      j++;
    L=(float)(l+1);
    J=(float)j;
    suma=suma+(J/L);
    printf("suma:%f \n",suma);
   }
   printf("La precision es:%f, Arroba10:%d\n",suma,Arroba10);
   HEAD->Precision=suma;
   HEAD->sgte=Rankear2(HEAD->sgte, Arroba30, Arroba10, S);
 }
 return(HEAD);
}
int ObtenerTopicQ(int idQuery,ENLACEQ  headQ)
{
 if(headQ!=NULL && headQ->idQuery==idQuery)
   return(headQ->Topic);
 else return(ObtenerTopicQ(idQuery,headQ->sgte));
}
int ObtenerTopicD(int idDoc,DOCUMENT headD)
```

```
{
 if(headD!=NULL && headD->idDoc==idDoc)
   return(headD->Topico);
 else return( ObtenerTopicD(idDoc,headD->sgte));
}
PENLACEQ INSERTAR(PENLACEQ head, struct Query_Doc AuxF, int idQuery,int i,ENLACEQ
headQ,DOCUMENT headD)
{
 if(head==NULL)
 {
   head=(PENLACEQ)malloc(sizeof(PLISTAQ));
   head->idQuery=idQuery;
   head->Topic=ObtenerTopicQ(idQuery,headQ);
   head->idDoc[i][0]=AuxF.idDoc;
   head->idDoc[i][1]=ObtenerTopicD(AuxF.idDoc,headD);
   head->PyR_idDoc[i]=0; // despues calculo la relevancia
   head->sgte=NULL;
 }
 else head->sgte=INSERTAR(head->sgte,AuxF,idQuery,i,headQ,headD);
 return(head);
}
int Contar_Lista(struct Query_Doc *head,int i)
{
```

```
 if(head!=NULL)
  return(Contar_Lista(head->sgte,(i+1)));
 else return(i);
}
PENLACEQ INSERTAR_(PENLACEQ head, struct idQueriesDoc *headQD,ENLACEQ
headQ,DOCUMENT headD)
{
 int i,N;
 struct Query_Doc Aux;
 if(head==NULL)
 {
  head=(PENLACEQ)malloc(sizeof(PLISTAQ));
  head->cantTermIguales=0;//esto lo agregue para contar los terminos que existen en los
nuevos documentos relevantes y tambien en los antiguos
  head->contador = 0;
  head->idQuery=headQD->idQuery;
  N=Contar_Lista(headQD->QD,0);
  printf("Numero de Documentos INSERTAR_ N:%d\n",N);
  if(N>30)
  {
   printf("SE DIO DENTRO DE INSERTAR_ N>30\n");
   N=30;
  }
```

```
   for(i=0;i<N;i++)
   {
     Aux=ObtenerDatosQ(headQD->QD,(i+1),1);
     head->idDoc[i][0]=Aux.idDoc;
     head->cantTermIgualesPorDocumento[i]=0;//esto lo agregue para contar los terminos iguales que estan en NDOQ y ODOQ
     head->NDoc=N;
     head->Dis[i][0]=Aux.Dis; // esto es la distancia
     head->Dis[i][1]=0.0; // significa que no está marcado 0.0
     printf("Query:%d  idDoc:%d, Topico:%d \n",headQD->idQuery,head->idDoc[i][0],head->idDoc[i][1]);
     head->PyR_idDoc[i]=0; // despues calculo la relevancia, e
   }
   head->sgte=NULL;
 }
 else head->sgte=INSERTAR_(head->sgte,headQD,headQ,headD);
 return(head);
}
PLISTAQ ObtenerDatosPENLACEQ(PENLACEQ head,int i,int j)
{
 PLISTAQ Aux;
 int k;
 if(head!=NULL && i==j)
```

```
 {
   for(k=0; k<30;k++)
   {
     Aux.idDoc[k][0]=head->idDoc[k][0];
     Aux.idDoc[k][1]=head->idDoc[k][1];
     Aux.PyR_idDoc[k]=head->PyR_idDoc[k];
     Aux.Dis[k][0]=head->Dis[k][0];
     Aux.Dis[k][1]=head->Dis[k][1];
   }
   Aux.idQuery=head->idQuery;
   Aux.Topic=head->Topic;
   Aux.NDoc=head->NDoc;
   Aux.sgte=NULL;
   return(Aux);
 }
 else if(head!=NULL)
   return(ObtenerDatosPENLACEQ(head->sgte,i,(j+1)));
}
int ObtenerLargo_(PENLACEQ head)
{
 if(head!=NULL)
   return(head->NDoc);
```

```
 else return(0);
}
void mostrarHEAD(PENLACEQ head){
 if (head != NULL)
 {
  for (int i = 0; i < 30; ++i)
  {
   printf("%d->\n",head->idDoc[i][0]);
  }
  mostrarHEAD(head->sgte);
 }
}
PENLACEQ OrdenarQD2(PENLACEQ head, int Arroba30)
{
 int i,j,NDoc,idDocAux,TopicAux,l;
 float DisAux,may,may2;
 if(head!=NULL)
 {
  NDoc=ObtenerLargo_(head);  // hay que colocar NDoc en la estructura
  printf("Dentro de Ordenar, NDoc:%d \n",NDoc);
  for(i=0; i<Arroba30;i++)
  {
```

```
    for(j=i+1,l=i,may=head->Dis[i][0];j<NDoc;j++)
    {
     may2=head->Dis[j][0];
     if(may2>may)
     {
       l=j;
       may=may2;
     }
     else continue;
    } //j<NDoc
    idDocAux=head->idDoc[i][0];
    TopicAux=head->idDoc[i][1];
    DisAux=head->Dis[i][0]; // esto es lo que hay que colocar en Past_Query
    head->idDoc[i][0]=head->idDoc[l][0];
    head->idDoc[i][1]=head->idDoc[l][1];
    head->Dis[i][0]=head->Dis[l][0];
    head->idDoc[l][0]=idDocAux;
    head->idDoc[l][1]=TopicAux;
    head->Dis[l][0]=DisAux;
    printf("idQuery:%d, idDoc:%d, Dis:%f \n",head->idQuery, head->idDoc[i][0], head-
>Dis[i][0] );
    // Aqui hago la rotacion
  }  // for i<Arroba30
```

```
   head->sgte=OrdenarQD2(head->sgte, Arroba30);
  }// head!=NULL
  return(head);
}
float Distancia(struct Doc_ P,struct Lista_ *head, int i, int j, DOCUMENT headD)
{
  int stop,k;
  unsigned short int TermDoc1[30],s,NDoc;
  float suma=0.0,f;
  for(k=0;k<30;k++)
    TermDoc1[k]=0;
  Marcar_Documento(headD,P.idDoc,TermDoc1,0);  // debo de agregar esta funcion
  for(k=0,stop=0;k<30 && stop==0;k++)
    if(EstaTermino(head,TermDoc1[k])==0)
      stop=1;
    else continue;
  if(stop==1)
  {
    NDoc=head->NDocClus+1;
    for(k=0,stop=0,f=1.0;k<30 ;k++)
      if(EstaTermino(head,TermDoc1[k])==0)
        suma=suma +log(NDoc/1.0); // esto va en actualizar
```

```
    else{
      s=NumeroTerminos(head,TermDoc1[k]);
      f=f*log(NDoc/((float) s))*log(NDoc/((float) s));
      suma= suma+log(NDoc/((float) s))*log(NDoc/((float) s)); // esto es lo que deberia dar
    }
  return(suma/sqrt(f));
 }
 else return(0.0);
}
struct Lista_ * ActualizarNDoc(struct Lista_ * L, unsigned short int NDoc)
{
 if(L!=NULL)
   L->NDocClus=NDoc;
 return(L);
}
int ObtenerNumeroDocumentos(struct Lista_ * L)
{
 if(L!=NULL)
   return(L->NDocClus);
 else return(0);
}
/*****************************************************************************************
*****************/
```

```
/*********************************** FIN K-MEANS
*****************************************************************/

/*************************************************************************************************
*****************/

struct Exponencial{

 float L_izq;

 float L_der;

        float ValorDis; // este valor es un real entre 0-1

 int ValorEnt;

};

struct Skewed{

 float L_izq;

 float L_der;

        float ValorDis; // este valor es un real entre 0-1

 int Posicion;

 float Uno_div_P;

};

void GenerarPareto(int Alfa, int Y, int NTerm)

{

 float u_,b,x;

 u_=rand()%(NTerm+1);

        b=pow(Y,Alfa+1);

 b=(b*u_)/Alfa;
```

```
 x=pow(b,(1/Alfa));
}
void GenerarZeta(int NTerm, float S)
{
 //Esto es para seleccionar los terminos de un topico para un determinado documento
 int i,y;
 struct Zeta *P;
 float suma,u_,e,b;
 P=(struct Zeta *)malloc(NTerm*sizeof(struct Zeta));
 for(i=0,suma=0.0;i<NTerm;i++)
 {
  P[i].Posicion=(i+1);
  P[i].Uno_div_P=1/(pow((float)(i+1),S));
  suma=suma+P[i].Uno_div_P;
 }
 //Esto deberia de hacer elegir un termino del documento bajo la distribucion Skewed
 u_=rand()%+101;
 u_=u_/100;
 b=(u_*suma);
 e=1/(pow(b,(1/S)));
 y=ceil(e);
 if( y<1 || y>NTerm)
```

```
   printf("Error en la distribucion Skewed \n");
}
void GenerarSkewed(int NTerm, float Teta)
{
 //Esto es para seleccionar los terminos de un topico para un determinado documento
 int i;
 struct Skewed *P;
 float A,suma,u_,e,b,x;
 P=(struct Skewed *)malloc(NTerm*sizeof(struct Skewed));
 for(i=0,suma=0.0;i<NTerm;i++)
 {
  P[i].Posicion=(i+1);
  P[i].Uno_div_P=1/(pow((float)(i+1),Teta));
  suma=suma+P[i].Uno_div_P;
 }
 A=1/suma;
 for(i=0;i<NTerm;i++)
 {
  P[i].ValorDis=A/pow(i+1,Teta);
  printf (" Term %d=  P sub x=%f \n",i+1,P[i].ValorDis);
 }
 //Esto deberia de hacer elegir un termino del documento bajo la distribucion Skewed
```

```
 u_=rand()%(NTerm+1);
 b=((u_*(-1.0*Teta+1))/A);
 e=1/(-1.0*Teta+1);
 x=pow(b,e);
 if( x<1 || x>NTerm)
   printf("Error en la distribucion Skewed \n");
}
void MarcarTerm(unsigned short int IdTerm,TERMS headT)
{
 if(headT!=NULL && headT->id==IdTerm && headT->Marcado==0)
 {
   headT->Marcado=1;
 }
 else if(headT->sgte!=NULL)
   MarcarTerm(IdTerm,headT->sgte);
}
/*****************************************************************************************
********/
/**************************        INICIO  GENERAR  QUERIES
************************/
/*****************************************************************************************
********/
int Buscar(int idTerm,unsigned short int TermQuery[])
```

```
{
 int i;
 for(i=0;i<8;i++)
  if(TermQuery[i]==idTerm)
    return(1);
  else continue;
 return(0);
}
int Esta(unsigned short int TermQuery[], ENLACEQ headQ)
{
 int i,l,m,stop;
 if(headQ!=NULL)
 {
  printf("Dentro de Esta\n");
  for(i=0,l=0; i<8;i++)
    if(TermQuery[i]!=0)
      l++;
    else continue;
  for(i=0,m=0; i<8;i++)
    if(headQ->TermDoc[i]!=0)
      m++;
    else continue;
```

```
  if(l==m)
  {
   for(i=0,stop=1; i<l && stop==1 ;i++)
     if(Buscar(headQ->TermDoc[i],TermQuery)==1)
       continue;
     else stop=0;
     if(stop==1)
     {
       printf("El valor a retornar por Esta=1\n");
       return(1);
     }
     else return(Esta(TermQuery,headQ->sgte));
  }
  else return(Esta(TermQuery,headQ->sgte));
 }
 else  return(0);
}
struct Term_Q{
 unsigned short int TermDoc;
 struct Term_Q *sgte;
};
struct Term_Q *InsertarT(struct Term_Q *headIDT,unsigned short int TermQuery)
```

```
{
 if(headIDT==NULL)
 {
   headIDT=(struct Term_Q *)malloc(sizeof(struct Term_Q));
   headIDT->TermDoc=TermQuery;
   headIDT->sgte=NULL;
 }
 else headIDT->sgte=InsertarT(headIDT->sgte,TermQuery);
 return(headIDT);
}
int EstaEn(unsigned short int TermDoc_, struct Term_Q *headIDT)
{
 if(headIDT!=NULL && headIDT->TermDoc==TermDoc_)
   return(1);
 else if(headIDT!=NULL)
   return(EstaEn(TermDoc_, headIDT->sgte));
 return(0);
}
ENLACEQ GenerarQueries_4(int NQqueries, ENLACEQ headQ,DOCUMENT headD,int NDoc)
{
 int i,idDoc,l,NTerm,IdTopic,idQuery,j,completo;
 unsigned short int TermDoc_[30],TermQuery[8],TermQuery2[8];
```

```
struct Term_Q *headIDT=NULL;
printf("Dentro de GenerarQueries_3 \n");
printf("El numero de queries es :%d \n",NQqueries);
printf("VERIFICANDO EL FUNCIONAMIENTO:%d \n",NQqueries);
idQuery=1;
do{
 for(i=0;i<8;i++)
 {
  TermQuery[i]=0;
  TermQuery2[i]=0;
 }
 for(i=0;i<30;i++)
  TermDoc_[i]=0;
 idDoc=(rand()%NDoc)+1;
 Obtener_TDocumento(headD,idDoc,TermDoc_,1);
 IdTopic=ObtenerTopicoD(idDoc,headD);
 for(i=0,l=0;i<30;i++)
  if(TermDoc_[i]!=0)
   l++;
  else continue;
 NTerm=(rand()%5)+3; // numero de terminos por query
 i=0;
```

```
j=0;
while(i<NTerm && j<l)
{
 if(EstaEn(TermDoc_[j],headIDT)==1 && j<l)
  j++;
 else TermQuery[i++]=TermDoc_[j++];
}// i<NTerm
completo=0;
if(j<l)
{
 for(i=0;i<NTerm;i++)
  TermQuery2[i]=TermQuery[i];
 while(j<l && completo==0)
 {
  if(EstaEn(TermDoc_[j],headIDT)==0)
  {
   TermQuery2[i-1]=TermDoc_[j];
   for(i=0;i<NTerm;i++)
    headIDT=InsertarT(headIDT,TermQuery[i]);
   headIDT=InsertarT(headIDT,TermQuery2[i-1]);
   headQ=InsertQ(headQ,TermQuery,idQuery,IdTopic);
   idQuery=idQuery+1;
```

```
        headQ=InsertQ(headQ,TermQuery2,idQuery,IdTopic);
        completo=1;
       }// if EstaEn
       else j++;
     }// j<l
   }// if j<l
   if(completo==1)
    idQuery=idQuery+1;
 }while(idQuery<=NQqueries);
 return(headQ);
}
/*************************************************************************************************
********/
/*************************        INICIO   GENERAR  DOCUMENTOS
************************/
/*************************************************************************************************
********/
DOCUMENT GenerarDocumentos(int NumberDocuments, int NumberTopics, int NTermst,
double Lambda,TERMS headT)//genera documentos antiguos
{
 int i,j,k,l,a,b,y,NTermsbyDoc,NDocbyTop,stop1,stop2,o,z,m;
 unsigned short int TermDoc[30],IdTerm,idDoc=0;
 float var,var1,inic,u_,FijoInterval,x,u;
 DOCUMENT headD=NULL;
```

```
struct Exponencial *A;
struct Topicos *T;
NDocbyTop=NumberDocuments/NumberTopics;
A=(struct Exponencial *)malloc(NumberTopics*sizeof(struct Exponencial ));
T=(struct Topicos *)malloc(NumberTopics*sizeof(struct Topicos ));
for(i=0;i<NumberTopics;i++)
{
 T[i].IdTopico=i+1;
 T[i].IdVar=i+1;
}
for(i=0;i<30;i++)
 TermDoc[i]=0;
var=(1.0)/(float)NumberTopics;
var1=var;
for(i=0, inic=0.0 ;i<NumberTopics;i++)//nose que hace este for
{
 A[i].ValorEnt=i+1;
 A[i].ValorDis=Lambda*exp(-1.0*(i+1)*Lambda);
 A[i].L_izq=inic;
 A[i].L_der=var1;
 inic=var1;
 var1=var1+var;
```

```
 }
 for(j=NumberTopics-1;j>=0;j--)
 {
  if(j==NumberTopics-1)
  {
   A[j].L_izq=0.0;
   A[j].L_der=A[j].ValorDis;
  }
  else{
   A[j].L_izq=A[j+1].ValorDis;
   A[j].L_der=A[j].ValorDis;
  }
 }
 FijoInterval=Lambda*exp(-1.0*1*Lambda); //1 porque es el valor maximo que puede tomar
(0-1)
 printf("DENTRO DE DISTRIBUCION EXPONENCIAL \n");
 printf("FijoInterval:%f \n", FijoInterval);
 for(i=1;i<=NumberTopics;i++)
 {
  for(j=1;j<=NDocbyTop;j++)
  {
   NTermsbyDoc=rand()%15+15;//%25+15 es el minimo
   for(k=1;k<=NTermsbyDoc;k++)
```

```
    {
     u_=rand()%101;
     u_=u_/100;
     u=u_* FijoInterval;
     for(m=0,stop1=0;m<NumberTopics && stop1==0;m++)  // Aqui hay que odernar de
acuerdo a dis
      if(u>=A[m].L_izq && u<A[m].L_der && m!=(NumberTopics-1))
      {
       x=(log(A[m].L_der))*((-1)*(1.0/Lambda));
       y=ceil(x); //esto es lo que tenia antes, porque no estaba con todos los decimales
       stop1=1;
      }
              else if(m==(NumberTopics-1))
      {
       x=(log(FijoInterval))*((-1)*(1.0/Lambda));
       y=ceil(x);
       stop1=1;
      }
     for(l=0,stop2=0; l<NumberTopics && stop2==0; l++)
     {
      while(stop2==0)
      {
       if(y==T[l].IdTopico)
```

```
{
                        b=0;
 while(b<NumberTopics && stop2==0)
 {
                        if(T[b].IdVar==T[l].IdTopico)
                                { // Agregar los que me falta para tener el termino del
topico
   // el while stop2 es porque se debe de agregar un termino no repetido en TermDoc
   while(stop2==0)
   {
    IdTerm= (rand() % NTermst) + 1;
    if((BuscarT_D(TermDoc,IdTerm))==0)
                                {
                                 AgregarTerm_Doc(TermDoc,IdTerm);
                                        MarcarTerm(IdTerm,headT);
                                 stop2=1;
                                }
                                else continue;
    }// while stop2==0
   }
   b++;
 }
} // y==T[l].IdTopico
```

```
        else l++; // y==T[i]
      } // while stop
    } // for l
  } // for k
  int borrar = 0;
  for(a=0;a<30 && TermDoc[a] != 0;a++){
    borrar++;
  }
  idDoc++;
  headD=InsertD(headD,TermDoc,y,idDoc);
            for(a=0;a<30;a++)
                    TermDoc[a]=0;
}// for j
// Aqui va el cambio de los topicos
for(o=0,z=i,stop2=0;o<NumberTopics && stop2==0;o++,z++)
if(z<NumberTopics)
  T[o].IdVar=T[z].IdTopico;
else{
  for(z=1;o<NumberTopics && stop2==0;o++,z++)
    T[o].IdVar=z;
    stop2=1;
}
```

```
 } // for i
 return(headD);
}
DOCUMENT GenerarDocumentos2(int NumberDocuments, int NumberTopics, int NTermst,
double Lambda,TERMS headT,TERMS headT2)//genera documentos nuevos
{
 int i,j,k,l,a,b,y,NTermsbyDoc,NDocbyTop,stop1,stop2,o,z,m;
 unsigned short int TermDoc[30],IdTerm,idDoc=0;
 float var,var1,inic,u_,FijoInterval,x,u;
 DOCUMENT headD=NULL;
 struct Exponencial *A;
 struct Topicos *T;
 NDocbyTop=NumberDocuments/NumberTopics;
 A=(struct Exponencial *)malloc(NumberTopics*sizeof(struct Exponencial ));
 T=(struct Topicos *)malloc(NumberTopics*sizeof(struct Topicos ));
 for(i=0;i<NumberTopics;i++)
 {
  T[i].IdTopico=i+1;
  T[i].IdVar=i+1;
 }
 for(i=0;i<30;i++)
  TermDoc[i]=0;
 var=(1.0)/(float)NumberTopics;
```

```
var1=var;
for(i=0, inic=0.0 ;i<NumberTopics;i++)//no se que hace este for
{
 A[i].ValorEnt=i+1;
 A[i].ValorDis=Lambda*exp(-1.0*(i+1)*Lambda);
 A[i].L_izq=inic;
 A[i].L_der=var1;
 inic=var1;
 var1=var1+var;
}
for(j=NumberTopics-1;j>=0;j--)
{
 if(j==NumberTopics-1)
 {
  A[j].L_izq=0.0;
  A[j].L_der=A[j].ValorDis;
 }
 else{
  A[j].L_izq=A[j+1].ValorDis;
  A[j].L_der=A[j].ValorDis;
 }
}
```

```
 FijoInterval=Lambda*exp(-1.0*1*Lambda); //1 porque es el valor maximo que puede tomar
(0-1)

 for(i=1;i<=NumberTopics;i++)

 {

  for(j=1;j<=NDocbyTop;j++)

  {

   NTermsbyDoc=rand()%15+15;//%25+15 es el minimo

   int cantOldTerms = ceil((NTermsbyDoc * 0.66));

   int cantNewTerms = NTermsbyDoc - cantOldTerms;

   for(k=1;k<=cantOldTerms;k++)

   {

    u_=rand()%101;

    u_=u_/100;

    u=u_* FijoInterval;

    for(m=0,stop1=0;m<NumberTopics && stop1==0;m++)  // Aqui hay que odernar de
acuerdo a dis

     if(u>=A[m].L_izq && u<A[m].L_der && m!=(NumberTopics-1))

     {

      x=(log(A[m].L_der))*((-1)*(1.0/Lambda));

      y=ceil(x); //esto es lo que tenia antes, porque no estaba con todos los decimales

      stop1=1;

     }

     else if(m==(NumberTopics-1))
```

```
 {
  x=(log(FijoInterval))*((-1)*(1.0/Lambda));
  y=ceil(x);
  stop1=1;
 }
for(l=0,stop2=0; l<NumberTopics && stop2==0; l++)
{
 while(stop2==0)
 {
  if(y==T[l].IdTopico)
  {
   b=0;
   while(b<NumberTopics && stop2==0)
   {
    if(T[b].IdVar==T[l].IdTopico)
    { // Agregar los que me falta para tener el termino del topico
     // el while stop2 es porque se debe de agregar un termino no repetido en TermDoc
     while(stop2==0)
     {
      IdTerm=(rand()%(NTermst/NumberTopics))+1;
      IdTerm=IdTerm+NDocbyTop*l;
      if((BuscarT_D(TermDoc,IdTerm))==0)
```

```
            {
              AgregarTerm_Doc(TermDoc,IdTerm);
              MarcarTerm(IdTerm,headT);
              stop2=1;
            }
            else continue;
          }// while stop2==0
        }
        b++;
      }
    } // y==T[l].IdTopico
    else l++; // y==T[i]
  } // while stop
 } // for l
} // for k
for(k=1;k<=cantNewTerms;k++)//k2
{
 u_=rand()%101;
 u_=u_/100;
 u=u_* FijoInterval;
 for(m=0,stop1=0;m<NumberTopics && stop1==0;m++)  // Aqui hay que odernar de
acuerdo a dis
  if(u>=A[m].L_izq && u<A[m].L_der && m!=(NumberTopics-1))
```

```
 {
  x=(log(A[m].L_der))*((-1)*(1.0/Lambda));
  y=ceil(x); //esto es lo que tenia antes, porque no estaba con todos los decimales
  stop1=1;
 }
 else if(m==(NumberTopics-1))
 {
  x=(log(FijoInterval))*((-1)*(1.0/Lambda));
  y=ceil(x);
  stop1=1;
 }
for(l=0,stop2=0; l<NumberTopics && stop2==0; l++)
{
 while(stop2==0)
 {
  if(y==T[l].IdTopico)
  {
   b=0;
   while(b<NumberTopics && stop2==0)
   {
    if(T[b].IdVar==T[l].IdTopico)
    { // Agregar los que me falta para tener el termino del topico
```

```
          // el while stop2 es porque se debe de agregar un termino no repetido en TermDoc
           while(stop2==0)
           {
            IdTerm=(rand()%(NTermst/NumberTopics))+1;
            IdTerm=IdTerm+NDocbyTop*l;
            if((BuscarT_D2(TermDoc,IdTerm,cantOldTerms))==0)
            {
             AgregarTerm_Doc(TermDoc,IdTerm);
             MarcarTerm(IdTerm,headT2);
             stop2=1;
            }
            else continue;
           }// while stop2==0
         }
         b++;
        }
      } // y==T[l].IdTopico
      else l++; // y==T[i]
    } // while stop
  } // for l
} // for k2
idDoc++;
```

```
    headD=InsertD(headD,TermDoc,y,idDoc);
    for(a=0;a<30;a++)
      TermDoc[a]=0;
   }// for j
   // Aqui va el cambio de los topicos
   for(o=0,z=i,stop2=0;o<NumberTopics && stop2==0;o++,z++)
    if(z<NumberTopics)
      T[o].IdVar=T[z].IdTopico;
    else{
      for(z=1;o<NumberTopics && stop2==0;o++,z++)
        T[o].IdVar=z;
      stop2=1;
    }
  } // for i
  return(headD);
}
TERMS buscarTerPorId(TERMS headT,int id){
  while(headT != NULL){
   if (headT->id == id)
   {
    return(headT);
   }
```

```
   headT=headT->sgte;
  }
}
DOCUMENT GenerarDocumentos2ContandoTerRelevantes(int NumberDocuments, int
NumberTopics, int NTermst,int NTermst2, double Lambda,TERMS headT,TERMS
headT2,TERMSR listaTerRel)
{
  int i,j,k,l,a,b,y,NTermsbyDoc,NDocbyTop,stop1,stop2,o,z,m,contadorRelevantes=0;
  unsigned short int TermDoc[30],IdTerm,idDoc=0;
  float var,var1,inic,u_,FijoInterval,x,u;
  TERMSR listaTerRelAux = NULL;
  DOCUMENT headD=NULL;
  struct Exponencial *A;
  struct Topicos *T;
  NDocbyTop=NumberDocuments/NumberTopics;
  A=(struct Exponencial *)malloc(NumberTopics*sizeof(struct Exponencial ));
  T=(struct Topicos *)malloc(NumberTopics*sizeof(struct Topicos ));
  for(i=0;i<NumberTopics;i++)
  {
   T[i].IdTopico=i+1;
   T[i].IdVar=i+1;
  }
  for(i=0;i<30;i++)
```

```
 TermDoc[i]=0;
var=(1.0)/(float)NumberTopics;
var1=var;
for(i=0, inic=0.0 ;i<NumberTopics;i++)//nose que hace este for
{
 A[i].ValorEnt=i+1;
 A[i].ValorDis=Lambda*exp(-1.0*(i+1)*Lambda);
 A[i].L_izq=inic;
 A[i].L_der=var1;
 inic=var1;
 var1=var1+var;
}
for(j=NumberTopics-1;j>=0;j--)
{
 if(j==NumberTopics-1)
 {
  A[j].L_izq=0.0;
  A[j].L_der=A[j].ValorDis;
 }
 else{
  A[j].L_izq=A[j+1].ValorDis;
  A[j].L_der=A[j].ValorDis;
```

```
  }
 }
 FijoInterval=Lambda*exp(-1.0*1*Lambda); //1 porque es el valor maximo que puede tomar (0-1)
 printf("DENTRO DE DISTRIBUCION EXPONENCIAL \n");
 printf("FijoInterval:%f \n", FijoInterval);
 for(i=1;i<=NumberTopics;i++)
 {
  for(j=1;j<=NDocbyTop;j++)
  {
   NTermsbyDoc=rand()%15+15;//%25+15 es el minimo
   int cantOldTerms = ceil((NTermsbyDoc * 0.66));
   int cantNewTerms = NTermsbyDoc - cantOldTerms;
   for(k=1;k<=cantOldTerms;k++)
   {
    u_=rand()%101;
    u_=u_/100;
    u=u_* FijoInterval;
    for(m=0,stop1=0;m<NumberTopics && stop1==0;m++)  // Aqui hay que odernar de acuerdo a dis
    if(u>=A[m].L_izq && u<A[m].L_der && m!=(NumberTopics-1))
    {
     x=(log(A[m].L_der))*((-1)*(1.0/Lambda));
```

```
 y=ceil(x); //esto es lo que tenia antes, porque no estaba con todos los decimales
 stop1=1;
}
else if(m==(NumberTopics-1))
{
 x=(log(FijoInterval))*((-1)*(1.0/Lambda));
 y=ceil(x);
 stop1=1;
}
for(l=0,stop2=0; l<NumberTopics && stop2==0; l++)
{
 while(stop2==0)
 {
  if(y==T[l].IdTopico)
  {
   b=0;
   while(b<NumberTopics && stop2==0)
   {
    if(T[b].IdVar==T[l].IdTopico)
    { // Agregar los que me falta para tener el termino del topico
     // el while stop2 es porque se debe de agregar un termino no repetido en TermDoc
      while(stop2==0)
```

```
 {
  IdTerm=(rand()%(NTermst-1))+1;
  if((BuscarT_D(TermDoc,IdTerm))==0)
  {
   listaTerRelAux = listaTerRel;
   while(listaTerRelAux != NULL){
    TERMS tAux = NULL;
    tAux = buscarTerPorId(headT2,IdTerm);
    if (listaTerRelAux->id == IdTerm)
    {
     contadorRelevantes++;
     break;
    }
    listaTerRelAux = listaTerRelAux->sgte;
   }
   AgregarTerm_Doc(TermDoc,IdTerm);
   MarcarTerm(IdTerm,headT);
   stop2=1;
  }
  else continue;
 }// while stop2==0
}
```

```
            b++;
           }
         } // y==T[l].IdTopico
         else l++; // y==T[i]
       } // while stop
      } // for l
    } // for k
    for(k=1;k<=cantNewTerms;k++)//k2
    {
      u_=rand()%101;
      u_=u_/100;
      u=u_* FijoInterval;
      for(m=0,stop1=0;m<NumberTopics && stop1==0;m++)  // Aqui hay que odernar de
acuerdo a dis
        if(u>=A[m].L_izq && u<A[m].L_der && m!=(NumberTopics-1))
        {
          x=(log(A[m].L_der))*((-1)*(1.0/Lambda));
          y=ceil(x); //esto es lo que tenia antes, porque no estaba con todos los decimales
          stop1=1;
        }
        else if(m==(NumberTopics-1))
        {
          x=(log(FijoInterval))*((-1)*(1.0/Lambda));
```

```
   y=ceil(x);
   stop1=1;
  }
for(l=0,stop2=0; l<NumberTopics && stop2==0; l++)
{
 while(stop2==0)
 {
  if(y==T[l].IdTopico)
  {
   b=0;
   while(b<NumberTopics && stop2==0)
   {
    if(T[b].IdVar==T[l].IdTopico)
    { // Agregar los que me falta para tener el termino del topico
     // el while stop2 es porque se debe de agregar un termino no repetido en TermDoc
      while(stop2==0)
      {
       IdTerm=(rand() % (NTermst2-1))+1;
       if((BuscarT_D2(TermDoc,IdTerm,cantOldTerms))==0)
       {
        AgregarTerm_Doc(TermDoc,IdTerm *-1);
        MarcarTerm(IdTerm,headT2);
```

```
                stop2=1;
               }
               else continue;
             }// while stop2==0
            }
            b++;
           }
         } // y==T[l].IdTopico
         else l++; // y==T[i]
       } // while stop
     } // for l
   } // for k2
   idDoc++;

headD=InsertDConTerRelevantes(headD,TermDoc,y,idDoc,contadorRelevantes,NTermsbyDoc
);
   contadorRelevantes = 0;
   for(a=0;a<30;a++)
     TermDoc[a]=0;
  }// for j
  // Aqui va el cambio de los topicos
  for(o=0,z=i,stop2=0;o<NumberTopics && stop2==0;o++,z++)
   if(z<NumberTopics)
```

```
      T[o].IdVar=T[z].IdTopico;
     else{
      for(z=1;o<NumberTopics && stop2==0;o++,z++)
        T[o].IdVar=z;
      stop2=1;
     }
 } // for i
 return(headD);
}
/*****************************************************************************************
********/
/************************************ FIN GENERAR DOCUMENTOS
************************************/
/*****************************************************************************************
********/
DOCUMENT Desmarcar_Documento(DOCUMENT headD)
{
 if(headD!=NULL)
 {
   headD->Marcado=0;
   headD->sgte=Desmarcar_Documento(headD->sgte);
 }
 return(headD);
```

```
}
int ContarHeadPQ(PENLACEQ HEAD,int i)
{
 if(HEAD!=NULL)
   return(ContarHeadPQ(HEAD->sgte,(i+1)));
 else return(i);
}
int ContarHeadQD(struct idQueriesDoc *headQD, int i)
{
 if(headQD!=NULL)
   return(ContarHeadQD(headQD->sgte, ( i+1)));
 else return(i);
}
void AsignarDistribucionZ(signed short int Rel[], int Arroba30, int Arroba10, float S, int flag)
{
// Si flag es 0 entonces se coloca tal cual, sino solo se asignan los nuevos 1
 int i,j,y;
 struct Zeta *P;
 float suma,u_,e,b,div;
 unsigned short int Interseccion[30];
 for(i=0;i<30;i++)
   Interseccion[i]=0;
```

```
P=(struct Zeta *)malloc(Arroba30*sizeof(struct Zeta));
for(i=0,suma=0.0;i<Arroba30;i++)
{
  P[i].Posicion=(i+1);
  div=(pow((float)(i+1),S));
  P[i].Uno_div_P=1/div;
  suma=suma+P[i].Uno_div_P;
}
for(j=0;j<30;j++)
{
  u_=rand()%(Arroba30+1);
  u_=u_/100;
  b=(u_*suma);
  e=1/pow(b,(1/S));
  y=ceil(e);
  Interseccion[j]=y;
  if( y<1 || y>Arroba30)
  printf("Error en la distribucion Skewed, e:%f, y:%f \n",e,y);
}
for(i=0;i<30;i++)
  for(j=i+1;j<30;j++)
    if(Interseccion[i]!=0 && Interseccion[i]==Interseccion[j])
```

```
      Interseccion[j]=0;
    else continue;
 for(i=0;i<30;i++)
  if(Interseccion[i]!=0)
    Interseccion[i]=1;
  else continue;
 if(flag==0)
  for(i=0;i<30;i++)
    Rel[i]=Interseccion[i];
 else{
  for(i=0;i<30;i++)
    if(Rel[i]==0 && Interseccion[i]==1)
      Rel[i]=1;
    else continue;
 }
}
PENLACEQ calcularPrecision(PENLACEQ ODOQC){
 int cantidadDoc = 0, cont = 0;
 float sumatoria = 0.0;
 if (ODOQC != NULL)
 {
  for (int i = 0; i < 30 && ODOQC->idDoc[i][0] != 0; ++i)
```

```
   {
    cantidadDoc++;
    if (ODOQC->vectorZ[i] == 1)
    {
     cont++;
    }
    sumatoria = sumatoria + ((float)(((float)cont)/((float)cantidadDoc)));
   }
   ODOQC->precision = sumatoria/((float)cantidadDoc);
   ODOQC->sgte = calcularPrecision(ODOQC->sgte);
  }
  return ODOQC;
}
void mostrarPrecisionUnidos(PENLACEQ ODOQC, PENLACEQ ODOQCUnidos){
  if (ODOQC != NULL && ODOQCUnidos != NULL)
  {
   printf("antigua Precision:%f nueva Precision:%f para Query:%d\n",ODOQC-
>precision,ODOQCUnidos->precision,ODOQC->idQuery);
   mostrarPrecisionUnidos(ODOQC->sgte,ODOQCUnidos->sgte);
  }
  else{
   printf("fin mostrarPrecision\n");
  }
```

```
}
void mostrarPromedioPrecision(PENLACEQ ODOQC, PENLACEQ ODOQCUnidos){
 float suma=0.0,sumaUnidos=0.0;
 while(ODOQC != NULL && ODOQCUnidos != NULL){
  suma+=ODOQC->precision;
  sumaUnidos+=ODOQCUnidos->precision;
  ODOQC=ODOQC->sgte;
  ODOQCUnidos = ODOQCUnidos->sgte;
 }
 printf("promedio Antigua=%f promedio Nueva=%f\n",(suma/30.0),(sumaUnidos/30.0));
}
PENLACEQ copiarHEAD(PENLACEQ destino,PENLACEQ origen){
 PENLACEQ ultimo = NULL;
 if (origen != NULL)
 {
  PENLACEQ aux=(PENLACEQ)malloc(sizeof(PLISTAQ));
  for (int i = 0; i < 30; ++i)
  {
   aux->idDoc[i][0] = origen->idDoc[i][0];
   aux->idDoc[i][1] = origen->idDoc[i][1];
   aux->PyR_idDoc[i] = origen->PyR_idDoc[i];
   aux->Dis[i][0] = origen->Dis[i][0];
```

```
  aux->Dis[i][1] = origen->Dis[i][1];
 }
 aux->NDoc = origen->NDoc;
 aux->TipoQ = origen->TipoQ;
 aux->Rankeado = origen->Rankeado;
 aux->idQuery = origen->idQuery;
 aux->Topic = origen->Topic;
 aux->Marcado = origen->Marcado;
 aux->NVertices = origen->NVertices;
 aux->Precision = origen->Precision;
 aux->sgte = NULL;
 if (destino == NULL)
 {
  destino=aux;
 }
 else{
  ultimo = destino;
  while(ultimo->sgte != NULL){
    ultimo = ultimo->sgte;
  }
  ultimo->sgte = aux;
 }
```

```
   copiarHEAD(destino,origen->sgte);
  }
  return destino;
}
PENLACEQ distribucionZHEAD(PENLACEQ H){
  if (H != NULL)
  {
   for (int i = 0; i < 30; ++i)
   {
    H->vectorZ[i] = H->idDoc[i][0];
   }
   AsignarDistribucionZ(H->vectorZ, 30, 10, 2, 0);
   distribucionZHEAD(H->sgte);
  }
  return H;
}
PENLACEQ funcionParaNoRepetirHEAD(DOCUMENT headD2, int
NumberDocuments2,ENLACEQ headQ2,int NQqueries){
  struct Lista *Clus2;
  struct Lista_ *Clus_2;
  int i,j,k=1,aux,m,b,idDoc,stop,N;
  int Arroba30=30,Arroba10=10,NDoc,idQuery/*,porcentaje*/;
  float min,min2;
```

```
 unsigned short int TermDoc_[30],s;
 PENLACEQ HEAD2=NULL;
 struct idQueriesDoc *headQD2=NULL;
 Clus2=(struct Lista*)malloc(k*sizeof(struct Lista)); // esto contiene los cluster documentos
 Clus_2=(struct Lista_*)malloc(k*sizeof(struct Lista_)); // esto contiene los cluster terminos y sera el centroide
 for(i=0,s=1,NDoc=1;i<k;i++)
 {
  Clus2[i].sgte=NULL;
  Clus_2[i].sgte=NULL;
  aux=(rand()%(NumberDocuments2))+1; // aqui habria que colocar un do while para que no se repita aux
  printf("aux=%d\n",aux);
  printf("Generando los cluster \n");
  for(b=0;b<30;b++)
   TermDoc_[b]=0;
  Marcar_Documento(headD2,aux,TermDoc_,1);  // debo de agregar esta funcion
  for(j=0;j<30;j++)
  {
   if(TermDoc_[j]!=0)
   {
    if(EstaTermino(Clus_2[i].sgte,TermDoc_[j])==0)
    {
```

```
      Clus_2[i].sgte=InsertarTC(Clus_2[i].sgte,TermDoc_[j],1,1);
     }
     else{   // Actualizar s e i
      s=NumeroTerminos(Clus_2[i].sgte,TermDoc_[j]);
      s=s+1;
      Clus_2[i].sgte=ActualizarTC(Clus_2[i].sgte,s,TermDoc_[j],1);
     }
    }
    else continue;
  }// for j=0
  NDoc=1;
  Clus_2[i].sgte=ActualizarNDoc(Clus_2[i].sgte,NDoc);
  Clus2[i].sgte=AgregarDocumento(Clus2[i].sgte,aux);   // Insertar
} // for i<k
for(i=1;i<=NumberDocuments2;i++)
{
  for(j=0,min=0.0;j<k;j++)
  {
    if(Marcado_Documento(headD2, i,1)==0)
    {
      min2=DISTANCIACLUSTER(i,Clus_2[j].sgte,Clus2[j].sgte,headD2);
      min2=1;
```

```
   if(min2>min)
   {
    idDoc=i;
    m=j;
   }
   else continue;
  }
  else continue;
}
if(Marcado_Documento(headD2, idDoc,1)==0)
{
  for(b=0;b<30;b++)
   TermDoc_[b]=0;
  Obtener_TDocumento(headD2,idDoc,TermDoc_,1) ;
  for(j=0;j<30;j++)
  {
   if(TermDoc_[j]!=0)
   {
    if(EstaTermino(Clus_2[m].sgte,TermDoc_[j])==0)
    {
     Clus_2[m].sgte=InsertarTC(Clus_2[m].sgte,TermDoc_[j],1,1);
    }
```

```
      else{   // Actualizar s e i
       s=NumeroTerminos(Clus_2[m].sgte,TermDoc_[j]);
       s=s+1;
       Clus_2[m].sgte=ActualizarTC(Clus_2[m].sgte,s,TermDoc_[j],1);
      }
     }
     else continue;
    }// for j=0
    NDoc=ObtenerNumeroDocumentos_(Clus2[m].sgte,0);
    Clus_2[m].sgte=ActualizarNDoc(Clus_2[m].sgte,NDoc);
    Clus2[m].sgte=AgregarDocumento(Clus2[m].sgte,idDoc);// Insertar
    Marcar_Documento2(headD2,idDoc,TermDoc_,1);
  }
  else continue;
  min=0.0;
} // for i, NumberDocuments2
headD2=Desmarcar_Documento(headD2);
ImprimirQueries(headQ2);
for(i=1,min=0.0;i<=NQqueries;i++)
{
  for(j=0;j<k;j++)
  {
```

```
    min2=DISTANCIA_Q_CLUSTER(i,Clus_2[j].sgte,headQ2);
    if(min2>min)
    {
     idQuery=i;
     m=j;
    }
    else continue;
   }
   NDoc=ObtenerNumeroDocumentos_(Clus2[m].sgte,0);
   if(NDoc>=Arroba10)
   {
    headQD2=InsertarGlobalQD(headQD2,idQuery,headD2,
Clus_2[m].sgte,Clus2[m].sgte,headQ2);
    printf("El numero de DOCUMENTOS:%d, para el cluster m:%d, de la
query:%d\n",NDoc,m,idQuery);
   }
   else{
    printf("HAY UN ERROR, la cantidad de documentos:%d < que Arroba10\n",NDoc);
    stop=0;
    do{
     printf("Ingrese Arroba30 \n");
     scanf("%d",&Arroba30);
     printf("Ingrese Arroba10 \n");
```

```
    scanf("%d",&Arroba10);
    if(Arroba30>Arroba10 && NDoc>=Arroba10)
     stop=1;
   }while(stop==0);
  }
  min=0.0;
} // for i, NQqueries
if(NDoc>=Arroba30)
{
  N=ContarHeadQD(headQD2, 0);
  if(N>30)
  {
   N=Arroba30;
  }
  for(i=1;i<=N;i++,headQD2=headQD2->sgte)
  {
   HEAD2=INSERTAR_(HEAD2,headQD2,headQ2,headD2);
  }
  HEAD2=OrdenarQD2(HEAD2, Arroba30); // 27 de Noviembre
  mostrarHEAD(HEAD2);
  PENLACEQ HEAD2Z = NULL;
  HEAD2Z = copiarHEAD(HEAD2Z,HEAD2);
```

```
   mostrarHEAD(HEAD2Z);
   distribucionZHEAD(HEAD2Z);
   return(HEAD2Z);
  }
  else{
      printf("Cuidado, el número de documentos para esta consulta en este cluster no es suficiente\n");
  }
}
PENLACEQ colocarSoloRel(PENLACEQ H){//coloca la id de los documentos relevantes de las consultas en el vector soloDRel
  int i,cont=0;
  if (H != NULL)
  {
   for (i = 0; i < 30; ++i)
   {
    H->soloDRel[i]=0;
   }
   for (i = 0; i < 30; ++i)
   {
    if (H->vectorZ[i] == 1)
    {
     H->soloDRel[cont] = H->idDoc[i][0];
```

```
      cont++;
     }
   }
   colocarSoloRel(H->sgte);
 }
 return H;
}
DOCUMENT buscarDocPorId(DOCUMENT headD,int id){
 while(headD != NULL){
   if (headD->idDoc == id)
   {
     return(headD);
   }
   headD=headD->sgte;
 }
}
int buscarTer(PENLACEQ O, DOCUMENT headD, TERMS headT, char s[]){
 if (O != NULL)
 {
   int i,j;
   DOCUMENT d;
   TERMS t;
```

```
  for (i = 0; i < 30 && O->soloDRel[i] != 0; ++i)
  {
   d=buscarDocPorId(headD,O->soloDRel[i]);
   for (j = 0;  j< 30 && d->TermDoc[j] != 0 ; ++j)
   {
    t=buscarTerPorId(headT,d->TermDoc[j]);
    if(strcmp(t->term,s) == 0){
     return(1);
    }
   }
  }
  return(buscarTer(O->sgte,headD,headT,s));
 }
 else{
  return(0);
 }
}
PENLACEQ cantidadTerIguales(PENLACEQ O,PENLACEQ N,DOCUMENT headD,DOCUMENT
headD2, TERMS headT,TERMS headT2){
 if (N != NULL)
 {
  int i,j;
  DOCUMENT d;
```

```
   TERMS t;
   for (i = 0; i < 30 && N->soloDRel[i] != 0; ++i)
   {
    d=buscarDocPorId(headD2,N->soloDRel[i]);
    for (j = 0;  j< 30 && d->TermDoc[j] != 0 ; ++j)
    {
      t=buscarTerPorId(headT2,d->TermDoc[j]);//t es cada termino nuevo
      if(buscarTer(O,headD,headT,t->term) == 1){
        N->cantTermIguales++;//esta es la sumatoria de todos los doc de la query
        N->cantTermIgualesPorDocumento[i]++;
      }
    }
   }
   N->sgte = cantidadTerIguales(O,N->sgte,headD,headD2,headT,headT2);
  }
  return N;
}
TERMSR crearListaDeTerminosRelevantes(PENLACEQ ODOQ, TERMSR
listaTerRel,DOCUMENT headD, TERMS headT){
  if (ODOQ != NULL)
  {
   DOCUMENT d = NULL;
   TERMS t = NULL;
```

```
  for (int i = 0; i < 30 && ODOQ->soloDRel[i] != 0; ++i)
  {
   d=buscarDocPorId(headD,ODOQ->soloDRel[i]);
   for (int j = 0;  j< 30 && d->TermDoc[j] != 0 ; ++j)
   {
    t=buscarTerPorId(headT,d->TermDoc[j]);//t es cada termino antiguo
    if (listaTerRel == NULL)
    {
     listaTerRel=(TERMSR)malloc(sizeof(TERMR));
     strcpy(listaTerRel->term,t->term);
     listaTerRel->id = t->id;
     listaTerRel->topico = t->topico;
     listaTerRel->sgte = NULL;
    }
    else{
     TERMSR aux = listaTerRel;//aux lo utilizo solo para buscar la ultima posicion de la lista
sin perder el puntero cabecera
     TERMSR tr =(TERMSR)malloc(sizeof(TERMR));//tr es donde copio el termino relevante
     strcpy(tr->term,t->term);
     tr->id = t->id;
     tr->topico = t->topico;
     tr->sgte = NULL;
     while(aux->sgte != NULL){
```

```
          aux = aux->sgte;
        }
        aux->sgte = tr;
      }
    }
  }
  listaTerRel = crearListaDeTerminosRelevantes(ODOQ->sgte,listaTerRel,headD,headT);
 }
 return (listaTerRel);
}
void mostrarListadeTerminosRelevantes(TERMSR listaTerRel){
 if (listaTerRel != NULL)
 {
  printf("%s->",listaTerRel->term );
  mostrarListadeTerminosRelevantes(listaTerRel->sgte);
 }
 else{
  printf("\nfin de listaTerRel\n");
 }
}
DOCUMENT crearListaDeDocumentosRelevantes(DOCUMENT D, DOCUMENT
listaDocRel){//genera la lista de los documentos nuevos que podrian ser relevantes
 if (D != NULL)
```

```
{
  if (D->porcentajeTerRelevantes >= 0.5)
  {
    if (listaDocRel == NULL)
    {
      listaDocRel = (DOCUMENT)malloc(sizeof(DOC));
      for (int i = 0; i < 30; ++i)
      {
        listaDocRel->TermDoc[i] = D->TermDoc[i];
      }
      listaDocRel->idDoc = (D->idDoc * -1);
      listaDocRel->Topico = D->Topico;
      listaDocRel->Marcado = D->Marcado;
      listaDocRel->cantidadTerRelevantes = D->cantidadTerRelevantes;
      listaDocRel->porcentajeTerRelevantes = D->porcentajeTerRelevantes;
      listaDocRel->sgte = NULL;
    }
    else{
      DOCUMENT aux = listaDocRel;
      DOCUMENT listaAux =(DOCUMENT)malloc(sizeof(DOC));
      while(aux->sgte != NULL){
        aux = aux->sgte;
```

```
      }
      for (int i = 0; i < 30; ++i)
      {
       listaAux->TermDoc[i] = D->TermDoc[i];
      }
      listaAux->idDoc = (D->idDoc * -1);
      listaAux->Topico = D->Topico;
      listaAux->Marcado = D->Marcado;
      listaAux->cantidadTerRelevantes = D->cantidadTerRelevantes;
      listaAux->porcentajeTerRelevantes = D->porcentajeTerRelevantes;
      listaAux->sgte = NULL;
      aux->sgte = listaAux;
    }
   }
   listaDocRel = crearListaDeDocumentosRelevantes(D->sgte,listaDocRel);
 }
 return(listaDocRel);
}
int distribucionExponencial(){
 int NumberTopics = 3;
 float Lambda = 1.5;
 int i,j,y,stop1,NTermsbyDoc,m;
```

```
 float var,var1,inic,u_,FijoInterval,x,u;
 struct Exponencial *A = (struct Exponencial *)malloc(NumberTopics*sizeof(struct
Exponencial ));
 var=(1.0)/(float)NumberTopics;
 var1=var;
 for(i=0, inic=0.0 ;i<NumberTopics;i++)//nose que hace este for
 {
  A[i].ValorEnt=i+1;
  A[i].ValorDis=Lambda*exp(-1.0*(i+1)*Lambda);
  A[i].L_izq=inic;
  A[i].L_der=var1;
  inic=var1;
  var1=var1+var;
 }
 for(j=NumberTopics-1;j>=0;j--)
 {
  if(j==NumberTopics-1)
  {
   A[j].L_izq=0.0;
   A[j].L_der=A[j].ValorDis;
  }
   else{
    A[j].L_izq=A[j+1].ValorDis;
```

```
      A[j].L_der=A[j].ValorDis;
    }
  }
  FijoInterval=Lambda*exp(-1.0*1*Lambda); //1 porque es el valor maximo que puede tomar (0-1)
  NTermsbyDoc=rand()%15+15;//%25+15 es el minimo
  u_=rand()%101;
  u_=u_/100;
  u=u_* FijoInterval;
  for(m=0,stop1=0;m<NumberTopics && stop1==0;m++)  // Aqui hay que odernar de acuerdo a dis
   if(u>=A[m].L_izq && u<A[m].L_der && m!=(NumberTopics-1))
   {
    x=(log(A[m].L_der))*((-1)*(1.0/Lambda));
    y=ceil(x); //esto es lo que tenia antes, porque no estaba con todos los decimales
    stop1=1;
   }
   else if(m==(NumberTopics-1))
   {
    x=(log(FijoInterval))*((-1)*(1.0/Lambda));
    y=ceil(x);
    stop1=1;
   }
```

```
   return(y);
}
PENLACEQ aumentarContadorPorId(PENLACEQ head, int id){
 if (head != NULL)
 {
   if (head->idQuery == id)
   {
    head->contador++;
   }
   else{
    head->sgte = aumentarContadorPorId(head->sgte, id);
   }
 }
 return head;
}
void mostrarContador(PENLACEQ head){
 if (head != NULL)
 {
   printf("idQuery=%d contador=%d\n",head->idQuery,head->contador);
   mostrarContador(head->sgte);
 }
 else{
```

```
   printf("\nfin de contador\n");
  }
}
DOCUMENT cambiarIdDoc(DOCUMENT D, int NumberDocuments){// cambia el id de los documentos nuevos, para que los id sigan desde el ultimo id de los doc antiguos
  if (D != NULL)
  {
   D->idDoc = (NumberDocuments + 1);
   D->sgte = cambiarIdDoc(D->sgte,NumberDocuments+1);
  }
  return D;
}
int buscarIdTer(TERMS headT, TERMS t){
  if(headT != NULL)
  {
   if(strcmp(headT->term,t->term) == 0)
   {
    return(headT->id);
   }
   buscarIdTer(headT->sgte,t);
  }
  else{
   return 0;//retorna 0 si termino t no esta en la lista de terminos headT
```

```
 }
}
int estaIdDoc(short unsigned int soloDRel[], int id){
 for (int i = 0; i < 30 && soloDRel[i] != 0; ++i)
 {
  if (soloDRel[i] == id)
  {
   return(1);
  }
 }
 return(0);
}
void algoritmoProbabilistico(int AuxOld[], int Arroba30, DOCUMENT headD, DOCUMENT
listaDocRel ,int NumberDocuments, short unsigned int soloDRel[])
{//esta funcion simula los juicios de usuarios para los documentos nuevos y mantiene la
relevancia de los documentos antiguos
 int Aux[30],num,random,i,stop;
 float p,p2;
 DOCUMENT d;
 for (i = 0; i < 30 && AuxOld[i] != 0; ++i)
 {
  stop = 0;
  if (AuxOld[i] <= NumberDocuments)
```

```
  {
    if (estaIdDoc(soloDRel,AuxOld[i]) == 1)// este if es para dejar en 1 los doc que ya se que
son relevantes
    {
      Aux[i] = 1;
      stop = 1;
      continue;
    }
    else{
      Aux[i] = 0;
      stop = 1;
      continue;
    }
    d = buscarDocPorId(headD,AuxOld[i]);
  }
  else{
   d = buscarDocPorId(listaDocRel,AuxOld[i]);
  }
  if (stop == 0)
  {
    p = d->porcentajeTerRelevantes;
    p2 = 51.0 - ((float)i) * 1.7;
    p2 = p2 / 100.0;
```

```
    num = ceil(((p * 100.0) + (p2 * 100.0)));
    random = (rand() % 100) + 1;
    if (random <= num)
    {
      Aux[i] = 1;
    }
    else{
      Aux[i] = 0;
    }
   }
  }
  for(i=0;i<30;i++)
    AuxOld[i]=Aux[i];
} // fin del algoritmo
PENLACEQ relevanciaParaNuevosDocumentos(PENLACEQ P,PENLACEQ ODOQC, DOCUMENT
headD, DOCUMENT listaDocRel, int NumberDocuments){
  if (P != NULL)
  {
    int vector[30],i;
    for (i = 0; i < 30; ++i)
    {
      vector[i] = P->idDoc[i][0];
    }
```

```
   algoritmoProbabilistico(vector,30,headD,listaDocRel,NumberDocuments,ODOQC-
>soloDRel);
   for (i = 0; i < 30 && P->idDoc[i][0] != 0; ++i)
   {
     P->vectorZ[i] = vector[i];
   }
   P->sgte = relevanciaParaNuevosDocumentos(P->sgte,ODOQC-
>sgte,headD,listaDocRel,NumberDocuments);
 }
 return P;
}
int estaIdTer(TERMSR listaTerRel, int id){
 if (listaTerRel != NULL)
 {
   if (listaTerRel->id == id)
   {
     return 1;
   }
   else
   return(estaIdTer(listaTerRel->sgte,id));
 }
 else{
   return 0;
```

```
 }
}
DOCUMENT agregarPorcentaje(DOCUMENT headD, TERMSR listaTerRel){//agregar
porcentaje y ter relevantes a doc antiguos
 if (headD != NULL)
 {
  int contador = 0,totalTerminos = 0;
  for (int i = 0; i < 30 && headD->TermDoc[i] != 0; ++i)
  {
   totalTerminos++;
   if(estaIdTer(listaTerRel,headD->TermDoc[i]) == 1){
    contador++;
   }
  }
  headD->cantidadTerRelevantes = contador;
  headD->porcentajeTerRelevantes = ((float)(((float)contador)/((float)totalTerminos)));
  headD->sgte = agregarPorcentaje(headD->sgte,listaTerRel);
 }
 return headD;
}
int estaDoc(DOCUMENT D, int id){
 while(D != NULL){
  if (D->idDoc == id)
```

```
  {
    return(1);
  }
  D=D->sgte;
 }
 return(0);
}
```

```
DOCUMENT obtenerDocRelNoRepetidos(PENLACEQ P, DOCUMENT D){//obtiene todos los documentos que son relevantes, de todas las consultas
 if (P != NULL)
 {
  DOCUMENT aux = NULL;
  for (int i = 0; i < 30 && P->soloDRel[i] ; ++i)
  {
   if (D == NULL)
   {
    D=(DOCUMENT)malloc(sizeof(DOC));
    D->idDoc=P->soloDRel[i];
    D->sgte=NULL;
   }
   else{
    //if (estaDoc(D,P->soloDRel[i]) == 0)//este if es para eliminar los repetidos
    //{
```

```
        aux = D;
        while(aux->sgte!=NULL){
          aux=aux->sgte;
        }
        aux->sgte=(DOCUMENT)malloc(sizeof(DOC));
        aux->sgte->idDoc = P->soloDRel[i];
        aux->sgte->sgte = NULL;
       //}
     }
   }
    D = obtenerDocRelNoRepetidos(P->sgte,D);
 }
 return D;
}
int main()
{
 srand(time(NULL));
 int i,j,k,aux,m,NTermst,NTopics,NumberDocuments,b,idDoc,stop;
 int NDoc;
 float min,min2,Lambda=1.5;//1.0,1.5
 struct Lista *Clus;
 unsigned short int TermDoc_[30],s;
```

```
 struct Lista_ *Clus_;
 DOCUMENT headD=NULL;
 TERMS headT=NULL;
 printf("Ingrese el numero de palabras a generar\n");
 scanf("%d",&NTermst);//se ingresa por teclado el numero de terminos antiguos
 headT= GenerarTERMS(NTermst,headT);
 //se ingresa el numero de terminos, topicos y documentos nuevos
 int NTermst2=1400,NTopics2=70,NumberDocuments2=700;//NumberDocuments2 debe ser mayor o igual a NTopics2 y ademas multiplo de éste
 DOCUMENT headD2=NULL;
 TERMS headT2=NULL;
 headT2= GenerarTERMS2(NTermst2,headT2,headT);
 headT2=GenerarTopics_(NTermst2,NTopics2,headT2);
 printf("Ingrese el numero de topicos\n");
 scanf("%d",&NTopics);//se ingresa por teclado el numero de topicos antiguos, en nuestro caso se recomienda ingresar 70
 headT=GenerarTopics_(NTermst,NTopics,headT);
 stop=0;
 do{
  printf("Ingrese el numero de documentos, debe de ser >= Numero de topicos y proporcional a este\n");
  scanf("%d",&NumberDocuments);// se ingresa por teclado el numero de documentos antiguos
  if(NumberDocuments>=NTopics && (NumberDocuments%NTopics)==0)
```

```
    stop=1;
 }while(stop==0);
 headD=GenerarDocumentos(NumberDocuments,NTopics,NTermst,Lambda,headT);//genera documentos antiguos

headD2=GenerarDocumentos2(NumberDocuments2,NTopics2,NTermst2,Lambda,headT,headT2);//genera documentos nuevos
 printf("Ingrese el numero de cluster\n");// el numero de cluster a ingresar debe ser 1
 scanf("%d",&k);
 Clus=(struct Lista*)malloc(k*sizeof(struct Lista)); // esto contiene los cluster documentos
 Clus_=(struct Lista_*)malloc(k*sizeof(struct Lista_)); // esto contiene los cluster terminos y sera el centroide
 for(i=0,s=1,NDoc=1;i<k;i++)
 {
  Clus[i].sgte=NULL;
  Clus_[i].sgte=NULL;
  aux=(rand()%(NumberDocuments))+1;
  for(b=0;b<30;b++)
   TermDoc_[b]=0;
  Marcar_Documento(headD,aux,TermDoc_,1);
  for(j=0;j<30;j++)
  {
   if(TermDoc_[j]!=0)
   {
```

```
    if(EstaTermino(Clus_[i].sgte,TermDoc_[j])==0)
    {
     Clus_[i].sgte=InsertarTC(Clus_[i].sgte,TermDoc_[j],1,1);
    }
    else{   // Actualizar s e i
     s=NumeroTerminos(Clus_[i].sgte,TermDoc_[j]);
     s=s+1;
     Clus_[i].sgte=ActualizarTC(Clus_[i].sgte,s,TermDoc_[j],1);
    }
   }
   else continue;
  }// for j=0
  NDoc=1;
  Clus_[i].sgte=ActualizarNDoc(Clus_[i].sgte,NDoc);
  Clus[i].sgte=AgregarDocumento(Clus[i].sgte,aux);   // Insertar
 } // for i<k
 for(i=1;i<=NumberDocuments;i++)
 {
  for(j=0,min=0.0;j<k;j++)
  {
   if(Marcado_Documento(headD, i,1)==0)
            {
```

```
   min2=DISTANCIACLUSTER(i,Clus_[j].sgte,Clus[j].sgte,headD);
   min2=1;
   if(min2>min)
   {
    idDoc=i;
    m=j;
   }
   else continue;
  }
  else continue;
}
if(Marcado_Documento(headD, idDoc,1)==0)
{
          for(b=0;b<30;b++)
   TermDoc_[b]=0;
  Obtener_TDocumento(headD,idDoc,TermDoc_,1) ;
  for(j=0;j<30;j++)
  {
   if(TermDoc_[j]!=0)
   {
    if(EstaTermino(Clus_[m].sgte,TermDoc_[j])==0)
    {
```

```
        Clus_[m].sgte=InsertarTC(Clus_[m].sgte,TermDoc_[j],1,1);
       }
       else{// Actualizar s e i
        s=NumeroTerminos(Clus_[m].sgte,TermDoc_[j]);
        s=s+1;
        Clus_[m].sgte=ActualizarTC(Clus_[m].sgte,s,TermDoc_[j],1);
       }
      }
     else continue;
    }// for j=0
    NDoc=ObtenerNumeroDocumentos_(Clus[m].sgte,0);
    Clus_[m].sgte=ActualizarNDoc(Clus_[m].sgte,NDoc);
    Clus[m].sgte=AgregarDocumento(Clus[m].sgte,idDoc);// Insertar
    //MarcarDocumento
    Marcar_Documento2(headD,idDoc,TermDoc_,1);
   }
  else continue;
    min=0.0;
 }// for i, NumberDocuments
 headD=Desmarcar_Documento(headD);
 ENLACEQ headQC=NULL;
 ENLACEQ headQV=NULL;
```

```
 headQC=GenerarQueries_4(60,headQC,headD,NumberDocuments);//genera 60 consultas, que se consideraran como constantes

 headQV=GenerarQueries_4(30,headQV,headD,NumberDocuments);//genera 30 consultas, que se consideraran como variables

 PENLACEQ ODOQC=NULL;

 ODOQC = funcionParaNoRepetirHEAD(headD,NumberDocuments,headQC,60);//se aplican las consultas constantes a los documentos antiguos

 PENLACEQ ODOQV= NULL;

 ODOQV = funcionParaNoRepetirHEAD(headD,NumberDocuments,headQV,30);//se aplican las consultas variables a los documentos antiguos

 ODOQC = colocarSoloRel(ODOQC);//selecciona solo los id de los documentos relevantes

 TERMSR listaTerRel = NULL;

 listaTerRel = crearListaDeTerminosRelevantes(ODOQC,listaTerRel,headD,headT);// se crea la lista de terminos relevantes extraidos de los

//documentos antiguos, que se utilizaran al construir los documentos nuevos

 headD = agregarPorcentaje(headD,listaTerRel);

 DOCUMENT headD3 = NULL;

headD3=GenerarDocumentos2ContandoTerRelevantes(NumberDocuments2,NTopics2,NTermst,NTermst2,Lambda,headT,headT2,listaTerRel);//se generan los nuevos documentos

 DOCUMENT listaDocRel = NULL;

 listaDocRel = crearListaDeDocumentosRelevantes(headD3,listaDocRel);//lista de documentos nuevos que podrian ser relevantes, pues tienen mas del 50%

                                                    // de terminos relevantes
```

```
 int cantidadDocRel = ContarDocument(listaDocRel,0);
 int dExp;
 int numeroQuery;
 for (int i = 0; i < 120; ++i)
 {
  dExp = distribucionExponencial();
  if (dExp == 1)
  {
   numeroQuery = (rand() % 60) + 1;
   ODOQC = aumentarContadorPorId(ODOQC,numeroQuery);
  }
  else{
   numeroQuery = (rand() % 30) + 1;
   ODOQV = aumentarContadorPorId(ODOQV,numeroQuery);
  }
 }
 mostrarContador(ODOQC);
 printf("\n");
 mostrarContador(ODOQV);
 ODOQC = calcularPrecision(ODOQC);//aqui se calcula la precision de los documentos antiguos
 listaDocRel = cambiarIdDoc(listaDocRel,NumberDocuments);// cambios de id para seguir ocupando la funcionParaNoRepetirHEAD
```

```
 /**********************guardar documentos antiguos en archivo
txt*****************************/
 DOCUMENT DA=NULL;
 DA = headD;
 int cont=0;
 TERMS t = NULL;
 FILE *file;
 file = fopen("antiguos.txt","w");// en este archivo se almacenan los documentos antiguos
 if (file == NULL)
 {
  printf("error al abrir el archivo\n");
 }
 while(cont < NumberDocuments){
  fprintf(file, ".I %d\n",DA->idDoc);
  fprintf(file, ".W\n");
  for (i = 0; i < 30 && DA->TermDoc[i] != 0; ++i)
  {
   t = buscarTerPorId(headT,DA->TermDoc[i]);
   printf("idDoc=%d term=%s\n",DA->idDoc,t->term);
   if (i<=28)
   {
    if (DA->TermDoc[i+1] == 0)
    {
```

```
        fprintf(file, "%s",t->term);
      }
      else{
        fprintf(file, "%s ",t->term);
      }
    }
    else{
      fprintf(file, "%s",t->term);
    }
  }
  fprintf(file, ".\n\n");
  DA=DA->sgte;
  cont++;
}
fclose(file);
/**fin primer archivo**/
DOCUMENT DR = NULL;
DR = obtenerDocRelNoRepetidos(ODOQC,DR);
DOCUMENT DR2 = NULL;
DR2 = DR;
FILE *fileR;
fileR = fopen("antiguosR.txt","w");//en este archivo se almacenan los id de los documentos antiguos relevantes
```

```
 while(DR2 != NULL){
  fprintf(fileR, "%d ",DR2->idDoc);
  DR2=DR2->sgte;
 }
 fclose(fileR);
 /**********************guardar documentos antiguos en archivo txt
fin*****************************/
 DOCUMENT dAux = headD;//agregando los documentos nuevos relevantes a la lista de
antiguos documentos
 while(dAux->sgte != NULL){
  dAux = dAux->sgte;
 }
 dAux->sgte = listaDocRel;
 int cantidadDocUnidos = ContarDocument(headD,0);
 PENLACEQ
ODOQCUnidos=funcionParaNoRepetirHEAD(headD,cantidadDocUnidos,headQC,60);//aplican
do las consultas constantes a la union de documentos antiguos y nuevos relevantes
 ODOQCUnidos = colocarSoloRel(ODOQCUnidos);
 ODOQCUnidos = relevanciaParaNuevosDocumentos(ODOQCUnidos,ODOQC, headD,
listaDocRel, NumberDocuments);// se aplica distribucion de probabilidad  para simular
juicios de usuarios
                                                                                //
para los documentos nuevos
 ODOQCUnidos = calcularPrecision(ODOQCUnidos);//aqui se calcula la precision de la union
de los documentos antiguos con los documentos nuevos potencialmente relevantes
```

```
 mostrarPrecisionUnidos(ODOQC,ODOQCUnidos);
 printf("\n");
 mostrarPromedioPrecision(ODOQC,ODOQCUnidos);
/**********************guardar documentos unidos en archivo
txt*******************************/
 DOCUMENT DA2=NULL;
 DA2 = headD;
 int cont2=0;
 FILE *file2;
 file2 = fopen("unidos.txt","w");//en este archivo se almacenan los documentos unidos
 if (file2 == NULL)
 {
  printf("error al abrir el archivo\n");
 }
 while(cont2 < cantidadDocUnidos){
  fprintf(file2, ".I %d\n",DA2->idDoc);
  fprintf(file2, ".W\n");
  for (i = 0; i < 30 && DA2->TermDoc[i] != 0; ++i)
  {
   if (DA2->TermDoc[i] > 0)
   {
    t = buscarTerPorId(headT,DA2->TermDoc[i]);
   }
```

```
    else{
      t = buscarTerPorId(headT2,(DA2->TermDoc[i] * -1));
    }
    if (i<=28)
    {
      if (DA2->TermDoc[i+1] == 0)
      {
        fprintf(file2, "%s",t->term);
      }
      else{
        fprintf(file2, "%s ",t->term);
      }
    }
    else{
      fprintf(file2, "%s",t->term);
    }
  }
  fprintf(file2, ".\n\n");
  DA2=DA2->sgte;
  cont2++;
}
fclose(file2);
```

```
 /**fin primer archivo**/

 DOCUMENT DRN = NULL;

 DRN = obtenerDocRelNoRepetidos(ODOQCUnidos,DRN);

 DOCUMENT DRN2 = NULL;

 DRN2 = DRN;

 FILE *fileR2;

 fileR2 = fopen("unidosR.txt","w");//en este archivo se almacenan los id de los documentos
unidos relevantes

 while(DRN2 != NULL){

  fprintf(fileR2, "%d ",DRN2->idDoc);

  DRN2=DRN2->sgte;

 }

 fclose(fileR2);

 /**********************guardar documentos unidos en archivo txt
fin*****************************/

 return 1;

}// main
```