UNIVERSIDAD DEL BÍO-BÍO

Facultad de Ciencias Empresariales
Departamento de Sistemas de Información
Campus Concepción

NUEVOS ALGORITMOS PARA PRODUCIR Y BUSCAR PATRONES DE SUBGRAFOS EN BASES DE DATOS DE GRAFOS RDF

Trabajo de investigación presentado en conformidad a los requisitos para obtener el título de Ingeniero Civil en Informática

Reyes Fariña, Gert Arnaldo
PROFESOR GUÍA | CLAUDIO GUTIÉRREZ – **PROFESOR INFORMANTE** | TATIANA GUTIÉRREZ

Contenido

Lista de Fórmulas

Lista de Figuras

Capítulo 1: Introducción

El presente informe tiene por objetivo presentar la investigación realizada el año 2016 por el alumno tesista de la Universidad del Bío-Bío, Gert Reyes, con ayuda del docente Claudio Gutiérrez como profesor guía del proyecto.

1.1 Origen Del Tema

Este trabajo de investigación se desarrolla para dar conformidad a la actividad de Tesis de la Carrera de Ingeniería Civil Informática.

El tema fue propuesto por el docente de la Universidad del Bío-Bío, Claudio Gutiérrez, con el objetivo de generar descubrimientos y nuevas líneas de investigación en relación a la producción y búsqueda de patrones de subgrafos en bases de datos de grafos RDF.

1.2 Justificación

Con la llegada de la Web semántica, ha cambiado la forma en la que se visualiza la estructura global de la Internet. Anteriormente, en la Web 2.0, se entendía el concepto de un Usuario consultando una colección de información o base de datos, desde donde obtiene toda la información que necesite. Sin embargo, con el tiempo esto ha contribuido a una sobrecarga de información, que o necesariamente corresponde a la deseada. Actualmente, en la Web 3.0, se pretende que las máquinas sean capaces de "entender" la información que están procesando para así optimizar los resultados entregados. Para cumplir este propósito, la Web Semántica define el modelo RDF, una estandarización del W3C, que permite representar la Web como un gran grafo dirigido, donde cada nodo corresponde a un recurso determinado. Gracias a esta estandarización, podemos reducir el problema de la búsqueda de información en la Web a la búsqueda de patrones entre subgrafos RDF, tarea que se lleva a cabo principalmente a través del match o emparejamiento de subgrafos RDF.

Hoy en día, es posible encontrar varias aproximaciones de la Web semántica dedicadas a mejorar el desempeño del emparejamiento entre grafos RDF. Aunque dichas aproximaciones tienen un buen desempeño, este desempeño no siempre es aceptable cuando la cantidad de nodos RDF y grafos RDF son considerables. Es por esto que resulta necesario el estudio de nuevos algoritmos orientados a la producción y búsqueda de patrones de subgrafos en bases de datos de grafos RDF.

1.3 Objetivos Del Estudio.

1.3.1 Objetivo general

El objetivo principal de esta tesis es estudiar, analizar, diseñar e implementar algoritmos que permitan afrontar la problemática de la generación eficiente de subgrafos RDF y el match entre subgrafos RDF.

1.3.2 Objetivos específicos

- Estudiar la problemática de la generación de subgrafos a partir de grafos.
- Estudiar las características de los grafos RDF, así como del lenguaje SPARQL.
- Estudiar y analizar diferentes aproximaciones de isomorfismos de Grafos RDF.
- Analizar, diseñar e implementar algoritmos para dichas problemáticas.
- Contrastar nuestra propuesta con otras en la literatura.

1.4 Software y Hardware

El algoritmo fue desarrollado en el lenguaje de programación C y fue probado en un computador de escritorio con sistema operativo Windows 10 Pro de 60 bits, un procesador de 8 núcleos de 4.0 Ghz, 8192 MB de memoria RAM.

1.5 Descripción De Capítulos

El documento está dividido en seis capítulos, organizados de la siguiente manera:

- ✓ Capítulo 1 En el primer capítulo se introduce el tema de investigación y su origen, se presenta la justificación y los objetivos del estudio.
- ✓ Capítulo 2 Tiene como objetivo presentar la información base necesaria para comenzar a entender los grafos RDF. Se abarcan los conceptos de grafo, subgrafo e isomorfismo.
- ✓ Capítulo 3 Se presenta el modelo RDF y se definen los grafos RDF, isomorfismo de grafos y el lenguaje SPARQL.
- ✓ Capítulo 4 Se expone el artículo en el que se basa la investigación y las impresiones del algoritmo propuesto en dicho artículo.
- ✓ Capítulo 5 Abarca el funcionamiento de la base de datos léxica WordNet, uno de los componentes fundamentales del algoritmo propuesto en el capítulo cuatro.
- ✓ Capítulo 6 Resume el análisis, diseño, implementación y pruebas realizadas en base al algoritmo propuesto en el capítulo cuatro, sus mejoras y sus resultados.

Capítulo 2: Grafos, Subgrafos e Isomorfismo.

En este capítulo se presenta la información preliminar necesaria para comprender el modelo RDF. Se definen los conceptos de grafos simples, dirigidos, etiquetados y bipartitos, de los cuales extiende el modelo de grafos RDF. Se presenta además las principales formas de representación de grafos, a través del uso de matrices y listas de incidencia y adyacencia. Luego, se presenta la definición de subgrafos como un subconjunto de un grafo y finalmente se expone el isomorfismo de grafos, que será fundamental para comprender el isomorfismo de grafos RDF.

2.1 Grafos

Para las Ciencias de la Computación y la matemática, un grafo es una representación gráfica de diversos puntos que se conocen como nodos o vértices, los cuales se encuentran unidos a través de líneas que reciben el nombre de aristas.

2.1.1 Definiciones

Un grafo $G = (V, E)$ consiste en un conjunto finito V cuyos miembros se llaman vértices y una familia finita $E \subset VxV$ de pares no ordenados de vértices a cuyos elementos llamaremos aristas o arcos y denotaremos por (u, v) donde $(u, v) \in V$ (Rodes, 2005).

El número de vértices, es decir la cardinalidad del conjunto V se denomina orden del grafo y se denota por $|V|$. Por lo general se utiliza n para denotar el orden de G. El número de aristas, es decir la cardinalidad de E, se denomina tamaño del grafo y se denota por $|E|$. Por lo general se utiliza m para denotar el tamaño de G.

Dado un grafo $G = (V, E)$:

a) se llama bucle o lazo (ver Figura 1) a toda arista de la forma (v, v),

b) se llaman aristas múltiples a las aristas que aparecen repetidas en E,

c) se dice que dos vértices son adyacentes si están unidos por una arista,

d) se dice que dos aristas son adyacentes si tienen un vértice en común,

e) se dice que una arista y un vértice son incidentes si el vértice es extremo de la arista,

f) se dice que un vértice es aislado si no es adyacente a ningún otro vértice (ver Figura 1),

h) se llama grado de un vértice al número de aristas incidentes en el, contando los bucles como dos aristas y se denota como $\deg(v) = 2$, para el vértice v (ver Figura 1).

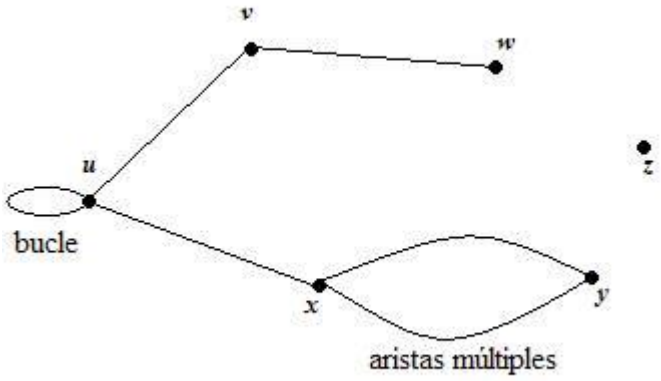


Figura 1 Grafo con bucle y aristas múltiples.

En el grafo anterior u y v son vértices adyacentes, (u, v) y (v, w) son aristas adyacentes, z es un vértice aislado.

2.1.1.1 Grafo Simple

Un grafo es *simple* si a lo sumo existe una arista uniendo dos vértices cualesquiera (Figura 2).

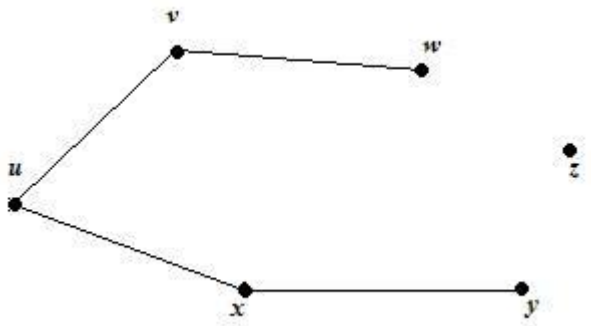


Figura 2 Grafo simple.

2.1.1.2 Grafos dirigidos

Un grafo *dirigido* o *digrafo*, es una estructura de datos formada por un conjunto de vértices y un conjunto de arcos, donde cada arco se especifica mediante un par ordenado de vértices, es decir, un arco (u, v) determina un arco que va del vértice u al vértice v (Jeltsch, 1998). En el digrafo de la Figura 3, el conjunto de vértices es $\{1,2,3,4\}$, y el conjunto de arcos es $\{(1,2), (2,4), (4,3), (1,3), (3,2)\}$.

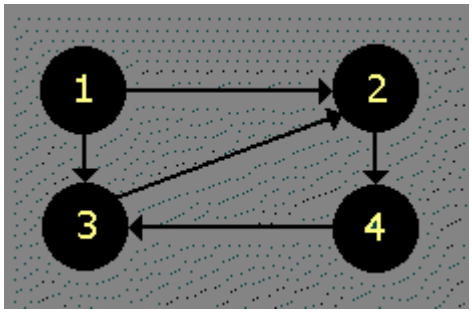


Figura 3 Grafo Dirigido.

2.1.1.3 Grafos etiquetados

Un grafo se dice *etiquetado* cuando sus aristas contienen datos (etiquetas). Una etiqueta puede ser un nombre, costo o un valor de cualquier tipo de dato.

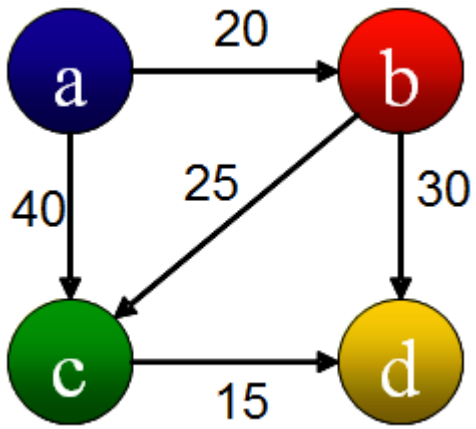


Figura 4 Grafo etiquetado.

En la Figura 4, se han etiquetado tanto los vértices (letras) como las aristas (números).

2.1.1.4 Grafo bipartito

El grafo *bipartito* es aquel con cuyos vértices pueden formarse dos *conjuntos disjuntos*[1] de modo que no haya adyacencias entre vértices pertenecientes al mismo conjunto (Bonilla, s.f.).

[1] Conjuntos disjuntos: Conjuntos que no tienen ningún elemento en común.

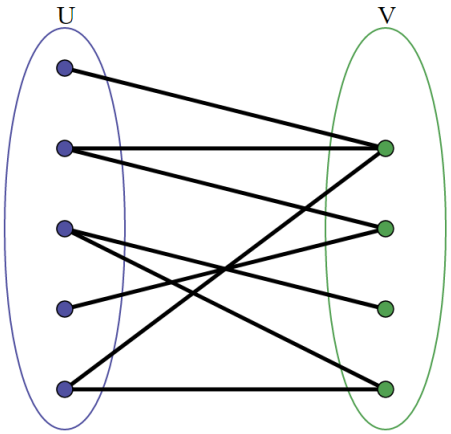


Figura 5 Grafo Bipartito

En la Figura 5, los nodos del grafo no dirigido, pueden agruparse en los conjuntos U y V.

2.1.2 Representación de grafos

Las principales formas de representar los grafos, es a través del uso de matrices o listas. Dentro de las matrices, tenemos las *matrices de incidencia* y las *matrices de adyacencia*. De la misma forma, en el caso de las Listas, podemos encontrar las *listas de incidencia* y las *listas de adyacencia*.

2.1.2.1 Matriz de incidencia

Una matriz de incidencia representa las relaciones binarias entre dos elementos, en nuestro caso entre un nodo y una arista de un grafo. Para construir la matriz de incidencia a partir de un grafo se debe realizar una matriz de $n\ x\ a$, donde n es el número de nodos o vértices y a es el número de aristas. En esta matriz, las columnas representan las aristas y las filas, los vértices. Para una matriz de incidencia A, cada celda $A[i][j]$ puede valer:

- 0 - si el vértice i no es incidente con la arista j, y
- 1 - si el vértice i es incidente con la arista j.

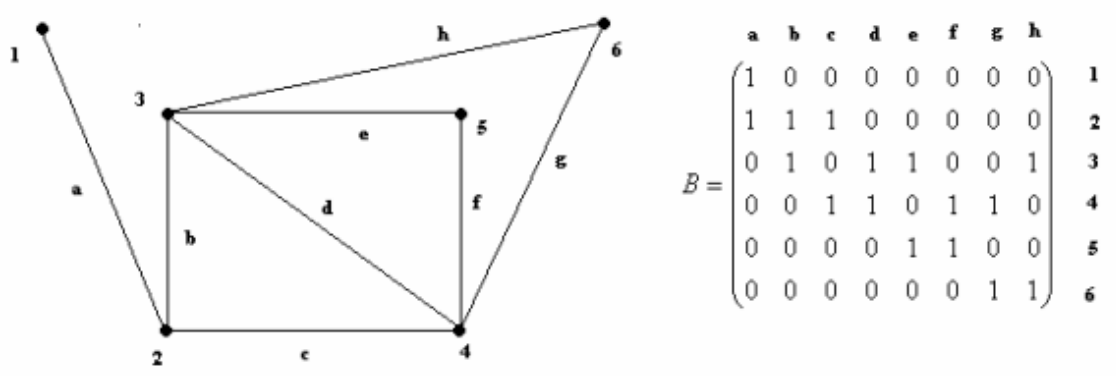


Figura 6 Matriz de Incidencia B del grafo a su izquierda.

2.1.2.2 Matriz de adyacencia

Dado un grafo $G = (V, E)$ con n vértices $\{v_1, \dots, v_n\}$ su matriz de adyacencia es la matriz de orden $n \times n$, $A(G) = (a_{ij})$ donde a_{ij} es el número de aristas que unen los vértices v_i y v_j.

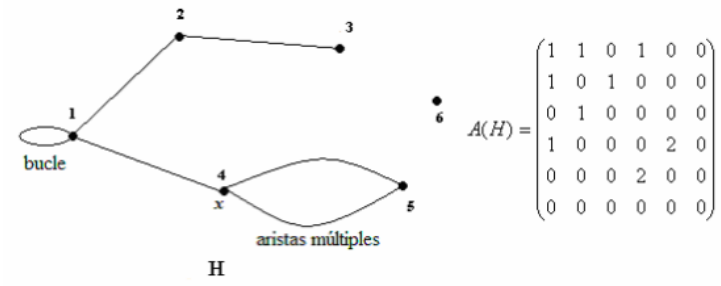


Figura 7 Matriz de Adyacencia A(H).

La matriz de adyacencia de un grafo es simétrica, si el grafo es simple entonces la matriz de adyacencia contiene solo ceros y unos (matriz binaria) y la diagonal esta compuesta sólo por ceros. Cuando el grafo no es simple (ver Figura 7) aparecen números mayores a 1 en su matriz de adyacencia (representando el número de aristas entre los vértices), y su diagonal no está compuesta sólo por ceros.

2.1.2.3 Lista de incidencia

Las aristas son representadas con un vector de pares (ordenados, si el grafo es dirigido), donde cada par representa una de las aristas.

2.1.2.4 Lista de adyacencia

Cada vértice tiene una lista de vértices los cuales son adyacentes a él (Figura 8). Permite búsquedas más rápidas al costo de almacenamiento extra.

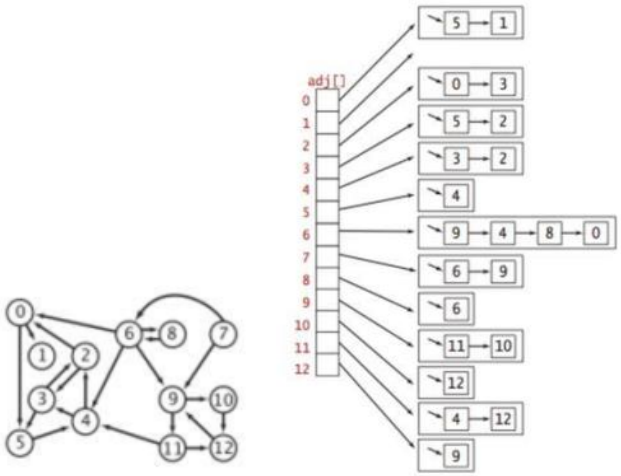


Figura 8 Grafo dirigido y su lista de adyacencia correspondiente.

2.2 Subgrafos

Un subgrafo se define como un grafo con vértices y aristas que son un subconjunto de un grafo padre. Llamaremos subgrafo de un grafo $G = (V, E)$ a cualquier otro grafo $G' = (V', E')$ con $E' \subset E$ y $V' \subset V$.

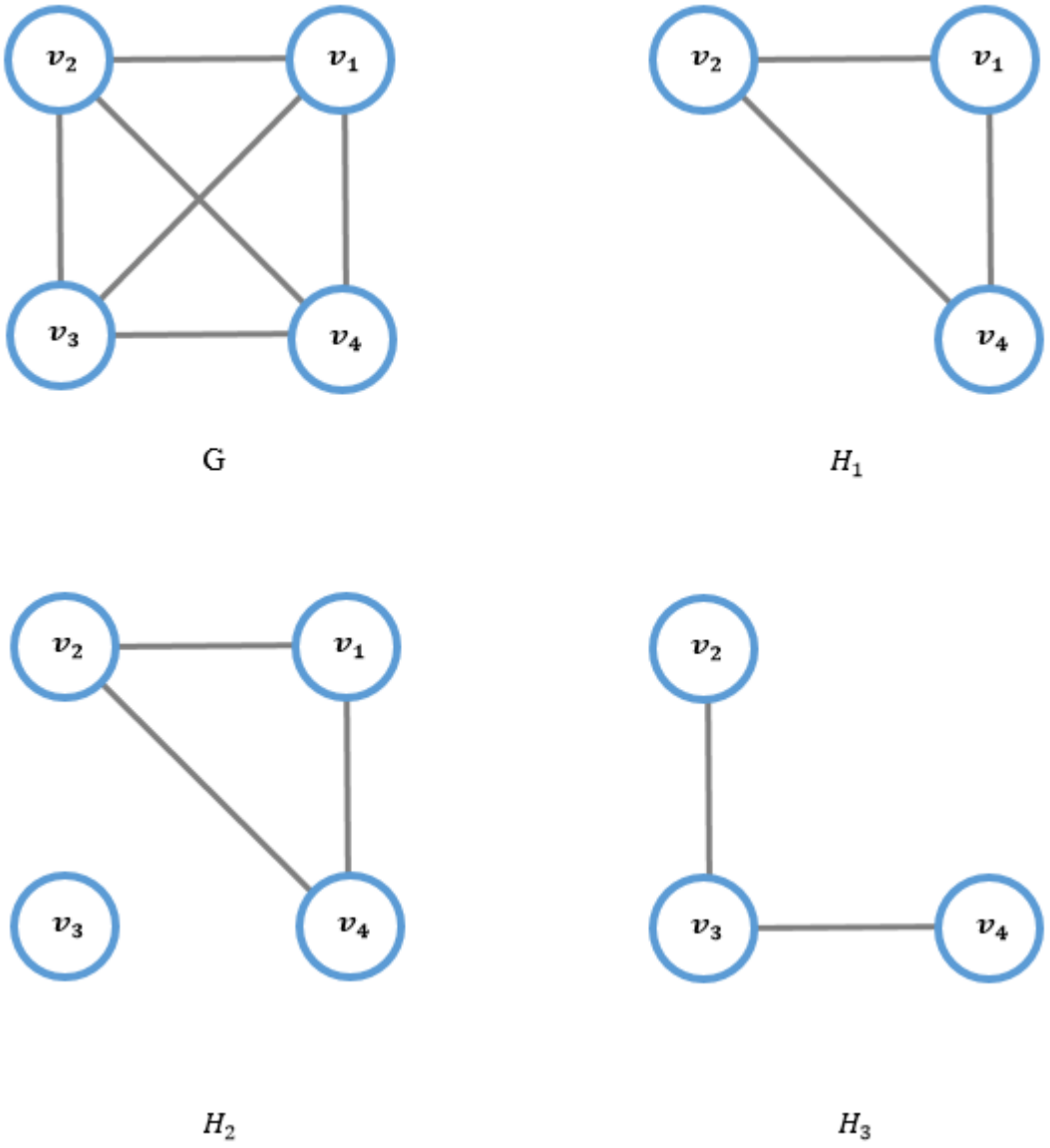


Figura 9 Subgrafos

$$G = (\{v_1, v_2, v_3, v_4\}, \{v_1v_2, v_1v_3, v_1v_4, v_2v_3, v_2v_4, v_3v_4\})$$

$$H_1 = (\{v_1, v_2, v_4\}, \{v_1v_2, v_1v_4, v_2v_4\})$$

$$H_2 = (\{v_1, v_2, v_3, v_4\}, \{v_1v_2, v_1v_4, v_2v_4\})$$

$$H_3 = (\{v_2, v_3, v_4\}, \{v_2v_3, v_3v_4\})$$

2.3 Isomosfismo De Grafos

2.3.1 Definición 1

Dos grafos $G_1 = (V_1, A_1)$ y $G_2 = (V_2, A_2)$ se dice que son isomorfos cuando existe una *biyección*[2] entre los conjuntos de sus vértices que conserva la adyacencia. Si los grafos G_1 y G_2 son isomorfos, notaremos $G_1 \sim G_2$.

2.3.2 Definición 2

Dos grafos, G_1 y G_2, son isomorfos si existe una correspondencia uno a uno entre los vértices de los grafos tal que todo par de vértices que son adyacentes en un grafo si y sólo si el correspondiente par de vértices son adyacentes en el otro grafo.

[2] Existe una biyección si todos los elementos del conjunto de salida tienen una imagen distinta en el conjunto de llegada, y a cada elemento del conjunto de llegada le corresponde un elemento del conjunto de salida.

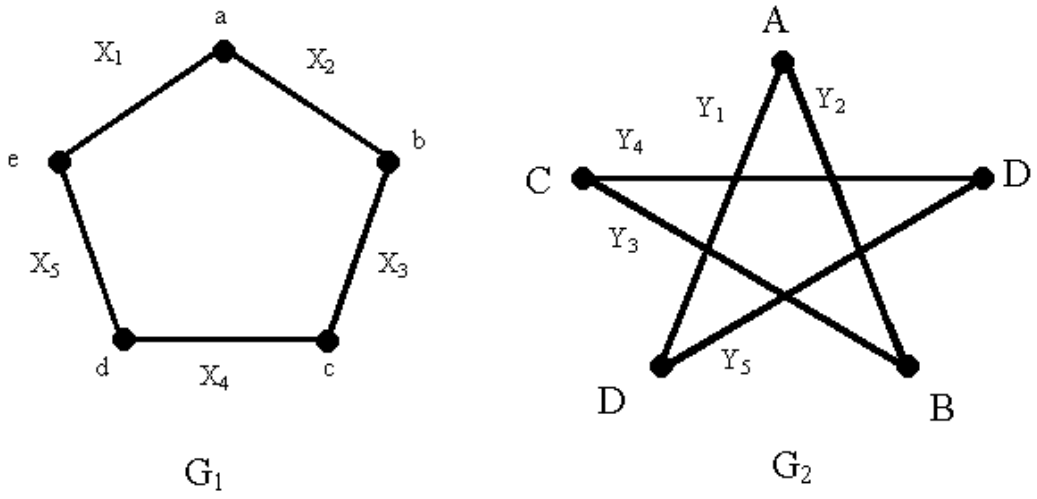


Figura 10 Grafos Isomorfos.

Para los grafos de la Figura 10, un isomorfismo estaría definido por:

$$f(a) = A$$

$$f(b) = B$$

$$f(c) = C$$

$$f(d) = D$$

$$f(e) = E$$

$$\text{y } g(X_i) = Y_b,\ i = 1, 2, \ldots, 5.$$

Los grafos G_1 y G_2 son isomorfos sí y sólo sí para alguna ordenación de vértices y lados sus matrices de incidencia son iguales (ver Figura 11).

$$G_1 = \begin{array}{c|ccccc} & X_1 & X_2 & X_3 & X_4 & X_5 \\ a & 1 & 1 & 0 & 0 & 0 \\ b & 0 & 1 & 1 & 0 & 0 \\ c & 0 & 0 & 1 & 1 & 0 \\ d & 0 & 0 & 0 & 1 & 1 \\ e & 1 & 0 & 0 & 0 & 1 \end{array}$$

$$G_2 = \begin{array}{c|ccccc} & Y_1 & Y_2 & Y_3 & Y_4 & Y_5 \\ A & 1 & 1 & 0 & 0 & 0 \\ B & 0 & 1 & 1 & 0 & 0 \\ C & 0 & 0 & 1 & 1 & 0 \\ D & 0 & 0 & 0 & 1 & 1 \\ E & 1 & 0 & 0 & 0 & 1 \end{array}$$

Figura 11 Matrices de incidencia de los grafos de la figura 10.

Propiedades

Si G_1 y G_2 son grafos isomorfos:

- Tienen el mismo número de vértices y aristas
- Tienen el mismo número de componentes conexas (Si uno es conexo el otro también)
- La lista formada por los grados de los vértices es la misma (vértices correspondientes tienen el mismo grado)
- Tienen el mismo número de lazos, si es que tienen.
- Así mismo, si uno posee un camino euleriano (o hamiltoniano), entonces el otro también.

El estudio de los conceptos abordados en este capítulo prepara al lector para entender el modelo RDF, las definiciones de grafo y subgrafo son fundamentales para comprender el isomorfismo de grafos, el cual se verá aplicado de forma práctica en el próximo capítulo.

Capítulo 3: RDF y Grafos RDF

Este capítulo comprende la definición de la estandarización RDF, el modelo de grafos RDF y sus componentes. Se abarcan los conceptos básicos de tripleta, sujeto, predicado y objeto. Se presentan además distintas formas de escribir la sintaxis de un documento RDF y la aplicación del isomorfismo a los grafos RDF. Finalmente se incluye el estudio obligado del lenguaje SPARQL, como recomendación del W3C.

3.1 RDF

RDF es un modelo desarrollado y estandarizado por la W3C (W3C, W3C, 2017) para el intercambio de información en la Web. RDF extiende la estructura de hipervínculos de la Web, usando identificadores de recursos para nombrar las relaciones entre las cosas. La estructura de hipervínculo forma un grafo etiquetado y dirigido, donde las aristas representan el hipervínculo etiquetado entre dos recursos, representados por los nodos del grafo. Esta forma de grafo es el modelo mental más sencillo para visualizar la estructura RDF y es a lo que nos dedicaremos en este capítulo. La figura 12 corresponde a un grafo RDF que representa el hipervínculo entre dos recursos: *Sujeto* y *Objeto*.

Según la W3C, The Resource Description Framework (RDF), como su nombre lo indica, es un Framework para describir recursos y se usa para representar información en la Web (W3C, 2014).

3.2 Modelo De Grafos RDF

La estructura principal de la sintaxis RDF es un conjunto de tripletas, cada una consistente de un sujeto, un predicado y un objeto. Un conjunto de dichas tripletas es lo que se conoce como Grafo RDF. Un grafo RDF puede ser visualizado como un diagrama de un nodo y arcos dirigidos, en el que cada tripleta está representada como un hipervínculo nodo-arco-nodo (ver Figura 12).

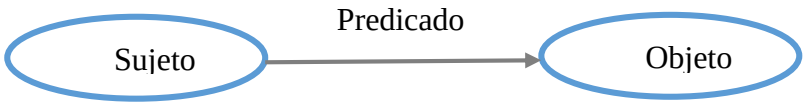


Figura 12 Grafo RDF con dos nodos (Sujeto y Objeto) y un arco conectándolos (Predicado).

Puede haber tres tipos de nodos en un grafo RDF: *IRIs, literales* y *nodos en blanco*.

Cualquier *IRI* o *literal* describe algo en el mundo, que se conoce como recursos. Todo puede ser un recurso, incluyendo cosas físicas, documentos, conceptos abstractos, números o cadenas de caracteres; el término es sinónimo de entidad.

URI, se traduce como un *Identificador de Recursos Uniformes* y es una cadena de caracteres que sirve para identificar un recurso determinado. Una de las URIs más conocidas es la URL, traducida como Localizador de Recursos Uniformes, que es una cadena de caracteres con una codificación y una sintaxis determinada, usada para identificar páginas Webs.

Un *URI* puede bien ser un URN o un URL. Una forma clara de ejemplificarlo es comparar un URN con el nombre de una persona, mientras que el URL puede ser comparado con la dirección de la casa de esa persona (Wikipedia, 2017). Vale decir: un URN identifica un ítem y un URL provee un método para encontrarlo.

Por otra parte *IRI* se define como un *Identificador Internacional de Recursos* y es considerado la generalización de las URIs, pues estas últimas sólo soportan codificación ASCI, mientras que las IRI soportan cracteres internacionales.

Los recursos identificados por *literales* pueden ser vistos como cadenas de caracteres que contengan cierta información precisa sobre algo, ya sea palabras o números.

Para objeto de este estudio trabajaremos con *URIs*, representadas por *URLs* y *literales*.

Una tripleta RDF afirma una relación, indicada por el predicado, que se mantiene entre los recursos identificados por el Sujeto y el Objeto. Esta afirmación dada por una tripleta RDF se conoce como ***RDF Statement*** (Afirmación RDF).
El predicado mismo es un IRI y expresa una propiedad, por lo que un recurso puede ser visto como una relación binaria. (Las relaciones entre más de dos recursos sólo pueden expresarse de forma indirecta) (W3C, Defining N-ary Relations on Semantic Web, 2006).

Un grafo RDF se define, entonces, como un conjunto de tripletas RDF, **donde** cada tripleta consta de 3 componentes:

1) El sujeto, que puede ser un IRI o un nodo en blanco.
2) El predicado, que puede ser un IRI.
3) El objeto, que puede ser un IRI, un literal o un nodo en blanco.

Las tripletas RDF suelen escribirse, por convención, nombrando primero al $Sujeto$, luego el $predicado$ y, al final, el $objeto$.

El conjunto de nodos de un grafo RDF corresponde al conjunto de sujetos y objetos de las tripletas, en el grafo; Mientras que el conjunto de los arcos, correspondería al conjunto de los predicados del grafo RDF.

3.3 Documentos y Sintaxis RDF

Un documento RDF es un archivo que codifica un Grafo RDF en una sintaxis RDF concreta. Esta codificación se conoce como serialización.

Hay varios lenguajes especializados en serialización RDF, por ejemplo: Turtle, N-Triples, N-Quads, Json-ld, N3 ó RDF/XML.

3.3.1 Turtle

(Terse RDF Triple Languaje) es un formato para expresar información en RDF. Es bastante simple, pues utiliza la forma básica de las tripletas, cada una de las cuales consiste en un sujeto, un predicado y un objeto.

```
<http://example.org/person/Mark_Twain>
   <http://example.org/relation/author>
   <http://example.org/books/Huckleberry_Finn>
```

Figura 13 Ejemplo de serialización Turtle.

3.3.2 N-Triples

Está basado en un sistema de líneas y texto plano y es un subconjunto del formato Turtle. Fue diseñado para ser más simple que la notación Turtle y N3, y por lo tanto fácil para los software de leer y generar.

```
<http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ↵
   <http://xmlns.com/foaf/0.1/Document> .
<http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://purl.org/dc/terms/title> "N-Triples"@en-US .
<http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://xmlns.com/foaf/0.1/maker> _:art .
<http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://xmlns.com/foaf/0.1/maker> _:dave .
_:art <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
_:art <http://xmlns.com/foaf/0.1/name> "Art Barstow".
_:dave <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
_:dave <http://xmlns.com/foaf/0.1/name> "Dave Beckett".
```

Figura 14 Ejemplo Serialización N-Triple.

3.3.3 N-Quads

Es una extensión del formato N-Triples con una opción de valor de contexto en la cuarta posición.

```
<http://one.example/subject1> <http://one.example/predicate1> <http://one.example/object1> <http://example.org/graph3> . # comments here
# or on a line by themselves
_:subject1 <http://an.example/predicate1> "object1" <http://example.org/graph1> .
_:subject2 <http://an.example/predicate2> "object2" <http://example.org/graph5> .
```

Figura 15 Ejemplo Notación N-Quads

3.3.4 N3 (Notation3)

N3 es un formato de serialización de modelos RDF, diseñado con el propósito de que fuera fácil de leer por el usuario. N3 es mucho más compacto y legible que la notación RDF/XML.

N3 tiene algunas características que van más allá de la serialización para modelos RDF, como el soporte para reglas basadas en RDF. El formato Turtle es un subconjunto simplificado y de uso específico para RDF.

```
@prefix dc: <http://purl.org/dc/elements/1.1/>.

<http://en.wikipedia.org/wiki/Tony_Benn>
  dc:title "Tony Benn";
  dc:publisher "Wikipedia".
```

Figura 16 Ejemplo de N3.

3.4 Isomorfismo De Grafos RDF

Según la definición de la W3C (W3C, RDF 1.1 Concepts and Abstract Syntax, 2014), dos grafos RDF: G y G' son isomorfos si hay una biyección M entre los conjuntos de nodos de los dos grafos, tal que:

1- M mapea nodos blancos a nodos blancos, si es que hay.

2- $M(lit)$ = lit para todos los literales RDF que son nodos de G.

3- $M(IRI)$ = IRI para todos los IRI que son nodos de G.

4- La tripleta (s, p, o) está en G sí y sólo sí la tripleta $(M(s), p, M(o))$ está en G'.

La definición de Isomorfismo de Grafos RDF, extiende de la definición de isomorfismo de Grafos, pero en la práctica puede ser interpretada como una correspondencia 1 a 1 entre los nodos de ambos grafos, donde a cada nodo del grafo G le corresponde un y sólo un nodo del grafo G', dicha correspondencia sería la biyección M.

3.5 SPARQL

SPARQL (Sparql Protocol and RDF Query Language) es un acrónimo recursivo que se traduce como Lenguaje de Protocolo Sparql y consulta RDF, y es un lenguaje estandarizado para la consulta de grafos RDF, normalizado por el RDF Data Access Working Group (W3C, s.f.) del World Wide Web Consortium (W3C). Se constituyó como recomendación oficial del W3C el 15 de enero de 2008 (W3C, SPARQL is a recommendation, 2008), y es por esta razón que se consideró digno de estudio para el desarrollo de esta investigación.

En un principio SPARQL únicamente incorpora funciones para la recuperación de sentencias RDF.

La mayoría de las consultas SPARQL contienen un conjunto de patrones de tripletas, el que se conoce como $patrón\ de\ grafo\ base$. Los patrones de tripletas son como las tripletas RDF, excepto que cada uno de los sujeto, predicado y objeto puede ser una variable.

La idea básica de SPARQL es consultar una base de datos y devolver un resultado; para poder realizar esta tarea el patrón de grafo base debe hacer un match con un subgrafo de la base RDF, y esto se cumple cuando los términos RDF que forman el subgrafo, pueden ser sustituidos por las variables y el resultado es un grafo RDF equivalente al subgrafo (W3C, SPARQL Query Language for RDF, 2008).

En la Figura 17, se muestra cómo se realiza una consulta en SPARQL, donde se solicita el título del libro $book1$.

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .
}
```

Figura 17 Ejemplo Query en SPARQL.

Muy similar a SQL, la primera clausula, SELECT, identifica las variables que aparecerán en el resultado. WHERE, en cambio, provee la información para realizar el match entre el patrón de grafo base contra la base de datos RDF.

El patrón de grafo base en este ejemplo consiste de un solo patrón de tripleta con una sola variable (título) en la posición del objeto.

La información a buscar en la base de datos RDF tiene la siguiente forma:

```
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial" .
```

Figura 18 Información en la base de datos RDF del ejemplo.

La consulta, aplicada a la base de datos, tiene como único resultado:

title
"SPARQL Tutorial"

Figura 19 Resultado de la consulta a la base de datos del ejemplo.

En términos generales, SPARQL funciona de forma similar a SQL, por lo que no se considera necesario describirlo en este estudio, pues dicha especificación se presenta como un manual de usuario para ser consultado al momento de trabajar con el lenguaje.

En este capítulo se definieron los elementos básicos de la estructura RDF y el modelo de grafos RDF, la sintaxis de sus documentos. El dominio de estos conceptos es una herramienta importante para entender los artículos y publicaciones sobre la producción y búsqueda de patrones de subgrafos en bases de datos de grafos RDF. También se investigó sobre el lenguaje Sparql, como recomendación del W3C, sin embargo, no es lo suficientemente versátil como para desarrollar un proyecto de este tipo, pues básicamente es sólo un lenguaje de consultas sobre grafos RDF.

Capítulo 4: Algoritmos De Estudio.

En el presente capítulo estudiaremos una de las publicaciones que más llamó nuestra atención y las razones de porqué decidimos centrarnos en él al desarrollar esta investigación.

Para una mejor claridad, estimamos presentar primero la propuesta de la publicación y luego exponer nuestras conclusiones.

A Similarity-oriented RDF Graph Matching Algorithm for Ranking Linked Data, traducido sería, *Un algoritmo para el emparejamiento entre grafos RDF, orientado a la similaridad, para ponderar información hipervinculada.*

Este artículo fue publicado por Dehai Zhang, Jun He, Yang Dong, Tianlong Song y Xingwei Shi, de la Uniersidad de Yunnan, en Kunming, China; en el marco de la doceava Conferencia Internacional en Computación y tecnologías de la información del "*Institute of Electrical and Electronics Engineers*" (IEEE).

Inserto en una de las diversas tareas que trae consigo la llegada de la Web Semántica, se encuentra el emparejamiento (Match) entre gafos RDF y el crear nuevas y mejores formas de llevarlo a cabo.

Actualmente la investigación con respecto al emparejamiento de grafos RDF sigue dos líneas principales: se preocupan por la semántica y la similaridad de los elementos de los grafos, o bien se centran en su estructura.

La propuesta concreta de este equipo de académicos de la Universidad de Yunnan, es desarrollar un algoritmo que tome en cuenta tanto la similaridad de los elementos del grafo, como así también, su estructura.

4.1 Similaridad Estructural

Para calcular la similaridad estructural, el artículo se basa en una publicación de Melnik: "Similarity Flooding Algorithm" (Melnik, 2002), que indica que los elementos correspondientes en un grafo son similares cuando sus elementos adyacentes son similares. En sí, el artículo de Melnik es mucho más complejo que esta idea, pero se puede extraer como aplicación de sus conceptos.

4.2 Similaridad Semántica

Para el cálculo de la similaridad semántica, se utiliza como base un tesauro, en este caso se utilizó WordNet (Princeton University, 2017). WordNet es una base de datos léxica, organizada jerárquicamente en grupos de sinónimos conocidos como synsets (Consultar Capítulo 5).

Definición RDF

El artículo define un grafo RDF como una tupla de 6 elementos.

$$G = (V, E, f, g, u, v)$$

- V es un conjunto de nodos de un grafo RDF.
- E es un conjunto de arcos en un grafo RDF.
- $f: V -> L_V$ es un mapeo de V al conjunto de etiquetas de los nodos L_V. Identifica únicamente cada nodo en un grafo RDF.
- $g: E -> L_E$ es un mapeo de E al conjunto de etiquetas de los arcos L_E. Identifica únicamente cada arco en un grafo RDF.
- $u: V -> T_V$ es un mapeo de V al tipo de nodos T_V. El tipo de nodos puede ser un URI, un literal o estar en blanco, vale decir $T\_V\^G = \{uri, literal, blank\}$.

- v: $E -> T_E$ es un mapeo de E al tipo de arcos T_E. El tipo de arco puede ser URI o literal, vale decir $T_E^G = \{uri, literal\}$.

Para mayor claridad se definen dos grafos RDF, en el proceso de emparejamiento.

G^Q = Query graph (Grafo Consulta).

G^T = Target graph (Grafo Objetivo).

t^q = Query Statement (Afirmación Consulta).

t^t = Target Statement (Afirmación Objetivo).

4.3 Cálculo De Similaridad

Para calcular la similaridad entre dos grafos RDF, el artículo propone dividir la tarea en una serie de niveles, donde se evaluarán distintos factores.

Teniendo en cuenta que los grafos RDF pueden ser vistos como un conjunto de afirmaciones RDF (Tripletas), se partirá por establecer la similaridad de las afirmaciones que lo conforman; pero para completar el cálculo de similaridad de las afirmaciones, debemos considerar tanto su parte lingüística como su parte estructural. Para calcular su parte estructural, debemos considerar que las afirmaciones pueden ser descompuestas en sujeto, predicado y objeto; y debemos calcular la similaridad correspondiente a cada uno de estos elementos con su respectiva contraparte del segundo grafo, lo que se define como la similaridad de las etiquetas. La similaridad de las etiquetas puede tener varios casos, dependiendo de sus componentes, pero en términos generales se resume a calcular su similaridad lingüística o "de cadena". La similaridad lingüística se realiza con la ayuda de WordNet y la similaridad de cadena, con el algoritmo de Levenshtein (Lavenshtein, 1966), para el cálculo de la distancia.

Por otra parte, la similaridad estructural se basa en la idea del algoritmo de Melnik (Similarity Flooding) (Melnik, 2002), donde se postula que elementos de dos modelos estructurales son similares cuando los elementos adyacentes son similares.

Esta descomposición se visualiza de mejor forma con el siguiente esquema:

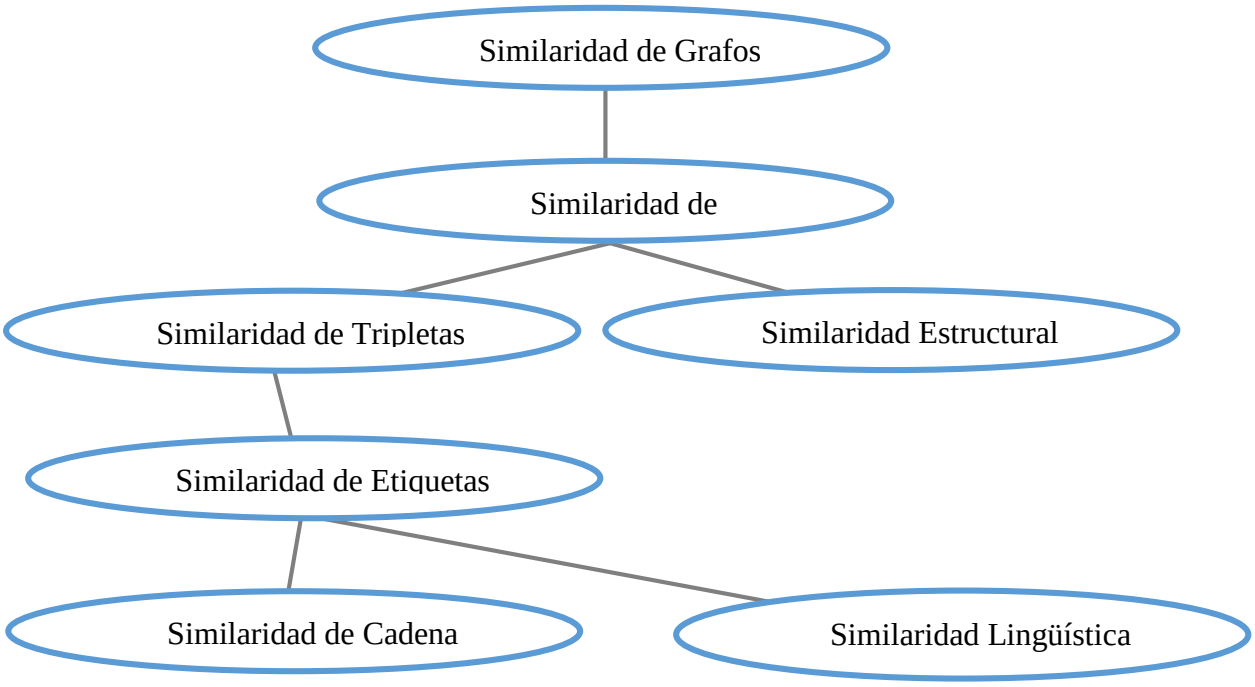


Figura 20 Niveles de cálculo de similaridad.

A continuación, estudiaremos en profundidad los niveles para el cálculo de la similaridad, desde abajo hacia arriba (Figura 20), pues necesitamos los resultados de los niveles inferiores para ir resolviendo las fórmulas de mayor jerarquía.

4.3.1 Similaridad de las etiquetas

Para calcular la similaridad entre dos tripletas de distintos grafos RDF, debemos calcular la similaridad de sus sujetos, predicados y objetos, respectivamente. Para lograr dicha tarea, tenemos

que tener en cuenta que los sujetos y objetos son, en realidad, nodos del grafo y que los predicados son los arcos del mismo grafo; teniendo esto en cuenta podemos ver el cálculo de similaridad como la comparación respectiva de sus etiquetas. La etiqueta de un arco o un nodo puede ser un string, una palabra, un URI o estar en blanco. Es por esta razón es que debemos considerar distintas estrategias para evaluar cada uno de los casos.

Similaridad de cadena entre etiquetas

Para el cálculo de similaridad de cadena entre etiquetas, se usa la fórmula de Levenshtein (Lavenshtein, 1966) para el cálculo de la distancia (Fórmula 1).

La siguiente fórmula ha sido normalizada, para su futuro uso.

$$Sim_{String}(label_1, label_2) = 1 - \frac{LD(label_1, label_2)}{\max(|label_1|,\ |label_2|)}$$

Fórmula 1: Similaridad de cadena.

Esta fórmula se usa para calcular la similaridad de dos etiquetas (de dos grafos RDF distintos, obviamente)

$label_1$= etiqueta del grafo query.

$label_2$ = etiqueta del grafo target.

$LD(label_1, label_2)$ = es la distancia Levenshtein entre ambas etiquetas.

$\max(|label_1|, |label_2|)$ = es el largo máximo de ambas etiquetas.

Similaridad lingüística entre etiquetas

Para el cálculo de la similaridad lingüística se usa WordNet, con el algoritmo de la similaridad de Lin (D, 1998) (Fórmula 2).

$$Sim_{Linguistic}(word_1, word_2) = \frac{2 \times \log P(synset_0)}{\log P(synset_1) + \log P(synset_2)}$$

Fórmula 2: Similaridad Lingüística.

La fórmula de similaridad lingüística se aplica a dos palabras de grafos RDF distintos.

$synset_1$ y $synset_2$ son $synsets$ en WordNet (conjuntos de sinónimos), donde $word_1$ pertenece al $synset_1$ y $word_2$ pertenece al $synset_2$.

Synset$_0$, se refiere al conjunto de sinónimos al que pertenecen tanto el synset$_1$ como el synset$_2$.

$P(synset_0)$, $P(synset_1)$ y $P(synset_2)$ son las probabilidades de que un objeto escogido al azar, pertenezca al $synset_0$, $synset_1$ y $synset_2$, respectivamente.

En el caso en que la palabra no pueda ser encontrada en WordNet, se procede a calcular la similaridad de cadena (Fórmula 1), en su lugar.

La similaridad entre URIs

Uno de los casos más recurrentes es el cálculo de similaridad entre dos URIs. Dado que las URIs tienen una estructura especial, se necesita un método especial para abordar el cálculo de su similaridad.

Una Uri puede descomponerse en cuatro partes:

- ✓ Host
- ✓ Path
- ✓ Authority
- ✓ Fragment

Aquí se extrae el texto de las URI para calcular la similaridad de cada una de sus partes.

Host

Dado que el Host por lo general no contiene semántica, para calcular su similaridad se usa la similaridad de cadena (Fórmula 3).

$$Sim_{Host}(URI_1, URI_2) = Sim_{String}(URI_1^{host}, URI_2^{host})$$

Fórmula 3: Similaridad del Host de las URIs de dos tripletas de dos grafos RDF.

URI_1 = URI de una tripleta del Grafo Query.

URI_2 = URI de una tripleta del Grafo Target.

Sim_{String} corresponde a la fórmula 1.

Authority

Esta parte tiene las mismas características que el Host, así que no se calcula la similaridad entre ellas.

Path

La similaridad de la Ruta o Path se define de la siguiente forma:

$$Sim_{Path}(URI_1, URI_2) = Sim_{Linguistic}(URI_1^{path}, URI_2^{path})$$

Fórmula 4: Similaridad de las Ruta de dos tripletas de dos grafos RDF.

URI_1^{path} = es la ruta de la URI de una tripleta del Grafo Query.

URI_2^{path} = es la ruta de la URI de una tripleta del Grafo Target.

$Sim_{Linguistic}$= es la similaridad Lingüística (fórmula 2).

Fragment

La similaridad de los fragmentos, si es que tienen, se define de la siguiente forma:

$$Sim_{Fragment}(URI_1, URI_2) = Sim_{Linguistic}(URI_1^{fragment}, URI_2^{fragment})$$

Fórmula 5: Similaridad de los fragmentos de dos tripletas de dos grafos RDF.

$URI_1^{fragment}$= es el fragmento de la URI de una tripleta del Grafo Query.

$URI_2^{fragment}$= es el fragmento de la URI de una tripleta del Grafo Target.

Cuando han sido calculadas las similaridades de cada parte de las URIs, se procede con el cálculo de la similaridad de las URIs, definida por la siguiente fórmula:

$$Sim_{URI}(URI_1, URI_2) = \frac{Sim_{Host} + Sim_{Path} + Sim_{Fragment}}{3}$$

Fórmula 6: Similaridad de dos URIs de dos tripletas de dos Grafos RDF.

Si tan sólo una tripleta contiene un URI y la otra, una palabra, entonces su similaridad está dada por el cálculo entre el Fragmento de la URI y la palabra contenida en la segunda URI.

$$Sim_{URI_{Word}}(word, URI) = Sim_{Linguistic}(word, URI^{fragment})$$

Fórmula 7: Similaridad entre una URI y una palabra.

4.3.2 Similaridad entre afirmaciones

En el artículo se calcula la similaridad entre afirmaciones RDF como la suma ponderada entre su similaridad lingüística y su similaridad estructural (Fórmula 8).

$$Sim_{Statement}(t^q, t^t) = \theta \times Sim_{Triple}(t^q, t^t) + \gamma \times Sim_{Structural}(t^q, t^t)$$

$$(\theta + \gamma = 1)$$

Fórmula 8: Similaridad entre afirmaciones de dos grafos RDF.

t^q = es la tripleta o afirmación RDF del grafo Query.

t^t= es la tripleta o afirmación RDF del grafo Target.

(θ y γ son referencias de pesos.)

Sim_{Triple} = es la similaridad de las tripletas t^t y t^q (Fórmula 9).

$Sim_{Structural}$ = es la similaridad estructural de las tripletas t^t y t^q (Fórmula 11).

4.3.2.1 Similaridad de las tripletas

Una afirmación en RDF es una tripleta que contiene un sujeto, un predicado y un objeto, por ejemplo:

$$t^q = (s^q, p^q, o^q)$$

El sujeto y el objeto son nodos y el predicado es un arco entre el sujeto y el objeto.

De acuerdo a los diferentes tipos de nodos y arcos, se puede calcular la similaridad de las tripletas (ver Fórmula 9).

$$Sim_{Triple}(t^q, t^t) = \lambda_1 \times Sim_{Label}(s^q, s^t) + \lambda_2 \times Sim_{Label}(p^q, p^t) + \lambda_3 \times Sim_{Label}(o^q, o^t)$$

$$(\lambda_1 + \lambda_2 + \lambda_3 = 1)$$

Fórmula 9: Similaridad de dos tripletas RDF.

$s^q, p^q \; y \; o^q$ se refieren al sujeto, predicado y objeto de una tripleta del grafo query; mientras que $s^t, p^t \; y \; o^t$ se refiere al sujeto, predicado y objeto de una tripleta del grafo target.

Sim_{Label} corresponde a la similaridad de dos etiquetas, en este caso se evalúan respectivamente, ambos sujetos, ambos predicado y ambos objetos.

(λ_1, λ_2 y λ_3 son referencias de pesos).

4.3.2.2 Similaridad estructural

Al comparar dos afirmaciones RDF, el artículo considera la relación entre las tripletas, pues se considera que si dos afirmaciones son similares, en distintos grafos, las afirmaciones adyacentes a éstas también son similares.

La forma de calcular la similaridad estructural se visualiza en el siguiente esquema

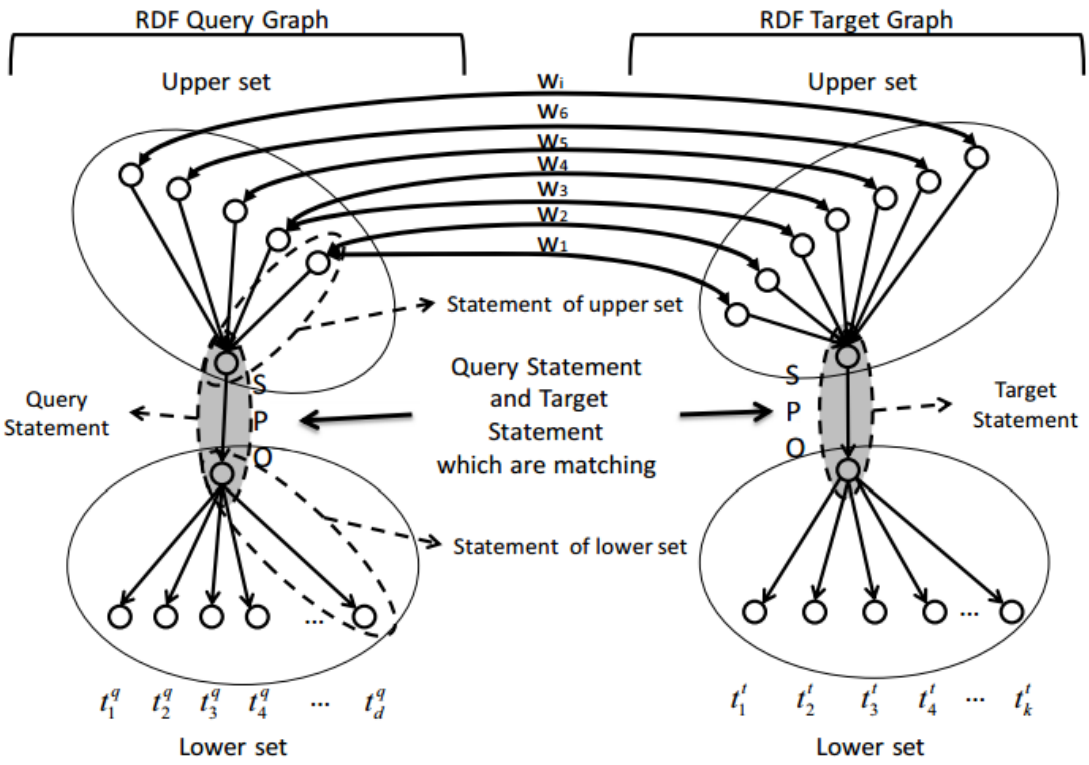


Figura 21 Similaridad estructural de dos tripletas RDF.

En la figura anterior, se visualiza el proceso de comparar estructuralmente dos tripletas RDF: la de la izquierda (Query Statement) corresponde a una afirmación de un grafo Query, mientras que la de la derecha (Target Statement), a una afirmación de un grafo Target.

La parte superior de ambas tripletas, se conoce como “The Upper Set” o el conjunto superior. El conjunto superior está compuesto por todas las tripletas cuyo objeto sea el sujeto de la tripleta que se está comparando.

Así mismo, la parte inferior de ambas tripletas, se conoce como “The Lower Set” o el conjunto inferior. El conjunto inferior está compuesto por todas las tripletas cuyo sujeto sea el objeto de la tripleta que se está comparando.

Tanto el conjunto superior, como el inferior se toman en cuenta al momento de calcular la similaridad estructural.

The upper set

El cálculo de similaridad del conjunto superior consiste en comparar todo el conjunto superior de una tripleta query con todo el conjunto superior de una tripleta target. Luego se almacenan los valores máximos para cada integrante del conjunto superior de la tripleta query y se promedian (ver Fórmula 10).

$$Sim_{upper}(t^q, t^t) = \frac{\sum_{j=1}^{k} \max(Sim_{1j}, Sim_{2j}, Sim_{3j}, \dots, Sim_{dj})}{k}$$

Fórmula 10: Similaridad del conjunto superior

The lower set

El cálculo de similaridad para el conjunto inferior tiene la misma forma que la fórmula 10, pero se denota como Sim_{Lower} en vez de Sim_{upper}.

Finalmente, la Similaridad Estructural queda dada en función los conjuntos superior e inferior (ver Fórmula 11).

$$Sim_{Structural}(t^q, t^t) = \alpha \times Sim_{upper}(t^q, t^t) + \beta \times Sim_{lower}(t^q, t^t)$$

$$(\alpha + \beta = 1)$$

Fórmula 11. Similaridad estructural de dos tripletas de dos grafos RDF.

(α y β son pesos referenciales).

4.3.3 Similaridad entre grafos RDF

Basado en la similaridad de las afirmaciones, se puede medir la similaridad de los grafos RDF usando el algoritmo de la maximalidad ponderada para el emparejamiento de grafos bipartitos.

Las afirmaciones $\{t_1^q, t_2^q, \dots, t_m^q\}$ en el grafo query pueden verse como un conjunto de nodos de un grafo bipartito, y las afirmaciones $\{t_1^t, t_2^t, \dots, t_n^t\}$ en el grafo target, como el otro conjunto de nodos del grafo bipartito, el arco $e(t_i^q, t_j^q)$ conecta los nodos t_i^q y t_j^q, y el peso w_{ij} en el arco $e(t_i^q, t_j^q)$ es la similaridad entre las afirmaciones t_i^q y t_j^q; lo que se puede visualizar finalmente como un gran grafo bipartito.

Finalmente, para evitar que el mismo subgrafo no pueda ser distinguido de diferentes escalas de grafos, como muestra la siguiente figura

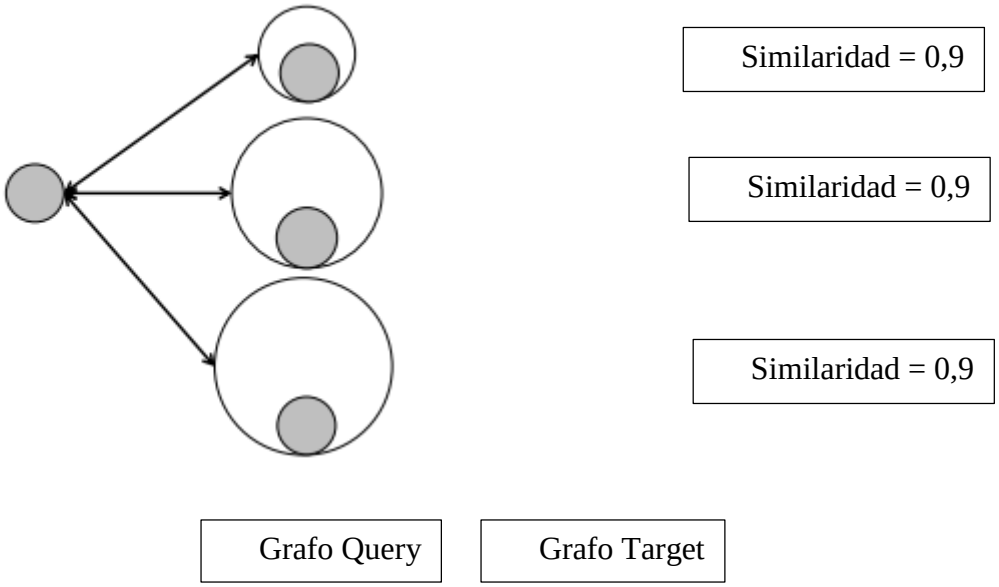


Figura 22 Emparejamiento según grafo Query.

Se necesita contar el grado de cada afirmación, pues mientras mayor sea, más importancia tiene dentro del grafo. El grado de una tripleta (Deg) está dado por la suma de su Indeg y su Outdeg.

$Indeg$ = Número de arcos que apuntan al sujeto de la afirmación.

$Outdeg$ = Número de arcos que salen desde el objeto de la afirmación.

$$Deg(t^t) = Indeg + Outdeg$$

Fórmula 12: Grado de una tripleta RDF.

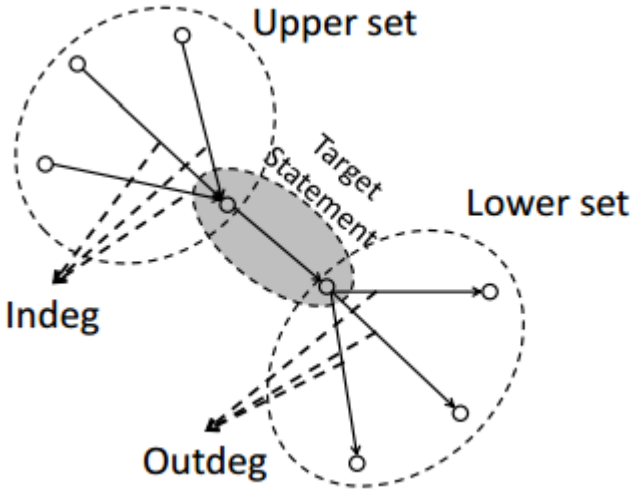


Figura 23 Indeg y Outdeg de una tripleta RDF.

Finalmente, la fórmula para el cálculo de la similaridad de dos grafos RDF se observa en la Fórmula 13.

$$Sim(G^Q, G^T) = \frac{\sum_{i=1}^{\min(m,n)} [Sim_{Structural}(t_i^q, t_i^t) \times Deg(t_i^t)]}{\sum_{i=1}^{\min(m,n)} Deg(t_i^t)}$$ [3]

Fórmula 13: Similaridad de dos Grafos RDF.

[3] En el artículo original dice Sim_{Triple}, pero debería decir $Sim_{Structural}$, pues es esa función la que devuelve la similaridad de las afirmaciones y no Sim_{Triple}, que, como se aprecia en la Figura 20, es de menos nivel jerárquico.

4.4 Complejidad Del Algoritmo

La complejidad del algoritmo está dado por el número de comparaciones entre los nodos y los arcos en los grafos RDF, lo que está dentro del orden de n^2. En la Figura 24 podemos observar los tiempos del algoritmo bajo distintos pesos de bases de datos RDF.

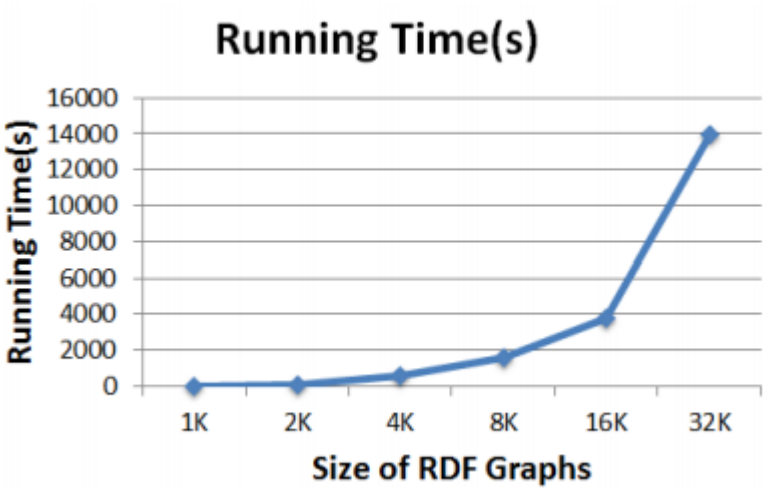


Figura 24 Tiempo del algoritmo para distintos pesos de bases de datos RDF.

Se encontraron fascinantes las ideas publicadas en el artículo de la universidad de Yunnan y dignas de estudio. Sobre todo, porque apuntan en la misma dirección que nuestra investigación, pues, como vimos en este capítulo, el resultado del algoritmo, es de hecho un subgrafo del grafo target, en función del grafo query.

Consideramos que el hecho de incorporar WordNet para el cálculo semántico es una línea de investigación importante, principalmente, por el valor que se le puede dar. Sin embargo, hay información que no queda muy clara, como la forma de obtener la similaridad de WordNet y el costo real del algoritmo.

En términos generales, el algoritmo propuesto es muy potente, pero desgraciadamente es demasiado costoso. Creemos que el número de comparaciones que realiza es mucho mayor que n2, como ellos indican, esto porque además de la comparación directa de las tripletas, subyace la comparación secuencial de la similaridad estructural, que necesita comparar el conjunto superior e inferior para cada tripleta.

Por lo tanto, decidimos incluir en nuestra investigación de Tesis el estudio, análisis e implementación del algoritmo de emparejamiento de grafos RDF orientado a la similaridad, propuesto por la Universidad de Yunnan, con el objetivo de contrastar la información presentada y además, diseñar mejoras para disminuir el costo del algoritmo.

Por otra parte, resulta necesario añadir a nuestra investigación de tesis el estudio, análisis e implementación de WordNet para entender su funcionamiento y aplicarlo en la búsqueda y emparejamiento de grafos RDF.

En el siguiente capítulo nos centraremos en comprender el funcionamiento de WordNet.

Capítulo 5: WordNet

Este capítulo se centra en WordNet, su codificación y funcionalidad, con el objetivo de comprender cómo se lleva acabo el cálculo de similaridad semántica con el uso de esta base de datos lingüística, en el artículo de la Universidad de Yunnan.

WordNet se define como una base de datos léxica, por lo que se esperaría que fueran archivos de datos, pero al consultar la página oficial (Princeton University, 2017) se obtiene un archivo ejecutable para instalar; Una vez instalado WordNet, la aplicación que se ejecuta es básicamente un buscador de palabras en inglés.

Con un poco más de investigación y luego de revisar los archivos instalados, se comprendió que la versión de WordNet que se ejecuta es una aplicación y que WordNet básicamente como tal (una base de datos léxica) se reduce a unos pocos archivos de texto y el resto son funciones para buscar y mostrar información en el software.

Sin embargo, el potencial de WordNet reside en la codificación de esos archivos de texto y es lo que se verá a continuación.

Básicamente, los archivos de WordNet están organizados de forma jerárquica en grupos de sinónimos llamados synsets.

5.1 Archivos Base

La base de datos de WordNet, propiamente como tal está formada por 8 archivos de texto. Dos archivos por cada categoría sintáctica cubierta por WordNet. WordNet maneja Sustantivos, Verbos, Adjetivos y Adverbios, por lo que se tienen dos archivos para cada una de estas categorías: Un index y un archivo Data. Básicamente, se tiene un índice y un lugar donde se almacena la información.

- Index.noun
- Data.noun
- Index.verb
- Data.verb
- Index.adj
- Data.adj
- Index.adv
- Data.adv

5.2 Codificación Del Index

Las primeras líneas de los archivos Index contienen un texto de copyright, y luego una lista de todas las palabras, alfabéticamente ordenadas, contenidas en el archivo Data.

En cada línea del archivo Index encontramos una codificación como la siguiente:

$$lemma\ pos\ synse\_cnt\ p\_cnt\ [ptr\_symbol\ \ldots]sense\_cnt\ tagsense\_cnt\ synset\_offset\ [synset\_offset\ \ldots]$$

Veamos un ejemplo con el index de los sustantivos (index.noun) para la palabra “cat”.

```
cat n 8 5 @ ~ #m + ; 8 1 02100898 10002338 09754264
```

Lemma: Nombre de la palabra. Ejemplo: (cat).

Pos: Categoría sintáctica (n para sustantivos, v para verbos, a para adjetivos y r para adverbios). Ejemplo: (n).

Synset_cnt: Número de synsets en los que aparece la palabra. Ejemplo: (8).

P_cnt: Número de diferentes punteros que la palabra tiene en todos los synsets que la contienen. Ejemplo: (5).

Ptr_symbol: una lista separada por un espacio de los diferentes tipos de punteros que la palabra tiene en todos los synsets que la contienen.

Las tablas 1, 2, 3 y 4 resumen los punteros para cada tipo sintáctico de WordNet.

Tabla 1
Tabla de punteros para sustantivos

Símbolo	Significado
!	Antonym
@	Hypernym
~	Hyponym
~i	Instance Hyponym
#m	Member holonym
#s	Substance holonym
#p	Part holonym
%m	Member meronym
%s	Substance meronym
%p	Part meronym
=	Attribute
+	Derivationally related form
;c	Domain of synset - TOPIC
-c	Member of this domain - TOPIC
;r	Domain of synset - REGION
-r	Member of this domain - REGION
;u	Domain of synset - USAGE

-u	Member of this domain - USAGE

Datos obtenidos de la especificación de WordNet.

Tabla 2
Tabla de punteros para verbos

Símbolo	Significado
!	Antonym
@	Hypernym
~	Hyponym
*	Entailment
>	Cause
^	Also see
$	Verb Group
+	Derivationally related form
;c	Domain of synset - TOPIC
;r	Domain of synset - REGION
;u	Domain of synset - USAGE

Datos obtenidos de la especificación de WordNet.

Tabla 3
Tabla de punteros para adjetivos

Símbolo	Significado
!	Antonym
&	Similar to
<	Participle of verb
\	Pertainym (pertenece a un sustantivo)
=	Attribute
^	Also see
;c	Domain of synset - TOPIC
;r	Domain of synset - REGION
;u	Domain of synset - USAGE

Datos obtenidos de la especificación de WordNet.

Tabla 4
Tabla de punteros para adverbios

Símbolo	Significado
!	Antonym
\	Derived from adjective
;c	Domain of synset - TOPIC
;r	Domain of synset - REGION
;u	Domain of synset - USAGE

Datos obtenidos de la especificación de WordNet.

Para el ejemplo: cat, sus punteros nos indican:

```
@ ~ #m + ;
```

Hypernym, Hyponym, Member holonym, Derivationally related from.

Luego de los punteros, tenemos:

Sense_cnt: Lo mismo que synsets_cnt. Ejemplo (8).

Synset_offset: La dirección en número de 8 dígitos de la posición en la que se encuentra la palabra en el archivo data correspondiente. Cada uno de estos offsets representa un sentido diferente de la palabra.

Ejemplo: `02100898 10002338 09754264`

5.3 Codificación Del Data

Al igual que los index, los archivos data comienzan con unas líneas de copyright, seguidos por las líneas con la información, ordenadas por el offset que indica su posición en el archivo.

En cada línea del data encontramos una codificación como esta:

$$synset\_offset\ lex\_filenum\ ss\_type\ w\_cnt\ word\ lex\_id$$

$$[word\ lex\_id\ \ldots]\ \ p\_cnt\ \ [ptr\ \ldots]\ \ [frames\ \ldots]\ \ |\ gloss$$

Siguiendo el primer offset, del ejemplo anterior, encontramos:

02100898 05 n 02 cat 0 true_cat 0 003 @ 02100294 n 0000 ~ 02101086 n 0000 ~ 02103677 n 0000 | feline mammal usually having thick soft fur and no ability to roar: domestic cats; wildcats

Synset_offset: Posición actual en el archivo representada como un decimal entero de 8 dígitos. Ejemplo: (02100898)

Lex_filenum: Decimal entero de 2 dígitos que indica el nombre lexicográfico que contiene el synset. Ejemplo (05)

Ss_type: Un carácter que indica el tipo de synset.

n	**Noun (Sustantivo)**
v	Verb (Verbo)
a	Adjective (Adjetivo)
s	Adjective Satellite (Adjetivo Satélite)
r	Adverb (Adverbio)

Ejemplo: (n) Noun.

W_cont: Entero hexadecimal de dos dígitos que indica el número de palabras en el synset.

Ejemplo: (02).

Word: Forma ASCII de las palabras contenidas en el synset.

Ejemplo: cat

Lex_id: Entero hexadecimal de un digito, que junto con el lemma, identifica de forma única a un sentido dentro de un archivo lexicográfico.

P_cnt: Entero decimal de 3 digitos que indica el número de punteros del synset actual a otros synsets. Si aparece “000” significa que no tiene punteros.

Ptr: Puntero de este synset a otro. Los punteros tienen la siguiente forma:

$$pointer\_symbol \;\; synset\_offset \;\; pos \; source/target$$

Pointer_symbo: es igual a los punteors descritos en el index.

Synset_offset: es la posición en la que se encuentra el synset apuntado

Pos: es el tipo de archivo al que apunta el synset actual

Source/target: es un campo de cuatro bytes, que contiene dos enteros hexadecimales de dos dígitos. Los primeros dos dígitos señalan la posición de la palabra en el synset actual (source), mientras que los dos últimos dígitos señalan la posición de la palabra apuntada por el synset actual, en el synset apuntado.
Si toma el valor "0000" quiere decir que se trata de una relación semántica entre el synset actual y el apuntado.

WordNet es una herramienta con mucho potencial y debido a su estructura, jerárquicamente ordenada, podría prestar mucha utilidad en el desarrollo de programas para la web semántica.

Uno de los problemas con WordNet y que puede ser la causa de la falta de información en la Web, se debe principalmente a que para trabajar con él, hay que ensuciarse las manos y meterse en sus archivos, crear código desde cero, a partir de sus archivos base, lo que en nuestro caso, no es ningún impedimento, pues es precisamente una herramienta de este tipo la que buscamos.

Ahora que entendemos el funcionamiento de WordNet, es tiempo de retomar el algoritmo orientado a la similaridad para seguir analizando sus características y comenzar a diseñar las estructuras y algoritmos y funciones específicas que nos permitan recrearlo.

Capítulo 6: Similarity-Oriented Algorithm: Analisis, Diseño, Implementación y Pruebas.

En el presente capítulo se resumen los descubrimientos del estudio sobre el artículo presentado en el capítulo cuatro, que propone el algoritmo orientado a la similaridad semántica y estructural. Se exponen los problemas encontrados y las soluciones que se diseñaron en respuesta. Se presenta además, las pruebas preliminares de la implementación de las soluciones, lo que se traduce en la replicación del algoritmo propuesto. Posteriormente se introduce el algoritmo LPS (Longest Path Subgraph) como una mejora para disminuir el costo del algoritmo orientado a la similaridad y, finalmente, se entregan las pruebas y resultados obtenidos.

6.1 Similaridad Lingüística

El primer problema que se nos presentó fue el cálculo de similaridad lingüística.

En el artículo se propone el uso de una fórmula para calcular la similaridad de dos términos en una taxonomía, basado en una publicación de Lin D. sobre el cálculo de similaridad en diversas áreas (D, 1998). En dicho artículo se expone el cálculo de la similaridad entre dos términos en una taxonomía, como es el caso de WordNet (ver Fórmula 14).

$$Sim(x_1, x_2) = \frac{2x \log P(C_0)}{\log P(C_1) + \log P(C_2)}$$

Fórmula 14: Similaridad entre dos términos de una taxonomía.

x_1 y x_2 son los términos a calcular la similaridad.

x_1 pertenece a una categoría C_1, dentro de la taxonomía.

x_2 pertenece a una categoría C_2, dentro de la taxonomía.

C_0 es la categoría general en la que se puede encontrar C_1 y C_2.

$P(C_0), P(C_1)$ y $P(C_2)$ son probabilidades de que un término aleatorio pertenezca a C_0, C_1 y a C_2, respectivamente.

Incluso se incluye un ejemplo con las palabras “hill” y “coast”.

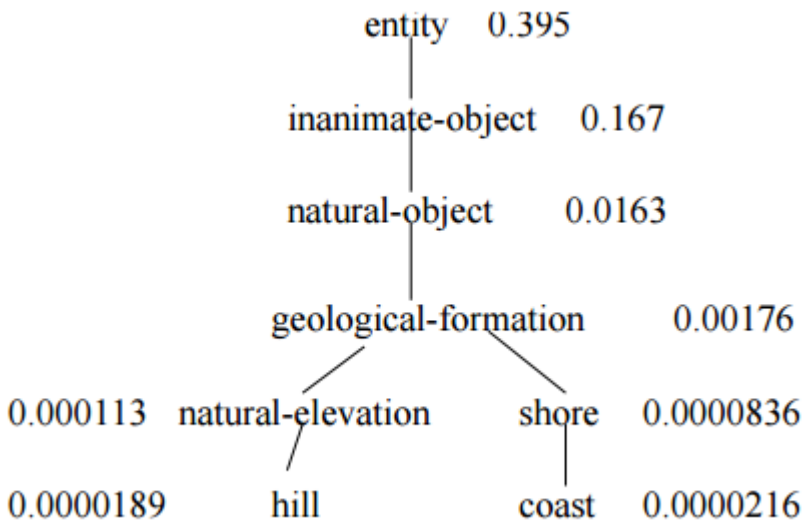


Figura 25 Fragmento de WordNet

En la figura 25, se aprecia la estructura jerárquica de dos palabras de WordNet, y al lado de cada palabra se tiene su probabilidad $P(C_X)$.

Luego, aplicando la fórmula, se tiene:

$$Sim(Hill, Coast) = \frac{2 \times \log P(Geological - Formation)}{\log P(Hill) + \log P(Coast)}$$

El problema es que ninguna de las dos publicaciones menciona cómo calcular las probabilidades, sólo indican el concepto básico de probabilidad:

P (C) = probabilities that a randomly selected object belongs to C.

Hill y Coast son sustantivos, por lo tanto sus synsets están contenidos en los archivos Index.noun y data.noun

La probabilidad de que una palabra al azar pertenezca al synset al que pertenece Hill, vendría dado por el número de palabras que contiene el synset en el que se encuentra Hill, dividido por el

número de palabras sustantivas (si se quiere ocupar sólo palabras sustantivas) o del número total de palabras de WordNet.

Sin embargo, con ninguno de los métodos mencionados se obtuvieron los valores presentados en el artículo.

Se consideró, además, todos los offsets para las palabras, con el número de palabras contenidas en cada offset y tampoco se obtuvo los resultados esperados.

Se probó el caso de versiones distintas de WordNet y que eso estuviese afectando el resultado. Se comprobó con las 3 versiones de WordNet: 2.0, 2.1 y 3.0, y con ninguna se obtuvo resultados parecidos sobre la probabilidad de cada palabra.

Sin embargo, la misma publicación de Lin D. incluía otra fórmula para calcular la similaridad entre términos de una taxonomía: la fórmula de Wu y Palmer.

Ambas fórmulas mostraban igual rendimiento y valores de similaridad cercanos, con variaciones despreciables (D, 1998).

6.2 Similaridad De Wu y Palmer.

La similaridad de Wu y Palmer propone una medida para la similaridad semántica que puede verse como un caso especial de la similaridad de Lin D.

$$Sim_{Wu\&Palmer}(A, B) = \frac{2 \times N_3}{N_1 + N_2 + 2 \times N_3}$$

Fórmula 15: Similaridad de Wu y Palmer.

N_1 y N_2 son los números de vínculos “is-a” de A y B a su superclase común más específica C.

N_3 es el número de vínculos “is-a” desde C a la raíz de la taxonomía.

Tomando el mismo ejemplo de Hill y Coast.

La superclase común más específica de Hill y Coast es “Geological-Formation”.

Entonces,

$N_1 = 2$ $N_2 = 2$ $N_3 = 3$

Así, la similaridad de Wu y Palmer entre las palabras Hill y Coast es de 0,6.

Los datos necesarios para calcular la similaridad de Wu y Palmer, pueden ser extraídos desde el archivo Data de cada palabra de WordNet.

Ante estos hechos y para efectos de estudio, se decidió trabajar con la fórmula de Wu y Palmer, en lugar de la fórmula de Similaridad de Lin D.

6.3 Análisis De Similaridad De Los Tipos Sintácticos De WordNet

Se comenzó con la idea de implementar la función de similaridad para los cuatro tipos sintácticos de WordNet, vale decir: Suntantivos, verbos, adjetivos y adverbios. Pero esta funcionalidad tiene un costo muy alto, determinado por el número de operaciones realizadas para establecer el tipo sintáctico de las palabras y posteriormente, una forma de comparación en función de términos comunes.

Hay que tener en cuenta que, en el peor de los casos, para identificar una palabra determinada, como uno de estos tipos sintácticos es necesario filtrarlos por los cuatro index. Este filtro se realiza a través de una búsqueda binaria sucesiva hasta encontrar la palabra en alguno de los archivos index. El mejor de los casos corresponde a encontrar la palabra en el primer index, ahorrando la búsqueda en los 3 archivos restantes.

Una idea para optimizar el proceso de identificación del tipo sintáctico de las palabras sería la ponderación de porcentajes de ocurrencia para determinar cuál es el tipo sintáctico más frecuente; de esta forma, pondríamos el archivo correspondiente a ese tipo sintáctico como prioridad a la hora

de filtrar la búsqueda, sin embargo, de todas formas habría casos en los que se debería filtrar por los cuatro tipos sintácticos hasta encontrarlos.

Una vez se ha obtenido el tipo sintáctico de las palabras, se procede a realizar la comparación entre ellas. En esta etapa, se debe tener en cuenta la relación que estos cuatro tipos sintácticos establecen entre ellos, en WordNet.

Volviendo a los punteros de los archivos index y data, nos damos cuenta de lo siguiente:

Primero que todo, la función de similaridad de Wu y Palmer, trabaja en función de los synsets "padres" de las palabras a buscar.

Por lo tanto es necesario que ambas palabras tengan el puntero "@" Hypernym – Apunta al synset padre.

Los sustantivos y los verbos tienen esta propiedad.

Lamentablemente ni los adjetivos ni los adverbios cuentan con esta propiedad, así que no se podría aplicar la fórmula de Wu y Palmer de forma directa.

Sin embargo, lo que sí se podría hacer es trabajar con otras propiedades.

Para el caso de los adverbios,

Tenemos el puntero "\" Derived from adjective y en el caso de los adjetivos, el puntero "\" Pertainym (pertains to noun) y "<" Participle of verb.

Entonces, una forma de calcular la similaridad sería:

Transformando las palabras, identificadas como adverbios, a su versión adjetiva, y una vez siendo adjetivos, transformarlos a su vez, en su versión sustantiva o forma verbal, ya que en estos dos últimos tipos sintácticos sí es posible aplicar la fórmula.

Hay que hacer notar, eso sí, que no todos los adverbios tienen versiones adjetivas ni todos los adjetivos, versiones sustantivas o verbales.

Por último, hay que tener en cuenta que todas estas comparaciones se deberían realizar para cada comparación entre dos palabras, sólo para obtener la similaridad lingüística.

Por lo tanto, por los motivos expuestos, se decidió sólo trabajar con sustantivos.

Además, en ninguna de las publicaciones se ha incluído ejemplo alguno que mencione comparaciones de palabras que no sean sustantivos.

6.4 Número De Comparaciones

Otro aspecto a considerar del algoritmo es el Nº excesivo de comparaciones, ya sea con la fórmula original de Lin D. o con la de Wu y Palmer.

A continuación veremos las comparaciones necesarias para realizar el cálculo de similaridad de dos palabras cualesquiera.

Establézcase la similaridad de “cat” y “dog”.

Para cada palabra, es necesario buscar en el index.noun (asumiendo que sólo trabajamos con sustantivos) hasta encontrar la palabra “cat” y/o “dog”.

Luego en la línea obtendremos mucha información, como ya vimos en el capítulo 9: WordNet.

Lo que nos interesa almacenar son todos los offsets, que vendrían siendo los sentidos de las palabras.

Luego debemos realizar una comparación de m x n, siendo m y n el número de offsets de cat y dog, respectivamente.

En cada comparación m x n, aplicamos la fórmula de similaridad escogida y nos quedamos con el resultado mayor.

Con la fórmula de Lin D. se requiere entrar a los archivos de WordNet para tomar los datos del synset al que corresponde la palabra a buscar, para obtener su probabilidad.

Con la fórmula de Wu y Palmer también, por supuesto, se requiere entrar a los archivos de WordNet para obtener las distancias respectivas de cada palabra a comparar.

Vale decir, que estas comparaciones son sólo para el cálculo de similaridad lingüística de dos palabras.

Se debe tener en cuenta que para comparar dos tripletas, se debe realizar al menos 3 comparaciones, que en su peor caso serían las 3 lingüísticas (sujeto con sujeto, predicado con predicado y objeto con objeto).

Esto hasta ahora, sólo para calcular la similaridad de las tripletas, porque para realizar la similaridad estructural son aún más comparaciones.

Para obtener la similaridad estructural se debe además, calcular la similaridad del conjunto superior e inferior entre ambas tripletas, y este cálculo hacerlo por cada tripleta del grafo query contra cada tripleta del grafo target.

Si bien, en teoría, el algoritmo es muy potente y entrega un resultado óptimo, es necesaria su implementación para probar realmente cuanto se demora y si realmente vale la pena el alto costo a pagar.

6.5 Estructura Base

Para el diseño e implementación del algoritmo propuesto por los docentes de la Universidad de Yunnan, se utilizó una estructura base que maneja listas dinámicas de adyacencia.

Para satisfacer los requerimientos de la propuesta, en cuanto a la similaridad estructural, se incluyen dos listas de adyacencia: una lista de llegada y una lista de salida, para cada nodo. Al mantener actualizadas estas listas, se asegura la rapidez en la obtención de datos al momento de

calcular la similaridad estructural entre tripletas, pues corresponde a la implementación directa del *Upper* y el *Lower set*.

6.6 Carga De Archivos

Lo primero que realiza el algoritmo es una carga de archivos RDF. Estos archivos están codificados en turtle. El algoritmo rescata las tripletas y las almacena como nodos y arcos, según sea el caso, en la estructura base.

Una vez han sido cargados los archivos y almacenados en la estructura base, se tienen dos grafos: un grafo Query y un grafo Target.

A estos dos grafos se les aplica la fórmula de similaridad entre dos grafos (Fórmula 14).

6.7 Algoritmo Orientado a La Similaridad: Pruebas Preliminares.

La primera prueba preliminar consiste en probar el algoritmo orientado a la similaridad con dos grafos iguales y aumentar progresivamente sus arcos, manteniendo la altura del grafo, igual a 1, como se muestra en la figura 26.

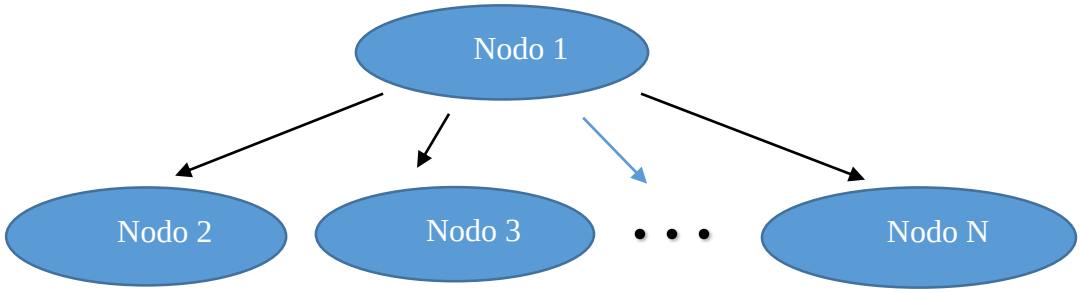


Figura 26 Estructura Pruebas preliminares.

Se medirá exclusivamente el tiempo de la función de similaridad de los Grafos y no la carga de los archivos.

Prueba 1 - 5 tripletas.

A continuación, se presentan los archivos RDF a comparar, dichos archivos están codificados en turtle y, para objeto de estas pruebas, tanto los archivos Query como los Target serán los mismos, el resultado esperado es que el algoritmo entregue una similaridad máxima de 1.

https://en.wikipedia.org/wiki/Leonardo_da_Vinci *https://www.wikidata.org/wiki/Property:P001* *https://en.wikipedia.org/wiki/painter*

https://en.wikipedia.org/wiki/Leonardo_da_Vinci *https://www.wikidata.org/wiki/Property:P002* *https://en.wikipedia.org/wiki/engineer*

https://en.wikipedia.org/wiki/Leonardo_da_Vinci *https://www.wikidata.org/wiki/Property:P003* *https://en.wikipedia.org/wiki/astronomer*

https://en.wikipedia.org/wiki/Leonardo_da_Vinci *https://www.wikidata.org/wiki/Property:P004* *https://en.wikipedia.org/wiki/philosopher*

https://en.wikipedia.org/wiki/Leonardo_da_Vinci *https://www.wikidata.org/wiki/Property:P005* *https://en.wikipedia.org/wiki/anatomist*

```
TIME: 2.289000

RDF SIMILARITY: 1.000000

Process returned 0 (0x0)   execution time : 2.469 s
Press any key to continue.
```

El algoritmo entrega efectivamente la similaridad máxima de 1, con un tiempo promedio de la función de similaridad de 2,278 s.

El tiempo de 2,469 s. contempla el tiempo de la carga de los archivos más el de la fórmula de similaridad de los grafos RDF.

Prueba 2 - 10 tripletas

A continuación se presentan los archivos query y target, que al igual que la prueba anterior, serán los dos iguales, pero está vez tendrán diez tripletas en lugar de sólo cinco. Se espera nuevamente que el algoritmo entregue la similaridad máxima de 1.

https://en.wikipedia.org/wiki/Leonardo_da_Vinci https://www.wikidata.org/wiki/Property:P001 https://en.wikipedia.org/wiki/painter

https://en.wikipedia.org/wiki/Leonardo_da_Vinci https://www.wikidata.org/wiki/Property:P002 https://en.wikipedia.org/wiki/engineer

https://en.wikipedia.org/wiki/Leonardo_da_Vinci https://www.wikidata.org/wiki/Property:P003 https://en.wikipedia.org/wiki/astronomer

https://en.wikipedia.org/wiki/Leonardo_da_Vinci https://www.wikidata.org/wiki/Property:P004 https://en.wikipedia.org/wiki/philosopher

https://en.wikipedia.org/wiki/Leonardo_da_Vinci https://www.wikidata.org/wiki/Property:P005 https://en.wikipedia.org/wiki/anatomist

https://en.wikipedia.org/wiki/Leonardo_da_Vinci https://www.wikidata.org/wiki/Property:P006 https://en.wikipedia.org/wiki/mathematician

https://en.wikipedia.org/wiki/Leonardo_da_Vinci https://www.wikidata.org/wiki/Property:P007 https://en.wikipedia.org/wiki/sculptor

https://en.wikipedia.org/wiki/Leonardo_da_Vinci https://www.wikidata.org/wiki/Property:P008 https://en.wikipedia.org/wiki/polymath

https://en.wikipedia.org/wiki/Leonardo_da_Vinci https://www.wikidata.org/wiki/Property:P009 https://en.wikipedia.org/wiki/architect

https://en.wikipedia.org/wiki/Leonardo_da_Vinci https://www.wikidata.org/wiki/Property:P010 https://en.wikipedia.org/wiki/civil_engineer

```
TIME: 3.862000

RDF SIMILARITY: 1.000000

Process returned 0 (0x0)   execution time : 4.162 s
Press any key to continue.
```

El algoritmo entrega la similaridad máxima, con un tiempo promedio de la función de similaridad de 4,0595 s. El tiempo total de 4,162 s. contempla el tiempo de la carga de los archivos más el de la fórmula de similaridad de los grafos RDF.

6.8 LPS: Longest Path Subgraph.

Si bien las pruebas preliminares entregaron los resultados esperados, es necesario trabajar en mejoras, que se traduzcan en un menor tiempo de ejecución.

Como mejora propuesta para el algoritmo orientado a la similaridad, se propone el uso del algoritmo LPS: Longest Path Subgraph (Subgrafo del camino más largo).

El algoritmo LPS fue publicado por Claudio Gutierrez, Pedro G. Campos y Julio Águila, en el marco de la Conferencia Internacional en Ciencias de la Computación en México, 2008.

El algoritmo LPS propone un nuevo algoritmo para detectar el isomorfismo de grafos RDF.

En términos simples, el algoritmo LPS hace un preprocesamiento de los grafos RDF a comparar. Durante dicho preprocesamiento el algoritmo genera dos subgrafos, uno de cada grafo (Grafo Query y Grafo Target) y luego procede a comparar los subgrafos para ver si son isomorfos.

Los subgrafos creados, como su nombre lo indica, son los caminos más largos, de cada grafo RDF. Si los subgrafos no son isomorfos, no es necesario verificar todos los nodos y arcos de los grafos RDF.

Se considera conveniente experimentar con el algoritmo LPS como preprocesamiento de los grafos RDF a ser tratados por el algoritmo de la Universidad Yunnan.

La implementación del algoritmo LPS, modificado para el estudio, se encuentra en los anexos correspondientes.

6.9 LPS: Longest Path Subgraph: Aplication.

Como mejora para el resultado del algoritmo de nuestro estudio, se pretende utilizar el algoritmo LPS para realizar un preprocesamiento a los grafos Query y Target y obtener su camino más largo. Luego, comparar con la fórmula de similaridad las tripletas que conforman el camino más largo de ambos grafos, para determinar si son isomorfos o no.

Al momento de replicar las pruebas preliminares realizadas al algoritmo RDF (capítulo 11), descubrimos un detalle importantísimo.

Siguienda la misma estructura base del grafo, expuesta en el capítulo 11, aumentamos el número de nodos a comparar de 10 a 100. Por lo tanto, se trata de una comparación de 100 x 100 nodos del grafo Query contra el grafo Target.

Lamentablemente, el computador donde se probó el algoritmo se quedó rápidamente sin memoria y no pudo seguir calculando el resto de comparaciones.

Se determinó el punto de quiebre en 20 Nodos: Al comparar 20 nodos versus 20 nodos, el algoritmo corre sin problemas y entrega el resultado esperado. Sim embargo más de 20 nodos no es capaz de comparar.

Dado el alto número de comparaciones que debe calcular, no es extraño que suceda.

Volvemos a correr las pruebas con 100 vs 100 Nodos, pero esta vez incluímos sólo palabras que no están en WordNet, de esta forma, el algoritmo sólo calculará la similaridad de cadena de dichas palabras y no entrará en los archivos de WordNet.

Tal como esperábamos el algoritmo, calcula sin problemas los 100 vs 100 nodos, entregando el resultado correcto y esperado de la similaridad máxima 1. Por lo tanto, el problema estaría en los archivos de WordNet, más precisamente en el acceso a ellos, pues deben ser accedidos al momento

de realizar una comparación entre palabras, debiendo abrir y cerrar oportunamente los archivos de texto, agotando la memoria rápidamente.

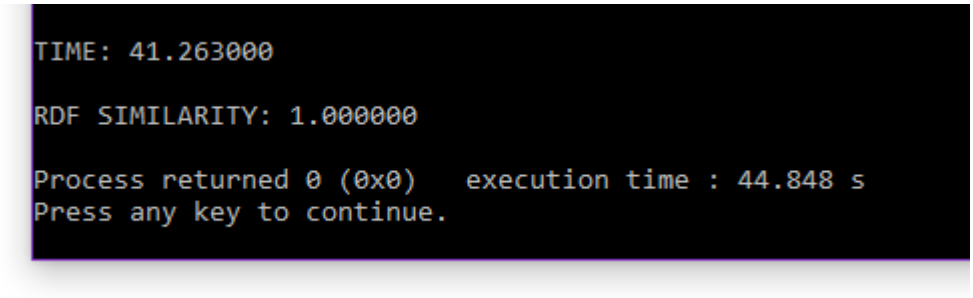
Lo que prueba el alto costo de una solución de este tipo.

Para combatir el impedimento, se tomaron dos medidas, la primera corresponde a una reselección de palabras incluídas en WordNet, pero que tuvieran una sola acepción, vale decir, un solo $offset$, lo que se traduce en menos comparaciones para el algoritmo, pues bastaría con una sola comparación entre dos palabras para determinar su similaridad. En los anexos A y B se presentan los archivos RDF que fueron creados a partir de esta medida, para objeto de las pruebas.

La segunda medida implementada fue la creación de una versión alternativa, para probar efectivamente que el problema son los archivos de WordNet.

6.10 Prueba Alternativa: WordNet Sin WordNet

El objetivo de esta prueba es simular el caso en que dos grafos: Query y Target son iguales: Tienen la misma estructura y los mismos datos. 100 Nodos en cada grafo con palabras contenidas en WordNet. La idea es buscar los datos de las palabras de los nodos en WordNet y almacenarlos en una estructura en memoria, desde donde se realizarán todas las consultas y comparaciones, de esta forma, sólo realizamos una llamada a los archivos de WordNet para cargar la estructura con los datos que sabemos que serán necesarios para el cálculo de similaridad de ambos grafos.



El resultado muestra que efectivamente el problema era el acceso excesivo a los archivos de WordNet. En esta instancia bastó probar la simulación con el algoritmo RDF, pues el que hace el mayor número de comparaciones, para comprobar que sin el uso reiterativo de los archivos de WordNet es viable una solución que maneje su información.

6.11 Pruebas Con Sinónimos.

Ahora que ya sabemos que podemos ocupar WordNet sin problemas, si implementamos una estructura capaz de albergar sus datos de forma eficiente, podemos ocupar parte de la potencialidad que ofrece la codificación de WordNet.

En esta simulación probaremos el reconocimiento de sinónimos de WordNet, con el fin de determinar si es viable o no una solución de este tipo.

En la siguiente simulación se presentará el caso de dos grafos: un grafo query y un grafo target. Estos grafos tendrán la misma estructura de árboles, con un número de arcos máximo de tres.

Ambos grafos contendrán sinónimos, vale decir, para una palabra de un nodo X del grafo target, habrá un sinónimo en el mismo nodo, pero del grafo target. La idea es utilizar la base de datos de Wordnet para que reconozca estas dos palabras diferentes, como sinónimos. Para probar esto, la estructura de los dos grafos será la misma y sólo variarán las palabras contenidas en el recurso descrito por cada nodo. Con esta simulación se pretende determinar la capacidad del algoritmo para detectar los sinónimos, su comportamiento en general y el tiempo de ejecución, contrastando las versiones RDF y LPS. La versión RDF corresponde a la implementación del algoritmo orientado a la similaridad, propuesto por los docentes de la Universidad de Yunnan, mientras que la versión LPS corresponde al algoritmo RDF con las mejoras basadas en el algoritmo LPS para el preprocesamiento de los grafos. Esta prueba se realizará para 20, 30, 40, 50, 60, 70, 80 y 90 nodos.

Pruebas con 20 nodos

Los grafos query y target tienen una estructura de árbol (ver Figura 27). Cada nodo numerado de la estructura, corresponde a un recurso en particular. Los nodos de ambos grafos pueden ser vistos detalladamente en el anexo Nodos Grafos Query y Target.

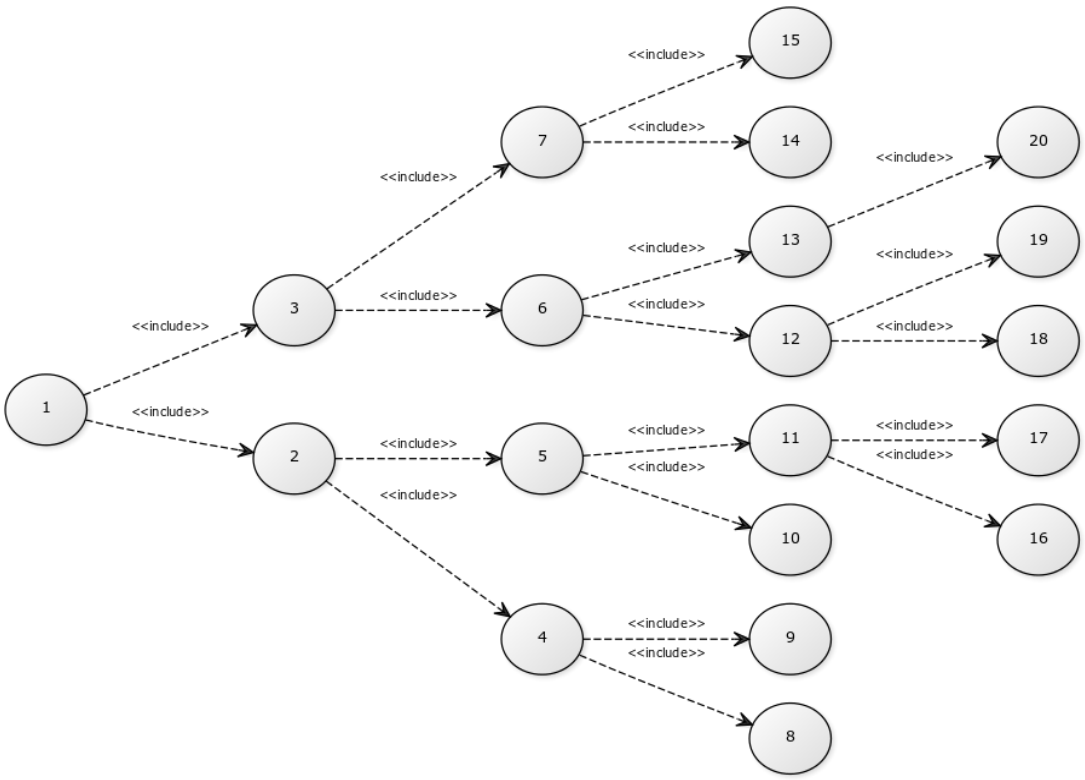


Figura 27 Estructura de árbol de 20 Nodos.

Resultados 20 nodos

RDF

```
TIME: 5.672000

RDF SIMILARITY: 1.000000

Process returned 0 (0x0)   execution time : 6.480 s
Press any key to continue.
```

El primer tiempo (5.672000 s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

LPS

```
TIME: 0.655000


LPS SIMILARITY: 1.000000

Process returned 0 (0x0)   execution time : 1.349 s
Press any key to continue.
```

El primero tiempo (0.655000 s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

Pruebas con 30 nodos

Los nodos de ambos grafos: Query y Target, tienen una estructura de árbol (ver Figura 28). Cada nodo numerado de la estructura, corresponde a un recurso en particular. Los nodos de ambos grafos pueden ser vistos detalladamente en el anexo Nodos Grafos Query y Target.

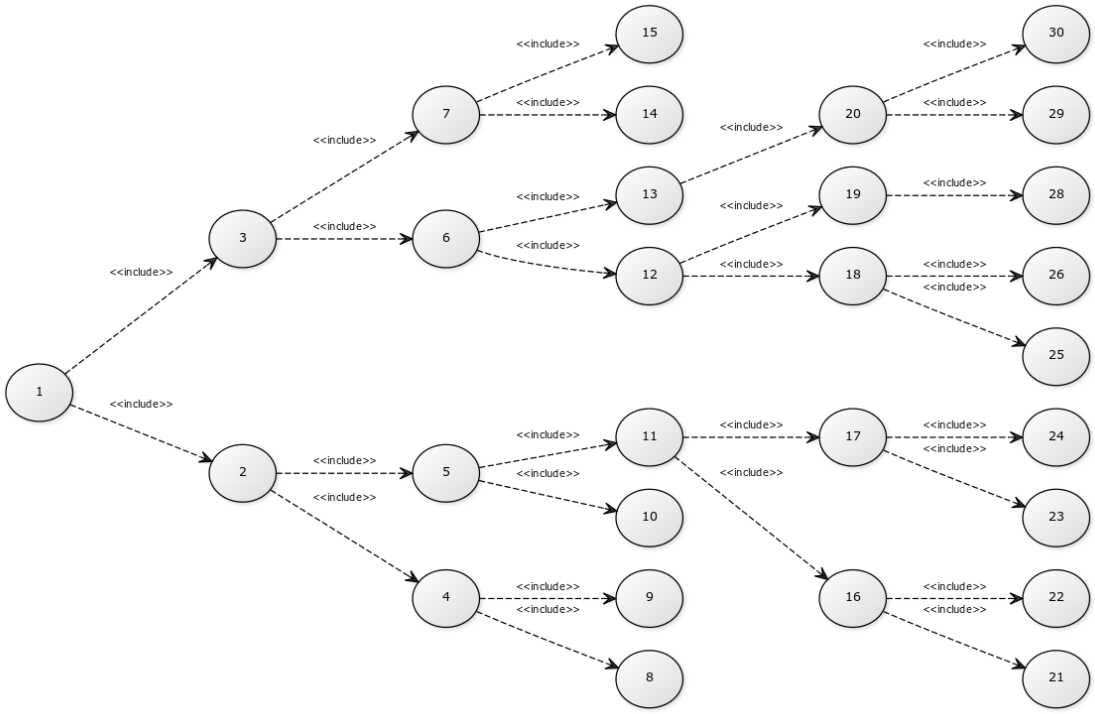


Figura 28 Estructura de árbol de 30 nodos.

Resultados 30 nodos

RDF

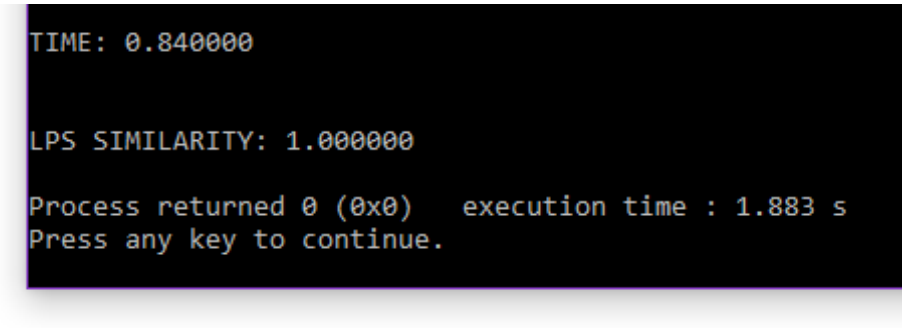
```
TIME: 10.578000

RDF SIMILARITY: 1.000000

Process returned 0 (0x0)   execution time : 11.756 s
Press any key to continue.
```

El primer tiempo (10.578000 s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

LPS



El primer tiempo (0.840000 s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

Pruebas con 40 nodos

Los nodos de ambos grafos: Query y Target, tienen una estructura de árbol (ver Figura 29). Cada nodo numerado de la estructura, corresponde a un recurso en particular. Los nodos de ambos grafos pueden ser vistos detalladamente en el anexo Nodos Grafos Query y Target.

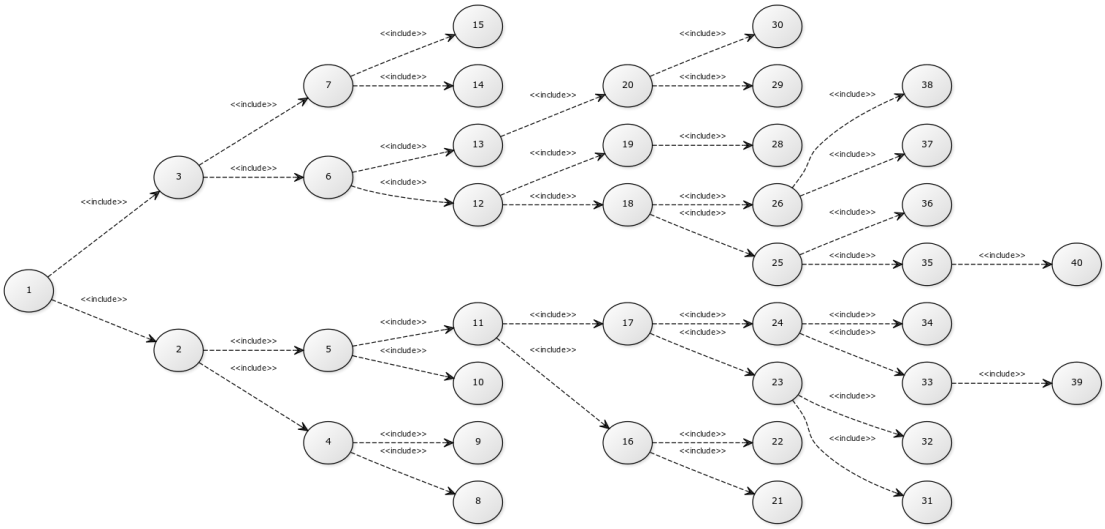


Figura 29 Estructura de árbol de 40 nodos.

Resultados 40 nodos

RDF

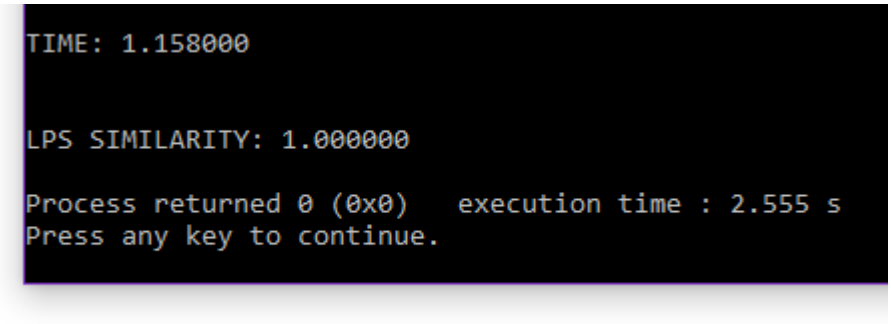
```
TIME: 17.712000

RDF SIMILARITY: 1.000000

Process returned 0 (0x0)   execution time : 19.212 s
Press any key to continue.
```

El primer tiempo (17.712000 s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

LPS



El primer tiempo (1.158000 s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

Pruebas con 50 nodos

Los nodos de ambos grafos: Query y Target, tienen una estructura de árbol (ver Figura 30). Cada nodo numerado de la estructura, corresponde a un recurso en particular. Los nodos de ambos grafos pueden ser vistos detalladamente en el anexo Nodos Grafos Query y Target.

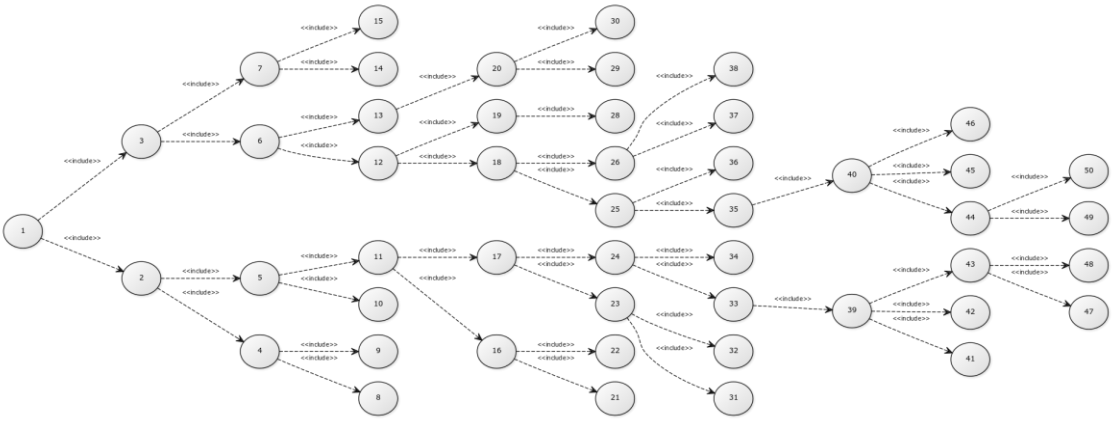


Figura 30 Estructura de árbol de 50 nodos.

Resultado 50 nodos

RDF

```
TIME: 27.238000

RDF SIMILARITY: 1.000000

Process returned 0 (0x0)   execution time : 28.987 s
Press any key to continue.
```

El primer tiempo (17.712000 s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

LPS

```
TIME: 1.592000


LPS SIMILARITY: 1.000000

Process returned 0 (0x0)   execution time : 3.347 s
Press any key to continue.
```

El primer tiempo (1.144000 s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

La similaridad entregada es de 0.142857. Más adelante analizaremos los resultados.

Pruebas con 60 nodos

Los nodos de ambos grafos: Query y Target, tienen una estructura de árbol. Cada nodo numerado de la estructura, corresponde a un recurso en particular. Los nodos de ambos grafos pueden ser vistos detalladamente en el anexo Nodos Grafos Query y Target.

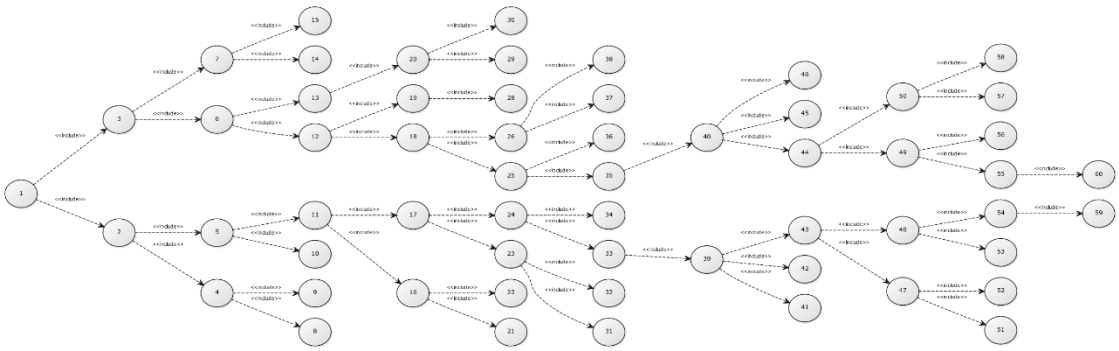


Figura 31 Estructura de árbol de 60 nodos.

Resultados 60 nodos

RDF

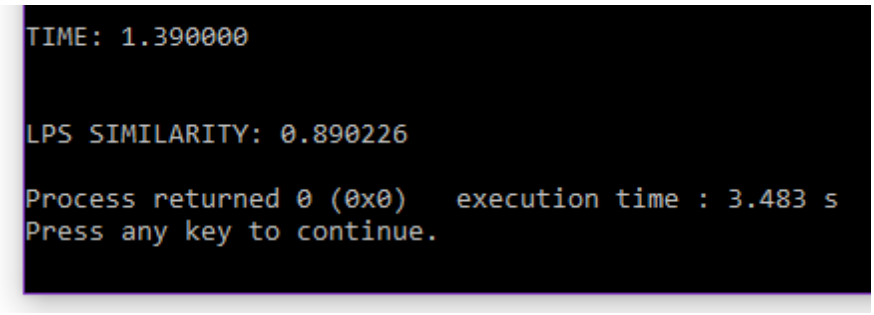
```
TIME: 38.690000

RDF SIMILARITY: 1.000000

Process returned 0 (0x0)   execution time : 40.877 s
Press any key to continue.
```

El primer tiempo (38.690000 s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

LPS



El primer tiempo (1.144000 s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

La similaridad entregada es de 0.890226. Más adelante analizaremos los resultados.

Pruebas con 70 nodos

Los nodos de ambos grafos: Query y Target, tienen una estructura de árbol. Cada nodo numerado de la estructura, corresponde a un recurso en particular. Los nodos de ambos grafos pueden ser vistos detalladamente en el anexo Nodos Grafos Query y Target.

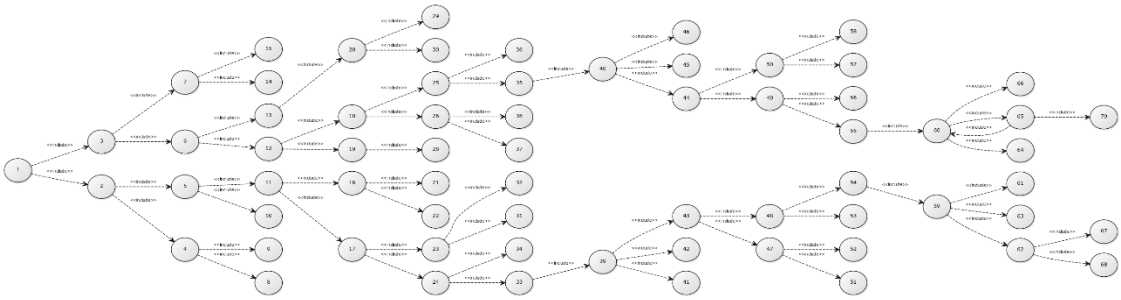


Figura 32 Estructura de árbol de 70 nodos.

Resultados 70 nodos

RDF

```
TIME: 52.962000

RDF SIMILARITY: 1.000000

Process returned 0 (0x0)   execution time : 55.543 s
Press any key to continue.
```

El primer tiempo (52.962000 s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

LPS

```
TIME: 1.201000


LPS SIMILARITY: 0.890226

Process returned 0 (0x0)   execution time : 3.645 s
Press any key to continue.
```

El primer tiempo (1.201000 s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

La similaridad entregada es de 0.890226. Más adelante analizaremos los resultados.

Pruebas con 80 nodos

Los nodos de ambos grafos: Query y Target, tienen una estructura de árbol (ver Figura 33). Cada nodo numerado de la estructura, corresponde a un recurso en particular. Los nodos de ambos grafos pueden ser vistos detalladamente en el anexo Nodos Grafos Query y Target.

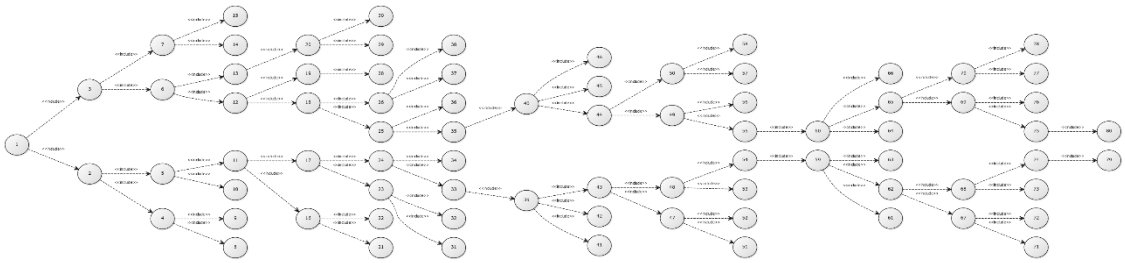


Figura 33 Estructura de árbol de 80 nodos.

Resultados 80 nodos

RDF

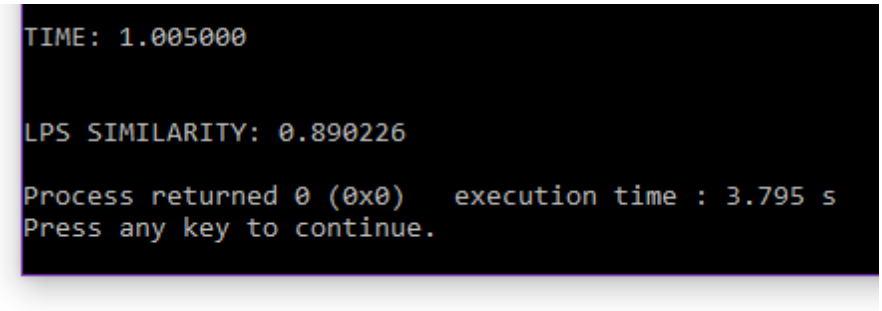
```
TIME: 59.627000

RDF SIMILARITY: 1.000000

Process returned 0 (0x0)   execution time : 62.392 s
Press any key to continue.
```

El primer tiempo (59.627000 s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

LPS



El primer tiempo (1.005000 s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

La similaridad entregada es de 0.890226. Más adelante analizaremos los resultados.

Pruebas con 90 nodos

Los nodos de ambos grafos: Query y Target, tienen una estructura de árbol (ver Figura 34). Cada nodo numerado de la estructura, corresponde a un recurso en particular. Los nodos de ambos grafos pueden ser vistos detalladamente en el anexo Nodos Grafos Query y Target.

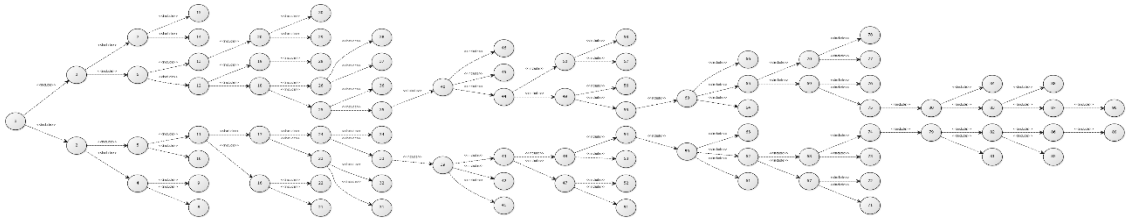


Figura 34 Estructura de árbol de 90 nodos.

Resultados 90 nodos

RDF

```
TIME: 77.862000

RDF SIMILARITY: 1.000000

Process returned 0 (0x0)   execution time : 80.653 s
Press any key to continue.
```

El primer tiempo (77.862000 s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

LPS

```
TIME: 0.878000


LPS SIMILARITY: 0.890226

Process returned 0 (0x0)   execution time : 4.073 s
Press any key to continue.
```

El primer tiempo (0.878000s) corresponde al tiempo de ejecución de la función de similaridad, mientras que el segundo tiempo es el tiempo de ejecución total del programa.

La similaridad entregada es de 0.890226. Más adelante analizaremos los resultados.

Explicación resultados LPS

	Similaridad (0 - 1)		Tiempo (s)	
Nº Nodos	**RDF**	**LPS**	**RDF**	**LPS**
20	1	1	5,672	0,655
30	1	1	10,578	0,84
40	1	1	17,712	1,158
50	1	1	27,238	1,592
60	1	0,89	38,69	1,39
70	1	0,89	52,962	1,201
80	1	0,89	59,627	1,005
90	1	0,89	77,862	0,878

En las pruebas anteriores, en cuanto a tiempos de ejecución, se obtuvo resultados muy favorables, superando el algoritmo LPS al RDF en todas las pruebas realizadas.

Sin embargo, en cuanto a la similaridad entregada, hubo casos (60, 70, 80 y 90 nodos) donde la similaridad entregada por el algoritmo LPS fue sólo del 80% en vez de mostrar el resultado esperado del 100%.

Por otro lado, nos parece favorable la similaridad entregada por el algoritmo LPS, con respecto a su tiempo de ejecución, muy por encima del algoritmo RDF, que si bien entrega el resultado esperado con exactitud, es demasiado costoso.

Analizando los datos, descubrimos que esto se debe a que el algoritmo LPS crea los caminos más largos de cada grafo de manera independiente, teniendo en cuenta el número de nodos y el peso individual y acumulado de cada tripleta que compone los caminos.

Por dicha razón, puede darse el caso que los caminos más largos (LPS) de cada grafo, no sean necesariamente los mismos. Para ejemplificar utilizaremos el caso de 90 Nodos.

```
- *** LPS QUERY *** -


*** MOSTRAR LPS ***

(https://en.wikipedia.org/wiki/da_vinci) - (https://www.wikidata.org/wiki/Property:P001) - (https://en.wikipedia.org/wiki/abdominal_aortic_aneurysm)
COSTO TOTAL: 217631
(https://en.wikipedia.org/wiki/abdominal_aortic_aneurysm) - (https://www.wikidata.org/wiki/Property:P004) - (https://en.wikipedia.org/wiki/aalborg)
COSTO TOTAL: 204477
(https://en.wikipedia.org/wiki/aalborg) - (https://www.wikidata.org/wiki/Property:P010) - (https://en.wikipedia.org/wiki/copland)
COSTO TOTAL: 191421
(https://en.wikipedia.org/wiki/copland) - (https://www.wikidata.org/wiki/Property:P016) - (https://en.wikipedia.org/wiki/abattoir)
COSTO TOTAL: 180282
(https://en.wikipedia.org/wiki/abattoir) - (https://www.wikidata.org/wiki/Property:P023) - (https://en.wikipedia.org/wiki/abdominoplasty)
COSTO TOTAL: 169011
(https://en.wikipedia.org/wiki/abdominoplasty) - (https://www.wikidata.org/wiki/Property:P032) - (https://en.wikipedia.org/wiki/velvetweed)
COSTO TOTAL: 156969
(https://en.wikipedia.org/wiki/velvetweed) - (https://www.wikidata.org/wiki/Property:P038) - (https://en.wikipedia.org/wiki/acacia_dealbata)
COSTO TOTAL: 144698
(https://en.wikipedia.org/wiki/acacia_dealbata) - (https://www.wikidata.org/wiki/Property:P042) - (https://en.wikipedia.org/wiki/abies_lasiocarpa)
COSTO TOTAL: 132428
(https://en.wikipedia.org/wiki/abies_lasiocarpa) - (https://www.wikidata.org/wiki/Property:P047) - (https://en.wikipedia.org/wiki/ablative)
COSTO TOTAL: 119580
(https://en.wikipedia.org/wiki/ablative) - (https://www.wikidata.org/wiki/Property:P053) - (https://en.wikipedia.org/wiki/abolition)
COSTO TOTAL: 107390
(https://en.wikipedia.org/wiki/abolition) - (https://www.wikidata.org/wiki/Property:P058) - (https://en.wikipedia.org/wiki/abor)
COSTO TOTAL: 95908
(https://en.wikipedia.org/wiki/abor) - (https://www.wikidata.org/wiki/Property:P061) - (https://en.wikipedia.org/wiki/abraham)
COSTO TOTAL: 84841
(https://en.wikipedia.org/wiki/abraham) - (https://www.wikidata.org/wiki/Property:P067) - (https://en.wikipedia.org/wiki/abrocoma)
COSTO TOTAL: 74025
(https://en.wikipedia.org/wiki/abrocoma) - (https://www.wikidata.org/wiki/Property:P073) - (https://en.wikipedia.org/wiki/abronia_villosa)
COSTO TOTAL: 62787
(https://en.wikipedia.org/wiki/abronia_villosa) - (https://www.wikidata.org/wiki/Property:P078) - (https://en.wikipedia.org/wiki/absinthe_oil)
COSTO TOTAL: 50679
(https://en.wikipedia.org/wiki/absinthe_oil) - (https://www.wikidata.org/wiki/Property:P081) - (https://en.wikipedia.org/wiki/absorbent_material)
COSTO TOTAL: 38137
(https://en.wikipedia.org/wiki/absorbent_material) - (https://www.wikidata.org/wiki/Property:P085) - (https://en.wikipedia.org/wiki/abstractionist)
COSTO TOTAL: 25288
(https://en.wikipedia.org/wiki/abstractionist) - (https://www.wikidata.org/wiki/Property:P088) - (https://en.wikipedia.org/wiki/absurdity)
COSTO TOTAL: 12186


*** FIN LPS ***
```

El camino LPS del grafo Query de 90 Nodos

```
- *** LPS TARGET *** -


*** MOSTRAR LPS ***

(https://en.wikipedia.org/wiki/leonardo_da_vinci) - (https://www.wikidata.org/wiki/Property:P002) - (https://en.wikipedia.org/wiki/aken)
COSTO TOTAL: 223508
(https://en.wikipedia.org/wiki/aken) - (https://www.wikidata.org/wiki/Property:P005) - (https://en.wikipedia.org/wiki/aalost)
COSTO TOTAL: 211642
(https://en.wikipedia.org/wiki/aalost) - (https://www.wikidata.org/wiki/Property:P011) - (https://en.wikipedia.org/wiki/ear-shell)
COSTO TOTAL: 200905
(https://en.wikipedia.org/wiki/ear-shell) - (https://www.wikidata.org/wiki/Property:P017) - (https://en.wikipedia.org/wiki/prioress)
COSTO TOTAL: 189693
(https://en.wikipedia.org/wiki/prioress) - (https://www.wikidata.org/wiki/Property:P024) - (https://en.wikipedia.org/wiki/paunchiness)
COSTO TOTAL: 178232
(https://en.wikipedia.org/wiki/paunchiness) - (https://www.wikidata.org/wiki/Property:P034) - (https://en.wikipedia.org/wiki/abyssinian_cat)
COSTO TOTAL: 166481
(https://en.wikipedia.org/wiki/abyssinian_cat) - (https://www.wikidata.org/wiki/Property:P039) - (https://en.wikipedia.org/wiki/golden_wattle)
COSTO TOTAL: 154136
(https://en.wikipedia.org/wiki/golden_wattle) - (https://www.wikidata.org/wiki/Property:P043) - (https://en.wikipedia.org/wiki/jebel_musa)
COSTO TOTAL: 141586
(https://en.wikipedia.org/wiki/jebel_musa) - (https://www.wikidata.org/wiki/Property:P048) - (https://en.wikipedia.org/wiki/able-bodied_seaman)
COSTO TOTAL: 129474
(https://en.wikipedia.org/wiki/able-bodied_seaman) - (https://www.wikidata.org/wiki/Property:P054) - (https://en.wikipedia.org/wiki/emancipationist)
COSTO TOTAL: 116954
(https://en.wikipedia.org/wiki/emancipationist) - (https://www.wikidata.org/wiki/Property:P059) - (https://en.wikipedia.org/wiki/abortion-inducing_drug)
COSTO TOTAL: 103876
(https://en.wikipedia.org/wiki/abortion-inducing_drug) - (https://www.wikidata.org/wiki/Property:P064) - (https://en.wikipedia.org/wiki/european_bream)
COSTO TOTAL: 90296
(https://en.wikipedia.org/wiki/european_bream) - (https://www.wikidata.org/wiki/Property:P068) - (https://en.wikipedia.org/wiki/chinchilla_rat)
COSTO TOTAL: 76851
(https://en.wikipedia.org/wiki/chinchilla_rat) - (https://www.wikidata.org/wiki/Property:P074) - (https://en.wikipedia.org/wiki/breaking_off)
COSTO TOTAL: 64226
(https://en.wikipedia.org/wiki/breaking_off) - (https://www.wikidata.org/wiki/Property:P079) - (https://en.wikipedia.org/wiki/perfect_pitch)
COSTO TOTAL: 51836
(https://en.wikipedia.org/wiki/perfect_pitch) - (https://www.wikidata.org/wiki/Property:P082) - (https://en.wikipedia.org/wiki/coefficient_of_absorption)
COSTO TOTAL: 39526
(https://en.wikipedia.org/wiki/coefficient_of_absorption) - (https://www.wikidata.org/wiki/Property:P086) - (https://en.wikipedia.org/wiki/reasoning)
COSTO TOTAL: 25824
(https://en.wikipedia.org/wiki/reasoning) - (https://www.wikidata.org/wiki/Property:P089) - (https://en.wikipedia.org/wiki/capital_of_nigeria)
COSTO TOTAL: 12528


*** FIN LPS ***
```

El camino LPS del grafo Target de 90 Nodos

Los caminos son totalmente diferentes, recordando el esquema de la estructura (Figura 34). Ambos LPS escogieron caminos distintos desde la bifurcación del primer nodo, lo que los vuelve diferentes. Esto se debe a que en la totalidad estructural se distingue dos grandes caminos con igual cantidad de nodos, el algoritmo LPS debe entonces discriminar basándose en la función Hash que pondera cada nodo y cada camino. En este caso, la función determinó independientemente para cada grafo, caminos distintos. Al tener sinónimos en cada grafo, no se puede asegurar que el camino escogido por el algoritmo será exactamente el mismo que su contraparte.

Sin embargo el algoritmo determina que tienen una similar cercana al 80%. ¿Por qué pasa esto? –Hay que tener en cuenta que la fórmula del algoritmo considera además la similaridad estructural, la similaridad de las tripletas y la de sus recursos (sujeto, predicado y objeto).

Con esto en cuenta, ya no nos parece tan descabellado el resultado, sobretodo sabiendo que cada recurso alude a la misma página web, diferenciándose sólo en 1/3 de la dirección URI.

Para probar esta teoría es necesario, entonces, simular 1 sólo camino en la estructura de ambos grafos, que el algoritmo deberá encontrar, y para asegurarse que sea el mismo en ambos grafos, quedará determinado por el número de nodos.

Pruebas con 90 nodos con sólo 1 camino potencial

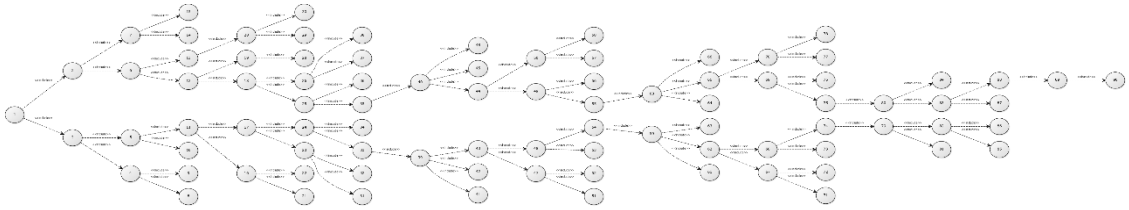


Figura 35 Nueva estructura para los 90 Nodos.

A simple vista, en la estructura se aprecia que el camino superior tiene un mayor número de nodos. Será eses, entonces el que el algoritmo LPS deberá encontrar y comparar con su contraparte que tendrá los sinónimos de sus recursos.

Resultados 90 nodos con solo 1 camino

RDF

```
TIME: 70.312000

RDF SIMILARITY: 1.000000

Process returned 0 (0x0)   execution time : 73.129 s
Press any key to continue.
```

LPS

```
TIME: 0.531000


LPS SIMILARITY: 1.000000

Process returned 0 (0x0)   execution time : 1.813 s
Press any key to continue.
```

Esta vez sí se obtiene el resultado esperado de la simulación, pues esta vez sí se obtuvo el mismo camino para ambos grafos.

```
- *** LPS QUERY *** -


*** MOSTRAR LPS ***

(https://en.wikipedia.org/wiki/da_vinci) - (https://www.wikidata.org/wiki/Property:P002) - (https://en.wikipedia.org/wiki/aachen)
COSTO TOTAL: 223170
(https://en.wikipedia.org/wiki/aachen) - (https://www.wikidata.org/wiki/Property:P005) - (https://en.wikipedia.org/wiki/alost)
COSTO TOTAL: 212058
(https://en.wikipedia.org/wiki/alost) - (https://www.wikidata.org/wiki/Property:P011) - (https://en.wikipedia.org/wiki/abalone)
COSTO TOTAL: 201225
(https://en.wikipedia.org/wiki/abalone) - (https://www.wikidata.org/wiki/Property:P017) - (https://en.wikipedia.org/wiki/abbess)
COSTO TOTAL: 190281
(https://en.wikipedia.org/wiki/abbess) - (https://www.wikidata.org/wiki/Property:P024) - (https://en.wikipedia.org/wiki/abdominousness)
COSTO TOTAL: 179254
(https://en.wikipedia.org/wiki/abdominousness) - (https://www.wikidata.org/wiki/Property:P034) - (https://en.wikipedia.org/wiki/abyssinian)
COSTO TOTAL: 167437
(https://en.wikipedia.org/wiki/abyssinian) - (https://www.wikidata.org/wiki/Property:P039) - (https://en.wikipedia.org/wiki/acacia_pycnantha)
COSTO TOTAL: 155170
(https://en.wikipedia.org/wiki/acacia_pycnantha) - (https://www.wikidata.org/wiki/Property:P043) - (https://en.wikipedia.org/wiki/abila)
COSTO TOTAL: 142757
(https://en.wikipedia.org/wiki/abila) - (https://www.wikidata.org/wiki/Property:P048) - (https://en.wikipedia.org/wiki/able_seaman)
COSTO TOTAL: 130917
(https://en.wikipedia.org/wiki/able_seaman) - (https://www.wikidata.org/wiki/Property:P054) - (https://en.wikipedia.org/wiki/abolitionist)
COSTO TOTAL: 119599
(https://en.wikipedia.org/wiki/abolitionist) - (https://www.wikidata.org/wiki/Property:P059) - (https://en.wikipedia.org/wiki/abortifacient)
COSTO TOTAL: 107492
(https://en.wikipedia.org/wiki/abortifacient) - (https://www.wikidata.org/wiki/Property:P064) - (https://en.wikipedia.org/wiki/abramis_brama)
COSTO TOTAL: 95137
(https://en.wikipedia.org/wiki/abramis_brama) - (https://www.wikidata.org/wiki/Property:P068) - (https://en.wikipedia.org/wiki/abrocome)
COSTO TOTAL: 82738
(https://en.wikipedia.org/wiki/abrocome) - (https://www.wikidata.org/wiki/Property:P074) - (https://en.wikipedia.org/wiki/abruption)
COSTO TOTAL: 70866
(https://en.wikipedia.org/wiki/abruption) - (https://www.wikidata.org/wiki/Property:P079) - (https://en.wikipedia.org/wiki/absolute_pitch)
COSTO TOTAL: 59362
(https://en.wikipedia.org/wiki/absolute_pitch) - (https://www.wikidata.org/wiki/Property:P082) - (https://en.wikipedia.org/wiki/absorptance)
COSTO TOTAL: 47199
(https://en.wikipedia.org/wiki/absorptance) - (https://www.wikidata.org/wiki/Property:P087) - (https://en.wikipedia.org/wiki/abstracter)
COSTO TOTAL: 34852
(https://en.wikipedia.org/wiki/abstracter) - (https://www.wikidata.org/wiki/Property:P088) - (https://en.wikipedia.org/wiki/absurdity)
COSTO TOTAL: 22927
(https://en.wikipedia.org/wiki/absurdity) - (https://www.wikidata.org/wiki/Property:P089) - (https://en.wikipedia.org/wiki/abuja)
COSTO TOTAL: 11188


*** FIN LPS ***
```

LPS Grafo Query 90 Nodos

```
- *** LPS TARGET *** -


*** MOSTRAR LPS ***

(https://en.wikipedia.org/wiki/leonardo_da_vinci) - (https://www.wikidata.org/wiki/Property:P002) - (https://en.wikipedia.org/wiki/aken)
COSTO TOTAL: 236484
(https://en.wikipedia.org/wiki/aken) - (https://www.wikidata.org/wiki/Property:P005) - (https://en.wikipedia.org/wiki/aalost)
COSTO TOTAL: 224618
(https://en.wikipedia.org/wiki/aalost) - (https://www.wikidata.org/wiki/Property:P011) - (https://en.wikipedia.org/wiki/ear-shell)
COSTO TOTAL: 213881
(https://en.wikipedia.org/wiki/ear-shell) - (https://www.wikidata.org/wiki/Property:P017) - (https://en.wikipedia.org/wiki/prioress)
COSTO TOTAL: 202669
(https://en.wikipedia.org/wiki/prioress) - (https://www.wikidata.org/wiki/Property:P024) - (https://en.wikipedia.org/wiki/paunchiness)
COSTO TOTAL: 191208
(https://en.wikipedia.org/wiki/paunchiness) - (https://www.wikidata.org/wiki/Property:P034) - (https://en.wikipedia.org/wiki/abyssinian_cat)
COSTO TOTAL: 179457
(https://en.wikipedia.org/wiki/abyssinian_cat) - (https://www.wikidata.org/wiki/Property:P039) - (https://en.wikipedia.org/wiki/golden_wattle)
COSTO TOTAL: 167112
(https://en.wikipedia.org/wiki/golden_wattle) - (https://www.wikidata.org/wiki/Property:P043) - (https://en.wikipedia.org/wiki/jebel_musa)
COSTO TOTAL: 154562
(https://en.wikipedia.org/wiki/jebel_musa) - (https://www.wikidata.org/wiki/Property:P048) - (https://en.wikipedia.org/wiki/able-bodied_seaman)
COSTO TOTAL: 142450
(https://en.wikipedia.org/wiki/able-bodied_seaman) - (https://www.wikidata.org/wiki/Property:P054) - (https://en.wikipedia.org/wiki/emancipationist)
COSTO TOTAL: 129930
(https://en.wikipedia.org/wiki/emancipationist) - (https://www.wikidata.org/wiki/Property:P059) - (https://en.wikipedia.org/wiki/abortion-inducing_drug)
COSTO TOTAL: 116852
(https://en.wikipedia.org/wiki/abortion-inducing_drug) - (https://www.wikidata.org/wiki/Property:P064) - (https://en.wikipedia.org/wiki/european_bream)
COSTO TOTAL: 103272
(https://en.wikipedia.org/wiki/european_bream) - (https://www.wikidata.org/wiki/Property:P068) - (https://en.wikipedia.org/wiki/chinchilla_rat)
COSTO TOTAL: 89827
(https://en.wikipedia.org/wiki/chinchilla_rat) - (https://www.wikidata.org/wiki/Property:P074) - (https://en.wikipedia.org/wiki/breaking_off)
COSTO TOTAL: 77202
(https://en.wikipedia.org/wiki/breaking_off) - (https://www.wikidata.org/wiki/Property:P079) - (https://en.wikipedia.org/wiki/perfect_pitch)
COSTO TOTAL: 64812
(https://en.wikipedia.org/wiki/perfect_pitch) - (https://www.wikidata.org/wiki/Property:P082) - (https://en.wikipedia.org/wiki/coefficient_of_absorption)
COSTO TOTAL: 52502
(https://en.wikipedia.org/wiki/coefficient_of_absorption) - (https://www.wikidata.org/wiki/Property:P087) - (https://en.wikipedia.org/wiki/abstractor)
COSTO TOTAL: 38800
(https://en.wikipedia.org/wiki/abstractor) - (https://www.wikidata.org/wiki/Property:P088) - (https://en.wikipedia.org/wiki/ridiculousness)
COSTO TOTAL: 25392
(https://en.wikipedia.org/wiki/ridiculousness) - (https://www.wikidata.org/wiki/Property:P089) - (https://en.wikipedia.org/wiki/capital_of_nigeria)
COSTO TOTAL: 13094


*** FIN LPS ***
```

LPS Grafo Target 90 Nodos.

Conclusiones

Como conclusiones de este arduo trabajo investigativo cabe resaltar el cumplimiento de todos los objetivos planteados al iniciar esta tesis. En cuanto a los objetivos específicos de esta investigación:

- Estudiar la problemática de la generación de subgrafos a partir de grafos.

No sólo se estudió la problemática de generación de subgrafos, sino que además se implementaron algoritmos que se encargan de esa tarea: se debió analizar, diseñar e implementar adaptaciones para el algoritmo orientado a la similaridad, así como también para el algoritmo LPS y sobretodo desarrollar las Funciones y estructuras que nos permitieran trabajar con WordNet de la forma en que era requerido, pues no había información al respecto y se debió trabajar desde cero.

- Estudiar las características de los grafos RDF y del lenguaje SPARQL.

Los capítulos 2 y 3 se encargan de cubrir la información básica necesaria para comprender los grafos RDF y es una parte fundamental para entender los artículos abordados en esta investigación.

El lenguaje Sparql también se incluye dentro del capítulo 3, pero fue descartado a la hora de implementar nuestra solución ya que era básicamente un motor de búsqueda, y era precisamente eso lo que a grandes rasgos, debíamos implementar, y las limitaciones propias del lenguaje no permitían desarrollar las funcionalidades necesarias. Por esta misma razón nos decantamos por el lenguaje C, que es mucho más versátil, permitiéndonos manipular completamente las estructuras de almacenamiento.

- Estudiar y analizar diferentes aproximaciones de isomorfismos de grafos RDF.

El isomorfismo de grafos se vio cubierto con el estudio, análisis e implementación de las 3 macro funciones que contempla nuestra herramienta: El algoritmo orientado a la similaridad, el algoritmo LPS y la algoritmia necesaria para hacer correr la similaridad de WordNet.

- Analizar, diseñar e implementar los algoritmos para dichas problemáticas.

Para las problemáticas descritas en el punto anterior, se analizaron, diseñaron e implementaron los algoritmos correspondientes, vale decir, Algoritmo orientado a la similaridad, Algoritmo LPS y algoritmos de WordNet.

- Contrastar nuestra propuesta con otra propuesta en la literatura.

Este punto en especial fue cumplido desde el comienzo, pues la investigación se basa en el análisis comparativo del algoritmo orientado a la similaridad, propuesto por la Universidad de Yunnan en China.

Por otro lado, a nivel personal, lo más sorprendente fue el crecimiento en cuanto a conocimiento alcanzado y la confianza en sí mismo, producto de creer en los objetivos planteados y luchar para lograrlos.

Lanzarse con los ojos cerrados contra lo desconocido puede resultar escalofriante, pero descubrir que uno cuenta con herramientas poderosísimas para enfrentarse a ello, lo vuelve exitante y, para sorpresa mía, bastante entretenido.

Nunca me había planteado la idea de realizar una investigación como actividad de título, pero se dieron las cosas y fue un verdadero agrado haberlo propuesto; gracias a la ayuda de mi profesor Guía me fui dando cuenta que aunque desconocía el tema, nunca fue un impedimento para trabajar en ello, sólo debía seguir investigando. Y eso fue lo que más me motivó al abordar temas

desconocidos para mí como el lenguaje RDF, los grafos RDF, el cálculo de la similaridad a nivel semántico y estructural, o WordNet y su tremendo potencial.

Sin lugar a dudas, ha sido un gran desafío y he aprendido mucho, pero así también comprendo que aún queda un sinfín de cosas por investigar, que la web semántica necesita gente con entusiasmo y nuevas ideas para seguir desarrollándose y llevarla al siguiente nivel.

Trabajos Futuros

Hay varias líneas investigativas que se podrían seguir de esta investigación. Como continuación directa de esta investigación, se podría trabajar en la implementación de una estructura de carga dinámica de la información de WordNet, para seguir desarrollando la potencialidad de WordNet. WordNet aún tiene mucha información desaprovechada, puede ser vista como una base de conocimiento de inteligencia artificial, se podría trabajar, además, en formas para reducir el costo de sus consultas. Para ello se podría investigar en la tecnología de las bases de datos en memoria, ya que cada vez se vuelve más barata.

También se podría trabajar en las inferencias del conocimiento de WordNet, o buscar alguna otra base de datos de conocimiento para crear motores de búsqueda con inteligencia artificial, para procesar el gran grafo de la Web semántica, cada vez más grande y vertiginoso.

Bibliografía

Bonilla, S. (s.f.). *Curso Matemáticas Discretas*. Obtenido de Clasificación de los grafos: https://sites.google.com/site/cursomatematicasdiscretas/5-2-clasificacion-de-los-grafos

D, L. (1998). An Information-Theoretic Definition of Similarity. *15th International Conference on Machine Learning.* San Francisco: Morgan Kaufmann.

Jeltsch, E. (1998). *Fundamentos de Informática Teórica*. Obtenido de Grafos Dirigidos: http://dns.uls.cl/~ej/fit_2009/teo_fit_2009/Lect_2009/Fit_1998/pagina_n9.htm

Lavenshtein. (1966). Binary code capable of correcting deletions, insertions, and reversals. *Soviet Physics-Doklady*, Vol. 10, NO.8.

Melnik, S. (2002). *Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching.* Obtenido de http://ilpubs.stanford.edu:8090/730/1/2002-1.pdf

Princeton University. (Marzo de 2017). *WordNet.* Obtenido de https://wordnet.princeton.edu/

Rodes, R. C. (2005). *Grafos. Nociones Básicas.* Obtenido de http://docencia.udea.edu.co/regionalizacion/teoriaderedes/definicionesu1.html

W3C. (12 de Abril de 2006). *Defining N-ary Relations on Semantic Web*. Obtenido de https://www.w3.org/TR/swbp-n-aryRelations/

W3C. (15 de Junio de 2008). *SPARQL is a recommendation*. Obtenido de https://www.w3.org/blog/SW/2008/01/15/sparql_is_a_recommendation/

W3C. (15 de Enero de 2008). *SPARQL Query Language for RDF*. Obtenido de https://www.w3.org/TR/rdf-sparql-query/

W3C. (25 de Febrero de 2014). *RDF 1.1 Concepts and Abstract Syntax.* Obtenido de W3C Recommendation: https://www.w3.org/TR/rdf11-concepts/

W3C. (2017). *W3C*. Obtenido de https://www.w3.org/

W3C. (s.f.). *SPARQL Working Group*. Obtenido de https://www.w3.org/2001/sw/DataAccess/homepage-20080115

Wikipedia. (2017). *Uniform Resource Identifier*. Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

ANEXO A Nodos Query – Sinónimos

1 https://en.wikipedia.org/wiki/da_vinci
2 https://en.wikipedia.org/wiki/abdominal_aortic_aneurysm
3 https://en.wikipedia.org/wiki/aachen
4 https://en.wikipedia.org/wiki/aland_islands
5 https://en.wikipedia.org/wiki/aalborg
6 https://en.wikipedia.org/wiki/alost
7 https://en.wikipedia.org/wiki/aare
8 https://en.wikipedia.org/wiki/aardvark
9 https://en.wikipedia.org/wiki/aardwolf
10 https://en.wikipedia.org/wiki/common_mullein
11 https://en.wikipedia.org/wiki/copland
12 https://en.wikipedia.org/wiki/abalone
13 https://en.wikipedia.org/wiki/abampere
14 https://en.wikipedia.org/wiki/foundling
15 https://en.wikipedia.org/wiki/abashment
16 https://en.wikipedia.org/wiki/abattis
17 https://en.wikipedia.org/wiki/abattoir
18 https://en.wikipedia.org/wiki/abbess
19 https://en.wikipedia.org/wiki/abbot
20 https://en.wikipedia.org/wiki/abridger
21 https://en.wikipedia.org/wiki/abdominal
22 https://en.wikipedia.org/wiki/abdominal_nerve_plexus
23 https://en.wikipedia.org/wiki/abdominocentesis
24 https://en.wikipedia.org/wiki/abdominoplasty
25 https://en.wikipedia.org/wiki/abdominousness
26 https://en.wikipedia.org/wiki/abducent
27 https://en.wikipedia.org/wiki/abducens_muscle
28 https://en.wikipedia.org/wiki/aspen_poplar
29 https://en.wikipedia.org/wiki/abelian_group
30 https://en.wikipedia.org/wiki/abelmoschus
31 https://en.wikipedia.org/wiki/aboulia
32 https://en.wikipedia.org/wiki/abutilon
33 https://en.wikipedia.org/wiki/velvetweed
34 https://en.wikipedia.org/wiki/abyss
35 https://en.wikipedia.org/wiki/abyssinian
36 https://en.wikipedia.org/wiki/abyssinian_banana
37 https://en.wikipedia.org/wiki/acacia_auriculiformis
38 https://en.wikipedia.org/wiki/acacia_cambegei
39 https://en.wikipedia.org/wiki/acacia_dealbata
40 https://en.wikipedia.org/wiki/acacia_pycnantha
41 https://en.wikipedia.org/wiki/abies_grandis

42 https://en.wikipedia.org/wiki/abies_concolor
43 https://en.wikipedia.org/wiki/abies_lasiocarpa
44 https://en.wikipedia.org/wiki/abila
45 https://en.wikipedia.org/wiki/abiogenesis
46 https://en.wikipedia.org/wiki/abjuration
47 https://en.wikipedia.org/wiki/abkhazian
48 https://en.wikipedia.org/wiki/ablative
49 https://en.wikipedia.org/wiki/able_seaman
50 https://en.wikipedia.org/wiki/ableism
51 https://en.wikipedia.org/wiki/abm
52 https://en.wikipedia.org/wiki/abnormalcy
53 https://en.wikipedia.org/wiki/aboriginal
54 https://en.wikipedia.org/wiki/abolition
55 https://en.wikipedia.org/wiki/abolitionist
56 https://en.wikipedia.org/wiki/abomasum
57 https://en.wikipedia.org/wiki/abominable_snowman
58 https://en.wikipedia.org/wiki/abominator
59 https://en.wikipedia.org/wiki/abor
60 https://en.wikipedia.org/wiki/abortifacient
61 https://en.wikipedia.org/wiki/abortion_pill
62 https://en.wikipedia.org/wiki/abraham
63 https://en.wikipedia.org/wiki/abraham's_bosom
64 https://en.wikipedia.org/wiki/abraham_stoker
65 https://en.wikipedia.org/wiki/abramis_brama
66 https://en.wikipedia.org/wiki/abrasive
67 https://en.wikipedia.org/wiki/abridgement
68 https://en.wikipedia.org/wiki/abrocoma
69 https://en.wikipedia.org/wiki/abrocome
70 https://en.wikipedia.org/wiki/abrogation
71 https://en.wikipedia.org/wiki/abronia
72 https://en.wikipedia.org/wiki/abronia_latifolia
73 https://en.wikipedia.org/wiki/abronia_maritima
74 https://en.wikipedia.org/wiki/abronia_villosa
75 https://en.wikipedia.org/wiki/abruption
76 https://en.wikipedia.org/wiki/abruptly-pinnate_leaf
77 https://en.wikipedia.org/wiki/abruzzi
78 https://en.wikipedia.org/wiki/absence_without_leave
79 https://en.wikipedia.org/wiki/absinthe_oil
80 https://en.wikipedia.org/wiki/absolute_pitch
81 https://en.wikipedia.org/wiki/absolute_viscosity
82 https://en.wikipedia.org/wiki/absorbent_material
83 https://en.wikipedia.org/wiki/absorptance
84 https://en.wikipedia.org/wiki/absorptivity

https://en.wikipedia.org/wiki/abstinent
https://en.wikipedia.org/wiki/abstractionist
https://en.wikipedia.org/wiki/abstract_thought
https://en.wikipedia.org/wiki/abstracter
https://en.wikipedia.org/wiki/absurdity
https://en.wikipedia.org/wiki/abuja
https://en.wikipedia.org/wiki/abelmosk
https://en.wikipedia.org/wiki/aberrance
https://en.wikipedia.org/wiki/abetment
https://en.wikipedia.org/wiki/abettor
https://en.wikipedia.org/wiki/abies
https://en.wikipedia.org/wiki/abies_alba
https://en.wikipedia.org/wiki/amabilis_fir
https://en.wikipedia.org/wiki/abies_balsamea
https://en.wikipedia.org/wiki/abies_venusta
https://en.wikipedia.org/wiki/abies_fraseri

ANEXO B Nodos Target – Sinónimos

1 https://en.wikipedia.org/wiki/leonardo_da_vinci
2 https://en.wikipedia.org/wiki/aaa
3 https://en.wikipedia.org/wiki/aken
4 https://en.wikipedia.org/wiki/ahvenanmaa
5 https://en.wikipedia.org/wiki/alborg
6 https://en.wikipedia.org/wiki/aalost
7 https://en.wikipedia.org/wiki/aare_river
8 https://en.wikipedia.org/wiki/orycteropus_afer
9 https://en.wikipedia.org/wiki/proteles_cristata
10 https://en.wikipedia.org/wiki/aaron's_rod
11 https://en.wikipedia.org/wiki/aaron_copland
12 https://en.wikipedia.org/wiki/ear-shell
13 https://en.wikipedia.org/wiki/abamp
14 https://en.wikipedia.org/wiki/abandoned_infant
15 https://en.wikipedia.org/wiki/bashfulness
16 https://en.wikipedia.org/wiki/abatis
17 https://en.wikipedia.org/wiki/slaughterhouse
18 https://en.wikipedia.org/wiki/prioress
19 https://en.wikipedia.org/wiki/archimandrite
20 https://en.wikipedia.org/wiki/abbreviator
21 https://en.wikipedia.org/wiki/abdominal_muscle
22 https://en.wikipedia.org/wiki/solar_plexus
23 https://en.wikipedia.org/wiki/paracentesis
24 https://en.wikipedia.org/wiki/tummy_tuck
25 https://en.wikipedia.org/wiki/paunchiness
26 https://en.wikipedia.org/wiki/abducens
27 https://en.wikipedia.org/wiki/lateral_rectus
28 https://en.wikipedia.org/wiki/white_aspen
29 https://en.wikipedia.org/wiki/commutative_group
30 https://en.wikipedia.org/wiki/genus_abelmoschus
31 https://en.wikipedia.org/wiki/abulia
32 https://en.wikipedia.org/wiki/genus_abutilon
33 https://en.wikipedia.org/wiki/butter-print
34 https://en.wikipedia.org/wiki/abysm
35 https://en.wikipedia.org/wiki/abyssinian_cat
36 https://en.wikipedia.org/wiki/ethiopian_banana
37 https://en.wikipedia.org/wiki/black_wattle
38 https://en.wikipedia.org/wiki/gidgee
39 https://en.wikipedia.org/wiki/silver_wattle
40 https://en.wikipedia.org/wiki/golden_wattle
41 https://en.wikipedia.org/wiki/lowland_fir

42 https://en.wikipedia.org/wiki/california_white_fir
43 https://en.wikipedia.org/wiki/alpine_fir
44 https://en.wikipedia.org/wiki/jebel_musa
45 https://en.wikipedia.org/wiki/autogenesis
46 https://en.wikipedia.org/wiki/recantation
47 https://en.wikipedia.org/wiki/abkhasian
48 https://en.wikipedia.org/wiki/ablative_case
49 https://en.wikipedia.org/wiki/able-bodied_seaman
50 https://en.wikipedia.org/wiki/able-bodism
51 https://en.wikipedia.org/wiki/antiballistic_missile
52 https://en.wikipedia.org/wiki/abnormal_condition
53 https://en.wikipedia.org/wiki/abo
54 https://en.wikipedia.org/wiki/abolishment
55 https://en.wikipedia.org/wiki/emancipationist
56 https://en.wikipedia.org/wiki/fourth_stomach
57 https://en.wikipedia.org/wiki/yeti
58 https://en.wikipedia.org/wiki/loather
59 https://en.wikipedia.org/wiki/miri
60 https://en.wikipedia.org/wiki/abortion-inducing_drug
61 https://en.wikipedia.org/wiki/mifepristone
62 https://en.wikipedia.org/wiki/ibrahim
63 https://en.wikipedia.org/wiki/bosom_of_abraham
64 https://en.wikipedia.org/wiki/bram_stoker
65 https://en.wikipedia.org/wiki/european_bream
66 https://en.wikipedia.org/wiki/abrasive_material
67 https://en.wikipedia.org/wiki/abridgment
68 https://en.wikipedia.org/wiki/genus_abrocoma
69 https://en.wikipedia.org/wiki/chinchilla_rat
70 https://en.wikipedia.org/wiki/repeal
71 https://en.wikipedia.org/wiki/genus_abronia
72 https://en.wikipedia.org/wiki/yellow_sand_verbena
73 https://en.wikipedia.org/wiki/beach_pancake
74 https://en.wikipedia.org/wiki/desert_sand_verbena
75 https://en.wikipedia.org/wiki/breaking_off
76 https://en.wikipedia.org/wiki/even-pinnate_leaf
77 https://en.wikipedia.org/wiki/abruzzi_e_molise
78 https://en.wikipedia.org/wiki/unauthorized_absence
79 https://en.wikipedia.org/wiki/wormwood_oil
80 https://en.wikipedia.org/wiki/perfect_pitch
81 https://en.wikipedia.org/wiki/coefficient_of_viscosity
82 https://en.wikipedia.org/wiki/absorbent
83 https://en.wikipedia.org/wiki/coefficient_of_absorption
84 https://en.wikipedia.org/wiki/absorption_factor

https://en.wikipedia.org/wiki/nondrinker
https://en.wikipedia.org/wiki/abstract_artist
https://en.wikipedia.org/wiki/reasoning
https://en.wikipedia.org/wiki/abstractor
https://en.wikipedia.org/wiki/ridiculousness
https://en.wikipedia.org/wiki/capital_of_nigeria
https://en.wikipedia.org/wiki/abelmoschus_moschatus
https://en.wikipedia.org/wiki/aberrancy
https://en.wikipedia.org/wiki/abettal
https://en.wikipedia.org/wiki/abetter
https://en.wikipedia.org/wiki/genus_abies
https://en.wikipedia.org/wiki/european_silver_fir
https://en.wikipedia.org/wiki/red_silver_fir
https://en.wikipedia.org/wiki/balsam_fir
https://en.wikipedia.org/wiki/santa_lucia_fir
https://en.wikipedia.org/wiki/fraser_fir

ANEXO C – Implementación Similaridad WordNet – Wu y Palmer

El anexo C presenta la implementación de la estructura necesaria y los métodos del algoritmo orientado a la similaridad de grafos RDF, propuesto por la Universidad de Yunnan, utilizando la función de similaridad de Wu y Palmer. Esta versión del código admite el reconocimiento de archivos RDF y trabaja con los archivos base de WordNet.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define LINEBUF              (15*1024)
//search index prototypes
#define KEY_LEN              (1024)
#define LINE_LEN    (1024*25)
#define MAXWORDS    155327
char *bin_search(char *searchkey, FILE *fp);
static char line[LINE_LEN];
static char lineB[LINE_LEN];
static char lineC[LINE_LEN];
static char notF[3] = "NF";
long last_bin_search_offset = 0;
//search index prototypes
typedef char *tlabel;
//ESTRUCTURA INDEX
typedef struct
{
   char *word;
   char *tipo;
   int off_cont;
   int ptr_cont;
   unsigned long *offsets;
} index_ofs;
typedef index_ofs *tindex_ofs;

//ESTRUCTURA DATA
typedef struct
{
   unsigned long   data_off;
   char           *syn_tipo;
   unsigned long   syn_cont;
   unsigned long   padre_ofs;
} syn_data;
```

```
typedef syn_data *tsyn_data;
//Lista dinámica que almacena los "sinónimos" de la estructura synsets
typedef struct synwords
{
   char *syn_word;
   struct synwords *sig;
} *tsynwords;
typedef struct
{
   unsigned long syn_off;
   unsigned long syn_cont;
   tsynwords sinonimos;
} synsets;
typedef synsets *tsynsets;
double look_data(unsigned long offset, unsigned long offsetB, FILE *df);
void insert_sw(tsynsets syn_aux, char *pt);
unsigned long get_padre(unsigned long offset, FILE *f);
double get_synCont(unsigned long offset, FILE *f);
double PadreComunFor(unsigned long offsetA, unsigned long offsetB, FILE *f);
double sim_lin(tlabel wordA, tlabel wordB);

double sim_lin(tlabel wordA, tlabel wordB)
{
   FILE *f;
   f = fopen("index.noun", "r");
   int i,j;
   char word;
   char *ptrtok, *ptB;
   if(f==NULL){
      printf("No se ha podido abrir el fichero.\nWord A: %s\nWord B: %s", wordA,
wordB);
      //exit(1);
      return -1;
   }
   // Consultas A y B
   //(A)
   strcpy(line, bin_search(wordA,f));
   printf("A: Line found on Index.pos:\n\n%s\n",line);
   if(strcmp(line,"NF") == 0)
   {
      fclose(f);
      return -1;
   }
   tindex_ofs aux = NULL;
   aux = (tindex_ofs)malloc(sizeof(index_ofs));
   /***    CORTAR CADENA    ***/
```

```
aux->word = (char *)malloc(sizeof(char));
ptrtok = strtok(line," \n");
aux->word = ptrtok;
aux->tipo = (char *)malloc(sizeof(char));
ptrtok = strtok(NULL," \n");
aux->tipo = ptrtok;
ptrtok = strtok(NULL," \n");
aux->off_cont = atoi(ptrtok);
ptrtok = strtok(NULL," \n");
aux->ptr_cont = atoi(ptrtok);
for(i=0; i<(aux->ptr_cont+2); i++)
{
   ptrtok = strtok(NULL," \n");
}
aux->offsets = (long *)malloc(aux->off_cont * (sizeof(long)));
for(i=0; i<aux->off_cont; i++)
{
   ptrtok = strtok(NULL," \n");
   aux->offsets[i] = atol(ptrtok);
}
printf("Information detail\n\n");
printf(" - WORD A - \n");
printf("Word: %s\n",aux->word);
printf("Tipo: %s\n",aux->tipo);
printf("Offsets totales: %i\n\n",aux->off_cont);
for(i=0; i<aux->off_cont; i++)
{
   printf("Offset[%i]: %.8lu \n",i+1, aux->offsets[i]);
}
```

```
// INDEX (B)
strcpy(lineB, bin_search(wordB,f));
printf("B: Line found on Index.pos:\n\n%s\n",lineB);
if(strcmp(lineB,"NF") == 0)
{
   fclose(f);
   free(aux);
   return -1;
}
tindex_ofs auxB = NULL;
auxB = (tindex_ofs)malloc(sizeof(index_ofs));
/***    CORTAR CADENA    ***/
```

```
    auxB->word = (char *)malloc(sizeof(char));
    ptB = strtok(lineB," \n");
    auxB->word = ptB;
    auxB->tipo = (char *)malloc(sizeof(char));
    ptB = strtok(NULL," \n");
    auxB->tipo = ptB;
    ptB = strtok(NULL," \n");
    auxB->off_cont = atoi(ptB);
    ptB = strtok(NULL," \n");
    auxB->ptr_cont = atoi(ptB);
    for(i=0; i<(auxB->ptr_cont+2); i++)
    {
       ptB = strtok(NULL," \n");
    }
    auxB->offsets = (long *)malloc(auxB->off_cont * (sizeof(long)));


    for(i=0; i<auxB->off_cont; i++)
    {
       ptB = strtok(NULL," \n");
       auxB->offsets[i] = atol(ptB);
    }
    printf("Information detail\n\n");
    printf(" - WORD B - \n");
    printf("Word: %s\n",auxB->word);
    printf("Tipo: %s\n",auxB->tipo);
    printf("Offsets totales: %i\n\n",auxB->off_cont);
    for(i=0; i<auxB->off_cont; i++)
    {
       printf("Offset[%i]: %.8lu \n",i+1, auxB->offsets[i]);
    }
    // FIN INDEX B

/*** PROBAR CON 1 SOLO FILE PER SIM_LIN ***/

    FILE *df;
    df = fopen("data.noun","r");
    if(df==NULL){
       printf("No se ha podido abrir el fichero Look Data.\n");
       //exit(1);
       return -1;
    }
    // FUNCION CALCULAR COMBINACIONES INDEX A,B
     double simL = 0, simAux = 0;
       for(i=0; i<aux->off_cont; i++)
       {
```

```
            for(j=0; j<auxB->off_cont; j++)
            {
                  simAux = look_data(aux->offsets[i], auxB->offsets[j], df);
                  printf("\nsimAux: %f\n\n", simAux);
                  printf("simL es: %f\n\n", simL);
                  if(simAux > simL)
                     simL = simAux;
            }
         }
      printf("La similitud entre A y B es de :%f\n", simL);
free(aux);
free(auxB);
fclose(df);
   return simL;
}
int get_syn_type(tlabel word)
{
   int tipo = 0;
   //0 = no se encuentra en la base de datos
   //1 = tipo noun
   //2 = tipo verb
   //3 = tipo adj
   //4 = tipo adv
   FILE *n;
   n = fopen("index.noun", "r");

   printf("Abre N\n");
   if(n == NULL)
   {
      printf("No se ha podido abrir el fichero Index.Noun.\n");
      exit(1);
   }
   printf("Antes de binsearch N\n");


   if(bin_search(word, n) != NULL)
   {
      strcpy(lineC, bin_search(word, n));
      printf("N: Line found on Index.Noun:\n\n%s\n",lineC);

      tipo = 1;
      fclose(n);
      return tipo;
   }
   else{
      FILE *v;
```

```
   v = fopen("index.verb", "r");
   if(v == NULL)
{
   printf("No se ha podido abrir el fichero Index.Verb.\n");
   exit(1);
}
if(bin_search(word, v) != NULL)
{
   strcpy(lineC, bin_search(word, v));
   printf("V: Line found on Index.Verb:\n\n%s\n",lineC);
   tipo = 2;
   fclose(v);
   return tipo;
}else{
   FILE *a;
   a = fopen("index.adj", "r");
   if(a == NULL)
{
   printf("No se ha podido abrir el fichero Index.Adj.\n");
   exit(1);
}
if(bin_search(word, a) != NULL)
{
   strcpy(lineC, bin_search(word, a));
   printf("V: Line found on Index.Adj:\n\n%s\n",lineC);
   tipo = 3;
   fclose(a);
   return tipo;
}else{
   FILE *b;
   b = fopen("index.adv", "r");
   if(b == NULL)
{
   printf("No se ha podido abrir el fichero Index.Adv.\n");
   exit(1);
}
if(bin_search(word, b) != NULL)
{
   strcpy(lineC, bin_search(word, b));
   printf("V: Line found on Index.Adv:\n\n%s\n",lineC);
   tipo = 4;
   fclose(b);
   return tipo;
}else return 0; //no se encontró la palabra en la base de datos.
}
}
```

```
    }
  }

  double PadreComunFor(unsigned long offsetA, unsigned long offsetB, FILE *f)
  {
    unsigned long offA, offB;
    int found = 0, sinpadre = 0;

    offB = offsetB;

    if(offsetA == offsetB)
      return get_synCont(offsetA, f);
    else for(offA = offsetA; offA; offA = get_padre(offA, f))
    {
      for(offB = offsetB; offB; offB = get_padre(offB, f))
      {
        if(offA == offB)
        {
          return get_synCont(offB, f);
        }
      }
    }
    //si no se encontró retorna 0;
    return 0;
  }
  double distRoot(unsigned long offset, FILE *f)
  {
    unsigned long off;
    double c;

    off = get_padre(offset, f);

    for(c = 0; off; off = get_padre(off, f))
    {
      c++;
    }
    return c;
  }
  double Sim_WuP(unsigned long offsetA, unsigned long offsetB, FILE *f)
  {
    unsigned long offA, offB;
    double cA, cB, c;
    double simWP;
    if(offsetA == offsetB)      //pertenecen al mismo synset, similitud máxima (1 máximo
valor);
      return 1;
```

```
   else for(offA = offsetA, cA = 0; offA; offA = get_padre(offA, f), cA++)
   {
      for(offB = offsetB, cB = 0; offB; offB = get_padre(offB, f), cB++)
      {
         if(offA == offB)
         {
            printf("wordA = %.8lu - wordB = %.8lu\n", offA, offB);
            printf("cA: %f - cB: %f\n", cA, cB);
            printf("\nDIST ROOT\n\n");
            c = distRoot(offA, f);
            printf("distRoot: %f\n", c);
            simWP = (2*c) / ( cA + cB + ( 2*c ) );
            return simWP;
         }
      }
   }
   //si no se encontró retorna cero
   return 0;
}
double look_data(unsigned long offset, unsigned long offsetB, FILE *df)
{
   /*** BUSCAR EN DATA.POS ***/
/***
   FILE *df;
   df = fopen("data.noun","r");
   if(df==NULL){
      printf("No se ha podido abrir el fichero Look Data.\n");
      exit(1);
   }
***/
   printf("\nLOOK DATA\n\nOffset: %.8lu\n", offset);
   static char linea[LINEBUF], lineaB[LINEBUF];
   char *pttk, *pttkB;
   fseek(df, offset, 0);
   fgets(linea, LINEBUF, df);
   printf("\nLine found on Data.pos: \n\n%s\n", linea);

   /*** INICIALIZAR ESTRUCTURA ***/
   tsyn_data dat_aux = NULL;
   dat_aux = (tsyn_data)malloc(sizeof(syn_data));
   /*** CORTAR CADENA ***/
   //1-data offset
   pttk = strtok(linea," \n");
   dat_aux->data_off = atol(pttk);
   //2-lex tipo
   pttk = strtok(NULL, " \n");
```

```
//3-syn tipo
dat_aux->syn_tipo = (char *)malloc(sizeof(char));
pttk = strtok(NULL, " \n");
dat_aux->syn_tipo = pttk;
//4-syn cont
pttk = strtok(NULL, " \n");
dat_aux->syn_cont = strtoul(pttk, NULL, 16);
//nos saltamos los siguientes valores hasta encontrar el padre: "@"
pttk = strtok(NULL, "@");
pttk = strtok(NULL, " \n");
dat_aux->padre_ofs = atol(pttk);
printf("Information on detail\n\n");
printf("Data off: %.8lu\n", dat_aux->data_off);
printf("Syn tipo: %s\n", dat_aux->syn_tipo);
printf("Syn cont: %u\n", dat_aux->syn_cont);
if(dat_aux->padre_ofs)
printf("Padre Offset: %.8lu\n", dat_aux->padre_ofs);
else printf("El Offset %.8lu no tiene padre definido en la base de datos\n");


/*** DATA (B) ***/
fseek(df, offsetB, 0);
fgets(lineaB, LINEBUF, df);
printf("\nB: Line found on Data.pos: \n\n%s\n", lineaB);
/*** INICIALIZAR ESTRUCTURA ***/
tsyn_data dat_B = NULL;
dat_B = (tsyn_data)malloc(sizeof(syn_data));
/*** CORTAR CADENA ***/
//1-data offset
pttkB = strtok(lineaB," \n");
dat_B->data_off = atol(pttkB);
//2-lex tipo
pttkB = strtok(NULL, " \n");
//3-syn tipo
dat_B->syn_tipo = (char *)malloc(sizeof(char));
pttkB = strtok(NULL, " \n");
dat_B->syn_tipo = pttkB;
//4-syn cont
pttkB = strtok(NULL, " \n");
dat_B->syn_cont = strtoul(pttkB, NULL, 16);
//nos saltamos los siguientes valores hasta encontrar el padre: "@"
pttkB = strtok(NULL, "@");
pttkB = strtok(NULL, " \n");
dat_B->padre_ofs = atol(pttkB);
printf("Information on detail\n\n");
printf(" - WORD B - \n");
```

```
    printf("Data off: %.8lu\n", dat_B->data_off);
    printf("Syn tipo: %s\n", dat_B->syn_tipo);
    printf("Syn cont: %u\n", dat_B->syn_cont);
    if(dat_B->padre_ofs)
    printf("Padre Offset: %.8lu\n", dat_B->padre_ofs);
    else printf("El Offset %.8lu no tiene padre definido en la base de datos\n");

    /*** FUNCION SIMILITUD ***/
    //AQUI TENDRIAMOS QUE LLAMAR A LA FUNCION DE SIMILITUD DE WUP
    double valorC = 0;
    //valorC = PadreComunFor(dat_aux->data_off, dat_B->data_off, df);
    valorC = Sim_WuP(dat_aux->data_off, dat_B->data_off, df);
    printf("valor C: %lf\n", valorC);
//  fclose(df);
    free(dat_aux);
    free(dat_B);
    return valorC;
}
unsigned long get_padre(unsigned long offset, FILE *f)
{
    static char linea[LINEBUF];
    char *pttk;
    unsigned long valor;
    fseek(f, offset, 0);
    fgets(linea, LINEBUF, f);
    printf("\nLinea: %s\n", linea);
    pttk = strtok(linea, "@");
    pttk = strtok(NULL, " \n");
    valor = atol(pttk);
    return valor;
}

double get_synCont(unsigned long offset, FILE *f)
{
    static char linea[LINEBUF];
    char *pttk;
    double valor;
    fseek(f, offset, 0);
    fgets(linea, LINEBUF, f);
    printf("\n\nLinea: %s\n", linea);
    pttk = strtok(linea, " \n");
    pttk = strtok(NULL, " \n");
    pttk = strtok(NULL, " \n");
    pttk = strtok(NULL, " \n");
     valor = (double)strtoul(pttk, NULL, 16);
     printf("valor: %lf\n", valor);
```

```
   return valor;
}

void insert_sw(tsynsets syn_aux, char *pt)
{
   //INICIALIZAR LISTA DINAMICA SYNWORDS
   tsynwords syn_words;
   syn_words = (tsynwords)malloc(sizeof(struct synwords));

   if(syn_words == NULL)
      printf("Error Memoria Insuficiente");

   //syn_words->sig = NULL;
   //syn_words->syn_word = '\0';

   // FIN INICIALIZACION LISTA DINAMICA
   syn_words->syn_word = (char *)malloc(sizeof(char));
   syn_words->sig = syn_aux->sinonimos;
   strcpy(syn_words->syn_word , pt);
   syn_aux->sinonimos = syn_words;
   printf("insert_sw: syn_words->syn_word: %s\n", syn_words->syn_word);
}

char *bin_search(char *searchkey, FILE *fp)
{
   int c;
   long top, mid, bot, diff;
   char *linep, key[KEY_LEN];
   int length;
   diff=666;
   linep = line;
   line[0] = '\0';
   fseek(fp, 0L, 2);
   top = 0;
   bot = ftell(fp);
   mid = (bot - top) / 2;
   do {
 fseek(fp, mid - 1, 0);
 if(mid != 1)
    while((c = getc(fp)) != '\n' && c != EOF);
      last_bin_search_offset = ftell( fp );
 fgets(linep, LINE_LEN, fp);
 length = (int)(strchr(linep, ' ') - linep);
 strncpy(key, linep, length);
 key[length] = '\0';
 if(strcmp(key, searchkey) < 0) {
```

```
    top = mid;
    diff = (bot - top) / 2;
    mid = top + diff;
 }
 if(strcmp(key, searchkey) > 0) {
    bot = mid;
    diff = (bot - top) / 2;
    mid = top + diff;
 }
   } while((strcmp(key, searchkey)) && (diff != 0));
   if(!strcmp(key, searchkey))
 return(line);
   else
 return(notF);
}
```

ANEXO D – Código Implementación algoritmo RDF y LPS

El anexo D incluye la implementación de las estructuras necesarias y las funcionalidades del preprocesamiento a través del algoritmo LPS aplicado a los grafos Query y Target, además de el algoritmo orientado a la similaridad de grafos RDF, utilizando la fórmula de Wu y Palmer y los archivos base de WordNet.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include "search_wn_SIM_WuP.h"
/*** VALORES INICIALES ***/
// Triple Sim variables
#define TRIPS 0.4
#define TRIPP 0.2
#define TRIPO 0.4
// Sim structural
#define ASTR 0.5
#define BSTR 0.5
// sim struct
#define A_STATE 0.5
#define B_STATE 0.5
/*** Estructuras ***/
typedef char *tlabel;       // puntero para las etiquetas de los nodos
typedef struct uri          // Estructura para almacenar las partes de las Uri
{
      tlabel sch;
      tlabel host;
      tlabel path;
      tlabel frag;
}*turi;



// LISTA PARA ALMACENAR LAS TRIPLETAS DEL GRAFO. //Sirve para "Arcos", "Salidas-Llegadas" y listas LPS
typedef struct triple
{
   // Uso Arcos
   struct nodo *sujeto;    // Puntero a nodo "origen"
   tlabel predicado;       // Etiqueta del predicado
   struct nodo *objeto;    // Puntero a nodo "destino"
```

```
    // Uso Salida
    int costo;            // Costo ascii de la tripleta salida (suj+pred+obj)

    // Uso LPS
    int longitud;           // Nº de tripletas en la lista LPS
    int costoTotal;       // Costo acumulado de las tripletas que conforman el LPS
    int bk;              // Bandera "backwards" (Inicia en 0)
    struct triple *inicio;  // Ptr a la tripleta inicio del LPS
    struct triple *fin;     // Ptr a la tripleta final del LPS ( final-sig => NULL )

    struct triple *sig;
  }*ttriple;




  // LISTA PARA ALMACENAR LOS NODOS DEL GRAFO.
  typedef struct nodo
  {
    int nodo;            // N° del nodo (Identificador)
    int tipo;            // 0=inicia, 1=URI, 2=variable
    tlabel label;         // Etiqueta del nodo
    ttriple llegada;      // Lista, almacena las tripletas que "llegan" al nodo actual
    ttriple salida;       // Lista, almacena las tripletas que "salen" desde el nodo actual

    // Uso LPS
    int cost;            // Costo ascii del label del nodo
    int marca;           // Marca para visitar el LPS    (Futuro Uso)
    int marcaC;          // Marca los Ciclos.          (Futuro Uso)
    int cant;            // Cantidad de elementos en la tripleta salida ("Aridad" del nodo
Objeto de la tripleta)

    struct nodo *sig;
  }*tnodo;
  // LISTA PARA ALMACENAR EL O LOS GRAFOS. (Pensado para más adelante
cuando se necesite ingresar varios grafos en la base de datos)

  typedef struct grafo
  {
    int n;              // número de nodos que tiene el grafo
    tnodo nodo;          // lista, almacena los nodos del grafo.
    struct grafo *sig;
```

```
}*tgrafo;

static char uriLine[1024];

/*** FUNCIONES ***/
tnodo crear(void);
tnodo inserta_nodo(tlabel label, tnodo g);
void imprime_nodo(tnodo g);
int existe_nodo(tlabel label, tnodo g);
tnodo get_nodo(tlabel label, tnodo g);
void inserta_tripleta(tlabel sujeto, tlabel predicado, tlabel objeto, tnodo g);
void datos_nodo(tnodo g);
void datos_triple(ttriple t);
void imprime_arcos(tnodo g);
void mostrar_grafo(tnodo g);
int get_label_type(tlabel label);
int Levenshtein(tlabel s1, int t1, tlabel s2, int t2);
double sim_string(tlabel s1, tlabel s2);
int Minimo(int a, int b);
void cut_uri(tlabel uriX, turi u);
double sim_uri_uri(tlabel uriA, tlabel uriB);
double sim_uri_wrd(tlabel uriA, tlabel wordB);
double sim_label(tlabel labelA, tlabel labelB);
/*** triple sim functions ***/
void load_file(FILE *f, tnodo g);
/*** Sim RDF Functions ***/
double sim_rdf(tnodo gq, tnodo gt);
tnodo first_nodo(tnodo g);
ttriple first_salida(tnodo n, tnodo g);
ttriple first_llegada(tnodo n, tnodo g);
ttriple next_salida(tnodo n, ttriple t, tnodo g);
ttriple next_llegada(tnodo n, ttriple t, tnodo g);
tnodo next_nodo(tnodo n, tnodo g);
tnodo nodo_suj(ttriple t, tnodo g);
tnodo nodo_obj(ttriple t, tnodo g);
tlabel label_triple(ttriple t, tnodo g);
double sim_triple(ttriple query, tnodo gq, ttriple target, tnodo gt );
double sim_upper(ttriple query, tnodo gq, ttriple target, tnodo gt);
double sim_lower(ttriple query, tnodo gq, ttriple target, tnodo gt);
double sim_struct(ttriple query, tnodo gq, ttriple target, tnodo gt);
double sim_state(ttriple query, tnodo gq, ttriple target, tnodo gt);
/*** LPS Functions ***/
ttriple crear_triple();
ttriple LPS(tnodo g);
ttriple getLps(ttriple t);
double sim_lps(ttriple qt, ttriple tt);
```

```
void mostrar_LPS(ttriple t);
void load_fileB(FILE *f, tnodo g);
int main()
{
   /*** Carga Archivos ***/
   // Grafo Query ---------
   tnodo gq;
   gq = crear();
   FILE *q;
   q = fopen("query.txt", "r");
   if(q==NULL){
      printf("No se ha podido abrir el fichero.\n");
      exit(1);
   }
   load_file(q, gq);
   fclose(q);

   //Mostrar Grafo Query
   printf("\n\n NODOS GRAFO QUERY \n\n");
   imprime_nodo(gq);
   printf("\n\n - GRAFO QUERY - \n\n");
   mostrar_grafo(gq);
   // Grafo Target ---------
   tnodo gt;
   gt = crear();
   FILE *t;
   t = fopen("target.txt", "r");
   if(t==NULL){
      printf("No se ha podido abrir el fichero.\n");
      exit(1);
   }
   load_fileB(t, gt);
   fclose(t);

   //Mostrar Grafo Target
   printf("\n\n NODOS GRAFO TARGET \n\n");
   imprime_nodo(gt);
   printf("\n\n - GRAFO TARGET - \n\n");
   mostrar_grafo(gt);



// 1 - RDF / 0 - LPS
```

```
   if(1)
   {
      printf("SIM RDF\n\n");
      double sr;
      clock_t start = clock();
      sr = sim_rdf(gq, gt);
      printf("\n\nTIME: %f\n", ((double)clock() - start) / CLOCKS_PER_SEC);
      printf("\nRDF SIMILARITY: %f\n", sr);
   }else{
      ttriple lps_Q, lps_G;
      lps_Q = LPS(gq);
      printf("\n\n - *** LPS QUERY *** - \n\n");
      mostrar_LPS(lps_Q);
      lps_G = LPS(gt);
      printf("\n\n - *** LPS TARGET *** - \n\n");
      mostrar_LPS(lps_G);
      double sl;
      clock_t start = clock();
      sl = sim_lps(lps_Q, lps_G);
      printf("\n\nTIME: %f\n\n", ((double)clock() - start) / CLOCKS_PER_SEC);
      printf("\nLPS SIMILARITY: %f\n", sl);
      }
   free(gq);
   free(gt);
return 0;
}


void mostrar_LPS(ttriple t)
{
   printf("\n\n*** MOSTRAR LPS ***\n\n");

   while(t != NULL)
   {
      printf("(%s) - (%s) - (%s)\n", t->sujeto->label, t->predicado, t->objeto->label);
      printf("COSTO TOTAL: %i\n", t->costoTotal);
      t = t->sig;
   }
   printf("\n\n*** FIN LPS ***\n\n");
}

ttriple LPS(tnodo g)
{
   tnodo n;
   ttriple MaxTotal, maxi;
   ttriple aux;
```

```
        int b = 0;
        n = first_nodo(g);
        while(n != NULL)    //Recorre los nodos del grafo
        {
           if(n->llegada == NULL)  // Verificación de que la tripleta sea la raíz (que no tenga
tripletas que "apuntan" a ella)(si no es el primer nodo, busca en el grafo)
           {
              aux = n->salida;
              b = 1;
              while(aux != NULL)
              {
                 if(b)                    // Si es la primera vez que entra al while
                 {
                    maxi = getLps(aux);
                    MaxTotal = maxi;
                    b = 0;
                 }else {
                       maxi = getLps(aux);
                       }
    /***
              printf("\n*** RESULTADO LPS ***\n\n");
              printf("\n*** INICIO ***\n\n");
              printf("Sujeto: %s\n", maxi->inicio->sujeto->label);
              printf("Predicado: %s\n", maxi->inicio->predicado);
              printf("Objeto: %s\n", maxi->inicio->objeto->label);
              printf("\n*** FINAL ***\n\n");
              printf("Sujeto: %s\n", maxi->fin->sujeto->label);
              printf("Predicado: %s\n", maxi->fin->predicado);
              printf("Objeto: %s\n", maxi->fin->objeto->label);
              mostrar_LPS(maxi);
    ***/
              if(maxi->longitud > MaxTotal->longitud)
              {
                 MaxTotal = maxi;
                 maxi = NULL;
              }else if(maxi->longitud == MaxTotal->longitud)
                    {
                       if(maxi->costoTotal > MaxTotal->costoTotal)
                       {
                          MaxTotal = maxi;
                          maxi = NULL;
                       }else if(maxi->costoTotal < MaxTotal->costoTotal)       // NO PUEDE
SER IGUAL, PORQUE NO HAY TRIPLETAS REPETIDAS
                             {
                                /*** Destroy Maxi ***/
                             }
```

```
                }else if(maxi->longitud < MaxTotal->longitud)
                        {
                           /*** Destroy Maxi ***/
                        }
            aux = aux->sig;
            }
        }
        n = next_nodo(n,g);
    }
return MaxTotal;
}
ttriple getLps(ttriple t)
{
    ttriple aux = t->objeto->salida;

    if(aux == NULL)         // Si no tiene hijos;
    {
        ttriple bkw = crear_triple();   // Inicializar tripleta
        // llenar bkw con los datos de la tripleta ingresada a la función ( t )
        bkw->sujeto = t->sujeto;

        bkw->predicado = (char *)malloc((strlen(t->predicado))*sizeof(char));
        if(bkw->predicado == NULL)
          printf("Error Memoria Insuficiente");
        strcpy(bkw->predicado, t->predicado);
        bkw->objeto = t->objeto;
        bkw->costo = t->costo;
        bkw->longitud = 1;
        bkw->costoTotal = t->costo;
        bkw->bk = 1;
        bkw->inicio = t;
        bkw->fin = t;
        bkw->sig = NULL;
        return bkw;
    }else{
        ttriple MaxTotal, maxi;
        int b = 1;
        while(aux != NULL)
    {
        if(b)                   // Es la primera vez que entra al while
        {
            //ttriple maxi;
            maxi = getLps(aux);
            MaxTotal = maxi;
            b = 0;
        }
```

```
   else{                    // No es la primera vez que entra al while
      //ttriple maxi;
      maxi = getLps(aux);
   }

   if(maxi->longitud > MaxTotal->longitud)
   {
      /*** A ***/
      MaxTotal = maxi;
      maxi = NULL;
   }else
      if(maxi->longitud == MaxTotal->longitud)
      {
         if(maxi->costoTotal > MaxTotal->costoTotal)
         {
            /*** A ***/
            MaxTotal = maxi;
            maxi = NULL;
         }else
            if(maxi->costoTotal < MaxTotal->costoTotal)
            {
               /*** DESTROY maxi ***/
            }
      }else
         if(maxi->longitud < MaxTotal->longitud)
         {
            /*** DESTROY maxi ***/
         }
   aux = aux->sig;
}
/*** Agregar la tripleta ( t ) al mayor de sus hijos (LPS acumulado)***/
ttriple head;
head = crear_triple();
//llenar cabeza con los datos de la tripleta ( t ) ingresada
head->sujeto = t->sujeto;

head->predicado = (char *)malloc((strlen(t->predicado))*sizeof(char));
   if(head->predicado == NULL)
      printf("Error Memoria Insuficiente");
strcpy(head->predicado, t->predicado);
head->objeto = t->objeto;
head->costo = t->costo;
head->longitud = MaxTotal->longitud + 1;
head->costoTotal = MaxTotal->costoTotal + t->costo;
head->bk = 1;
head->inicio = head;
```

```
        head->fin = MaxTotal->fin;
        head->sig = MaxTotal;
        MaxTotal->inicio = head;
        return head;
        }
    }
    ttriple crear_triple()
    {
        ttriple aux;
        aux = (ttriple)malloc(sizeof(struct triple));
        if(aux == NULL)
            printf("Crear triple: Error al crear triple\n");
        else{
            aux->sujeto = NULL;
            aux->predicado = NULL;
            aux->objeto = NULL;
            aux->costo = 0;
            aux->longitud = 0;
            aux->costoTotal = 0;
            aux->bk = 0;
            aux->inicio = NULL;
            aux->fin = NULL;
            aux->sig = NULL;
            return aux;
        }
    }
    double sim_lps(ttriple qt, ttriple tt)
    {
        ttriple q = qt;
        ttriple t = tt;
        double simrdf = 0;
        int deg_tot = 0;

        printf("\n\n*** INSIDE SIM LPS ***\n\n");

        while(q != NULL)
        {
            printf("| %s | -> %s -> | %s |\n",q->sujeto->label, q->predicado, q->objeto->label);

                double simestate = 0, aux = 0;
                int deg_t = 0;
            while(t != NULL)
            {
                printf("        | %s | -> %s -> | %s |\n",t->sujeto->label, t->predicado, t->objeto-
>label);
```

```
            printf("\nCOMPARAR:\n(%s) - %s - (%s) vs (%s) - %s - (%s)\n", q->sujeto-
>label, q->predicado, q->objeto->label, t->sujeto->label, t->predicado, t->objeto->label);
            aux = sim_state(q, NULL, t, NULL);
            if(aux > simestate)
                  {
                     simestate = aux;
                     if(t->inicio == t || t->fin == t)
                        deg_t = 1;
                     else deg_t = 2;
                     //deg_t = deg(t, NULL);
                     printf("aux>sim_state: %s-%s-%s vs %s-%s-%s\n", q->sujeto->label, q-
>predicado, q->objeto->label, t->sujeto->label, t->predicado, t->objeto->label);
                  }
                  printf("Sim_rdf: Sim_state: %f - deg_t: %i\n", simestate, deg_t);
            t = t->sig;
         }
         //guardar el resultado para la primera tripleta, antes de llamar a la siguiente.
            int deg_aux = 0;
            deg_aux = deg_t;
            if(deg_aux == 0)
               deg_aux = 1;
            simrdf = simrdf + (simestate * deg_aux);
            deg_tot = deg_tot + deg_aux;
            q = q->sig;
            printf("SIM RDF === %f\n", simrdf);
            printf("Deg_t === %i\n", deg_t);
            printf("Deg_tot === %i\n", deg_tot);
      }
   //return simrdf / deg_tot;      // fórmula original

   return simrdf;      // modificacion para LPS
   }
   double sim_rdf(tnodo gq, tnodo gt)
   {
      tnodo q, t;
      ttriple sq, st;
      double simrdf = 0;
      int deg_tot = 0;

      q = first_nodo(gq);
      while(q != NULL)
      {
         printf("\nQUERY: Nodo N %i - ( %s )\n", q->nodo, q->label);
         sq = first_salida(q, gq);
         while(sq != NULL)
```

```
        {
            printf("| %s | -> %s -> | %s |\n",sq->sujeto->label, label_triple(sq,q), sq->objeto-
>label);
            double simestate = 0, aux = 0;
            int deg_t = 0;
            /*** recorrer el grafo target ***/
            t = first_nodo(gt);
            while(t != NULL)
            {
               printf("\n    TARGET: Nodo N %i - ( %s )\n", t->nodo, t->label);
               st = first_salida(t, gt);
               while(st != NULL)
               {
                  printf("       | %s | -> %s -> | %s |\n",st->sujeto->label, label_triple(st,t), st-
>objeto->label);
                  printf("\nCOMPARAR:\n(%s) - (%s) vs (%s) - (%s)\n", sq->sujeto->label,
sq->objeto->label, st->sujeto->label, st->objeto->label);
                  aux = sim_state(sq, gq, st, gt);
                  if(aux > simestate)
                  {
                     simestate = aux;
                     deg_t = deg(st, gt);
                     printf("aux>sim_state: %s-%s-%s vs %s-%s-%s\n", sq->sujeto->label, sq-
>predicado, sq->objeto->label, st->sujeto->label, st->predicado, st->objeto->label);
                  }
                  printf("Sim_rdf: Sim_state: %f - deg_t: %i\n", simestate, deg_t);

                  st = next_salida(t, st, gt);
               }
               t = next_nodo(t, gt);
            }
            //guardar el resultado para la primera tripleta, antes de llamar a la siguiente.
            int deg_aux = 0;
            deg_aux = deg_t;
            if(deg_aux == 0)
               deg_aux = 1;
            simrdf = simrdf + (simestate * deg_aux);
            deg_tot = deg_tot + deg_aux;
            sq = next_salida(q, sq, gq);

            printf("simrdf === %f\n", simrdf);
            printf("deg_t === %i\n", deg_t);
            printf("deg_tot === %i\n", deg_tot);
        }

    q = next_nodo(q, gq);
```

```
        }
    return simrdf / deg_tot;
    }
    double sim_state(ttriple query, tnodo gq, ttriple target, tnodo gt)
    {
       double ss = 0;
       ss = A_STATE * sim_triple(query, gq, target, gq) + B_STATE * sim_struct(query, gq,
target, gt);
       printf("Sim_state: %f\n", ss);
       return ss;
    }
    double sim_struct(ttriple query, tnodo gq, ttriple target, tnodo gt)
    {
       if(deg(query, gq) == 0 && deg(target, gt) == 0)
       {
        printf("Sim_struct: 1\n[deg(query) y deg(target) son cero]\n");
        return 1;
       }
       else
       {
          double ss = 0;
       ss = ASTR * sim_upper(query, gq, target, gt) + BSTR * sim_lower(query, gq, target,
gt);
       printf("Sim_struct: %f\n", ss);
       return ss;
       }
    }
    double sim_upper(ttriple query, tnodo gq, ttriple target, tnodo gt)
    {
      ttriple Qlleg, Tlleg;
      Qlleg = first_llegada(query->sujeto, gq);
      Tlleg = first_llegada(target->sujeto, gt);
      if(Qlleg == NULL && Tlleg == NULL)
        return 1;
      int k = 0, b = 0;
      double maxUp = 0, simUpper = 0, simUp = 0, aux = 0;
      //recorrer lista llegada del sujeto de la tripleta query
      Qlleg = first_llegada(query->sujeto, gq);
      while(Qlleg != NULL)
      {
          // recorrer lista llegada del sujeto de la tripleta target

          Tlleg = first_llegada(target->sujeto, gt);
          while(Tlleg != NULL)
          {
             aux = sim_triple(Qlleg, gq, Tlleg, gt);
```

```
            if(aux > simUp)
                simUp = aux;

            printf("simUp: %f\n", simUp);
           Tlleg = next_llegada(target->sujeto, Tlleg, gt);
        }
        maxUp = maxUp + simUp;
        k++;
        b = 1;
        Qlleg = next_llegada(query->sujeto, Qlleg, gq);
  }
  if(b)
    return simUpper = maxUp / k;
    else return 0;
}

double sim_lower(ttriple query, tnodo gq, ttriple target, tnodo gt)
{
  ttriple Qsal, Tsal;
  Qsal = first_salida(query->objeto, gq);
  Tsal = first_salida(target->objeto, gt);

  if(Qsal == NULL && Tsal == NULL)
    return 1;
  int k = 0, b = 0;
  double maxUp = 0, simLower = 0, simLw = 0, aux = 0;
```

```
//recorrer lista salida del objeto de la tripleta query
      Qsal = first_salida(query->objeto, gq);
      while(Qsal != NULL)
      {
          // recorrer lista salida del objeto de la tripleta target

          Tsal = first_salida(target->objeto, gt);
          while(Tsal != NULL)
          {
             aux = sim_triple(Qsal, gq, Tsal, gt);

             if(aux > simLw)
                simLw = aux;

             printf("simLw: %f\n", simLw);

            Tsal = next_salida(target->objeto, Tsal, gt);
          }

          maxUp = maxUp + simLw;
          k++;
          b = 1;
          Qsal = next_salida(query->objeto, Qsal, gq);
      }

      if(b)
        return simLower = maxUp / k;
        else return 0;
    }

    int indeg(ttriple t, tnodo gt)
    {
       ttriple Tlleg;
       int in = 0;

       Tlleg = first_llegada(t->sujeto, gt);
       while(Tlleg != NULL)
       {
          in++;

          Tlleg = next_llegada(t->sujeto, Tlleg, gt);
       }
       printf("Indeg: %i\n", in);
       return in;
    }
```

```
int outdeg(ttriple t, tnodo gt)
{
   ttriple Tsal;
   int out = 0;

   Tsal = first_salida(t->objeto, gt);
   while(Tsal != NULL)
   {
      out++;

      Tsal = next_salida(t->objeto, Tsal, gt);
   }
   printf("Outdeg: %i\n", out);
   return out;
}

int deg(ttriple t, tnodo gt)
{
   int dg = 0;
   dg = indeg(t, gt) + outdeg(t, gt);
   printf("DEG: %i\n", dg);
   return dg;
}
double sim_triple(ttriple query, tnodo gq, ttriple target, tnodo gt )
{
   double Strip, Ptrip, Otrip, SimT;

   Strip = TRIPS*sim_label(query->sujeto->label, target->sujeto->label);

   Ptrip = TRIPP*sim_label(query->predicado, target->predicado);

   Otrip = TRIPO*sim_label(query->objeto->label, target->objeto->label);

   SimT = Strip + Ptrip + Otrip;

   printf("Sim_triple: %f\n", SimT);

   return SimT;
}

void load_file(FILE *f, tnodo g)
{
   char A[100],B[100],C[100];
   tlabel x,y,z;
   x = A;
   y = B;
```

```
    z = C;

    while(!feof(f)){
        fscanf(f, "%s", A);
        fscanf(f, "%s", B);
        fscanf(f, "%s", C);


        printf("- Tripleta a Ingresar - \n\n");
        printf(" A es igual a: %s\n", x);
        printf(" B es igual a: %s\n", y);
        printf(" C es igual a: %s\n\n", z);
        inserta_tripleta(x,y,z,g);
    }
}
void load_fileB(FILE *f, tnodo g)
{
    char A2[100],B2[100],C2[100];
    tlabel x2,y2,z2;
    x2 = A2;
    y2 = B2;
    z2 = C2;
    while(!feof(f)){
        fscanf(f, "%s", A2);
        fscanf(f, "%s", B2);
        fscanf(f, "%s", C2);
        printf("- Tripleta a Ingresar - \n\n");
        printf(" A es igual a: %s\n", x2);
        printf(" B es igual a: %s\n", y2);
        printf(" C es igual a: %s\n\n", z2);


        inserta_tripleta(x2,y2,z2,g);
    }
}
double sim_label(tlabel labelA, tlabel labelB)
{
    int tipoA, tipoB;
    double sLabel = 0;
    tipoA = get_label_type(labelA); // 1-uri - 2-word
    tipoB = get_label_type(labelB);
    if(tipoA == -1 || tipoB == -1)
        return 0;
    else if(tipoA == 1 && tipoB == 1)
        sLabel = sim_uri_uri(labelA, labelB);
        else if(tipoA == 1 && tipoB == 2)
```

```
            sLabel = sim_uri_wrd(labelA, labelB);
            else if(tipoA == 2 && tipoB == 1)
               sLabel = sim_uri_wrd(labelB, labelA);
               else if(tipoA == 2 && tipoB == 2)
               {
                  sLabel = sim_lin(labelA, labelB);
                  if(sLabel == -1)
                     sLabel = sim_string(labelA, labelB);
               }
               return sLabel;
}
double sim_uri_wrd(tlabel uriA, tlabel wordB)
{
   turi uriQ;

   uriQ = (turi)malloc(sizeof(struct uri));
   if(uriQ == NULL)
   {
      printf("Sim_uri_wrd: No hay espacio disponible para UriQ");
   }
   cut_uri(uriA, uriQ);
   /*** FRAGMENT ***/   /*** DEBEN ESTAR EN MINUSCULAS ***/
   double sFrag = 0;

   if(uriQ->frag == NULL)
      return 0;
   else sFrag = sim_lin(uriQ->frag, wordB);
   if(sFrag == -1)
      sFrag = sim_string(uriQ->frag, wordB);
   printf("Sim Frag: %f\n", sFrag);
   return sFrag;
}
double sim_uri_uri(tlabel uriA, tlabel uriB)
{
   turi uriQ, uriT;
   uriQ = (turi)malloc(sizeof(struct uri));
   if(uriQ == NULL)
   {
      printf("Sim_uri_uri: No hay espacio disponible para UriQ");
   }
   uriT = (turi)malloc(sizeof(struct uri));
   if(uriT == NULL)
   {
      printf("Sim_uri_uri: No hay espacio disponible para UriT");
   }
   printf("\n uriA: %s\n", uriA);
```

```
        printf("\n uriB: %s\n", uriB);


        cut_uri(uriA, uriQ);
        cut_uri(uriB, uriT);

        printf("\nAFTER CUT URI\n");

        // CALCULAR SIMILITUDES
        printf("Q - Sch: %s\n Host: %s\n Path: %s\n Frag: %s\n\n", uriQ->sch, uriQ->host,
uriQ->path, uriQ->frag);
        printf("T - Sch: %s\n Host: %s\n Path: %s\n Frag: %s\n\n", uriT->sch, uriT->host, uriT-
>path, uriT->frag);
        /*** HOST ***/
        double sHost = 0;
        sHost = sim_string(uriQ->host, uriT->host);

        printf("Sim Host: %f\n\n", sHost);
        /*** PATH ***/
        double sPath = 0;
        sPath = sim_string(uriQ->path, uriT->path);
        printf("Sim Path: %f\n", sPath);
        /*** FRAGMENT ***/   /*** DEBEN ESTAR EN MINUSCULAS ***/
        double sFrag = 0;
        if(uriQ->frag == NULL && uriT->frag == NULL)
           sFrag = 1;
        else if(uriQ->frag == NULL || uriT->frag == NULL)
           sFrag = 0;
           else sFrag = sim_lin(uriQ->frag, uriT->frag);
        if(sFrag == -1)
           sFrag = sim_string(uriQ->frag, uriT->frag);
        printf("Sim Frag: %f\n", sFrag);



        /*** SIM TOTAL URI URI ***/
        double uriTot = (sHost + sPath + sFrag)/3;
        printf("Uri Total: %f\n", uriTot);
        return uriTot;
    }
    void cut_uri(tlabel uriX, turi u)   // recibe la linea uri y la struct que almacenará las partes
    {
        // CORTAR CADENA

        char *upt = NULL;
           strcpy(uriLine, uriX);
```

```
        u->sch = (char *)malloc(100*sizeof(char));
        upt = strtok(uriLine,":");
        //u->sch = upt;
        strcpy(u->sch, upt);

        u->host = (char *)malloc(100*sizeof(char));
        upt = strtok(NULL,"/");
        //u->host = upt;
        strcpy(u->host, upt);
        /*** Si tiene Fragment (Separado por #)
        u->path = (char *)malloc(100*sizeof(char));
        upt = strtok(NULL,"#\n");
        //u->path = upt;
        strcpy(u->path, upt);
        u->frag = (char *)malloc(100*sizeof(char));
        upt = strtok(NULL,"\n");
        if(upt)
           strcpy(u->frag, upt);
        else u->frag = upt;

           ***/

        /*** Si no tiene Fragment - (Tipo wikipedia: "https://wiki.org/path/RECURSO"  ***/

        u->path = (char *)malloc(100*sizeof(char));
        upt = strtok(NULL,"/");
        //u->path = upt;
        strcpy(u->path, upt);

        u->frag = (char *)malloc(100*sizeof(char));
        upt = strtok(NULL,"\n");
        if(upt)
           strcpy(u->frag, upt);
        else u->frag = upt;

        printf("\nLinea: %s\n", uriX);
        printf(" Sch: %s\n Host: %s\n Path: %s\n Frag: %s\n", u->sch, u->host, u->path, u-
>frag);
    }
    tnodo crear(void)
    {
       tnodo aux;
       aux = (tnodo)malloc(sizeof(struct nodo));
       if(aux == NULL){
          printf("Error al Crear.");
       } else {
```

```
        aux->nodo = 0;
        aux->tipo = 0;
        aux->label = NULL;
        aux->llegada = NULL;
        aux->salida = NULL;
        aux->cost = 0;
        aux->marca = 0;
        aux->marcaC = 0;
        aux->cant = 0;
        aux->sig = NULL;
        return aux;
    }
}
tnodo inserta_nodo(tlabel label, tnodo g)
{
    tnodo aux,p;
    aux = (tnodo)malloc(sizeof(struct nodo));
    if(aux == NULL)
        printf("Error Memoria Insuficiente");
    else {
        p = g;
        while(p->sig != NULL)
            p = p->sig;
        aux->label = (char *)malloc((strlen(label))*10*sizeof(char));
        if(aux == NULL)
            printf("Error Memoria Insuficiente");
        aux->nodo = p->nodo+1;
        aux->tipo = get_label_type(label);
        strcpy(aux->label,label);
        aux->llegada = NULL;
        aux->salida = NULL;

        aux->cost = ascii(label);
        aux->marca = 0;
        aux->marcaC = 0;
        aux->cant = 0;
        aux->sig = NULL;
        p->sig = aux;
        g->nodo++;
    }
        return aux;
}
void imprime_nodo(tnodo g)
{
    tnodo p;
    p=g;
```

```
   while(p->sig != NULL)
   {
      p=p->sig;
      printf("( Nodo N:%i - %s )",p->nodo ,p->label);

      if(p->tipo==0)
         printf(" - Aún no se ha asignado tipo\n");
      else if(p->tipo==1)
         printf(" - Label tipo URI\n");
      else if(p->tipo==2)
         printf(" - Label tipo WORD\n");
      else printf("%i - ERROR - Tipo fuera de parámetros\n", p->tipo);

   }
}

int existe_nodo(tlabel label, tnodo g)
{
   tnodo p;
   p=g;
   while(p->sig != NULL)
   {
      p=p->sig;
      if(strcmp(p->label,label)==0)
         return 1;
      else;
   }
   return 0;
}
tnodo get_nodo(tlabel label, tnodo g)
{
   tnodo p;
   p=g;
   while(p->sig != NULL)
   {
      p=p->sig;
      if(strcmp(p->label,label)==0)
         return p;
      else ;
   }
   return NULL;
}
void inserta_tripleta(tlabel sujeto, tlabel predicado, tlabel objeto, tnodo g)
{
   tnodo suj, obj;
   suj = get_nodo(sujeto, g);
```

```
        obj = get_nodo(objeto, g);

        if(suj == NULL && obj == NULL)    //si alguno de los dos no ha sido ingresado... se
ingresa
        {
           suj = inserta_nodo(sujeto, g);
           obj = inserta_nodo(objeto, g);
        }else if(suj != NULL && obj == NULL)
              obj = inserta_nodo(objeto, g);
              else if(suj == NULL && obj != NULL)
                   suj = inserta_nodo(sujeto, g);
                 else {printf("\nWARNING - EL SUJETO Y EL OBJETO YA ESTÁN
INGRESADOS - REVISAR QUE EL PREDICADO ES DISTINTO\n\n");
                 }
       ttriple llegada, salida, inicio, fin ;
       salida = (ttriple)malloc(sizeof(struct triple));
       llegada = (ttriple)malloc(sizeof(struct triple));
       inicio = (ttriple)malloc(sizeof(struct triple));
       fin = (ttriple)malloc(sizeof(struct triple));
       if((salida == NULL) || (llegada == NULL) || (inicio == NULL) || (fin == NULL))
           printf("salida,llegada,inicio,fin: Error Memoria Insuficiente: \nsalida: %s - llegada:
%s - inicio: %s - fin: %s\n",salida, llegada, inicio, fin);
        else{
           salida->predicado = (char*)malloc(strlen(predicado)*sizeof(char));
           if(salida->predicado == NULL)
           printf("salida->predicado: Error Memoria Insuficiente");
           salida->sujeto = suj;
           strcpy(salida->predicado, predicado);
           salida->objeto = obj;
           salida->costo = ascii(sujeto) + ascii(predicado) + ascii(objeto);
           salida->longitud = 0;
           salida->costoTotal = 0;
           salida->bk = 0;
           salida->inicio = inicio;
           salida->fin = fin;
           salida->sig = suj->salida;
           suj->salida = salida;

           suj->cant = suj->cant + 1;      // aumenta el contador de tripletas de salida en el nodo
sujeto
           llegada->predicado = (char*)malloc(strlen(predicado)*sizeof(char));
           if(llegada->predicado == NULL)
           printf("3 Error Memoria Insuficiente");
           llegada->sujeto = suj;
           strcpy(llegada->predicado, predicado);
           llegada->objeto = obj;
```

```
        llegada->costo = 0;
        llegada->longitud = 0;
        llegada->costoTotal = 0;
        llegada->bk = 0;
        llegada->inicio = NULL;
        llegada->fin = NULL;
        llegada->sig = obj->llegada;
        obj->llegada = llegada;
    }
    printf("Despues de ingresar la tripleta (dentro de la funcion)\n\n");
    printf("suj->label: %s\n", suj->label);
    printf("suj->salida->predicado: %s\n", suj->salida->predicado);
    printf("obj->label: %s\n\n", obj->label);
    printf(" - fin tripleta - \n\n");
}
void datos_nodo(tnodo g)
{
    printf("Datos del nodo\n");
    printf("Nodo N: %i\n", g->nodo);
    printf("Label: %s\n\n", g->label);
}


void datos_triple(ttriple t)
{
    printf("Datos del arco\n");
    datos_nodo(t->sujeto);
    printf("Predicado: %s\n\n", t->predicado);
    datos_nodo(t->objeto);
}
tnodo first_nodo(tnodo g)
{
    return (g->sig);
}
ttriple first_salida(tnodo n, tnodo g)
{
    return(n->salida);
}
ttriple first_llegada(tnodo n, tnodo g)
{
    return(n->llegada);
}
ttriple next_salida(tnodo n, ttriple t, tnodo g)
{
    return(t->sig);
}
```

```
ttriple next_llegada(tnodo n, ttriple t, tnodo g)
{
   return(t->sig);
}
tnodo next_nodo(tnodo n, tnodo g)
{
   return(n->sig);
}
tnodo nodo_suj(ttriple t, tnodo g)
{
   return(t->sujeto);
}
tnodo nodo_obj(ttriple t, tnodo g)
{
   return(t->objeto);
}
tlabel label_triple(ttriple t, tnodo g)
{
   return(t->predicado);
}
void mostrar_grafo(tnodo g)
{
   tnodo n;
   ttriple s,l;
   int b;
   n=first_nodo(g);
   while(n != NULL)
   {
      s = first_salida(n,g);
      printf("\n************************\n");
      printf("   Nodo N:%i - %s   \n", n->nodo, n->label);
      printf("************************\n");
      printf(" - ARCOS DE SALIDA - \n\n");
      b=0;
      while(s!=NULL)
      {
         printf("| %s | -> %s -> | %s |\n",s->sujeto->label, label_triple(s,n), s->objeto-
>label);
         s=next_salida(n,s,g);
         b=1;
      }if(b)
      printf("El nodo N:%i ( %s ) no tiene mas arcos de salida\n\n",n->nodo, n->label);
      else printf("El nodo N:%i ( %s ) no tiene arcos de salida\n\n",n->nodo, n->label);
      printf("\n - ARCOS DE LLEGADA - \n\n");
      l = first_llegada(n,g);
      b=0;
```

```
            while(l!=NULL)
            {
               printf("| %s | -> %s -> | %s |\n", l->sujeto->label, label_triple(l,n), l->objeto-
>label);
               l=next_llegada(n,l,g);
               b=1;
            }if(b)
            printf("El nodo N:%i ( %s ) no tiene mas arcos de llegada\n\n",n->nodo, n->label);
            else printf("El nodo N:%i ( %s ) no tiene arcos de llegada\n\n",n->nodo, n->label);
            n=next_nodo(n,g);
         }
      }
      int get_label_type(tlabel label)
      {
         int i=0;
         if(label == NULL)
            return -1;
            else if(label[0]=='h' & label[1]=='t' & label[2]=='t' & label[3]=='p')
                  return 1; //ES URI
                  else return 2; //ES UNA PALABRA
      }

      double sim_string(tlabel s1, tlabel s2)    // B. (a) - String similarity between Labels
      {
         int t1, t2, lav;
         double ss;
      // Calcula tamanios strings
         t1=strlen(s1); t2=strlen(s2);
         printf("\nA: %s\nB: %s\n", s1, s2);
         printf("t1: %i\nt2: %i\n", t1, t2);
         lav = Levenshtein(s1,t1,s2,t2);
         printf("Lav: %i\n", lav);
         if ( lav == (-1))
            return -1; //ERROR
          else ss = 1 - ((double)lav / (maximo(t1,t2)));
          printf("sim_string: %f\n\n", ss);
         return ss;
      }
      int Levenshtein(tlabel s1, int t1, tlabel s2, int t2)
      {   int i,j,*m,costo,res,ancho;
      // Verifica que exista algo que comparar
         if (t1==0) return(t2);
         if (t2==0) return(t1);
         ancho=t1+1;
      // Reserva matriz con malloc                  m[i,j] = m[j*ancho+i] !!
         m=(int*)malloc(sizeof(int)*(t1+1)*(t2+1));
```

```
   if (m==NULL) return(-1); // ERROR!!
// Rellena primera fila y primera columna
   for (i=0;i<=t1;i++) m[i]=i;
   for (j=0;j<=t2;j++) m[j*ancho]=j;
// Recorremos resto de la matriz llenando pesos
   for (i=1;i<=t1;i++) for (j=1;j<=t2;j++)
      { if (s1[i-1]==s2[j-1]) costo=0; else costo=1;
        m[j*ancho+i]=Minimo(Minimo(m[j*ancho+i-1]+1,     // Eliminacion
                    m[(j-1)*ancho+i]+1),             // Insercion
                    m[(j-1)*ancho+i-1]+costo); }      // Sustitucion

// Devolvemos esquina final de la matriz
   res=m[t2*ancho+t1];
   free(m);
   return(res);
}
int Minimo(int a, int b)
{
   if(a<b)
      return a;
   else return b;
}
int maximo(int a, int b)
{
   if(a>b)
      return a;
   else return b;
}
/*** FUNCIONES LPS ***/
// COSTO ASCII TRIPLETA
int triple_ascii(ttriple t)
{
   int a = 0;
   return a = ascii(t->sujeto->label) + ascii(t->predicado) + ascii(t->objeto->label);
}
// COSTO ASCII LABEL
int ascii(tlabel t)
{
   int i, a = 0;
   for(i=0; i<strlen(t); i++)
   {
      a = a + t[i];
   }

   return a;
}
```

ANEXO E – Implementación sinónimos RDF y LPS sin WordNet.

El anexo E incluye la implementación de la función de reconocimiento de sinónimos de WordNet, utilizando como base el algoritmo orientado a la similaridad de grafos RDF, propuesto por la Universidad de Yunnan, la fórmula de similaridad Wu y Palmer, pero sin ocupar los archivos base de WordNet, para ello, se diseñó una estructura alternativa donde se almacena la información necesaria de WordNet para realizar los cálculos de similaridad. En dicha estructura se cargan los datos necesarios, correspondiente para el caso de las pruebas de 25 palabras sinónimas que deberán ser identificadas por el algoritmo. En una primera instancia se cargan las palabras en la estructura y posteriormente se procede a leer en los archivos de WordNet por cada palabra en la estructura y almacenar la información correspondiente. De esta forma, cuando se quiera calcular la similaridad de los palabras cualesquiera, se ocupará la información almacenada en memoria en la estructura y no en los archivos base de WordNet.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
#include "search_wn_SIM_WuP.h"

/*** VALORES INICIALES ***/

// Triple Sim variables

#define TRIPS 0.4
#define TRIPP 0.2
#define TRIPO 0.4

// Sim structural

#define ASTR 0.5
#define BSTR 0.5

// sim struct
#define A_STATE 0.5
#define B_STATE 0.5
```

```
/*** Estructuras ***/

typedef char *tlabel;       // puntero para las etiquetas de los nodos

typedef struct uri          // Estructura para almacenar las partes de las Uri
{
      tlabel sch;
      tlabel host;
      tlabel path;
      tlabel frag;
}*turi;

// LISTA PARA ALMACENAR LAS TRIPLETAS DEL GRAFO. //Sirve para "Arcos",
"Salidas-Llegadas" y listas LPS

typedef struct triple
{
   // Uso Arcos
   struct nodo *sujeto;    // Puntero a nodo "origen"
   tlabel predicado;       // Etiqueta del predicado
   struct nodo *objeto;    // Puntero a nodo "destino"

   // Uso Salida
   int costo;              // Costo ascii de la tripleta salida (suj+pred+obj)

   // Uso LPS
   int longitud;             // Nº de tripletas en la lista LPS
   int costoTotal;         // Costo acumulado de las tripletas que conforman el LPS
   int bk;                 // Bandera "backwards" (Inicia en 0)
   struct triple *inicio;  // Ptr a la tripleta inicio del LPS
   struct triple *fin;     // Ptr a la tripleta final del LPS ( final-sig => NULL )

   struct triple *sig;
}*ttriple;

// LISTA PARA ALMACENAR LOS NODOS DEL GRAFO.

typedef struct nodo
{
   int nodo;               // N° del nodo (Identificador)
   int tipo;               // 0=inicia, 1=URI, 2=variable
   tlabel label;           // Etiqueta del nodo
   ttriple llegada;        // Lista, almacena las tripletas que "llegan" al nodo actual
   ttriple salida;         // Lista, almacena las tripletas que "salen" desde el nodo actual

   // Uso LPS
```

```
    int cost;           // Costo ascii del label del nodo
    int marca;          // Marca para visitar el LPS    (Futuro Uso)
    int marcaC;         // Marca los Ciclos.          (Futuro Uso)
    int cant;           // Cantidad de elementos en la tripleta salida ("Aridad" del nodo Objeto de
la tripleta)

    struct nodo *sig;
}*tnodo;


// LISTA PARA ALMACENAR EL O LOS GRAFOS. (Pensado para más adelante cuando se
necesite ingresar varios grafos en la base de datos)

typedef struct grafo
{
    int n;              // número de nodos que tiene el grafo
    tnodo nodo;         // lista, almacena los nodos del grafo.
    struct grafo *sig;
}*tgrafo;

static char uriLine[1024];

typedef struct grilla
{
    int id;
    tlabel word;
    tindex_ofs inf;
}*tgrilla;

/*** FUNCIONES ***/

tnodo crear(void);
tnodo inserta_nodo(tlabel label, tnodo g);
void imprime_nodo(tnodo g);
int existe_nodo(tlabel label, tnodo g);
tnodo get_nodo(tlabel label, tnodo g);
void inserta_tripleta(tlabel sujeto, tlabel predicado, tlabel objeto, tnodo g);
void datos_nodo(tnodo g);
void datos_triple(ttriple t);
void imprime_arcos(tnodo g);
void mostrar_grafo(tnodo g);
int get_label_type(tlabel label);
int Levenshtein(tlabel s1, int t1, tlabel s2, int t2);
double sim_string(tlabel s1, tlabel s2);
int Minimo(int a, int b);
void cut_uri(tlabel uriX, turi u);
```

```
double sim_uri_uri(tlabel uriA, tlabel uriB);
double sim_uri_wrd(tlabel uriA, tlabel wordB);
double sim_label(tlabel labelA, tlabel labelB);

/*** triple sim functions ***/
void load_file(FILE *f, tnodo g);

/*** Sim RDF Functions ***/
double sim_rdf(tnodo gq, tnodo gt);
tnodo first_nodo(tnodo g);
ttriple first_salida(tnodo n, tnodo g);
ttriple first_llegada(tnodo n, tnodo g);
ttriple next_salida(tnodo n, ttriple t, tnodo g);
ttriple next_llegada(tnodo n, ttriple t, tnodo g);
tnodo next_nodo(tnodo n, tnodo g);
tnodo nodo_suj(ttriple t, tnodo g);
tnodo nodo_obj(ttriple t, tnodo g);
tlabel label_triple(ttriple t, tnodo g);
double sim_triple(ttriple query, tnodo gq, ttriple target, tnodo gt );
double sim_upper(ttriple query, tnodo gq, ttriple target, tnodo gt);
double sim_lower(ttriple query, tnodo gq, ttriple target, tnodo gt);
double sim_struct(ttriple query, tnodo gq, ttriple target, tnodo gt);
double sim_state(ttriple query, tnodo gq, ttriple target, tnodo gt);

/*** LPS Functions ***/
ttriple crear_triple();
ttriple LPS(tnodo g);
ttriple getLps(ttriple t);
double sim_lps(ttriple qt, ttriple tt);
void mostrar_LPS(ttriple t);

void load_fileB(FILE *f, tnodo g);

double sim_lin2(tlabel A, tlabel B);

tgrilla query;
tgrilla target;

int main()
{
   int i;

   query = (tgrilla)malloc(110*sizeof(struct grilla));
   if(query == NULL)
   {
      printf("Error al crear grilla query\n");
```

```
   }else{
      for(i=0;i<27;i++)
         {
            query[i].id = i;
            query[i].word = (char *)malloc(100*sizeof(char));
            if(query[i].word == NULL)
            printf("Query[%s]->word - Error Memoria Insuficiente", i);
            query[i].inf = NULL;
         }
      }

   // QUERY

strcpy(query[1].word,"Leonardo_da_Vinci");
strcpy(query[2].word,"abdominal_aortic_aneurysm");
strcpy(query[3].word,"aachen");
strcpy(query[4].word,"aland_islands");
strcpy(query[5].word,"aalborg");
strcpy(query[6].word,"alost");
strcpy(query[7].word,"aare");
strcpy(query[8].word,"aardvark");
strcpy(query[9].word,"aardwolf");
strcpy(query[10].word,"common_mullein");
strcpy(query[11].word,"copland");
strcpy(query[12].word,"abalone");
strcpy(query[13].word,"abampere");
strcpy(query[14].word,"foundling");
strcpy(query[15].word,"abashment");
strcpy(query[16].word,"abattis");
strcpy(query[17].word,"abattoir");
strcpy(query[18].word,"abbess");
strcpy(query[19].word,"abbot");
strcpy(query[20].word,"abridger");
strcpy(query[21].word,"abdominal");
strcpy(query[22].word,"abdominal_nerve_plexus");
strcpy(query[23].word,"abdominocentesis");
strcpy(query[24].word,"abdominoplasty");
strcpy(query[25].word,"abdominousness");
strcpy(query[26].word,"abducent");
```

```
//VISUALIZACION
printf("\n\nDATOS QUERY\n\n");
for(i=1;i<27;i++)
{
   printf("ID: %i\n", query[i].id);
   printf("%i) %s\n", i, query[i].word);

}
//
   target = (tgrilla)malloc(110*sizeof(struct grilla));
   if(target == NULL)
   {
      printf("Error al crear grilla target\n");
   }else{
      for(i=0;i<27;i++)              // i < (total - 1)
         {
            target[i].id = i;
            target[i].word = (char *)malloc(100*sizeof(char));
            if(target[i].word == NULL)
            printf("Target[%s]->word - Error Memoria Insuficiente", i);

            target[i].inf = NULL;
         }
      }

strcpy(target[1].word,"Leonardo_da_Vinci");
strcpy(target[2].word,"aaa");
strcpy(target[3].word,"aken");
strcpy(target[4].word,"ahvenanmaa");
strcpy(target[5].word,"alborg");
strcpy(target[6].word,"aalost");
strcpy(target[7].word,"aare_river");
strcpy(target[8].word,"orycteropus_afer");
strcpy(target[9].word,"proteles_cristata");
strcpy(target[10].word,"aaron's_rod");
strcpy(target[11].word,"aaron_copland");
strcpy(target[12].word,"ear-shell");
strcpy(target[13].word,"abamp");
strcpy(target[14].word,"abandoned_infant");
strcpy(target[15].word,"bashfulness");
strcpy(target[16].word,"abatis");
strcpy(target[17].word,"slaughterhouse");
strcpy(target[18].word,"prioress");
strcpy(target[19].word,"archimandrite");
strcpy(target[20].word,"abbreviator");
strcpy(target[21].word,"abdominal_muscle");
```

```
    strcpy(target[22].word,"solar_plexus");
    strcpy(target[23].word,"paracentesis");
    strcpy(target[24].word,"tummy_tuck");
    strcpy(target[25].word,"paunchiness");
    strcpy(target[26].word,"abducens");


    //VISUALIZACION
    printf("\n\nDATOS TARGET\n\n");
    for(i=1;i<27;i++)
    {
       printf("ID: %i\n", target[i].id);
       printf("%i) %s\n", i, target[i].word);

    }
    //***/

    //

    //**

    // CARGAR DATOS  // INDEX A STRUCT

       FILE *f;
       f = fopen("index.noun", "r");

       char *ptrtok, *ptB;

       if(f==NULL){
          printf("No se ha podido abrir el fichero.\n");
          exit(1);
       }

       //*** CARGAR MANUALMENTE LOS DATOS DE QUERY[1] => LEONARDO DA
VINCI

       tindex_ofs aux_A = NULL;
       aux_A = (tindex_ofs)malloc(sizeof(index_ofs));

       //aux_A->word = '\0';
       //aux_A->tipo = '\0';
       //aux_A->off_cont = 1;
       //aux_A->ptr_cont = 0;

       aux_A->offsets = (long *)malloc(sizeof(long));
```

```
    aux_A->offsets[0] = 00000000;

    query[1].inf = aux_A;
    query[1].id = 1;
    aux_A = NULL;

//*** CARGAR DATOS QUERY DEL INDEX A LA ESTRUCTURA

    int z;
    for(z=2; z<27; z++)
    {
       ptrtok = NULL;
       strcpy(line, bin_search(query[z].word,f));
       //printf("A: Line found on Index.pos:\n\n%s\n",line);

       if(strcmp(line,"NF") == 0)
          {
          fclose(f);
          printf("No está la palabra en WN\n");
          exit(1);
          }

       tindex_ofs aux = NULL;
       aux = (tindex_ofs)malloc(sizeof(index_ofs));


//       /***    CORTAR CADENA


       aux->word = (char *)malloc(sizeof(char));
       ptrtok = strtok(line," \n");
       aux->word = ptrtok;

       aux->tipo = (char *)malloc(sizeof(char));
       ptrtok = strtok(NULL," \n");
       aux->tipo = ptrtok;

       ptrtok = strtok(NULL," \n");
       aux->off_cont = atoi(ptrtok);

       ptrtok = strtok(NULL," \n");
       aux->ptr_cont = atoi(ptrtok);

       for(i=0; i<(aux->ptr_cont+2); i++)
       {
          ptrtok = strtok(NULL," \n");
```

```
        }

        aux->offsets = (long *)malloc(aux->off_cont * (sizeof(long)));


        for(i=0; i<aux->off_cont; i++)
        {
            ptrtok = strtok(NULL," \n");
            aux->offsets[i] = atol(ptrtok);
        }

        query[z].inf = aux;
        aux = NULL;
    }

    //VERIFICACION
    printf("\n\nVerificacion Carga - QUERY\n\n");
    int k;
    for(k=1; k<27; k++)
    {
        printf("\nINFO\n");
        printf("ID: %i\n", query[k].id);
        printf("tlabel word: %s\n", query[k].word);
        //printf("Word: %s\n", query[k].inf->word);
        //printf("Tipo: %s\n", query[k].inf->tipo);
        //printf("Off cont: %i\n", query[k].inf->off_cont);
        //printf("Ptr cont: %i\n", query[k].inf->ptr_cont);
        //printf("Offsets totales: %i\n\n",query[k].inf->off_cont);


        printf("Offset[0]: %.8lu \n", query[k].inf->offsets[0]);




    }

    printf("\nFIN\n");


  //*** CARGAR DATOS TARGET DEL INDEX A LA ESTRUCTURA


    //*** CARGAR MANUALMENTE LOS DATOS DE TARGET[1] => LEONARDO DA
VINCI
```

```
tindex_ofs aux_B = NULL;
aux_B = (tindex_ofs)malloc(sizeof(index_ofs));
//aux_B->word = '\0';
//aux_B->tipo = '\0';
//aux_B->off_cont = 1;
//aux_B->ptr_cont = 0;
aux_B->offsets = (long *)malloc(sizeof(long));
aux_B->offsets[0] = 00000000;
target[1].inf = aux_B;
target[1].id = 1;
aux_B = NULL;

for(z=2; z<27; z++)
{
   ptB = NULL;
   strcpy(lineB, bin_search(target[z].word,f));
   //printf("A: Line found on Index.pos:\n\n%s\n",line);

   if(strcmp(lineB,"NF") == 0)
      {
      fclose(f);
      printf("No está la palabra en WN - target\n");
      exit(1);
      }

   tindex_ofs auxB = NULL;
   auxB = (tindex_ofs)malloc(sizeof(index_ofs));


//       /***    CORTAR CADENA

   auxB->word = (char *)malloc(sizeof(char));
   ptB = strtok(line," \n");
   auxB->word = ptB;

   auxB->tipo = (char *)malloc(sizeof(char));
   ptB = strtok(NULL," \n");
   auxB->tipo = ptB;

   ptB = strtok(NULL," \n");
   auxB->off_cont = atoi(ptB);

   ptB = strtok(NULL," \n");
   auxB->ptr_cont = atoi(ptB);
```

```
for(i=0; i<(auxB->ptr_cont+2); i++)
      {
         ptB = strtok(NULL," \n");
      }

      auxB->offsets = (long *)malloc(auxB->off_cont * (sizeof(long)));


      for(i=0; i<auxB->off_cont; i++)
      {
         ptB = strtok(NULL," \n");
         auxB->offsets[i] = atol(ptB);
      }

      target[z].inf = auxB;
      auxB = NULL;
   }

   free(f);


   //VERIFICACION
   printf("\n\nVerificacion Carga - TARGET\n\n");

   for(k=1; k<27; k++)
   {
      printf("\nINFO\n");
      printf("ID: %i\n", target[k].id);
      printf("tlabel word: %s\n", target[k].word);
      //printf("Word: %s\n", target[k].inf->word);
      //printf("Tipo: %s\n", target[k].inf->tipo);
      //printf("Off cont: %i\n", target[k].inf->off_cont);
      //printf("Ptr cont: %i\n", target[k].inf->ptr_cont);
      //printf("Offsets totales: %i\n\n",target[k].inf->off_cont);
      printf("Offset[0]: %.8lu \n", target[k].inf->offsets[0]);
   }
   printf("\nFIN\n");

   //*** Carga Archivos

   // Grafo Query ---------

   tnodo gq;
   gq = crear();
```

```
FILE *q;
   q = fopen("query.txt", "r");

   if(q==NULL){
      printf("No se ha podido abrir el fichero.\n");
      exit(1);
   }
   load_file(q, gq);
   fclose(q);

   //Mostrar Grafo Query
   printf("\n\n NODOS GRAFO QUERY \n\n");
   imprime_nodo(gq);
   printf("\n\n - GRAFO QUERY - \n\n");
   mostrar_grafo(gq);

   // Grafo Target ---------

   tnodo gt;
   gt = crear();

   FILE *t;
   t = fopen("target.txt", "r");

   if(t==NULL){
      printf("No se ha podido abrir el fichero.\n");
      exit(1);
   }
   load_fileB(t, gt);
   fclose(t);

   //Mostrar Grafo Target
   printf("\n\n NODOS GRAFO TARGET \n\n");
   imprime_nodo(gt);
   printf("\n\n - GRAFO TARGET - \n\n");
   mostrar_grafo(gt);

// /*** CALCULO DE SIMILARIDAD RDF Y LPS ***/

   // 1 - RDF / 0 - LPS
```

```
if(1)
  {
     printf("SIM RDF\n\n");
     double sr;

     clock_t start = clock();

     sr = sim_rdf(gq, gt);

     printf("\n\nTIME: %f\n", ((double)clock() - start) / CLOCKS_PER_SEC);

     printf("\nRDF SIMILARITY: %f\n", sr);
  }else{
     ttriple lps_Q, lps_G;

     lps_Q = LPS(gq);
     printf("\n\n - *** LPS QUERY *** - \n\n");
     mostrar_LPS(lps_Q);

     lps_G = LPS(gt);
     printf("\n\n - *** LPS TARGET *** - \n\n");
     mostrar_LPS(lps_G);

     double sl;
     clock_t start = clock();
     sl = sim_lps(lps_Q, lps_G);
     printf("\n\nTIME: %f\n\n", ((double)clock() - start) / CLOCKS_PER_SEC);

     printf("\nLPS SIMILARITY: %f\n", sl);
     }




  free(gq);

   free(gt);



return 0;
}
```

```
void mostrar_LPS(ttriple t)
{
   printf("\n\n*** MOSTRAR LPS ***\n\n");

   while(t != NULL)
   {
      printf("(%s) - (%s) - (%s)\n", t->sujeto->label, t->predicado, t->objeto->label);
      printf("COSTO TOTAL: %i\n", t->costoTotal);

      t = t->sig;
   }

   printf("\n\n*** FIN LPS ***\n\n");
}

ttriple LPS(tnodo g)
{
   tnodo n;
   ttriple MaxTotal, maxi;
   ttriple aux;
   int b = 0;

   n = first_nodo(g);

   while(n != NULL)    //Recorre los nodos del grafo
   {
      if(n->llegada == NULL)  // Verificación de que la tripleta sea la raíz (que no tenga
tripletas que "apuntan" a ella)(si no es el primer nodo, busca en el grafo)
      {
         aux = n->salida;
         b = 1;
         while(aux != NULL)
         {
            if(b)                    // Si es la primera vez que entra al while
            {
               maxi = getLps(aux);
               MaxTotal = maxi;
               b = 0;
            }else {
                  maxi = getLps(aux);
                  }
```

```
   if(maxi->longitud > MaxTotal->longitud)
            {
               MaxTotal = maxi;
               maxi = NULL;
            }else if(maxi->longitud == MaxTotal->longitud)
                  {
                     if(maxi->costoTotal > MaxTotal->costoTotal)
                     {
                        MaxTotal = maxi;
                        maxi = NULL;
                     }else if(maxi->costoTotal < MaxTotal->costoTotal)       // NO PUEDE SER
IGUAL, PORQUE NO HAY TRIPLETAS REPETIDAS
                           {
                              /*** Destroy Maxi ***/
                           }
                  }else if(maxi->longitud < MaxTotal->longitud)
                        {
                           /*** Destroy Maxi ***/
                        }

            aux = aux->sig;
            }

         }

         n = next_nodo(n,g);
      }
   return MaxTotal;
   }

   ttriple getLps(ttriple t)
   {
      ttriple aux = t->objeto->salida;

      if(aux == NULL)         // Si no tiene hijos;
      {
         ttriple bkw = crear_triple();   // Inicializar tripleta
         // llenar bkw con los datos de la tripleta ingresada a la función ( t )
         bkw->sujeto = t->sujeto;

         bkw->predicado = (char *)malloc((strlen(t->predicado))*sizeof(char));
         if(bkw->predicado == NULL)
            printf("Error Memoria Insuficiente");

         strcpy(bkw->predicado, t->predicado);
         bkw->objeto = t->objeto;
```

```
   bkw->costo = t->costo;
   bkw->longitud = 1;
   bkw->costoTotal = t->costo;
   bkw->bk = 1;
   bkw->inicio = t;
   bkw->fin = t;
   bkw->sig = NULL;

   return bkw;
}else{
   ttriple MaxTotal, maxi;
   int b = 1;
   while(aux != NULL)
{
   if(b)                    // Es la primera vez que entra al while
   {
      //ttriple maxi;
      maxi = getLps(aux);
      MaxTotal = maxi;
      b = 0;
   }
   else{                    // No es la primera vez que entra al while
      //ttriple maxi;
      maxi = getLps(aux);
   }

   if(maxi->longitud > MaxTotal->longitud)
   {
      /*** A ***/
      MaxTotal = maxi;
      maxi = NULL;

   }else
      if(maxi->longitud == MaxTotal->longitud)
      {
         if(maxi->costoTotal > MaxTotal->costoTotal)
         {
            /*** A ***/
            MaxTotal = maxi;
            maxi = NULL;

         }else
            if(maxi->costoTotal < MaxTotal->costoTotal)
            {
               /*** DESTROY maxi ***/
            }
```

```
        }else
           if(maxi->longitud < MaxTotal->longitud)
           {
              /*** DESTROY maxi ***/
           }

      aux = aux->sig;
   }

   /*** Agregar la tripleta ( t ) al mayor de sus hijos (LPS acumulado)***/

   ttriple head;
   head = crear_triple();
   //llenar cabeza con los datos de la tripleta ( t ) ingresada
   head->sujeto = t->sujeto;

   head->predicado = (char *)malloc((strlen(t->predicado))*sizeof(char));
      if(head->predicado == NULL)
         printf("Error Memoria Insuficiente");

   strcpy(head->predicado, t->predicado);
   head->objeto = t->objeto;
   head->costo = t->costo;
   head->longitud = MaxTotal->longitud + 1;
   head->costoTotal = MaxTotal->costoTotal + t->costo;
   head->bk = 1;
   head->inicio = head;
   head->fin = MaxTotal->fin;
   head->sig = MaxTotal;

   MaxTotal->inicio = head;

   return head;
   }
}

ttriple crear_triple()
{
   ttriple aux;
   aux = (ttriple)malloc(sizeof(struct triple));
   if(aux == NULL)
      printf("Crear triple: Error al crear triple\n");
   else{
      aux->sujeto = NULL;
      aux->predicado = NULL;
      aux->objeto = NULL;
```

```
        aux->costo = 0;
        aux->longitud = 0;
        aux->costoTotal = 0;
        aux->bk = 0;
        aux->inicio = NULL;
        aux->fin = NULL;
        aux->sig = NULL;
        return aux;
    }
}

double sim_lps(ttriple qt, ttriple tt)
{
    ttriple q = qt;
    ttriple t = tt;
    double simrdf = 0;
    int deg_tot = 0;

    printf("\n\n*** INSIDE SIM LPS ***\n\n");

    while(q != NULL)
    {
        printf("| %s | -> %s -> | %s |\n",q->sujeto->label, q->predicado, q->objeto->label);

            double simestate = 0, aux = 0;
            int deg_t = 0;


        while(t != NULL)
        {
            printf("       | %s | -> %s -> | %s |\n",t->sujeto->label, t->predicado, t->objeto->label);


            printf("\nCOMPARAR:\n(%s) - %s - (%s) vs (%s) - %s - (%s)\n", q->sujeto->label, q-
>predicado, q->objeto->label, t->sujeto->label, t->predicado, t->objeto->label);

            aux = sim_state(q, NULL, t, NULL);

            if(aux > simestate)
                {
                    simestate = aux;

                    if(t->inicio == t || t->fin == t)
                        deg_t = 1;
                    else deg_t = 2;
```

```
                    //deg_t = deg(t, NULL);
                    printf("aux>sim_state: %s-%s-%s vs %s-%s-%s\n", q->sujeto->label, q-
>predicado, q->objeto->label, t->sujeto->label, t->predicado, t->objeto->label);
                }

                printf("Sim_rdf: Sim_state: %f - deg_t: %i\n", simestate, deg_t);

            t = t->sig;
        }

        //guardar el resultado para la primera tripleta, antes de llamar a la siguiente.

            int deg_aux = 0;
            deg_aux = deg_t;
            if(deg_aux == 0)
               deg_aux = 1;

            simrdf = simrdf + (simestate * deg_aux);
            deg_tot = deg_tot + deg_aux;

            q = q->sig;

            printf("SIM RDF === %f\n", simrdf);
            printf("Deg_t === %i\n", deg_t);
            printf("Deg_tot === %i\n", deg_tot);
    }

  //return simrdf / deg_tot;      // formula original
  return simrdf / deg_tot;      // variación no considerada en el artículo.
  }


  double sim_rdf(tnodo gq, tnodo gt)
  {
     tnodo q, t;
     ttriple sq, st;
     double simrdf = 0;
     int deg_tot = 0;

     q = first_nodo(gq);
     while(q != NULL)
     {
        printf("\nQUERY: Nodo N %i - ( %s )\n", q->nodo, q->label);
```

```
        sq = first_salida(q, gq);
        while(sq != NULL)
        {
           printf("| %s | -> %s -> | %s |\n",sq->sujeto->label, label_triple(sq,q), sq->objeto-
>label);

           double simestate = 0, aux = 0;
           int deg_t = 0;

           /*** recorrer el grafo target ***/
           t = first_nodo(gt);
           while(t != NULL)
           {
              printf("\n    TARGET: Nodo N %i - ( %s )\n", t->nodo, t->label);

              st = first_salida(t, gt);
              while(st != NULL)
              {
                 printf("       | %s | -> %s -> | %s |\n",st->sujeto->label, label_triple(st,t), st-
>objeto->label);


                 printf("\nCOMPARAR:\n(%s) - (%s) vs (%s) - (%s)\n", sq->sujeto->label, sq-
>objeto->label, st->sujeto->label, st->objeto->label);

                 aux = sim_state(sq, gq, st, gt);

                 if(aux > simestate)
                 {
                    simestate = aux;
                    deg_t = deg(st, gt);
                    printf("aux>sim_state: %s-%s-%s vs %s-%s-%s\n", sq->sujeto->label, sq-
>predicado, sq->objeto->label, st->sujeto->label, st->predicado, st->objeto->label);
                 }


                 printf("Sim_rdf: Sim_state: %f - deg_t: %i\n", simestate, deg_t);

                 st = next_salida(t, st, gt);

              }



              t = next_nodo(t, gt);
           }
```

```
            //guardar el resultado para la primera tripleta, antes de llamar a la siguiente.

            int deg_aux = 0;
            deg_aux = deg_t;
            if(deg_aux == 0)
               deg_aux = 1;

            simrdf = simrdf + (simestate * deg_aux);
            deg_tot = deg_tot + deg_aux;
            sq = next_salida(q, sq, gq);

            printf("simrdf === %f\n", simrdf);
            printf("deg_t === %i\n", deg_t);
            printf("deg_tot === %i\n", deg_tot);

          }
          q = next_nodo(q, gq);
          }
    return simrdf / deg_tot;
    }

    double sim_state(ttriple query, tnodo gq, ttriple target, tnodo gt)
    {
       double ss = 0;
       ss = A_STATE * sim_triple(query, gq, target, gq) + B_STATE * sim_struct(query, gq,
target, gt);
       printf("Sim_state: %f\n", ss);
       return ss;
    }

    double sim_struct(ttriple query, tnodo gq, ttriple target, tnodo gt)
    {
       if(deg(query, gq) == 0 && deg(target, gt) == 0)
       {
        printf("Sim_struct: 1\n[deg(query) y deg(target) son cero]\n");
        return 1;
       }
       else
       {
          double ss = 0;
       ss = ASTR * sim_upper(query, gq, target, gt) + BSTR * sim_lower(query, gq, target, gt);
       printf("Sim_struct: %f\n", ss);
       return ss;
       }
    }
```

```
double sim_upper(ttriple query, tnodo gq, ttriple target, tnodo gt)
{
 ttriple Qlleg, Tlleg;
 Qlleg = first_llegada(query->sujeto, gq);
 Tlleg = first_llegada(target->sujeto, gt);

 if(Qlleg == NULL && Tlleg == NULL)
  return 1;

 int k = 0, b = 0;
 double maxUp = 0, simUpper = 0, simUp = 0, aux = 0;

 //recorrer lista llegada del sujeto de la tripleta query
 Qlleg = first_llegada(query->sujeto, gq);
 while(Qlleg != NULL)
 {
      // recorrer lista llegada del sujeto de la tripleta target

      Tlleg = first_llegada(target->sujeto, gt);
      while(Tlleg != NULL)
      {
         aux = sim_triple(Qlleg, gq, Tlleg, gt);

         if(aux > simUp)
            simUp = aux;

         printf("simUp: %f\n", simUp);

        Tlleg = next_llegada(target->sujeto, Tlleg, gt);
      }

      maxUp = maxUp + simUp;
      k++;
      b = 1;
      Qlleg = next_llegada(query->sujeto, Qlleg, gq);
 }

 if(b)
  return simUpper = maxUp / k;
  else return 0;
}

double sim_lower(ttriple query, tnodo gq, ttriple target, tnodo gt)
{

 ttriple Qsal, Tsal;
```

```
  Qsal = first_salida(query->objeto, gq);
  Tsal = first_salida(target->objeto, gt);

  if(Qsal == NULL && Tsal == NULL)
    return 1;

  int k = 0, b = 0;
  double maxUp = 0, simLower = 0, simLw = 0, aux = 0;

  //recorrer lista salida del objeto de la tripleta query
  Qsal = first_salida(query->objeto, gq);
  while(Qsal != NULL)
  {
       // recorrer lista salida del objeto de la tripleta target

       Tsal = first_salida(target->objeto, gt);
       while(Tsal != NULL)
       {
          aux = sim_triple(Qsal, gq, Tsal, gt);

          if(aux > simLw)
             simLw = aux;

          printf("simLw: %f\n", simLw);

         Tsal = next_salida(target->objeto, Tsal, gt);
       }

       maxUp = maxUp + simLw;
       k++;
       b = 1;
       Qsal = next_salida(query->objeto, Qsal, gq);
  }

  if(b)
    return simLower = maxUp / k;
    else return 0;
}

int indeg(ttriple t, tnodo gt)
{
    ttriple Tlleg;
    int in = 0;

    Tlleg = first_llegada(t->sujeto, gt);
    while(Tlleg != NULL)
```

```
    {
        in++;

        Tlleg = next_llegada(t->sujeto, Tlleg, gt);
    }
    printf("Indeg: %i\n", in);
    return in;
}

int outdeg(ttriple t, tnodo gt)
{
    ttriple Tsal;
    int out = 0;

    Tsal = first_salida(t->objeto, gt);
    while(Tsal != NULL)
    {
        out++;

        Tsal = next_salida(t->objeto, Tsal, gt);
    }
    printf("Outdeg: %i\n", out);
    return out;
}

int deg(ttriple t, tnodo gt)
{
    int dg = 0;
    dg = indeg(t, gt) + outdeg(t, gt);
    printf("DEG: %i\n", dg);
    return dg;
}

double sim_triple(ttriple query, tnodo gq, ttriple target, tnodo gt )
{
    double Strip, Ptrip, Otrip, SimT;

    Strip = TRIPS*sim_label(query->sujeto->label, target->sujeto->label);

    Ptrip = TRIPP*sim_string(query->predicado, target->predicado);

    Otrip = TRIPO*sim_label(query->objeto->label, target->objeto->label);

    SimT = Strip + Ptrip + Otrip;

    printf("Sim_triple: %f\n", SimT);
```

```
   return SimT;
}

void load_file(FILE *f, tnodo g)
{
   char A[100],B[100],C[100];
   tlabel x,y,z;
   x = A;
   y = B;
   z = C;

   while(!feof(f)){
      fscanf(f, "%s", A);
      fscanf(f, "%s", B);
      fscanf(f, "%s", C);

      printf("- Tripleta a Ingresar - \n\n");
      printf(" A es igual a: %s\n", x);
      printf(" B es igual a: %s\n", y);
      printf(" C es igual a: %s\n\n", z);

      inserta_tripleta(x,y,z,g);
   }
}

void load_fileB(FILE *f, tnodo g)
{
   char A2[100],B2[100],C2[100];
   tlabel x2,y2,z2;
   x2 = A2;
   y2 = B2;
   z2 = C2;

   while(!feof(f)){
      fscanf(f, "%s", A2);
      fscanf(f, "%s", B2);
      fscanf(f, "%s", C2);


      printf("- Tripleta a Ingresar - \n\n");
      printf(" A es igual a: %s\n", x2);
      printf(" B es igual a: %s\n", y2);
      printf(" C es igual a: %s\n\n", z2);


      inserta_tripleta(x2,y2,z2,g);
```

```
    }
}

double sim_label(tlabel labelA, tlabel labelB)
{
    int tipoA, tipoB;
    double sLabel = 0;

    tipoA = get_label_type(labelA); // 1-uri - 2-word
    tipoB = get_label_type(labelB);

    if(tipoA == -1 || tipoB == -1)
        return 0;
    else if(tipoA == 1 && tipoB == 1)
        sLabel = sim_uri_uri(labelA, labelB);
        else if(tipoA == 1 && tipoB == 2)
            sLabel = sim_uri_wrd(labelA, labelB);
            else if(tipoA == 2 && tipoB == 1)
                sLabel = sim_uri_wrd(labelB, labelA);
                else if(tipoA == 2 && tipoB == 2)
                {
                    sLabel = sim_lin2(labelA, labelB);
                    if(sLabel == -1)
                        sLabel = sim_string(labelA, labelB);
                }
                return sLabel;
}

double sim_lin2(tlabel A, tlabel B)
{
    int i, j;
    int idA = 0, idB = 0;

   //if(strcmp(A, Leonardo_da_Vinci))

    idA = get_idx(A, 1);
    idB = get_idx(B, 0);

    if(query[idA].inf->offsets[0] == target[idB].inf->offsets[0])
    {
        return 1;
    }else return -1;
}
```

```
int get_idx(tlabel A, int x)
{
   int i, j, b;
   b = 1;
   int id = 0;


   if(x)      // 1 = QUERY // 0 = TARGET
   {
      for(i=1; i< 27 && b; i++)
         {
            if(strcmp(A,query[i].word) == 0)
            {
               b = 0;
               id = query[i].id;
            }
         }

   }else{
         for(i=1; i< 27 && b; i++)
         {
            if(strcmp(A,target[i].word) == 0)
            {
               b = 0;
               id = target[i].id;
            }
         }
      }

return id;
}

double sim_uri_wrd(tlabel uriA, tlabel wordB)
{
   turi uriQ;

   uriQ = (turi)malloc(sizeof(struct uri));
   if(uriQ == NULL)
   {
      printf("Sim_uri_wrd: No hay espacio disponible para UriQ");
   }

   cut_uri(uriA, uriQ);

   /*** FRAGMENT ***/   /*** DEBEN ESTAR EN MINUSCULAS ***/
```

```
    double sFrag = 0;

    if(uriQ->frag == NULL)
       return 0;
    else sFrag = sim_lin2(uriQ->frag, wordB);
    if(sFrag == -1)
       sFrag = sim_string(uriQ->frag, wordB);


    printf("Sim Frag: %f\n", sFrag);

    return sFrag;
 }

 double sim_uri_uri(tlabel uriA, tlabel uriB)
 {
    turi uriQ, uriT;

    uriQ = (turi)malloc(sizeof(struct uri));
    if(uriQ == NULL)
    {
       printf("Sim_uri_uri: No hay espacio disponible para UriQ");
    }

    uriT = (turi)malloc(sizeof(struct uri));
    if(uriT == NULL)
    {
       printf("Sim_uri_uri: No hay espacio disponible para UriT");
    }

    printf("\n uriA: %s\n", uriA);
    printf("\n uriB: %s\n", uriB);


    cut_uri(uriA, uriQ);
    cut_uri(uriB, uriT);

    printf("\nAFTER CUT URI\n");

    // CALCULAR SIMILITUDES

    printf("Q - Sch: %s\n Host: %s\n Path: %s\n Frag: %s\n\n", uriQ->sch, uriQ->host, uriQ-
>path, uriQ->frag);
    printf("T - Sch: %s\n Host: %s\n Path: %s\n Frag: %s\n\n", uriT->sch, uriT->host, uriT-
>path, uriT->frag);
```

```
    /*** HOST ***/

    double sHost = 0;
    sHost = sim_string(uriQ->host, uriT->host);

    printf("Sim Host: %f\n\n", sHost);


    /*** PATH ***/

    double sPath = 0;
    sPath = sim_string(uriQ->path, uriT->path);

    printf("Sim Path: %f\n", sPath);

    /*** FRAGMENT ***/   /*** DEBEN ESTAR EN MINUSCULAS ***/

    double sFrag = 0;
    if(uriQ->frag == NULL && uriT->frag == NULL)
       sFrag = 1;
    else if(uriQ->frag == NULL || uriT->frag == NULL)
       sFrag = 0;
       else sFrag = sim_lin2(uriQ->frag, uriT->frag);

    if(sFrag == -1)
       sFrag = sim_string(uriQ->frag, uriT->frag);

    printf("Sim Frag: %f\n", sFrag);

    /*** SIM TOTAL URI URI ***/

    double uriTot = (sHost + sPath + sFrag)/3;

    printf("Uri Total: %f\n", uriTot);

    return uriTot;
}
```

```
void cut_uri(tlabel uriX, turi u)   // recibe la linea uri y la struct que almacenará las partes
{
   // CORTAR CADENA

   char *upt = NULL;

      strcpy(uriLine, uriX);

      u->sch = (char *)malloc(100*sizeof(char));
      upt = strtok(uriLine,":");
      //u->sch = upt;
      strcpy(u->sch, upt);

      u->host = (char *)malloc(100*sizeof(char));
      upt = strtok(NULL,"/");
      //u->host = upt;
      strcpy(u->host, upt);

      /*** Si tiene Fragment (Separado por #)

      u->path = (char *)malloc(100*sizeof(char));
      upt = strtok(NULL,"#\n");
      //u->path = upt;
      strcpy(u->path, upt);

      u->frag = (char *)malloc(100*sizeof(char));
      upt = strtok(NULL,"\n");
      if(upt)
         strcpy(u->frag, upt);
      else u->frag = upt;

         ***/

      /*** Si no tiene Fragment - (Tipo wikipedia: "https://wiki.org/path/RECURSO"  ***/

      u->path = (char *)malloc(100*sizeof(char));
      upt = strtok(NULL,"/");
      //u->path = upt;
      strcpy(u->path, upt);

      u->frag = (char *)malloc(100*sizeof(char));
      upt = strtok(NULL,"\n");
      if(upt)
         strcpy(u->frag, upt);
      else u->frag = upt;
```

```
        printf("\nLinea: %s\n", uriX);
        printf(" Sch: %s\n Host: %s\n Path: %s\n Frag: %s\n", u->sch, u->host, u->path, u->frag);

    }

    tnodo crear(void)
    {
       tnodo aux;
       aux = (tnodo)malloc(sizeof(struct nodo));
       if(aux == NULL){
          printf("Error al Crear.");
       } else {
          aux->nodo = 0;
          aux->tipo = 0;
          aux->label = NULL;
          aux->llegada = NULL;
          aux->salida = NULL;

          aux->cost = 0;
          aux->marca = 0;
          aux->marcaC = 0;
          aux->cant = 0;


          aux->sig = NULL;
          return aux;
       }
    }


    tnodo inserta_nodo(tlabel label, tnodo g)
    {
       tnodo aux,p;
       aux = (tnodo)malloc(sizeof(struct nodo));
       if(aux == NULL)
          printf("Error Memoria Insuficiente");
       else {
          p = g;
          while(p->sig != NULL)
             p = p->sig;
          aux->label = (char *)malloc((strlen(label))*10*sizeof(char));
          if(aux == NULL)
             printf("Error Memoria Insuficiente");
          aux->nodo = p->nodo+1;
          aux->tipo = get_label_type(label);
```

```
        strcpy(aux->label,label);
        aux->llegada = NULL;
        aux->salida = NULL;

        aux->cost = ascii(label);
        aux->marca = 0;
        aux->marcaC = 0;
        aux->cant = 0;

        aux->sig = NULL;
        p->sig = aux;
        g->nodo++;
    }
        return aux;
}

void imprime_nodo(tnodo g)
{
    tnodo p;
    p=g;
    while(p->sig != NULL)
    {
        p=p->sig;
        printf("( Nodo N:%i - %s )",p->nodo ,p->label);

        if(p->tipo==0)
            printf(" - Aún no se ha asignado tipo\n");
        else if(p->tipo==1)
            printf(" - Label tipo URI\n");
        else if(p->tipo==2)
            printf(" - Label tipo WORD\n");
        else printf("%i - ERROR - Tipo fuera de parámetros\n", p->tipo);
    }
}

int existe_nodo(tlabel label, tnodo g)
{
    tnodo p;
    p=g;
    while(p->sig != NULL)
    {

        p=p->sig;
        if(strcmp(p->label,label)==0)
            return 1;
        else;
```

```
    }
    return 0;
}

tnodo get_nodo(tlabel label, tnodo g)
{
    tnodo p;
    p=g;
    while(p->sig != NULL)
    {

        p=p->sig;
        if(strcmp(p->label,label)==0)
            return p;
        else ;
    }
    return NULL;
}

void inserta_tripleta(tlabel sujeto, tlabel predicado, tlabel objeto, tnodo g)
{
    tnodo suj, obj;
    suj = get_nodo(sujeto, g);
    obj = get_nodo(objeto, g);

    if(suj == NULL && obj == NULL)    //si alguno de los dos no ha sido ingresado... se
ingresa
    {
        suj = inserta_nodo(sujeto, g);
        obj = inserta_nodo(objeto, g);
    }else if(suj != NULL && obj == NULL)
            obj = inserta_nodo(objeto, g);
            else if(suj == NULL && obj != NULL)
                    suj = inserta_nodo(sujeto, g);
                else {printf("\nWARNING - EL SUJETO Y EL OBJETO YA ESTÁN
INGRESADOS - REVISAR QUE EL PREDICADO ES DISTINTO\n\n");
                }
    ttriple llegada, salida, inicio, fin ;
    salida = (ttriple)malloc(sizeof(struct triple));
    llegada = (ttriple)malloc(sizeof(struct triple));

    inicio = (ttriple)malloc(sizeof(struct triple));
    fin = (ttriple)malloc(sizeof(struct triple));
```

```
      if((salida == NULL) || (llegada == NULL) || (inicio == NULL) || (fin == NULL))
          printf("salida,llegada,inicio,fin: Error Memoria Insuficiente: \nsalida: %s - llegada: %s -
inicio: %s - fin: %s\n",salida, llegada, inicio, fin);
       else{
          salida->predicado = (char*)malloc(strlen(predicado)*sizeof(char));
          if(salida->predicado == NULL)
          printf("salida->predicado: Error Memoria Insuficiente");

          salida->sujeto = suj;
          strcpy(salida->predicado, predicado);
          salida->objeto = obj;

          salida->costo = ascii(sujeto) + ascii(predicado) + ascii(objeto);

          salida->longitud = 0;
          salida->costoTotal = 0;
          salida->bk = 0;
          salida->inicio = inicio;
          salida->fin = fin;

          salida->sig = suj->salida;
          suj->salida = salida;

          suj->cant = suj->cant + 1;      // aumenta el contador de tripletas de salida en el nodo
sujeto


          llegada->predicado = (char*)malloc(strlen(predicado)*sizeof(char));
          if(llegada->predicado == NULL)
          printf("3 Error Memoria Insuficiente");

          llegada->sujeto = suj;
          strcpy(llegada->predicado, predicado);
          llegada->objeto = obj;

          llegada->costo = 0;
          llegada->longitud = 0;
          llegada->costoTotal = 0;
          llegada->bk = 0;
          llegada->inicio = NULL;
          llegada->fin = NULL;


          llegada->sig = obj->llegada;
          obj->llegada = llegada;
```

```
    }
    printf("Despues de ingresar la tripleta (dentro de la funcion)\n\n");
    printf("suj->label: %s\n", suj->label);
    printf("suj->salida->predicado: %s\n", suj->salida->predicado);
    printf("obj->label: %s\n\n", obj->label);
    printf(" - fin tripleta - \n\n");
}

void datos_nodo(tnodo g)
{
    printf("Datos del nodo\n");
    printf("Nodo N: %i\n", g->nodo);
    printf("Label: %s\n\n", g->label);
}

void datos_triple(ttriple t)
{
    printf("Datos del arco\n");
    datos_nodo(t->sujeto);
    printf("Predicado: %s\n\n", t->predicado);
    datos_nodo(t->objeto);
}

tnodo first_nodo(tnodo g)
{
    return (g->sig);
}

ttriple first_salida(tnodo n, tnodo g)
{
    return(n->salida);
}

ttriple first_llegada(tnodo n, tnodo g)
{
    return(n->llegada);
}

ttriple next_salida(tnodo n, ttriple t, tnodo g)
{
    return(t->sig);
}

ttriple next_llegada(tnodo n, ttriple t, tnodo g)
{
    return(t->sig);
```

```
}

tnodo next_nodo(tnodo n, tnodo g)
{
   return(n->sig);
}

tnodo nodo_suj(ttriple t, tnodo g)
{
   return(t->sujeto);
}

tnodo nodo_obj(ttriple t, tnodo g)
{
   return(t->objeto);
}

tlabel label_triple(ttriple t, tnodo g)
{
   return(t->predicado);
}

void mostrar_grafo(tnodo g)
{
   tnodo n;
   ttriple s,l;
   int b;
   n=first_nodo(g);

   while(n != NULL)
   {
      s = first_salida(n,g);

      printf("\n************************\n");
      printf("   Nodo N:%i - %s   \n", n->nodo, n->label);
      printf("************************\n");

      printf(" - ARCOS DE SALIDA - \n\n");

      b=0;
      while(s!=NULL)
      {
         printf("| %s | -> %s -> | %s |\n",s->sujeto->label, label_triple(s,n), s->objeto->label);
         s=next_salida(n,s,g);
         b=1;
      }if(b)
```

```
        printf("El nodo N:%i ( %s ) no tiene mas arcos de salida\n\n",n->nodo, n->label);
        else printf("El nodo N:%i ( %s ) no tiene arcos de salida\n\n",n->nodo, n->label);

        printf("\n - ARCOS DE LLEGADA - \n\n");

        l = first_llegada(n,g);

        b=0;
        while(l!=NULL)
        {
           printf("| %s | -> %s -> | %s |\n", l->sujeto->label, label_triple(l,n), l->objeto->label);
           l=next_llegada(n,l,g);
           b=1;
        }if(b)
        printf("El nodo N:%i ( %s ) no tiene mas arcos de llegada\n\n",n->nodo, n->label);
        else printf("El nodo N:%i ( %s ) no tiene arcos de llegada\n\n",n->nodo, n->label);

        n=next_nodo(n,g);
    }
}

int get_label_type(tlabel label)
{
    int i=0;
    if(label == NULL)
        return -1;
        else if(label[0]=='h' & label[1]=='t' & label[2]=='t' & label[3]=='p')
                return 1; //ES URI
                else return 2; //ES UNA PALABRA
}

double sim_string(tlabel s1, tlabel s2)    // B. (a) - String similarity between Labels
{
    int t1, t2, lav;
    double ss;
// Calcula tamanios strings
    t1=strlen(s1); t2=strlen(s2);

    printf("\nA: %s\nB: %s\n", s1, s2);
    printf("t1: %i\nt2: %i\n", t1, t2);

    lav = Levenshtein(s1,t1,s2,t2);

    printf("Lav: %i\n", lav);

    if ( lav == (-1))
```

```
        return -1; //ERROR
     else ss = 1 - ((double)lav / (maximo(t1,t2)));

     printf("sim_string: %f\n\n", ss);

    return ss;
}

int Levenshtein(tlabel s1, int t1, tlabel s2, int t2)
{   int i,j,*m,costo,res,ancho;

// Verifica que exista algo que comparar
    if (t1==0) return(t2);
    if (t2==0) return(t1);
    ancho=t1+1;

// Reserva matriz con malloc                    m[i,j] = m[j*ancho+i] !!
    m=(int*)malloc(sizeof(int)*(t1+1)*(t2+1));
    if (m==NULL) return(-1); // ERROR!!

// Rellena primera fila y primera columna
    for (i=0;i<=t1;i++) m[i]=i;
    for (j=0;j<=t2;j++) m[j*ancho]=j;

// Recorremos resto de la matriz llenando pesos
    for (i=1;i<=t1;i++) for (j=1;j<=t2;j++)
       { if (s1[i-1]==s2[j-1]) costo=0; else costo=1;
         m[j*ancho+i]=Minimo(Minimo(m[j*ancho+i-1]+1,     // Eliminacion
                     m[(j-1)*ancho+i]+1),             // Insercion
                     m[(j-1)*ancho+i-1]+costo); }      // Sustitucion

// Devolvemos esquina final de la matriz
    res=m[t2*ancho+t1];
    free(m);
    return(res);
}


int Minimo(int a, int b)
{
    if(a<b)
        return a;
    else return b;

}
```

```
int maximo(int a, int b)
{
   if(a>b)
      return a;
   else return b;
}

/*** FUNCIONES LPS ***/

// COSTO ASCII TRIPLETA

int triple_ascii(ttriple t)
{
   int a = 0;
   return a = ascii(t->sujeto->label) + ascii(t->predicado) + ascii(t->objeto->label);

}

// COSTO ASCII LABEL

int ascii(tlabel t)
{

   int i, a = 0;
   for(i=0; i<strlen(t); i++)
   {
      a = a + t[i];
   }

   return a;
}
```