Universidad del Bío-Bío, Chile

Facultad de Ciencias Empresariales

Departamento de Sistemas de Información

Desarrollo de una aplicación que computa consultas de agregación sobre Data Warehouses almacenados en la estructura compacta k^2-treap

Proyecto de Título presentado por Carlos Felipe Aburto González
para obtener el título de Ingeniero Civil en Informática
Dirigido por Dra. Mónica Caniupán Marileo

2017

Dedicatoria

A mi Madre.

Resumen

Una estructura de datos compacta es una estructura de datos modificada para ocupar poco espacio manteniendo su funcionalidad, es decir, permitiendo manipular los datos almacenados en ella de forma directa. Una estructura compacta permite mejorar el rendimiento a la hora de procesar datos debido a que ésta se puede localizar en los altos niveles de la jerarquía de memoria principal.

En este Proyecto de Título se implementa un sistema que computa consultas de agregación considerando la función `SUM` sobre un Data Warehouse (DW) representado en estructuras compactas. Un DW es un almacén de datos orientado a un determinado ámbito, integrado, no volátil y variable en el tiempo, el cual tiene por finalidad apoyar la toma de decisiones en las organizaciones, permitiendo generar reportes de interés y análisis de datos. Estos almacenes pueden alcanzar grandes volúmenes de datos, lo que va en desmedro de la eficiencia en el procesamiento OLAP (On-Line Analytic Processing) que soportan los DWs, el cual básicamente consiste en la generación de consultas de agregación (cubos de datos) con distintos niveles de granularidad. Un DW se organiza mediante el modelo multidimensional, considerando dimensiones y hechos. Las *dimensiones* son un concepto abstracto que permiten dar contexto a los *hechos*, que corresponden a datos cuantitativos asociados a diferentes dimensiones. Como ilustración, los hechos pueden ser las *ventas* de productos por tiendas, donde las dimensiones son *tiendas* y *productos*. Las dimensiones se organizan en estructuras jerárquicas de niveles los cuales contienen elementos. La jerarquía de una dimensión facilita la navegación y agrupación de datos.

En este proyecto los elementos de los niveles en las dimensiones se representan por medio de tablas de una columna, las relaciones entre elementos de los niveles de una dimensión (jerarquía) se representan por medio de estructuras compactas llamadas

bitmaps, y los cubos de datos (agregaciones a diferentes niveles de granularidad) se representan en la estructura compacta llamada k^2-treap. La representación del DW en estructuras compactas permite ahorrar espacio de almacenamiento en memoria principal y mejoras considerables en el procesamiento de consultas de agregación (navegación desde cubos bases), comparado con utilizar un Sistema de Gestión de Bases de Datos (SGBD) como PostgresQL.

Palabras Claves — Data Warehouses, Cubos de Datos, Estructuras Compactas, Memoria Principal.

Agradecimientos

A mi profesora guía, Mónica Caniupán, por el respaldo entregado en cada una de las etapas del proyecto de titulación.

A mis padres que nunca dejaron de creer en mi y en mis capacidades.

Este proyecto de título fue parcialmente financiado por el proyecto "Estructuras de Datos Compactas para procesar eficientemente datos espaciales y espacio-temporales en el contexto de Big-Data". Proyecto Exploratorio, Programa Ingeniería 2030, UBB, código 1638.

Índice general

Índice de figuras

Índice de Algoritmos

Índice de tablas

Capítulo 1

Introducción

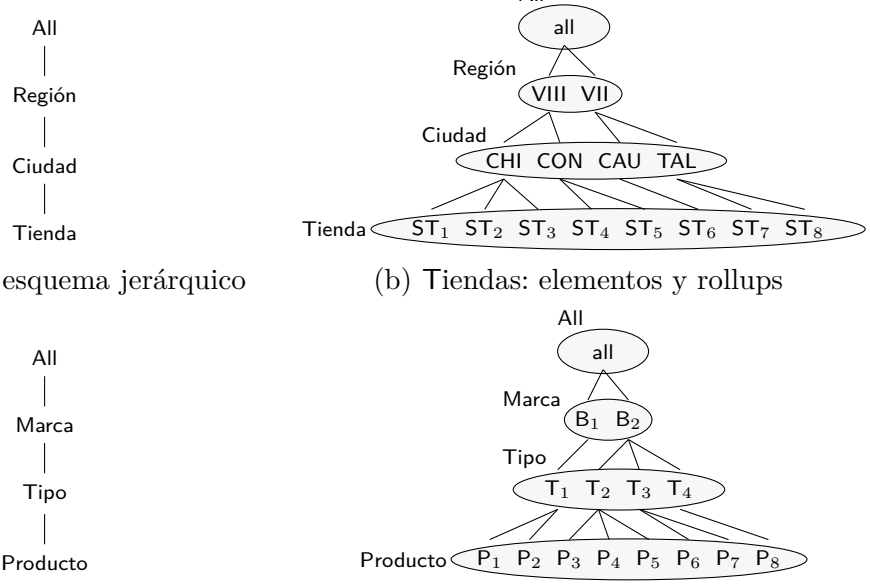
En este capítulo se presenta la motivación del proyecto de título, el planteamiento del problema a abordar y finalmente se describe la organización del resto del documento.

1.1. Motivación

En este proyecto trabajamos con repositorios de datos históricos que son utilizados por las organizaciones para apoyar la toma de decisiones. Estos repositorios se denominan Data Warehouses (DWs) (**?**). Un DW es un almacén que integra datos desde numerosas fuentes de datos, los organiza de acuerdo a *dimensiones* y *hechos*, los que permiten la generación de reportes para la toma de decisiones. Un DW puede contener muchas dimensiones y por eso son comúnmente conocidos como bases datos multidimensionales (**??**). Dado el carácter de repositorio histórico, un DW puede almacenar altas magnitudes de datos (por ejemplo varios Terabytes).

El Ejemplo 1.1 muestra una ilustración de un DW para una empresa que vende productos (dimensión) en diversas tiendas (dimensión).

Ejemplo 1.1 La Figura 1.1 representa la organización de un DW con dimensiones Tiendas y Productos, ambas dimensiones son lineales porque para cada una de ellas existe solo un camino desde el nivel inferior de la *jerarquía* hasta el nivel superior denominado ALL. La dimensión Tiendas esta compuesta por cuatro niveles (también conocidos como categorías), incluyendo el nivel ALL, representados en la Figura



(a) Tiendas: esquema jerárquico

(b) Tiendas: elementos y rollups

(c) Productos: esquema jerárquico

(d) Productos: elementos y rollups

Figura 1.1: Dimensiones Tiendas y Productos

1.1(a). La Figura 1.1(c) representa la dimensión Productos. Los elementos de cada dimensión se encuentran relacionados con los elementos del nivel inmediatamente superior (relación conocida como rollup). La Figuras 1.1(b) muestra los elementos y relaciones roolup para la dimensión Tiendas y la Figura 1.1(d) muestra los elementos y relaciones rollup para la dimensión Productos. □

Los hechos son datos cuantitativos relacionados con las dimensiones, por ejemplo, para el DW del Ejemplo 1.1 los hechos podrían ser la cantidad de ventas de productos por tienda. Al momento de analizar los hechos se pueden utilizar diferentes criterios haciendo uso de la organización de las dimensiones (jerarquías de niveles), logrando entregar respuestas a consultas de agregación tales como: *entregar el total de productos vendidos agrupados por marca y ciudad*. Generalmente, los resultados de las consultas se visualizan en cubos de datos (**?**). Los cubos de datos son representaciones específicas y segmentadas del data warehouse, y por medio de operaciones rollup, drill-down, slide, and dice se puede navegar por el cubo (**??**). El Ejemplo 1.2 muestra un cubo de datos para el DW de la Figura 1.1.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
ST_1	1	1	1	1	2	1	3	1
ST_2	1	2	2	2	1	2	2	1
ST_3	1	0	2	2	1	2	2	1
ST_4	2	2	0	1	0	1	0	4
ST_5	1	2	1	2	1	2	2	3
ST_6	1	2	3	2	1	2	1	0
ST_7	0	3	0	4	1	2	0	1
ST_8	1	2	3	1	0	0	1	2

(a) Ventas por Tienda y Producto

	T_1	T_2	T_3	T_4
CHI	6	14	12	3
CON	7	5	5	7
CAU	3	6	3	0
TAL	6	9	3	3

(b) Ventas por Ciudad y Tipo

Figura 1.2: Cubo de datos para el DW con las dimensiones Tiendas y Productos

Ejemplo 1.2 La Figura 1.2(a) representa un cubo de datos organizados por las dimensiones Tiendas y Productos, donde se muestran las unidades vendidas por cada producto en la respectiva tienda. Este cubo de datos es un cubo base porque considera los niveles inferiores de cada una de las dimensiones que lo componen, por lo tanto, vía las relaciones rollup de cada una de las dimensiones se pueden generar nuevos cubos de datos o agregaciones de datos a distintos niveles de granularidad. Como ilustración, la Figura 1.2(b) muestra el resultado de una consulta de agregación que obtiene la cantidad de productos vendidos agrupados por ciudad y tipo. Para obtener este cubo se utiliza la relación rollup entre Producto y Tipo que contiene los elementos {(P_1, T_1), (P_2, T_1), (P_3,T_2), (P_4,T_2), (P_5, T_2), (P_6,T_3), (P_7, T_3), (P_8, T_4)} de la dimensión Productos y la relación rollup entre los niveles Tienda y Ciudad de la dimensión Tiendas que contiene los elementos {(ST_1, CHI), (ST_2, CHI), (ST_3, CHI), (ST_4, CON), (ST_5, CON), (ST_6, CAU), (ST_7,TAL), (ST_8, TAL)}. □

Para realizar consultas de agregación se requiere navegar a través de las jerarquías de las dimensiones que componen el DW. Teniendo en cuenta lo grande que pueden llegar a ser los DW, nace la necesidad de buscar la forma de realizar consultas de agregación de forma eficiente sin tener que recurrir a prácticas como crear datos pre-computados o utilizar índices sobre tablas de datos pre-computados para mejorar el acceso a los datos (**???**). Otro acercamiento en un intento de disminuir el volumen de los datos es comprimirlos (**??????**), pero se pierde precisión al momento de entregar las respuestas a las consultas de agregación.

Normalmente la manera de realizar consultas a grandes conjuntos de datos es traer los los a memoria principal desde la memoria secundaria, implementando algunos índices para mejorar el acceso a memoria secundaria. Sin embargo, ha tomado relevancia, el tratar de utilizar al máximo la capacidad de almacenamiento de la memoria principal evitando los accesos a disco y de este forma lograr mayor eficiencia en el procesamiento de consultas debido a la alta velocidad de esta memoria (**???**). Ahora si bien la tecnología del hardware ha mejorado y las memorias principales se vuelven cada día más poderosas, ¿por qué no se almacena completamente un DW en memoria principal? La respuesta a esto es que la memoria principal aún sigue siendo más cara que la secundaria y por ende el acceso a memorias principales de mayor tamaño sigue siendo difícil. Además, es muy probable que un DW no alcance a ser completamente almacenado en memoria principal dado su gran tamaño.

1.2. Planteamiento del problema

Existen muchos trabajos que tratan de optimizar el procesamiento de consultas sobre DWs a través de la materialización de cubos de datos (forma típica de analizar los hechos de acuerdo a diferentes niveles de granularidad de las dimensiones, ver Ejemplo 1.2), tales como los presentados en (**??????????**)

Existen trabajos que permiten almacenar grandes conjuntos de datos en memoria principal de manera compacta, no comprimida y poder realizar consultas sobre la forma compacta. Esto se logra a través de la implementación de las *estructuras de datos compactas* (**?**). Estas estructuras son estructuras especiales que ocupan poca cantidad de memoria y permiten aprovechar al máximo el potencial de la memoria principal, logrando un cómputo eficiente de consultas y a la vez reducir la cantidad de memoria utilizada por los datos. Las estructuras compactas han sido utilizadas en diversos escenarios, por ejemplo, en (**????**) se utilizan para representar grafos de la Web, en (**?**) para representar y procesar grandes documentos de texto, entre otros escenarios.

En este proyecto utilizaremos la estructura compacta k^2-treap que se introduce en (**?**) para computar consultas de tipo top-k dentro de un rango de una matriz, por ejemplo, *encontrar el producto más vendido en las tiendas*. Sin embargo, es posible

utilizarla para computar consultas de agregación con la función `SUM` sumando los elementos contenidos en el rango entregado por la función top-k implementada en la estructura.

Además, proponemos la implementación de una aplicación en C++ que hace uso de la estructura k^2-treap para representar un cubo de datos en memoria principal y permita realizar diversas agregaciones. También utilizamos bitmaps para representar las relaciones rollup de los elementos de las dimensiones del DW, y éstas últimas son representadas en tablas de una columna.

Las contribuciones de este Proyecto de Título son las siguientes:

- Generación de algoritmos para computar consultas de agregación (utilizando la función `SUM`) sobre un cubo de datos representado en la estructura de datos compacta k^2-treap.
- Generación de bitmaps y tablas de una columna para representar las relaciones rollup entre elementos de los niveles de una dimensión y los elementos de la dimensión, respectivamente.
- Experimentación de los algoritmos sobre cubos de datos sintéticos para demostrar el ahorro de espacio y mejoras en los tiempos de respuesta a las consultas comparando nuestros algoritmos con la ejecución y almacenamiento del DW en un Sistema de Gestión de Base de Datos (SGBDs), en particular PostgreSQL[1].

1.3. Organización del documento

El resto del documento está organizado en los siguientes capítulos. El Capítulo 2 presenta los objetivos, alcances y limitaciones del proyecto. El Capítulo 3 se presentan conceptos preliminares necesarios para el desarrollo del proyecto, tales como la definición de Data Warehouse y de la estructura compacta k^2-treap, abordando su generación a partir de una matriz y navegación. El Capítulo 4 presenta la forma de representar un DW en estructuras compactas y los algoritmos desarrollados para realizar las funciones fundamentales de la aplicación. El Capítulo 5 presenta la aplicación desarrollada en este proyecto, mostrando su arquitectura, las etapas de

[1] https://www.postgresql.org/

su implementación y su interfaz. En el Capítulo 6 se presentan los resultados de la experimentación realizada para medir la eficiencia en términos de ahorro de espacio y tiempo de ejecución de consultas al implementar un DW sobre estructuras compactas. Finalmente en el Capítulo 7 se presentan las conclusiones obtenidas como resultado del proyecto de título y se describen algunas ideas para trabajo futuro.

Capítulo 2

Objetivos

2.1. Objetivos Generales

Implementación de una aplicación que realice consultas de agregación sobre cubos de datos almacenados en la estructura compacta k^2-treap.

2.2. Objetivos Específicos

1. Implementar o adaptar el código existente de la estructura compacta k^2-treap en C++.

2. Implementar las clases o funciones necesarias para interactuar con la estructura de datos compacta k^2-treap.

3. Diseñar e implementar tablas de una columna para almacenar los elementos de los niveles de las dimensiones, representar las relaciones rollup en bitmaps, y representar cubos de datos en la estructura compacta de datos k^2-treap.

4. Implementar algoritmos para realizar consultas de agregación utilizando la función de agregación `SUM` sobre DWs representados en estructuras compactas.

5. Realizar experimentación y comparación de los algoritmos propuestos con diferentes tamaños de cubos de datos, considerando el SGBD PostgreSQL.

2.3. Alcances y límites de la aplicación

En este proyecto de titulo sólo se consideran dimensiones con jerarquías lineales, es decir, aquellas donde existe un único camino desde el nivel inferior al nivel superior All (como las mostradas en la Figura 1.1). También se limita el DW a dos dimensiones. Además, solo se considera la función de agregación `SUM`.

Por último, para la experimentación se utilizan Data Warehouses ficticios.

Capítulo 3

Conceptos preliminares

En este capítulo se presenta el concepto de DW y la estructura compacta k^2-treap.

3.1. Data Warehouses

Un DW es un almacén de datos mediante el cual una organización o empresa almacena todos aquellos datos necesarios para realizar procesamiento OLAP. La forma de organizar los datos en el DW es a través de *dimensiones* y *hechos* (**?**). Las *dimensiones* representan de manera abstracta áreas del negocio sobre las cuales se analizan los *hechos*, que corresponden a valores numéricos asociados a las dimensiones. Como la información necesita disponer de varios niveles de granularidad las dimensiones están organizadas por jerarquías de niveles, cada una con elementos. Por ejemplo, consideremos la dimensión Tiempo con jerarquía: fecha $\rightarrow$ día $\rightarrow$ semana $\rightarrow$ mes $\rightarrow$ trimestre $\rightarrow$ año. Cada nivel de jerarquía tiene una relación hijo-padre con otro nivel llamada *rollup*, es por esto que cada elemento de un nivel se relaciona con otro elemento de otro nivel solo si sus niveles de jerarquía también se relacionan en el esquema jerárquico de la respectiva dimensión. El siguiente ejemplo muestra un DW para una empresa de ventas de productos en varias sucursales de un país.

Ejemplo 3.1 La Figura 1.1(a) representa el esquema jerárquico de la dimensión Tiendas y la Figura 1.1(b) muestra los elementos de cada nivel y las relaciones rollup entre ellos. Esta dimensión posee cuatro niveles de jerarquía: Tienda, Ciudad, Región, y el nivel superior All (que está presente en todas las dimensiones). La Figura 1.1(c)

muestra el esquema jerárquico para la dimensión Productos y la Figura 1.1(d) muestra una instancia de la dimensión Productos (elementos y rollups). □

Al momento de utilizar el DW se necesita analizar los datos (hechos) a distintos niveles de abstracción. Un cubo de datos permite representar los hechos de acuerdo a diferentes niveles de las dimensiones. En este proyecto nos focalizaremos en cubos de datos considerando solo dos dimensiones. Un cubo de datos esta compuesto por celdas que pueden o no contener elementos numéricos, los cuales corresponden a los hechos (también conocidos como medidas). A partir del *cubo de datos base* (que se compone de hechos relacionados a los niveles inferiores de dos dimensiones) podemos obtener distintos cubos dependiendo de los niveles de jerarquía solicitados en la consulta de agregación. En la Figura 1.2(b) podemos observar el cubo generado con por el nivel Ciudad de la dimensión Tiendas y el nivel Tipo de la dimensión Productos.

3.2. k^2-treap

El k^2-treap (**?**) es una estructura de datos compacta que puede almacenar una matriz y a la vez permite realizar operaciones directamente sobre los datos compactados. Fue diseñada originalmente para responder consultas del tipo top-k, las cuales consisten en obtener k mayores valores contenidos en un rango de la matriz (representada en el k^2-treap), como por ejemplo obtener el tipo de productos con mayores niveles de ventas. Las consultas de tipo top-k pueden realizarse sobre toda la matriz o realizarlas estableciendo un rango. La estructura k^2-treap está basada en la estructura compacta k^2-tree (**?**) y en la estructura compacta treap (**?**).

3.2.1. Generación de un k^2-treap

Dada una matriz $M[n \times n]$, en la que sus celdas pueden tener valores positivos o ceros, se realiza la búsqueda del valor más alto contenida en ella, pasando éste y su coordenada a ser la raíz de un árbol, este valor se borra de la matriz. Una vez asignada la raíz del árbol, la matriz es dividida en k^2 submatrices y en cada una de ellas se busca nuevamente el elemento de mayor valor pasando a ser hijos del nodo raíz, así de forma recursiva se realiza esta acción hasta haber almacenado cada uno

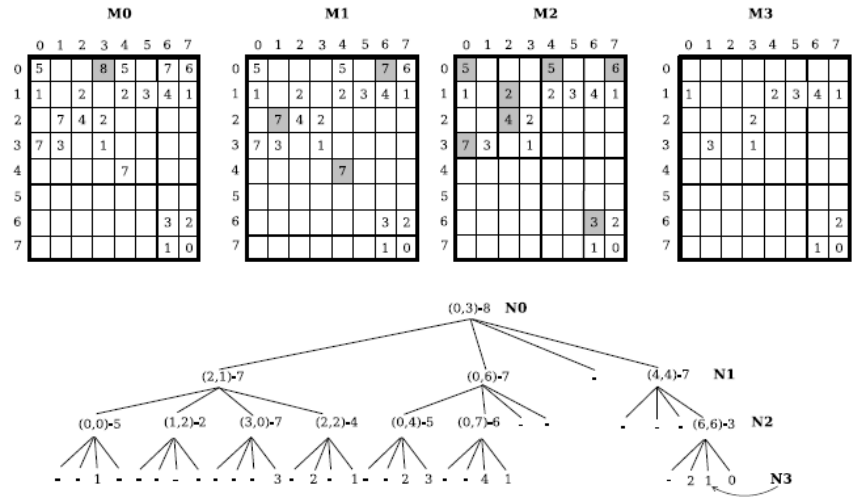


Figura 3.1: Construcción del k^2-treap para la matriz del Ejemplo 3.2 (**?**)

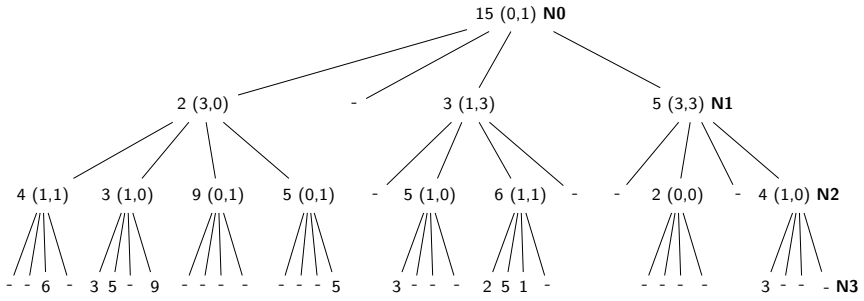


Figura 3.2: Recodificación del árbol de la Figura 3.1

de los elementos distintos de cero en la estructura de árbol. El siguiente ejemplo ilustra la creación de un k^2-treap y fue presentado en (**?**).

Ejemplo 3.2 Consideremos la matriz $M0[8 \times 8]$ de la Figura 3.1. El valor más alto es 8 y se encuentra en la coordenada $(0, 3)$, este valor y su coordenada pasa a ser la raíz del árbol y es eliminado de la matriz original como podemos observar en la matriz $M1$. Ahora la matriz $M1$es dividida en 4 submatrices, considerando k^2 con $k = 2$, y en cada una de las submatrices se busca nuevamente el valor mas alto y su coordenada, las que son asignadas como hijos del nodo raíz. Este proceso se realiza de forma recursiva hasta cargar cada valor en el árbol. □

Para el caso de las celdas que no poseen un valor, estas se representan con el símbolo "-"en el árbol. Una vez terminado el proceso y con la totalidad de los valores almacenados en el árbol, éste se re-codifica con el objetivo de utilizar menos espacio, haciéndolo más eficiente en memoria. Las nuevas coordenadas corresponden a las posiciones en las submatrices, y los valores en cada nodo se representan como la diferencia con respecto al valor del nodo padre. La Figura 3.2 muestra la re-codificación para el árbol de la Figura 3.1. De esta forma el árbol construido puede ser representado en la estructura compacta k^2-treap compuesta por:

- Un k^2-tree para el cual se genera un arreglo de bits T (bitmap) que entrega la información de la existencia o no de elementos en el árbol.
- Arreglos que guardan las coordenadas re-codificadas de los elementos almacenados en el árbol. En la Figura 3.3 corresponde a los arreglos *coord*[0], *coord*[1] y *coord*[2].
- Un arreglo que almacena los valores numéricos del árbol modificado (arreglo *values* en Figura 3.3).
- Un arreglo que indica la posición de inicio de cada nivel en el arreglo de valores *values* (Arreglo $first$ en Figura 3.3) .

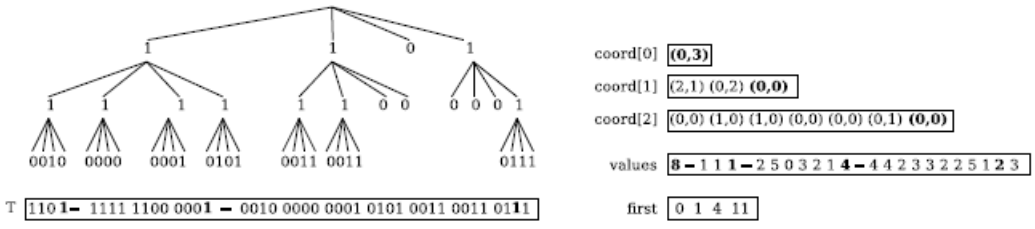


Figura 3.3: k^2-treap para la matriz del cubo de datos del Ejemplo 3.2

3.2.2. Navegación en el k^2-tree

Supongamos que deseamos obtener la coordenada $C(x, y)$ en el k^2-treap de la Figura 3.3. La navegación parte por la raíz del k^2-tree, que corresponde al nodo

con coordenadas (x_0, y_0) y valor $v = values[0]$. Si $C = (x_0, y_0)$, entonces el valor v debe ser retornado, de lo contrario, se necesita encontrar el cuadrante donde la celda se localiza en el k^2-tree. Sea p la posición de un nodo en el arreglo T (que almacena los valores del k^2-tree), si $T[p] = 0$ la correspondiente sub-matriz está vacía y por lo tanto el algoritmo no prosigue con la búsqueda y retorna. De lo contrario, se necesita encontrar las coordenadas y el valor del nodo. Para ello se computa la función $r = rank_1(T, p)$ que retorna el número de ocurrencias de 1 en T hasta la posición p, el valor del nodo es $v = values[r]$ y su coordenada corresponde a $coord[l][r - first[l]]$, donde l es el nivel actual en el árbol. El valor v tiene que volver a ser reconstruido, puesto que el arreglo `values` almacena valores de manera diferencial. Por lo tanto, $v_1 = v - values[r]$. Supongamos que la coordenada en $coord[l][r - first[l]]$ corresponde a (x_1, y_1), si $C = (x_1, y_1)$, entonces se retorna el valor v_1, en caso contrario, se vuelve a buscar el nodo en el árbol.

3.2.3. Cómputo de consultas top-k sobre un k^2-treap

El proceso para obtener las top-k respuestas a una consulta top-k sobre un rango de la matriz $Q[x_1, x_2] \times [y_1, y_2]$ parte en la raíz del árbol, implementando una cola de prioridad que almacena valores máximos. El primer elemento de esta cola corresponde al valor en la la raíz del árbol. Luego de manera iterativa el proceso extrae el primer elemento de la cola, si la coordenada del valor cae en el rango definido por Q, se despliega una respuesta. Todos los hijos del nodo extraído cuyas coordenadas intersecten el rango Q se insertan en la cola de prioridad. El proceso continua hasta que se obtienen las k respuestas. El siguiente ejemplo ilustra el proceso.

Ejemplo 3.3 Considere la matriz $M0[8 \times 8]$ de la Figure 3.1, y la consulta "obtener los tres top valores en el rango $[4, 4], [7, 7]$", esto es, en el cuarto cuadrante de $M0$. El primero elemento de la cola de prioridad es la raíz del árbol, con coordenada $(0, 3)$ y valor 8, dado que esta coordenada no cae dentro del rango de la consulta, el valor es descartado como respuesta. Luego, todos los hijos de la raíz que intersectan el rango de la consulta se insertan en la cola de prioridad, esto es el cuarto hijo en el nivel $N1$ de la Figura 3.1, con coordenadas $(4, 4)$ y valor 7, que corresponde a la primera respuesta. Luego, los hijos no vacíos del nodo se adhieren a la cola, esto es, el nodo con coordenada $(6, 6)$ y valor 3, que corresponde a la segunda respuesta. Finalmente,

los hijos no vacíos de éste nodo se insertan en la cola de prioridad, esto corresponde a los nodos con coordenadas $(6, 7)$, $(7, 6)$ y $(7, 7)$, que corresponden a hojas del árbol, con valores $2, 1$ y 0 respectivamente. Por lo tanto, se genera la tercera respuesta con valor 2. En resumen, las top-3 respuestas son $\{7, 3, 2\}$. □

Capítulo 4

Algoritmos para representar DWs en Estructuras Compactas

En este capítulo se muestra como se realiza la representación de un DW en estructuras compactas. Los cubos de datos se representan en un k^2-treap, los elementos de los niveles de las dimensiones en tablas de una columna, y las relaciones rollup de los elementos de las dimensiones en bitmaps. También se presentan los algoritmos para computar consultas de agregación con la función `SUM`. Esta forma de representación y los algoritmos fueron propuestos en (**?**).

4.1. Representación de Dimensiones

Los elementos de los niveles de las dimensiones se almacenan en tablas de una columna, exceptuando el nivel inferior, que se encuentra en el cubo de datos base, y el nivel superior All que posee un único elemento all.

Ejemplo 4.1 Considere las dimensiones Tienda y Productos del DW en el Ejemplo 1.1 con los esquemas jerárquicos de las Figuras 1.1(a) y 1.1(c), respectivamente. La Figura 4.1 muestra las tablas de una columna que se necesitan para estas dimensiones. □

Al momento de representar las relaciones rollup, es necesario recurrir a secuencias de bits, los cuales permiten determinar cuantos elementos de un determinado nivel

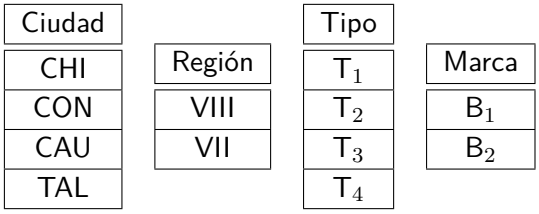


Figura 4.1: Tablas de una columna para los niveles de las dimensiones Tiendas y Productos del Ejemplo 1.1

hacen rollup o se relacionan con un elemento del nivel superior. Se debe crear una secuencia de bits o bitmap para cada nivel perteneciente a la dimensión, exceptuando el nivel superior All. La primera posición del bitmap corresponde al primer elemento del nivel que representa el bitmap y se le asigna un 1 entendiendo que este elemento hace rollup al primer elemento del nivel inmediatamente superior. Posteriormente se asignan ceros a cada elemento que hace rollup al mismo elemento hasta encontrar un elemento que haga rollup al siguiente elemento del nivel superior, asignando un 1 y siguiendo el mismo procedimiento para el resto de los elementos.

Ejemplo 4.2 Consideremos el nivel Tienda y el nivel Ciudad de la dimension Tiendas, el bitmap que representa las relaciones rollup entre ambos niveles se construye asignando un 1 en la primera posición, ya que la primera tienda pertenece a la ciudad CHI, posteriormente asignamos un 0, haciendo referencia a que la segunda tienda también pertenece a la ciudad CHI, repetimos este proceso hasta encontrar la primera tienda que pertenece a la siguiente ciudad del nivel Ciudad asignando 1, lo que indica que esta tienda pertenece a la ciudad CON y de manera recursiva se sigue el proceso hasta computar todas las relaciones rollup. La Figura 4.2 muestra los bitmaps que almacenan las relaciones rollup del DW en el Ejemplo 1.1. □

El Algoritmo 1 genera un bitmap para una relación rollup de la forma $R(n_1, n_2)$ donde n_1 es un nivel que se relaciona (rollup) con el nivel n_2 en una dimensión D. La relación rollup se lee desde el archivo RUp, el cual es asignado por completo a un string por medio de la función `contenidoFichero()`, la que realiza la concatenación de los elementos en el formato: elemento1+" "+elemento2+" "+elementoN. Posteriormente se realiza la separación de cada elemento contenido en el string element

R_1	
Tienda	Bitmap
ST_1	1
ST_2	0
ST_3	0
ST_4	1
ST_5	0
ST_6	1
ST_7	1
ST_8	0

R_2	
Ciudad	Bitmap
CHI	1
CON	0
CAU	1
TAL	0

(a) Bitmaps para la dimensión Tiendas

R_3	
Producto	Bitmap
P_1	1
P_2	0
P_3	1
P_4	0
P_5	0
P_6	1
P_7	0
P_8	1

R_4	
Tipo	Bitmap
T_1	1
T_2	1
T_3	0
T_4	0

(b) Bitmaps para la dimensión Productos

Figura 4.2: Bitmaps para almacenar las relaciones rollup del DW en el Ejemplo 1.1

con la función `strtok()`, la cual permite partir una cadena en subcadenas usando como separador el espacio. Para la primera posición del bitmap siempre se asigna un 1, posteriormente si el elemento hace rollup al mismo elemento que el anterior se asigna 0 en la siguiente posición y en el caso contrario (el elemento hace rollup al siguiente elemento del nivel superior (n_2), y por lo tanto, se asigna un 1 en el bitmap.

4.2. Representación de cubos de datos

Los cubos de datos de un DW se representan en un k^2-treap, cada fila de un cubo de datos base de dos dimensiones representa el nivel inferior de una dimensión, y la columna representa el nivel inferior de la segunda dimensión. El tamaño de la matriz que corresponde al cubo de datos está dado por la cantidad de elementos que posee el nivel inferior de cada dimensión. Como cada valor contenido en un cubo de datos de un DW representa una cantidad, solo existen valores positivos o ceros. Para el caso de los ceros, estos representan una ocurrencia nula de un hecho, por lo tanto, no se consideran al momento de la construcción del k^2-treap con la finalidad de optimizar el espacio de almacenamiento (si se almacenan en un SGBD).

Ejemplo 4.3 La Figura 4.3(a) muestra la matriz del cubo de datos original para las dimensiones Tiendas y Productos del Ejemplo 1.1. La Figura 4.3(b) corresponde a la misma matriz del cubo de datos sin los ceros. □

Algoritmo 1: Crear Bitmap

```
   input  : Fichero rollup RUp, cantidad de elementos de n_1
   output: Estructura Bitmap BM
 1 rollups←contenidoFichero(RUp);
 2 element←strtok(rollups," ");
 3 cont=0;
 4 while element != NULL do
 5     for i = 0; i < n × 2; i++ do
 6         if i == 1 then
 7             BM[cont]=1;
 8             cont++;
 9             elementAux=element;
10         if i mod 2 ≠ 0 and i ≠ 1 then
11             if element == elementAux then
12                 BM[cont]=0;
13                 cont++;
14             else
15                 BM[cont]=1;
16                 cont++;
17                 elemetAux=element;
18         element←strtok(NULL," ");
19 return BM;
```

Posteriormente el cubo de datos es cargado en el k^2-treap mediante el proceso descrito en la generación de la estructura k^2-treap. El siguiente ejemplo ilustra el proceso para el cubo de datos de la Figura 4.4(b).

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
ST_1	0	15	7	5	0	0	0	0
ST_2	3	9	10	1	0	0	0	0
ST_3	0	4	0	8	0	0	0	0
ST_4	13	0	0	3	0	0	0	0
ST_5	0	0	4	0	0	0	8	0
ST_6	0	0	7	12	0	0	0	0
ST_7	4	1	0	0	0	0	3	0
ST_8	5	6	0	0	0	0	7	10

(a) Ventas por Tienda y Producto

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
ST_1		15	7	5				
ST_2	3	9	10	1				
ST_3		4		8				
ST_4	13			3				
ST_5			4				8	
ST_6			7	12				
ST_7	4	1					3	
ST_8	5	6					7	10

(b) Ventas por Tienda y Producto, sin considerar ceros

Figura 4.3: Cubo de datos para el DW del Ejemplo 1.1

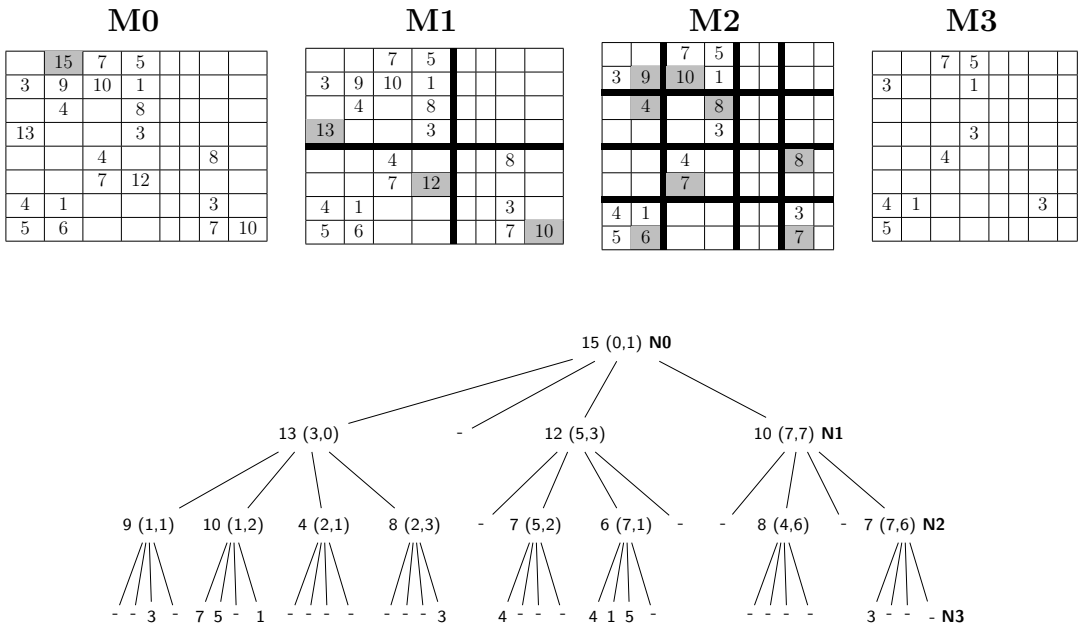


Figura 4.4: Construcción del k^2-treap para el cubo de datos de la Figura 4.3(b)

Ejemplo 4.4 Considere el cubo de datos $M0$ en la Figura 4.4 (matriz del cubo de datos en la Figura 4.3(b)), donde ya se realizó el descarte de los elementos iguales

a cero. Se procede a buscar el mayor valor para ser asignado a la raíz del árbol. Se puede observar que el valor es 15 y sus coordenadas son $(0, 1)$. El elemento es eliminado de la matriz $M0$ resultando la matriz $M1$. De esta forma la matriz es subdividida y el proceso se repite asignando los valores mas altos de cada submatriz a los nodos hijos del nodo raíz ($M2$ y $M3$ de la Figura 4.4). □

Cada valor asignado en los nodos del árbol de la Figura 4.4 es re-codificado en un valor mas pequeño obtenido como resultado de la diferencia respecto del valor en el respectivo nodo padre. Esto se observa en la Figura 4.5. Lo mismo se aplica a las coordenadas de cada nodo.

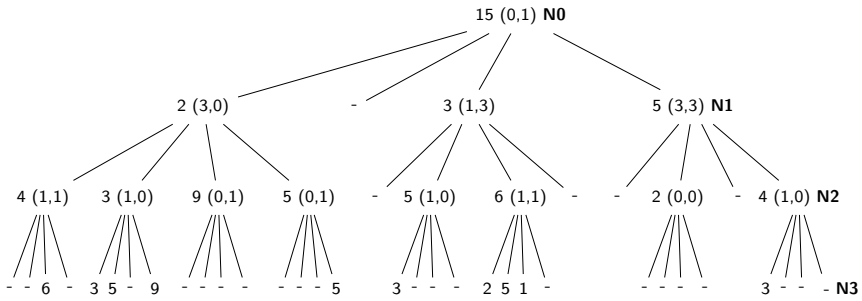


Figura 4.5: Re-codificación del árbol obtenido a partir del cubo de datos de la Figura 4.3(b

El bitmap T y los arreglos *coord*, *values* y $first$ son básicamente la estructura compacta, ya que a través de ellos se puede representar el cubo de datos de acuerdo a sus dimensiones y niveles.

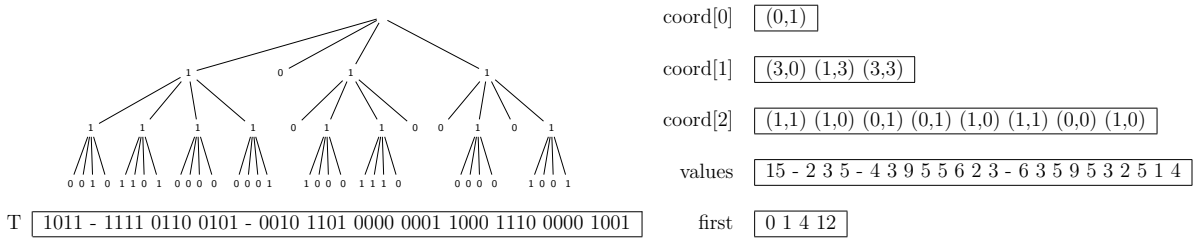


Figura 4.6: k^2-treap para el cubo de datos de la Figura 4.3(b)

4.3. Procesamiento de consultas de agregación sobre un cubo de datos almacenado en un k^2-treap

Las consultas sobre un cubo de datos almacenado en una estructura k^2-treap se deben realizar por medio de una selección por rango, permitiendo obtener los elementos contenidos en un rectángulo dado por una coordenada inicial y una coordenada final. Las consultas que se pueden computar con el k^2-treap son del tipo top-k, las cuales entregan los k valores mas altos en el rectángulo seleccionado. Para obtener las respuestas a una consulta de agregación con función `SUM` se utiliza la función top-k implementada en la estructura compacta k^2-treap, pero en vez de retornar los k valores, se colectan todos los valores del rango de la consulta y se suman. Para ello es fundamental obtener los niveles de las dimensiones involucradas en la consulta de agregación.

Para definir el rango de una consulta de agregación se debe recurrir a los niveles de jerarquía de las dimensiones que componen el cubo de datos y sus relaciones de rollup, ya que el rango esta dado por el nivel granularidad de la consulta. De esta forma el rango queda definido por $\{(x_1, y_1), (x_2, y_2)\}$, donde:

- x_1: Numero de la fila de la matriz donde comienza el nivel de jerarquía a consultar.
- x_2: Numero de la fila de la matriz donde termina el nivel de jerarquía a consultar.
- y_1: Numero de la columna de la matriz donde comienza el nivel de jerarquía a consultar.
- y_2: Numero de la fila de la matriz donde termina el nivel de jerarquía a consultar.

Ejemplo 4.5 Considere la consulta de agregación "Obtener las cantidades vendidas de Productos de Marca B_2 vendidas en cada Tienda de la Ciudad de Talca". La Figura 4.7 representa la matriz del cubo de datos con sus respectivos niveles de jerarquía,

donde cada una de las filas es una tienda y cada una de las columnas es un producto vendido en esas tiendas. La Figura 4.8 muestra el rango de selección donde se debe buscar la respuesta a la consulta, que corresponde a la suma de los valores en el rectángulo y cuyo valor es 20. □

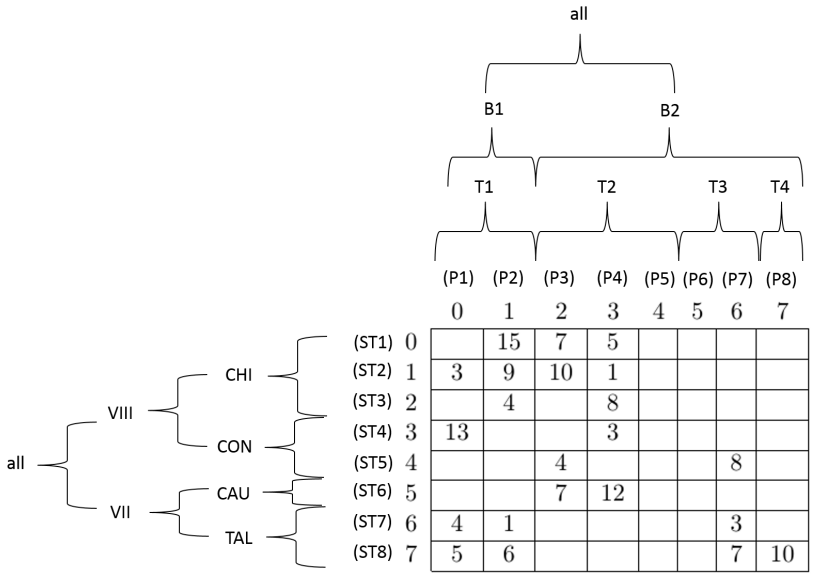
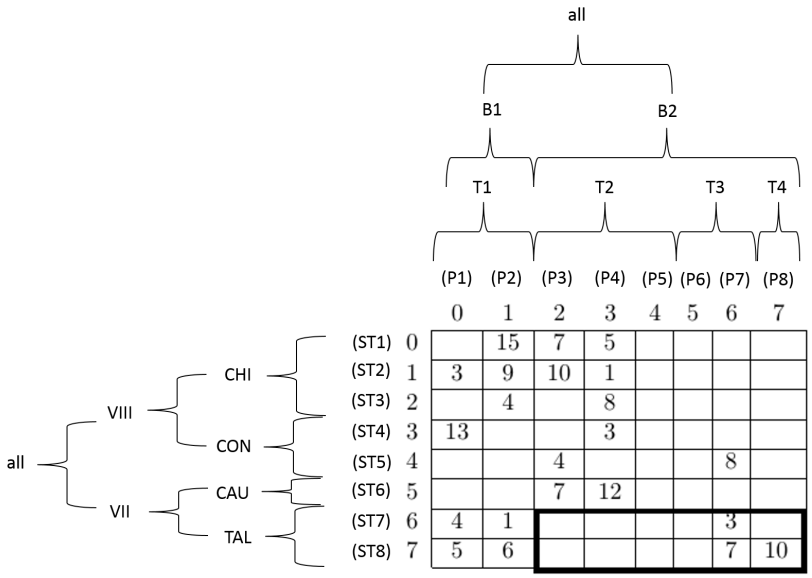


Figura 4.7: Jerarquías de las dimensiones del cubo de datos de la Figura 4.3



	0	1	2	3	4	5	6	7
(ST1) 0		15	7	5				
(ST2) 1	3	9	10	1				
(ST3) 2		4		8				
(ST4) 3	13			3				
(ST5) 4			4				8	
(ST6) 5			7	12				
(ST7) 6	4	1					3	
(ST8) 7	5	6					7	10

Figura 4.8: Rango para responder a consulta de agregación del Ejemplo 4.5

Para obtener el rango de la consulta de agregación se utilizan los *bitmaps*, permitiendo identificar a qué elemento del nivel superior hace rollup cada elemento de un determinado nivel.

Lo fundamental de los *bitmaps* al momento de representar las relaciones rollup es que permiten realizar las operaciones $select_1()$ y $rank_1()$. Para la operación $select_1(B, n)$, dado un *bitmap* B, permite saber en qué posición de B se encuentra el $n - ésimo$ 1, y la operación $rank_1(B, n)$, para un *bitmap* B, permite saber cuántos 1 hay hasta la $n - ésima$ posición de B.

Ejemplo 4.6 En la Figura 4.8 el rango para consulta: cantidades vendidas de Productos de Marca B_2 vendidas en cada Tienda de la Ciudad de Talca es obtenido realizando las operaciones $x_1 - select_1(R_x, 4)$, $x_2 = select_1(R_x, 5) - 1$, $y_1 = select_1(R_y, 2)$ e y_2: $select_1(R_y, 3) - 1$, teniendo a R_x y R_y como los bitmaps correspondientes a los niveles de agregación para la consulta realizada. □

El Algoritmo 2 (**?**) realiza la generación de la respuesta a la consulta de agregación, recibiendo como entrada un k^2-treap, los bitmaps correspondientes a los niveles involucrados en la consulta y los niveles de la consulta de agregación ($nivY$ y $nivX$).

Para obtener los valores del nivel superior realiza la suma de los elementos del nivel inferior contendidos en el rango definido por medio de la función *select*() aplicada a los bitmaps.

Algoritmo 2: Generar respuesta a consulta de agregación con función `SUM`

```
   input  : Estructura k²-treap, estructuras Bitmap BMpX y BMpY, tamaños
            dimensiones NDimX y NDimY
   output: Respuesta a consulta SUM
 1 for i=0; i<NDimX; i++ do
 2     for j=0; j<NDimY; j++ do
 3         x1=select1(BMpX, i );
 4         y1=select1(BMpY, i )-1;
 5         x2=select1(BMpX, i );
 6         y2=select1(BMpY, i )-1;
 7         SUM = 0;
 8         R − topk = topk(k²-treap, {x1,y1}, {x2,y2});
 9         foreach i in R-topk do
10             R=R+i.valor;
11         SUM = R
12 return SUM;
```

Capítulo 5

Main Memory DW

En este capítulo se presenta la arquitectura del sistema construido en C++, el cual se denomina *Main Memory DW*. También se presentan los pasos que se siguieron para la implementación y la interfaz.

5.1. Arquitectura de Main Memory DW

La Figura 5.1 muestra la arquitectura del sistema *Main Memory DW* cuyas entradas son: el nombre del fichero que contiene la matriz del cubo de datos, el nombre y la cantidad de niveles por cada una de las dos dimensiones del cubo de datos, los nombres de los archivos de las relaciones rollup y la consulta de agregación.

La salida generada por *Main Memory DW* es un cubo de datos en un reporte denominado "reporte.txt" con el resultado de la consulta de agregación solicitada por el usuario. Esta consulta puede ser realizada eligiendo los niveles de agregación, o realizarla para generar el reporte con los resultados para todos los niveles de jerarquía.

La arquitectura de *Main Memory DW* (Figura 5.1) consiste en:

1. Cargar Datos: Este módulo se encarga de realizar la carga de la información de la matriz del cubo de datos en la estructura compacta k^2-treap. El algoritmo que realiza la carga recibe la matriz del cubo de datos y la información necesaria para la organización de las dimensiones del mismo. La información de las dimensiones solicitada por *Main Memory DW* consiste en:

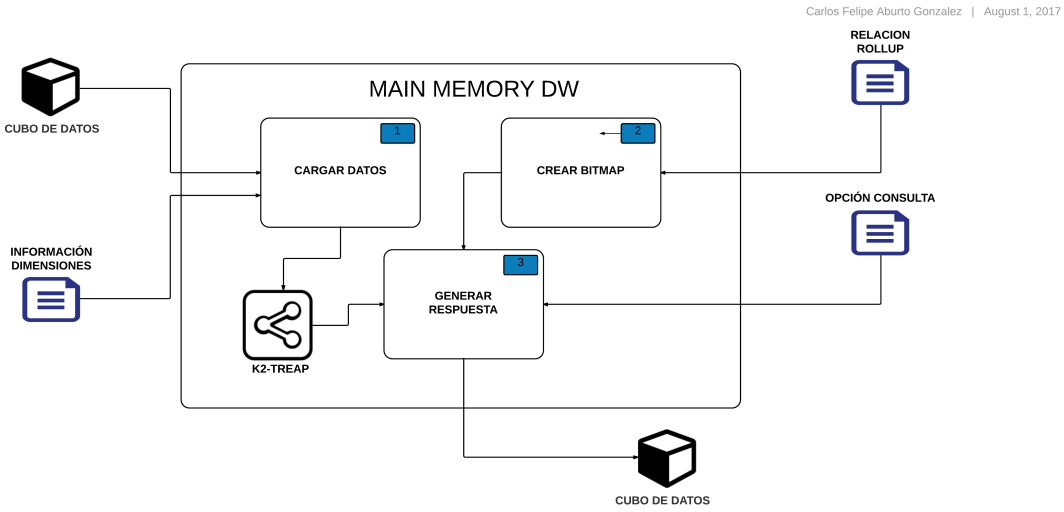


Figura 5.1: Arquitectura de la Aplicación *Main Memory DW*

- Nombre del archivo que contiene el cubo de datos en formato txt.
- Nombre de la dimensión del eje Y de la matriz que representa el cubo de datos.
- Cantidad de niveles de la dimensión Y.
- Nombre del archivo de relaciones rollup en formato txt (el sistema solicitará automáticamente la cantidad de archivos requeridos).
- Nombre de la dimensión del eje X de la matriz que representa el cubo de datos.
- Cantidad de niveles de la dimension X.
- Nombre del archivo de relaciones rollup de la dimensión X en formato txt (el sistema solicitará automáticamente la cantidad de archivos requeridos).

2. Crear Bitmap: Este módulo realiza la transformación de la información contenida en el archivo relación rollup a una representación en bits de las relaciones rollup para cada nivel en una estructura tipo bitmap.

3. Generar Respuesta: Módulo encargado de generar los reportes con la información de la matriz del cubo de datos. Recibe la estructura k^2-treap, los bitmaps creados en el módulo "Crear Bitmap" y la opción de consulta a realizar. Permite generar reportes por niveles específicos o para todos las combinaciones posibles entre niveles de las dos dimensiones.

5.2. Implementación

Para la construcción de *Main Memory DW*, la implementación se llevó a cabo empleando y adaptando metodologías agiles de desarrollo, debido que fue esta fue realizada solo por el alumno que presenta la tesis. Además, la implementación está realizada utilizando las estructuras implementadas en las librerías SDSL (Succinct Data Structure Library) para C++. Las etapas consideradas son:

- Análisis: Se recopiló información sobre la estructura compacta k^2-treap para lograr entender el manejo de los datos y el comportamiento de esta estructura. Se realizaron pruebas con la librería SDSL y sus ejemplos para la posterior adaptación de la estructura.

- Diseño: En esta etapa, contando con los resultados de la etapa de análisis, se diseña la arquitectura de *Main Memory DW*, se crean los algoritmos que realizan las funciones de la aplicación y generan los reportes solicitados por el usuario.

- Codificación: Construyendo la aplicación *Main Memory DW*, se codifica cada uno de los algoritmos necesarios para su funcionalidad, obteniendo un sistema que permite el ingreso de información proveniente de un DW y entrega reportes con resultados a consultas de agregación realizadas directamente sobre los datos almacenados en una estructura compacta k^2-treap.

- Pruebas: Se realizaron pruebas en la etapa de Codificación para analizar el correcto resultado de los datos resultados de la función de agregación realizada a partir de los parámetros entregados por las entradas del sistema. Además, se

realizaron pruebas en una etapa final para la comprobar de la mejora de eficiencia en tiempo de respuesta al realizar consultas sobre los datos en memoria principal y la cantidad de memoria utilizada por los datos.

La implementación de *Main Memory DW* se realiza en lenguaje C++, en el Sistema Operativo Linux distribución Ubuntu 16.04 y utilizando las estructuras y funciones implementadas en la librería SDSL. La librería SDSL fue utilizada para la aplicación porque esta consta de las estructuras necesarias ya implementadas, realizando la adaptación de estas para responder a las consultas realizadas por los algoritmos creados. *Main Memory DW* se ejecutó en dentro de las etapas del proyecto en una máquina con Sistema Operativo Linux distribución Ubuntu 16.04, Procesador Intel Core I5 2.30 GHz y 4 GB de memoria RAM.

5.3. Interfaz

Como el objetivo del proyecto tiene enfoque más funcional, creando una aplicación capaz de realizar consultas de agregación SQL sobre un DW almacenado en estructura de datos compacta k^2-treap, no se considera el diseño de una interfaz gráfica más elaborada, se muestra una interfaz básica a través de consola entregándole al usuario las diferentes funcionalidades de *Main Memory DW*. La Figura 5.2 muestra el menú principal de la aplicación *Main Memory DW* el cual consta de cuatro opciones: Carga de Datos, generar reporte consulta de agregación por niveles, generar reporte consulta todos los niveles y salir.

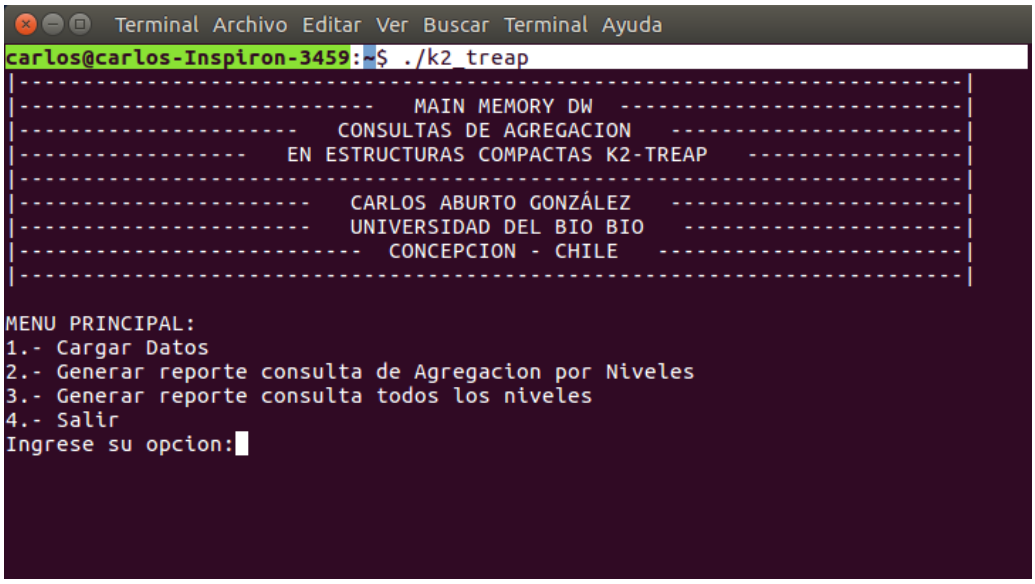


Figura 5.2: Menú Principal *Main Memory DW*

La Figura 5.3 muestra el ingreso de los parámetros realizado por el usuario en la opción cargar datos para un ejemplo de prueba de la aplicación.

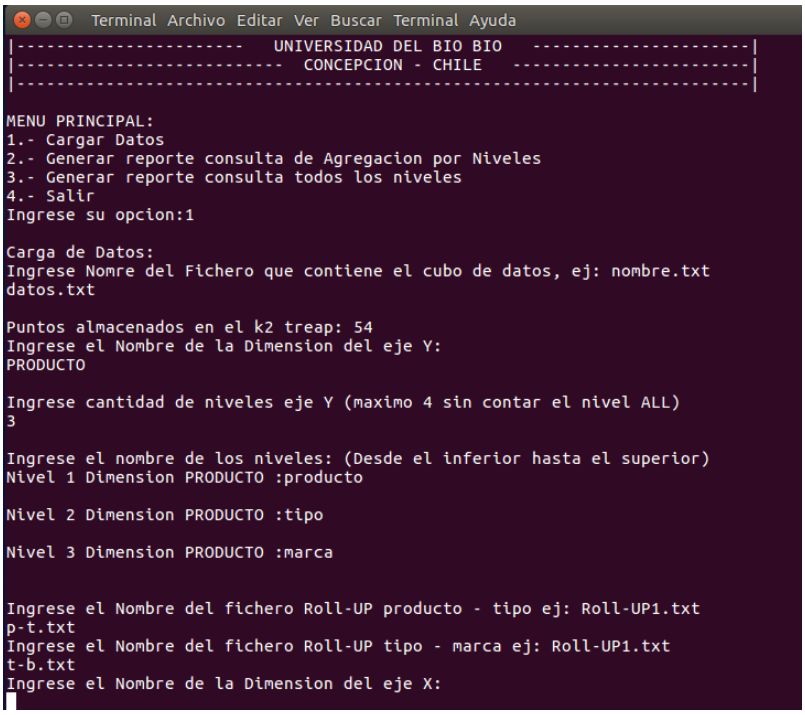


Figura 5.3: Menú Carga de Datos

La Figura 5.4 muestra las opciones de los niveles de cada dimensión para generar el reporte y la Figura 5.5 muestra el reporte generado a través de las opciones seleccionadas por el usuario.

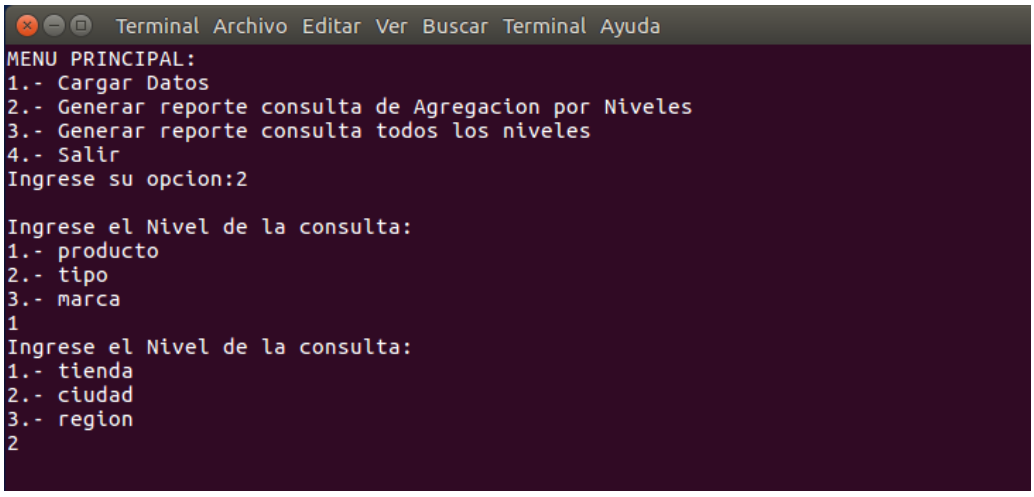


Figura 5.4: Opción generar reporte consulta de agregación por niveles

La Figura 5.5 muestra la selección de niveles para una una consulta de agregación para los niveles Producto - Ciudad y el reporte generado por Main Memory DW.

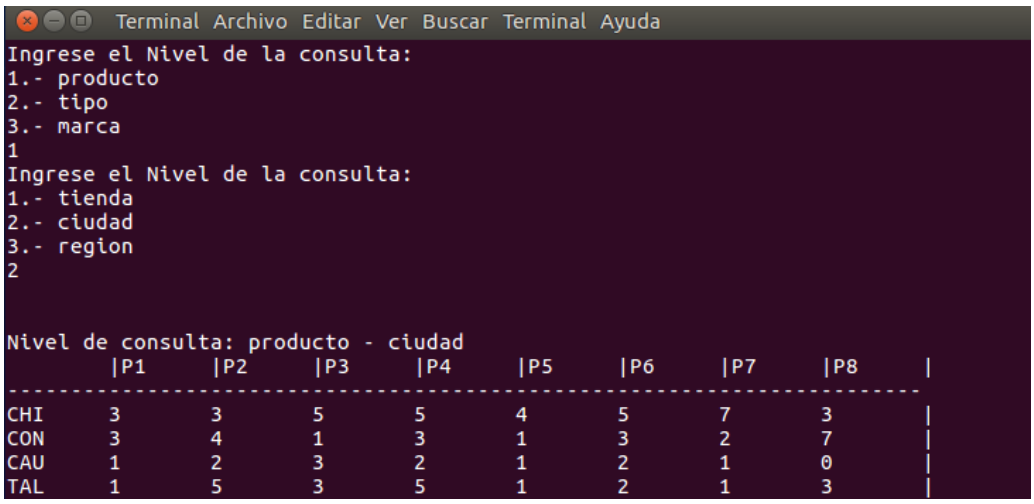


Figura 5.5: Reporte generado con selección de niveles para la consulta de agregación

Además, la Figura 5.6 muestra un reporte generado a través de consola para todos los niveles de jerarquía de las dimensiones.

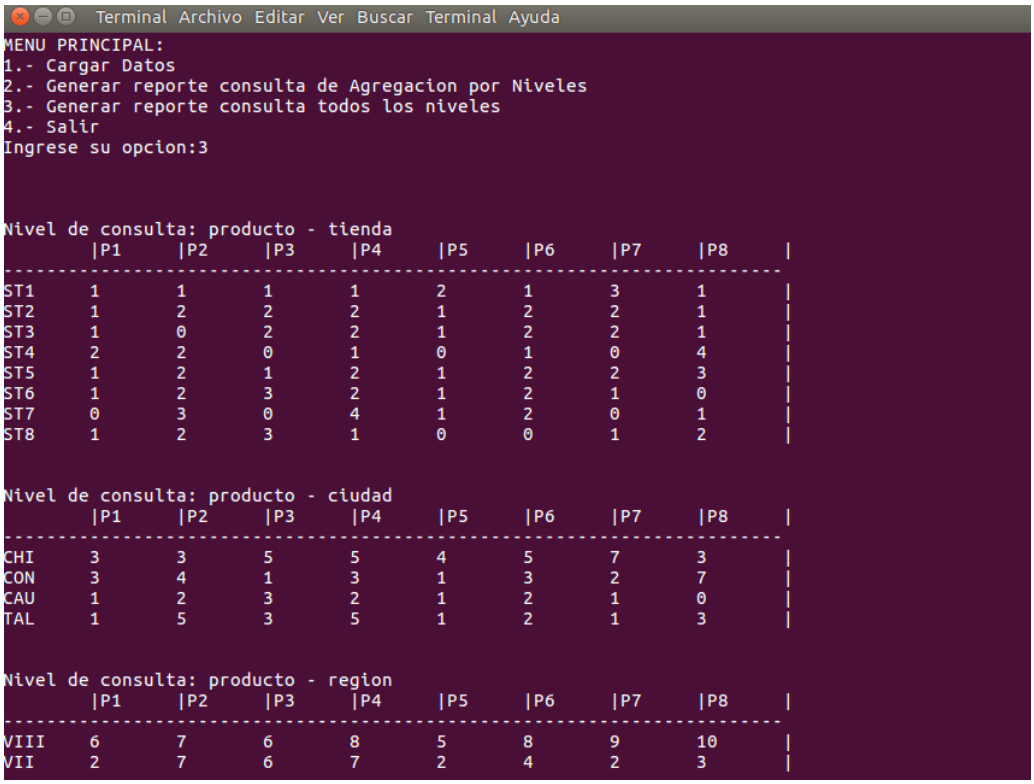


Figura 5.6: Interfaz menú Generar reportes con respuesta a consulta para todos los niveles

Capítulo 6

Experimentación

En este capitulo se presentan los resultados de las pruebas realizadas a la aplicación *Main Memory DW* para medir su eficiencia comparando los resultados con los obtenidos en las pruebas a una base de datos cargada en PostgreSQL versión 9.3.12 utilizando un un esquema de copo de nieve (Figura 6.1) para el modelo presentado en la Figura 1.1. Los factores a considerar en esta etapa de pruebas son los tiempos de respuesta a las consultas de agregación y el espacio en memoria utilizado por los datos.

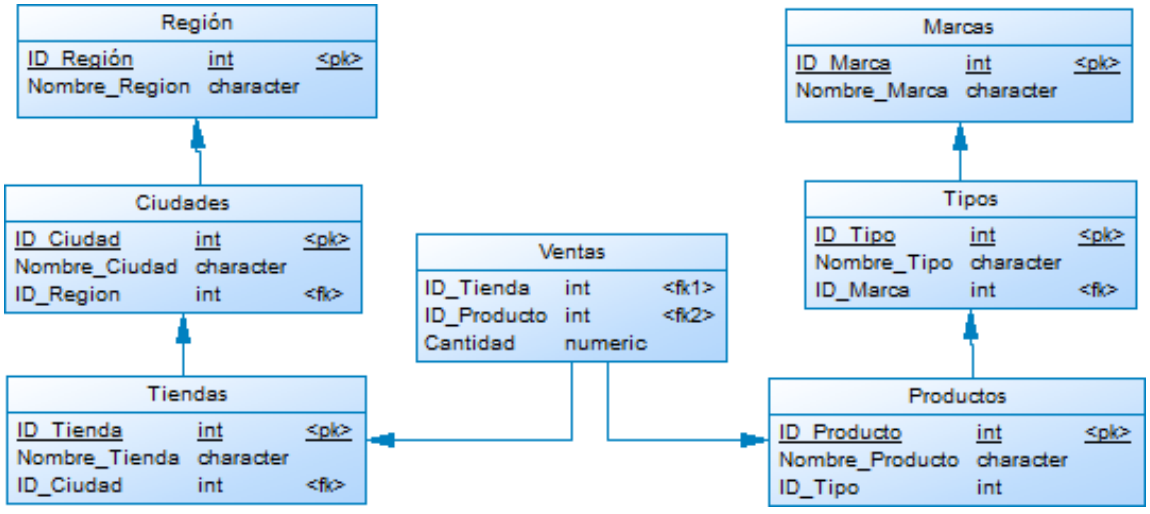


Figura 6.1: Modelo copo de nieve para DW utilizado en las pruebas

Las pruebas fueron realizadas en una maquina con Sistema Operativo Linux

distribución Ubuntu 16.04, Procesador Intel Core I5 2.30 GHz, 4 GB de memoria principal (memoria RAM) y 500 GB de memoria interna.

6.1. Caracterización de los datos de prueba

Las pruebas realizadas a la aplicación consideran la estructura del DW que se ha utilizado para la mayoría de los ejemplos en este proyecto, teniendo un cubo de datos organizado por las dimensiones Tiendas y Productos, con los respectivos niveles de jerarquía para cada dimensión.

Las pruebas se ejecutaron para ocho cubos de datos de distintos tamaños, generando valores con una distribución normal, a través de algoritmos para la creación de estos, obteniendo como resultado cubos de $2,500$, $5,625$, $10,000$, $40,000$, $160,000$, $360,000$, $640,000$ y $1,0000,000$ de elementos, con el objetivo de poder comparar las diferencias entre tamaño de memoria en los datos sin compactar respecto a los datos compactados en la estructura. También se ejecutaron pruebas midiendo el tiempo de respuesta a las consultas de agregación realizadas por la aplicación para los distintos cubos generados, comparándolos con los tiempos obtenidos al ejecutar las consultas equivalentes en la base de datos cargada en PostgreSQL SGBD.

6.2. Resultados de las pruebas

En esta sección se representan los resultados obtenidos en las pruebas realizadas a la aplicación *Main Memory DW*, las cuales buscan medir los tiempos que tardan los datos en ser cargados en la estructura, comparar las diferencias entre los tamaños en memoria de los cubos de datos, antes y después de ser compactados (Tabla 6.1) y medir el tiempo de respuesta a las consultas de agregación realizadas sobre los datos.

6.2.1. Pruebas de eficiencia en memoria

Uno de los objetivos más importante de este proyecto de título es lograr compactar los datos logrando una disminución del tamaño de la memoria utilizada por estos versus el tamaño real de los datos. A continuación se presentan los resultados en la de las pruebas realizadas bajo este criterio:

# elementos	# ceros	Tamaño SGBD (kB)	Tamaño Compacto (kB)
2.500	173	7.400	4,86
5.625	415	7.624	11,34
10.000	774	8.128	19,85
40,000	3.077	9.864	77,52
160.000	12.297	17.000	309,20
360.000	27.780	30.000	695,27
640.000	48.959	48.000	1.235,72
1.000.000	76.576	71.000	1.930,49

Tabla 6.1: Resultados carga cubo de datos

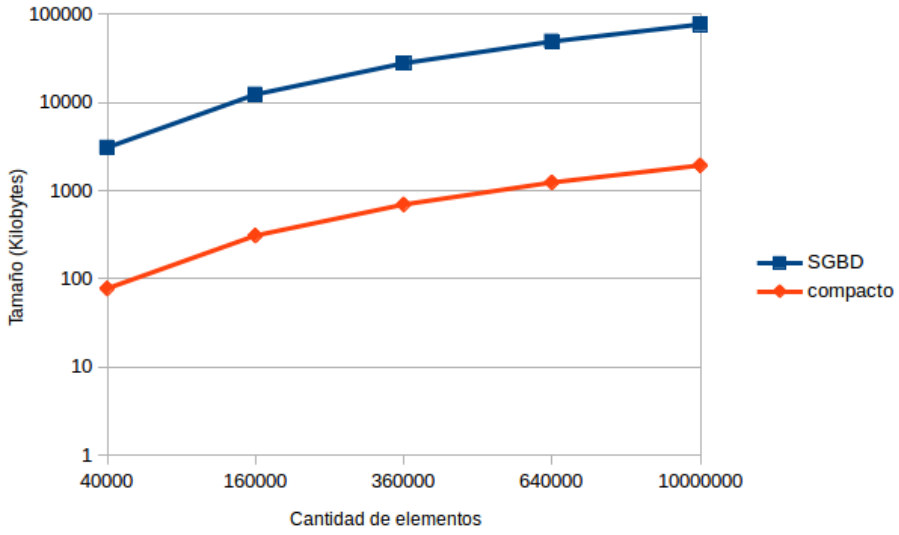


Figura 6.2: Almacenamiento de los cubos de datos en el k^2-treap y en el SGBD PostgreSQL

La Figura 6.2 muestra la gráfica de los valores obtenidos obtenidos en las pruebas al momento de medir la memoria utilizada por los cubos de datos compactados y no compactados, dejando en evidencia la mejora al momento del tamaño a utilizado por los datos al ser compactados en la estructura k^2-treap por medio de la aplicación *Main Memory DW*.

6.2.2. Pruebas de eficiencia en tiempos de respuesta a consultas de agregación

Disminuir sustancialmente los tiempos de respuesta a las consultas de agregación es otro de los ejes centrales de este proyecto de título, por lo que la aplicación *Main Memory DW* maneja los datos almacenados en la estructura compacta k^2-treap poniendo en evidencia todo el trabajo teórico y de investigación realizado a lo largo de la implementación de esta aplicación. A continuación se presentan los resultados de las pruebas realizadas para medir los tiempos de respuestas a las consultas de agregación realizadas en datos almacenados en PostgreSQL SGBD y paralelamente compactado en una estructura k^2-treap:

Nivel Consulta	# elememntos en el cubo							
	2,500		5,625		10,000		40,000	
	k^2-treap	SGBD	k^2-treap	SGBD	k^2-treap	SGBD	k^2-treap	SGBD
Tienda - Producto	0,035	89	0,037	198	0,035	338	0,037	1300
Tienda - Tipo	0,051	25	0,035	26	0,040	36	0,013	76
Tienda - Marca	0,037	26	0,035	33	0,120	34	0,038	54
Tienda - All	0,034	24	0,036	28	0,041	37	0,035	45
Ciudad - Producto	0,042	26	0,045	34	0,036	37	0,048	52
Ciudad - Tipo	0,039	27	0,037	34	0,035	35	0,131	43
Ciudad - Marca	0,037	23	0,015	36	0,041	25	0,036	44
Ciudad - All	0,036	23	0,033	25	0,045	35	0,040	46
Region - Producto	0,034	23	0,032	25	0,045	28	0,039	73
Region - Tipo	0,040	27	0,036	31	0,039	35	0,036	55
Region - Marca	0,050	26	0,091	35	0,042	35	0,037	51
Region - All	0,041	25	0,033	29	0,034	32	0,036	44
All - Producto	0,047	25	0,038	27	0,057	32	0,039	43
All - Tipo	0,046	24	0,039	26	0,114	34	0,057	44
All - Marca	0,037	23	0,037	26	0,037	36	0,035	56
All - All	0,036	24	0,032	25	0,045	28	0,038	33

Tabla 6.2: Tiempos de ejecución de consultas de agregación para los cubos de datos

Como podemos apreciar en la Tabla 6.2 y Tabla 6.3 en todas las pruebas ejecutadas, para distintos tamaños de cubos, nuestros algoritmos tienen mejores tiempos de ejecución, con diferencias de grandes magnitudes, en relación a la proporción de sus datos. Las Figuras 6.3, 6.4 y 6.5 muestran los tiempos de ejecución para las consultas del nivel inferior, la consulta de agregación agrupara por los niveles Ciudad y Tipo y la consulta de agregación agrupada por el nivel All de cada dimensión. Esta última consulta, obtiene las ventas totales de todos los productos en todas las tiendas.

Nivel Consulta	# elememntos en el cubo							
	160,000		360,000		640,000		1,000,000	
	k^2-treap	SGBD	k^2-treap	SGBD	k^2-treap	SGBD	k^2-treap	SGBD
Tienda - Producto	0,036	5400	0,039	12500	0,043	22300	0,045	34700
Tienda - Tipo	0,018	240	0,032	467	0,033	760	0,050	1100
Tienda - Marca	0,450	208	0,049	437	0,050	766	0,053	1200
Tienda - All	0,036	124	0,045	219	0,047	379	0,047	569
Ciudad - Producto	0,051	176	0,059	317	0,061	490	0,063	754
Ciudad - Tipo	0,052	127	0,056	224	0,062	390	0,029	600
Ciudad - Marca	0,048	125	0,049	235	0,052	398	0,045	588
Ciudad - All	0,098	107	0,120	228	0,133	361	0,089	539
Region - Producto	0,042	462	0,047	1010	0,052	1800	0,035	2900
Region - Tipo	0,038	357	0,048	754	0,051	1300	0,049	2100
Region - Marca	0,051	281	0,053	501	0,057	910	0,067	1500
Region - All	0,039	104	0,045	220	0,047	377	0,054	579
All - Producto	0,059	136	0,062	241	0,064	389	0,078	578
All - Tipo	0,130	115	0,134	216	0,168	366	0,170	588
All - Marca	0,039	104	0,041	206	0,046	387	0,065	547
All - All	0,048	85	0,051	169	0,057	274	0,068	415

Tabla 6.3: Tiempos de ejecución de consultas de agregación para los cubos de datos

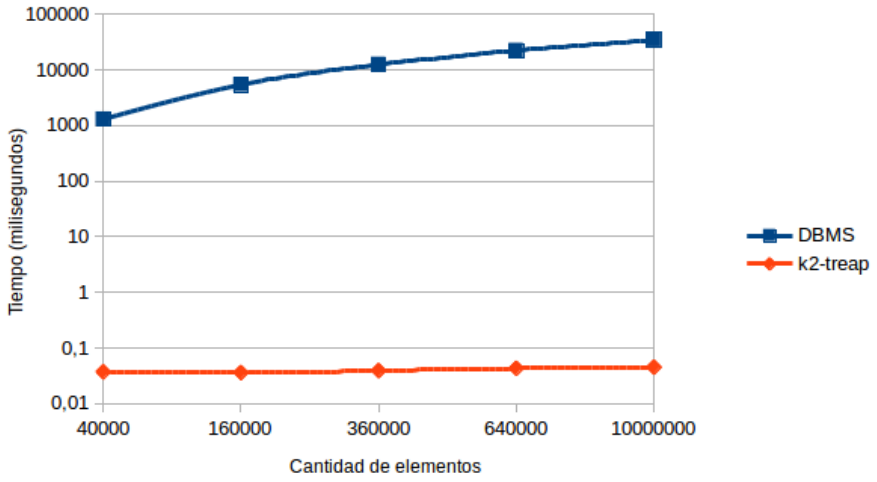


Figura 6.3: Consulta de agregación nivel Tienda (Dimensión Tiendas) y Producto (Dimensión Productos)

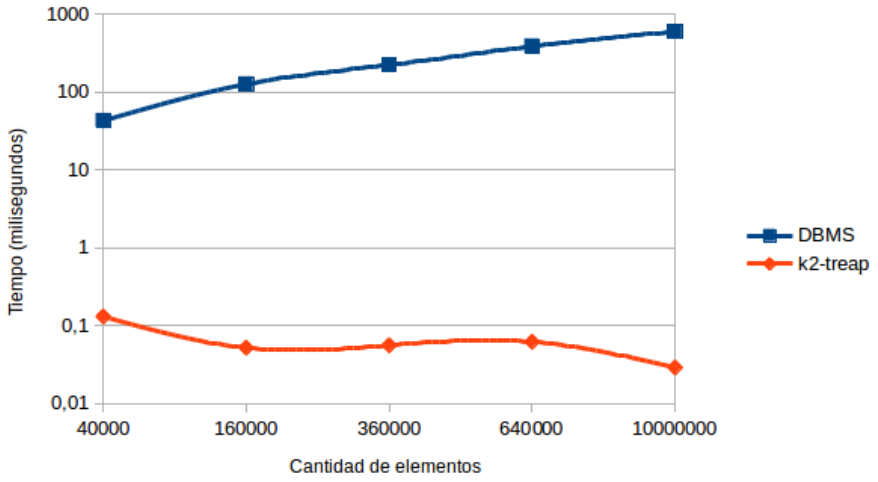


Figura 6.4: Consulta de agregación nivel Ciudad (Dimensión Tiendas) y Tipo (Dimensión Productos)

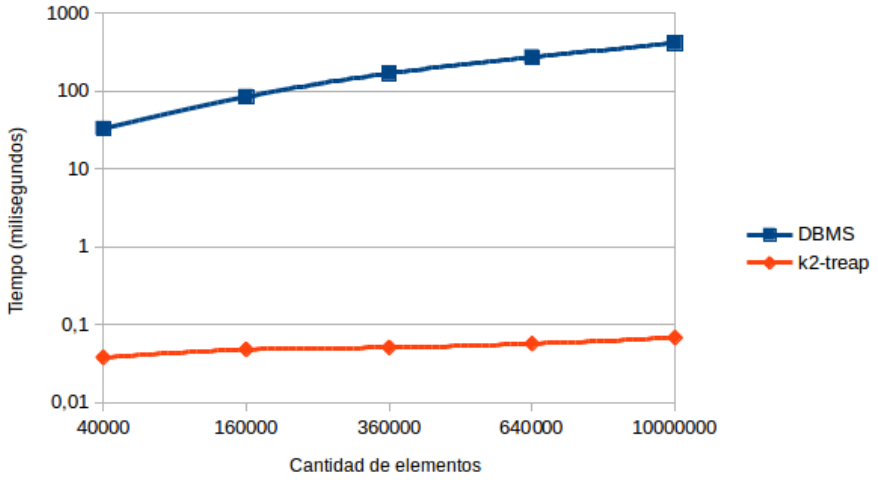


Figura 6.5: Consulta de agregación nivel All (Dimensión Tiendas) y All (Dimensión Productos)

Capítulo 7

Conclusión

En este proyecto se presenta la aplicación *Main Memory DW* que implementa consultas de agregación sobre DWs almacenados en estructuras de datos compactas. En particular los elementos de dimensiones se representan en tablas de una columna, las relaciones rollup en bitmaps y los cubos de datos en la estructura compacta k^2-treap.

La experimentación realizar sobre diferentes tamaños de cubos de datos demuestra que el uso de estructuras compactas es viable para DWs con respecto a la implementación en SGBD. Para todos los casos evaluados se concluye que el sistema basado en estructuras compactas responde a consultas de agregación entregando resultados con tiempos muy eficientes y a un bajo costo de memoria.

Al momento de implementar la solución propuesta en el presente proyecto, además de una labor de codificación e investigación en el área del lenguaje de programación, se estableció una sólida base teórica y analítica sobre estructuras de datos compactas. Como trabajo futuro se puede mencionar la implementación de otras consultas de agregación con funciones tales como `MIN`, `MAX` y `AVG` (promedio), y también el uso de otras estructuras compactas para la representación de cubos de datos con más de dos dimensiones.