UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Estudio de espacios métricos y análisis de algoritmos de indexación, aplicados en una base de datos NoSQL.

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

MATEMÁTICA

P R E S E N T A:

Rosa María Linares Rojo

DIRECTOR DE TESIS:
M. en C. Fernando García Ruíz
2016

Ciudad Universitaria, CDMX

1. Datos del alumno.

Linares
Rojo
Rosa María
55 43 73 16 33
Universidad Nacional Autónoma de México
Facultad de Ciencias
Matemáticas
305067991

2. Datos del asesor.

García
Ruíz
Fernando

Cobían
Campos
José Alfredo

3. Datos de la Tesis

Estudio de espacios métricos y análisis de algoritmos de indexación, aplicados en una base de datos NoSQL.
99 p.
2016

Estudio de espacios métricos y análisis de algoritmos de indexación, aplicados en una base de datos NoSQL.

Agradecimientos

A mi Dios, por siempre escuchar mis incansables suplicas, por guiarme, cuidarme y rodearme de personas lindas que me han apoyado durante este recorrido. Son muchas las metas que me ha permitido lograr con fe y esperanza. Dios siempre serás mi luz y apoyo.

A mis padres por darme la vida e impulsarme a cumplir mis metas, principalmente a mi madre Rosalina Rojo Rojo por apoyarme siempre y llenarme de amor infinito, amor que me dio la confianza de seguir adelante sin titubear, su apoyo y los valores impartidos de respeto, amistad, sinceridad, sencillez, responsabilidad me ayudaron a enfrentar las metas planteadas hasta cumplirlas, mami te amo.

A mis hermanos, que son los mejores del mundo, Oscar Linares Rojo gracias por ser mi cómplice, confidente, amigo y hermano; por ser tan amoroso conmigo, por estar siempre en las buenas y las malas, por cuidarme y hacerme reír casi toda mi vida , te amo infinitamente. A Jesús Linares Rojo por ser mi inspiración para ser una mejor persona cada día, por ser una fuente de distracción y diversión cuando el estrés se iba apoderando de mí, gracias por hacerme sentir las mejores emociones de mi vida, eres mi corazón. . . te amo.

A mis amigas, casi hermanas, Aurora Márquez Espinoza, Ana Lilia Cruz y Nidia Elizabeth Gómez, por todos los momentos felices y no tan felices que hemos compartido, gracias por ser mi abrigo, mi refugio y esperanza, por estar siempre a mi lado y por todas sus muestras de cariño.

A mis tíos, en especial a mi tía Francisca Rojo por sus consejos y muestras de cariño, gracias por acercarme a Dios cuando más necesitaba de Él y por ayudarme a abrir mi corazón para poder apreciar todo el amor que me rodea. A mi tío Efrén Ávila por cuidarme y quererme como a una hija, gracias tío por tu amor, consejos y cuidados. A mi tío Cesar Rojo por ser mi tío, mi amigo y compañero de locuras, gracias por toda tu confianza y cariño.

A mis compañeros de clase y amigos, que han hecho que este gran reto en mi vida, se llevara de forma más amena, porque no solo la Facultad de Ciencias ha servido para formarme como matemática, sino que en ella he encontrado muchas cosas más. Me ha hecho madurar y encontrar amigos extraordinarios, que ya son parte de mi familia. Quiero hacer una mención especial a Oscar Jiménez Díaz, Juan Manuel de la Huerta, Erik Lara, Rafael Navarro, Juan Carlos Díaz, Christopher Villagra, Salvador Zaragoza, Cesar Téllez y Jair Cantú; por todo el tiempo compartido a lo largo de la carrera, por su comprensión y perseverancia para superar los momentos difíciles.

A Cristina Barbosa y Rodrigo Fragoso, por sus consejos, muestras de cariño y por hacer divertida la recta final de este proyecto.

Debo agradecer de manera especial y sincera al M. en C. Fernando García Ruíz, por mostrarme que la vida es un continuo caminar, un caminar donde lo que le da sentido a cada paso es la discreta certeza de que estamos aportando algo a los demás. Es para mí un honor haber realizado este trabajo bajo su dirección, le estaré eternamente agradecida porque ha dedicado su valioso tiempo para ello.

Al Ing. José Alfredo Cobían Campos, infinitas gracias por aceptar ser parte de este proyecto, por darme la confianza y herramientas necesarias, por haberme guiado en el terreno profesional, por compartirme sus conocimientos éticos y profesionales que hicieron posible la elaboración de la presente tesis.

Al Dr. Miguel Ehécatl Morales por su paciencia, sabiduría y comentarios constructivos para la culminación de este proyecto.

Dedicatoria

A Paula Onésima Rojo Barrera y Marcos Rojo Chávez

Por su amor, apoyo y oraciones.

Índice general

Introducción 3
Resumen 3
Introducción 4
Objetivo 5

Parte 1. Conocimientos previos 7

Capítulo 1. Bases de Datos 9
1.1. Historia de las Bases de Datos 9
1.2. Conceptos Básicos de Bases de Datos 9

Capítulo 2. Búsquedas 17
2.1. Espacios métricos 17
2.2. Tipos de Búsquedas 30
2.2.1. Búsquedas Exactas 30
2.2.2. Búsquedas por Similitud 31
2.2.2.1. Búsquedas por Rango. 32
2.2.2.2. Búsquedas de los k-Vecinos más Cercanos (kNN) 33
2.2.2.3. Búsqueda del Vecino más Cercano 34
2.2.2.4. Búsqueda Inversa de los k-Vecinos más Cercanos 34
2.2.2.5. Búsqueda Join por Similitud 35
2.2.2.6. Combinación de Búsquedas 36
2.2.3. Estrategias Utilizadas en Búsquedas por Similitud 36
2.3. Índices Métricos 37
2.3.1. Indexación Basada en Pivotes 38
2.3.2. Indexación Basada en Particiones Compactas 39
2.3.2.1. Indexación que utiliza Radio de Cobertura 40
2.3.2.2. Indexación que utiliza Hiperplanos 41
2.4. Teoría de Gráficas 44
2.5. Descripción de Algoritmos Existentes en Espacios Métricos 47
2.5.1. Algoritmos de Indexación Basado en Pivotes 48
2.5.1.1. Burkhard-Keller Tree (BKT) 48

2.5.1.2. Fixed Queries Tree (FQT) 49
2.5.1.3. Fixed-Height Queries Tree (FHQT) 50
2.5.1.4. Fixed Queries Array (FQA) 51
2.5.1.5. Vantage Point Tree (VPT) 52
2.5.1.6. Multi–Way Vantage Point Tree (mw-VPT) 53
2.5.1.7. Approximating Search Algorithm (AESA) 54
2.5.1.8. Lineal AESA (LAESA) 54
2.5.2. Algoritmos de Indexación Basados en Particiones Compactas 55
2.5.2.1. Bisector Tree (BST) 55
2.5.2.2. Voronoi Tree (VT) 55
2.5.2.3. Generalized Hyperplane Tree (GHT) 56
2.5.2.4. Geometric Near-neighbor Access Tree (GNAT) 56
2.5.2.5. M-Tree (MT) 57
2.5.2.6. Spatial Approximation Tree (SAT) 58
2.5.2.7. Lista de Cluster (LC) 58
2.5.3. Eficiencia de los algoritmos 59

Parte 2. Implementación del algoritmo en la BD 63

Capítulo 3. Descripción del Problema 65
3.1. Introducción 65
3.2. Problema a Tratar 69
3.3. Propuesta de solución 71
3.3.1. Asignación de Longitud 71
3.3.2. Primera Parte del Índice 72
3.3.3. Asignación de Peso 74
3.3.4. Segunda Parte del Índice 78
3.3.5. Definición de un α para Refinar la Búsqueda. 79

Capítulo 4. Ejecución de la propuesta 81
4.1. Antecedentes 81
4.2. Búsquedas 81
4.3. Experimentación 83
4.4. Futuras Implementaciones del Algoritmo 87

Conclusiones 89

Bibliografía 91

Introducción

Resumen

La finalidad de esta tesis es realizar el diseño e implementación de un algoritmo de búsqueda así como el análisis y selección del algoritmo de indexación que mejor se acople a las características de la base de datos y los requerimientos de la búsqueda. Para ello se llevo a cabo lo siguiente:

* En el capítulo 1 se presenta una introducción a los conceptos fundamentales de bases de datos y espacios métricos, para ir adentrándonos al contexto y a las características que se deben tomar en cuenta para el desarrollo de la tesis.
* En el capítulo 2 se describen las estrategias utilizadas en consultas orientadas a las búsquedas por similitud, también se agregan de manera formal los fundamentos necesarios que serán utilizados en capítulos posteriores. En éste también se analizaron los métodos más relevantes de búsquedas por similitud sobre espacios métricos, considerando los que están basados en pivotes y particiones compactas.
* En el capítulo 3 se hizo una propuesta de búsqueda y se selecciona un algoritmo de indexación para hacerla más eficiente.
  En este capítulo se plantea que la mayoría de los procesos de búsqueda en las bases de datos asumen que la comunicación escrita siempre tiene la misma estructura, la cual se ha implementado y asumido que todos debemos utilizar. Sin embargo, el crecimiento del acceso a Internet y la facilidad que se tiene para que cada uno de nosotros comparta información en la web, permite que ésta no tenga la estructura inicialmente planteada pues cada uno puede darle su estilo y originalidad. Esta situación es muy común en la información que se comparte en redes sociales. Por ello se considera que existe la necesidad de implementar una función a nivel de base de datos para poder realizar consultas con los datos extraídos de estas fuentes, las cuales arrojan información que es de utilidad para distintas organizaciones.

Posteriormente se propone una función de búsqueda y se detalla la selección del algoritmo que se considera pertinente para un rendimiento optimo de la base de datos.

* Finalmente en el capítulo 4 se presentan los diferentes experimentos y sus respectivos resultados, así como modificaciones que se pueden implementar posteriormente para que el algoritmo sea útil en bases de datos estáticas y dinámicas.

  En la última parte del trabajo se exponen las conclusiones obtenidas con el análisis de todo el trabajo.

Introducción

Para atender los requerimientos de los sistemas de información de nuestra época, las bases de datos actuales deben tener la capacidad de almacenar diversos tipos de datos. Por ejemplo, cadenas de caracteres, audio, video, imágenes, huellas digitales, marcas del iris, etcétera. Organizar estos datos y adecuarlos al concepto tradicional de *búsqueda exacta*, es muy costoso ya que actualmente las bases de datos almacenan objetos que regularmente no son iguales.

Sin embargo aún se tiene la necesidad de realizar diversas consultas en estas bases de datos, por lo tanto la alternativa que se tiene es recuperar objetos similares a uno dado. A este tipo de búsquedas se les conoce con el nombre de *búsquedas por proximidad* o *búsquedas por similitud*.

Al considerar lo anterior nos podemos percatar que es importante estructurar la información de tal manera que la proyección de una búsqueda sea rápida, lo más similar posible y además el consumo de recursos de computo sea mínimo. Por ello los espacios métricos constituyen un modelo matemático que permite formalizar las búsquedas por similitud, ya que en ellos se puede definir una *función distancia* que refleje la semejanza entre los objetos de la base. Esto hace posible el planteamiento de métodos de búsqueda que sean eficientes.

Por ejemplo, dos tipos de consultas para este tipo de búsquedas son las consultas por rango y las consultas por los k-Vecinos más cercanos. Con la finalidad de minimizar el costo de estas consultas se han desarrollado algoritmos de búsqueda en espacios métricos para reducir el número de evaluaciones, de ahí la necesidad de indexar la base.

Los algoritmos de indexación se dividen en dos enfoques uno basado en *particiones compactas* y otro en *pivotes*.

La idea general de los índices basados en *particiones compactas* es dividir la base de datos en zonas tan compactas como sea posible, y a cada una de éstas asignarle

un centro. Existen dos criterios, uno basado en *regiones de Voronoi* y el otro en *cobertura de radios*.

En cuanto a los índices basados en *pivotes* se utiliza un conjunto U de k elementos llamados "*pivotes*", es decir, $U = \{p_1, \ldots, p_k\}$. Luego se almacena para cada elemento x del conjunto de datos, su distancia a los k pivotes $(d(x, p_1), \ldots, d(x, p_k))$. Dada una consulta c con rango de búsqueda r, sus distancias a los k pivotes son calculadas $(d(c, p_1), \ldots, d(c, p_k))$. Si para algún pivote p_i se tiene que $|d(c, p_i) - d(x, p_i)| > r$, entonces esos objetos serán descartados y no se evaluarán en la consulta. Todos los demás elementos que no se descarten deberán ser evaluados.

El uso de espacios métricos, permite elegir el enfoque más adecuado para indexar la base, sin restar importancia al modelado y diseño de los algoritmos. Esto permite obtener resultados más precisos y detectar fallas o errores para que las consultas sean más eficientes.

Objetivo

Este trabajo pretende analizar los algoritmos de indexación más importantes sobre espacios métricos, para seleccionar el que mejor se adecúe a las características de la base de datos. Por otro lado se propone desarrollar un algoritmo de consulta de bases de datos semiestructuradas, que sea capaz de responder a las búsquedas por similitud más comunes sobre documentos.

Parte 1

Conocimientos previos

Capítulo 1

Bases de Datos

1.1. Historia de las Bases de Datos

Desde la antigüedad el hombre ha tenido la necesidad de guardar información sobre los acontecimientos que ocurren a su alrededor día con día, los sucesos que se consideraban más importantes eran preservados en pinturas, grabados, papiros y después en papel. Con el paso del tiempo, la sociedad se volvió más compleja y la manera de guardar la información que ésta generaba también se alteró.

Por ejemplo, el surgimiento de organizaciones bien establecidas con distintos fines: económicos o sociales, trajo como consecuencia la necesidad de utilizar libros de registros. El crecimiento de estas organizaciones produjo que dichos registros se volvieran difíciles de manejar e interpretar.

La llegada de las computadoras trajo diversos beneficios, algunos de ellos son: medios de registro y procesamiento más simples y ágiles, de esta manera surgió una nueva tecnología de almacenamiento de datos, la estandarización de los mismos así como su análisis. En la Figura 1.1.1 se describe brevemente la historia y tendencia de las bases de datos, así como la evolución del uso que se les está dando.

[1]

En seguida se mostrarán algunos conceptos básicos y necesarios que serán de gran utilidad a lo largo de este trabajo.

1.2. Conceptos Básicos de Bases de Datos

El concepto más importante y que por si sólo carece de significado, es un *dato*, éste representa un hecho del mundo real y adquiere un significado cuando se le da una interpretación [**1**]. Por otro lado, al conjunto de datos interrelacionados,

[1] La Figura 1.1.1 es una adaptación de:
https://unpocodejava.files.wordpress.com/2015/08/image0011.jpg

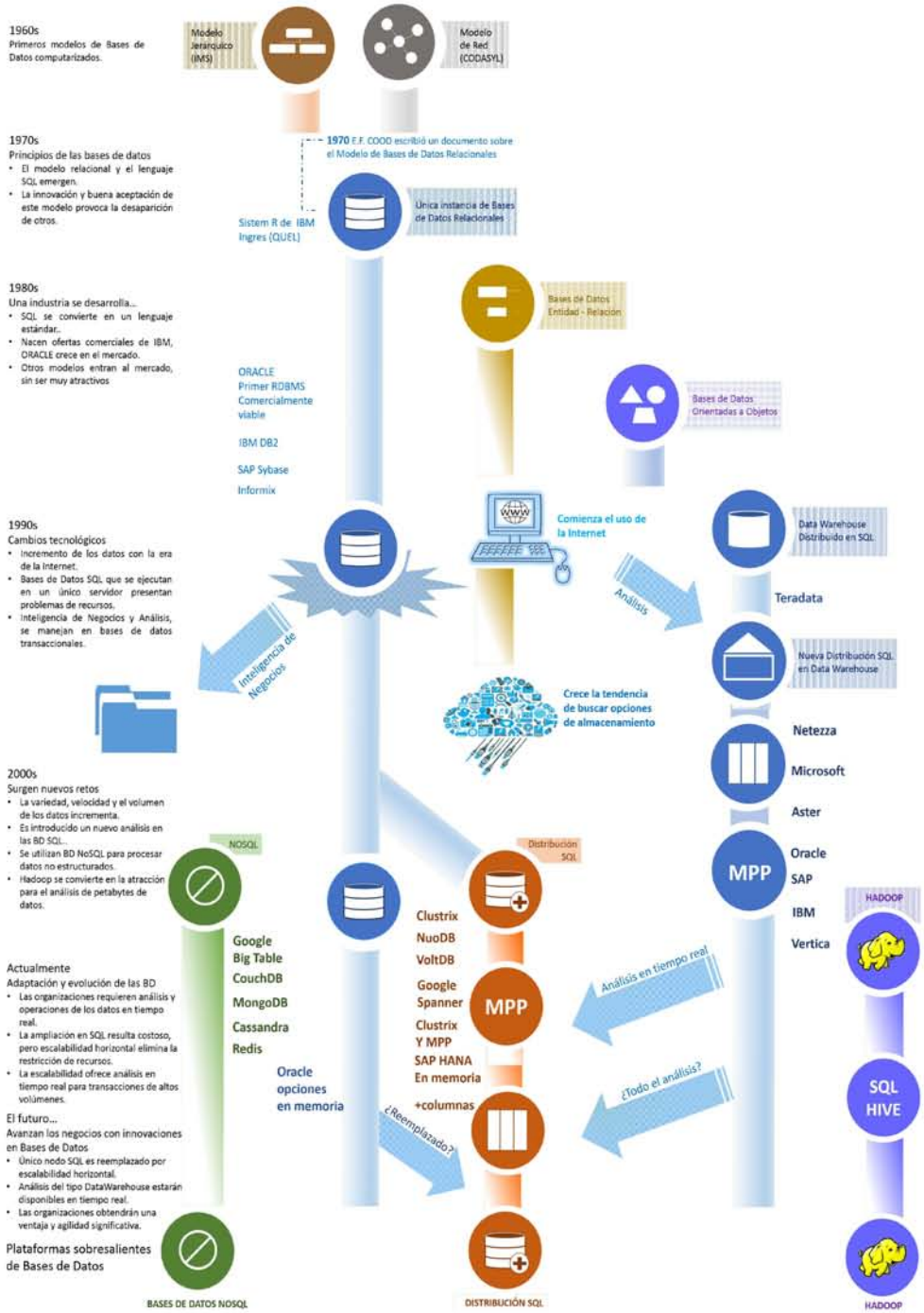


FIGURA 1.1.1. Descripción histórica de las bases de datos.

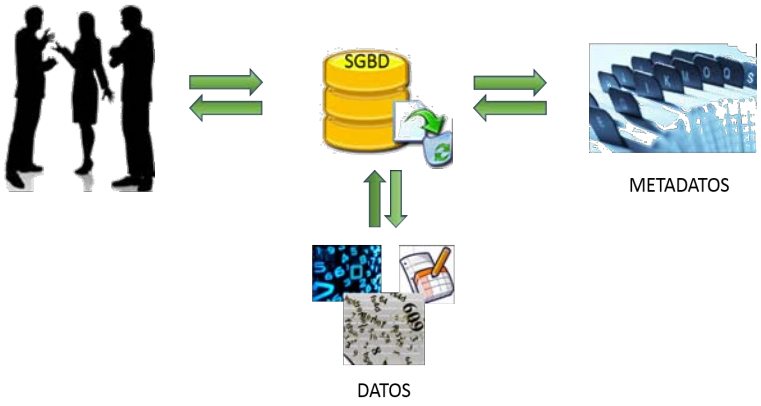


FIGURA 1.2.1. Interacción entre usuarios y SGBD.

que mantienen cierto orden y se encuentran guardados en un lugar específico se le denomina *Base de Datos* (BD) [**1**].

Al conjunto de programas que permiten la definición, construcción, manipulación y acceso a las BD, se les llama *Sistema de Gestión de Bases de Datos* o *Sistema Gestor de Bases de Datos* (SGBD) , como es de esperarse los archivos que se generen en un equipo de computo mediante cualquier software al procesar los datos, generan más datos, los cuales son llamados *metadatos,* conocidos coloquialmente como "los datos de los datos". La Figura 1.2.1 muestra la interacción del usuario con los datos por medio del SGBD.

Los tipos de BD se pueden clasificar de varias formas, de acuerdo al contexto que se esté manejando, la utilidad de las mismas o las necesidades que satisfagan.

La primera clasificación que se va a realizar es de acuerdo a la variabilidad de los datos, si la BD solamente se utiliza para lectura, es decir, se almacenan datos históricos que posteriormente se usan para analizar el comportamiento de un conjunto de datos a través del tiempo, realizar proyecciones, entre otros; entonces es una BD estática. Sin embargo, si la BD almacena información que puede ser modificada con el tiempo, es decir permite que los datos sean actualizados, borrados, consultados y aditados, además de las operaciones fundamentales de consulta; entonces es una BD dinámica. Un ejemplo de este tipo de bases son las que se utilizan en los bancos.

2

[2]Texto adaptado de http://datateca.unad.edu.co/contenidos/301330/Contenido_Linea_EXE-1/tipos_de_bases_de_datos_segn_la_variabilidad.html

Cuando se define una BD se debe realizar una representación de sus especificaciones entre ellas están la estructura, las restricciones y los tipos de datos. Por ello las BD también se pueden clasificar de acuerdo al modelo de administración de sus datos, a esta representación se le denomina *Modelo de Datos* [**2, 3**]. Los modelos de datos que se utilizan con mayor frecuencia son:

* Modelo de Datos Jerárquico

En estas BD se almacena la información en una estructura jerárquica, este modelo se clasifica en estructuras lineales y de arborescentes. El objetivo principal en este tipo de bases es establecer una jerarquía de fichas, de tal forma que una ficha puede contener a otras y así sucesivamente. Otra de sus características es que su relación es una a varias, es decir, sólo una ficha puede tener una jerarquía más alta que otras.

Las ventajas de este modelo es que se pueden manejar volúmenes de información muy grandes, además de que su estructura es estable y de gran rendimiento. Las desventajas de este modelo es que presenta ciertas anomalías en la inserción y modificación de los datos así como ineficiencia en la redundancia de datos.

* Modelo de Datos Reticular (Red)

Estas BD tienen una ligera modificación con respecto a las anteriores, la diferencia más relevante es que permite la relación varios a varios. Con ello varias fichas pueden tener una jerarquía mayor que una o varias. Este cambio solucionó eficientemente al problema de redundancia de datos, sin embargo este modelo también presenta inconvenientes, entre las desventajas más destacadas se tiene que es complicado definir nuevas relaciones y que se desperdician muchos recursos computacionales.

* Modelo de Datos Relacional

El modelo de este tipo de BD, es muy utilizado ya que con él se pueden representar diversas problemáticas de la vida cotidiana y administrar datos de forma dinámica. En este modelo los datos están organizados estrictamente en tablas de valores, sobre las cuales se ejecutan ciertas operaciones a las cuales se les ha denominado álgebra relacional. Durante la realización del diseño de una BD relacional se realiza un proceso de normalización de la misma, el cual consiste en un conjunto de relaciones y características que debe cumplir la base.

* Modelo de Datos Orientado a Objetos

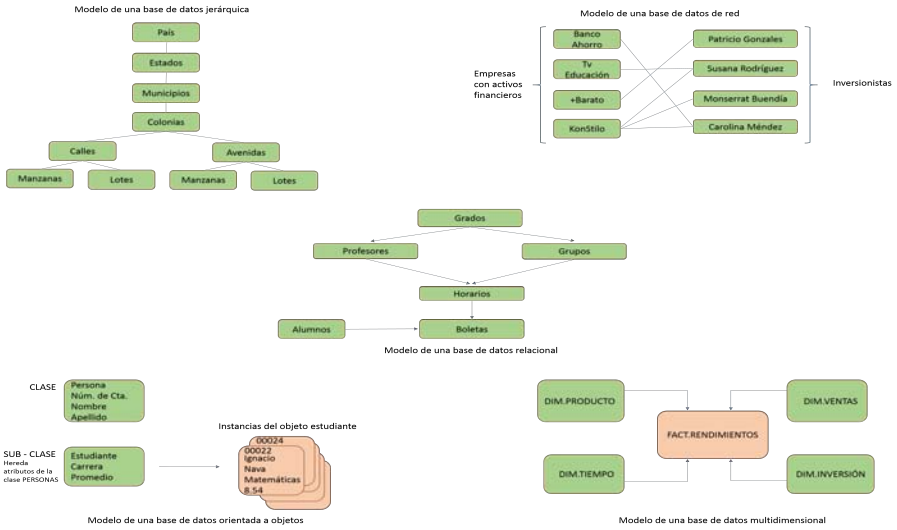


FIGURA 1.2.2. Ejemplos de modelos de bases de datos.

Estas BD organizan la información que contienen en atributos y al comportamiento de estos se les denomina operaciones. En ellas se almacenan tipos de datos complejos, los cuales se integran con lenguajes de programación y tienen un rendimiento elevado.

* Modelo de Datos Multidimensional

Las bases de datos multidimensionales son llamadas así porque los datos están almacenados en arreglos de varias dimensiones. Estos arreglos pueden ser interpretados en términos de variables dependientes e independientes. Las últimas determinan la dimensión de la BD.

Algunos ejemplos de cómo se utilizan este tipo de modelos se pueden observar en la Figura 1.2.2.

A partir de que ya se tiene un modelo conceptual de la BD, se debe establecer un modelo lógico, en éste se definen una colección de reglas generales de integridad, en donde se establecerán relaciones, vínculos y atributos de los objetos de la BD.

El objetivo del modelo lógico es convertir los esquemas conceptuales en un esquema que integre las características de la BD con las del SGBD sobre el que se va a implementar la base.

Finalmente se realiza un modelo físico de la BD, en donde se especifica cómo se almacenan los datos, el espacio que se va a ocupar, métodos de acceso, fuentes de datos, entre otros.

El avance de la tecnología se ha convertido en el principal protagonista para incrementar de manera impresionante tanto volumen como la diversidad de los datos, pues estos se obtienen de diversas fuentes tales como redes sociales, paginas web, aplicaciones, etcétera.

Los datos que actualmente se almacenan son estructurados y no estructurados, los primeros están fuertemente definidos es decir son de un tipo y un tamaño, los no estructurados son datos sin tipos pre-definidos y se almacenan como "objetos" sin estructura uniforme. Los datos que la mayoría de las organización y los humanos generamos son no estructurados.

A continuación se mencionan algunos ejemplos de los datos no estructurados que actualmente se generan [**4**]:

* Datos científicos: Algunos de ellos son datos sísmicos, atmosféricos y de diversos tipos de energía a gran escala, también fotografías y videos de seguridad, vigilancia y tráfico. Esto incluye datos del tiempo o datos que el gobierno capta en sus imágenes de vigilancia por satélite. Basta con pensar en Google Earth.
* Datos de medios de comunicación social: Estos se genera a partir de las plataformas redes sociales como YouTube, Facebook, Twitter, LinkedIn, etcétera.
* Datos móviles: Entre los más comunes son mensajes de texto, ubicaciones, fotografías, videos, audio, entre otros.

Sería bueno pensar que para cada innovación en la gestión de datos, hay un nuevo comienzo desconectado del pasado. Sin embargo, no es así, la mayoría de las nuevas etapas se construyen sobre sus predecesores. La evolución de la gestión de datos incluye avances tecnológicos en hardware, almacenamiento, redes, y modelos de computación como la virtualización y "almacenamiento en la nube".

La convergencia de las tecnologías emergentes y la necesidad de reducir los costos computacionales han transformado la manera de manipular y aprovechar los datos. La última tendencia que surgió a causa de los factores ya mencionados es *Big Data* [**5**], se puede definir como cualquier tipo de fuente de datos que tiene al menos tres características comunes [**5**]:

1. Volumen. Genera grandes volúmenes de datos
2. Velocidad. La velocidad con la que se genera la información hace imposible gestionarla con sistemas de BD convencionales.
3. Variedad. Genera información con diferentes tipos de datos y sin estructura.

Lo relevante de *Big Data* es que permite a las organizaciones recopilar, almacenar, administrar y manipular grandes cantidades de datos a la velocidad y tiempo adecuados, con la finalidad de tomar decisiones basadas en evidencias y datos empíricos, en tiempo real, es decir, al momento.

Hasta este momento se ha visto que los datos, la manera en la que se almacenan y la estructura de los SGBD han cambiado, es natural pensar que la forma de consultarlos y analizarlos también debe cambiar. Anteriormente era muy fácil realizar búsquedas por igualación, ya que las BD contenían cantidades de datos eran menores y además por lo general eran cadenas de caracteres. Sin embargo actualmente los datos tienen características muy diferentes las cuales hacen que las búsquedas se compliquen.

Los SGBD han incorporado la capacidad de almacenar datos no estructurados, en donde las consultas buscan elementos de la bases de datos que sean *similares* o *próximos* a un objeto de la BD dado. En el siguiente capitulo se van a definir las características, propiedades y las búsquedas que se han implementado para este tipo de bases.

Capítulo 2

Búsquedas

En este capítulo se definen formalmente los conceptos y conocimientos necesarios que se van a utilizar a lo largo de este trabajo. Primero se mencionarán las definiciones fundamentales de espacios métricos y algunas búsquedas que se pueden realizar en ellos. Posteriormente se realizará un análisis de las estrategias más utilizadas en las búsquedas mencionadas.

El uso o implementación de las búsquedas que se van a proponer, estará condicionado a un contexto, donde se tomarán en cuenta factores como el tipo de base de datos, los datos que almacena y el tipo de consulta requerida para realizar las búsquedas de manera adecuada. Independientemente del contexto o situación en la se esté trabajando, se van a utilizar las siguientes definiciones.

Una base de datos multimedia contiene objetos tales como cadenas de caracteres, audio, video, imágenes, etcétera. Dentro de estas bases se van a realizar consultas mediante una búsqueda por similitud.

La similitud entre diversos objetos de una base de datos quedará definida como una función de similitud la cual formalmente llamaremos *métrica* [**6**]. Ésta se va a modelar a través de un espacio métrico, donde los objetos de la base serán el conjunto de elementos y la búsqueda por similitud sera una medida de semejanza entre los elementos de ese conjunto, es decir, una métrica.

2.1. Espacios métricos

Definición 1. Sea X un conjunto y d una función tal que $d : X \times X \to [0, \infty)$. La pareja (X, d) es llamada espacio métrico si cumple las siguientes propiedades.

Para todo v, u, w elementos de X, d es:

$m1$) No negativa

$$d(v, w) \geq 0, \;\; d(v, w) = 0 \quad si \quad y \quad solamente \quad si \;\; v = w.$$

$m2$) Simétrica

$$d(v, w) = d(w, v).$$

$m3$) Desigualdad del triángulo

$$d\left(v,w\right) \leq d(v,u) + d\left(u,w\right).$$

Cualquier función con las tres propiedades anteriores es llamada una función distancia o una métrica [**7**].

Es importante mencionar que independientemente de la métrica que se proponga, ésta debe cumplir las propiedades correspondientes para poder modelar la BD con un espacio métrico. Sin embargo, en caso de que una de las condiciones no se cumpla, siempre se tendrá la alternativa de modificar la función distancia.

El problema principal en las bases que almacenan objetos multimedia como audio, video e imágenes; es que a nivel de recursos computacionales las consultas son muy costosas, además de que su cálculo tiende a ser demasiado complejo. Lo anterior ha creado la necesidad de implementar funciones para indexar las BD y finalmente poder trabajar con el menor número de evaluaciones posibles.

En la actualidad se han creado estrategias para que los objetos multimedia, puedan ser almacenados en las BD de tal forma que su manipulación sea más sencilla. Estas estrategias proponen extraer las características más importantes de dichos objetos y representarlas en la base mediante vectores de dimensión n, a este tipo de bases se les denomina *Bases de Datos Multimedia (BDM),* las cuales almacenan sus datos en vectores de varias dimensiones. Debido a ello, se definirá a la extracción σ como la transformación de un objeto $Obj\left(t\right)$ a un vector $X_n = \left(x_i\right)_{i=1}^{n}$. Es decir,

$$\sigma : Obj\left(t\right) \rightarrow X_n.$$

La Figura 2.1.1 muestra una representación del proceso que se lleva a cabo para representar un objeto de la base en un elemento de $n - dimensiones$.

[1]

Una vez que se ha podido utilizar un modelo matemático para representar los objetos multimedia, en este caso vectores, se van a definir los espacios que contienen este tipo de elementos.

Definición 2. Un espacio vectorial V, sobre un campo $\mathbb{R}$ cuyos elementos se denominan escalares, es un conjunto cuyos elementos se denominan vectores, en el cual se encuentran definidas dos operaciones [**7**].

---

[1] La Figura 2.1.1 es una adaptación de:
http://imagenesinfantiles.net/wp-content/uploads/2013/08/imagenes-de-carritos-infantiles-6.jpg
y https://mamaaddisart.files.wordpress.com/2014/01/dibujos-para-colorear-coches-2.jpg

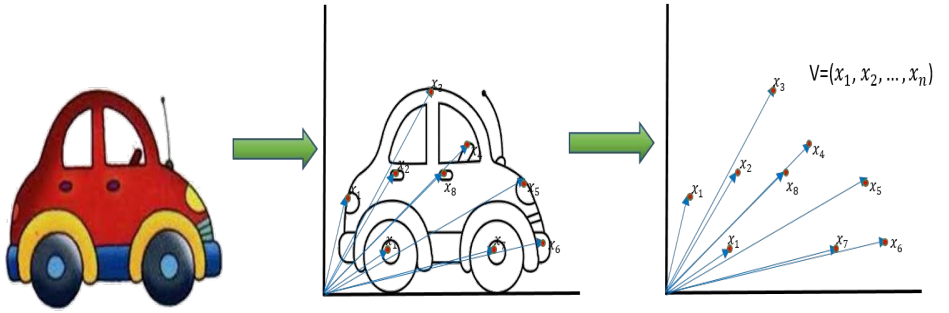


FIGURA 2.1.1. Representación del proceso de transformación de un objeto en un vector de n-dimensiones.

La suma vectorial, $+ := V \times V \to V$ y el producto por escalares $\cdot := K \times V \to V$.

El espacio V cumple con las siguientes propiedades:

Para todo u,v y w elementos de V, y para todo α, β escalares de K se tiene que:

* La suma de vectores es asociativa.

$$(u + v) + w = u + (v + w).$$

* La suma de vectores es conmutativa.

$$u + v = v + u.$$

* Existencia del neutro aditivo.

$$u + \bar{0} = u = \bar{0} + u.$$

* Existencia del inverso aditivo.

$$v + (-v) = \bar{0}.$$

* La multiplicación por escalares es distributiva con respecto a la suma de vectores.

$$\alpha\,(u + v) = \alpha u + \alpha v.$$

* La multiplicación por escalares es distributiva con respecto a la suma de escalares.

$$(\alpha + \beta)\,u = \alpha u + \beta u.$$

* La multiplicación por escalares es asociativa.

$$(\alpha\beta)\,u = \alpha\,(\beta u)\,.$$

* Existencia del neutro multiplicativo del campo.

$$1u = u.$$

Para hablar de métricas sólo se necesitaba un conjunto X, pero si ese conjunto tiene una estructura de espacio vectorial, entonces se obtiene un tipo especial de métricas. Éstas se van a obtener a partir de las normas.

DEFINICIÓN 3. Sea V un espacio vectorial sobre $\mathbb{R}$. Una norma en V es una función $\|\cdot\|: V \to \mathbb{R}^+$, donde $\mathbb{R}^+ = [0, \infty)$. En este caso V cumple las propiedades siguientes:

$n1$) $\| u \| = 0$ si y solamente si $v = 0$.

$n2$) $\| \lambda v \| = |\lambda| \| v \|$, para toda $\lambda \in \mathbb{R}$.

$n3$) $\| v + w \| \leq \| v \| + \| w \|$, para toda v, w elementos de V.

Un espacio normado será un espacio vectorial V provisto por una norma $\|\cdot\|$ y lo denotaremos por $(V, \|\cdot\|)$. Cuando no sea necesario especificar la norma sólo lo denotaremos por $\|\cdot\|$.

Ahora se va a comprobar que un espacio normado es al mismo tiempo un espacio métrico donde la métrica depende de la norma [**8, 9**].

PROPOSICIÓN 4. *Todo espacio normado* $(V, \|\cdot\|)$ *es un espacio métrico con la distancia inducido por la* $\|\cdot\|$, *donde la* $d_{\|\cdot\|}(v, w) = \| v - w \|$.

DEMOSTRACIÓN. Sean u, v y w elementos de V y λ un escalar de $\mathbb{R}^+$.

Primero se demostrará que

$$d_{\|\cdot\|}(v, w) = 0 \quad si \quad y \quad solamente \quad si \quad \| v - w \| = 0.$$

Si

$$d_{\|\cdot\|}(v, w) = 0,$$

si y sólo si

$$\| v - w \| = 0,$$

por ser norma

$$v - w = 0,$$

si y solamente si

$$v = w.$$

Continuando con la demostración, ahora se mostrará que $\| \lambda v \| = |\lambda| \| v \|$, para toda $\lambda \in \mathbb{R}^+$.

Por demostrar que $d_{\|\cdot\|}(v, w) = d_{\|\cdot\|}(w, v)$.

Por definición se sabe que

$$d_{\|\cdot\|}(v, w) = \| v - w \|,$$

utilizando $n2$) se obtiene lo siguiente

$$\begin{aligned} |-1| \parallel v-w \parallel &= \parallel w-v \parallel \\ &= d_{\|\cdot\|}(w,v). \end{aligned}$$

Para finalizar se demostrará que $d_{\|\cdot\|}(v.w) \leq d_{\|\cdot\|}(v,u) + d_{\|\cdot\|}(u,w)$.

Por definición de $d_{\|\cdot\|}$

$$\begin{aligned} d_{\|\cdot\|}(v.w) &= \parallel v-w \parallel \\ &= \parallel v+(u-u)-w \parallel \\ &\leq \parallel v-u \parallel + \parallel u-w \parallel \\ &= d_{\|\cdot\|}(v,u) + d_{\|\cdot\|}(u,w). \end{aligned}$$

Por tanto se cumple $n3$). □

Al considerar los resultados anteriores, se puede concluir que $d_{\|\cdot\|}$ es una métrica inducida por la norma $\|\cdot\|$.

A continuación se van analizar las funciones distancia más utilizadas en diversas aplicaciones, para ello es necesario demostrar previamente unas desigualdades.

LEMA 5. *(Desigualdad de Young) Sean $p,q \in (1,\infty)$, tales que $\frac{1}{p}+\frac{1}{q}=1$, entonces para cualesquier par de números $a,b \geq 0$ se cumple que*

$$ab \leq \frac{1}{p}a^p + \frac{1}{q}b^q.$$

DEMOSTRACIÓN. Se puede apreciar que si a o b son 0 entonces la desigualdad es obvia. Por ello se va a suponer que a y b son distintos de cero.

Se sabe que la función exponencial es una función convexa, es decir, para cualesquiera x_0, x_1 reales y t_0, t_1 elementos de $[0,1]$ tal que $t_0+t_1=1$, se cumple que

$$e^{t_0x_0+t_1x_1} \leq t_0e^{x_0} + t_1e^{x_1}.$$

Si se considera a $x_0 = \ln a^p, x_1 = \ln b^q$, $t_0 = \frac{1}{p}$ y $t_1 = \frac{1}{q}$, se obtiene lo siguiente

$$e^{\left[\frac{1}{p}\ln a^p + \frac{1}{q}\ln b^q\right]} \leq \frac{1}{p}a^p + \frac{1}{q}b^q,$$

pero

$$ab = e^{\left[\frac{1}{p}\ln a^p + \frac{1}{q}\ln b^q\right]}.$$

Por tanto

$$ab \leq \frac{1}{p}a^p + \frac{1}{q}b^q.$$

□

Ahora se va a aplicar la desigualdad de Young para demostrar la desigualdad de Hölder.

LEMA 6. *(Desigualdad de Hölder) Sean p, q elementos de $(1, \infty)$, tal que $\frac{1}{p} + \frac{1}{q} = 1$, entonces para cualesquiera $x, y \in \mathbb{R}^n$, donde $x = (x_k)_{k=1}^n$ y $y = (y_k)_{k=1}^n$. Se cumple que*

$$\sum_{k=1}^{n} \mid x_k y_k \mid \leq \left(\sum_{i=1}^{n} \mid x_k \mid^p \right)^{\frac{1}{p}} \left(\sum_{k=1}^{n} \mid y_k \mid^q \right)^{\frac{1}{q}},$$

es decir,

$$\| xy \|_1 \leq \| x \|_p \| y \|_q$$

donde xy es el producto punto usual.

DEMOSTRACIÓN. Si se cumple que $xy = 0$, entonces la desigualdad es trivial. No obstante se va tomar en cuenta que a y b son distintos de cero.

Utilizando la desigualdad de Young para

$$a_k = \frac{\mid x_k \mid}{\| x \|_p} \text{y} b_k = \frac{\mid y_k \mid}{\| y \|_q},$$

se obtiene lo siguiente

$$\frac{\mid x_k y_k \mid}{\| x \|_p \| y \|_q} \leq \frac{\mid x_k \mid^p}{p \| x \|_p^p} + \frac{\mid x_k \mid^q}{q \| x \|_q^q}.$$

Al sumar estas desigualdades para $k = 1, \ldots, n$, se sigue que

$$\begin{aligned} \frac{1}{\| x \|_p \| y \|_q} \sum_{k=1}^{n} \mid x_k y_k \mid &\leq \frac{1}{p \| x \|_p^p} \left(\sum_{k=1}^{n} \mid x_k \mid^p \right) + \frac{1}{q \| x \|_q^q} \left(\sum_{k=1}^{n} \mid y_k \mid^p \right) \\ &= \frac{1}{p} + \frac{1}{q} \\ &= 1. \end{aligned}$$

Por lo tanto

$$\| xy \|_1 \leq \| x \|_p \| y \|_q .$$

□

Una vez que se han demostrado estas dos desigualdades, se demostrará que $\| \cdot \|_p$ es una norma.

PROPOSICIÓN 7. *Para todo $p \in [0, \infty)$ la función $\| \cdot \|_p$ es una norma en $\mathbb{R}^n$.*

DEMOSTRACIÓN. Para llevar a cabo la demostración se tomarán a x, y elementos de $\mathbb{R}^n$ y a $\lambda \in [0, \infty)$.

A continuación se demostrará $n1$), es decir,

$$\| x \|_p = 0 \text{ si y solamente si } x = 0.$$

Si

$$\begin{aligned} 0 &= \| x_p \| \\ &= \left(\sum_{i=1}^{n} | x_i |^p \right) \\ &= (| x_1 |^p + | x_2 |^p + \ldots + | x_n |^p) \end{aligned}$$

Pero se sabe que cualquier $| x_i |^p$ es positivo o cero y ademas si $| x |^{\frac{1}{p}} = 0$, entonces

$$x = 0,$$

por tanto

$$(| x_1 |^p + | x_2 |^p + \ldots + | x_n |^p) = (| 0_1 |^p + | 0_2 |^p + \ldots + | 0_n |^p),$$

si y solamente si, para toda $i \in \{1, 2, \ldots, n\}$

$$x_i = 0,$$

si y solamente si

$$x = 0,$$

Ahora se continuará con la demostración de $n2)$.

Para ello se demostrará que

$$\| \lambda v \| = |\lambda| \, \| v \| .$$

Se tiene que

$$\begin{aligned} \| \lambda x \|_p &= \left(\sum_{i=1}^{n} | \lambda x_i |^p \right) \\ &= \left(\sum_{i=1}^{n} | \lambda |^p \| x_i \|^p \right) \\ &= (| \lambda |^p) \left(\sum_{i=1}^{n} \| x_i \|^p \right)^{\frac{1}{p}} \\ &= | \lambda | \| x \|_p . \end{aligned}$$

Finalmente se demostrará que

$$\| x + y \|_p \leq \| x \|_p + \| y \|_p.$$

Si $x = 0$ la desigualdad es trivial, debido a ello se va a suponer que $x \neq 0$ y se va a aplicar la desigualdad de Hölder a x y

$$\left((\mid x_1 \mid + \mid y_1 \mid)^{p-1}, \ldots, (\mid x_n \mid + \mid y_n \mid)^{p-1}\right).$$

Definiendo a $q := \frac{p}{p-1}$ se obtiene

$$\sum_{k=1}^{n} \mid x_k \mid (\mid x_k \mid + \mid y_k \mid)^{p-1} \leq \| x \|_p \left(\sum_{k=1}^{n} (\mid x_k \mid + \mid y_k \mid)^p\right)^{\frac{1}{q}}.$$

Análogamente

$$\sum_{k=1}^{n} \mid y_k \mid (\mid x_k \mid + \mid y_k \mid)^{p-1} \leq \| y \|_p \left(\sum_{k=1}^{n} (\mid x_k \mid + \mid y_k \mid)^p\right)^{\frac{1}{q}}.$$

Al sumar las dos desigualdades anteriores se obtiene lo siguiente

$$\sum_{k=1}^{n} (\mid x_k \mid + \mid y_k \mid)^p \leq (\| x \|_p + \| y \|_p) \left(\sum_{k=1}^{n} (\mid x_k \mid + \mid y_k \mid)^p\right)^{\frac{1}{q}}.$$

Dividiendo entre

$$\left(\sum_{k=1}^{n} (\mid x_k \mid + \mid y_k \mid)^p\right)^{\frac{1}{q}}$$

y utilizando la desigualdad del triángulo para números reales se tiene que

$$\begin{aligned} \| x + y \|_p &= \left(\sum_{i=1}^{n} (\mid x_k \mid + \mid y_k \mid)^p\right)^{\frac{1}{q}} \\ &\leq \left(\sum_{k=1}^{n} (\mid x_k \mid + \mid y_k \mid)^p\right)^{\frac{1}{q}} \\ &\leq \| x \|_p + \| y \|_p . \end{aligned}$$

Con esto finalmente se ha terminado de demostrar la proposición. □

A continuación se van a mencionar las funciones distancia más utilizadas en este tipo de dominios.

Ejemplo 8. La norma $\| \cdot \|_1$ mejor conocida como *Distancia Manhattan*, es un ejemplo de las normas $\| \cdot \|_p$, ésta representa la distancia entre dos objetos mediante la suma de las diferencias absolutas coordenada a coordenada. Es muy común utilizar esta función en BD que almacenan elementos de diversas dimensiones, por ejemplo en aplicaciones de análisis forense, videojuegos, gps, entre otros.

Esta función distancia queda definida de la siguiente manera.

Sea V una base de datos cualquiera con objetos n-dimensionales, la función

$$d_1 = \| \cdot \|_1$$

FIGURA 2.1.2. Ejemplo de la utilidad que tiene la *Distancia Manhatan*.

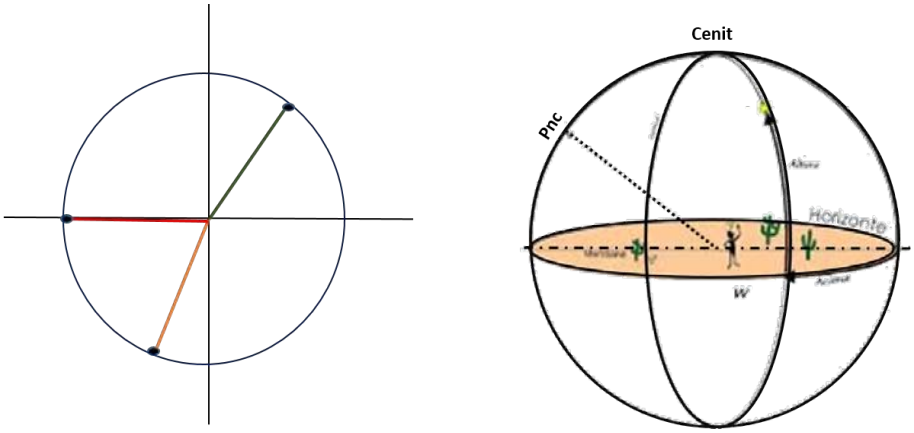


FIGURA 2.1.3. Uso de la Distancia Euclídea en proyecciones meteorológicas.

queda definida de la siguiente forma: para cualesquiera x, y elementos de $V = \mathbb{R}^n$, donde $x = (x_i)_{i=1}^n$ y $y = (y_i)_{i=1}^n$. Se cumple que

$$d_1(x, y) := \sum_{i=1}^{n} | x_i - y_i | .$$

En la Figura 2.1.2 se muestra uno de los usos que tiene esta métrica.

EJEMPLO 9. Otro ejemplo de las normas es la norma $\| \cdot \|_2$, también conocida como *Distancia Euclídea*, ésta corresponde a la noción de distancia que la mayoría de nosotros tiene. En la Figura 2.1.3 se muestra uno de los usos que se le puede dar a ésta métrica.

La *Distancia Euclídea* actualmente tiene una gran importancia en diversos aplicaciones de minería de datos, por ejemplo el análisis de la información recabada en redes sociales. Uno de los usos que tiene consiste en asignar escalas de puntuación a esa información para poder determinar comportamientos o tendencias, por ejemplo que tan valorados son los grados de amistad al nombrar a pocos o muchos amigos en una publicación, registros de frecuencias de interacción o simplemente para ver la similitud entre perfiles de los usuarios. También es bastante utilizada en aviación y estudio de comportamientos celestes, fenómenos naturales, entre otros.

La métrica $d_2 = \| \cdot \|_2$ va a quedar definida sobre $\mathbb{R}^n$, con n elemento de los $\mathbb{N}$ y

$$d_2(x, y) := \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2},$$

para cualesquiera x, y elementos de $\mathbb{R}^n$, donde $x = (x_k)_{k=1}^n$ y $y = (y_k)_{k=1}^n$.

Ejemplo 10. La *Distancia Máxima* o también conocida como *Distancia de Chebyshev,* consiste en obtener la diferencia entre dos vectores y tomar la mayor de ellas. Ésta queda definida de la siguiente manera: sea $V = \mathbb{R}^n$ un espacio vectorial y

$$\| \cdot \|_\infty (x, y) := max \{ | x_i - y_i | : i \in (1, 2, 3 \ldots, n) \},$$

con $x = (x_k)_{k=1}^n$ y $y = (y_k)_{k=1}^n$, elementos de $\mathbb{R}^n$. A continuación se va demostrar que $\| \cdot \|_\infty$ sea una métrica bien definida sobre V.

Se comenzará con demostrar que $\| \cdot \|_\infty (x, y) = 0$ si y solamente si $x = y$.

Primero se mostrará la ida. Por definición se tiene que

$$\| \cdot \|_\infty (x, y) = max \{ | x_i - y_i | : i \in (1, 2, 3 \ldots, n) \},$$

y se estará suponiendo que

$$max \{ | x_i - y_i | : i \in (1, 2, 3 \ldots, n) \} = 0,$$

de lo cual es inmediato que para toda i elemento de $\{1, 2, \ldots, n\}$

$$| x_i - y_i | = 0,$$

por tanto

$$x = y.$$

Ahora se demostrará el regreso. Para ello se supondrá que $x = y$, entonces

$$| x_i - y_i | = 0,$$

para toda $i \in \{1, 2, \ldots, n\}$. De lo cual se obtiene que

$$\| \cdot \|_\infty (x, y) = max \{| 0 |, | 0 |, \dots | 0 |\} = 0.$$

Se ha demostrado $m1$), se continuará con $m2$).

Por demostrar que $\| \cdot \|_\infty (x, y) = \| \cdot \|_\infty (y, x)$. Por definición se tiene que

$$\| \cdot \|_\infty (x, y) = max \{| x_i - y_i |, i \in (1, 2, \dots, n)\},$$

pero se sabe que para toda $i = 1, 2, \dots, n$

$$| x_i - y_i | = | y_i - x_i |,$$

entonces

$$max \{| x_1 - y_1 |, \dots, | x_n - y_n |\} = max \{| y_1 - x_1 |, \dots, | y_n - x_n |\}$$

de lo cual se tiene que

$$\| \cdot \|_\infty (x, y) = \| \cdot \|_\infty (y, x).$$

Quedó demostrado $m2)$.

Finalmente se mostrará $m3)$, es decir, $\| \cdot \|_\infty (x, y) \leq \| \cdot \|_\infty (x, z) + \| \cdot \|_\infty (z, y)$.

Por la desigualdad del triangulo se sabe que$| x_i - y_i | \leq | x_i - z_i | + | z_i - y_i |$ para toda i elemento de $\{1, 2, \dots, n\}$,

de lo anterior se tiene que

$$\begin{aligned}
\| \cdot \|_\infty (x, y) &= max \{| x_i - y_i | : i \in (1, 2, \dots, n)\} \\
&\leq max \{| x_i - z_i | + | z_i - y_i | : i \in (1, 2, \dots, n)\} \\
&= max \{| x_i - z_i | : i \in (1, 2, \dots, n)\} + max \{| z_i - y_i | : i \in (1, 2, \dots, n)\} \\
&= \| \cdot \|_\infty (x, z) + \| \cdot \|_\infty (z, y).
\end{aligned}$$

Se ha corroborado que $(\mathbb{R}^n, \| \cdot \|_\infty)$ es un espacio métrico.

Ejemplo 11. Métrica Discreta. Esta función es común utilizarla en bases de datos relacionales, ya que generalmente los objetos comparados son iguales. Ésta se define de la siguiente manera:

Dado un conjunto X cualquiera, se define la función d_{dis} de la siguiente manera:

$$d_{dis} (x, y) = \begin{cases} 0 & \text{si } x = y \\ 1 & \text{si } x \neq y. \end{cases}$$

A continuación se va a comprobar que la pareja (X, d_{dis}) sea un espacio métrico. Para ello se verificará que se cumplan las propiedades correspondientes.

En este caso la primera proposición se cumple por definición. Ahora se demostrará la segunda.

Por lo que se tiene que demostrar $d(v,w) = d(w,v)$.

Sean v y w elementos de X, por definición de d_{dis} se tiene que:

$$d_{dis}(v,w) = \begin{cases} 0 & \text{si } v = w \\ 1 & \text{si } v \neq w. \end{cases}$$

Suponiendo que

$$d_{dis}(v,w) = 0,$$

entonces

$$v = w,$$

si y solamente si

$$w = v,$$

lo cual implica que

$$d_{dis}(w,v) = 0.$$

Análogamente, si se supone que

$$d_{dis}(v,w) = 1,$$

entonces

$$v \neq \quad w.$$

Por lo tanto se cumple $m2)$. Ahora se continuará con la última propiedad.

Se van a tomar u, v y w elementos de X. Suponiendo que $v \neq w$, entonces $v \neq u$ ó $u \neq w$. Esto permite aseverar lo siguiente:

$$1 = d_{dis}(v,w) \leq d_{dis}(v,u) + d_{dis}(u,w).$$

Finalmente se ha demostrado que se cumplen $m1)$, $m2)$ y $m3)$. Por lo tanto (X, d_{dis}) es un espacio métrico.

Ejemplo 12. Distancia Hamming. Esta métrica es muy utilizada para comparar cadenas de caracteres de un alfabeto con la finalidad de obtener las diferencias que existen entre ellas, estas diferencias determinaran la distancia entre las dos cadenas. Para definir esta métrica primero definiremos un conjunto A llamado alfabeto, este conjunto puede contener números, letras, caracteres especiales, etcétera. Ahora se va a utilizar la métrica discreta d_{dis} donde

$$d_{dis} := A \times A \to [0,1].$$

Proposición 13. *Sea* $A^n = \prod_{i=1}^{n} A_i$ *y* μ *una función definida en* A^n, *donde*

$$\mu := A^n \times A^n \to \mathbb{R}^+,$$

con $\mathbb{R}^+ = [0, \infty)$ *y*

$$\mu(x, y) := \sum_{i=1}^{n} d_{dis}(x_i, y_i),$$

para cualesquiera $x = (x_i)_{i=1}^n$ *y* $y = (y_i)_{i=1}^n$ *elementos de* A^n. *La pareja* (A^n, μ) *es un espacio métrico.*

Demostración. Primero se va a demostrar $m1$).

$$\mu(x, y) \geq 0, \mu(x, y) = 0 \text{ si y solamente si } x = y.$$

Por definición se tiene

$$\mu(x, y) = \sum_{i=1}^{n} d_{dis}(x_i, y_i),$$

entonces

$$\sum_{i=1}^{n} d_{dis}(x_i, y_i) \geq 0.$$

Caso 1.

$$\sum_{i=1}^{n} d_{dis}(x_i, y_i) > 0,$$

si y sólo si

$$d_{dis}(x_i, y_i) > 0,$$

si y solamente si, para algún $i = 1, \ldots, n$

$$x_i \neq y_i.$$

Caso 2.

$$\sum_{i=1}^{n} d_{dis}(x_i, y_i) = 0,$$

si y sólo si

$$d_{dis}(x_i, y_i) = 0,$$

si y solamente si

$$x_i = y_i.$$

Para toda $i = 1, \ldots, n$.

Por lo que

$$x = y.$$

Por lo tanto

$$\mu(x, y) \geq 0.$$

A continuación se va a mostrar a $m2)$, es decir, $\mu(x, y) = \mu(y, x)$.

Por definición se tiene

$$\begin{aligned}\mu(x,y) =& \sum_{i=1}^{n} d_{dis}(x_i, y_i) \\ &= d_{dis}(x_1, y_1) + \ldots + d_{dis}(x_n, y_n) \\ &= d_{dis}(y_1, x_1) + \ldots + d_{dis}(y_n, x_n) \\ =& \sum_{i=1}^{n} d(y_i, x_i) \\ &= \mu(y,x).\end{aligned}$$

Por tanto también cumple $m2)$, finalmente se va a comprobar que cumpla $m3)$.

Nuevamente utilizando la definición se tiene que

$$\begin{aligned}\mu(x,y) =& \sum_{i=1}^{n} d(x_i, y_i) \\ \geq& \sum_{i=1}^{n} d_{dis}(x_i, z_i) + d(z_i, y_i) \\ =& \sum_{i=1}^{n} d(x_i, z_i) + \sum_{i=1}^{n} d_{dis}(z_i, y_i) \\ &= \mu(x,z) + \mu(z,y).\end{aligned}$$

De esta forma se puede concluir que $\mu(x,y) \leq \mu(x,z) + \mu(z,y)$. □

Se ha podido ver que (A^n, μ) cumple las tres propiedades correspondientes por lo que forman un espacio métrico.

2.2. Tipos de Búsquedas

Los métodos de búsqueda pueden varíar de acuerdo al SGBD y al tipo de datos que se almacenen en él, ya que depende de las características de estos y las necesidades de los usuarios. También se debe tomar en cuenta el uso y la manipulación, que se les da a los datos para mantener su integridad con la finalidad de obtener información lo más precisa posible. Los tipos de búsqueda que actualmente se utilizan son búsquedas exactas y por similitud.

2.2.1. Búsquedas Exactas.

Las BD tradicionales poseen una estructura completamente determinada, ya que almacenan conjuntos de información con características similares. En estas BD se manipulan tablas o colecciones, en las que sus registros son generalmente de tipo

numérico, de hora y/o fecha, alfanuméricos, booleanos (por ejemplo, 0 ó 1; Sí o No; etcétera.), entre otros.

Las características de estos bases permiten que puedan ser representadas mediante un modelo estructurado, el cual ha sido determinado con determinados estándares.

Debido a ello las búsquedas que se realizan comparan ciertos valores, los cuales deberán ser completamente iguales o en algunos casos parcialmente ya que se restringe la igualdad a unas posiciones determinadas. Por ejemplo, una búsqueda por fecha, puede ser exacta al tomar en cuenta día, mes y año o puede ser parcial si sólo se restringe por año.

2.2.2. Búsquedas por Similitud.

Cada vez se desarrollan más aplicaciones en las que los datos no estructurados tienen un crecimiento mayor que los datos estructurados, ya que es muy común ver aplicaciones que contienen datos no estructurados tales como imágenes, audio, video, documentos XML, entre otros. Debido a ello se ha tenido la necesidad de generar BD que puedan almacenar este tipo de objetos, sin embargo más allá de almacenarlos se pretende utilizarlos y manipularlos para convertirlos en información de valor para la toma de decisiones. Las búsquedas exactas en este tipo de objetos son complicadas pues no tienen una estructura determinada, por ello se han implementado métodos para obtener los objetos más similares o parecidos a uno dado, a partir de ahí surgen términos como búsqueda por similitud o búsqueda por proximidad. Es natural que a este tipo de problemáticas se les genere un modelo matemático para darles una solución formal y tratar de predecir futuras complicaciones.

En cuanto al escenario de las búsquedas por similitud el modelo matemático que mejor se adapta a las características dadas es una función que cumpla con las propiedades de una función distancia en un espacio métrico, ya que este modelo permite efectuar transformaciones de un espacio a otro de tal forma que la definición de métricas sea factible.

El resto de la tesis se va a centrar solamente en búsquedas por similitud, las cuales consisten en lo siguiente: Dado un espacio métrico (X, d), donde X es el conjunto que contiene a todos los objetos base y d una función distancia dada, así como $C \subset X$ tal que $\mid C \mid = k$. Si se proporciona un objeto de consulta q, se pretende proyectar los elementos de C más cercanos a q.

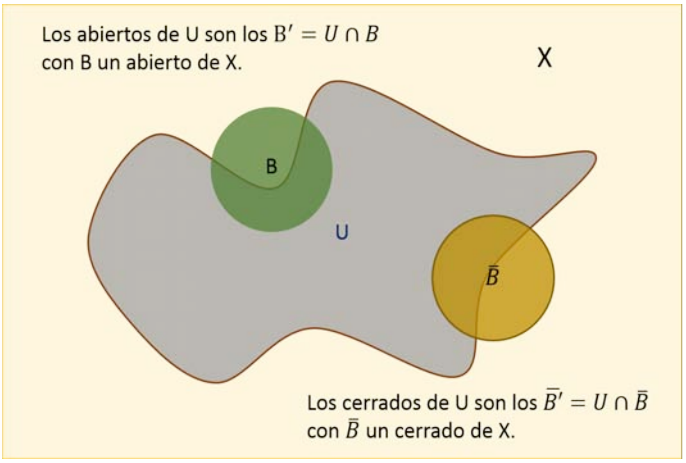


FIGURA 2.2.1. Bola abierta y cerrada en $U \subset X$.

Es necesario formalizar la idea anterior, así que se establecerá una terminología para los conceptos topológicos que se utilizarán a lo largo de este trabajo. Por esto es importante mencionar la definición de *"bolas"* en espacios métricos [**8, 9**].

DEFINICIÓN 14. Sea (X, d) un espacio métrico, dado q elemento de X y $r > 0$, se define la bola abierta centrada en q de radio r (que se denotará por $B(q, r)$) como:

$$B(q, r) := \{y \in X : d(q, y) < r\}.$$

La Figura 2.2.1, muestra una representación de la definición de bola abierta.

DEFINICIÓN 15. Sea (X, d) un espacio métrico, dado q elemento de X y $r > 0$, se define la bola cerrada centrada en q de radio r (que se denotará por $\overline{B}(q, r)$) como:

$$\overline{B}(q, r) := \{y \in X : d(q, y) \leq r\}.$$

La Figura 2.2.1, muestra una representación de la definición de bola cerrada.

Se van a estudiar dos tipos de búsquedas por similitud: las búsquedas por rango y las búsquedas por los k-Vecinos más cercanos, existen algunas variantes de ellas pero prácticamente pertenecen a uno de los dos enfoques antes mencionados.

2.2.2.1. Búsquedas por Rango. En este criterio se pretende proyectar los x elementos de U que tienen un rango de distancia menor o igual a r de un objeto de búsqueda q, a partir de una función distancia definida d. Es decir,

$$\overline{B}(q, r) := \{x \epsilon U : d(x, q) \leq r\},$$

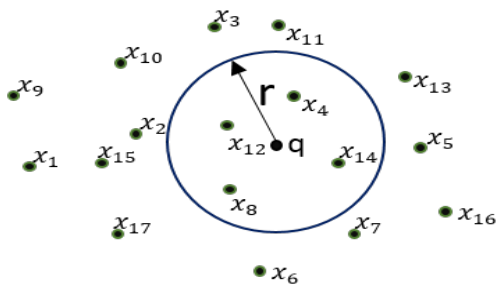


FIGURA 2.2.2. Representación de una búsqueda por rango.

en este tipo de búsquedas se desea recuperar los elementos de una base de datos que están a una distancia no mayor que cierto radio de tolerancia a un elemento de consulta dado.

Cabe mencionar que mientras menos restrictivas sean las características de q, más grande será la longitud del radio y esto provocará una mayor dificultar al momento de realizar la búsqueda. La Figura 2.2.2 muestra un ejemplo de este tipo de búsqueda.

Si se tuviera el caso en el que $q \in U$, entonces se estaría obteniendo la búsqueda exacta sobre q.

En muchas aplicaciones la búsqueda por rango no es muy útil porque hay veces que no hay elementos dentro de un radio dado y estar haciendo búsquedas a "prueba y error" de tal forma que se incremente cada vez más el radio, no es factible, pues se consumen recursos computacionales sin obtener resultados favorables. Por ello en muchas aplicaciones se busca obtener la proyección de los vecinos más cercanos a un objeto q dado.

2.2.2.2. Búsquedas de los k-Vecinos más Cercanos (kNN). Este tipo de búsqueda puede ser una muy buena alternativa al problema planteado previamente, pues en ésta se pretende proyectar al subconjunto U con los k elementos más cercanos (NN por sus siglas en inglés de Nearest Neighbors) a un objeto q dado. Éste se puede definir con la siguiente función

$$kNN : X \to P(U).$$

Donde si

$$x \notin kNN(q),$$

entonces

$$d(q, x) \geq d(q, y),$$

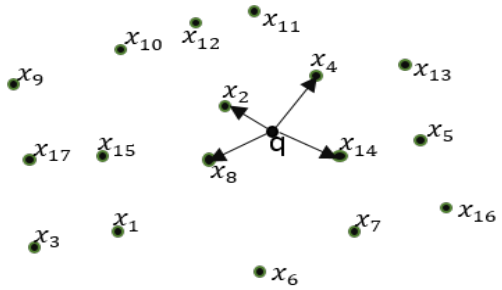


FIGURA 2.2.3. Ejemplo de una búsqueda de los 4-vecinos más cercanos a q.

para toda $y \in kNN\,(q)$ y $|kNN\,(q)\,| \leq k$.

Un ejemplo de este tipo de búsqueda es el que se muestra en la Figura 2.2.3.

Si el conjunto X donde se está realizando la búsqueda tiene menos de k objetos, la consulta proyectará toda la base de datos.

Cabe mencionar que cuando m objetos se encuentran a la misma distancia de q, la búsqueda se resolverá de manera arbitraria.

2.2.2.3. Búsqueda del Vecino más Cercano. Este tipo de búsqueda puede ser un caso particular del anterior, ya que en él sólo se busca obtener la proyección del vecino x más cercano a un objeto q dado, es decir,

$$NN(q) = x.$$

Donde

$$NN(q) = \{x \in X : \forall y \epsilon X, d(q,x) \leq d(q,y)\}\,.$$

Si hubieran varios elementos a la misma distancia, de igual forma que la anterior, se elegiría uno de manera arbitraria.

2.2.2.4. Búsqueda Inversa de los k-Vecinos más Cercanos. En diversas aplicaciones se necesita saber cómo se comporta el objeto de consulta q con respecto a otros objetos del conjunto de datos. Aquí se van a proyectar los objetos que contengan a q entre sus k vecinos más cercanos (RNN por sus siglas en inglés de Reverse Nearest Neighbors). Este tipo de búsqueda quedará definida de la siguiente forma:

$$kRNN\,(q) \subset X,$$

de tal manera que

$$x \in kRNN\,(q)\,.$$

En la Figura 2.2.4 se hace una representación de este tipo de búsquedas.

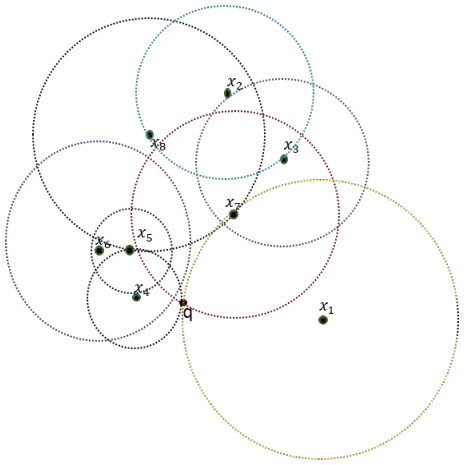


FIGURA 2.2.4. Ejemplo de la búsqueda inversa de los 2-vecinos más cercanos a q.

Este cambio por ligero que parezca puede alterar mucho la proyección. Por ejemplo, puede haber un elemento p cercano al objeto de búsqueda q pero puede ser que p no está en el conjunto de la búsqueda inversa, porque existen otros objetos más cercanos a él de lo que está q.

2.2.2.5. Búsqueda Join por Similitud. En la actualidad han ido evolucionando los servicios de la Internet y con ellos los diferentes medios que se tienen para acceder a esos servicios. Por ello en diversas aplicaciones se requiere integrar bases de datos de distinta naturaleza en un sólo destino. Hasta el momento se ha visto que muchas de ellas contienen datos no estructurados mientras que los servicios a los que son destinados a menudo requieren datos estructurados. El objetivo principal es proporcionar datos consistentes y sin errores, lo que implica la limpieza de datos, regularmente implementado por una especie de unión de similitudes.

Los objetos que se desean proyectar en este tipo de búsquedas quedan definidos con la siguiente función

$$J : P(x) \setminus \{\emptyset\} \times P(x) \setminus \{\emptyset\} \to P(x),$$

tal que

$$J(A, B, \varepsilon) := \{(a, b) \in A \times B : d(a, b) \leq \varepsilon\},$$

donde $A \subseteq X$, $B \subseteq X$ y $0 \leq \varepsilon$.

Si llegara a ocurrir que $A = B$, entonces se tiene el caso de Búsqueda Autojoin por Similitud, éste se denota por $AJ(\varepsilon) := J(A, A, \varepsilon)$, donde

$$AJ(\varepsilon) := \{(a, b) \in A \times A : d(a, b) \leq \varepsilon\},$$

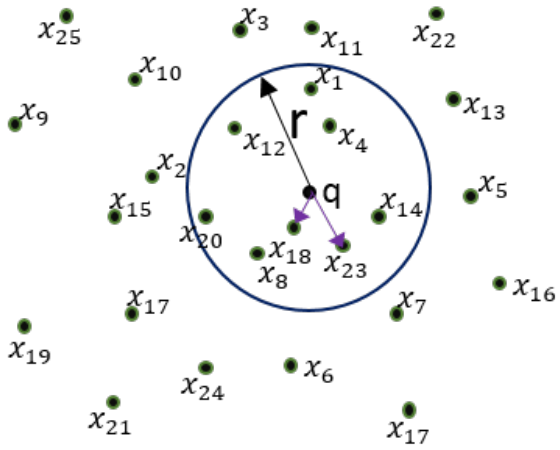


FIGURA 2.2.5. Ejemplo de la combinación de una búsqueda, por rango y los 2-vecinos más cercanos a q.

con $A \subseteq X$ y $0 \leq \varepsilon$.

Este tipo de búsquedas se utiliza constantemente en minería de datos para encontrar objetos duplicados.

2.2.2.6. Combinación de Búsquedas. Pueden definirse tipos de búsquedas adicionales a partir de la combinación de las anteriores. Por ejemplo, se puede combinar la Búsqueda por Rango con la búsqueda de los k Vecinos más Cercanos y el nuevo tipo de búsqueda quedaría así,

$$kNN\left(q,r\right) = kNN\left(q\right) \cap B\left(q,r\right).$$

En la Figura 2.2.5 se hace una representación de este tipo de búsquedas.

Análogamente se pueden combinar Búsqueda Inversa de los k-Vecinos más Cercanos con Búsqueda por Rango. No hay límite de combinaciones, ya que la combinación correcta de éstas las convierte en una gran herramienta para los desarrolladores y administradores de las *BDM*.

2.2.3. Estrategias Utilizadas en Búsquedas por Similitud.

Hasta este momento ya se ha visto que en la búsqueda por similitud se pretende obtener la proyección de elementos que no son exactamente iguales al objeto de búsqueda. La forma en la que se determina este tipo de búsquedas tiene un gran

parecido a como lo hacemos los humanos, al reconocer la similitud entre objetos mediante sus características fundamentales. Es decir, primero se resaltan un conjunto de características relevantes y después se les trata de reconocer de acuerdo a un contexto. En cuanto a los espacios métricos primero convierten esas características en valores concretos mediante el uso de una función distancia y después se realiza una evaluación para proyectar los elementos que cumplan con ciertos criterios definidos durante la consulta.

Al pensar en la manera de implementar la búsqueda por similitud lo natural sería que se comparara el objeto de búsqueda con todos los de la base, sin embargo hacer todo el recorrido generaría un uso de recursos computacionales demasiado alto, si a esto le agregamos que algunas veces se trabaja con bases de gran tamaño es fácil ver que ésta no es la mejor opción. Debido a este tipo de inconvenientes, ha sido necesario crear métodos de búsqueda en espacios métricos en donde se reduce la mayor cantidad posible de evaluaciones mediante una métrica.

El objetivo principal de estos métodos es realizar un postprocesamiento de la colección mediante la construcción de un índice el cual contendrá datos que se utilizarán para el proceso de búsqueda.

Cabe mencionar que la utilización de un índice gasta recursos computacionales durante la construcción del mismo y en su almacenamiento. Este consumo de recursos se lleva a cabo antes del proceso de consulta, lo cual genera un costo inicial que generalmente los métodos de indexación no toman en cuenta durante el proceso de búsqueda.

Durante la consulta, que conlleva recorrer el índice, se determina qué elementos no deben compararse con el objeto de búsqueda y al final sólo se evalúa una colección de elementos seleccionados.

2.3. Índices Métricos

Todas las propuestas de indexación se basan en obtener un subconjunto de objetos relevantes de la base de datos, el cual comúnmente se genera mediante procesos heurísticos. Este subconjunto de datos representa la totalidad del espacio con el propósito de que la búsqueda pueda resolverse sin que sea necesario calcular un número elevado de distancias. Este planteamiento ha originado diferentes algoritmos que pueden clasificarse de acuerdo a sus características, sin embargo en este trabajo se estudiarán más a detalle los basados en *pivotes* y en *particiones compactas*. A continuación se muestra la clasificación de los métodos, basada de acuerdo al tipo de indexación.

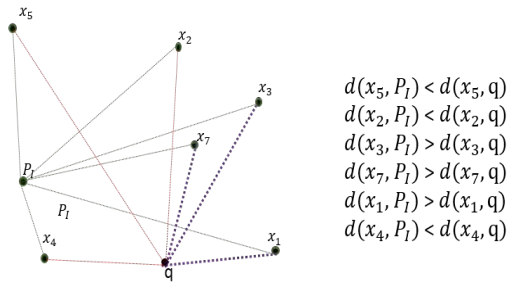


FIGURA 2.3.1. Ejemplo de una indexación basada en pivotes.

2.3.1. Indexación Basada en Pivotes.

La idea general de crear un índice basado en pivotes consiste en elegir un conjunto U de k elementos llamados "*pivotes*", donde $U = \{p_1, p_2, \ldots, p_k\}$ y almacenar para x elemento de X, su distancia a los k pivotes $\{d(x, p_1), \ldots, d(x, p_k)\}$. Dada una consulta $R = (q, r)$, se calculan las distancias de q a los elementos de U, es decir, $\{d(q, p_1), \ldots, d(q, p_k)\}$. Si para algún pivote p_i se tiene que $d(q, p_i) < d(x, p_i)$, entonces esos objetos x serán descartados y no se evaluarán en la consulta. Todos los demás elementos que no se descarten deben ser evaluados, tal como se representa en la Figura 2.3.1.

Como ya se tienen calculadas y almacenadas las distancias, al momento de generar una consulta sobre un objeto de búsqueda q, la distancia $d(q, p_i)$ siempre estará acotada inferior y superiormente.

LEMA 16. *Dado un espacio métrico (X, d) y tres objetos arbitrarios o, p y q elementos de X. Se puede asegurar que*

$$| d(q, o) - d(o, p) | \leq | d(q, p) | \leq d(q, o) + d(o, p)$$

En consecuencia, la distancia $d(q, p)$ esta acotada inferior y superiormente, siempre y cuando se conozcan las distancias de $d(q, o)$ y $d(o, p)$.

DEMOSTRACIÓN. Se va a comenzar por la cota inferior, para ello se demostrará que

$$| d(q, o) - d(o, p) | \leq d(q, p).$$

Por la desigualdad del triángulo se puede asegurar que

$$d(o,p) \leq d(o,q) + d(q,p)$$
$$d(o,q) \leq d(o,p) + d(p,q).$$

entonces

$$d(o,p) - d(o,q) \leq d(q,p)$$
$$d(o,q) - d(o,p) \leq d(p,q) = d(q,p).$$

Al tomar en cuenta las dos desigualdades se concluye que

$$| d(o,p) - d(o,q) | \leq d(q,p).$$

En cuanto a la cota superior, se cumple por la desigualdad del triángulo. □

La forma de elegir los k pivotes y el número de ellos, son dos de los criterios más importantes para que este tipo de indexación sea efectivo. Por un lado la elección de los pivotes es de suma importancia ya que en esta parte se determina la zona conjunta en la que no deberán descartarse elementos. Por otro lado, el número de pivotes, va a indicar la cantidad de evaluaciones que se deben realizar al objeto de búsqueda, es claro que sólo algunos de ellos serán efectivos, sin embargo al tomarlos en cuenta a todos hará que se tenga la cobertura suficiente. En algunos casos, si llegará a ocurrir que los pivotes que para algún tipo de consulta no son relevantes para otras si lo son.

Es importante mencionar que este tipo de indexación asume que todas las distancias entre los objetos de la base de datos y los pivotes son conocidas, sin embargo algunas estructuras necesitan optimizar los recursos al construir el índice, como el almacenamiento de esa cantidad de datos no es aceptable se necesita buscar una alternativa. La más aceptable es almacenar únicamente un intervalo de distancia en donde los objetos de la base de datos se identificarán con un p_i.

A continuación se verá en qué consiste.

2.3.2. Indexación Basada en Particiones Compactas.

En este tipo de algoritmos se divide al espacio en zonas tan compactas como sea posible. La idea general consiste en seleccionar un conjunto de k objetos, $U = \{c_1, c_2, \ldots, c_k\}$ y dividir el espacio entre k regiones.

Los c_k serán llamados *"centros",* donde a cada zona R_k le corresponde un c_k. Éstas quedan definidas de la siguiente manera

$$R_k := \{x \in X : d(c_k, x) \leq d(c_j, x)\}.$$

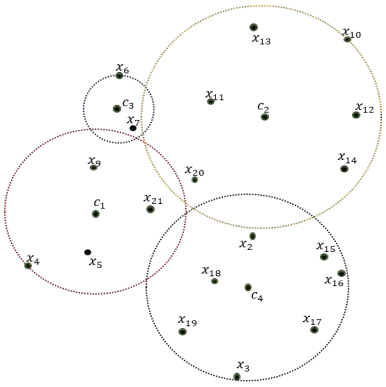


FIGURA 2.3.2. Representación de la división de una base de datos en zonas esféricas.

De esta manera el espacio se distribuye al asignar a cada región R_k los elementos más cercanos a su centro. El particionamiento se realiza hasta que ya no sea posible dividir el espacio. Dependiendo del proceso de búsqueda, estos algoritmos de indexación se dividen en dos criterios *Radio de Cobertura* e *Hiperplanos*.

2.3.2.1. Indexación que utiliza Radio de Cobertura. Los algoritmos que utilizan *Radio de Cobertura* o también llamados *Zonas Esféricas*, en estos se utiliza un radio r_c el cual toma en cuenta la distancia máxima que hay del centro a otro objeto de la misma zona. Como se muestra en la Figura 2.3.2.

Para el otro criterio es necesario conocer la siguientes definiciones.

DEFINICIÓN 17. Sean $x = (x_i)_{i=1}^n$ y α elementos de $\mathbb{R}^n$ y $\mathbb{R}$ respectivamente. Se llama *Hiperplano* al conjunto

$$\mathcal{H} := \left\{ y \in \mathbb{R}^n : x^T y = \alpha, x^T \neq 0 \right\},$$

con $x \in \mathbb{R}^n$ y x^T es transpuesta de x.

Un *Hiperplano* $\mathcal{H}$ divide al espacio en dos partes llamados subespacios, estos se definen como

$$\mathcal{H}^- := \left\{ y \in \mathbb{R}^n : x^T y \leq \alpha \right\}$$

y

$$\mathcal{H}^+ := \left\{ y \in \mathbb{R}^n : x^T y > \alpha \right\}.$$

La siguiente definición sólo se utilizará cuando se trabaje con espacios vectoriales.

DEFINICIÓN 18. Sean d una función distancia y $C = \{c_1, \ldots, c_n\}$ un conjunto de elementos de $X = \mathbb{R}^n$, los cuales son llamados centros. Una *Región de Voronoi* se define como

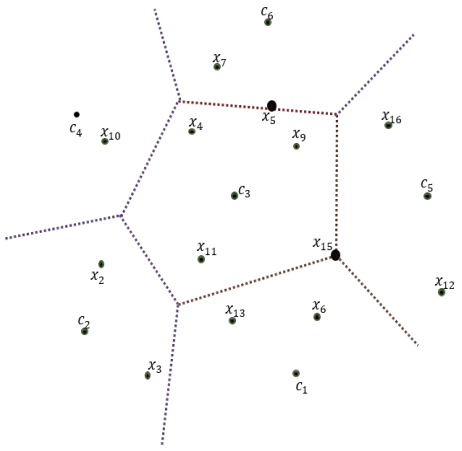


FIGURA 2.3.3. Ejemplo de un objeto situado sobre un borde y de otro situado sobre un vértice.

$$R_i := \{x \epsilon X : d(x, c_i) \leq d(x, c_j), i \neq j, (i, j = 1, 2, 3 \ldots, n)\},$$

y el *Borde de una Región* queda determinado por

$$B_{ij} := \{x \epsilon X : d(x, c_i) = d(x, c_j), i \neq j, (i, j = 1, 2, 3 \ldots, n)\}.$$

Se llama *Diagrama de Voronoi* al conjunto

$$D_v := \bigcup_{i,j=1}^{n} B_{ij} \text{con} i \neq j.$$

En la Figura 2.3.3 se representa el caso donde un objeto se encuentra en el Borde de una Región y cuando se encuentra en un vértice, es decir, cuando un objeto pertenece a tres o más regiones.

2.3.2.2. Indexación que utiliza Hiperplanos. Los procedimientos de indexación que están basados en *Hiperplanos*, eligen k objetos c_k, siendo cada uno el centro de una región R_k. Las R_k contienen a los objetos que están a una distancia de c_k menor que los centros de las zonas restantes. Debido a la distribución que tienen las regiones se les puede asociar con el modelo matemático llamado "*Las regiones de Voronoi*". La Figura 2.3.4 muestra una representación de como se dividiría una base utilizando regiones de Voronoi.

Independientemente del método de particionado que se utilice, se tienen k centros y R_k regiones con centro. Al momento de construir un índice se guarda la información de cada zona, es decir, el centro c_k, el radio de cobertura r_c o ambos.

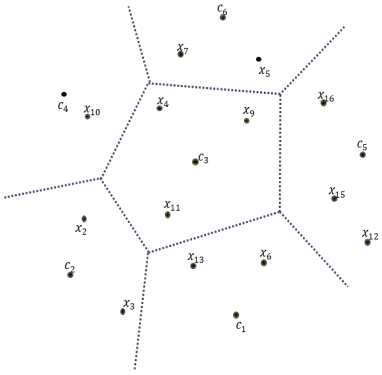


FIGURA 2.3.4. Ejemplo de una base de datos dividida en regiones de Voronoi.

Dada una consulta por rango $R(q,r)$ sobre un objeto de búsqueda q y un radio r, primero se calculan las distancias del objeto q a cada uno de los centros c_k y posteriormente se obtienen las siguientes diferencias

$$d\left(q,c_k\right) - r_{c_k} > r,$$

otra forma de escribirlas es

$$d\left(q,c_k\right) - r > r_{c_k}.$$

Donde r_{c_k} es el radio correspondiente a una R_k.

Se realizan estas operaciones con la finalidad de descartar algunas regiones y de esta manera evitar la comparación del objeto de búsqueda q con los elementos de las mismas. Considerando esta idea se puede observar que si existe alguna intersección entre la región $R_i\left(c_i,r_i\right)$ y $R\left(q,r\right)$, entonces se van a evaluar los objetos de dicha región, en caso contrario se descartarán.

El Lema 19 formaliza esta idea al utilizar zonas esféricas.

LEMA 19. *Sea (X,d) un espacio métrico, $R_i\left(c_i,r_i\right)$ una región que pertenece a X y p elemento de R_i es decir, $d\left(c_i,p\right) \leq r_i$. Dado $q \in X$ se tiene que:*

$$máx\left\{d\left(q,c_i\right) - r_i, 0\right\} \leq d\left(q,p\right) \leq d\left(q,c_i\right) + r_i.$$

DEMOSTRACIÓN. Por la desigualdad del triángulo y por hipótesis se sabe que

$$\begin{aligned} d\left(q,c_i\right) \leq & d\left(q,p\right) + d\left(p,c_i\right) \\ d\left(p,c_i\right) \leq & r_i. \end{aligned}$$

Usando estas dos ecuaciones se tiene que

$$d\left(q,c_i\right) - d\left(q,p\right) \leq d\left(p,c_i\right) \leq r_i,$$

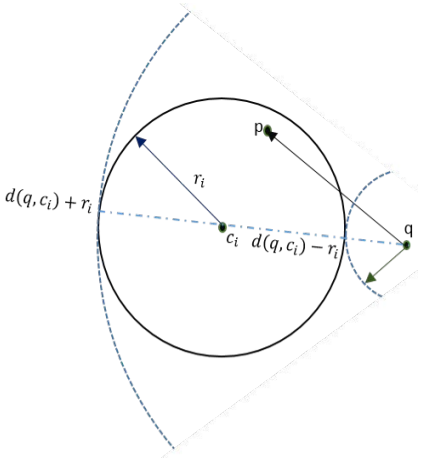


FIGURA 2.3.5. Representación del *Lema 1.14*.

entonces

$$d(q,c_i) - r_i \leq d(q,p).$$

Por otro lado se sabe que

$$d(q,p) \geq 0,$$

entonces el límite inferior es

$$máx\{d(q,p) - r_c, 0\} \leq d(q,o).$$

En el otro límite se tiene que

$$d(q,p) \leq d(q,c_i) + r_i$$

Por la desigualdad del triángulo se cumple lo siguiente

$$d(q,p) \leq d(q,c_i) + d(c_i,p),$$

por la desigualdad anterior y por hipótesis se concluye que

$$d(q,p) \leq d(q,c_i) + r_i.$$

□

En la Figura 2.3.5 se muestra un ejemplo del Lema 19.

Luego de ver en qué consisten los criterios de indexación que actualmente se utilizan, se realizará un análisis de los algoritmos de indexación más relevantes. Debido a que la mayoría de los algoritmos utilizan árboles para modelar la construcción del índice o el procedimiento de búsqueda, es relevante tener los conocimientos necesarios para hacer las representaciones correspondientes de los algoritmos.

2.4. Teoría de Gráficas

Los gráficas constituyen una herramienta muy importante para el modelado, diseño y desarrollo de fenómenos discretos, por lo que se ha convertido en un recurso elemental para comprender las estructuras de datos y el análisis de algoritmos para el tratamiento de los mismos [**10, 11**]. Por ello a continuación se definirán los conceptos necesarios para la construcción de estos modelos.

Definición 20. Sea V un conjunto no vacío y $A \subset V \times V$. Se llama *gráfica* al par $G = (V, A)$, donde V está formado por un conjunto de puntos, llamados *vértices* o *nodos* y A es un conjunto de aristas. Donde para todo $a \in A$ se cumple que

$$a = (p, q)\,,$$

donde p y q son elementos de V y $p \neq q$. Se dice que dos vértices $p, q \in V$ son adyacentes si $(p, q) \in A$.

Se dice que la gráfica es dirigida si nos importa el orden de los elementos de las aristas de A. Si $(p, q) \in A$ se dice que la arista parte de p y termina en q. Cuando las aristas de G tienen una dirección u orientación definida, se les denomina *gráficas dirigidas* o *digráficas.* A las aristas de una digráfica se les llama *flechas*, y a cada una de éstas le corresponde un nodo de salida y otro de llegada, por tanto $(p, q) \neq (q, p)$.

Para determinar la cantidad de aristas o flechas de cada nodo se definirá el siguiente concepto.

Definición 21. El *grado* de un nodo p, denotado por $\delta(p)$, es

$$C := \{(x, y) \in A : x = p \acute{o} y = p\}\,.$$

Si G es una digráfica entonces se calcula el grado de salida y grado de entrada, para cada nodo. Estos se denotan por δ_S y δ_E respectivamente. Donde

$$\delta_S = \{(x, y) \in A : x = p\}$$

y

$$\delta_E = \{(x, y) \in A : y = p\}\,.$$

En la Figura 2.4 se muestra un ejemplo de una gráfica y una digráfica.

Cuando el grado de un nodo es cero, entonces no está conectado a alguna arista, a este tipo de nodos se les llama *nodos aislado*s. Estos nodos representan casos particulares o exclusiones de algún algoritmo o procedimiento, por ello es recomendable

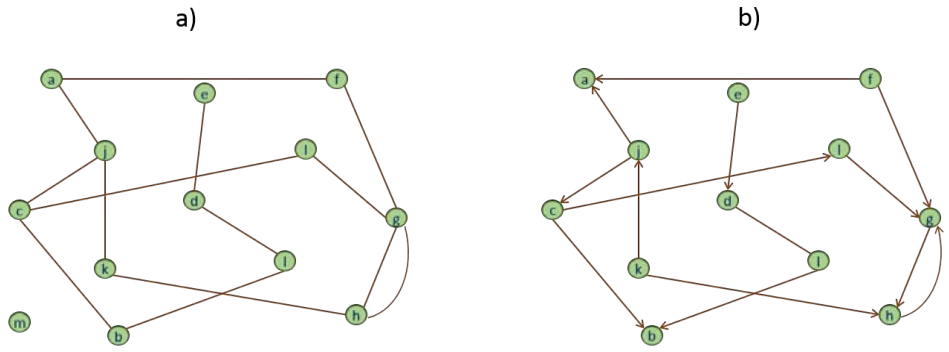


FIGURA 2.4.1. La imagen a) representa una gráfica y la b) una digráfica.

evitar tenerlos en el modelo. Debido a esto se requiere que exista una trayectoria de un nodo a otro, la siguiente definición formalizará esta idea.

DEFINICIÓN 22. Un *camino* C en una gráfica G es una sucesión

$$C = \{v_0, a_1, v_1, \ldots, v_{r-1}, a_r, v_r\},$$

tal que cada arista a_i es incidente a los nodos v_{i-1} y v_i.

Si $a_i \neq a_j$, entonces C es un *camino simple*. Por otro lado cuando hay un camino $C \subset G$ que comienza en p y termina en q, entonces los nodos p y q son conexos. Para generalizar esta idea en toda la gráfica$G$ se definirá lo siguiente.

DEFINICIÓN 23. Se le llama *gráfica conexa* a toda aquella en la que para cualquier par de vértices (p, q) existe al menos un camino de p hacia q.

Si un camino tiene al inicio y al final el mismo nodo, es decir, $p = q$ entonces es un *circuito*. Hay que notar que en un circuito se pueden repetir aristas y esto no permite modelar un algoritmo eficiente, la siguiente definición restringirá esta situación.

DEFINICIÓN 24. Un circuito que no repite ninguna arista se llama *ciclo*.

Para garantizar un modelado, diseño e implementación eficientes es importante evitar repeticiones de aristas o nodos, las siguientes gráficas permitirán cumplir con estos requisitos.

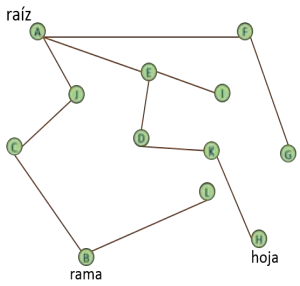


FIGURA 2.4.2. Representación de un árbol.

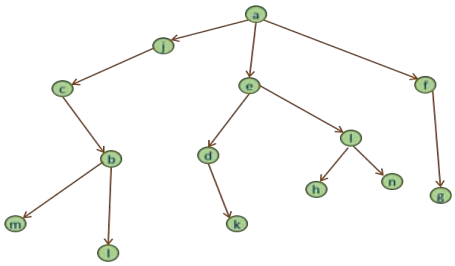


FIGURA 2.4.3. Arborescencia con raíz a.

DEFINICIÓN 25. Un *árbol* $B = (V, A)$ es una gráfica conexa sin ciclos, a los nodos de B de grado 1 se les llama *hojas* y a los que tienen un grado mayor que 1 se les denomina *ramas*. La figura 2.4.2, muestra el ejemplo de un árbol no dirigido.

DEFINICIÓN 26. Una *arborescencia*, es un árbol dirigido con un nodo llamado raíz, de tal forma que sólo existe un camino de la raíz a cualquier otro nodo del árbol. Ese camino no repite vértices ni aristas y respeta la dirección de las flechas. La Figura 2.4.3 muestra el ejemplo de una arborescencia.

Además si $v_{i-1}, v_i \in V$ y $(v_{i-1}, v_i) \in A$, entonces a v_{i-1} se le denomina el padre de v_i y a v_i se le denomina el hijo de v_{i-1}. En el ejemplo de la Figura 2.4.3, l y m son hijos de b y d es el padre de k.

PROPOSICIÓN 27. *Si B es una arborescencia, entonces contiene un único nodo con grado de entrada* 0, *al cual se le llama* raíz.

DEMOSTRACIÓN. Primero se demostrará la existencia de un nodo $r_0 \in B$, tal que $\delta_E(r_0) = 0$ y posteriormente su unicidad.

Sea r_0 el nodo raíz, se va a suponer que $\delta_E(r_0) = 1$. Entonces, eso implica que existe $(z, r_0) \in A$.

Sea

$$T = \{r_0, (r_0, r_1), r_1, \ldots, r_n = z\}$$

una trayectoria.

Por la existencia de (z, r_0) se tiene que

$$T = \{r_0, (r_0, r_1), r_1, \ldots, r_n = z, (z, r_0), r_0\}$$

es un ciclo.

Lo cual es una contradicción, pues B es un árbol.

Se ha comprobado la existencia de un nodo raíz $r \in B$ que tiene $\delta_E(r) = 0$. Para continuar con la demostración ahora se demostrará la unicidad de éste.

Se supondrá que existen r_0 y r_1 elementos de V, tal que son raíces de B. Es decir,

$$\delta_E(r_0) = 0$$

y

$$\delta_E(r_1) = 0.$$

Esto implica que no existe $a \in A$, tal que $(a, r_0) \in A$ o $(a, r_1) \in A$.

Dado que B es arborescencia entonces existe un sólo camino de r_0 a cualquier otro $v_i \in V$. Como $r_1 \in V$, existe un sólo camino de r_0 a r_1 entonces existe $a \in A$, tal que $(a, r_1) \in A$.

Por lo tanto

$$\delta_E(r_1) \neq 0.$$

Lo cual es una contradicción con la suposición inicial.

Análogamente ocurriría con r_0.

Entonces sólo unos de los dos nodos propuestos es raíz o $r_0 = r_1$.

Por lo tanto sólo existe un nodo raíz r en B, tal que $\delta_E(r) = 0$. □

2.5. Descripción de Algoritmos Existentes en Espacios Métricos

Los algoritmos para la búsqueda por similitud en espacios métricos aprovechan las características de las BDM, ya que en ellas se pueden proponer las funciones distancia que mejor se adapten a la base. Esta ventaja permite que los algoritmos se puedan modificar, debido a que el objetivo principal de su implementación es

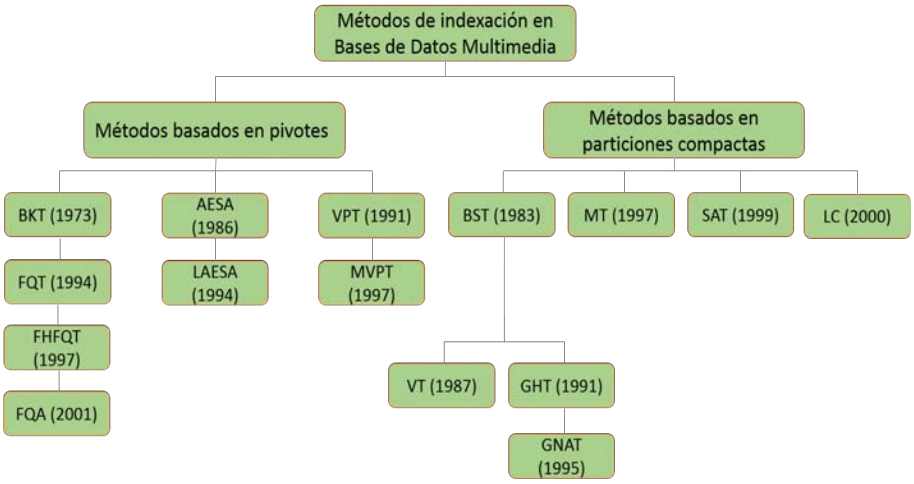


FIGURA 2.5.1. Algoritmos de indexación en espacios métricos más relevantes.

mejorar el manejo de la BD y disminuir los costos computacionales sin afectar los resultados de la consulta.

En esta sección se van a analizar los algoritmos más destacados en búsquedas por similitud sobre espacios métricos, se han excluido algunas variantes de ellos ya que las modificaciones que presentan han sido para adecuarlas al ambiente en el que se implementaron. Debido a la diversidad de los algoritmos se van a clasificar de a cuerdo al tipo de indexación en la que se hayan basado. La Figura 2.5.1 muestra un esquema que contiene los algoritmos que se van a analizar en esta sección.

2.5.1. Algoritmos de Indexación Basado en Pivotes.

En este criterio de indexación, primero se calculará la distancia para cada objeto de la base de datos a los pivotes establecidos, después se calculará la distancia del objeto de búsqueda a los pivotes, posteriormente se descartarán aquellos objetos que no cumplan con la distancia requerida para finalmente evaluar los objetos restantes. Los siguientes métodos son los más destacados en cuanto a este tipo de indexación.

2.5.1.1. Burkhard-Keller Tree (BKT). Fue presentado por Burkhard y Kellen en el año 1973 [**12**], probablemente esta solución sea la primera solución de búsqueda que se presentaba en espacios métricos.

El árbol asume una función a distancia discreta y es construida recursivamente de la siguiente manera: dada una base de datos X indexada, se toma un objeto

arbitrario p elemento de X el cual es seleccionado como el nodo raíz del árbol. Por cada distancia $i \geq 0$, se obtienen los subconjuntos X_i definidos de la siguiente forma

$$X_i = \{x \in X \mid d(x,p) = i\}.$$

A cada conjunto X_i no vacío, le corresponde un nodo hijo de p. Todos los nodo hijo, es decir, las ramas u hojas, pueden ser particionados recursivamente hasta que ya no sea posible. Cuando se divide uno de estos, algún objeto x_j del conjunto X_i es escogido como representante del conjunto. Un conjunto X_i no será dividido si éste contiene sólo un elemento. Los objetos escogidos como raíces de los subárboles son llamados pivotes.

En cuanto al algoritmo para consultas por rango, consiste en lo siguiente: la consulta por rango para la búsqueda $R(q,r)$, inicia en el nodo raíz p del árbol y éste se compara con el objeto q. Si p satisface la consulta , es decir, la $d(q,p) \leq r$. Se proyectará al objeto p, en caso de que ocurra lo contrario se analizará otro objeto. El procedimiento se realiza recursivamente con los nodos que están hacía abajo, de tal forma que todos cumplan lo siguiente:

$$max\left\{d\left(q,p\right) - r, 0 \leq i \leq d\left(q,p\right) + r\right\}. \tag{2.5.1}$$

Al analizar la ecuación 2.5.1 se puede notar que el algoritmo elimina algunas ramas del árbol, esto es consecuencia directa de las cotas que contiene la desigualdad anterior. Por ello cuando el radio de la consulta crece, el numero de subárboles seleccionados aumenta y también el costo de la búsqueda, pues los cálculos realizados para medir las distancias incrementara. Esto se debe a que durante la evaluación de la búsqueda, el algoritmo atraviesa el árbol y determina distancias de los pivotes a los nodos internos. En la Figura 2.5.2 se muestra el procedimiento para la construcción del árbol y él de la búsqueda.

2.5.1.2. Fixed Queries Tree (FQT). El FQT [**13**], es sólo una modificación del BKT ya que en la construcción del índice el BKT tiene los pivotes en niveles individuales y diferentes. En cuanto al FQT utiliza solamente un pivote para todos lo nodos que están en el mismo nivel.

Dado un espacio métrico (X,d) y un subconjunto $C \subset X$, todos los objetos de C están almacenados en los mismos niveles y nodos internos de la base, esto facilita la navegación en ellos durante la búsqueda o inserción. Debido a que reduce la complejidad interna, es decir, solo se necesita la distancia de un objeto p al pivote x_i del nivel correspondiente.

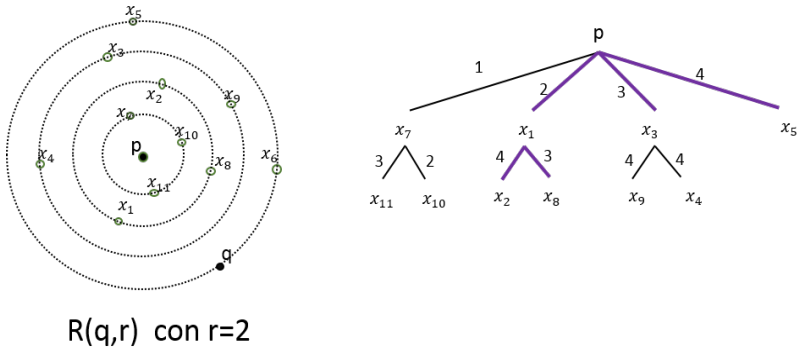


FIGURA 2.5.2. Ejemplo de indexación utilizando el algoritmo *BKT*.

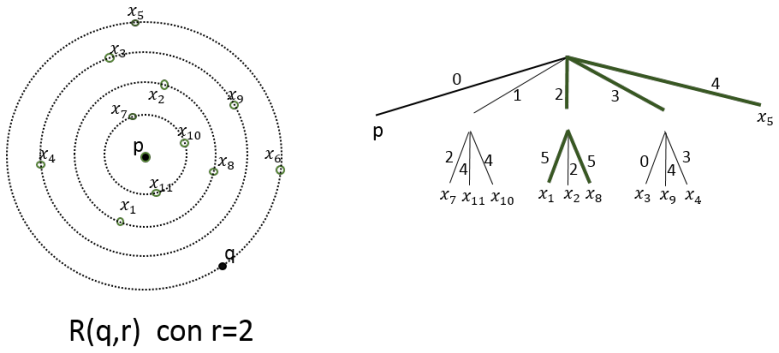


FIGURA 2.5.3. Ejemplo utilizando el algoritmo *FQT*.

En cuanto al rango del algoritmo de búsqueda, es el mismo que para el BKT. La ventaja de esta estructura es que se reduce considerablemente el número de distancias calculadas, porque incluso si se evalúa más de un subárbol durante la consulta, sólo se calcula una distancia entre el objeto p y un pivote especifico x_j. La Figura 2.5.3 muestra un ejemplo del algoritmo (FQT).

2.5.1.3. Fixed-Height Queries Tree (FHQT). El FHQT también de la autoria de R. Baeza [**14**], es una variante del FQT. El proceso de construcción es similar sin embargo la diferencia radica en la manera de representar los nodos hoja y los nodos internos.

La estructura del FHQT tiene todos sus nodos hoja en el mismo nivel, es decir los niveles están a la misma distancia h, de hecho en este algoritmo al tomar la distancia de los pivotes a los caminos más largos no implicaría costos adicionales, debido a que estas distancias se calculan de acuerdo a las necesidades de búsqueda

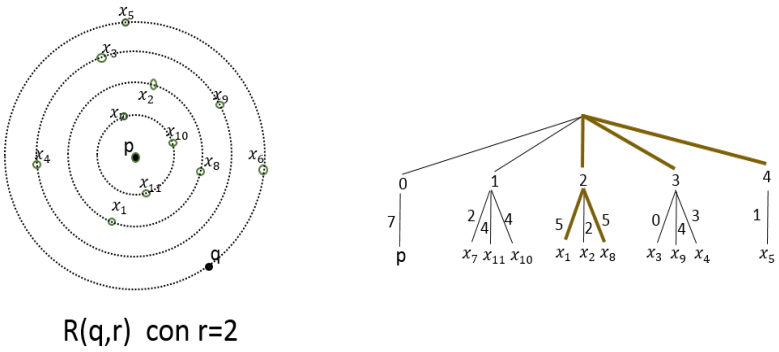


FIGURA 2.5.4. Ejemplo utilizando el algoritmo *FHQT*.

de otros caminos. Por ejemplo, si se incrementa la altura del árbol treinta veces más, sólo se agregan treinta cálculos de distancia más para toda la búsqueda por similitud. Otra ventaja es que se pueden introducir más nodos transversales y el costo de operación sería muy barato. La Figura 2.5.4 muestra un ejemplo de cómo se llevaría a cabo el FHQT.

2.5.1.4. Fixed Queries Array (FQA). El Fixed Queries Array presentado por E. Chavez y cía, [**15, 16**], es un algoritmo que tiene una estructura estrechamente ligada a la del FHQT, la diferencia principal es la forma en la que se acomodan los nodos hoja, los internos y sus distancias, el FHQT tiene estructura de árbol y el FQA de matriz. A continuación se analizará con más detalle la relación que existe entre ellos y el proceso de construcción del FQA.

Dado un espacio métrico (X, d), el FHQT con altura h es construido con un conjunto de datos $C \subset X$. Donde los caminos raíz-hoja del FHQT son recorridos de izquierda a derecha y después se acomodan estos datos en una matriz, el resultado obtenido es el FQA. Cada columna consiste de h números representando las distancia de cada pivote utilizado en el FHQT. De hecho, la secuencia de h números es el camino desde la raíz del FHQT a la hoja.

La estructura del FQA simplemente almacena los datos de los objetos lexicográficamente ordenados por esta secuencia de distancias. Específicamente, los objetos inicialmente son almacenados con respecto al primer pivote y después se considera la distancia de estos con respecto al segundo pivote y nuevamente son almacenados, se realiza el mismo proceso hasta terminar con todos los pivotes. Un ejemplo del FQA se muestra en la Figura 2.5.5.

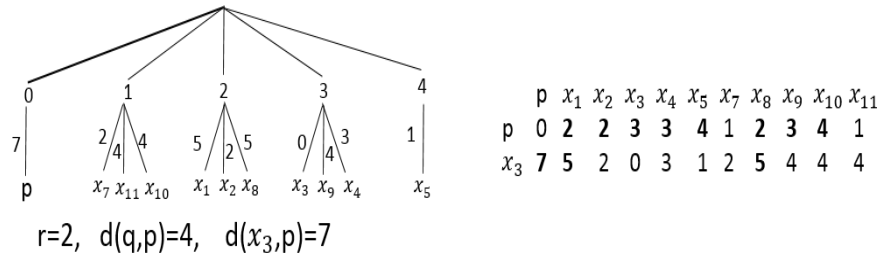


FIGURA 2.5.5. Ejemplo de una búsqueda por rango utilizando el *FQA*.

2.5.1.5. Vantage Point Tree (VPT). Es un algoritmo presentado en 1993 [**17**], está diseñado para métricas continuas, sin embargo también soporta métricas discretas. Este algoritmo está basado en el principio del particionamiento y consiste en lo siguiente:

Dado un espacio métrico (X, d) y un subconjunto $S \subseteq X$, donde se divide al conjunto S en dos subconjuntos S_1 y S_2, a partir del objeto p elegido, al cual llamaremos raíz, se toma la distancia media de p a S_1 y S_2. Esta técnica se empieza con todos los objetos del conjunto S y se aplica el procedimiento de particionamiento recursivamente a todas las hojas, de esta manera se construye un árbol binario balanceado. Aplicando la mediana de ir dividiendo un conjunto de datos en dos puede ser reemplazado por una estrategia donde se empleé el significado de distancias desde p a todos los objetos $\{X\} \setminus p$.

Este método del punto medio puede mejorar el rendimiento para datos en espacios vectoriales de grandes dimensiones. Sin embargo una desventaja de la estrategia del punto medio es que se puede generar un árbol desbalanceado, impactando negativamente un algoritmo de búsqueda eficiente.

En cuanto al algoritmo de búsqueda por rango $R(q, r)$ se va recorriendo el VPT desde la raíz a los demás "niveles". Para cada nodo interno se evalúa la distancia $d(q, p)$, entre el pivote y el objeto de búsqueda q. Si la distancia $d(q, p) \leq r$, el pivote p es proyectado. Para los nodos internos, el algoritmo también puede elegir a qué subárboles se debe accesar.

El siguiente ejemplo muestra el procedimiento que se lleva acabo para formar el árbol correspondiente a la BD. Primero se elige un elemento cualquiera como raíz, en este caso x_6 y se calcula la media M del conjunto S, ésta se calculara de la siguiente manera,

$$M = \text{media}\left\{d(x_6, x_i)\,\text{con}x_i \in S \setminus x_6\right\}.$$

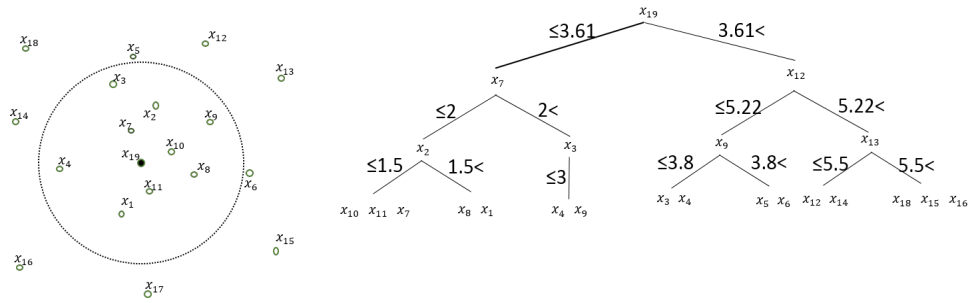


FIGURA 2.5.6. Ejemplo de indexación utilizando el *VPT*, con x_6 como raíz.

Posteriormente los elementos $x_i \in X$ tales que $d(x_6, x_i) \leq M$ son insertados en el subárbol izquierdo y el resto en el subárbol derecho. Se realiza este procedimiento recursivamente hasta que ya no sea posible dividir los subárboles.

Para realizar la búsqueda se calcula la distancia $d(x_6, q)$, para algún q dado, si $d(x_6, q) \leq M$ se acceda al subárbol izquierdo, en caso contrario se ingresa en el derecho. Se puede dar el caso en el que se ingrese a ambos subárboles. La Figura 2.5.6 muestra un ejemplo del VPT.

2.5.1.6. Multi–Way Vantage Point Tree (mw-VPT). El mw-VPT [**18**], es un poco parecido al anterior pues examina todos los objetos de la BD. El VPT toma en cuenta un número considerable de pivotes, esto implica que se deben realizar muchos cálculos para obtener las distancias del objeto p a los pivotes que le anteceden desde el nodo raíz, a pesar de ello aún faltan distancias por calcular, otro de los errores es que el VPT en algunas ocasiones generaba árboles desbalanceados (como el de la Figura 2.4.6). Por ello el método mw-VPT fue planteado como una alternativa para mejorar los problemas antes mencionados.

En el mw-VPT se deja de utilizar un árbol binario y se propone un árbol n-ario, con $n > 2$. De esta manera se deja de utilizar la media y se comienzan a utilizar percentiles $d_{m_1}, d_{m_2}, \ldots, d_{m_n}$. Para particionar la base se hacen subconjuntos parecidos a cortes espirales.

Sin embargo, diversos experimentos revelan que el mw-VPT no siempre mejora el rendimiento debido a los cortes por espiral. Esto regularmente ocurre en dominios de grandes dimensiones. Un ejemplo de indexación utilizando el mw-VPT se muestra en la Figura 2.5.7.

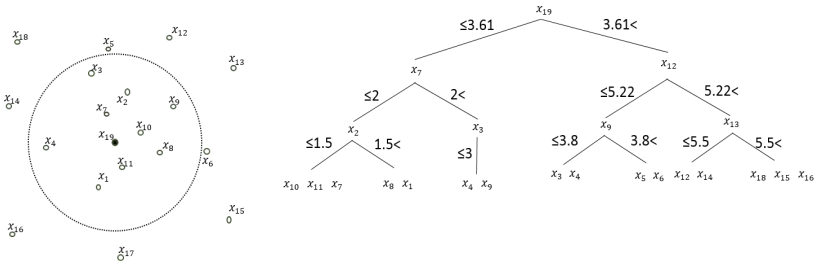


FIGURA 2.5.7. Ejemplo de indexación utilizando el mw-VPT

2.5.1.7. Approximating Search Algorithm (AESA). El AESA presentado por E. Vidal [**19**], almacena en una matriz las distancias entre los objetos de la BD, las cuales han sido calculadas durante la creación de la estructura AESA, su cálculo se realiza de la siguiente forma.

Dado un espacio métrico (X, d), donde $\mid X \mid = n$, sean x_i y x_j elementos de X, se calcula $d(x_i, x_j)$ y se almacena en una matriz. Es así como el AESA es una matriz de $n \times x$ en la que se almacenan las distancias entre todos los pares de objetos de la BD. Debido a la simetría de las funciones métricas únicamente se almacenan las distancias almacenadas en el triángulo superior, por tanto se almacenan $\frac{n(n-1)}{2}$ distancias. El inconveniente de este método es que cada objeto del AESA, tiene un rol de pivote.

En cuanto a la consulta, la búsqueda por rango $R(q, r)$, se avalua la distancia de q a p y es utilizada para descartar algunos objetos. Un objeto x_i, puede ser descartado si $\mid d(q, p) - (q, x_i) \mid > r$, esto es válido pues todas las distancias son conocidas.

El algoritmo elige entre los objetos que aún permanecen, la elección de los pivotes es influenciada por la cota inferior, es decir, $\mid d(q, p) - (q, x_i) \mid$, si se quiere maximizar el efecto de descarte, es suficiente con hacer más pequeña esta cota, es así como se estarán proyectando los objetos más cercanos a q.

El proceso se repite hasta obtener el conjunto de elementos más cercanos de tal forma que ya no sea posible reducirlo. Finalmente los objetos obtenidos serán comparados con q, para que los objetos que satisfagan la búsqueda cumplan con lo siguiente: $d(q, p) < r$.

2.5.1.8. Lineal AESA (LAESA). El algoritmo LAESA [**20**], mejora la complejidad del espacio utilizada en AESA, ya que sólo elige m pivotes. Éste utiliza una matriz que almacena $n \times m$ distancias, sin embargo se presenta un nuevo problema ¿Cómo elegir de manera adecuada los m pivotes?

Durante la selección de pivotes el algoritmo intenta elegir los pivotes de tal forma que estén lo más alejados posible uno de otro.

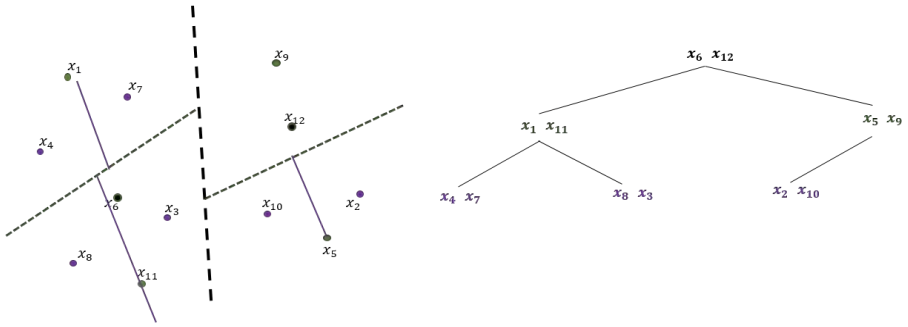


FIGURA 2.5.8. Indexación de una base utilizando el BST.

El procedimiento de búsqueda es similar al AESA, excepto por el hecho de que algunos objetos podrían no ser pivotes o en su defecto durante el proceso de descarte pudo haberse eliminado a algunos de ellos.

2.5.2. Algoritmos de Indexación Basados en Particiones Compactas.

2.5.2.1. Bisector Tree (BST). Presentado en 1983 [**21**], probablemente ésta sea la primera estructura de indexacion en utilizar la partición generalizada de hiperplanos. La estructura del BST es un árbol binario que se construye recursivamente de la siguiente manera. Dado un espacio métrico (X, d) y un subconjunto $S \subset X$, primero se seleccionan dos pivotes p_1y p_2, elementos de S. Luego se hacen dos particiones al hiperplano donde a cada pivote le corresponde una de ellas, los objetos más cercanos a cada árbol forman el subárbol de cada nodo. La cobertura de radio es la máxima distancia entre el pivote y el último objeto del subárbol. El proceso se repite recursivamente sobre los objetos pertenecientes a cada zona, el particionado dará como resultado un árbol en donde a cada nodo le corresponderán dos centros.

El algoritmo de consulta por rango $R(q, r)$, accesa a un subárbol si $d(q, p_i) - r_i \leq r$. La Figura 2.5.8 muestra un ejemplo de este algoritmo.

2.5.2.2. Voronoi Tree (VT). El Voronoi Tree [**22**], mejora las particiones de la base pues permite asignar 2 ó 3 elementos por nodo. Además se puede utilizar en bases de datos dinámicas, ya que cuando se crea un nuevo nodo, éste se insertará en el elemento más cercano que pertenezca al padre.

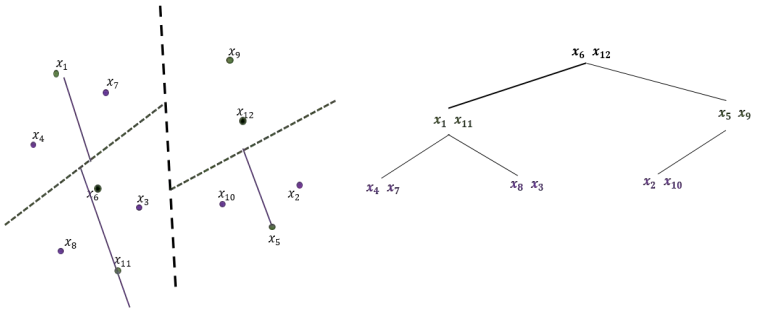


FIGURA 2.5.9. Indexación de una base llevando a cabo el procedimiento del GHT.

El VT tiene la propiedad de que el radio de cobertura se reduce a medida que se desciende por el árbol. Los autores muestran que el VT es superior y mejor balanceado que el BST, de la misma manera que el VT los pivotes indexan localmente, es decir, por cada subárbol.

2.5.2.3. Generalized Hyperplane Tree (GHT). El algoritmo GHT [**23**], presenta una construcción similar al BST. Sin embargo, el algoritmo utiliza los hiperplanos como condición para descartar ramas del árbol, en lugar del radio de cobertura.

Para el proceso de descarte se utiliza el hiperplano entre pivotes p_1 y p_2, para ver que subárbol visita. El criterio para recorrer el subárbol izquierdo es el siguiente

$$d(q, p_1) - r < (q, p_2) + r,$$

para accesar al subárbol derecho el criterio es

$$d(q, p_2) - r < (q, p_1) + r.$$

Un ejemplo de este algoritmo se muestra en la Figura 2.5.9.

2.5.2.4. Geometric Near-neighbor Access Tree (GNAT). El GNAT presentado por S. Brin en 1995 [**24**], utiliza m pivotes en cada nodo interno. Específicamente, un conjunto de pivotes $P = \{p_1, \ldots, p_m\}$, con $P \subset X$. De esta manera se divide en $S_1, \ldots, S_m$ regiones. Los elementos de cada S_i, se asignan de acuerdo a la distancia que hay de los objetos a cada p_i correspondiente. En otras palabras, para algún objeto $x_i \in X - P$, x_i es elemento de S_i si y sólo si

$$d(p_i, x_i) \leq d(p_j, x_i),$$

para todos los $i, j = 1, \ldots, m$.

Si se aplica este procedimiento recursivamente se construye un árbol $m - ario$, este proceso de construcción está estrechamente relacionado con las particiones de Voronoi en espacios vectoriales.

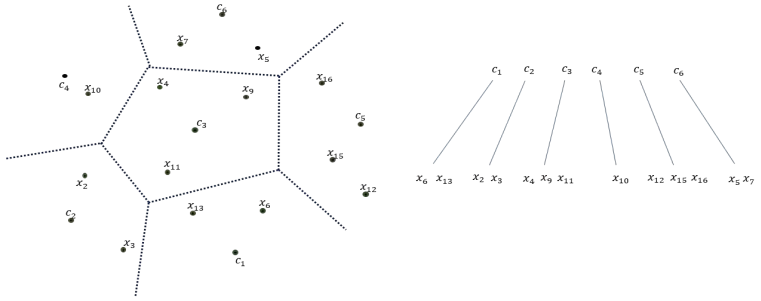


FIGURA 2.5.10. Particionamiento y construcción del árbol, utilizando el GNAT.

Cuando se aplica el principio de particionamiento $m-ario$, el GNAT también conserva distancias de otros pivotes. Esto permite realizar una búsqueda más eficiente, dando como resultado un algoritmo por rango muy diferente a los mencionados anteriormente.

En cada nodo interno, se tiene una matriz de $m \times m$ distancias almacenadas, especialmente la distancia mínima y la máxima, entre cada pivote p_i y los objetos de cada subconjunto. Si son almacenados, formalmente el rango es

$$\left[r_l^{ij}, r_h^{ij}\right],$$

con $i, j = 1, \ldots, m$. Finalmente es definido de la siguiente forma

$$r_l^{ij} = \underset{x_i \in S_i \cup \{P_j\}}{mín} d\left(p_i, x_i\right)$$
$$r_h^{ij} = \underset{x_i \in S_i \cup \{P_j\}}{máx} d\left(p_i, x_i\right).$$

El algoritmo de la consulta por rango para la búsqueda $R\left(q, r\right)$ procede de la siguiente forma: primero elige un pivote $p_i \in P$, calcula la distancia $d\left(q, p_i\right)$. Si $d\left(q, p_i\right) \leq r$, el pivote es proyectado como resultado de la búsqueda. Después, para todos los $p_j \in P$ se remueve p_j de P si $d\left(q, p_i\right) - r > r_h^{ij}$ o $d\left(q, p_i\right) + r > r_l^{ij}$.

Una vez que todos los pivotes en P son examinados, los subárboles del nodo N correspondiente a los nodos P que permanecen son visitados. La Figura 2.5.10 muestra un ejemplo de este algoritmo.

2.5.2.5. M-Tree (MT). El MT [**25**], tiene la ventaja de que puede manejar una estructura dinámica, es decir, puede desarrollarse en dominios donde los datos cambian de tamaño constantemente, son bases de datos que frecuentemente tienen inserciones o eliminación de datos.

Para la construcción del árbol, se elige un conjunto de objetos representantes y a cada uno de ellos es la raíz de un subárbol, el cual se forma con los elementos más cercanos a ella. La estructura tiene una cierta semejanza con el GNAT donde los elementos cercanos a un representante son colocados en un subárbol, la principal diferencia es cómo se insertan los elementos en el MT, ya que no siempre se inserta en el más cercano como es el caso del GNAT.

Se considera que los elementos son almacenados en las hojas y cuando éstas llegan a su capacidad, se produce una separación en dos nodos y un elemento es promovido hacia el padre. Cada elemento almacena la distancia a su padre, lo que hace que este algoritmo use $O(n)$ de espacio. El criterio de descarte de ramas es el mismo del radio de cobertura.

2.5.2.6. Spatial Approximation Tree (SAT). El SAT dasarrollado por G. Navarro [**26**], es un algoritmo basado en los diagramas de Voronoi, pero en contraste con el GNAT él utiliza la estructura de la gráfica de Delaunay. Esta gráfica queda definida de la siguiente manera, cada nodo de la gráfica representa una celda del diagrama de Voronoi, los nodos son conectados entre si, si en el diagrama de Voronoi tienen una frontera en común.

Para llevar a cabo el algoritmo se lleva a cabo lo siguiente: Dado un espacio métrico (X, d) y $P \subset X$ donde $P = \{p_1, \ldots, p_n\}$, se selecciona un elemento p_i arbitrario el cual será la raíz del árbol, después se elige el conjunto de vecinos más cercanos a p_i, éste se denotará por $N(p_i)$. A éste se conectan todos los x_i elementos de X que estén más cerca a p_i que a otro p_j. Los demás x_k se insertan en el conjunto correspondiente al p_k más cercano, este procedimiento se lleva a cabo de forma recursiva hasta que todos los x_i pertenezcan a un conjunto $N(p_i)$.

El algoritmo de consulta para el vecino más cercano de un objeto de búsqueda q, inicia con un objeto arbitrario (un nodo de la gráfica de Delaunay) y continua con el vecino más cercano a q, tanto como le sea posible. Si alcanza un objeto x_i donde todos los vecinos cercanos de x_i, son más cercanos de q que de x_i entonces el vecino x_i es un vecino cercano a q. La Figura 2.5.11 muestra un ejemplo de indexación utilizando el SAT.

2.5.2.7. Lista de Cluster (LC). Este algoritmo planteado por M. Mamede [**27**], es muy utilizado en espacios de altas dimensiones. En cuanto al algoritmo de construcción, consiste en seleccionar un elemento p_i como centro y agrupar sus k elementos más cercanos. El conjunto restante aplica el mismo proceso hasta que todos los elementos hayan sido agrupados.

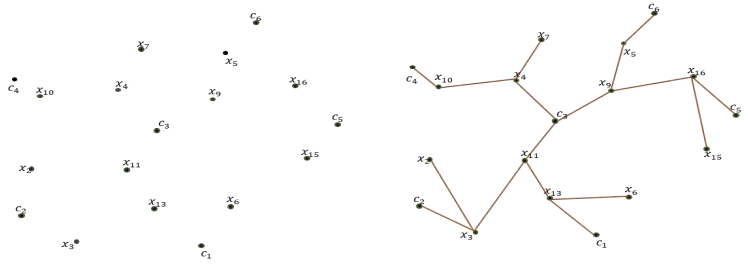


FIGURA 2.5.11. Ejemplo indexación utilizando el SAT.

Cada centro almacena un radio de cobertura, el cual después es utilizado para descartar elementos en una consulta. El LC recorre la lista secuencialmente y emplea dos criterios para agrupar a los elementos de X:

* Fija un número k de elementos más cercanos a p_i.
* Utiliza un radio de tolerancia.

Los resultados mejoran cuando se limita el número de elementos más cercanos.

2.5.3. Eficiencia de los algoritmos.

Para medir la eficiencia de un algoritmo se debe prescindir de factores como la máquina y el software que se está utilizando pues estos van a variar dependiendo del proyecto o necesidades de los desarrolladores. Por ello sólo se considerarán factores primordiales tales como tiempo de ejecución, complejidad de consulta y espacio ocupado [**28**].

La medición de los recursos computacionales utilizados para la ejecución de un algoritmo es evaluada en función del tamaño de la entrada, a continuación se describirá cómo se obtiene la medida de los factores primordiales.

Se denotará por $T(n)$ la función

$$T : K \to \mathbb{R}^+,$$

con $K \subset \mathbb{N}$.

Esta función expresa el tiempo empleado para ejecutar el algoritmo con una entrada de tamaño n. Existen varios métodos para calcular a $T\,(n)$, sin embargo la mayoría lo hace mediante procedimientos empíricos, es decir, se ejecuta el mismo programa sobre diferentes instancias y se mide el tiempo de ejecución para cada una de ellas, con los datos obtenidos se realizan cálculos y estimaciones estadísticas sobre la eficiencia del algoritmo.

Otro factor importante para evaluar un algoritmo es mediante el conteo del número de operaciones que realiza cuando se ejecuta, esta evaluación se define como la complejidad o coste en f y se denota por la función $O(f)$, la cual se define de la siguiente manera,

$$O : K \to \mathbb{N},$$

donde $K = \{1, \ldots, j\}$.

Finalmente se considera el espacio que ocupan los datos que se almacenan de acuerdo al formato estructurado declarado, en este caso son distancias calculadas.

De acuerdo al análisis presentado por cada uno de los autores descritos anteriormente, la Figura 2.5.12 muestra la eficiencia de cada algoritmo.

Medir la eficiencia de un algoritmo es un procedimiento muy útil que sirve para comprender el comportamiento de los algoritmos, sin embargo es muy importante comprobar su rendimiento antes de implementarlos, ya que éste puede variar de acuerdo al dominio de las aplicaciones y las características de las BD.

Algoritmo	Criterio de indexación	Espacio ocupado	Complejidad de la consulta	Tiempo de ejecución
Approximating Search Algorithm (AESA)	Pivotes	n^2 dist	$O(1)^d$	$n \ldots n^2$
Bisector Tree (BST)	Particiones compactas	n ptrs	$O(n^{\alpha})$	-
Burkhard-Keller Tree (BKT)	Pivotes	n ptrs	$O(n^{\alpha})$	-
Fixed Queries Array (FQA)	Pivotes	$nb \ldots nhb$ bits	$O(\log(n))^b$	log$n$ * # distancias
Fixed Queries Tree (FQT)	Pivotes	$n \ldots nh$ ptrs	$O(n^{\alpha})$	-
Fixed-Height Queries Tree (FHQT)	Pivotes	$n \ldots nh$ ptrs	$O(\log n)^b$	-
Generalized Hyperplane Tree (GHT)	Particiones compactas	n ptrs	Sin analizar	-
Geometric Near-neighbor Access Tree (GNAT)	Particiones compactas	nm^2	Sin analizar	-
Lineal AESA (LAESA)	Pivotes	kn dist	$k + O(1)^b$	$n \ldots kn$
Lista de Clusters (LC)	Particiones compactas	n ptrs	Sin analizar	-
M-Tree (MT)	Particiones compactas	n ptrs	Sin analizar	-
Multi-Way Vantage Point Tree (mw-VPT)	Pivotes	n ptrs	$O(\log n)^b$	-
Spatial Approximation Tree (SAT)	Particiones compactas	n ptrs	$O(n^{1-\alpha * \frac{1}{\log \log n)}})$	-
Vantage Point Tree (VPT)	Pivotes	k ptrs	$O(n^{1-\alpha} \log n)$	-
Voronoi Tree (VT)	Particiones compactas	n ptrs	Sin analizar	-

FIGURA 2.5.12. Medición de eficiencia de los algoritmos de indexación más importantes.

Parte 2

Implementación del algoritmo en la BD

Capítulo 3

Descripción del Problema

3.1. Introducción

La generación masiva de datos no estructurados se ha convertido en algo habitual, para entender la dimensión del reto al que se están enfrentando los desarrolladores y administradores. En la Figura 3.1.1 se muestra una cantidad aproximada de los datos que se generan en Internet durante un minuto, esto de acuerdo con los datos obtenidos en http://pennystocks.la/internet-in-real-time/.

Al principio lo más importante era almacenar los datos, sin embargo al ir detectando que estos se podían convertir en información importante para predecir ciertos acontecimientos, es de esperarse que las prioridades vayan cambiando. Debido a ello la calidad, la integración, la manipulación y la extracción de valor de los datos no estructurados se han convertido en características fundamentales para poder realizar un buen análisis de la información que se ha obtenido con ellos, es así como la tecnología *Big Data* se convierte en pieza clave para este tipo de situaciones.

En la actualidad existen una gran diversidad de herramientas y software, para el tratamiento de tecnologías como Big Data y NoSQL. Sin embargo, al momento

FIGURA 3.1.1. Datos generados en Internet durante un minuto.

de hablar de software para tratar grandes almacenes de datos lo primero que se debe de tomar en cuenta es el tipo de objetos que se van a almacenar en la BD y de acuerdo a esto buscar una herramienta que se adecue a las necesidades de la organización.

Las bases de datos NoSQL se clasifican dependiendo de la manera en la que almacenan la información [**29, 30**]. La clasificación de acuerdo a los tipos más utilizados es la siguiente:

1. Orientada a documentos, no se utiliza un esquema definido para especificar de antemano qué registros estarán en cada documento, cada registro puede contener un conjunto de valores diferente y se pueden añadir o eliminar registros sin penalización. Algunos gestores orientados a documentos son CouchDB y MongoDB.
2. Almacenamiento llave-valor, por el momento, ésta es la más simple de las bases de datos NoSQL, simplemente guardan secuencias de valores agrupados en una clave y su valor, tampoco requieren de un esquema definido previamente, son flexibles y escalables. Riak, Redis y DynamoDB, son algunos gestores que utilizan este enfoque.
3. Columnas, este tipo de BD funcionan de forma parecida a las bases de datos relacionales, pero los datos se almacenan en columnas de datos en lugar de registros. Son ideales para cuando se tienen grandes cantidades de datos con una variedad muy amplia, debido a que solamente se deben seguir añadiendo columnas. Algunos manejadores son Hypertable, HBase y Cassandra.
4. Grafos, la configuración de estas BD está basada en la teoría de grafos, donde la estructura fundamental se llama "nodo-relación". Los nodos y las relaciones representan los datos almacenados y por medio de ellos se pueden seguir las conexiones que hay entre entre ellos. Estas bases puede utilizarse para gestionar datos geográficos para la exploración de petróleo o para modelar y optimizar redes y proveedores de telecomunicaciones. InfiniteGraph, Neo4j y AllegroGraph, son gestores orientados a grafos.

Otras de las características que se deben considerar son las siguientes [**4, 29**]:

* Lenguaje de consulta. No existe una sola opción correcta con respecto a los lenguajes de consulta en BD. Aunque SQL es el lenguaje de consulta de base de datos más utilizado hoy en día, existen otros lenguajes que pueden proporcionar una manera más eficaz o eficiente para resolver una consulta. Por ejemplo, si se utiliza un modelo relacional es probable que se utilice

SQL, sin embargo también se puede utilizar lenguajes alternativos como Python o Java.

* Lo mismo ocurre con las bases de datos NoSQL, ya que ofrecen alternativas de almacenamiento y acceso a ellos. Cada SGBD ofrece su propia interfaz, esto hace imposible intercambiar o reutilizar el código de la misma consulta en varias bases.
* Map Reduce, es un marco de trabajo que permite almacenar, procesar y analizar grandes cantidades de datos no estructurados, de forma paralela a través de un grupo distribuido de procesadores.
* Tipo de transacciones, es importante entender qué tipos de datos pueden ser manipulados por la BD y si estos son compatibles y confiables con cierto comportamiento transaccional. A este comportamientos se le describe con el acronimo ***ACID***, su significado es el siguiente:

  Atomicidad: Una transacción es "*todo o nada*" cuando es atómica. Si cualquier parte de la transacción o el sistema subyacente falla, falla toda la transacción.
  Consistencia: Sólo las transacciones con datos válidos se llevarán a cabo en la BD. Si los datos son inadecuados, entonces la transacción no se realizará y los datos no se escriben en la base de datos. Con ellos se garantiza que cualquier dato que se escriba en la base de datos tiene que ser válido de acuerdo a las características definidas.
  Aislamiento: Las transacciones simultáneas múltiples no interferirán entre sí. Todas las transacciones válidas se ejecutarán hasta que se completen y en el orden en que fueron enviadas para su procesamiento.
  Durabilidad: Una vez terminada la transacción los datos que se escriben en la base de datos se quedan allí *"para siempre"*.

La Figura 3.1.2 muestra las características más relevantes de cada una bases de datos NoSQL, de acuerdo a su enfoque.

Este trabajo se enfocará en base de datos NoSQL orientada a documentos, de las cuales existen dos tipos. El primer tipo a menudo se describe como un repositorio para almacenar el contenido completo de documentos tales como archivos de Word, páginas web completas, etcétera. El otro tipo es una BD para el almacenamiento permanente de componentes de documentos, los cuales permanecen como una entidad estática o para el montaje dinámico de las partes de un documento, por ejemplo los que sólo contienen registros de la forma nombre-valor.

Tipo de almacenamiento	Lenguaje de Consulta	Map Reduce	Tipo de Transacciones	Ejemplos de SGBD NoSQL
Columnas	Ruby	Hadoop	Sí, si está habilitado	HBase
Grafos	Walking, Search, Cypher	No	ACID	Neo4j AllegroGraph
Documentos	Comandos	JavaScript	No	MongoDB, CouchDB
Llave-Valor	Comandos	JavaScript	No	Riak, Redis

FIGURA 3.1.2. Tabla comparativa de SGBD de NoSQL.

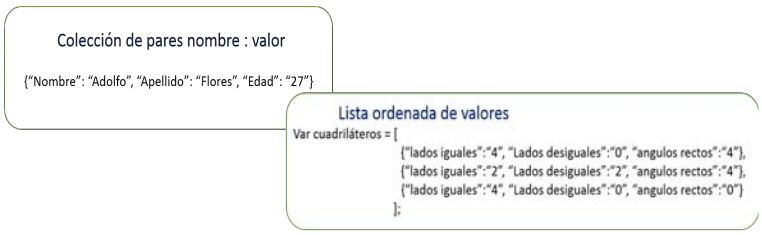


FIGURA 3.1.3. Ejemplo de arquitecturas básicas JSON.

Este tipo de bases gestionan los registros de datos estructurados de forma jerárquica y proporciona medios para recuperar registros en función de su contenido real. Esta categoría de bases de datos NoSQL proporciona la capacidad de manejar millones de lecturas simultáneas, ya tienen una lectura simple. La estructura de los documentos y sus partes es proporcionada por JavaScript Object Notation (JSON) o BSON (BSON) [**4**].

La estructura JSON es un formato de intercambio de datos, basado en un subconjunto de lenguaje de programación JavaScript. El uso de éste no causa demasiados inconvenientes ya que es muy fácil de leer y escribir, además de que es soportado por otros lenguajes de programación. La arquitectura de JSON está constituido por dos estructuras, las cuales son:

* Colección de pares nombre-valor: Está representada mediante programación como objetos, registros, diccionarios, entre otros.
* Lista ordenada de valores: Está implementada como arrays, listas o secuencias.

La Figura 3.1.3 muestra un ejemplo de cada una de las estructuras JSON.

FIGURA 3.1.4. Arquitectura principal de MongoDB.

Para este proyecto se va a utilizar a MongoDB como SGBD. La jerarquía principal de MongoDB viene dada por los elementos que a continuación se presentan:

* Cada servidor puede tener tantas bases de datos como el usuario quiera y el equipo lo permita.
* Las bases de datos estarán formadas por colecciones.
* Los documentos se pueden agrupar en colecciones.
* Cada registro o conjunto de campos y su valor correspondiente, se denomina documento.

La Figura 3.1.4 muestra un diagrama donde se puede apreciar la jerarquía principal de MongoDB.

En MongoDB, no existe un esquema estándar para trabajar con los datos, sin embargo se puede proponer uno que se adecue al proyecto que se está realizando. De hecho, la mayoría de las veces se trabaja con documentos semiestructurados, ya que no todos siguen el mismo esquema, sino que cada uno podría tener el suyo.

La BD que se va a utilizar almacena documentos obtenidos de los perfiles de diversas redes sociales. No todos los documentos tienen el mismo esquema, sin embargo contienen todos o un subconjunto de los registros que se muestran en el esquema de la Figura 3.1.5.

3.2. Problema a Tratar

Al considerar que la información almacenada en la BD corresponde a los datos compartidos en redes sociales, se debe tomar en cuenta que no todos los documentos contienen los mismos registros. Por ejemplo, algunos proporcionan nombre completo, otros un nombre y un apellido, en casos extremos sólo se tiene el registro de un nombre de usuario o “nickname”.

FIGURA 3.1.5. Registros que puede contener un documento de la colección.

Otro factor que se está tomando en cuenta es que en muchos perfiles de redes sociales es muy común que un nombre se escriba de diversas formas. Por ejemplo, el nombre Anahí, se puede encontrar como Anahi, Anaíh, Anaí, Anay, Annahí. Si se realiza una búsqueda exacta sobre esos datos con "Anahí", la consulta no traerá documentos. Sin embargo en algunos casos se puede proponer el uso de una función ya definida por el SGBD, para proyectar los documentos que tengan como subcadena a "Anahí", el inconveniente que se presenta en este caso es que nuevamente la consulta no proyectará ningún dato. Si se tiene paciencia y tiempo se podría ir reduciendo la cadena en la función correspondiente, es decir, se tendría que estar "atinando" al utilizar "Anah", "Ana" y así sucesivamente hasta obtener por lo menos un documento. El procedimiento antes mencionado puede ser una opción para solucionar el problema, pero no la mejor.

La mayoría de las funciones que actualmente se utilizan, están considerando que siempre se trabaja con colecciones de datos donde las reglas gramaticales son respetadas, probablemente en diversas actividades para la gran mayoría de nosotros esta hipótesis es verdadera. Sin embargo cuando se habla de información compartida en redes sociales, estas reglas o estructuras no se ponen en practica tan rigurosamente.

Debido a ello se requiere obtener el nombre de pila de cada documento tomando en cuenta la forma de escribirse más popular, es decir, dado un nombre propio con la forma de escribirse más utilizada, buscar en la BD los nombres con una escritura similar. Se considerará que algunos usuarios escriben su nombre con alguna variación o que existen nombres propios que se escriben diferente a la forma convencional.

Para este tipo de búsquedas se va a proponer una función que realice la consulta por similitud y un índice para que durante la búsqueda se optimicen los recursos computacionales.

3.3. Propuesta de solución

Al considerar el algoritmo que se va a proponer y la necesidad de utilizar una búsqueda por similitud considerando las características de la BD con la que se va a trabajar, se ha decidido implementar el VPT (Vantage-Point Tree) [**17**], con un ligero cambio. Originalmente se plantea utilizar la media para ir construyendo el árbol que modela el índice, termina cuando los nodos hoja tienen la capacidad máxima de elementos establecida. En cuanto a la búsqueda sólo se realiza la comparación del valor que tiene el nodo interno y el objeto de búsqueda. La diferencia que se esta realizando es proponer una variable para ajustar la proximidad del objeto de búsqueda con el valor que tiene asignado cada nodo interno.

A partir de este momento cuando se haga referencia a un objeto x_i, se va a estar hablando indistintamente de algún documento de la base de datos y cuando se mencione a un p_{x_i}, se hará alusión a que el documento también cumple la función de pivote para los índices que están basados en pivotes, o como centro de alguna región si el índice está basado en particiones compactas. En caso de que el pivote o centro, no sea parte de la base sólo se denotará como p_i. En cuanto a la BD se le empezará a llamar espacio y se denotará con X.

Durante el análisis de los métodos de indexación con pivotes, se observo que la eficiencia de los métodos de búsqueda por similitud depende del conjunto $P = \{p_1, \ldots, p_k\}$ que se ha sido elegido como referencia, ya que este conjunto determinará la capacidad del índice para descartar elementos y evitar que sean comparados durante la consulta. Debido a ello el número de elementos utilizados cómo referencia, su ubicación en el espacio y su ubicación con respecto a otros elementos, serán determinantes para el número de evaluaciones finales.

Por lo anterior, determinar el número de elementos que sirven como referencia es un paso importante, pues la eficiencia de una búsqueda depende de este parámetro. Aunque también cabe la posibilidad de que el valor de k varíe de acuerdo al espacio métrico en el que se esté trabajando.

La implementación del índice que se está proponiendo se llevara a cabo en dos partes, en la primera se propondrá una función d que dependerá de la longitud de x_i y en la segunda una que dependerá del peso. El número de elementos k en cada una de las partes será diferente, a continuación se verá en qué consiste cada una.

3.3.1. Asignación de Longitud.

Para la implementación del primera parte se tomará en cuenta el registro “nombre”, puede llegar a ocurrir que un x_i no tenga ese registro. Sin embargo, no hay algún

inconveniente ya que MongoDB permite generar índices aunque no todos los documentos contengan ese registro. Por otro lado se agregará a todos los documentos el registro "long_nom", éste contendrá el número de caracteres que componen el registro nombre.

Para comenzar con la construcción, previamente se debe calcular la longitud de cada objeto x_i de la BD. La longitud de x_i, se denotará como $L(x_i)$ y se calculará contando el número de caracteres que tiene el registro "nombre" del objeto x_i. Por ejemplo, si el objeto es el siguiente

$$x_p = \{\text{"nombre" : "ejemplo", "apellido" : "ejemplito"}\},$$

la longitud de x_p es :

$$L(x_p) = 7.$$

Todos los x_i que contengan el registro nombre, van a tener un nuevo registro, este será "long_nom" y contendrá el valor de $L(x_i)$, es decir,

$$\text{long\_nom} = L(x_i),$$

considerando el objeto del ejemplo anterior, después del *insert,* x_p tendría los siguientes registros

$$x_p = \{\text{"nombre" : "ejemplo","apellido" : "ejemplito","long\_nom" : "7"}\}.$$

A continuación se muestra el psudocódigo que se ha utilizado para la asignación de longitud en cada uno de los documentos.

Inicio
{
Si x_i **tiene registro "nombre",**
    entonces
        var nickname = "nombre"
        var $L(x_i)$**= nickname.length**
    Insertar long_nom = $L(x_i)$
    }
Fin.

3.3.2. Primera Parte del Índice. Sean X el conjunto que contiene todos los objetos de la BD y d_{long} una función distancia definida de la siguiente forma

$$d_{long}(x_i, x_j) :=\mid L(x_i) - L(x_j) \mid,$$

para todo x_i elemento de X, donde $L(x_i) \geq 2$.

La pareja (X, d_{long}) es un espacio métrico, el cual será utilizado en esta parte.

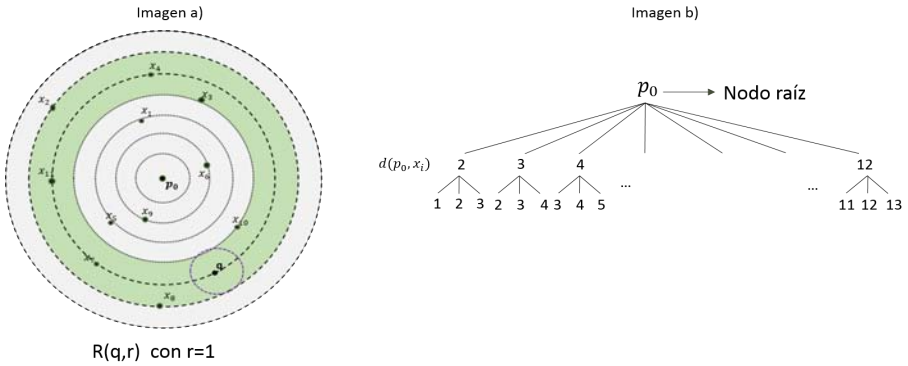


FIGURA 3.3.1. En la imagen a) se representa la proyección de los objetos que se encuentran en la bola de radio 1 y en b) se realiza el modelado con un árbol.

Cuando se realice la consulta sobre q lo primero que se va a calcular es $L(q)$, y posteriormente se aplicará la métrica d_{long} de q a todos los objetos x_i, es decir,

$$d_{long}(q, x_i) := \mid L(q) - L(x_i) \mid .$$

En esta parte se pretende dar como margen de error un carácter, es decir, al realizar la consulta se considera que pueda faltar o sobrar un carácter, de hecho también se contempla que el número de caracteres sea el mismo. Por ello se tomarán en cuenta todos los objetos que se encuentren en la bola de radio 1 con centro en q, es decir,

$$B(q, 1) := \{x_i \in X : d_{long}(q, x_i) \leq 1\} .$$

En la Figura 3.3.1 se puede apreciar que en la imagen a) se seleccionan los objetos que serán incluidos en la primera parte del índice y en la imagen b) se hace una representación utilizando un árbol.

Utilizar una bola con un radio específico ha servido para comparar menos objetos x_i con el objeto de búsqueda q; sin embargo hay casos en donde se necesita refinar esta exclusión. Por ejemplo, al aplicar la primera parte del índice a los objetos que tienen $L(x_i) = 6$, se extrae a los x_i de longitud 5, 6 o 7, lo cual representa proyectar mas del 50 % del total de elementos de la BD con la que se está experimentando. Estas cantidades se pueden observar en la gráfica de la Figura 3.3.2.

Por otro lado se puede distinguir que cuando la longitud es 2, 3, 9, 10, 11 o 12; será suficiente evaluar los objetos que están dentro la bola de radio 1. Debido a que la cantidad de objetos es mínima, pero la frecuencia de nombres con esa longitud

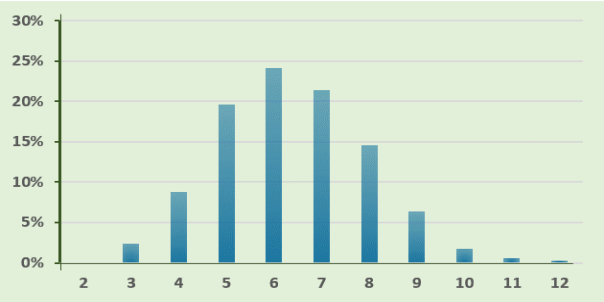


FIGURA 3.3.2. Porcentaje de los nombres de acuerdo a su longitud.

es pequeña. Al tomar como referencia las situaciones antes expuestas se considera necesario crear una entrada de exclusión para generar un índice compuesto y al final evaluar sólo los objetos necesarios.

3.3.3. Asignación de Peso.

Anteriormente se mencionó que las búsquedas que se proponen en este algoritmo no dependen del orden de los caracteres que conforman las palabras, en este caso se esta considerando a una palabra como una sucesión de caracteres, en dónde lo más importante es el peso que se le asignará a cada carácter, el cual dependerá de la cantidad de veces que se repita.

Para obtener este valor primero se definirá el conjunto $C = \{c_i, \ldots, c_k\}$, éste contiene todos los caracteres que pueden ser utilizados para formar un "nombre", un ejemplo de este conjunto puede ser el abecedario. Posteriormente se obtienen el valor del registro "nombre" de cada x_i. Estos conformarán el conjunto $\mathcal{N} = \{N_1, \ldots, N_i\}$.

Los elementos de C y $\mathcal{N}$ se acomodarán en una matriz M de $i \times k$ entradas, donde $i =$ número de nombres y $k =$ número de carácteres. El valor de cada entrada m_{ik}, quedara definido de la siguiente manera:

$$m_{ik} := \begin{cases} 0, & \text{si el carácter } c_k \notin N_i \\ 1, & \text{si el carácter } \mathrm{c_k} \in \mathrm{N_i}. \end{cases}$$

El arreglo debe quedar como el que se muestra en la Figura 3.3.3.

Para saber cuántas veces se repite cada carácter se sumarán los valores que contiene cada columna correspondiente a un c_k, este valor se representará con r_{c_k}, es decir,

$$r_{c_k} = \sum_{a=1}^{i} m_{ik}.$$

$N_i \backslash C_k$	C_1	C_2	.	.	.	.	C_k
N_1	1	0					1
N_2	1	0					0
.							0
.							0
.							1
N_i	1	0					0

FIGURA 3.3.3. Arreglo matricial de nombres y caracteres.

C_1	C_2	.	.	.	.	C_k
1	0					1
+ 1	+ 0					+ 0
+	+					+ 0
						0
						1
+ 1	+ 0					+ 0
r_{c_1}	r_{c_2}					r_{c_k}

FIGURA 3.3.4. Proceso para obtener el número de veces que se utiliza cada carácter.

Finalmente para determinar el total de caracteres se realizará lo siguiente,

$$R = \sum_{a=1}^{k} r_{c_a}.$$

La Figura 3.3.4 muestra la forma en la que se obtendrían los valores r_{c_k}, los cuales se utilizarán para determinar la frecuencia de cada carácter. Ésta quedará determinada por $z = \frac{r_{c_k}}{R}$.

La gráfica de la Figura 3.3.5 muestra la frecuencia con la que se utiliza cada carácter, de acuerdo a los datos que se tienen almacenados en la base.

Hay que observar que esta búsqueda empieza a centrarse en la repetición de los caracteres, debido a ello si un carácter se repite poco debería dar más información que uno que se repite mucho. Por ejemplo, si se ingresa una búsqueda que contenga a la w se sabe que sólo se puede encontrar en dos nombres o en uno que contenga a la w dos veces, por lo tanto la búsqueda sería más rápida. Pero si se ingresa

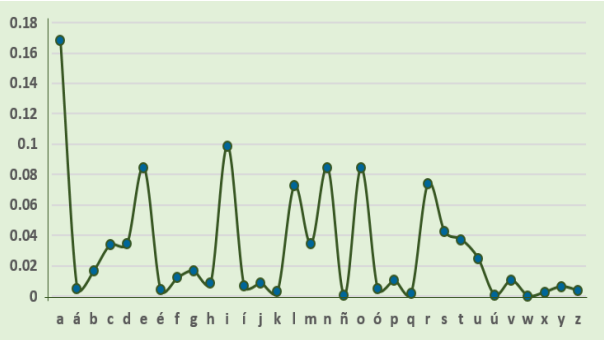


FIGURA 3.3.5. Frecuencia con la que se utilizan los caracteres.

una búsqueda que contenga a la a, la búsqueda se vuelve un poco más complicada porque el uso de ese carácter es mucho más frecuente.

Para empezar a trabajar con los valores, se propone utilizar una función en donde se evalúe la frecuencia de cada carácter y a partir de ahí definir una métrica que permita manipular los datos de una manera más eficiente. La función que se propone es

$$f = \frac{\exp^{10+z}}{10^5},$$

ésta permitirá obtener distancias de a cuerdo a los r'_{c_k}s calculados anteriormente. La función será utilizada para definir la métrica de los pesos d_p, la cual quedará definida de la siguiente forma

$$d_p\left(c_a, c_b\right) :=\mid f\left(\frac{r_{c_a}}{R}\right) - f\left(\frac{r_{c_b}}{R}\right) \mid .$$

Es importante mencionar que la función f puede cambiar dependiendo de la BD con la que se esté trabajando.

En las gráficas de la Figura 3.3.6 se puede observar el comportamiento de f al evaluar el valor de cada r_{c_k}. La primera gráfica contiene los valores ordenados alfabéticamente y la segunda los valores ordenados de menor a mayor.

Finalmente se calculará el peso de los N_i, los cuales pertenecen a un objeto x_i, debido a ello a este valor se le llamará peso de x_i y se denotará como $W\left(x_i\right)$. Los pesos se calculan de la siguiente forma

$$W\left(x_i\right) = \sum_{a=1}^{k} m_{ia} \cdot f\left(\frac{r_{c_a}}{R}\right).$$

Nuevamente todos los x_i que contengan el registro "nombre", van a tener un registro de peso llamado "peso_nom" y contendrá el valor de $W\left(x_i\right)$, es decir,

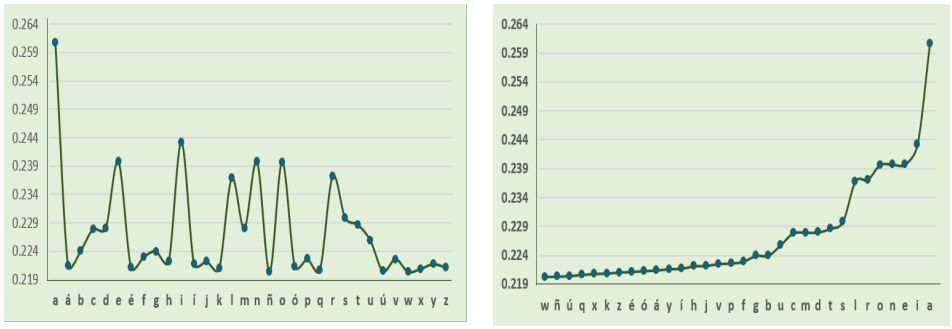


FIGURA 3.3.6. Gráfica de f al utilizar los valores de la frecuencia.

$$\text{peso\_nom} = W\left(x_i\right).$$

De esta manera el objeto x_p, utilizado en el ejemplo anterior, tendrá los siguientes registros

$$x_p = \{\text{"nombre":"ejemplo","apellido":"ejemplito",}$$
$$\text{"long\_nom":"7","peso\_nom":"W}(x_p)\text{"}\}$$

A continuación se el pseudocódigo que se ha utilizado para la asignación de peso en cada uno de los documentos que contenga el registro "nombre".

Inicio

var count_nombres = #documentos con registro **"**nombre**"**)
var count_caracteres = #carácteres que se pueden utilizar)
var M = Array (cout_nombres)*(count_caracteres)

{
para cada nombre de M
evaluar
{
if el carácter $c_k \in Nombre$,
entonces colocar 1
else colocar 0
}
Sumar los valores de cada columna y generar las variables que contendrán esos valores.
var $suma\_c_k$ ***= suma valores columna*** c_k
Obtener el total de caracteres

var Total $= suma\_c_1 + \ldots + suma\_c_k$
Aplicar la función a cada frecuencia y generar las variables que guardarán ese valor.

var $peso\_c_k$ $=$ ***f***$(suma\_c_k/Total)$
Guardar los pesos en una array m_{k1}
Multiplicar M*m para obtener el peso de cada nombre y guardarlos una variable W(x_i)***.***
Insertar peso_nom = $\boldsymbol{W}(x_i)$
}
Fin.

Al termino de esta parte, cada documento x_i de la base tendrá dos registros adicionales a los que originalmente tenía, estos son long_nom y peso_nom, los cuales contienen los valores de $L(x_i)$ y $W(x_i)$ respectivamente.

Una vez que se realizó la preparación de la base se definirá la segunda parte del índice.

3.3.4. Segunda Parte del Índice.

En esta parte se utilizarán las longitudes y los pesos para definir los pivotes.

Por el procedimiento anterior se sabe que $L(x_i)$ esta acotado de la siguiente manera

$$2 \leq L(x_i) \leq 12,$$

donde 2 es la longitud mínima y 12 la máxima.

El valor de cada longitud será representada por un nodo raíz, por tanto se tendrán 11 raíces. Estos se denotarán con C_i, donde $i = 2, \ldots, 12$.

Cada p_i será un nodo interno del árbol que se forme a partir de un C_i, posteriormente se calculará el promedio de los pesos correspondientes a C_i, este valor se denotará con V_{p_i}.

Se va a utilizar a V_{p_i} para dividir los objetos de C_i en dos subconjuntos, el primero contendrá a los x_i que tienen un peso menor o igual a V_{p_i} y segundo a los que tienen un peso mayor a V_{p_i}. El primer subconjunto se ubicará en el nodo izquierdo, éste corresponderá al pivote $Ip_{1\_}$ y el segundo en el pivote derecho, Dp_{1_+}. Después se vuelve a aplicar el mismo procedimiento para los elementos ubicados en $Ip_{1\_}$ y Dp_{1_+}.

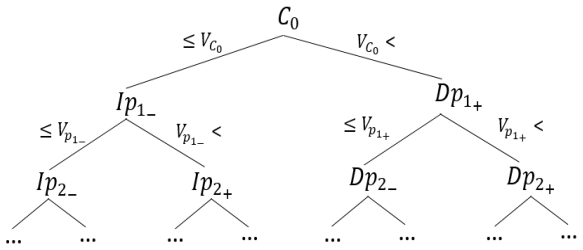


FIGURA 3.3.7. Árbol que se obtiene al aplicar el algoritmo completo.

Este procedimiento se va a aplicar recursivamente hasta que los nodo hoja tengan a lo más 10 objetos. La Figura 3.3.7 hace una representación de cómo se obtendría el árbol completo.

Si se realiza una búsqueda sobre q primero se toman en cuenta los objetos que se encuentran en la bola $B_{(q,1)}$ y después se verifica a qué nodo interno pertenece el peso $W(q)$. La ecuación 3.3.1 describe las condiciones que deben cumplir los objetos a evaluar.

$$B(q,1) := \left\{ x_i \in X : d(q,x_i) \leq 1 \wedge W(q) \in p_{i_\pm} \right\}. \tag{3.3.1}$$

3.3.5. Definición de un α para Refinar la Búsqueda.

Al realizar experimentos que cumplen la ecuación 3.3.1 se ha detectado que se genera cierto error de proximidad en el algoritmo. Debido a que en los casos donde

$$L(x_i) \neq L(q),$$

existe el peso de un carácter que en algunos casos genera una diferencia muy grande de q con respecto a esos x_i.

Por ello se ha tenido que generar una variable α, que tiene el siguiente valor

$$\alpha = \frac{\sum_{i=1}^{n} f\left(\frac{r_{c_i}}{R}\right)}{n}.$$

donde se esta tomando en cuenta a los n caracteres con el peso más alto, la selección del número de caracteres va a hacer que la α, sea tan grande o pequeña como se quiera.

Una vez que se tiene el valor de α, se define lo siguiente

* Si

$$L(x_i) < L(q),$$

entonces se considera el valor del nodo interno

$$(W(q) - \alpha) \in p_{i_\pm}.$$

* Si

$$L(x_i) = L(q),$$

entonces se considera el valor del nodo interno

$$W(q) \in p_{i_\pm}.$$

* Si

$$L(x_i) > L(q),$$

entonces se considera el valor del nodo interno

$$(W(q) + \alpha) \in p_{i_\pm}.$$

A este proceso se le denominará *criterio de ajuste.*

Capítulo 4

Ejecución de la propuesta

4.1. Antecedentes

Se requiere ejecutar el algoritmo para obtener peso y longitud de los documentos existentes en la BD.

Inicialmente los documentos no contienen algún dato que haga referencia a la longitud o peso en alguno de sus registros. Por ejemplo el registro de la Figura 4.1.1.

Posteriormente al realizar la primera parte del algoritmo el documento tiene un registro más. La Figura 4.1.2 contiene el documento con un registro adicional que hace alusión a la longitud.

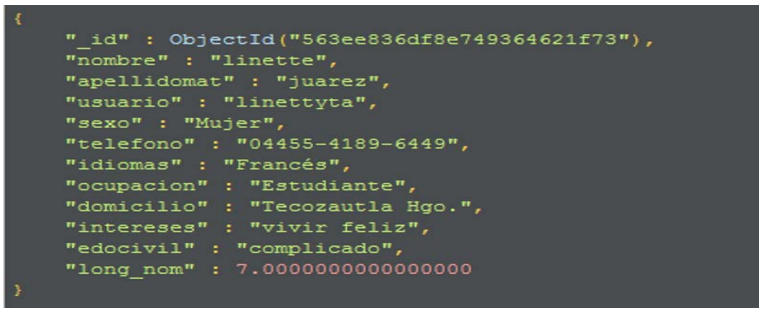
Después de realizar la segunda parte del algoritmo el documento tiene dos registros adicionales, uno para la longitud y otro para el peso. Tal como lo muestra la Figura 4.1.3.

4.2. Búsquedas

Dada una consulta por rango $Q = (q, r)$, donde q no necesariamente debe ser un elemento del espacio métrico definido por la BD y r el radio de búsqueda el cual

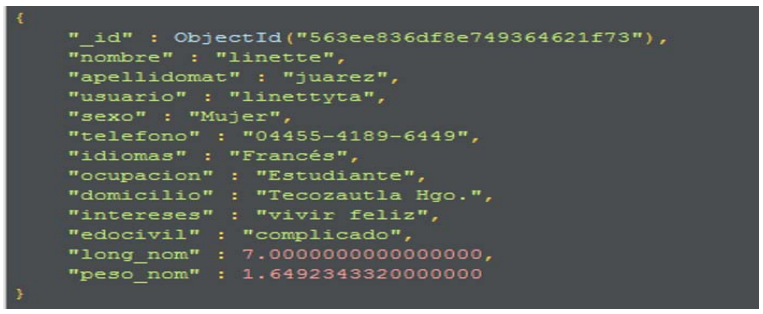
(6) ObjectId("563ee836df8e7493...	{ 11 fields }	Object
_id	ObjectId("563ee836df8e749364621f73")	ObjectId
nombre	linette	String
apellidomat	juarez	String
usuario	linettyta	String
sexo	Mujer	String
telefono	04455-4189-6449	String
idiomas	Francés	String
ocupacion	Estudiante	String
domicilio	Tecozautla Hgo.	String
intereses	vivir feliz	String
edocivil	complicado	String

FIGURA 4.1.1. Documento de la BD antes de ejecutar el algoritmo.



```
{
    "_id" : ObjectId("563ee836df8e749364621f73"),
    "nombre" : "linette",
    "apellidomat" : "juarez",
    "usuario" : "linettyta",
    "sexo" : "Mujer",
    "telefono" : "04455-4189-6449",
    "idiomas" : "Francés",
    "ocupacion" : "Estudiante",
    "domicilio" : "Tecozautla Hgo.",
    "intereses" : "vivir feliz",
    "edocivil" : "complicado",
    "long_nom" : 7.0000000000000000
}
```


FIGURA 4.1.2. Documento actualizado después de ejecutar la primera parte del algoritmo.



```
{
    "_id" : ObjectId("563ee836df8e749364621f73"),
    "nombre" : "linette",
    "apellidomat" : "juarez",
    "usuario" : "linettyta",
    "sexo" : "Mujer",
    "telefono" : "04455-4189-6449",
    "idiomas" : "Francés",
    "ocupacion" : "Estudiante",
    "domicilio" : "Tecozautla Hgo.",
    "intereses" : "vivir feliz",
    "edocivil" : "complicado",
    "long_nom" : 7.0000000000000000,
    "peso_nom" : 1.6492343320000000
}
```


FIGURA 4.1.3. Documento con registros actualizados después de ejecutar el algoritmo completo.

equivale al número de objetos que devolverá la consulta. Primero se calcula $L(q)$, y se aplica la métrica d_{long} de q a todos los centros establecidos inicialmente, es decir,

$$d_{long}\left(q, C_i\right) =\mid L\left(q\right) - C_i \mid,$$

en esta parte se proyectará a los objetos que cumplan lo siguiente

$$d_{long}\left(q, C_i\right) =\mid L\left(q\right) - C_i \mid\leq 1.$$

Posteriormente se calculará el peso de q y se aplicará el *criterio de ajuste*. Una vez obtenidos los valores correspondientes a cada caso se identificará a que nodo hijo se debe accesar.

Finalmente se evaluarán los objetos que se encuentren en cada hoja y se devolverán como resultado final los objetos x_i que tengan $n-1$ caracteres en común con q.

La ecuación 4.2.1 describe las condiciones que deben cumplir los x_i a evaluar.

$$\begin{aligned} B\left(q, r\right)_3 := \big\{ x_i \in X : d\left(q, x_i\right) \leq 1 \text{y} W\left(q\right) \in p_{i_\pm} \\ \text{y } x_i = q \text{ en } \left(n-1\right) c_k\text{'s} \big\}. \end{aligned} \tag{4.2.1}$$

El pseudocódigo utilizado para ejecutar la búsqueda se muestra a continuación.

Inicio

0) Dada una consulta $R(q, r)$.

1) Obtener $L(q)$.
Si $d(C_i, q) \leq 1$, **retornar todos los resultados en caso contrario ignorar.**

2) Obtener $W(q)$ **y aplicar el criterio de ajuste a** $W(q)$
Ingresar al nodo interno que contenga a los valores de a cuerdo al criterio de ajuste.
Devolver todos los resultados de esos subconjuntos.
Comparar los x_i **con** q **y devolver sólo los que coincidan en** $(n-1)$ **caracteres.**
Ordenar primero por longitud y después por peso.
Mostrar como resultado final los r **documentos que se soliciten en la consulta.**

Fin.

4.3. Experimentación

En esta sección del trabajo se presentan los experimentos y los resultados que se fueron obteniendo al realizar las pruebas necesarias para maximizar los recursos computacionales sin afectar la efectividad de la consulta. Para la implementación de los algoritmos se utilizó MongoDB 3.0.7 como SGBD y el entorno de desarrollo fue Robomongo 0.8.5. La BD que se utilizó contiene 1000582 documentos.

Para el análisis en este trabajo cada documento se modeló como un elemento del espacio métrico y la búsqueda por similitud consistió en encontrar documentos que sean más parecidos con un objeto de consulta q dado, el registro que se utilizó como referencia fue "nombre". El procedimiento que se llevo a cabo para este registro puede ser implementado en otro e inclusive pueden tomarse varios registros y aplicar el mismo criterio.

* Caso 1. Se propusieron búsquedas con nombres de longitud más usual, uno de ellos fue oscar y los resultados que proyecto la consulta fueron: casto, oscar, oskar, osmar y carlos. Otra de ellas karla y su resultado fue cala,

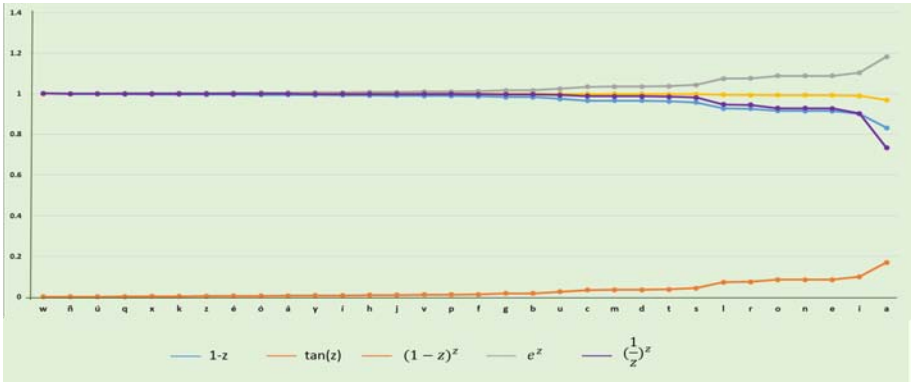


FIGURA 4.3.1. Comportamiento de los pesos al evaluarlos en varias funciones.

lara, carl, caila, carla, clara, carol, laura, karla, karly, karlaa, carola, carlos y charly.

* Caso 2. Se realizaron búsquedas con nombres de longitud no tan común, por ejemplo antonela y su resultado fue antolina, antonela, antonieta, donatela, leonena y antoniaa.

Una de las características más sobresalientes del algoritmo que se propone en este trabajo, es que no se ha considerado a las palabras como una cadena de caracteres con un orden establecido. En esta propuesta a las palabras se les consideró cómo una sucesión de caracteres independientes, en donde el valor de los mismo sólo depende de la frecuencia con la que son utilizados.

Una vez que se detectó la importancia de conocer la frecuencia con la que se utiliza cada carácter, dentro del mismo algoritmo de búsqueda se implementó un procedimiento para conocer las veces que se repite cada uno. Posteriormente se evaluaron esos valores en distintas funciones, para poder definir una métrica que se utilizara en la consulta.

Las primeras funciones que se propusieron fueron evaluadas en intervalos de $(0, 1.5)$, debido a que los valores que se obtuvieron al buscar la frecuencia están en ese intervalo y las evaluaciones de la mayoría también. Las primeras funciones propuestas se encuentran en la Figura 4.3.1.

Se puede observar que en algunas de las funciones de la imagen 4.3.1 los valores obtenidos son tan cercanos que no aportan mucha información al momento de utilizarlos, por ello se utilizaron funciones con intervalos donde sus valores en el codominio tienen una diferencia muy grande aunque en el dominio la diferencia se pequeña. Estas funciones pueden apreciarse en la Figura 4.3.2.

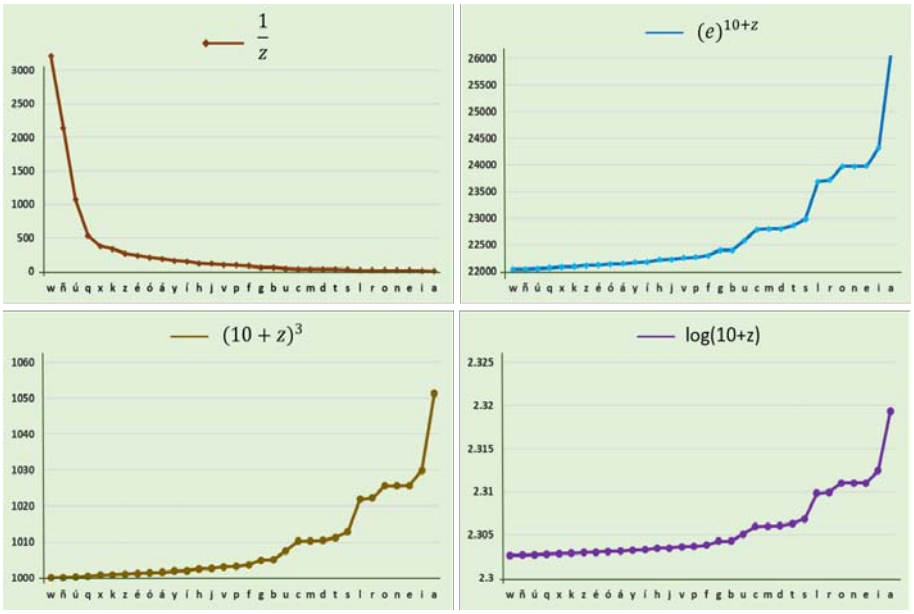


FIGURA 4.3.2. Pesos aplicados a funciones sin rangos específicos.

Se ha considerado que en la función $\exp^{10+z}$ las diferencias son más grandes, debido a ello se eligió esta función , sin embargo para utilizar valores pequeños, esta vez se recorrió al intervalo $(0, 1)$.

El algoritmo se implemento de dos maneras diferentes, la primera fue sin utilizar un *criterio de ajuste* y la segunda fue implementando el uso de una variable.

En la gráfica 4.3.3 se muestran los objetos x_i que cumplen las condiciones de la ecuación 3.3.1 y los que se presentan al final como resultado de la consulta, es decir, los que cumplen los requisitos de la ecuación 4.2.1 son los que están en el recuadro verde. Durante la ejecución de estas pruebas no se aplicó el criterio de ajuste.

Mientras se llevaban a cabo los experimentos se detectó que cuando se extraían objetos que tenían una longitud distinta a la de q, se hacían muchas comparaciones con ellos y al final muy pocos de ellos cumplían las condiciones de la última ecuación. Por ello fue necesario generar una variable que equilibrará la proximidad entre el peso de q y los valores ya guardados en los nodos internos. Después de implementar el criterio de ajuste la consulta fue más efectiva, ya que el numero de comparaciones

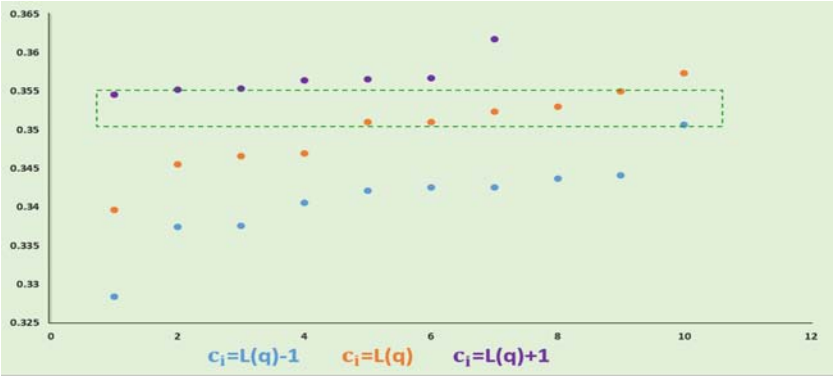


FIGURA 4.3.3. Representación del algoritmo durante la consulta.

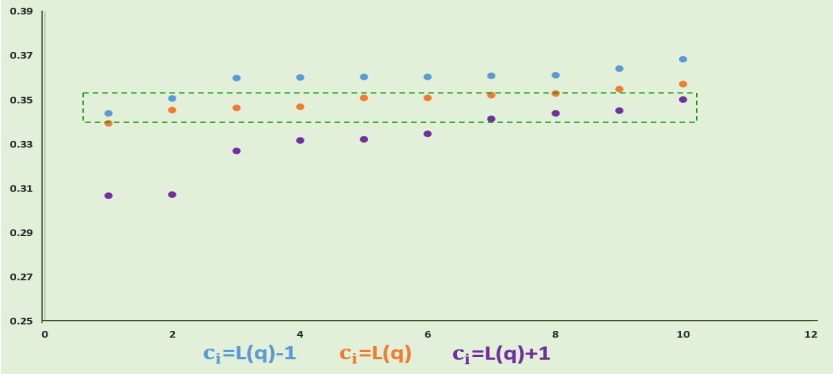


FIGURA 4.3.4. Representación del algoritmo utilizando un criterio de ajuste.

en algunos casos fue igual o menor pero en el resultado final el número de objetos proyectado incremento.

La Figura 4.3.4 muestra las gráficas de los resultados obtenidos al utilizar la variable α, la gráfica muestra los objetos que cumplen la ecuación 3.3.1 y los x_i que se proyectan al final de la consulta, estos últimos están dentro del recuadro verde.

Se puede observar que en la consulta donde se utiliza una variable para ajustar los pesos los objetos que se muestran al final de la consulta son más próximos al objeto de consulta q, de hecho incrementa considerablemente el número de objetos que tienen un longitud distinta a q.

Para evaluar la efectividad de las propuestas se realizaron varias pruebas en cada una de ellas y se consideraron factores tales como el número de objetos que se extraían en la primera parte del índice, el número de objetos que se proyectaban

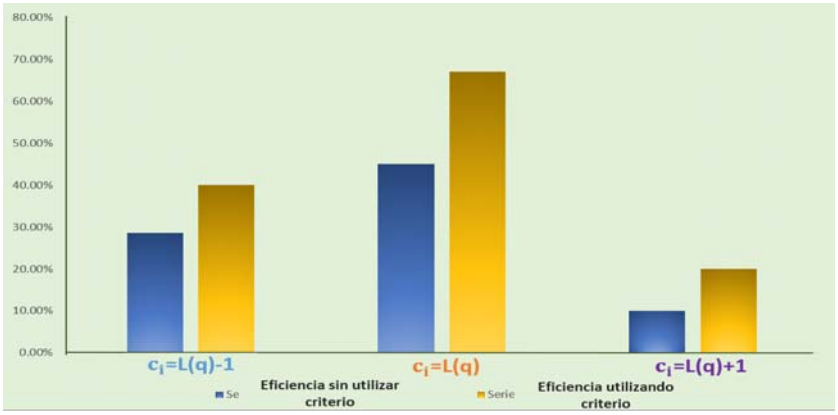


FIGURA 4.3.5. Comparación de la eficiencia al implementar ambos algoritmos.

como resultado de la consulta y el tiempo de ejecución. Después de evaluar y analizar las dos implementaciones, se realizó comparación de su efectividad.

La gráfica 4.3.5 contiene el porcentaje de la eficiencia al implementar los dos algoritmos de indexación.

Finalmente luego de comprobar que nuestra propuesta mejora el resultado de la consulta, se incremento el número de caracteres, documentos y colecciones con la finalidad de verificar si se continuaba con la misma tendencia. Los resultados y conclusiones finales se comentarán a continuación.

4.4. Futuras Implementaciones del Algoritmo

Si bien el presente trabajo se desarrolló en una base de datos estática, con un escenario que contempla un conjunto de cadenas de texto, su peso y longitud. Consideramos sería interesante crear ambientes de bases de datos dinámicas o que aún estén vaciás y modificar los algoritmos para soportar inserción o modificación de datos en tiempo real.

Por otro lado, el algoritmo de búsqueda que se propone en este trabajo se enfoca principalmente en la frecuencia con la que se utilizan los caracteres de un lenguaje determinado. Debido a ello se considera que puede ser implementado en bases de datos que contienen otro tipo de objetos, por ejemplo imágenes.

Se hace esta suposición porque se toma en cuenta que las imágenes se forman utilizando una parrilla de pixeles, paletas con una cantidad específica de colores y/o utilizando profundidades de color de un número de bits establecido. Si se toma

a este conjunto de elementos como un *"lenguaje"*, se cree que el algoritmo puede funcionar aplicando el mismo criterio para búsquedas por similitud.

Finalmente se reconoce que hay un campo de investigación muy importante relacionado con problemas "reales" ya conocidos e incluso resueltos, que pueden ser representados con varios modelos matemáticos para tener un sustento teórico que permitirá ofrecer una gama de soluciones a estos problemas y así hacer uso eficiente de cada recurso.

Conclusiones

A lo largo del presente trabajo se constató que explotar datos no estructurados implica un reto que debe contemplar diversas variantes, tales como el almacenamiento, el análisis y manipulación adecuada de los mismos, sin dejar de lado las necesidades y objetivos de las organizaciones. Por ello es necesario contemplar que estas acciones se puedan llevar a cabo en tiempo real o con datos almacenados en un Data WareHouse (DWH).

Se detectó, qué es importante analizar detalladamente las características de los SGBD, pues cumplen una función vital para una buena administración de los datos. En cuanto a este trabajo el enfoque principal fue a manejadores de datos no estructurados. Se encontró que al igual que otros tipos de manejadores, éstos también se apoyan del uso de indices para mejorar el rendimiento de una consulta.

Al trabajar con los datos, se observó que uno de los instrumentos más importantes de una BD, es el índice, ya que al utilizarlo se puede mejorar el rendimiento de una consulta. Esto se debe a que en él sólo se encuentra la información de donde está almacenado un objeto dentro de la base.

Indexar una base puede generar beneficios muy significativos, sin embargo un mal uso de ellos puede generar lo contrario. Tal como se percibió en la primera ejecución del algoritmo, la indexación que se habıa implementado no fue útil para las consultas realizadas durante las pruebas.

En cuestión de tiempo y recursos computacionales, se notaba una mejoría importante, sin embargo al analizar los objetos que proyectaba la consulta se verificó que el índice estaba afectando los resultados obtenidos. En resumen, los datos que se proyectaban al final de las consultas no contenían ni el 50 % del total que se debería obtener.

Situaciones como éstas permiten concluir que una indexación incorrecta puede afectar al análisis de los datos.

Además se debe tener en cuenta que un índice ocupa espacio dentro de la base, por ello es importante definir índices que se acoplen a la estructura de la BD y que

reduzcan el consumo de recursos tales como uso de CPU, uso de memoria física, cantidad de registros, tiempo de ejecución, entre otros; al realizar una consulta.

Otra ventaja que se tuvo al momento de implementar el índice, fue que el manejador permite administrar datos semiestructurados, los cuales se pudieron representar mediante un modelo matemático para poder manipularlos indistintamente.

La implementación de las búsquedas por similitud en este trabajo se modeló utilizando espacios métricos, debido a que estos permiten determinar la similitud entre dos objetos mediante una función distancia llamada métrica. Se comprobó que se podían utilizar tantas como se necesitaran, sin afectar o modificar un objeto de la BD.

Finalmente se pudo comprobar que no hay un método de indexación mejor que otro, ni una métrica más eficiente que otra; tanto los índices como las métricas se pueden modificar tanto como se quiera o necesite, siempre con la finalidad de optimizar los recursos durante las consultas. Se ha considerado que si un gestor de bases de datos permite utilizar diversos métodos de análisis e implementar procesos de indexación entonces, se tienen las condiciones necesarias para la implementación de métodos de búsqueda por similitud, los cuales pueden ser útiles para la automatización de procedimientos u orientados a características y tareas específicas.

Bibliografía

[1] G Beekman, E Peake, and R Rivera. *Computación & informática hoy: una mirada a la tecnología del mañana*. Addison-Wesley Iberoamericana, 1995.

[2] S. K. Sudarshan. Fundamentos de bases de datos. *Mc Graw Hill*, 2002.

[3] D Rowland and E Macdonald. *Information Technology Law*. Cavendish, 2005.

[4] J Hurwitz, A Nugent, F Halper, and M Kaufman. *Big data for dummies*. John Wiley & Sons, 2013.

[5] K Davis. *Ethics of Big Data: Balancing risk and innovation*. .ºReilly Media, Inc.", 2012.

[6] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity search: the metric space approach*, volume 32. Springer Science & Business Media, 2006.

[7] TM Apostol. *Análisis matemático*. Reverté, 1996.

[8] W. Rudin and L. B. García. *Análisis funcional*. Reverté, 1979.

[9] JM Moreno. *Introducción a la topología de los espacios métricos*. Universidad de Cádiz Servicio de Publicaciones, 1998.

[10] FG Merayo. *Matemática discreta*. Paraninfo, 2001.

[11] JF Becerra and AG Sandoval. *Matemáticas Discretas: Aplicaciones y Ejercicios*. Ingeniería y Ciencia Básicas. Grupo Editorial Patria, 2014.

[12] WA Burkhard and RM Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, 1973.

[13] R Baeza-Yates, W Cunto, U Manber, and S Wu. Proximity matching using fixed-queries trees. In *Combinatorial Pattern Matching*, pages 198–212. Springer, 1994.

[14] R Baeza-Yates. Searching: an algorithmic tour. *Encyclopedia of Computer Science and Technology*, 37:331–359, 1997.

[15] E Chávez. Optimal discretization for pivot based algorithms. *Manuscript. ftp://garota. fismat. umich. mx/pub/users/elchavez/minimax. ps. gz*, 1999.

[16] E Chávez and G Navarro. A compact space decomposition for effective metric indexing. *Pattern Recognition Letters*, 26(9):1363–1376, 2005.

[17] PN Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, volume 93, pages 311–321, 1993.

[18] T Bozkaya and M Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *ACM SIGMOD Record*, volume 26, pages 357–368. ACM, 1997.

[19] EV Ruiz. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recognition Letters*, 4(3):145–157, 1986.

[20] ML Micó, J Oncina, and E Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15(1):9–17, 1994.

[21] I Kalantari and G McDonald. A data structure and an algorithm for the nearest point problem. *Software Engineering, IEEE Transactions on*, (5):631–634, 1983.

[22] FK Dehne and H Noltemeier. Voronoi trees and clustering problems. *Information Systems*, 12(2):171–175, 1987.

[23] JK Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information processing letters*, 40(4):175–179, 1991.

[24] S Brin. Near neighbor search in large metric spaces. pages 574–584, 1995.

[25] P Ciaccia, M Patella, and P Zezula. Deis-csite-cnr. In *Proceedings of the... International Conference on Very Large Data Bases*, volume 23, page 426. Morgan Kaufmann Pub, 1997.

[26] G Navarro. Searching in metric spaces by spatial approximation in pro. In *6th South American Symposium on String Processing and Information Retrieval (SPIRE'99)*, pages 141–148, 1999.

[27] M Mamede. Recursive lists of clusters: A dynamic data structure for range queries in metric spaces. In *Computer and Information Sciences-ISCIS 2005*, pages 843–853. Springer, 2005.

[28] N Ziviani. *Diseño de algoritmos con implementaciones en Pascal y C*. Editorial Paraninfo, 2007.

[29] P. Warden. *Big data glossary*. O'Reilly Media, Inc., 2011.

[30] M Manoochehri. *Data Just Right: Introduction to Large-scale Data & Analytics*. Addison-Wesley, 2013.