Universidad Michoacana de San Nicolás de Hidalgo

Facultad de Ingeniería Eléctrica

INDEXADOR DE TRIXELES UTILIZANDO ÁRBOLES-B BASADOS EN ARCHIVOS CON ACCESO ALEATORIO

TESIS

Que para obtener el grado de

INGENIERO EN COMPUTACIÓN

Presenta

Bryan Eduardo Martínez Guzmán

Jaime Cerda Jacobo

Doctor en Ingeniería Eléctrica y Electrónica

Director de Tesis

Morelia Michoacán Febrero 2016

Quiero agradecer a la Facultad de Ingeniera Eléctrica de la UMSNH por el tiempo brindado en sus aulas, durante estos 5 años,

Así como a mi tutor y asesor el Dr. Jaime Cerda Jacobo, por los conocimientos brindados para la realización de este trabajo, y la amistad cosechada en este lapso de tiempo,

Así como al MC. José Maria Valencia Ramirez, por su importante apoyo en la realización de este trabajo,

Así como a mi madre Jackeline Guzmán Reyes, mi abuela Victoria Reyes Balverde, y mi tío el Ing. A. Ramiro Guzmán Reyes, por su apoyo mientras fui estudiante.

Así como a mis amigos de la facultad, con los que pude contar cuando los necesitaba Yessica Calderón Torres, Ivan Valdovinos Garcia, Gerardo Mauricio López y Miguel Angel Arrollo Villalobos.

Resumen

Los indexadores son una herramienta de software que ayudan a completar palabras, frases, o bien obtener archivos de texto o archivos multimedia como imágenes, audio, videos, etc. de una manera rápida. El objetivo de los indexadores es disminuir el costo de recuperación de los archivos, y así disminuir el número de operaciones de I/O al disco duro. El indexador implementado usa un árbol-B como motor de búsqueda, donde cada nodo del árbol tiene internamente otro índice, para generar la referencia a los archivos que se almacenan por parte del servidor que utiliza como base de datos un archivo de acceso aleatorio. En este trabajo se propone un enfoque diferente a los indexadores comunes, ya que el indexador, utiliza referencias a posiciones geoespaciales, es decir, puntos localizados sobre la superficie de la esfera terrestre. De esta manera se crea una relación entre el algoritmo de malla triangular jerárquica, y el sistema de coordenadas latitud-longitud para indexar con los parámetros geoespaciales que utilizan los indexadores en la actualidad. Finalmente, se muestran algunos ejemplos de cómo se obtiene el conjunto de trixeles a partir de un archivo a indexar, además se muestra la expansión del archivo de acceso aleatorio cuando se va acercando a su límite definido en bytes.

Palabras clave: búsquedas geoespaciales, árboles de expansión, archivo de acceso aleatorio, indexación, recuperación.

Abstract

Indexers are software tools to help complete words, phrases, or obtain text files or multimedia files such as pictures, audio, videos, etc. in a quick manner. The goal of indexing is to reduce the cost of file recovery, by reducing the number of I/O operations to the hard disk. The indexer implemented uses a B-tree as search engine, where each node has internally another index to generate the reference to files that are stored on the server side, which database used is a random access file. In this document a different common indexing approach is proposed of the indexer, uses references to geospatial positions, i.e. points located on the surface of the earth sphere. With that relationship between the hierarchical triangular mesh algorithm is establish, and the system of latitude-longitude coordinates for indexing the geospatial parameters used by indexers today. Finally, some examples area shown, how the set of trixeles is obtained from a file indexing is also expanding the random access file is displayed when it approaches its limit defined in bytes.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Antecedentes

Los indexadores son herramientas de software diseñadas para almacenar documentos textuales y proveer medios para su recuperación a partir de palabras o frases de consulta. Los indexadores elaboran listas metódicas ordenadas que permiten encontrar información en un documento, por lo tanto *indexar* hace referencia cuando se obtienen los parámetros necesarios para que un archivo sea almacenado por un indexador y se tengan los medios necesarios para su recuperación. Una de las características de los indexadores, es que en general están preparados para trabajar con un gran volumen de información, por ello, implementan filtros de texto que sirven para limitar el contenido a **términos**. Un término es una palabra clave que sirve como identificador de búsquedas, con el cual se pueden generar relaciones de los documentos almacenados [6].

Existen múltiples herramientas clasificadas como indexadores, pero solo algunas se destacan y marcan tendencias globales por las características que integra el indexador, por ejemplo, es importante la forma en la que son creadas las listas metódicas, porque la velocidad de respuesta del indexador, depende de lo rápido que se puede acceder al conjunto de documentos y de la cantidad de operaciones que se deben realizar [6]. En las últimas décadas algunos indexadores de texto han sobresalido, por ejemplo se encuentran:

- Lucene Apache Solr
- Xapian
- ElasticSearch

- Net Search Extender (IBM)

El indexador Apache Solr es uno de los principales, este recibe documentos XML, CSV y JSON como entrada, los conduce a través de un conjunto de filtros y analizadores de texto, para generar un **índice invertido**, sobre el cual se ejecutan los algoritmos de búsqueda [30].

Un **índice invertido** es un índice de almacenamiento de estructura de datos a un mapeo de contenido como palabras o números, a su ubicación en una de base de datos, un documento o un conjunto de documentos. La meta de un índice invertido es permitir búsquedas de texto completo rápidas, a un costo de un mayor procesamiento cuando se añade un documento a la base de datos.

Otro tipo de indexadores son los que están incluidos por defecto en los sistemas operativos, como Mac Os, Linux, Windows, Android, etc. Estos emplean las ventajas de los índices para mantener el control de archivos como documentos, audio, imágenes, y otros archivos multimedia que se encuentran almacenados en el dispositivo.

La ventaja de los indexadores es la velocidad para encontrar los archivos, pero la desventaja, se relaciona al tiempo que se emplea para crear los índices de los archivos [5]. Algunos factores que intervienen son:

- La cantidad a archivos a indexar.
- La cantidad de términos que son designados como palabras clave.
- El volumen de información que está contenida dentro del documento, archivo, etc.

La característica de un indexador de texto, es que además de regresar una aproximación de palabras o frases, también puede regresar los documentos que se están solicitando, y se ofrezcan sugerencias de otras palabras o documentos que están relacionados, a base de los términos que ingresamos como búsqueda. Algunos indexadores como Solr, Xapian entre otros, proporcionan estas relaciones tomando en cuenta la siguientes funciones:

- Facetas (Facets). Las facetas son términos semejantes que resaltan características específicas, por ejemplo, búsquedas por color *azul* que son de tamaño *pequeño*, u otras características que definen el modelo.

- Semejante (Like). Es una manera de semejar objetos, se usan como parámetros algunas características que definen el documento y mientras cumpla con alguna(s), se obtiene su referencia, por ejemplo, al solicitar los libros que contengan las palabras *madera* y *morelia*, obtendrán los documentos que contengan esas palabras sin importar el tipo de clasificación al cual corresponden.

- Borroso (Fuzzy). Es una manera de buscar objetos que utilizan un rango definido que se puede extender o reducir una sección de búsqueda, por ejemplo, al buscar *abeja* obtendremos resultados relacionados como: *abejorro*, *avispa* y *avispón*.

Para mejorar el esquema de los indexadores es necesario reducir los accesos de lectura y escritura sobré archivos (operaciones de I/O), y también reducir el consumo de memoria, con esto se reduce la *métrica* que se relaciona al costó de recuperación de los archivos entre la distancia de objetos en el indexador y se garantizan mejores resultados [6].

Un indexador puede ser capaz de realizar búsquedas por proximidad, donde la proximidad se refiere a obtener los documentos que se asemejan utilizando algunos parámetros, como palabras en común, fecha de publicación, lugar de publicación, autores, etc. Las búsquedas por proximidad se relacionan principalmente con tres tipos de consultas, las cuales son:

- Rangos de las consultas. Se obtienen todos los documentos que son especificados por un intervalo de valores, por ejemplo, un rango alfabético de la A-D, o también todas las palabras que se encuentran en el rango entre *facultad* a *universidad*.

- Los vecinos más cercanos. Se reciben todos lo documentos que se encuentran lo más cercano desde un punto de origen específico, y puede ser necesario u opcional usar un límite de documentos máximo y/o mínimos que se obtengan.

- Los K vecinos más cercanos. La idea de está consulta es obtener los vecinos más cercanos ordenados por conjuntos.

El costo al utilizar los indexadores se relaciona respecto a la siguiente ecuación [6]:

$$T = n \times t_a + t_c + t_o \tag{1.1}$$

Donde :

n, número de operaciones que se requiere para recuperar los archivos.

t_a, complejidad del algoritmo implementado.

t_c, tiempo de procesamiento del CPU.

t_o, tiempo de las lecturas al disco (operaciones de I/O).

La meta del indexador es minimizar T. En muchas ocasiones, la evaluación de las distancias n, es tan costoso que los otros componentes de la fórmula pueden ser despreciados. Los indexadores se utilizan principalmente para crear el índice de documentos a bases de datos, y pueden ser utilizados para crear índices de zonas en la esfera terrestre, utilizando el identificador y el sistema de coordenadas geoespaciales. A lo largo de la historia se han implementado diferentes estrategias para crear un modelo de la esfera terrestre, por ejemplo, utilizando conjuntos de triángulos llamados **trixeles** [10], con los cuales se pueden utilizar para reconstruir zonas y la curvatura del planeta. Por otra parte, también se han utilizado otros métodos para subdividir la esfera terrestre, utilizando como característica que el área entre zonas sea igual [15], finalmente otro modelo es encerrar la esfera terrestre dentro de un cubo, y con esto tener un sistema que se basa en la proyección de las caras de cubo, hacia la superficie terrestre [16].

Los indexadores que son utilizados para crear índices de documentos multimedia como archivos de video, audio, pdf, etc. Además de solo texto, requieren una mejor estrategia para crear sus índices, la razón es que cada documento gasta tiempo de CPU, con lo cual la escritura y recuperación de los archivos se puede volver lenta. Para eliminar este problema algunas compañías como Facebook, implementan en sus bases de datos, archivos con acceso aleatorio, para almacenar las fotos de los usuarios en un solo archivo, e implementan un sistema de recuperación, de está forma, se tiene un archivo que se puede utilizar para escribir y recuperar las imágenes, realizando más eficiente las operaciones de disco [20].

Finalmente, es importante mencionar que la estructura del indexador, en expansión se asemeja a un ***Quadtree*** [23]. Un Quadtree es una estructura de datos de árbol, en

la que cada nodo interno tiene exactamente cuatro nodos hijos. Los Quadtrees son los más utilizados para dividir un espacio por recursividad en cuatro cuadrantes o regiones. Está estructura de datos fue nombrado por Raphael Finkel y JL Bentley en 1974. Algunas características que tiene este árbol son:

- Descomponen el espacio en celdas adaptables.
- Cada celda tiene una capacidad máxima.

1.2. Justificación

En la actualidad los indexadores de texto para recuperar información o archivos emplean términos de letras, palabras o frases, y los indexadores que trabajan con coordenadas geoespaciales, utilizan la georreferencia cuando se realizan búsquedas por rango, búsquedas por proximidad o búsquedas acotadas. En los tres tipos de búsquedas, especialmente en las búsquedas acotadas, se sobrepone un círculo, cuadrado, rectángulo o cualquier otro polígono convexo centrado sobre una posición geoespacial, y se toma como referencia las coordenadas de un documento, una zona o cualquier otra posición definida para obtener todos los documentos que se encuentren dentro de la figura, al igual se puede incluir opcionalmente un número máximo y/o mínimo de documentos que se regresan con el indexador [31]. De está manera es como un indexador de texto trabaja con las coordenadas geoespaciales para realizar búsqueda por zonas. Por lo tanto se puede decir que los indexadores en la actualidad que utilizan georreferencias, emplean las coordenadas geoespaciales para sobreponer polígonos convexos, y así recuperar información, dejando todo el procesamiento a análisis de cadenas de texto, a diferencia del indexador propuesto, porque usa los identificadores de las posiciones geoespaciales para crear el árbol de indexación, con lo cual se realizan mas rápidas las búsquedas, porque solo accede a una referencia y no realiza cálculos sobre cadenas de texto.

Un indexador que tiene un lista metódica de las posiciones geoespaciales, puede regresar los documentos que se encuentran en un punto específico. El indexador propuesto funciona de está manera, será rápido con las búsquedas por zonas, porque indexará a nivel de **trixeles**, utilizando la posición geoespacial de los archivos, cada trixel podrá tener un conjunto de archivos indexados, y se implementaran dos tipos de búsquedas, las cuales son, búsquedas por rango y búsquedas por proximidad. Se limitarán los accesos al disco con

operaciones de I/O, por que el indexador implementa archivos con acceso aleatorio como bases de datos, y para la recuperación del archivo, se proporciona la ubicación y la cantidad de bytes que se ocupan.

Un ejemplo de está aplicación se relata a continuación: En Morelia, Michoacán, existen muchos lugares dedicados a proveer información impresa como librerías, bibliotecas, puestos de revistas, etc. Que tienen libros, catálogos, folletos, con diferente tipo de información que puede ser científica, cultural, social, recetarios de cocina, etc. Suponiendo que para Morelia, se requieren de varios trixeles para representar y mantener una referencia geoespacial de cada uno de estos lugares usando un identificador. Ahora si se tiene tres librerías A, B y C. Al solicitar todos los libros que se encuentran en la librería A o librería B o librería C, un indexador de texto creó previamente el índice de los libros que pertenecen a Morelia y que pertenece a cada una de las librerías A, B o C, el cual fue costoso en tiempo de computo por razones antes mencionadas. En cambio con este indexador, el índice que se obtiene solo requiere de la posición geoespacial de los documentos, y esto requiere de menos operaciones de computo a diferencia de analizar cadenas de texto. Además en un indexador se debe seguir procesando las cadenas ingresadas para regresar únicamente los documentos de alguna librería, y es más costoso a solamente acceder al trixel que pertenece a cada lugar. Al final se cumple la meta en ambos casos para obtener los documentos que se encuentran en cada lugar. Pero el trabajo que se realiza se puede disminuir a simplemente acceder a una posición de referencia.

El indexador propuesto, puede simplificar el trabajo al filtrar información y posteriormente, se pueden utilizar las ventajas de un indexador de texto, ya que es un buena herramienta que puede servir para filtrar los archivos en conjuntos de menor tamaño.

1.3. Objetivos de la Tesis

1.3.1. Objetivo general

Implementar un indexador que use puntos del sistema de coordenadas latitud y longitud, para crear un índice de referencias a posiciones geoespaciales, utilizando como modelo de la esfera terrestre, el algoritmo de malla triangular jerárquica y con esto mantener una relación con las bases de datos para indexar y recuperar archivos de una manera eficiente, con el fin de disminuir la cantidad de operaciones en el motor del indexador y

disminuir el acceso al disco duro.

En los objetivos particulares se mencionan algunos puntos específicos que se requieren para alcanzar el objetivo general.

1.3.2. Objetivos particulares

Como se mencionó en el objetivo general, está tesis se divide en dos secciones, las cuales hablan de:

- Implementar un indexador que utilice coordenadas geoespaciales para indexar trixeles. Las tareas que se encuentran en este primer objetivo son:
  - El motor del indexador debe ser una estructura de árbol semejante a un Quadtree.
  - Utilizar la menor cantidad de memoria mientras se expande el árbol de indexación.
  - Utilizar la cantidad mínima de accesos en el indexador propuesto para guardar y recuperar archivos almacenados en las bases de datos.
  - Utilizar la menor cantidad de memoria para crear el identificador de los trixeles y realizar operaciones a nivel de bits, para obtener las características que se encuentran codificados dentro del identificador de los trixeles.

- Implementar un archivo de acceso aleatorio que almacene documentos en diferentes formatos y proporcione los métodos para su recuperación. Las tareas que se encuentran dentro de este segundo objetivo son:
  - Almacenar y recuperar diferentes tipos de archivos multimedia como texto, audio, videos o también archivos de word, excel, powerpoint a partir de un solo archivo.
  - Disminuir la cantidad de accesos al disco para recuperar los archivos multimedia utilizando un solo proceso.
  - Utilizar una base de datos con un tamaño en bytes definido por las características del SO.
  - Crear un método para eliminar archivos almacenados en las bases de datos.

1.4. Contenido

El indexador que se propone, esta enfocado para trabajar con coordenadas geoespaciales, utilizando uno de los modelos que sirven para crear un sistema de georreferencia de la esfera terrestre, lo cuales se presentan en el capítulo 2, en este capítulo se mencionan cinco modelos principales que requieren de un algoritmo o algún método para seccionar la superficie de la esfera terrestre. El algoritmo principal utilizado en este trabajo, fue el algoritmo HTM tanto por las características que lo conforman y además que en expansión se asemeja a un Quadtree, este último es una estructura de árboles de expansión y estos se explican de una mejor manera en el capítulo 3. En este capítulo se habla más formal de los árboles de expansión que en esencia son el motor de búsqueda e indexación de los indexadores de texto en la actualidad, de esta manera se llega al capítulo 4, donde se explican las características básicas e implementación del indexador propuesto y la relación con las bases de datos, y en el capítulo 5 se muestran algunos ejemplos para indexar y recuperar archivos, al finalizar en el capítulo 6 se presenta la conclusión general, conclusiones particulares y trabajo futuro.

Capítulo 2

Algoritmo de Malla Triangular Jerárquica

En el capítulo anterior se habló de los objetivos y justificaciones con las cuales se desarrolló este trabajo, se explicaron las características, funcionamiento y el costo computacional de los indexadores para crear el o los índices de palabras o archivos. En este capítulo se va a hablar de las coordenadas geoespaciales y su aplicación, se presentan cinco modelos con los cuales se puede crear un sistema de referencia geoespacial y se puede utilizar dentro de un índice para indexar y buscar usando zonas geoespaciales, de estos algoritmos se resalta al algoritmo de Malla Triangular Jerárquica porque se utiliza como base para la indexación en este trabajo, finalmente se habla de una manera para encerrar un conjunto de puntos que pertenecen a uno o varios trixeles, y obtener un número máximo o mínimo de trixeles generados utilizando un porcentaje de error.

2.1. Coordenadas Geoespaciales

Las coordenadas geoespaciales son en esencia la forma de ubicar un punto sobre la esfera terrestre que está regido por dos coordenadas que son latitud y longitud. La latitud es la distancia que existe entre un punto y el Ecuador, medida sobre el meridiano que pasa por dicho punto, y la longitud proporciona la localización de un lugar, en dirección Este u Oeste desde el meridiano de referencia 0^o, también conocido como meridiano de Greenwich.

Actualmente cualquier dispositivo electrónico aprovecha la ventaja de detectar nuestra posición sobre la superficie terrestre y es capaz de ofrecernos todo tipo de infor-

mación en base a una ubicación de referencia, algunos ejemplos son las empresas como Google, Facebook, etc. Por ejemplo, Google utiliza las coordenadas geoespaciales y ofrece servicios de ubicación como Google Maps o bien al realizar cualquier búsqueda se regresan los resultados más cercanos a la zona de referencia.

Existen varios administradores de bases de datos que se encargan de almacenar las coordenadas geoespaciales (latitud y longitud), o bien un punto de referencia con el que se pueden realizar búsquedas en base a su localización geográfica. Algunos ejemplos de indexadores actuales y administradores de bases de datos que emplean las coordenadas geoespaciales son:

- MongoDB.
- Xapian.
- ElasticSearch.
- Solr.

Estos indexadores y administradores de bases de datos, usan las coordenadas geoespaciales para indexar documentos utilizando un punto geográfico, y con ellos se ofrecen búsquedas basadas por la posición del documento, algunas funciones que se pueden utilizar son:

- Búsquedas por proximidad. Sirven para obtener los documentos ordenados que se encuentran más cerca de un punto geográfico especificado.
- Búsquedas acotadas. Sirven para obtener documentos que se encuentran dentro de un área como rectángulo, círculo o cualquier otro tipo de polígono convexo.

Una de las principales diferencias entre los dos tipos de búsquedas son que las búsquedas por proximidad, regresa usualmente un número fijo de documentos dentro de un rango que puede o no estar limitado. A diferencia de las búsquedas acotadas que son un rango definido y se regresan todos o un número fijo de documentos que se incluyen dentro del polígono convexo.

En las últimas décadas se han desarrollado diferentes maneras para reconstruir la esfera terrestre y tener un modelo de georreferencia. Uno de los primeros modelos para tener referencias de la esfera terrestre son la latitud y longitud. Es poco conocido que no

se puede dividir la superficie de una esfera creando una **teselación** regular definida por los cinco sólidos platónicos, los cuales son:

- Tetraedro, con cuatro caras triangulares equiláteros.
- Cubo, con seis caras cuadradas.
- Octaedro, con ocho caras triangulares equiláteros.
- Dodecaedro, con doce caras pentagonales regulares.
- Icosaedro, con veinte caras triangulares equiláteros.

Una **teselacion** hace referencia a una regularidad o patrón de figuras que recubren o pavimentan completamente una superficie plana que cumple los siguientes requisitos:

- Que no queden espacios vacíos.
- Que no se superpongan las figuras.

El sistema geoespacial por latitud y longitud es simple, por lo que en las últimas décadas se han desarrollado algunas alternativas para tener una referencia más exacta de la tierra, y tener una mejor organización de las zonas o teselaciones que conforman la superficie terrestre. En este trabajo las teselaciones son utilizadas para crear la referencia que se van a utilizar con él indexador propuesto, por esto los siguientes modelos fueron seleccionados, ya que son compatibles con él indexador propuesto asemejándose a las características con el árbol de indexación, este árbol se menciona de una manera mas formal en el capítulo 3, y porque las teselaciones son similares en tamaño, o bien por alguna otra característica que puede hacer al indexador propuesto eficiente, algunos ejemplos para obtener un modelo de la esfera terrestre se muestran a continuación:

- Algoritmo HTM, Herarchical Triangular Mesh.

  El algoritmo de Malla Triangular Jerárquica (HTM) es un método para subdividir la superficie esférica en triángulos casi iguales en forma y tamaño. El algoritmo HTM es una herramienta de consulta poderosa para encontrar los objetos de la esfera terrestre según la ubicación geoespacial. Como se puede ver en la figura 2.1, se muestra la ejecución del algoritmo HTM, donde las líneas negras más gruesas representan el

octaedro que se usa como base para este algoritmo. Dado que este es el método principal que se utilizó para el desarrollo de este trabajo, en la siguiente sección se explican el funcionamiento completo y parámetros para la generación de trixeles con este algoritmo [9].

Figura 2.1: Georreferencias con el algoritmo HTM [40].

- Modelo de Hipparchus Voronoi.

  El matemático ruso Hipparchus Voronoi fue el primero acreditado con la descripción de un mosaico irregular del planeta. Él observó que el uso de regla y compás, son elementales en la construcción geométrica, las cuales podía dividir una superficie plana en una red de polígonos que tienen propiedades muy importantes.

  Voronoi observó que si se toma un conjunto arbitrario de puntos sobre una superficie plana (como un mapa), y conectado ciertos puntos con una línea recta, y luego se dibujan las bisectrices de estas líneas, lo que se obtiene son polígonos convexos que encierran puntos. Estos polígonos se llaman polígonos de Voronoi, y tienen está propiedad significativa:

  – Un punto que cae dentro de un polígono, por definición, se encuentra más cerca del punto central de ese polígono que el punto de centro de cualquier otro polígono.

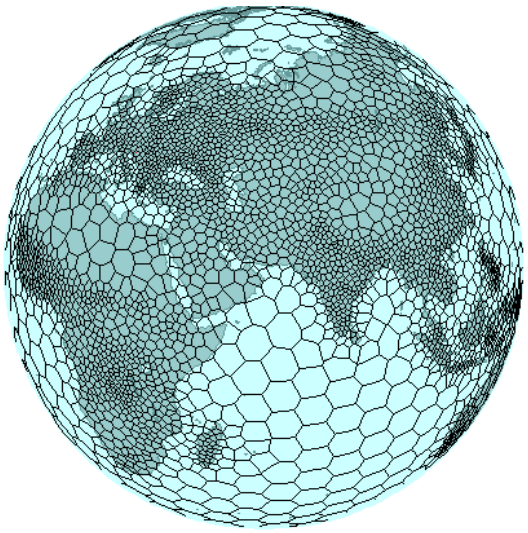

Figura 2.2: Georreferencias con el modelo de Hipparchus Voronoi [41].

Voronoi indexa la esfera terrestre con una estructura que subdivide la superficie con aproximadamente 2500 celdas. En la figura 2.2, se muestra la esfera terrestre dividida utilizando las celdas de Voronoi [11]. Una característica de este método es que donde hay mucha gente, hay muchas celdas pequeñas, y en las regiones inferiores del planeta, donde menos personas residen, hay menos celdas y son más grandes en tamaño.

- Curvas de Hilbert.

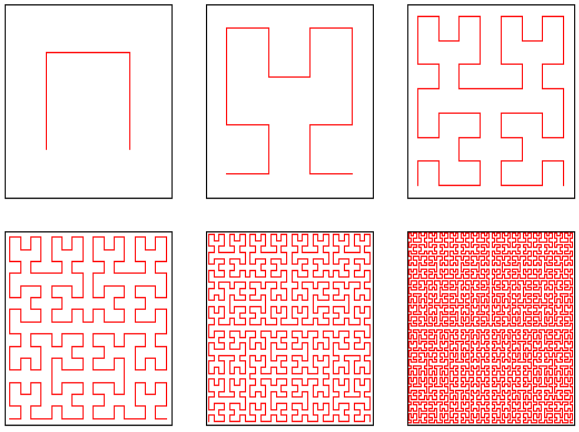

Figura 2.3: Georreferencias con las curvas de Hilbert [42].

El matemático alemán David Hilbert describe está curva en 1891, como una curva fractal continua que recubre el plano, en la que cada una de las curvas que aproximan a la curva final no se corta a sí misma. Este es un tipo de curva que rellena un cuadrado, de tal forma que el inicio de la curva se encuentra en la esquina inferior izquierda y el final se encuentra en la parte inferior derecha [18]. Su construcción se realiza siguiendo estos pasos:

Se parte de un cuadrado que se divide en 4 subdivisiones iguales. Después, cada subdivisión se numera de forma que dos números consecutivos se asocien a dos subdivisiones contiguas.

Posteriormente, con cada subdivisión se realiza el mismo procedimiento que en el paso anterior, teniendo en cuenta además que la numeración debe hacerse de forma que las primeras subdivisiones que se recorran sean las correspondientes al primer cuadrado recorrido en el paso anterior.

Así se va realizando este proceso un número n de veces suficientemente grande hasta que el tamaño de las áreas de cada subdivisión tienda a cero y la curva tienda a ocupar toda la superficie del cuadrado (la trayectoria de la curva es densa en el cuadrado, y en el infinito igual al cuadrado). En la figura 2.3 se muestra este proceso para dibujar la curva de Hilbert. Está curva es utilizada para la indexación del modelo de la tierra, dividiendo la esfera terrestre por un plano cuadrado, y cada ubicación de los lugares indexados representan una posición con respecto a la línea completa que cubre el plano indexado.

- Método con HEALPix.

  HEALPix es un acrónimo por su significado en ingles Hierarchical Equal Área isoLatitude Pixelization of a sphere que su posible traducción es Pixelización Jerárquica de isoLatitud de Área Igual de una esfera [13]. En la figura 2.4 se puede observar este modelo para dividir la esfera terrestre en teselación del mismo tamaño. La pixelación produce una subdivisión de una superficie esférica en la que cada píxel cubre la misma superficie que cada dos pixeles. Otra característica de la red HEALPix es que los centros de pixeles, se producen en un número discreto de anillos de latitud constante, el número de timbres constante latitudes depende de la resolución de la rejilla HEALPix, y tiene las siguientes tres propiedades:

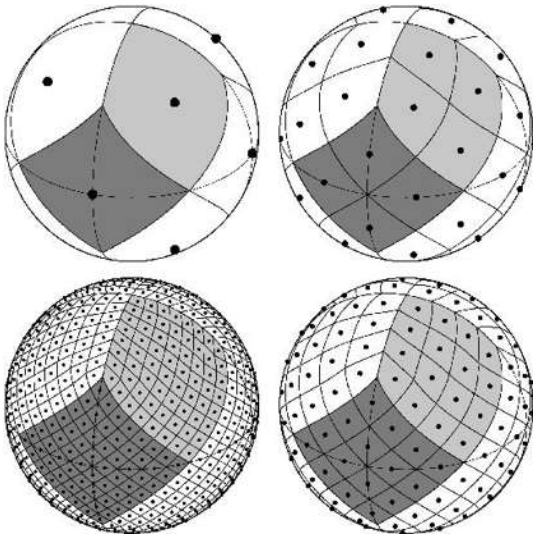

Figura 2.4: Georreferencias con el modelo HEALPix [43].

- La esfera es teselar jerárquicamente en cuadriláteros curvilíneos. La partición más baja resolución está compuesto por 12 píxeles de base, y la resolución de la teselación aumenta por la división de cada píxel en cuatro nuevos.
- El áreas de todos los píxeles con una resolución dada son idénticos.
- Los píxeles se distribuyen en las líneas de latitud constante. Está propiedad es esencial para todas las aplicaciones de análisis de armónicos que involucran armónicos esféricos.

Estos últimos cuatro modelos se pueden utilizar para indexar la esfera terrestre, al igual existen otros un poco más complejos como es el caso del Quadrilateralized Spherical Cube [16], este es un mapeo de áreas equivalentes y el esquema de agrupación de los datos recogidos en una superficie esférica. Se propuso por primera vez en 1975 por Chan y O'Neill para el Fondo para la Investigación Predicción Ambiental Naval, y se puede observar este modelo en la figura 2.5.

2.2. Algoritmo HTM

El algoritmo HTM describe un método para subdividir la superficie de la espera terrestre usando triángulos similares, lo que significa que no están completamente iguales

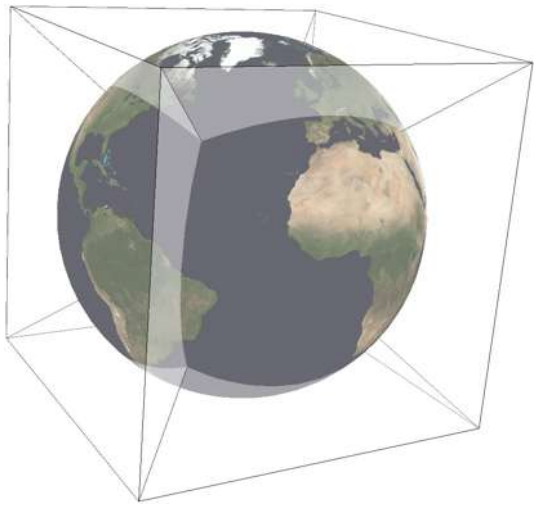

Figura 2.5: Georreferencias con el modelo de Quadrilateralized Spherical Cube [44] .

en tamaño, y proporciona una base para la indexación. Este algoritmo emplea una manera sencilla para modelar completamente la superficie, y no toma en cuenta los problemas de simetrías esféricas por parte de los polos [10]. El modelo de indexación espacial que se obtiene, identifica las secciones por medio de un identificador, y es el punto de partida para la generación de los trixeles.

Para subdividir la esfera terrestre y generar los triángulos, se usa un procedimiento recursivo además del octaedro base, para iniciar la subdivisión de la esfera, inicialmente se cuenta con 8 triángulos de tamaño igual que representa al octaedro antes mencionado, con este se obtiene la primera parte del modelo que se asemeja a tener una esfera con radio de una unidad, por lo que los ejes del plano también tendrán una unidad en cualquier dirección (x, y, z), así los vértices del octaedro son designados respecto de las siguientes posiciones:

$v0 = (0, 0, 1)$ $v3 = (-1, 0, 0)$
$v1 = (1, 0, 0)$ $v4 = (0, -1, 0)$
$v2 = (0, 1, 0)$ $v5 = (0, 0, -1)$

En la figura 2.6 se puede apreciar como se construye un icosaedro que conforma una mitad del modelo de la esfera terrestre usando el algoritmo HTM. En la figura 2.7, se muestra el octaedro que representa el algoritmo HTM. Después de asignar los vértices se dividen por secciones los icosaedros para obtener el cuadrante que representa del ecuador

al polo norte (N), y del ecuador al polo sur (S), así se obtienen las reconstrucciones de los cuadrantes incluyendo los cuatro primeros trixeles como se muestra a continuación:

$$S0 = (v1, v5, v2)\ N0 = (v1, v0, v4)$$

$$S1 = (v2, v5, v3)\ N1 = (v4, vo, v3)$$

$$S2 = (v3, v5, v4)\ N2 = (v3, vo, v2)$$

$$S3 = (v4, v5, v1)\ N3 = (v2, vo, v1)$$

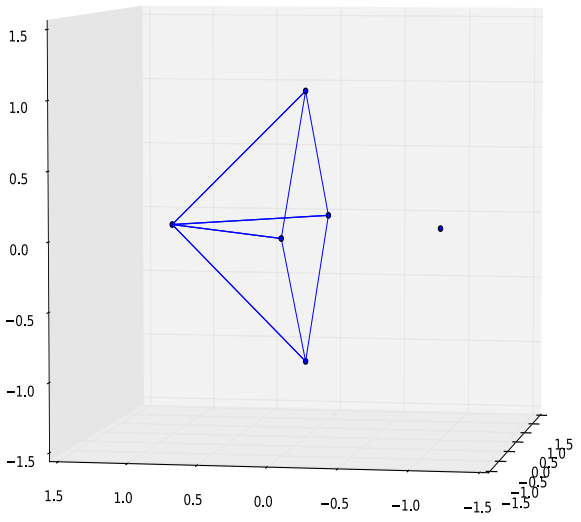

Figura 2.6: Icosaedro del algoritmo HTM.

En la figura 2.8 se muestra el modelo de la esfera terrestre usando un octaedro que se relaciona como una esfera con el algoritmo HTM. El icosaedro azul representa el cuadrante N(Norte), y el icosaedro rojo representa el cuadrante S(Sur). Para calcular los vértices que forman los nuevos triángulos cuando inicia el proceso de recursión, se requieren las ecuaciones 2.1, 2.2 y 2.3 que utilizan el sentido contrario a las manecillas del reloj para obtener un punto intermedio entre la línea que une dos vértices.

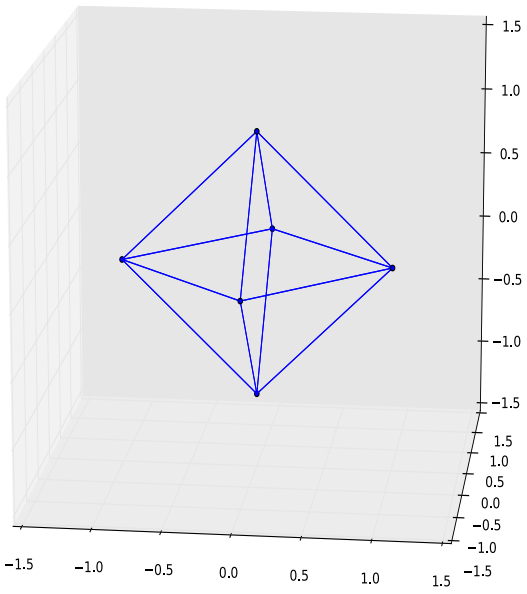

Figura 2.7: Octaedro del algoritmo HTM.

$$w0 = \frac{v1 + v2}{|v1 + v2|} \tag{2.1}$$

$$w1 = \frac{v0 + v2}{|v0 + v2|} \tag{2.2}$$

$$w2 = \frac{v0 + v1}{|v0 + v1|} \tag{2.3}$$

Ahora bien, juntando ambos conjuntos de vértices, los nuevos $w0, w1$ y $w2$, con los anteriores $v0, v1$ y $v2$, se obtienen los triángulos que dividen al cuadrante N, estos vértices de los nuevos triángulos corresponden a:

$$Triángulo0 = (v0, w2, w1)$$
$$Triángulo1 = (v1, w0, w2)$$

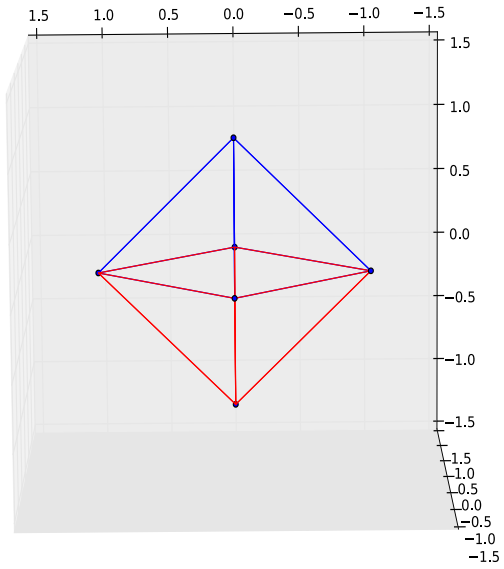

Figura 2.8: Principales cuadrantes del algoritmo HTM.

$$Triángulo2 = (v2, w1, w0)$$
$$Triángulo3 = (w0, w1, w2)$$

En la figura 2.9 se muestra gráficamente este proceso para generar los triángulos, ahí se puede ver que el nombramiento y generación de los nuevos triángulos utilizan el orden contrario a las manecillas del reloj, la recursión puede ser repetida hasta alguna profundidad deseable del número de nodos N del triángulo y dada una profundidad d, donde $d > 0$ se obtiene $N(d) = 8 \times 4^{(d-1)}$, y en el caso de una profundidad igual a cero, la base es una esfera.

De esta manera se empieza a formalizar este modelo de indexación y se describe más formal el término *trixel*. Un *trixel* es un triángulo que se genera durante las subdivisiones del algoritmo HTM. El identificador de los trixeles tiene dos características, una corresponde a la localización en la esfera terrestre y la otra es la profundidad en el modelo de indexación. Para el nombramiento de los trixeles es importante desde el cuadrante en que

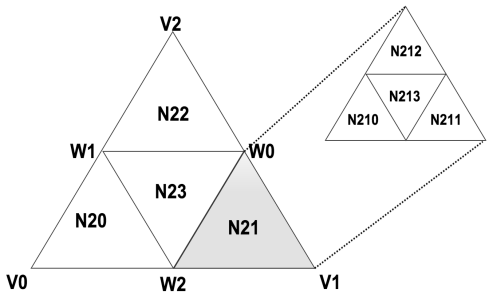

Figura 2.9: Generación y nombramiento de los trixeles.

se localiza y toda la ruta que se obtiene para acceder a el. Algunas maneras para nombrar los trixeles se muestran a continuación:

- Concatenación de cadenas. Se usa N o S para seleccionar el cuadrante al que corresponde el trixel, y para seguir la ruta se usa *0*, *1*, *2* y *3*, un ejemplo de está codificación del identificador *N2013*, el cuadrante se representa por N y la profundidad de 4 niveles es *2013*.

- Codificación binaria. La codificación en binario requiere de dos bits para representar el cuadrante, y dos bits para los niveles, por ejemplo, N se representa con *11*, y S se representa con *10*, en cada nivel se usa *0 = 00, 1 = 01, 2 = 10, 3 = 11*, así que *N2013* se traduce como la siguiente cadena de bits *1110000111*.

- Codificación hexadecimal. Emplea algunas similitudes de la codificación binaria, pero en vez de utilizar dos bits para la codificación, se requieren de cuatro bits, y recordando que en hexadecimal se pueden obtener valores de 0-9 y a-f, pero como es una ruta reducida se usan solamente los valores del *0x0 - 0x3* que representan *0 - 3* y *0xa - 0xb* que representan *N - S*, Por lo tanto la codificación *N2013* se traduce como *0xa2013*.

Como ya se mencionó anteriormente es importante señalar que la profundidad está dada sin tomar en cuenta el cuadrante, por ejemplo, para el siguiente trixel *N0102030232123 2102210231203*, la profundidad está dada por los 26 caracteres *01020302321232102210231203*. La indexación a un nivel 26 con este algoritmo, obtiene un aproximación en la esfera terrestre que equivale a indexar aproximadamente a 30 centímetros cuadrados. Ahora bien teniendo en cuenta como se pueden nombrar los trixeles, y como se realizan las divisiones, es importante mencionar como funciona la generación de los trixeles, pero antes se introduce

mas formal el término de *espacio* en la generación de trixeles, un *espacio* es una superficie o lugar con unos límites determinados y con características o fines comunes, y es importante porque durante la reconstrucción de espacios, por cada operación interviene algún tipo de error que depende de que tan estricto sea el modelo de la reconstrucción, además esto puede hacer lenta la generación, por cada región un trixel puede estar en alguno de los siguientes estados:

- FULL. Un trixel se encuentra dentro completamente de una región.
- PARCIAL. Una parte del trixel se encuentra dentro de la región, pueden ser dos vértices, un vértice o cualquier otra sección que se atraviese dentro del área del trixel.
- OUTSIDE. El trixel no tiene ningún tipo de intersección con una región.

Para la generación de los trixeles y demostrar si un trixel está dentro o no de una región, existen principalmente tres tipos de algoritmos [8], los cuales son:

- Intersección de un trixel con un espacio.
- Intersección de un espacio con el límite de un triángulo.
- Intersección de un trixel con una envoltura convexa.

Para más información de la generación y construcción de la esfera terrestre utilizando trixeles, se puede visitar la siguientes referencias [8], [9] y [10].

2.3. Intersección de un trixel con un espacio

Para conocer si un trixel se encuentra dentro de una región, primero debemos saber cuantos vértices están dentro de una zona. Sí tomamos en cuenta las características antes definidas de FULL, PARCIAL y OUTSIDE, y adaptándonos a este algoritmo podemos decir que un trixel es FULL, únicamente cuando los tres vértices se encuentran dentro de un región, es PARCIAL cuando uno o dos vértices se encuentran dentro una región, y es OUTSIDE cuando ninguno de los vértices se encuentra dentro de una zona.

En la figura 2.10, se muestran los trixeles encerrados por regiones, el inciso a) muestra un trixel cuando es FULL, el inciso b) muestra cuando un trixel es PARCIAL, en el inciso d) se cubre una sección del trixel, pero no llega a ser PARCIAL porque no cubre

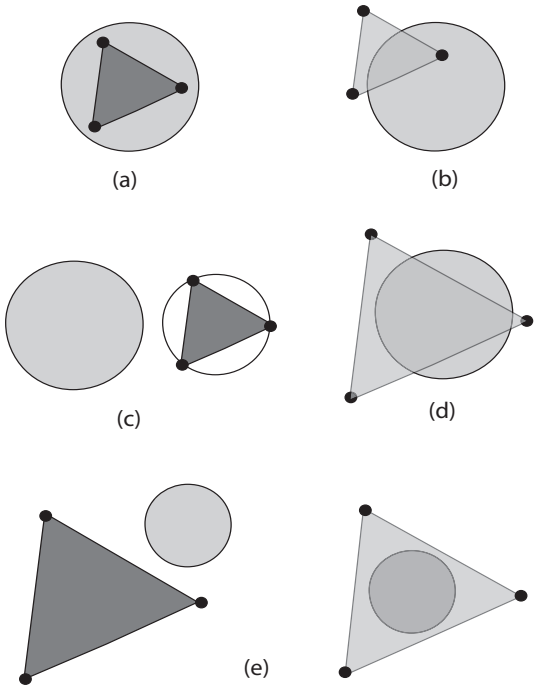

Figura 2.10: Intersección de un trixel con un espacio.

un vértice. El inciso c), muestra cuando un trixel es OUTSIDE pero cumple con todas las características para que pueda ser FULL, solo que se encuentra desfasado. Además el inciso e), muestra dos casos en especiales, en ambos el trixel es más grande que la región, pero el problema es definir si se requiere dividir el trixel, porque se encuentra sobrepuesto o se encuentra OUTSIDE.

El siguiente algoritmo describe el comportamiento de este método de intersección, en este se puede observar que primero se calcula, si en conjunto los vértices pertenecen a FULL, PARCIAL u OUTSIDE. Si es FULL termina el algoritmo, en caso contrario se calcula cuantos vértices y cuanta área del trixel se intercepta con una región, si no hay intersección implica que es OUTSIDE.

Require: Un polígono convexo.
Ensure: Un conjunto de trixeles.
1: **if** Todos los vértices están dentro del polígono **then**
2: **return** FULL
3: **else**

```
4:   if Existe al menos un vértice dentro del polígono then
5:     return PARTIAL
6:   else
7:     if Hay intersección del trixel con los límites de un espacio then
8:       if Algún vértice de un trixel se cruza con alguno de los límites then
9:         return PARTIAL
10:      else
11:        if Alguna área del trixel se cruza con un espacio then
12:          return PARTIAL
13:        else
14:          return OUTSIDE
15:        end if
16:      end if
17:    else
18:      return OUTSIDE
19:    end if
20:  end if
21: end if
```

2.4. Intersección de un espacio con el límite de un triángulo

Este tipo de intersección calcula el área que se encuentra dentro de un región por trixel utilizando las aristas del triángulo. Las aristas de cada triángulo está definidos por sus extremos de los vértices v1, v2. Estos a su vez están conectados por el segmento de una región, debido a que está no es única, se especifica siempre que están conectados por la línea más corta de las dos conexiones de segmento posibles. Los tres vértices de un trixel siempre se encuentran en la misma media esfera. El algoritmo que describe el comportamiento de este método se describe a continuación, en este se utiliza un método de intersección con un círculo:

Require: Un polígono convexo.
Ensure: Un conjunto de trixeles.

```
1: if Todos los vértices están dentro de los límites del polígono then
```

```
2:    return FULL
3: else
4:    if Por lo menos un vértice está dentro de los límites del polígono then
5:       return PARTIAL
6:    else
7:       if Los límites de la región intercepta a los vértices then
8:          if Al menos un lado se intercepta con el límite del área then
9:             if Existe otro límite que no toca el trixel then
10:               if El trixel está limitado then
11:                  return PARTIAL
12:               else
13:                  return OUTSIDE
14:               end if
15:            else
16:               return PARTIAL
17:            end if
18:         else
19:            if El trixel está limitado then
20:               return PARTIAL
21:            else
22:               return OUTSIDE
23:            end if
24:         end if
25:      else
26:         return OUTSIDE
27:      end if
28:   end if
29: end if
```

Como se puede apreciar en la figura 2.11, Se muestra el árbol de decisión de la intersección de los trixeles del algoritmo HTM con los límites de un triángulo.

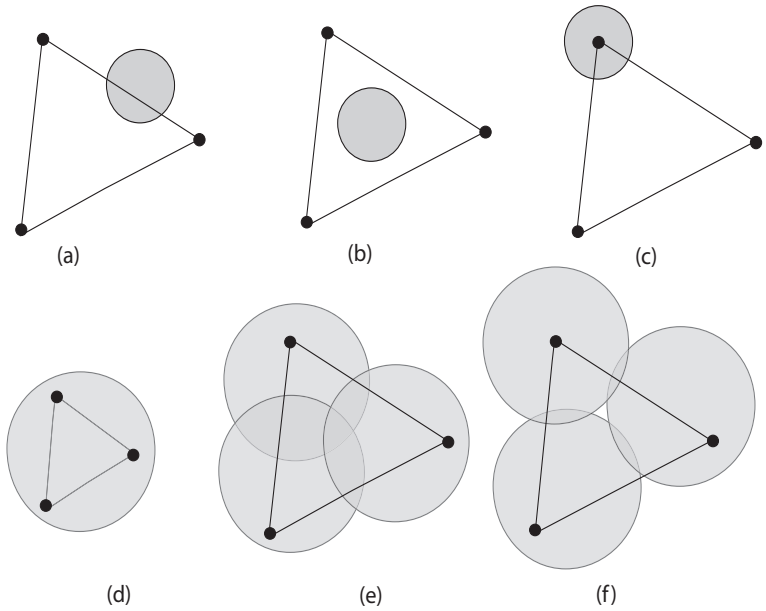

Figura 2.11: Intersección de un espacio con el límite de un triángulo.

2.5. Intersección de un trixel con una envoltura convexa

Para explicar este último método de intersección de trixeles con un espacio, es conveniente mencionar que es una envolvente convexa, así como características, ventajas y desventajas.

En matemáticas se define la envolvente convexa o envoltura convexa de un conjunto de puntos X de dimensión n, como la intersección de todos los conjuntos convexos que contienen a X. La envoltura convexa se emplea en diferentes métodos para agrupar valores, en los cuales se aísla los resultados para su análisis usando una figura. Con esto la envoltura convexa corresponde a un polígono convexo cuyos vértices son algunos de los puntos del conjunto inicial de puntos. Una forma simple de ver la envolvente, es imaginar una banda elástica estirada que encierra todos los puntos.

Existen varios algoritmos que se encargan de crear está envoltura convexa, y se dividen por su complejidad con la que trabajan. El tiempo de complejidad de cada algoritmo está en términos de un número de puntos n que se ingresan y el número de puntos en la envolvente convexa h. Cabe resaltar que en el peor de los casos h puede ser de la misma dimensión que n, A continuación se muestran algunos algoritmos para crear la envoltura convexa, ordenados por su fecha de publicación.

- Gift wrapping aka Jarvis march, con una complejidad O(nh) [1].
- Graham scan, con una complejidad O($n\,log\,n$) [1].
- Quickhull, con una complejidad O($n\,log\,n$) y en el peor de los casos O(n^2) [33].
- Divide and conquer, con una complejidad O($n\,log\,n$) [34].
- Monotone chain aka Andrew's algorithm, con una complejidad O($n\,log\,n$) [35].
- Incremental convex hull algorithm, con una complejidad O($n\,log\,n$) [36].
- The ultimate planar convex hull algorithm, con una complejidad O($n\,log\,h$) [37].
- Chan's algorithm, con una complejidad O($n\,log\,h$) [38].

La estabilidad es una característica importante en los algoritmos, ya que con ella se puede tener un parámetro en el comportamiento. La estabilidad en los algoritmos está dada, por la base de la complejidad computacional [1], estos pueden ser:

- Estable. Son los algoritmos que independiente del número de elementos y el orden de los mismos, su complejidad se mantiene constante, en este caso se pueden tener un algoritmo tiene una complejidad de O(n^2), al ordenar 100 y 1000 elementos, el tiempo se mantiene a proporción de lo que especifica el algoritmo.
- Inestable. Los algoritmos inestables son aquellos que dependiendo del tipo de modelo tienen una respuesta diferente, este caso es más común con el tipo de algoritmo de ordenamiento QuickSort, donde su complejidad en el mejor de los casos es O($n\,log\,n$), y en el peor de los casos es O(n^2).
- Ineficiente. Los algoritmos ineficientes son aquellos que no aseguran un resultado confiable y además tienen una complejidad computacional costosa, un caso es el algoritmo Bogosort que tiene una complejidad de O($n \times n!$).

Cualquier algoritmo que tenga una complejidad O($n\,log\,n$) y que sea estable es una buena alternativa. En está trabajo se utilizó el algoritmo de Graham Scan para encontrar la envoltura convexa en un conjunto finito de puntos. Este algoritmo fue nombrado por Ronald Graham, quien lo publico en el año de 1972, para más información del algoritmo se puede consultar [32].

El algoritmo de Graham Scan recibe un conjunto de puntos, los cuales son ordenados por ciertas características de encontrar al mínimo, se ordena el conjunto de puntos restante y algunos son seleccionados para formar un polígono convexo en orden contrario al sentido de las manecillas del reloj.

Require: Un conjunto de puntos S de tamaño n.
Ensure: Envolvente convexa de S.

```
1: Buscar un punto con la menor coordenada ordenada y coordenada abscisa
2: Ordenar el conjunto de puntos restantes S-1, a base al ángulo que forman con la horizontal o bien con la cotangente
3: Insertar en una pila P, los tres primeros puntos del conjunto ordenado S
4: i = 1
5: while i < n − 1 do
6:    t = P.pop
7:    if ContraReloj(P.top, t, S[i] ) then
8:       P.push(t)
9:       P.push(i)
10:      i += 1
11:   end if
12: end while
```

Como se muestra, el algoritmo Graham funciona eligiendo un elemento pivote y ordenando los elementos n-1 que restan, para el ordenamiento existen varios algoritmos que se encargan, en los cuales se encuentran:

- Burbuja con una complejidad O(n^2) [1].
- QuickSort con una complejidad variable de O($n\,log\,n$) a O(n^2) [1].
- Heapsort con una complejidad O($n\,log\,n$) [1].

Entre otros algoritmos que existen para ordenar datos. Cabe mencionar que no es de importancia que algoritmo de ordenamiento se utilice, pero si es necesario que sea rápido

y que sea estable. El algoritmo de ordenamiento que se implemento en este trabajo fue el HeapSort, el cual cumple estos requisitos.

En la figura 2.12, se muestra la envoltura convexa que se genera con el algoritmo del Graham Scan. Una vez después de explicar las envolventes convexas, podemos continuar con la generación de trixeles. Este método de generación es más complicado porque depende del signo del convexo.

En este último algoritmo además de la definición del polígonos convexos, se requiere agregar una propiedad adicional, y es que pueden ser una área simple o áreas disjuntas. En este algoritmo los convexos se componen por las regiones que se define del área convexa, las región son subespacios, y el signo está dado por los límites del plano y la dirección al plano. Un subespacio es un vector, de un punto origen dentro del subespacio y su normal a los límites del plano del subespacio, y la distancia d, del plano al origen que pertenece al vector. Si está distancia d es menor que 0, entonces el signo del subespacio es negativo, y por lo tanto el convexo es negativo, en caso contrario es positivo.

Existen tres tipos de métodos que requieren de los signos de los trixeles, como se muestra continuación:

- Convexo cero y positivo. Si el signo del convexa es cero o uno, el área que define una sección es convexa en la esfera. El árbol de decisión para una trixel es casi idéntica a la del árbol de decisión para un medio de espacio simple(algoritmo 2). La única complicación es que se necesita probar que cada medio espacio por la intersección con el trixel sea PARCIAL o bien OUTSIDE.

- Convexo negativo. El árbol de decisiones de los convexos positivos no es aplicable para convexos negativos en absoluto. Si los tres vértices está dentro de todos los semi espacios del convexo negativa, todavía tenemos que comprobar si hay un agujero en el interior del triángulo. Se hace esto mediante pruebas del vector de cada semi espacio en el convexa para determinar si se encuentra dentro del nodo. Si hay un semi espacio tal que puede marcar el triángulo como PARCIAL. Si no la hay, la prueba de si uno de las regiones de contorno del semi espacio negativo se cruza con uno de las aristas del triángulo. En caso de ser cierto, el triángulo posiblemente es PARCIAL. Si no, es FULL. Si sale una o dos esquinas están dentro de todos semi espacios, entonces se puede asumir el triángulo como PARCIAL. Este paso es el mismo para todos convexo.

  Si todas las esquinas están fuera del convexo, todavía se tiene que investigar más

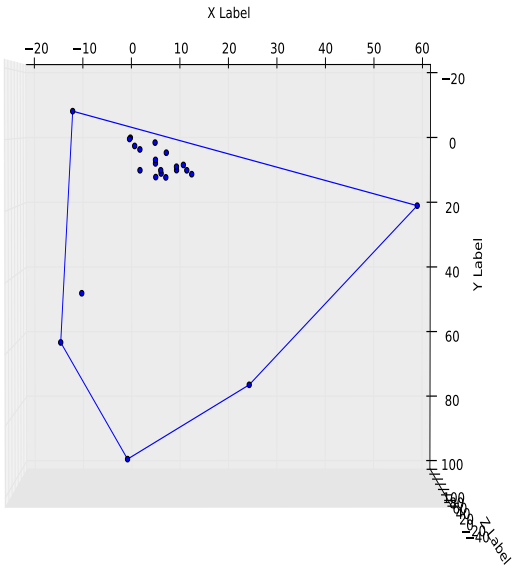

Figura 2.12: Envoltura convexa producida por el algoritmo de Graham Scan.

a fondo si el triángulo es PARCIAL o OUTSIDE. Si ninguno de los semi espacios se cruzan con las aristas del triángulo, el triángulo se puede suponer con seguridad ser OUTSIDE del convexo. Si hay intersecciones, se puede tener patrones de análisis complicados dentro del triángulo, por lo tanto se supone que es PARCIAL.

- Convexo mixto. Los convexos mixtos son aquellos con los dos semi espacios positivos y negativos. Podemos imaginar una zona convexa positivo en la esfera donde la orilla es de algo de el. El árbol de decisiones usado para este caso es una mezcla de los árboles de decisión positivos y negativos. Tiene que haber por lo menos un semi espacio positivo, y si el triángulo se encuentra totalmente fuera de cualquier semi espacio positivo en el convexa mixta, es sin duda OUTSIDE del convexo.

  Además, si sale uno o dos de los tres vértices que están dentro de la convexa y los otros vértices están fuera, el triángulo puede ser marcado como PARCIAL. Si las tres esquinas se encuentran dentro de todos los semi espacios y ninguno de los semi espacios intercepta los lados del triángulo, el triángulo de forma segura se puede asumir FULL

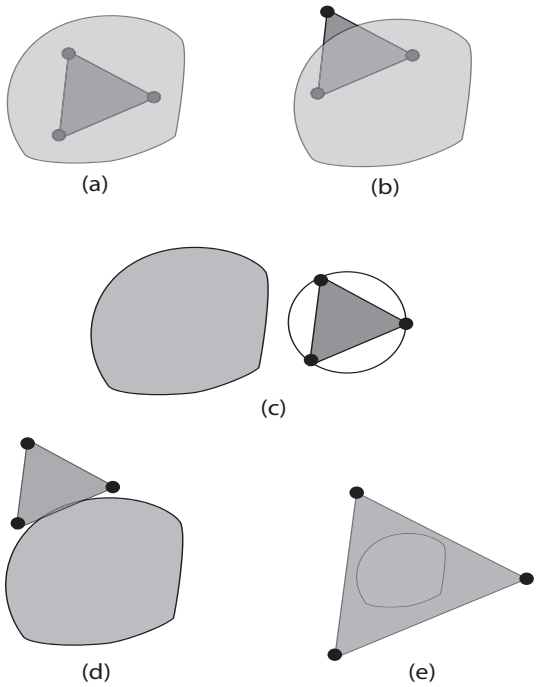

Figura 2.13: Intersección de un trixel con una envoltura convexa.

dentro de la convexa.

Si ninguno de los casos más comunes descritos se aplican, por lo general el triángulo se marca como PARCIAL. En la figura 2.13, se muestran algunos ejemplos para este método de generación de los trixeles. El algoritmo que describe el comportamiento de este método se describe a continuación:

Require: Un polígono convexo.

Ensure: Un conjunto de trixeles.

```
1: if Todos los vértices están dentro del polígono then
2:   if Cruza algún límite por dentro del trixel then
3:     return PARTIAL
4:   else
5:     if Algún otro límite se intercepta por dentro del polígono then
6:     else
7:       return OUTSIDE
8:     end if
```

```
    end if
else
    if Algunos vértices están dentro del espacio then
        return PARTIAL
    else
        if Existe otro límite que se intercepta dentro del espacio then
            return PARTIAL
        else
            return OUTSIDE
        end if
    end if
end if
```

2.6. Límite de error para la generación de trixeles

Como se vio anteriormente existen varias maneras para generar trixeles, pero en cualquiera de los casos debe de existir un margen de error para decidir si es factible crear o no el nuevo trixel. El límite de error se aplica a los trixeles que son de tipo de PARCIAL.

Con cualquiera de los tres métodos de generación, es importante conocer cuales son lo requisitos mínimos para que se pueda crear un trixel. En la figura 2.14, los trixeles blancos no se indexarán, los trixeles que están dentro de la figura del pentágono se indexarán, y finalmente aquellos que están en la parte sombreada de gris son aquello trixeles del tipo PARCIAL que se pueden indexar de manera opcional. En caso de no tomar en cuenta este tipo de error, los trixeles que se indexarán, se encuentran dentro del polígono que está encerrado en la línea negra gruesa.

2.7. Comentarios finales

En este capítulo se hablo de lo que son las coordenadas geoespaciales, y de sus aplicaciones actuales en los servicios que se ofrecen al usuario, además se introdujo formalmente el término de *trixel* que es la base para el algoritmo de malla triangular jerárquica con el cual se puede reconstruir el modelado de la esfera terrestre. Se mencionaron tres métodos que se pueden utilizar para reconstruir zonas usando trixeles, además del polígono

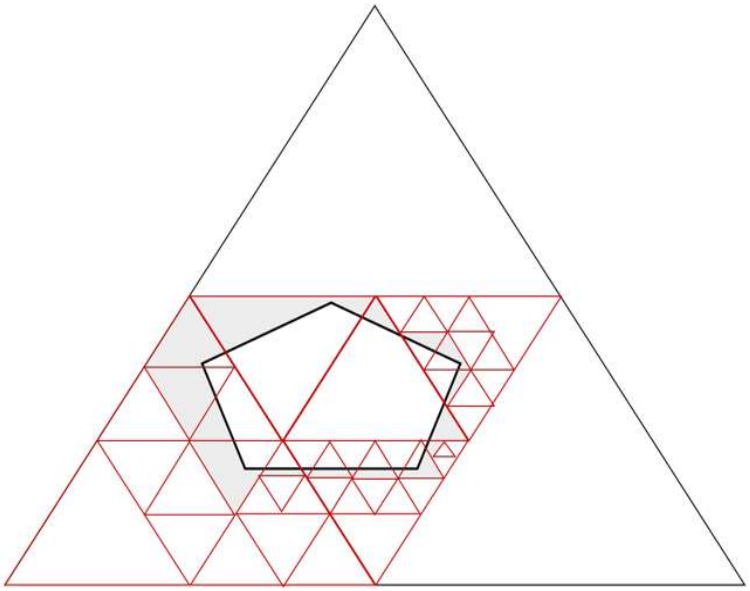

Figura 2.14: Margen de error en la generación de trixeles.

convexo que sirve para delimitar zonas, con esto en el siguiente capítulo se va a hablar de los árboles de indexación y los archivos de acceso aleatorio.

Capítulo 3

Indexadores

En el capítulo 2 se explicaron lo que son las coordenadas geoespaciales, además se mencionaron tres métodos para generar trixeles, uno de los cuales necesita de polígonos convexos, con el cual fue introducido el algoritmo Graham Scan que sirve para crear convexos de una manera sencilla. En este capítulo se va hablar de dos temas importantes, el primero de ellos es el motor de los indexadores, es decir, el árbol de indexación que se utiliza para disminuir el costo de recuperación de los datos y con esto hacer más eficientes las operaciones para indexar y buscar, el segundo tema son los archivos de acceso aleatorios que se utilizan como bases de datos para almacenar los archivos por parte del servidor.

3.1. Árboles de indexación

Los indexadores basan su funcionamiento interno a estructuras de datos de tipo árbol que crean los índices de los archivos que se almacenaran por parte del servidor, para que de una manera eficiente se pueda indexar y obtener los archivos disminuyendo el tiempo de computo. Algunos ejemplos de estructuras de datos que son más comunes en los indexadores actuales como Xapian, ElasticSearch y Solr, se muestran a continuación.

- Árbol-Kd. Un árbol-kd es un árbol k-dimensional que opta por una estructura de datos que particiona el espacio, donde se organizan los puntos en un espacio euclidiano de k dimensiones [4], [5]. Un árbol-kd emplea planos perpendiculares a los ejes del sistema de coordenadas y a todos los nodos, desde el nodo raíz hasta los nodos hoja se almacena un punto, como consecuencia cada plano debe pasar a través de uno de los puntos del

árbol, un ejemplo de esta estructura de datos de tipo árbol se muestra en la figura 3.1, por último un árbol-kd balanceado es aquel donde cada nodo hoja se encuentra a la misma distancia de la raíz.

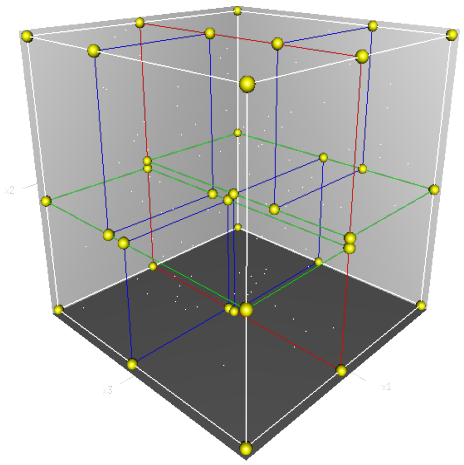

Figura 3.1: Estructura del árbol-Kd [47]

- Árbol-B. Los árboles-B son estructuras de datos que están diseñadas para que los nodos internos tengan un número variable de nodos hijo dentro de un rango predefinido [1], cuando se inserta o se elimina un dato de la estructura, la cantidad de nodos hijos es variable dentro de un nodo, la ventaja de los árboles-B es que no tienen la necesidad de re balancearse frecuentemente, dado que al indexar cada nuevo dato, automáticamente el modelo buscar el lugar adecuado para agregarlo, partiendo y uniendo nodos para que se agrega en su sitio adecuado, dependiendo del orden de los datos que se almacenen. Por otro lado, se puede desperdiciar memoria, dado que los nodos no permanecen totalmente ocupados, lo cual se convierte en una desventaja, por esto existen algunas variantes de los árboles-B que se utilizan para optimizar el uso de la memoria, algunos ejemplos se muestran a continuación:

  - Árboles-B+. Un árbol-B+ es un tipo de estructura de datos de árbol que representa una colección ordena de elementos, donde se permite de una manera eficiente las operaciones de inserción y borrado de elementos. En un árbol-B+ todos los nodos hojas se encuentran en el mismo nivel, además se encuentran unidos entre si como una lista enlazada que permite la recuperación en rangos mediante búsquedas secuenciales, un ejemplo se muestra en la figura 3.2.

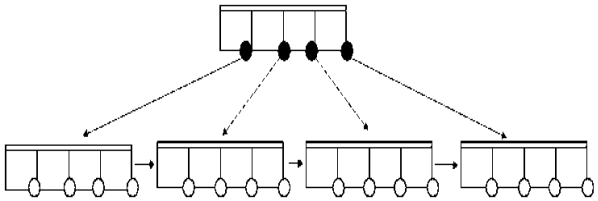

Figura 3.2: Estructura del árbol-B+

– Árboles-B*. Un árbol-B* es una estructura de datos de árbol que requiere que los nodos no raíz estén por lo menos a 2/3 de ocupación. Para mantener esto, en lugar de generar inmediatamente un nodo cuando se llena, se comparten sus claves con el nodo adyacente, cuando ambos están llenos, entonces los dos nodos se transforman en tres, también se requiere que la clave más a la izquierda no sea usada nunca, un ejemplo se muestra en la figura 3.3.

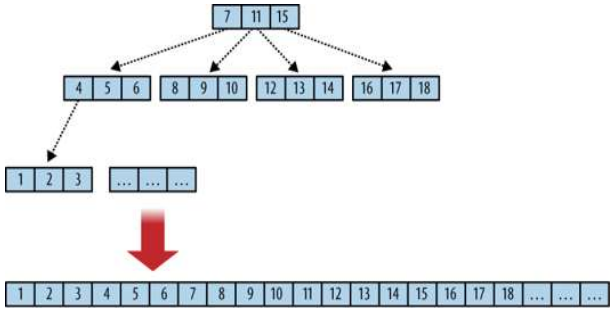

Figura 3.3: Estructura del árbol-B* [48]

- Árbol-R. Los árboles-R son estructuras de datos de tipo árbol similar a los árboles-B, con la diferencia de que se utilizan para métodos de acceso espacial, es decir, para indexar información multidimensional [25], [26]. La estructura de datos de este árbol divide el espacio de forma jerárquico en conjuntos, posiblemente superpuestos, donde cada nodo de un árbol-R tiene un número de entradas predefinidas. Cada entrada de un nodo interno almacena dos datos: el primero es una forma de identificar a un nodo hijo y el segundo es el conjunto límite de todas las entradas de ese nodo, finalmente en este tipo de árbol al igual que los anteriores los algoritmos de indexación, borrado y búsqueda utilizan conjuntos límite para decidir en que nodo buscar, de este modo, la mayoría de los nodos del árbol nunca son examinados durante una búsqueda, un

ejemplo de esta estructura de datos de tipo árbol se muestra en la figura 3.4.

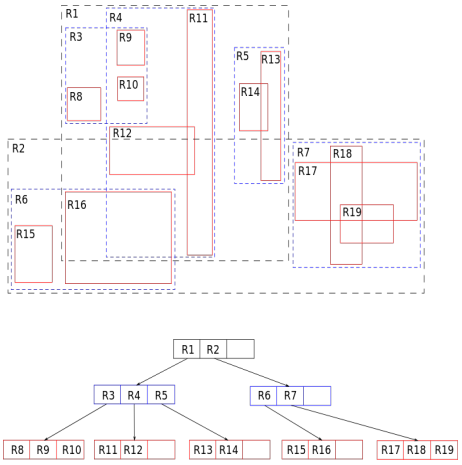

Figura 3.4: Estructura del árbol-R [46]

3.2. Archivos de acceso aleatorio

Los archivos son una colección de información, localizada o almacenada como una unidad en alguna parte de la computadora. Por si mismo, un archivo consiste en nada más que una serie de bytes relacionados y ubicados en discos [3]. Los archivos de acceso aleatorio son más versátiles, permiten acceder a cualquier parte del archivo en cualquier momento, como arreglos en memoria, y las operaciones de lectura y/o escritura se pueden hacer en cualquier punto del archivo [2]. En general se permite establecer ciertas normas para la creación para los archivos de acceso aleatorio, por ejemplo:

- Abrir el archivo en modo que se permite leer y escribir.
- Abrirlo en modo binario.
- Usar funciones que permitan leer, escribir y buscar datos sobre el archivo de acceso aleatorio situando el puntero en el lugar apropiado.

Facebook en la actualidad almacena cerca de 260 billones de imágenes que han subido los usuarios, en conjunto el tamaño todas estas imágenes es aproximadamente a 20 peta bytes de datos, por problemas que tuvo con el sistema de archivos por la cantidad de nodos que utilizaba, implementó este sistema de almacenamiento optimizado como bases

datos, el cual fue llamado como **Haystack** [20]. Haystack proporciona una solución de rendimiento menos costoso y más alto, la clave de este diseño estuvo en reducir cuidadosamente todas las operaciones de búsqueda de metadatos en la memoria principal, y se conservan operaciones específicas de disco para la lectura de los datos y por lo tanto aumenta el rendimiento general. Buscar documentos en un directorio es extremadamente ineficiente, por lo tanto Haystack reduce los accesos a los discos usando solamente tres operaciones para obtener los archivos, las operaciones utilizadas sirven para:

- Las lecturas de los metadatos dentro de la memoria.
- Cargar los nodos dentro de memoria.
- Leer archivos donde se almacena el dato que se busca.

3.3. Indexadores populares de uso general

A continuación se mencionarán algunas características de los tres principales indexadores de texto que se utilizan en la actualidad, entre estos se encuentra Xapian, Apache Solr y ElasticSearch, pero antes es conveniente mencionar algunas características que tienen en común, por ejemplo, los tres se construyen por un conjunto de API's, donde una API es una interfaz de programación de aplicaciones, y es básicamente un conjunto de funciones o procedimientos que se ofrecen para que sean utilizados por otro software como una capa de abstracción. Además, estos indexadores tienen características RESTful, que ésta es la transferencia de estado que representa al estilo del world wide web (www), aplicado al diseño de los componentes en un sistema que se comunican a través de Protocolo de transferencia de hipertexto HTTP con las consultas GET, POST, PUT, DELETE, etc. que los navegadores web utilizan para recuperar las páginas web y enviar datos a servidores remotos.

3.3.1. Xapian

Xapian es una biblioteca de código abierto de recuperación probabilista de información, publicada bajo la Licencia Pública General (GPL/GNU), y su uso se basa principalmente como motor de búsqueda de texto [27].

Xapian está diseñado para ser un conjunto de herramientas adaptables que permiten a los desarrolladores agregar fácilmente indexación y búsqueda avanzada en sus propias

aplicaciones. Las búsquedas geoespaciales que usa Xapian, almacena un documento asociando las coordenadas de la latitud y longitud. Xapian puede utilizar tres tipos de búsquedas geoespaciales, las cuales sirven para:

- Regresa una lista ordenada de documentos por medio de la distancia de una ubicación, y se crea por medio de un conjunto con las consultas de Xapian.
- Regresa una lista de documentos con una distancia de la localidad, usado un conjunto de consultas, y diferentes maneras de ordenamiento.
- Regresa un conjunto de documentos en orden diferente a un orden base sobre una distancia de una localidad y su importancia.

Las ubicaciones de los documentos indexados se almacenan en las ranuras de valor, lo que permite múltiples ubicaciones independientes sean utilizadas para un documento. Por otra parte también es posible almacenar múltiples coordenadas en una única ranura, en cuyo caso, la coordenada más cercana será utilizada para los cálculos de distancia.

Para indexar un conjunto de documentos con la ubicación, se necesita almacenar una coordenada codificando latitud y longitud en una ranura, además de los valores obligatorios y opcionales del archivo a indexar. A continuación se va a mostrar algunos ejemplos con los cuales se puede utilizar Xapian, para indexar y hacer búsquedas utilizando las características geoespaciales y la ayuda del lenguaje de programación C++.

- Para almacenar un documento en una ranura

  Xapian :: doc Documento;

  doc.add_value (0, Xapian :: LatLongCoord (51,53, 0,08).serialise ());

- Para almacenar múltiples coordenadas en una sola ranura, se utiliza una clase llamada LatLongCoords:

  Xapian :: doc Documento;

  Xapian :: LatLongCoords coords;

  coords.append (Xapian :: LatLongCoord (51.53, 0.08));

  coords.append (Xapian :: LatLongCoord (51.51, 0.07));

  coords.append (Xapian :: LatLongCoord (51.52, 0.09));

  doc.add_value (0, coords.serialise ());

Xapian en las búsquedas, primero ordenar los resultados por distancia o alguna métrica, y la información de ubicación se encuentra asociada con cada documento que se almacena, esto permite que se inicie rápidamente en el tiempo de búsqueda, de modo que una distancia que se consulta pueda ser calculada fácilmente. Sin embargo, este método requiere que la distancia se encuentre marcada, y en contraparte puede ser costoso en tiempo de computo.

Xapian en su terminologías de funcionamiento y estructura son semejantes con los demás indexadores de texto, algunos ejemplos se muestran a continuación:

- Bases de datos (Databases). Las bases de datos en Xapian y en otros indexadores también pueden ser llamados índices. La razón es que en Xapian una base de datos consiste en poco más de documentos indexados, esto refleja el propósito de Xapian como un sistema de recuperación de información, en lugar de un sistema de almacenamiento de información.

- Consultas (Queries). En Xapian, las consultas se componen de un árbol booleano estructurado, sobre la cual se impone ponderaciones probabilistas, cuando se realiza la búsqueda, los documentos devueltos son filtrados de acuerdo con la estructura booleana, ponderados y ordenados de acuerdo con el modelo probabilista de la información para su recuperación.

Xapian trabaja con los dos tipos de consultas que son la booleana y la probabilista además las puede mezclar para hacer más eficiente y confiables los resultados, esta mezcla sólo se puede hacer por medio de dos ejemplos, los cuales son:

- Realizar primero una búsqueda booleana para crear un subconjunto de toda la colección de documentos y luego la búsqueda probabilista en este subgrupo.

- Realizar primero una búsqueda probabilista y luego filtrar los documentos resultantes con consulta booleana.

De esta mezcla de consultas existe una diferencia notable, ya que en el primer ejemplo, la estadística de la consulta probabilística serán determinados por el subconjunto documento que se obtiene mediante la ejecución de la consulta booleana, y en el segundo, las estadísticas de recolección para la consulta probabilística están determinadas por toda la colección de documentos. Con esto se puede notar que estas diferencias pueden afectar el resultado final de lo que se está solicitado.

3.3.2. Apache Solr

Apache Solr es un motor de búsquedas de código abierto que proporciona potentes búsqueda y navegación usando diferentes métodos para recopilar datos desde diversas perspectivas, escrito en su mayoría usando el lenguaje de programación Java [31]. Solr tiene una funcionalidad que puede ser difícil de implementar sobre una base de datos relacional. Además, es capaz de manejar complejos criterios de búsqueda, corrige ortografía en las consultas, permite realzar resultados entre otras características. La indexación en Solr significa agregar contenido a un índice y agregar las características para poder modificar, editar, eliminar tanto a los índices y al contenido que es indexado. Solr agrega localización con datos de texto, esto a menudo se llama la búsqueda espacial o búsqueda geoespacial. La mayoría de estas aplicaciones tienen que hacer varias cosas por ejemplo:

- Representar en el índice los datos espaciales.
- Filtrar por algún concepto espacial como un círculo, cuadro o alguna otra figura que sirve como limitador.
- Ordenar por distancia.

Un índice Solr puede aceptar datos de muchas fuentes diferentes, incluyendo archivos XML, y los datos que se contienen en archivos separados por comas (CSV), además de archivos en formatos comunes como Microsoft Word o PDF. Solr tiene una capa de interacción con el usuario, donde se puede establecer comunicación con ayuda de:

- Apache Tika [39] para indexar archivos con formatos como .doc, .pdf, .rtf, etc.
- Peticiones HTTP al servidor Solr usando archivos con extensión .xml.
- Programas hechos en el lenguaje de programación Java para indexar los datos a través de un cliente en Solr.

Apache Tika es un conjunto de herramientas desarrolladas por la Fundación Apache para detectar, extraer metadatos y el contenido de texto estructurado de muchos tipos de documentos, algunos ejemplos son: gzip , .mid , .pdf , tar , zip , etc. Algunas otras características adicionales que componen a Solr son:

- Servidor con interfaz tipo RESTful.

- Utiliza varias memorias cache para agilizar las búsquedas.
- Interfaz web de administración.
- Navegación de resultados por facetas.
- Puede expandirse para utilizar varios servidores.
- Módulos de importación de datos desde bases de datos, e-mail y archivos de texto enriquecido (PDF, Word, Excel).
- Análisis de texto, es decir, procesa cadenas de texto que puede fragmentar para búsqueda de patrones semejantes.

La característica que define a las bases de datos relacionales es su modelo de datos basado en tablas y la posibilidad de relacionarlas entre si mediante llaves. Solr es no relacional entonces el paso de un esquema relacional a un esquema no relacional requiere un cambio en la estructura de la información, en la mayoría de los casos, este proceso es directo y consiste en un proceso de normalización de los datos. Sin embargo, en ocasiones puede resultar difícil o imposible, especialmente donde existen relaciones muchos a muchos, con múltiples campos susceptibles a búsquedas, de este modo, para una consulta que involucre varías tablas, es necesario el uso de la operación JOIN para unirlas, pero como esta operación puede ser muy lenta si las tablas son muy grandes, Solr utiliza un modelo de datos orientado a documentos que pueden pensarse como una base de datos de una sola tabla, por lo que una operación análoga a JOIN es inexistente, lo cual le brinda a Solr una característica de velocidad.

De las últimas características acerca de Solr, es que es simplemente un conjunto de campos, como una tupla en una tabla de una base de datos, con la diferencia de que cada columna puede ser multi valuada. Otra diferencia importante entre Solr y una base de datos es que en Solr no existe la operación equivalente a la operación UPDATE. Si cierta parte de un documento necesita ser actualizada, el documento debe ser eliminado y agregar en su lugar el documento actualizado.

3.3.3. ElasticSearch

ElasticSearch es un servidor de búsqueda basado en Lucene que es una API de código abierto para recuperación de información que es útil para cualquier aplicación que requiera indexado y búsqueda a texto completo. Provee un motor de búsqueda de texto completo, distribuido y con capacidad de multi tenencia con una interfaz web RESTful y con documentos JSON. Elasticsearch está desarrollado en Java y está publicado como código abierto bajo las condiciones de la licencia Apache [29].

ElasticSearch puede ser usado para buscar todo tipo de documentos, provee búsqueda escalable, tiene tiempo de búsqueda cercano a tiempo real, es distribuido, lo que significa que los índices pueden ser divididos en fragmentos y cada fragmento puede tener cero o más réplicas, así cada nodo alberga uno o más fragmentos, y actúa como un coordinador para delegar operaciones a los fragmentos correctos, con esto el rebalanceo y ruteo se realizan automáticamente. Estás y algunas otras características son las que tiene ElasticSearch como RESTful, por ejemplo, cuenta con:

- Motor de búsqueda distribuido y de alta disponibilidad.
- Cada índice es totalmente fragmentado con un número configurable de fragmentos.
- Cada fragmento puede tener uno o más réplicas.
- Operaciones de leer y buscar realizadas a cada uno de los fragmentos de réplica.
- Soporte por más de un tipo por índice.
- Nivel del índice de configuración (número de fragmentos, almacenamiento índice).
- Varios conjuntos de APIs.
- HTTP API REST.
- Todas las APIs pueden hacer desvío de operación nodo automático.
- Orientado a documento.
- No hay necesidad de definición de esquema inicial.
- El esquema de configuración se puede definir según el tipo de personalización del proceso de indexación.

- Cada fragmento es un índice de Lucene completamente funcional.
- Todo el poder de Lucene se expone fácilmente a través de sencillos plugins de configuración.

3.4. Comentarios finales

En este capítulo se explicaron las estructuras de datos más comunes que se implementan para los indexadores, mostrando algunas características que las distinguen a cada una, con las cuales se puede incrementar la eficiencia y disminuir el consumo de memoria, finalmente se menciona brevemente de algunos indexadores de texto actuales, resaltando características de funcionamiento en general, procesamiento de datos, consumo de memoria, interfaz de comunicación y además de su funcionamiento con relación al usó de las coordenadas geoespaciales. En el capítulo 4 se habla del indexador propuesto que se implementó en este trabajo, se explican las características con la que se diseñó para crear los índices y con esto indexar documentos por puntos o zonas, además del diseño de las bases de datos.

Capítulo 4

Desarrollo del Indexador

En el capítulo 3 se explicaron algunas características que pertenecen a los indexadores actuales, una de las más importantes es el árbol de indexación, porque la velocidad del indexador depende de la cantidad de operaciones y la relación entre el indexador y los archivos almacenados en la base de datos. Por otro lado se habla de una manera eficiente de almacenar y recuperar archivos a partir de otro archivo, con el fin de reducir el número de operaciones de I/O sobre el disco duro. En este capítulo se va a explicar las características del indexador y del archivo con acceso aleatorio propuestos, al igual se muestran los alcances y limitaciones que componen el sistema. Se explica la manera que se utilizó para realizar el nombramiento de los trixeles, indexación de los trixeles, y el método de recuperación de archivos desde las bases de datos.

4.1. Descripción general acerca del indexador

En esta sección se van a explicar las características de cada uno de los módulos que componen el indexador propuesto, como se muestra en la figura 4.1 la arquitectura completa resaltando los módulos principales para:

- Sección a, generar y administrar los trixeles.
- Sección b, crear el índice de los trixeles y la comunicación con el archivo de acceso aleatorio.

También se muestran las tres funciones básicas que se aceptan por el indexador, para poder interactuar con el usuario, las cuales son:

- Indexar archivos.
- Buscar archivos.
- Eliminar archivos.

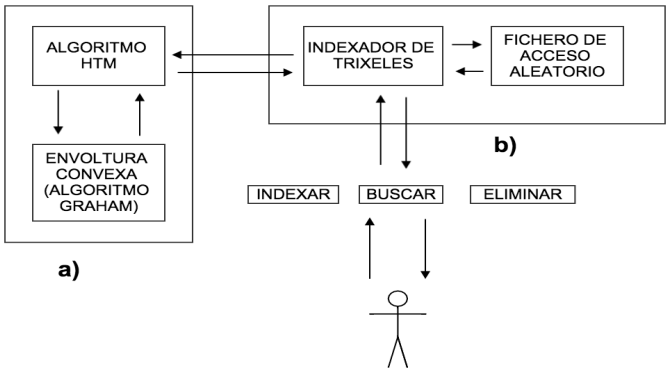

Figura 4.1: Arquitectura general del indexador propuesto.

La primera característica que se tomó en cuenta en el diseño del indexador, fue una estructura de datos que permite crear relaciones entre los diferentes tipos de los nodos y se expanda de una manera balanceada entre ellos. El árbol de indexación implementado es un árbol-B, un ejemplo de esta estructura de datos se muestra en la figura 4.2. Se modifico la estructura del árbol-B para limitarlo en profundidad, en la figura 4.3, se muestra otro ejemplo del árbol de indexación completo que integra el indexador propuesto, también se muestran los dos principales cuadrantes (N y S), y el límite que corresponde al diseño, en este caso es de 25 niveles de indexación. La ventaja de este tipo de estructuras de datos, es que es un árbol balanceado por la manera de generación de los nodos, ya que es una estructura definida de cuatro nodos hijos, pero la desventaja es donde soló se utiliza un nodo hijo, ya que se desperdician 3/4 partes del nodo, esta fue la razón por la cual se implementó un árbol-B y no un QuadTree. Un árbol-B permite disminuir el consumo de memoria que se reserva para el árbol de indexación, ya que se reutilizan los nodos hasta que sea necesario expandir el árbol, en la figura 4.4 se muestra en el inciso a) un quadtree, y los cuadrados en negro hacen referencia a los nodos que tienen algo indexado, se puede notar que el espacio que se desperdicia es bastante, al ajustar un poco el árbol del indexador, se puede observar en el inciso b) que se pueden ahorrar reservar memoria para tres nodos, pero es conveniente mencionar que aunque se ajusten a un nodo, se implementaron algunas

funciones que permiten separarlos para semejarse con las características del árbol-B como se muestra en el inciso c), al llegar nuevo documento a indexar perteneciente a un nodo especificó, se reserva memoria, y se extraen los hijos que no pertenecen a ese nodo, para que este nuevo documento se pueda indexar, como se muestra en la figura el cuadrado en color gris.

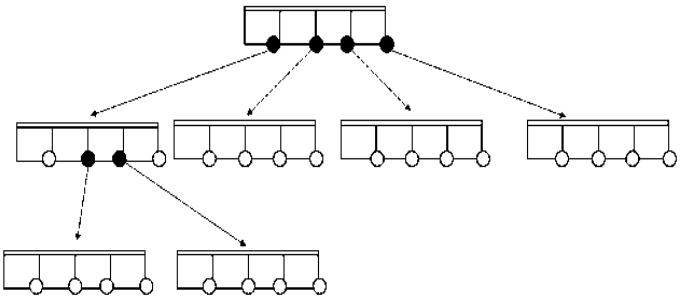

Figura 4.2: Árbol de indexación tipo B (B-Tree).

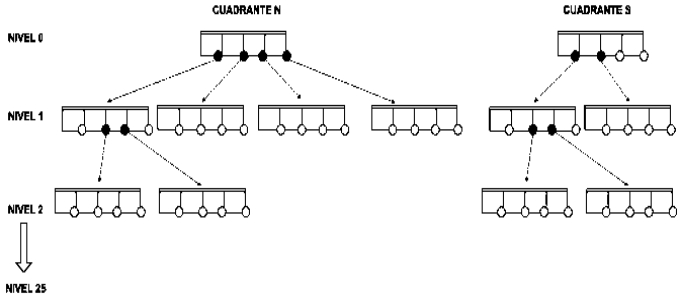

Figura 4.3: Esquema del árbol de indexación, del indexador propuesto.

En el árbol de indexación cada nodo representa un trixel que consta con los parámetros básicos para crear búsquedas geoespaciales, y mantener una relación del almacenamiento de documentos, entre otras características. Cada trixel para encontrar los nodos hijos a cualquier nivel, utiliza como acceso directo el trixel padre que se define al inicio de la generación de la ruta por medio del identificador del trixel. Cada trixel se compone por los siguientes elementos:

- ID. Es el identificador del trixel usando un número entero de 64 bits, codificado en binario.
- Depth. Es la profundidad con la que cuenta cada trixel, y se requieren de 8 bits.

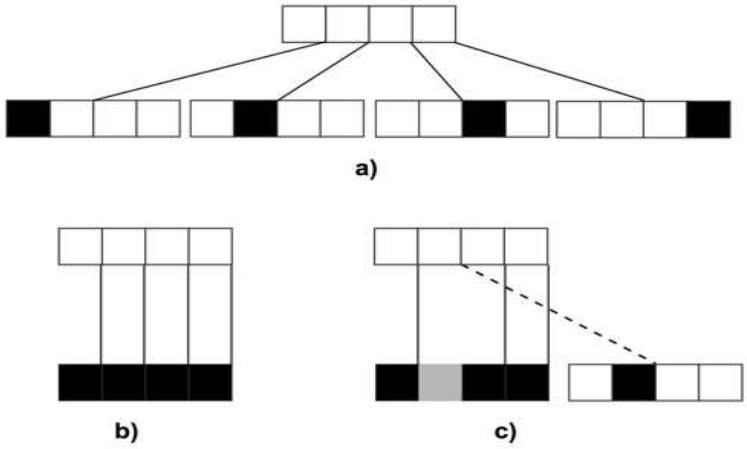

Figura 4.4: Diferencias entre un árbol-B y un Quadtree.

- Coords. Son las coordenadas que conforman el trixel, es un arreglo de enteros dobles para las coordenadas en *x, y, z*.

- Documents. Es la referencia de a los documentos que son almacenados por cada trixel indexado.

El indexador toma las coordenadas y el identificador de cada trixel que son generadas por el algoritmo HTM. Para que el algoritmo HTM sea utilizado de manera global y se pueda expandir en trabajos futuros conjunto con otros indexadores, se implementó una relación del octaedro base que utiliza este algoritmo, con los ejes *x, y, z*, del sistema de coordenadas latitud y longitud el cual se muestra en la figura 4.5. Aunque existe algunos algoritmos que tienen implementada esta relación, se creó una aproximación utilizando una conversión, donde se relaciona del octaedro el eje *y*, como el meridiano de 0^o, y el eje *x*, como el ecuador (paralelo de 0^o), con esto se puede despreciar el eje *z*, así se obtuvo un sistema de coordenadas en (x, y) solo se cuidó la referencia de los meridianos de 0^o y 180^o.

Por último, este indexador está diseñado para realizar operaciones a nivel de bits, utilizando corrimientos a la izquierda o derecha, para obtener la ruta del trixel y el cuadrante al que pertenecen, y se requiere de la profundidad para conocer de la cantidad de bits utilizables de cada identificador, por cada dos corrimientos se obtiene la rama donde se descienden los nodos hijos, en casó que existan sucesores.

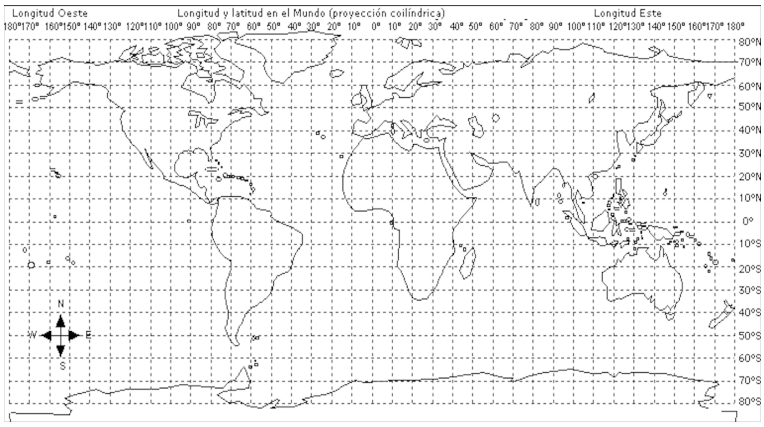

Figura 4.5: Mapa de coordenadas latitud-longitud [45].

4.1.1. Nombramiento de los trixeles

El nombramiento de los trixeles, es de suma importancia en el modelo del indexador, ya que en el peor de los casos el indexador estará trabajando con millones de trixeles indexados, y el consumo de memoria únicamente para almacenar el identificador es costoso.

Por parte del árbol de indexación, es importante la profundidad de los trixeles para indexar a cada uno en su lugar adecuado, cabe mencionar que la profundidad puede estar implícita o no, dependiendo si el nombramiento de los trixeles está dado por cadenas de texto o utilizando simplemente bits, por ejemplo, en cadenas de texto, la profundidad está dada por la longitud de la cadena, a diferencia de la codificación en binario o hexadecimal que se requiere de un campo adicional para conocer la profundidad. Suponemos el siguiente id *01000000000000000000000000*, si analizamos detenidamente la información contenida, es que pertenece a algún cuadrante por el *01*, pero no menciona exactamente a que trixel corresponde, con esto, el campo adicional menciona acerca de la profundidad que se le atribuye. La desventaja de utilizar más variables, está directamente relacionado a utilizar más memoria, pero la ventaja es que no es tan costoso, y se requieren de al menos 8 bits para guardar está profundidad y no excede a un nivel 25 que es la altura máxima del árbol de indexación. Es importante la distribución de los niveles del indexador, el cuadrante ocupa la posición 0, e inician los trixeles desde el nivel 1 y terminan en el nivel 25.

En el peor de los casos, se indexarán trixeles al último nivel disponible, en la tabla 4.1, se muestra la cantidad de bits que se requieren para indexar al último nivel tomando en consideración si hablamos de cadenas o a nivel de bits. Ahora se va a mostrar el consumo de

memoria al indexar trixeles sin documentos relacionados. Si indexamos 10,000,000 trixeles, con cadenas de texto requerimos de 248 MB, con codificación en Hexadecimal se requiere de 124 MB y con codificación en Binario se requieren de 62 MB.

Codificación	Cantidad de bits
Cadenas	208
Hexadecimal	104
Binario	52

Tabla 4.1: Cantidad de bits requeridos por cada uno de los tipos de codificación.

Para optimizar el uso de la memoria, es mejor crear un tipo de dato que se ajuste completamente a los bits que se necesitan, y como eso no es posible, se utiliza variables que se acerca lo más a las necesidades que tenemos. El lenguaje de programación utilizado fue C++, y para la codificación en binario, existen variables enteros de 64 bits, con la cual se puede representar el ID del trixel, y enteros de 8 bits para representar la profundidad del trixel, así que 64 + 8 = 72 bit's = 9 bytes que se necesitan para nombrar un trixel en codificación en binario. Tomando el ejemplo nuevamente para indexar 10,000,000 trixeles, ahora se requieren de 86 MB, lo cual es mejor y es aproximadamente a la tercera parte de la codificación usando concatenación de cadenas. De esta manera se puede observar que no consumiremos mucha memoria por nombrar trixeles.

Finalmente, en la tabla 4.2, se muestran los valores que se utilizaran para de la representación de los cuadrantes en binario y en la tabla 4.3, se muestra la representación en binario de cada rama que se encuentra en el identificador de cada trixel. Con esto llegamos a la tabla 4.4, donde se muestran los dos tipos de codificación de la ruta de los trixeles que corresponden al cuadrante N y el cuadrante S.

Cuadrante	Representación Bits
N	11
S	10

Tabla 4.2: Codificación de los cuadrantes en binario.

Número	Representación Bits
0	00
1	01
2	10
3	11

Tabla 4.3: Codificación de las ramas en binario.

ID Trixel	Representación Bits	Ruta en decimal	Profundidad
N12	11011000000000000000000000000000000000000000000000000000	12	25
S132	10011110000000000000000000000000000000000000000000000000	132	25

Tabla 4.4: Codificación de los identificadores de trixeles en binario.

4.1.2. Indexar un documento

El indexador acepta como parámetros el trixel a indexar, con este se puede calcular la rama a la que pertenece, utilizando su identificador, en este proceso se van creando los nodos necesarios para construir el árbol de indexación, obteniendo dos maneras para indexar, las cuales se basan en un:

- Nuevo trixel. En este caso se crean los punteros del nodo padre a los nodos hijos, y se reserva la memoria necesaria, para indexar el nuevo trixel en el árbol de indexación.
- Trixel existente. En este casó sigue la ruta antes creada y se almacena en el nodo designado por el identificador..

Durante este proceso, el indexador se comunica con el fichero de acceso aleatorio para obtener los datos donde se almacena el archivo.

4.1.3. Buscar un documento

Para las búsquedas en el indexador propuesto se contemplaron los siguientes casos:

- Obtener los trixeles de un padre. Obtener todos los trixeles de un nodo es una primera aproximación de búsqueda, recorre todos los nodos utilizando un método recursivo y

de esta manera obtener los documentos que se encuentran almacenados. En la figura 4.6, se muestran los trixeles (triángulos verdes) que se pueden obtener a partir de un trixel padre, suponiendo que el trixel padre es el triángulo que se forma con las líneas de color negro.

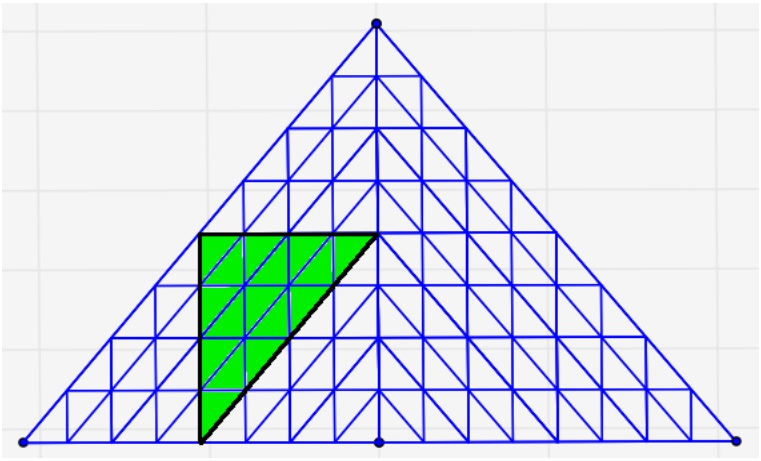

Figura 4.6: Trixeles que se pueden generar desde un trixel padre.

- Los trixeles más cercanos. Para buscar los trixeles más cercanos se toma en cuenta las coordenadas del trixel para sobreponer una figura geométrica como un círculo, triángulo, cuadrado, rectángulo o cualquier otro polinomio convexo que cubra una extensión de área, y por medio del algoritmo HTM, se realiza una reconstrucción de todos los trixeles que se almacenan, y además están dentro de esa área. En la figura 4.7, se muestra un ejemplo sobre el árbol de indexación, como se relacionan los trixeles más cercas de un trixel específico. En la figura 4.8 se muestran los trixeles más cerca(triángulos grises) de un trixel (triángulo verde), además en este ejemplo se puede notar que la figura sobrepuesta es un rectángulo.

  En las búsquedas de los trixeles más cercanos, el algoritmo HTM es muy útil mientras se realiza la reconstrucción. Una de las razones por las que se implemento el algoritmo de Graham Scan, es que en su envoltura convexa, crea conjuntos de trixeles, y está es una buena estrategia para acceder a los trixeles más cercanos, porque se obtienen todos los trixeles que se encuentran dentro una envoltura.

  En la implementación de este tipo de búsquedas, al obtener conjuntos, es un evento probabilístico, dado que por medio del identificador del trixel y su profundidad es

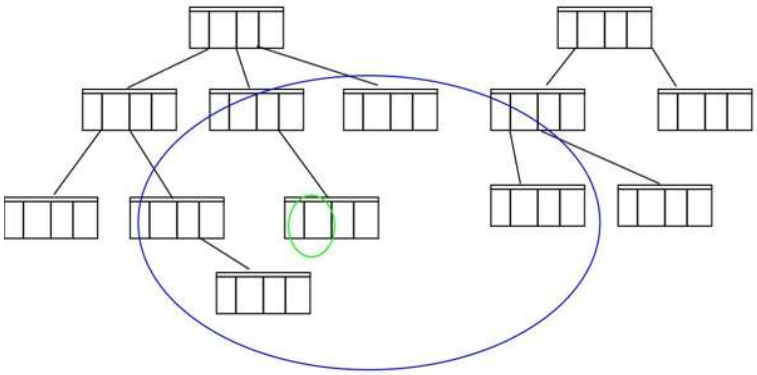

Figura 4.7: Búsquedas de los trixeles más cerca visto desde el árbol de indexación.

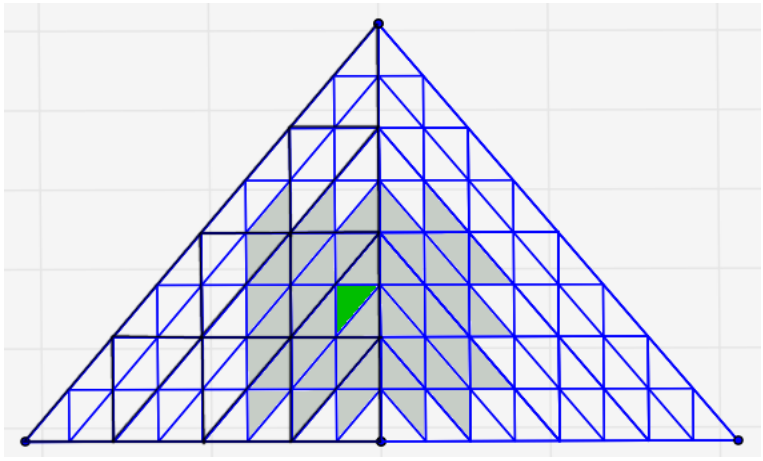

Figura 4.8: Búsquedas de los trixeles más cerca visto desde el algoritmo HTM.

posible hacer una aproximación de los trixeles hijos que son generados y cumplen con los que son requeridos, por medio de las características del QuadTree. Esto se relaciona con Xapian es un indexador probabilístico, y sus funciones dependen del tipo de búsqueda que se requieren. El algoritmo programado para obtener los trixeles más cercanos funciona de la siguiente manera:

- Primero utilizando la expansión del QuadTree, se calcula el número de trixeles que se pueden obtener utilizando la profundidad del trixel que se paso por referencia, y se asciende hasta llegar a una profundidad donde la expansión del QuadTree asegura que existe la cantidad solicitada.
- Segundo, descendemos un nivel adicional para llegar al padre del trixel y realizar

la búsqueda. Descender un nivel adicional ayuda a expandir la recuperación a hacia los nodos hermanos y recuperar más trixeles si es necesario.

– Sí la cantidad de trixeles recuperados no es suficiente, se regresa al segundo paso. Sí llegamos al nodo cuadrante y aún no se han recuperado todos los trixeles, significa que el indexador no contiene todos los trixeles solicitados, por lo cual sólo se imprime un mensaje.

Como comentario final de este algoritmo, para obtener los trixeles se tomo como prioridad la rama que fue especificada desde la base del id de un trixel.

- Búsquedas por rangos de trixeles. Este tipo de búsquedas realiza una comparación de los identificadores hasta encontrar un valor semejante en la misma posición de ambos identificadores de los trixeles que corresponde a su trixel padre. El trixel padre que encierra el conjunto de trixeles se encuentran en un rango, en la figura 4.9, se muestra como se asemeja una búsqueda por rangos en un árbol de indexación, y en la figura 4.10 se muestran pares de trixeles a los cuales se les pueden aplicar búsquedas por rango, por ejemplo, en el par de trixeles verde su rango está dentro del cuadrante y en el caso de los trixeles amarillos, las búsquedas por rangos se desplaza hasta los dos cuadrantes que son una representación de la búsqueda por rango que se muestra en la figura 4.9.

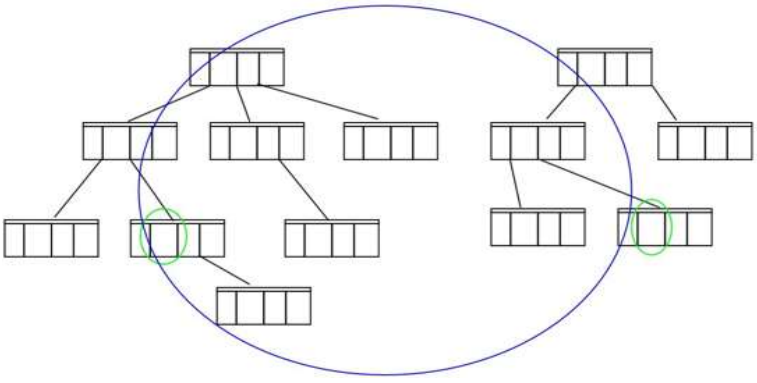

Figura 4.9: Búsquedas de rangos de trixeles visto desde el árbol de indexación.

Para finalizar con las búsquedas, es conveniente mencionar que el algoritmo que se implementó se basa en utilizar los rangos que se denotan por las rutas del identificador

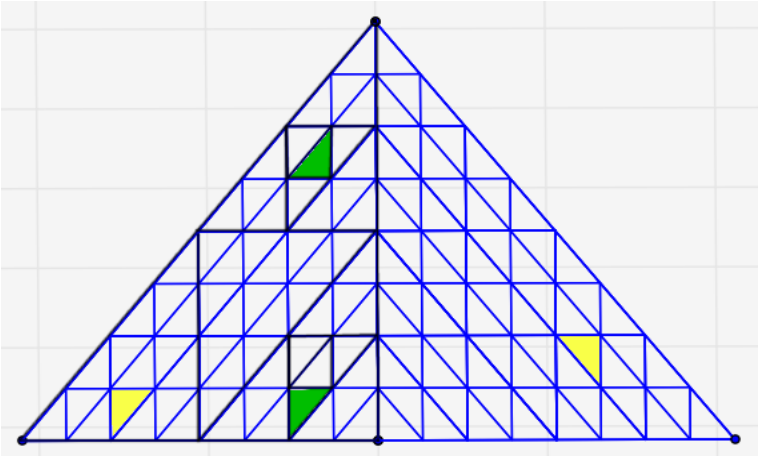

Figura 4.10: Búsquedas rangos de trixeles visto desde el algoritmo HTM.

del trixel, es decir, si el identificador de un trixel A = 0123 y el identificador de un trixel B = 2222. Primero se obtiene una llamada recursiva por el trixel padre completo que le corresponde el identificador 1000, después se obtienen de manera recursiva todos los trixeles que se encuentren a la derecha del trixel 0123 y finalmente se obtienen de manera recursiva todos los trixeles a la izquierda del trixel 2222. Este proceso se muestra en la figura 4.11. En está figura se puede ver que el trixel completo se asemeja al triángulo amarillo, al cual le corresponde el inciso b). En la rama del inciso a), suponemos que se asemeja al identificador del trixel A, en el cual la ruta se resalta por los círculos de color verde, y lo que se busca es obtener son los trixeles que se representan de color amarillo. De esta misma manera ocurre para el inciso c) que se asemeja al trixel B y se busca obtener los círculos amarillos, tomando como límite la línea con círculos azules que es la rama del identificador del trixel B. Un caso que no se menciona es el inciso d) es un nodo al cual no se le ha indexado nada, y está apuntando a nulo.

4.1.4. Eliminar un documento

Para eliminar un documento, se tomaron en consideración algunos casos como:

- Eliminar un documento de un trixel.
- Eliminar algunos documentos de un trixel.
- Eliminar todos los documentos de un o varios trixeles.

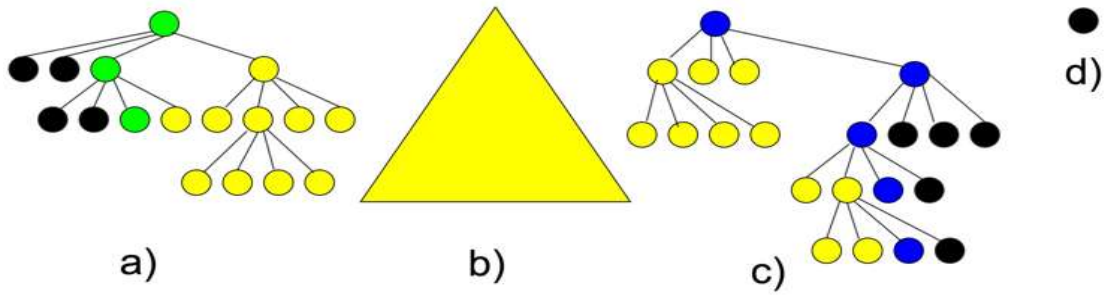

Figura 4.11: Búsquedas por ramas en el árbol de indexación propuesto.

La idea básica del indexador es por medio de la ruta del trixel liberar la memoria reservada del nodo y en las bases de datos sobre escribir las cabeceras del campo **Delete** para que sea marcado ese archivo como **hueco**, o bien utilizar una búsqueda por rango de trixeles para liberar todos los trixeles que se encuentren en un rango.

4.2. Descripción general del archivo de acceso aleatorio

Las cabeceras de los archivos que se almacenan en las bases de datos (archivos de acceso aleatorio), se componen por los siguientes elementos:

- Key. Identificador del archivo indexado, se requieren de 64 bits.
- Delete. Campo booleano que indica si el archivo sigue activo en la base de datos o fue eliminado por el usuario, por defecto el valor es false.
- Size. Cantidad de bytes que contiene el archivo a indexar, se requieren de 64 bits para almacenar el tamaño del archivo, este campo incluye la cantidad de bytes completa que se indexaron del archivo(Key, Delete, Size, Data, Checksum).
- Data. Bytes que componen el archivo que se almacena en las bases de datos.
- Checksum. Requiere de 32 bits que se utiliza para la verificación del archivo.

En la figura 4.12, se muestran las cabeceras de un archivo que se almacena en la base de datos.

Cada vez que se indexa un archivo, por parte del usuario se almacenan los parámetros para la recuperación del archivo, con los siguientes valores:

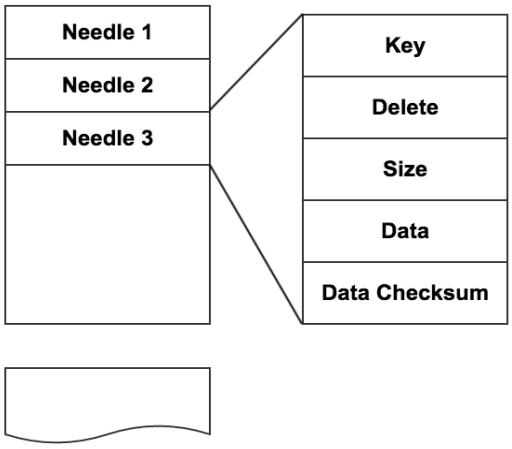

Figura 4.12: Estructuras de datos que componen el archivo de acceso aleatorio.

- Key. Identificador del archivo indexado, se requieren de 64 bits.
- Delete. Campo booleano utilizado para eliminar un documento del archivo aleatorio.
- Offset. Requiere de 64 bits, y con este se indica la posición del archivo indexado.
- Size. Requiere 64 bits que indican la cantidad de bytes que se indexaron, este campo incluye la cantidad de bytes completa que se indexaron del archivo(Key, Delete, Size, Data, Checksum).

En la figura 4.13, se muestran los parámetros para obtener un archivo almacenado en las bases de datos.

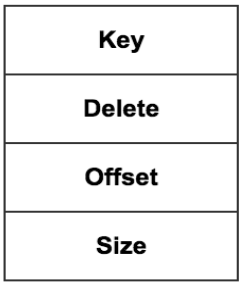

Figura 4.13: Estructura de datos para recuperar archivos de un archivo de acceso aleatorio.

Para seleccionar el tamaño total en bytes del archivo de acceso aleatorio, se tomó en cuenta las siguientes necesidades:

2 GB = 2147483648 bytes, necesita 32 bits
4 GB = 4294967296 bytes, necesita 64 bits
8 GB = 8589934592 bytes, necesita 64 bits
16 GB = 17179869184 bytes, necesita 64 bits
32 GB = 34359738368 bytes, necesita 64 bits

De acuerdo con esos valores se utilizaron 64 bits para representar el Offset y el Size, porque es un amplio rango para mantener una relación en bytes entre el archivo de acceso aleatorio y un identificador.

Las ventajas de un archivo con acceso aleatorio es limitar la cantidad de accesos al disco con operaciones, por lo que archivos de 2 ó 4 GB no ofrecen mucha ventaja, y los archivos de 8 GB, 16 GB y 32 GB, son mejores porque es más espacio de almacenamiento, pero la desventaja es que mientras más grandes son las bases de datos, tardara un poco más el proceso para almacenar un nuevo archivo. En este trabajo se utilizaron archivos en tamaño de 32 GB, y la razón es por convenciones del Sistema Operativo y capacidad del disco duro del equipo de computo utilizado.

Supongamos que en el peor de los casos, se requiere indexar un archivo de tamaño de 32 GB. Con esto el archivo de acceso aleatorio debe aceptar un porcentaje de error que está dado por la suma de las cabeceras como se muestra a continuación.

Key 64 bits
Delete 1 bit
Size 64 bits
Checksum 32 bits
Total : 161 bits = 20.125 bytes, si redondeamos hacia arriba se tienen aproximadamente 21 bytes

Un archivo de 32 GB es igual a 34359738368 bytes más 21 bytes da un total de 34359738389 bytes máximos que puede ocupar como límite para indexarse. Debido a las características del sistema donde se implementó, la asignación de bytes no son múltiplos de 1024, y son múltiplos de 1000, el archivo que se utilizó tiene 32000000000 bytes, con un máximo de 32000000021. Como ultima característica del archivo de acceso aleatorio es que es un archivo binario, con las operaciones de lectura, escritura, tamaño del archivo, buscar

una posición, y están hechas a base de operaciones en binario. Algunas de las funciones básicas que integra el archivo con acceso aleatorio son:

- Almacenar un archivo. Este es el proceso en que el archivo de acceso aleatorio, lee el archivo proporcionado por el usuario para escribirlo, realiza los cálculos para verificar si el archivo puede ser agregado la base de datos y escribe las cabeceras necesarias para su recuperación.
- Obtener un archivo. Para recuperar un archivo se utiliza los parámetros de ubicación en el archivo y tamaño de los datos que se proporcionan una vez que fue almacenado.
- Eliminar un documento. Al eliminar un archivo se sobre escribe el parámetro **Delete** que corresponde a la cabecera del archivo solicitado, y mediante la compactación se fragmenta las bases de datos y se crean secciones contiguas de almacenamiento.

La compactación es el proceso donde al archivo de acceso aleatorio se le aplica un recorrido lineal, leyendo únicamente las cabeceras en busca de los campos que marcan el archivo como eliminado, el objetivo de este proceso es eliminar los huecos del fichero y ofrecer más almacenamiento. Cabe mencionar que este proceso no es muy común que se realice porque elimina la información que interviene en las bases de datos, y es más sencillo crear un nuevo archivo del mismo tamaño de almacenamiento y que este disponible para su utilización. Pero en todo caso depende de las necesidades y características del sistema implementado.

4.3. Comentarios finales

En este capítulo se describieron las características del indexador propuesto, además se explican algunas características del archivo con acceso aleatorio. Al igual se menciona una función que es importante para liberar espacio que se llama compactación, y su importancia radica además de que libera espacio disponible, elimina huecos creando secciones contiguas en las bases de datos. En el capítulo 5 se muestran algunos ejemplos, se explica como se realizaron las pruebas y se habla del comportamiento completo del indexador propuesto.

Capítulo 5

Resultados

En el capítulo 4 se explicaron las características del árbol de indexación y del archivo con acceso aleatorio que sirve como base de datos. En este capítulo se va a mostrar algunos ejemplos para indexar y recuperar archivos en diferentes formatos, también se va a mostrar la eficiencia del indexador propuesto para recuperar documentos, y algunos conjuntos de trixeles indexados. El equipo de computo que se utilizó cumple con las siguientes características, laptop Mac Book Air, con procesador 1.6 GHz Intel Core i5, memoria RAM de 4 GB 1600 MHz DDR3, disco duro de estado sólido de 128 GB y con SO X Yosemite ver 10.10.5.

5.1. Pruebas del indexador

Para comprobar la velocidad del indexador propuesto se realizaron pruebas para indexar los siguientes conjuntos de trixeles:

- 10,000
- 100,000
- 1,000,000
- 10,000,000

Los identificadores de los trixeles que se generaron para pruebas son valores crecientes, utilizando una profundidad al nivel 25, con esto se observa que los trixeles indexados se encuentran en el caso más costoso ya que todos son nodos hojas. Cabe mencionar que

no se realizaron pruebas para eliminar trixeles, porque el algoritmo implementado cuando se requiere eliminar, primero realiza una búsqueda y posteriormente libera la memoria que fue antes reservada, además que está operación puede ser ignorada porque tiene una complejidad de tiempo constante O(1). Otros parámetros que se tomaron en cuenta durante la indexación, es el tiempo promedio para indexar, el tiempo promedio para buscar documentos y el porcentaje de recuperación en rangos de archivos desde un acceso aleatorio.

Para las pruebas que se realizaron con el indexador se utilizaron dos tipos de búsquedas:

- Primero búsquedas para obtener un número n de los trixeles más cercanos.
- Segundo búsquedas para obtener un rango de trixeles.

Para ambos casos los trixeles que sirven como punto de partida son valores aleatorios y el rango es cercano a 100 trixeles. Para obtener un número n de trixeles más cercanos, se implementaron dos métodos, el primero consiste en que cada vez que se desciende a un nivel se calcula todos los trixeles que se encuentran y así sucesivamente mientras va ascendiendo, el segundo método es una optimización del primero, al ascender un nivel, se guardó previamente el trixel padre que antes fue calculado, y sólo se realiza una búsqueda sobre sus nodos hermanos.

La tabla 5.1, representa el primer tipo de búsqueda para obtener conjuntos de n elementos, se puede notar que se recuperaron conjuntos de 100 y 1000 trixeles, además de mostrar el tiempo de indexación, se muestra el tiempo que se le invirtió por cada uno de los métodos de búsqueda. En la figura 5.1, se puede observar el tiempo que se tardó el indexador al obtener los trixeles. Las líneas sólidas muestran el comportamiento del indexador al recuperar los trixeles con el primer método, mientras las líneas punteadas muestran el comportamiento al utilizar el segundo método, se puede observar que al utilizar los datos que ya tenemos calculados, la velocidad de indexador aumenta y el tiempo se disminuye aproximadamente a la tercera parte con respecto al primer método.

La tabla 5.2, representa el segundo tipo de búsqueda, los rangos que se seleccionaron fueron aleatorios en diferentes secciones del árbol de indexación, pero se cuidó un parámetro aproximado a 100 trixeles, sin importar si se encuentra a un valor superior o inferior al conjunto. En la tabla 5.3, se puede observar además de un promedio de los valores antes calculados de la tabla anterior que en los tres primeros renglones se obtienen

Nº Trixeles	Conjuntos	T. de Indexación (seg.)	T. Método 1 (seg.)	T. Método 2 (seg.)
10,000	100	0.102145	0.000338	0.000164
	1000	0.118885	0.000895	0.000655
100,000	100	0.982483	0.000354	0.000194
	1000	0.991079	0.001108	0.000725
1,000,000	100	10.0213	0.000614	0.000255
	1000	10.0531	0.001632	0.000898
10,000,000	100	108.531	0.003205	0.00052
	1000	109.324	0.010916	0.006856

Tabla 5.1: Pruebas al recuperar los 100 y 1000 trixeles más cercanos.

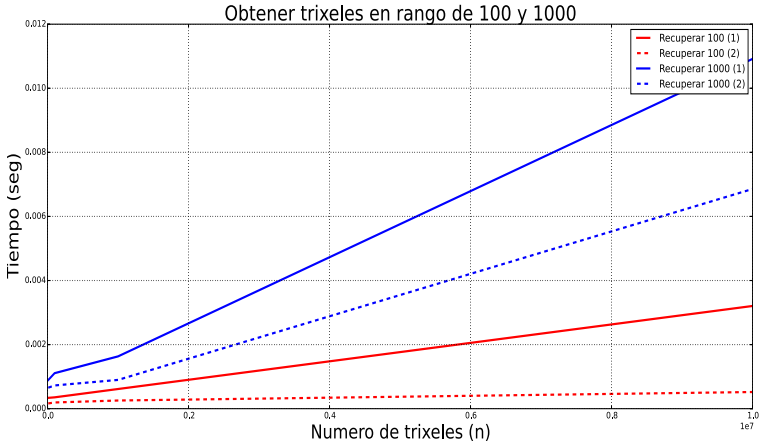

Figura 5.1: Indexar y recuperar en conjuntos de 100 y 1000 trixeles.

datos con un error de aproximadamente un diez por ciento por debajo de los 100 trixeles, a diferencia del renglón 4, donde se puede observar que se recuperan más valores de los que fueron seleccionados, finalmente en la figura 5.2, donde se puede observar gráficamente la expansión de las curvas al indexar y buscar trixeles.

Nº Trixeles	Tiempo indexación (seg.)	Documentos recuperados	Tiempo de búsqueda (seg.)
10,000	0.104028	89	0.011095
	0.105095	95	0.011783
	0.111768	91	0.00681
100,000	0.960475	97	0.001804
	0.971812	90	0.000854
	0.965352	93	0.001861
1,000,000	9.92232	89	2.92609
	10.1108	94	3.72307
	9.97194	109	3.10444
10,000,000	111.913	109	75.5713
	108.288	109	74.9628
	114.056	131	75.1964

Tabla 5.2: Pruebas al indexar y recuperar trixeles por rangos.

Nº Trixeles	Tiempo indexación (seg.)	Documentos Promedio	Tiempo de búsqueda (seg.)
10,000	0.106963667	91.66666667	0.001506333
100,000	0.965879667	93.33333333	0.009896
1,000,000	10.00168667	97.33333333	3.2512
10,000,000	111.419	116.3333333	75.2435

Tabla 5.3: Tiempo promedio entre rangos de trixeles.

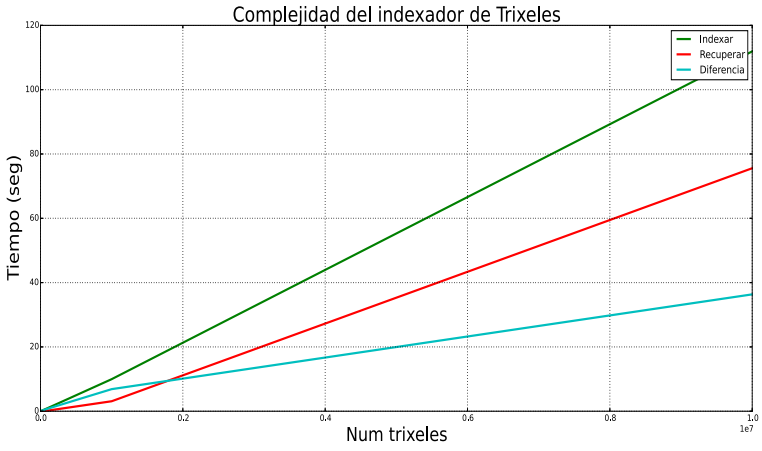

Figura 5.2: Indexar y recuperar en rangos aproximados a 100 trixeles.

5.2. Pruebas del expansión del archivo

Para comprobar la funcionalidad y medir la expansión del archivo de acceso aleatorio propuesto, se tomaron en consideración varios aspectos como se muestran a continuación:

- Indexar una colección mezclada de varios archivos con formato doc, pdf, jpeg, png, mp3 y txt o su equivalencia con archivos multimedia.
- Crear colecciones de archivos de tamaños aproximados a 20 KB, 1.3 MB y 10.7 MB que cumplan con él inciso anterior.

En la tabla 5.4, representa el tiempo que tardo el indexador, al expandirse hasta el límite de 32 GB, pasando por los estados de 2, 4, 8 y 16 GB. En la figura 5.3, se muestra la expansión de la curva en cada uno de los tres tipos de archivos que se utilizaron como pruebas. La línea azul muestra que mientras más archivos se almacenan tarda más por la cantidad de operaciones que se realizan, a diferencia de la línea verde, donde el tamaño en bytes de los archivos es mayor por lo cual se tarda menos.

La complejidad de realizar operaciones con el archivo de acceso aleatorio, tienen como base complejidad lineal, ya que leer y escribir un archivo tienen O(n), donde n, es el número de bytes que se pueden leer. En esta prueba se utiliza un número k de archivos, donde la complejidad está dada por O(kn), en el peor de los casos es cuando k es igual a n, por lo tanto la complejidad es igual a O(n^2).

Base de datos (GB)	Archivos 20 KB(seg.)	Archivos 1.3 MB(seg.)	Archivos 10.7 MB(seg.)
2	22.7703	21.3328	19.9822
4	44.9325	42.3528	41.1377
8	89.9322	86.1461	79.0201
16	181.3	177.558	155.567
32	352.663	317.498	318.436

Tabla 5.4: Tiempo de expansión del archivo de acceso aleatorio.

5.3. Comentarios finales

En esta sección se mostraron algunos ejemplos al ejecutar el indexador con las operaciones para indexar y buscar con rangos, al igual para buscar utilizando conjuntos de

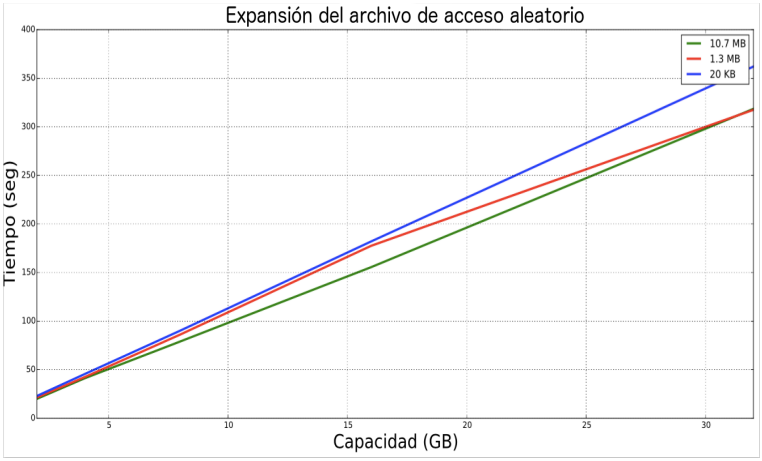

Figura 5.3: Expansión del archivo de acceso aleatorio.

trixeles, y finalmente se muestra el comportamiento de la expansión al archivo de acceso a aleatorio. En la figura 5.1, se observa de las líneas punteadas que se mejoró la eficiencia al recuperar conjuntos de trixeles, ya que este tipo de búsqueda primero se basa en la probabilidad para obtener una cantidad de trixeles desde el identificador y su profundidad desde un trixel padre, además se mantiene la referencia de los nodos que ya se visitaron y obtiene los nodos que hacen falta. En la figura 5.2, se observa el tiempo para indexar y el tiempo para buscar con ramas, la diferencia entre estas dos operaciones son que al indexar, primero se busca para crear las estructuras de datos reservando la memoria necesaria, y en las búsquedas se obtiene el trixel padre entre los identificadores de los trixeles y se realiza un proceso recursivo para acceder a las ramas. La cantidad de operaciones adicionales que se realizan se muestran en la tercera línea que es la diferencia de tiempo al indexar y buscar trixeles. En la figura 5.3, se muestra el comportamiento al almacenar archivos pequeños y grandes en las bases de datos, además se hizo la prueba con otro archivo que no es ni pequeño ni grande, y el comportamiento muestra que en una base de datos vacía, la expansión se asemeja a un archivo pequeño, hasta que llega a un tamaño y se muestra que disminuye la velocidad al expandirse la base de datos, lo cual es favorable porque se asemeja a almacenar archivos de mayor tamaño.

Capítulo 6

Conclusiones y Trabajo Futuro

En el capítulo 5 se muestran algunos ejemplos del indexador propuesto, realizando algunas pruebas de búsquedas e indexación, para obtener el tiempo que se tarda, además se muestran dos tipos de búsquedas para obtener los n trixeles más cercanos de un trixel, ahí se mostró que al hacer una modificación en el algoritmo de búsqueda se obtiene una mejor velocidad. En este último capítulo, se habla de la conclusión general y conclusiones particulares acerca del trabajo realizado. En la sección de trabajos futuros, se hablan de algunas modificaciones y las características adicionales que se pueden agregar al indexador propuesto, para hacerlo simple de utilizar.

6.1. Conclusiones

6.1.1. Conclusión general

El desarrollo de esta tesis fue un trabajo muy interesante, ya que tuve la oportunidad de conocer acerca del funcionamiento que realizan los indexadores de texto de manera general y aplicarlo con un enfoque diferente. Proponer un indexador, fue una tarea un tanto compleja, ya que investigue el funcionamiento de los árboles de indexación y la manera de mantener la relación con el índice de datos, finalmente se pudo comprobar que la manera de crear las estructuras de datos, las funciones, comunicación de procesos y la cantidad de operaciones son un factor importante en la eficiencia del indexador y la relación consistente con las bases de datos. Para la base de datos se agregó la estrategia de los archivos con acceso aleatorio que implementa Facebook, ya que son sencillos de implementar y ofrecen buena

velocidad para almacenar y recuperar archivos, además por la cantidad de documentos que se almacenan es otra ventaja a los nodos del sistema de archivos del SO.

6.1.2. Conclusiones particulares

Las conclusiones particulares que se obtuvieron al desarrollar el indexador propuesto son:

- El algoritmo de malla triangular jerárquica es una buena estrategia para obtener una reconstrucción de la superficie de la esfera terrestre, y así mismo obtener un punto de referencia al buscar lugares con un sistema geoespacial, pero es importante señalar que mientras se busque minimizar el error del área de los trixeles se requiere más tiempo de computo. En algunas ocasiones esto no es un buen comportamiento, pero todo depende de la zona que se requiere reconstruir.

- El algoritmo Graham Scan es una buena herramienta que permite crear zonas que agrupan puntos geoespaciales. Si tomamos como referencia los puntos que se encuentran dentro del algoritmo del Graham para obtener los trixeles que se encierran en la envoltura, obtener un conjunto de trixeles se realiza de una manera más rápida, porque basta de un puntero a la zona que encierra el Graham, y es una buena estrategia que ayuda a la velocidad de respuesta del indexador propuesto.

- El máximo número de ramas que se pueden crear en este indexador por un cuadrante, está dado por 4^{25} que es igual a 1125899906842624 nodos en el árbol de indexación.

- Para obtener estos resultados se realizaron previamente algunos conjuntos de pruebas, en las que se estuvo mejorando las características de los algoritmos implementados, en especial al designar como se construyen las rutas de los trixeles y las llamadas recursivas.

- La expansión del archivo de acceso aleatorio crece a proporción de los archivos indexados en los trixeles, la diferencia se puede notar en almacenar archivos pequeños, ya que se tarda más a respecto de almacenar archivos grandes en tamaño de Bytes.

- Las estructuras de datos de tipo árbol en especial los árboles balanceados pero no son necesariamente óptimos para todas las aplicaciones, ya que se depende de los datos que se están almacenando y como se puede acceder a ellos.

6.2. Trabajo Futuro

Las posibles extensiones de este indexador propuesto para que sea más fácil de usar y se pueda asemejar con las características y funciones de los indexadores actuales, son por ejemplo:

- Programar un algoritmo para sobreponer un polígono convexo o alguna otra figura como un cuadrado, rectángulo o circulo sobre los trixeles y realizar búsquedas o poder recuperarlos utilizando esa referencia.
- Obtener un algoritmo que se encargue de obtener de una mejor manera los archivos que se encuentran almacenados en cada trixel indexado, especialmente en el casó de búsquedas recursivas por ramas de izquierda a la derecha o de derecha a izquierda.
- Implementar una interfaz de comunicación para realizar consultas al indexador de trixeles, además de implementar una API para mantener la comunicación entre el cliente y servidor, y de esta manera se puede expandir el sistema de almacenamiento a diferentes bases de datos, donde cada una será representada por un archivo de acceso aleatorio.

Referencias

[1] Cormen, T. H. (2009). Introduction to algorithms. MIT press.

[2] Kernighan, B. W., Ritchie, D. M. (1988). The C programming language (Vol. 2). Englewood Cliffs: prentice-Hall.

[3] Luis Joyanes aguilar, Lucas Sanchez Garcia, Ignacio Zahonero Martinez, Estructuras de datos en C++,

[4] W.A. Burkhard, Best-Match File Searching, University of California, San Diego and R.M. Keller Princeton University

[5] José Martín Ruiz Pérez, Antonio Camarena-Ibarrola, On the use of Proximity Indexes for Isolated Word Recognition, Universidad Michoacana de San Nicolas de Hidalgo Facultad de Ingeniería Eléctrica División de Estudios de Posgrado Morelia, Michoacán, México.

[6] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín, Searching in metric spaces, ACM Comput. Surv., vol. 33, no. 3, pp. 273 321, Sep. 2001. [Online]. Available: http://doi.acm.org/10.1145/502807.502808

[7] Ricardo Baeza-Yates Gonzalo Navarro, Fast Approximate String Match in a Dictionary

[8] Alexander S. Szalay, Jim Gray, George Fekete Peter Z. Kunszt, Peter Kukol, and Ani Thakar, Indexing the Sphere with the Hierarchical Triangular Mesh,

[9] Zhenhua Lv, Yingjie Hu, Haidong Zhong, Bailang Yu, Bo Li, Hui Zhao, Spatial Indexing of Global Geographical Data with HTM

[10] Geoffrey Dutton, Encoding and Handling Geospatial Data with Hierarchical Triangular Meshes, Department of Geography, University of Zürich.

[11] Gold, C., Angel, P. (2006). Voronoi hierarchies. In Geographic Information Science (pp. 99-111). Springer Berlin Heidelberg.

[12] Zhao, X., Chen, J., Li, Z. (2002). A QTM-based Algorithm for Generation of the Voronoi Diagram on a Sphere. In Advances in Spatial Data Handling (pp. 269-283). Springer Berlin Heidelberg.

[13] Gorski, K. M., Hivon, E., Banday, A. J., Wandelt, B. D., Hansen, F. K., Reinecke, M., Bartelmann, M. (2005). HEALPix: a framework for high-resolution discretization and fast analysis of data distributed on the sphere. The Astrophysical Journal, 622(2), 759.

[14] Hivon, E., Hansen, F. K., Banday, A. J. (1999). The healpix primer. arXiv preprint astro-ph/9905275.

[15] Górski, K. M., Banday, A. J., Hivon, E., Wandelt, B. D. (2002). HEALPix—a Framework for High Resolution, Fast Analysis on the Sphere. In Astronomical Data Analysis Software and Systems XI (Vol. 281, p. 107).

[16] White, R. A., Stemwedel, S. W. (1992). The quadrilateralized spherical cube and quadtree for all sky data. In Astronomical Data Analysis Software and Systems I (Vol. 25, p. 379).

[17] Tobler, W., Chen, Z. T. (1986). A quadtree for global information storage. Geographical Analysis, 18(4), 360-371.

[18] Lawder, J. K. (2000). Calculation of mappings between one and n-dimensional values using the hilbert space-filling curve. School of Computer Science and Information Systems, Birkbeck College, University of London, London Research Report BBKCS-00-01 August.

[19] Alber, J., Niedermeier, R. (2000). On multidimensional curves with Hilbert property. Theory of Computing Systems, 33(4), 295-312.

[20] Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, Peter Vajgel, Finding a needle in Haystack: Facebook's photo storage.

[21] Whitaker, A., Cox, R. S., Gribble, S. D. (2004, December). Configuration Debugging as Search: Finding the Needle in the Haystack. In OSDI (Vol. 4, pp. 6-6).

[22] Dalenius, T. (1986). Finding a needle in a haystack or identifying anonymous census records. Journal of official statistics, 2(3), 329.

[23] Samet, H. (1984). The quadtree and related hierarchical data structures. ACM Computing Surveys (CSUR), 16(2), 187-260.

[24] Finkel, R. A., Bentley, J. L. (1974). Quad trees a data structure for retrieval on composite keys. Acta informatica, 4(1), 1-9.

[25] Kamel, I., Faloutsos, C. (1993). Hilbert R-tree: An improved R-tree using fractals.

[26] Katayama, N., Satoh, S. I. (1997, June). The SR-tree: An index structure for high-dimensional nearest neighbor queries. In ACM SIGMOD Record (Vol. 26, No. 2, pp. 369-380). ACM.

[27] Xapian/xapian GitHub. Disponible en `https://github.com/xapian/xapian`.

[28] Elastic Revealing Insights from Data (Formerly Elasticsearch). Disponible en `https://www.elastic.co/.`

[29] Elastic/elasticsearch: Open Source, Distributed, RESTful Search Engine GitHub. Disponible en `https://github.com/elastic/elasticsearch`.

[30] Apache Lucene - Welcome to Apache Lucene. Disponible en `https://lucene.apache.org/`.

[31] Mirror of Apache Solr. Disponible en `https://github.com/apache/solr`.

[32] Chan, T. M. (1996). Optimal output-sensitive convex hull algorithms in two and three dimensions. Discrete Computational Geometry, 16(4), 361-368.

[33] Barber, C. B., Dobkin, D. P., Huhdanpaa, H. (1993). The quickhull algorithm for convex hull. Technical Report GCG53, The Geometry Center, MN.

[34] Overmars, M. H., Van Leeuwen, J. (1981). Maintenance of configurations in the plane. Journal of computer and System Sciences, 23(2), 166-204.

[35] Bouldin, D. W. The Use of a VLSI Sorter Coprocessor to Find the Convex Hull* Arnulfo Perez Moaigi A. Ahidi.

[36] Barber, C. B., Dobkin, D. P., Huhdanpaa, H. (1993). The quickhull algorithm for convex hull. Technical Report GCG53, The Geometry Center, MN.

[37] Kirkpatrick, D. G., Seidel, R. (1986). The ultimate planar convex hull algorithm?. SIAM journal on computing, 15(1), 287-299.

[38] Chan, T. M. (1996). Output-sensitive results on convex hulls, extreme points, and related problems. Discrete Computational Geometry, 16(4), 369-387.

[39] Kuc, R. (2013). Apache Solr 4 Cookbook. Packt Publishing Ltd. Disponible en `https://books.google.com.mx/books?hl=es&lr=&id=JGHGdv2NET8C&oi=fnd&pg=PT13&dq=Tika+Solr+apache+lucene&ots=CwFg173T6D&sig=KsDkDzvy0QJq5qik_sSLBXkwsM4#v=onepage&q&f=false`

[40] Ilustración del algoritmo de malla triangular jerárquica. Disponible en `http://www.geog.ucsb.edu/~good/176b/qtm.jpg`.

[41] Ilustración del modelo de Hipparchus Voronoi. Disponible en `http://www.geodyssey.com/papers/hlauto8fig1.png`.

[42] Ilustración de las curvas de Hilbert. Disponible en `https://upload.wikimedia.org/wikipedia/en/thumb/a/a5/Hilbert_curve.svg/400px-Hilbert_curve.svg.png`.

[43] Ilustración del modelo HEALPix. Disponible en `http://healpix.sourceforge.net/images/gorski_f1.jpg`.

[44] Ilustración del modelo Quadrilateralized Spherical Cube. Disponible en `https://trac.osgeo.org/proj/raw-attachment/wiki/proj3Dqsc/sphere-in-cube.jpg`.

[45] Ilustración del sistema de coordenadas latitud-longitud. Disponible en `http://www.aularagon.org/files/espa/atlas/Latitud_longitud_mundo.gif`

[46] Ilustración del árbol-R. Disponible en `https://www.google.com.mx/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwiH29iSg-XKAhXF7SYKHTJkCGMQjRwIBw&url=https%3A%2F%2Fes.wikipedia.org%2Fwiki%2F%25C3%2581rbol-R&psig=AFQjCNFDy1yrExeWzh8bLcqX9AdNESro2Q&ust=1454912924642938`

[47] Ilustración del árbol-Kd. Disponible en `https://www.google.com.mx/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjAq7Kmg-XKAhUMTCYKHUI5CZ4QjRwIBw&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FK-d_tree&psig=AFQjCNE2LXS_1ka3paEtQAL_PugS3B6-SA&ust=1454912965745091`

[48] Ilustración del árbol-B*. Disponible en `https://encrypted-tbn1.gstatic.com/images?q=tbn:ANd9GcSKAL0sOcWd9J2CzDls_1I4spYcpzqwgjh1FkLmLKGEUr9rbaZz`