Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería en Ciencias de la Computación
CC4102 Diseño y analisis de algoritmos

Mtree

Tarea 1

Integrantes: Amaro Zurita
Javier Andrés Facondi
Profesores: Benjamín Bustos
Auxiliar: Sergio Rojas H.

1. Introducción

El M-tree es un árbol balanceado fundamental en la búsqueda de datos en espacios métricos, donde la eficiencia de las consultas depende en gran medida de la organización de los datos y la minimización de la superposición entre las regiones espaciales representadas por los nodos del árbol.

El presente informe aborda la implementación y evaluación de dos métodos de construcción de M-Tree, a saber, el Método Ciaccia-Patella (CP) y el Método Sexton-Swinbank (SS). Ambos métodos buscan optimizar la estructura del árbol para mejorar el rendimiento de las búsquedas, reduciendo la superposición entre las regiones espaciales representadas por los nodos.

Nuestra hipótesis es que el Metodo Sexton-Swinbank(SS) va a tener un tiempo de construcción mas largo en comparación con el Metodo Ciaccia-Patella, ya que este involucra un mayor numero de operaciones y ajustes.

A medida que avanzamos en este informe, examinaremos en detalle cada uno de estos métodos de construcción de M-Trees y viendo cómo influyen en la eficiencia de la búsqueda utilizando el lenguaje de programación C++.

2. Desarrollo

Para esta tarea, se tuvo que programar 2 métodos de construcción de M-tree, los cuales son: Ciaccia-Patella (CP) y Sexton-Swinbank (SS).

Por último, se implementa un método de búsqueda recursivo para el M-Tree, así como también funciones que permiten realizar tests para comparar el rendimiento de estos métodos al realizar búsquedas.

A continuación se detallan estas implementaciones.

2.1. Estructuras y Clases para M-Tree:

Se creo la siguiente estructura

- *Point*: Representa un punto y contiene como atributo sus coordenadas x e y.

  Junto con las siguientes clases:

- *Entry*: Representa una entrada. Contiene 3 atributos:

  1. Point : Punto correspondiente a la entry.
  2. Covering Radius : Radio cobertor según sus hijos.
  3. Child : Puntero a un Mtree que es hijo de la entrada.

- *MtreeNode* : Representa a un Mtree el cual contiene 2 atributos.

  1. vector de entries: Vector donde se almacenan todos las entradas que a su vez contienen los subárboles del árbol conformando la estructura.
  2. isLeaf : Valor booleano que indica si el nodo es una hoja o no.

Cuando se hace uso de estas 2 clases y la estructura Point obtendremos la siguiente estructura de MTree:

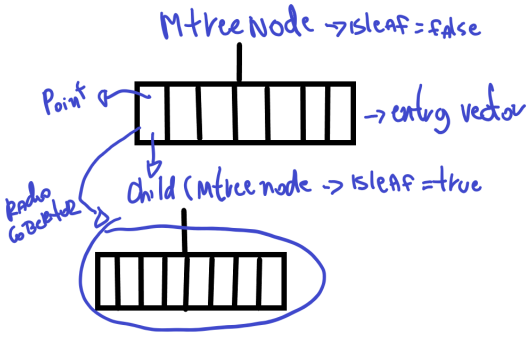

Figura 2.1: Estructura Mtree

2.1.1. Método Ciaccia-Patella (CP)

Para la implementación de este método se utilizaron la siguientes funciones auxiliares:

- *height* : Mediante uso de la recursion recorre el árbol y retorna la altura máxima de un MtreeNode
- *rellenar_TSup* : Función auxiliar que recibe un MtreeNode correspondiente al árbol superior, un vector de arboles y un vector de puntos. Esta función designa los arboles pertenecientes al vector con sus entries correspondientes pertenecientes al árbol Tsup mediante uso de comparaciones y recursion.
- *radiosCobertores*: Función que recibe un MtreeNode y designa los radios cobertores de todas sus entries.
- *mtreeheightfinder*: Función auxiliar que recibe un MtreeNode, una altura de búsqueda, un vector de MtreeNode's y un vector de puntos. Esta función busca subárboles de la altura h y los almacena en el vector de arboles y a su vez almacena su punto padre en el vector de puntos.
- *mtreebooleansetter*: Funcion auxiliar que recibe un MtreeNode y settea los valores de isLeaf en el árbol.

Funcionamiento del Método

El método recibe un conjunto P de n puntos y tiene B/sizeofentry() como la máxima cantidad de hijos de cada nodo del árbol considerando B como el tamaño del bloque de memoria, el método en caso de que se tengan menos elementos en el conjunto que la cantidad máxima de hijos crea un árbol hoja en el que inserta todos los puntos como entries, en cambio si el tamaño es mayor crea un vector de puntos samples y un vector por cada punto sample en

el que se almacenaran el resto de puntos basándose en su distancia euclidiana para que estos se asignen a su sample mas cercano en caso de que un sample no sea óptimo y tenga menos de la mitad de la cantidad máxima de hijos sera eliminado y estos puntos redistribuidos entre los samples restantes, si el método encuentra mas de un punto sample proseguirá con la creación de subárboles para cada vector. Luego de seleccionar los subárboles se hará un calculo de la altura mínima entre los mismos, en caso de que un árbol no cumpla con esta altura usara la función *mtreeheightfinder* para hacer una búsqueda de todos los subárboles existentes dentro del árbol que cumplan con la altura mínima dentro del conjunto. Finalmente con el conjunto de samples se creara un MtreeNode el cual sera rellenado usando la función *rellenarT_Sup* con los subarboles encontrados previamente, además de settear sus valores isLeaf con *mtreebooleansetter* y sus radios cobertores con *radiosCobertores* una vez estos 2 procesos terminados se retorna el Mtree correspondiente.

2.1.2. Método Sexton-Swinbank

Funciones auxiliares:

- *medoide*: Función auxiliar que recibe un vector de puntos(cluster) y entrega su medoide. Para hacer esto, se calcula iterando sobre todos los puntos y calculando la suma de las distancias de este punto a todos los demas puntos del cluster. Luego el metoide es el punto tal que la suma total de las distancias desde este punto a todos los demás puntos del cluster es mínima.

- *parMasCCercano*: Función auxiliar que recibe un conjunto de clusters, y entrega un par de clusters tal que su distancia en mínima en comparación con cual otra distancia entre clusters del conjunto. Para lograr esto, primero se calculan los medoides de cada cluster en el conjunto. Luego, se itera sobre todos los pares de medoides y se calcula la distancia euclidiana entre ellos. Se selecciona el par con la distancia mínima y se devuelve como resultado. Esto optimiza el cálculo de las distancias entre clusters al calcular primero los medoides de cada uno.

- *vecinoMasCercano*: Función auxiliar que recibe un conjunto de clusters y un cluster específico, y devuelve el cluster más cercano al cluster dado. Para hacer esto, primero se calcula el medoide del cluster dado. Luego, se itera sobre todos los clusters en el conjunto y se calcula la distancia euclidiana entre el medoide del cluster dado y el medoide de cada uno de los otros clusters. Se selecciona el cluster con la distancia mínima y se devuelve como resultado.

- *calculateMaxCoveringRadius*: Función auxiliar que recibe dos grupos de puntos y sus respectivos medoides, y devuelve el radio de cobertura máximo entre los dos grupos. Para hacer esto, la función itera sobre todos los puntos en ambos grupos y calcula la distancia euclidiana entre cada punto y su respectivo medoide. Luego, selecciona la distancia máxima entre todas estas distancias y la devuelve como el radio de cobertura máximo entre los dos grupos.

- *splitMinMax*: Función auxiliar que recibe un conjunto de puntos y los divide en dos grupos utilizando la política de división MinMax. Para hacer esto, la función itera sobre todos los pares de puntos en el conjunto y calcula el radio de cobertura máximo entre los grupos resultantes si se dividen el conjunto en dos partes usando los puntos actuales como medoides. Luego, la función selecciona el par de puntos que minimiza el radio de cobertura máximo y divide el conjunto en dos grupos utilizando esos puntos como medoides. Finalmente, devuelve un par de vectores que representan los dos grupos resultantes.

- *Cluster*: Función auxiliar que recibe un conjunto de puntos y retorna un conjunto de clusters, donde cada cluster tiene un tamaño entre b y B. Inicialmente, crea un cluster para cada punto de entrada y los almacena en C. Luego, mientras haya más de un cluster en C, la función une los clusters más cercanos si su tamaño combinado no excede B, o agrega uno de los clusters a Cout si no. Después, toma el último cluster restante en C y lo une con su vecino más cercano de Cout si su tamaño combinado no excede B, o divide el conjunto en dos grupos usando la política MinMax y agrega los grupos resultantes a Cout. Finalmente, la función retorna Cout, que contiene los clusters resultantes.

- *OutputHoja*: Función auxiliar que toma un conjunto de puntos Cin y retorna una tupla (g, r, a), donde g es el medoide primario de Cin, r es el radio cobertor, y a es el puntero al nodo hoja correspondiente. Inicialmente, encuentra el medoide primario g de Cin y establece r como 0. Luego, crea un nuevo nodo hoja C y agrega cada punto de Cin como una entrada en C, actualizando r con la distancia máxima entre g y los puntos. Finalmente, guarda el puntero a C como a y retorna la tupla (g, r, a).

- *OutputInterno*: Función auxiliar que recibe un conjunto de entradas Cmra, donde cada entrada contiene un punto, un radio de cobertura y una dirección de hijo. La función construye un nodo de árbol C que contiene las entradas del conjunto Cmra. Para hacer esto, primero extrae los puntos del conjunto de entradas Cmra y calcula el medoide G de estos puntos utilizando la función medoide. Luego, inicializa el radio de cobertura R como 0 y crea un nuevo nodo de árbol C. La función itera sobre todas las entradas en Cmra, agrega cada entrada al nodo de árbol C y actualiza el radio de cobertura R como el máximo entre el radio de cobertura actual y la suma de la distancia entre el medoide G y el punto de la entrada más el radio de cobertura de esa entrada. Finalmente, la función devuelve una entrada e que contiene el medoide G, el radio de cobertura R y un puntero al nodo de árbol C.

Funcionamiento del Método

El metodo recibe un conjunto de puntos Cin y devuelve la raiz del M-tree construido. El caso base que tenemos es que si el tamaño del conjunto de puntos Cin es menor o igual que el parámetro B, se llama a la función OutputHoja(Cin) para crear una hoja del M-tree y se devuelve esta hoja.

Luego se divide el conjunto de puntos Cin en clusters utilizando la función Cluster(Cin), guardandolos en Cout, y se inicializa un conjunto de entradas C vacío.

Por cada cluster en Cout, se llama a la funcion OutputHoja(c), para crear una entrada correspondiente a ese cluster y se añade esta entrada al conjunto C.

Mientras el tamaño del conjunto C sea mayor que B, creamos un conjunto de entradas Cmra. Para cada cluster c en Cout, se seleccionan las entradas en C que tengan sus puntos en c, y estos se agregan a Cmra. Luego se vacia C y se llena nuevamente con las entradas al llamar a OutputInterno para cada entrada en Cmra.

Ya fuera del while, se llama a OutputInterno para el conjunto final de entradaas en C y se guarda el resultado en x. Finalmente retorna la raiz del M-tree construido, es decir, x.child.

2.1.3. Método de Búsqueda

Se implementa el método de búsqueda recursivo del árbol llamado *search* el cual recibe un MtreeNode, un punto de búsqueda, el radio de búsqueda y un vector donde se irán almacenando puntos durante la recursion. Se accede al árbol y se revisan todas sus entries en donde si se cumple la condición de que $dist\_euc(entry.punto, puntodebusqueda) <= radiodebusqueda + entry.radioCobertor$ revisa si el punto esta dentro del radio de búsqueda y si lo esta lo agrega independiente de esto inicia una búsqueda recursiva dentro del árbol del entry en caso de que sea una hoja no se aplica recursividad.

2.1.4. Test

Se crean tests para probar la construcción del árbol con cada método y posteriormente medir los accesos a discos y el tiempo de ejecución en la búsqueda, para esto se escogió B = 4096, dado que cada nodo ocupa 32 bytes se tendrá que la cantidad máxima de entries por MtreeNode es de 128 bytes. Posteriormente se crean un set de puntos aleatorios con coordenadas entre 0 y 1 con tamaño n con n e $\{2^{10}, ..., 2^{25}\}$(para el método SS se intento ejecutar en valores sobre n^{14}, pero este era muy lento) con todo esto se crean los arboles con estos set de puntos y se ejecutan 100 consultas a la función *search* guardando un archivo .txt con la duración promedio de cada consulta, intervalo de confianza y la cantidad de accesos a disco.

3. Resultados

Para la ejecución del código se utilizo un pc con las siguientes especificaciones:

- Procesador:
  - Modelo: I9-12900k
  - Cores: 6
  - Threads: 12
  - Frecuencia: 2,9 GHz a 4,3 GHz
  - Cache N2: 1,5MB
  - Cache N3: 12MB
- Memoria principal (RAM): 16 GB a 2666Mhz
- Memoria secundaria :
  - Modelo: WDC PC SN730 SDBQNTY-256G-1001
  - Capacidad: 239 GB
  - Velocidad de lectura: 140 KB/s
  - Velocidad de escritura: 98,1 KB/s
  - Tamaño de bloque: 4096 bytes
- Sistema operativo: Windows 11
- Compilador: GCC 13.2

A continuación presentaremos los gráficos de los resultados obtenidos al realizar los experimentos, luego se analizara y discutirá acerca del comportamiento y rendimiento individual de los métodos de construcción implementados usando el método de búsqueda.

3.1. Metodo Ciaccia-Patella (CP)

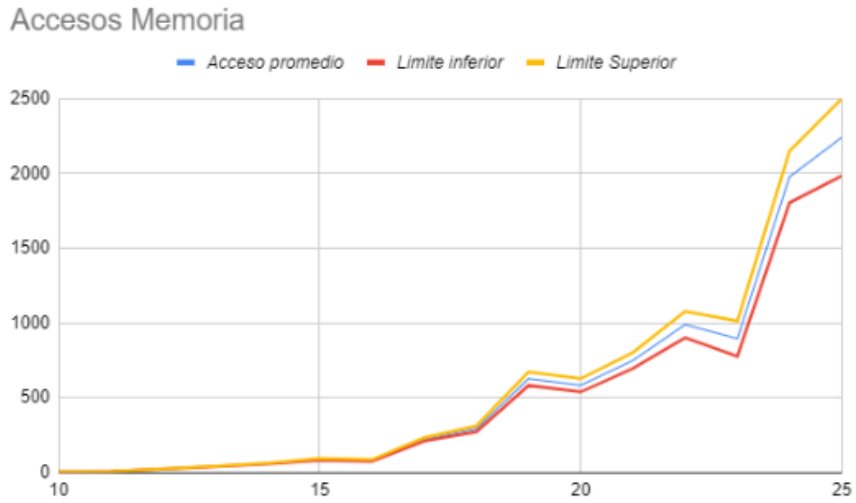

Figura 3.1: Accesos a memoria

Se puede observar de la figura 3.1 que el comportamiento de los accesos aumenta proporcionalmente a N, específicamente toma un comportamiento similar a $NlogN$. los accesos varían entre 7 y 2240 tomando valor promedio.

Figura 3.2: Tiempo promedio de búsqueda en µS

Análogamente, el tiempo promedio de búsqueda también tiende a $blogN$ con b los puntos dentro del radio de búsqueda. variando desde 40 a 14250µS

3.2. Sexton-SwinBank (SS)

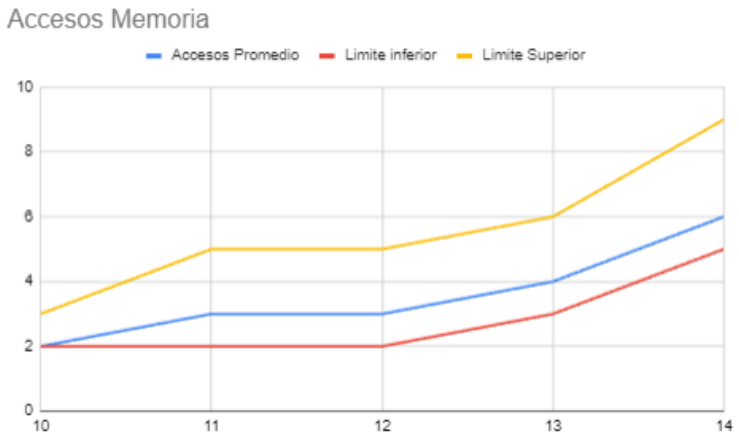

Figura 3.3: Accesos a memoria

Se puede observar de la figura 3.3 que que los accesos incrementan con N de forma similar a $NLlogN$ desde N=12, los accesos varían entre 2 y 9 en promedio.

Figura 3.4: Tiempo promedio de búsqueda en µS

Se puede observar que el tiempo promedio de búsqueda en SS es similar a un tiempo lineal que varia entre 10 a 60µS

3.3. Comparación de métodos

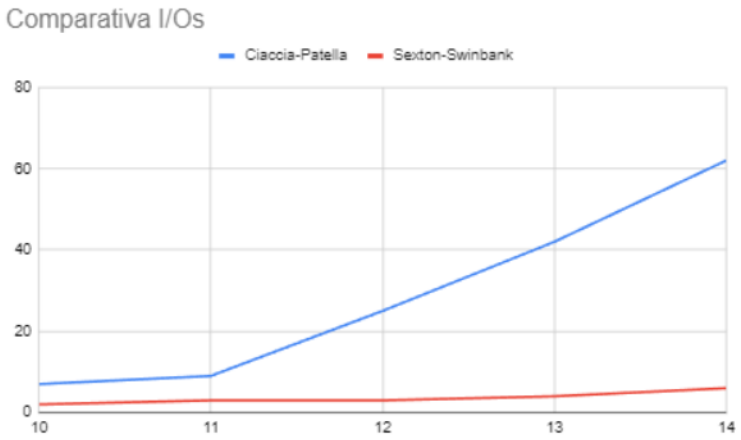

Figura 3.5: Comparativa I/Os

Podemos notar que Ciaccia-Patella hace una cantidad notablemente mayor de accesos a discos comparada con el método Sexton-Swinbank, esto se puede prever, dados sus comportamientos

Figura 3.6: Comparativa Tiempo de búsqueda en µS

Análogamente el tiempo de búsqueda también es mayor en Ciaccia-Patella.

Figura 3.7: Construcción método Ciaccia-Patella

Figura 3.8: Construcción método Sexton-swinbank

Podemos observar en las figuras 3.7 y 3.8 la abismal diferencia entre el tiempo de construcción de ambos métodos y esto estaba previsto dados los pasos de ajuste del método Sexton-Swinbank.

3.4. Discusión

De acuerdo a los resultados obtenidos, se puede concluir que el método mas eficiente con respecto al tiempo y accesos a disco es el método Sexton-Swinbank.

Con respecto a que el SS sea mas eficiente esto se da a la forma final del Mtree generado, ya que la clusterizacion consigue minimizar la cantidad de accesos necesarios para encontrar los puntos a diferencia de el método Ciaccia Patella que asigna samples aleatorios en la formación del Mtree. Esto se resume en que el método Sexton-Swinbank agrupa sus puntos de manera mas uniforme.

Sin embargo, esto produce de que el método Sexton-Swinbank se demore significativamente mas que el método Ciaccia-Patella en **construir el árbol**. Esto se debe a que durante el armado se tienen que llamar múltiples veces a funciones con un costo considerable, como el medoide, que no se puede calcular en menos que O(n^2), o el subproblema del par mas cercano, que en nuestra implementación tiene complejidad O(n^2), aunque este problema se puede optimizar para que tenga complejidad O(nlogn), pero no se pudo lograr. La idea de esta optimización era hacer un sort, lo cual tarda O(nlogn) y con eso trabajar para poder devolver el par más cercano.

A diferencia del método Ciaccia-Patella el cual en su iteración mas pesada durante su construcción solo llega a O($(N-1)^2 logN$) que ocurre durante la etapa de balanceamiento al comprobar las alturas con el peor caso posible donde un subárbol tenga una altura menor a todos los demás arboles.

No menos importante es mencionar que el método Ciaccia-Patella puede tener un mejor rendimiento con datos no uniformes, dado que esto lograra que los puntos se distribuyan de mejor manera entre los samples aleatorios y se realicen menos accesos por el método de búsqueda.

Es destacable el hecho de que los intervalos de confianza obtenidos(visibles en las figuras 3.1 y 3.3) varían bastante mientras se va aumentando el N esto podría ser un indicio de que nuestros métodos carecen de optimización, ya que no son consistentes para grandes cantidades de datos.

4. Conclusión

Luego de realizar la implementación de dos métodos de construcción para M-Tree y la implementación de búsqueda sobre este, analizando también el tiempo y cantidad de accesos en promedio, podemos concluir que sin un manejo claro de los conceptos teóricos la implementación de estos se complica bastante, por lo que es necesario el estudio de estos previo a implementar.

Recapitulando, se ha observado que el Método Ciaccia-Patella destaca por su rapidez en la etapa de construcción, mientras que el Método Sexton-Swinbank se distingue por su eficiencia en búsquedas, requiriendo menos accesos a disco y logrando resultados más rápidos en este aspecto.

Como se menciono anteriormente, durante el proceso de implementación y evaluación, se observó consistentemente que el Método Sexton-Swinbank requería significativamente más tiempo para construir el M-Tree en comparación con el Método Ciaccia-Patella. Esto era de esperarse, sin embargo no de una manera tan grande, ya que el tiempo para ejecutar el 2^{15} puntos era tan grande que no se alcanzo a completar para el segundo método(Sexton-Swinbank). Esto probablemente se deba a que en alguna parte de nuestro código debe de haber una mala optimización que se podría mejorar, que provoca que aumente considerablemente el tiempo de ejecución de la construcción del método, ya que este debería de poder construirse en $O(n^3)$ aproximadamente.

Este hallazgo apoya la premisa inicial de nuestra hipótesis, que predijo que el Método Ciaccia-Patella se demoraría menos que el Método Sexton-Swinbank en ejecutar. Por lo tanto, podemos concluir con confianza que nuestra hipótesis se cumplió en este estudio debido a todas las operaciones que debe de hacer Sexton-Swinbank.