Universidad Nacional de San Agustín

Estructura de Datos Avanzados
Tema: Laboratorio 06

Docente:
FRANKLIN LUIS ANTONIO CRUZ GAMERO

Realizado por:
Edwar Jhoel Vargas Herhuay
Florez Aguilar Fabian Vladimir
Luis Angel BustamenteTorres

Arequipa - Perú
2021

1. Introducción

En este trabajo se implementará un R-tree en C++. Los árboles R-Tree son usados para indexar información multidimensional como coordenadas geográficas, rectángulos o polígonos. El árbol R fue propuesto por Antonin Guttman en 1984 y se ha utilizado de forma significativa tanto en contextos teóricos como aplicados.

2. Marco Teórico

a. Definición

Los árboles M- Tree son estructuras de datos en forma de árbol similares a los árboles R y a los árboles B. Se construyen utilizando una métrica y se basan en la desigualdad del triángulo para realizar consultas eficientes sobre el rango y el vecino más cercano (k-NN).
Aunque los árboles M pueden funcionar bien en muchas condiciones, el árbol también puede tener un gran solapamiento y no existe una estrategia clara sobre la mejor manera de evitarlo.
Además, sólo puede utilizarse para funciones de distancia que satisfagan la desigualdad del triángulo, mientras que muchas funciones de disimilitud avanzadas utilizadas en la recuperación de información no satisfacen esta.

3. Metodología

- Algoritmo implementado en C++

4. Implementación

a. Algoritmo

Un árbol M tiene estos componentes y subcomponentes:

1. Nodos no hoja
    - Un conjunto de objetos de enrutamiento NRO.
    - Puntero al objeto padre del nodo Op.
2. Nodos hoja
    - Un conjunto de objetos N_O.
    - Puntero al objeto padre del nodo Op.

3. Objeto de enrutamiento
    - (Valor de característica del) objeto de enrutamiento Or.
    - Radio de cobertura r(Or).
    - Puntero al árbol de cobertura T(Or).
    - Distancia de Or a su objeto padre d(Or,P(Or))

4. Objeto
    - (Valor de característica del) objeto Oj.
    - Identificador del objeto oid(Oj).
    - Distancia de Oj a su objeto padre d(Oj,P(Oj))

INSERT

La idea principal es encontrar primero un nodo hoja N al que pertenezca el nuevo objeto O. Si N no está lleno entonces simplemente se adjunta a N. Si N está lleno entonces se invoca un método para dividir N. Si N no está lleno, simplemente se adjunta a N. Si N está lleno, se invoca un método para dividir N. El algoritmo es el siguiente:

Algorithm Insert
Input: Node N of M-Tree MT, Entry O_n
Output: A new instance of MT containing all entries in original MT plus O_n

```
N_e ← N's routing objects or objects
if N is not a leaf then
{
     /* Look for entries that the new object fits into */
     let N_in be routing objects from N_e's set of routing objects N_RO such that d(O_r, O_n) ≤ r(O_r)
     if N_in is not empty then
     {
        /* If there are one or more entry, then look for an entry such that is closer to the new object */
        O*_r ← min_{O_r ∈ N_in} d(O_r, O_n)
     }
     else
     {
        /* If there are no such entry, then look for an object with minimal distance from */
        /* its covering radius's edge to the new object */
        O*_r ← min_{O_r ∈ N_in} d(O_r, O_n) − r(O_r)
        /* Upgrade the new radii of the entry */
        r(O*_r) ← d(O*_r, O_n)
     }
     /* Continue inserting in the next level */
     return insert(T(O*_r), O_n);
else
{
     /* If the node has capacity then just insert the new object */
     if N is not full then
     { store(N, O_n) }
     /* The node is at full capacity, then it is needed to do a new split in this level */
     else
     { split(N, O_n) }
}
```

SPLIT

Si el método de división llega a la raíz del árbol, entonces elige dos objetos de enrutamiento de N, y crea dos nuevos nodos que contienen todos los objetos de N original, y los almacena en la nueva raíz. Si el método de división llega a un nodo N que no es la raíz del árbol, el método elige dos nuevos objetos de enrutamiento de N, reorganiza cada objeto de enrutamiento en N en dos nuevos nodos N1 y N2, y almacena estos nuevos nodos en el nodo padre Np de N original. La división debe repetirse si Np no tiene suficiente capacidad para almacenar N2 . El algoritmo es el siguiente:

```
Algorithm Split
  Input: Node N of M-Tree MT, Entry O_n
  Output: A new instance of MT containing a new partition.
```

```
/* The new routing objects are now all those in the node plus the new routing object */
let be NN entries of N ∪ O
if N is not the root then
{
   /*Get the parent node and the parent routing object*/
   let O_p be the parent routing object of N
   let N_p be the parent node of N
}
/* This node will contain part of the objects of the node to be split */
Create a new node N'
/* Promote two routing objects from the node to be split, to be new routing objects */
Create new objects O_p1 and O_p2.
Promote(N, O_p1, O_p2)
/* Choose which objects from the node being split will act as new routing objects */
Partition(N, O_p1, O_p2, N_1, N_2)
/* Store entries in each new routing object */
Store N_1's entries in N and N_2's entries in N'
if N is the current root then
{
    /* Create a new node and set it as new root and store the new routing objects */
    Create a new root node N_p
    Store O_p1 and O_p2 in N_p
}
else
{
    /* Now use the parent routing object to store one of the new objects */
    Replace entry O_p with entry O_p1 in N_p
    if N_p is no full then
    {
        /* The second routing object is stored in the parent only if it has free capacity */
        Store O_p2 in N_p
    }
    else
    {
         /*If there is no free capacity then split the level up*/
         split(N_p, O_p2)
    }
}
```

RANGO DE CONSULTAS

Una consulta de rango es aquella en la que se especifica un valor de similitud mínima/distancia máxima. Para un objeto de consulta dado Q e D y una distancia de búsqueda máxima r(Q), la consulta de rango range(Q, r(Q)) selecciona todos los objetos indexados Oj tales que d(Oj,Q) <= r(Q)
El algoritmo comienza en el nodo raíz y recorre recursivamente todos los caminos que no pueden ser excluidos de llevar a los objetos calificados.

Algorithm RangeSearch
Input: Node N of M-Tree MT, Q: query object, $r(Q)$: search radius

Output: all the DB objects such that $d(Oj, Q) \leq r(Q)$

```
{
  let O_p be the parent object of node N;

  if N is not a leaf then {
    for each entry(O_r) in N do {
          if |d(O_p, Q) − d(O_r, O_p)| ≤ r(Q) + r(O_r) then {
            Compute d(O_r, Q);
            if d(O_r, Q) ≤ r(Q) + r(O_r) then
              RangeSearch(*ptr(T(O_r)),Q,r(Q));
          }
    }
  }
  else {
    for each entry(O_j) in N do {
          if |d(O_p, Q) − d(O_j, O_p)| ≤ r(Q) then {
            Compute d(O_j, Q);
            if d(O_j, Q) ≤ r(Q) then
              add oid(O_j) to the result;
          }
    }
  }
}
```

5. Actividades

a. Implemente un M-Tree considerando los procedimientos de inserción y búsqueda.
Función insertar

```cpp
void Mtree::addObject(Embedding embedding) {
    size++;
    root->addObject(embedding);
}
```

Función Búsqueda por rango

```cpp
vector<Embedding> ConsultaRango(Node * N, Embedding embedding, float range){
    vector<Embedding> results;
    Entry * Op = N->parentEntry;
    if (!N->isRoot()) {
        if (!N->isLeaf) {
            for (Entry Or : N->entries) {
                if (abs( distance(*(Op->embedding), embedding) - Or.distanceToParent ) <= range + Or.radius) {
                    if (distance(*(Or.embedding), embedding) <= range + Or.radius) {
                        vector<Embedding> tempResults = ConsultaRango(Or.subTree, embedding, range);
                        results.insert(results.end(), tempResults.begin(), tempResults.end());
                    }
                }
            }
        } else {
            for (Entry Oj : N->entries) {
                if (abs( distance(*(Op->embedding), embedding) - Oj.distanceToParent ) <= range) {
                    if (distance(*(Oj.embedding), embedding) <= range) {
                        results.push_back(*(Oj.embedding));
                    }
                }
            }
        }
    } else {
        if (!N->isLeaf) {
            for (auto Or : N->entries) {
                if (distance(*(Or.embedding), embedding) <= range + Or.radius) {
                    vector<Embedding> tempResults = ConsultaRango(Or.subTree, embedding, range);
                    results.insert(results.end(), tempResults.begin(), tempResults.end());
                }
            }
        } else {
            for (auto Oj : N->entries) {
                if (distance(*(Oj.embedding), embedding) <= range) {
                    results.push_back(*(Oj.embedding));
                }
            }
        }
    }
    return results;
}
```

b. Cual es el país más próximo de Hungary. (mostrar la búsqueda ejecutada y el resultado en el lenguaje de programación escogido)

```
Address internal nodo 0x5600b49afa40
no parent
Peru (11.3137,-1)   -   Slovenia (14.1421,-1)   -


Address internal nodo 0x5600b49aeb00
parent: Peru(11.3137,-1)
Peru (2.82843,0)   -   Montenegro (2.82843,4.24264)   -   Czechia (2.82843,8.48528)   -


Address internal nodo 0x5600b49af780
parent: Slovenia(14.1421,-1)
Gibraltar (2.82843,11.3137)   -   Croatia (2.82843,7.07107)   -   Slovenia (2.82843,0)   -


Address leaf nodo 0x5600b49ade70
parent: Peru(2.82843,-1)
Peru (-1,0)   -   Bulgaria (-1,1.41421)   -   Bosnia_and_Herzegovina (-1,2.82843)   -
```

```
parent: Montenegro(2.82843,-1)
Montenegro (-1,0)   -   North_Macedonia (-1,1.41421)   -   Hungary (-1,2.82843)   -


Address leaf nodo 0x55eebdb11ce0
parent: Czechia(2.82843,-1)
Czechia (-1,0)   -   Georgia (-1,1.41421)   -   Romania (-1,2.82843)   -


Address leaf nodo 0x55eebdb11fd0
parent: Gibraltar(2.82843,-1)
Gibraltar (-1,0)   -   Brazil (-1,1.41421)   -   San_Marino (-1,2.82843)   -


Address leaf nodo 0x55eebdb12290
parent: Croatia(2.82843,-1)
Croatia (-1,0)   -   Slovakia (-1,1.41421)   -   Argentina (-1,2.82843)   -


Address leaf nodo 0x55eebdb12620
parent: Slovenia(2.82843,-1)
Armenia (-1,2.82843)   -   Lithuania (-1,1.41421)   -   Slovenia (-1,0)   -   Colombia (-1,1.41421)   -   USA (-1,2.8
2843)   -



ConsultaRango:
[ 1 2 ]
[ 2 3 ]
[ 3 4 ]
[ 4 5 ]


ConsultaRango Radio:
Bosnia_and_Herzegovina   Bulgaria   Czechia   Montenegro   Peru
```

c. Utilice el conjunto de datos proporcionado en la tabla 2 para determinar si existe influencia del orden de inserción de los datos en el resultado de la búsquedas en términos de los nodos recorridos para generar la respuesta.

```
20
USA 148104 2406
Slovenia 202419 2512
Slovakia 124480 2639
San_Marino 175688 2733
Romania 93390 2966
Peru 66477 5983
North_Macedonia 103485 3639
Montenegro 250528 3673
Lithuania 176235 2525
Hungary 114600 3586
Gibraltar 215221 2910
Georgia 212561 3030
Czechia 200245 3080
Croatia 149455 2678
Colombia 98156 2489
Bulgaria 101106 4139
Brazil 102912 2863
Bosnia_and_Herzegovina 84584 3870
Armenia 113938 2547
Argentina 116439 2547
Root: 0x559e79c3fc10
```

```
Address internal nodo 0x559e79c3ea70
parent: USA(11.3137,-1)
USA (2.82843,0)   -    San_Marino (2.82843,4.24264)   -    North_Macedonia (2
.82843,8.48528)   -


Address internal nodo 0x559e79c3f8f0
parent: Bosnia_and_Herzegovina(14.1421,-1)
Hungary (2.82843,11.3137)   -    Czechia (2.82843,7.07107)   -    Bosnia_and_
Herzegovina (2.82843,0)   -


Address leaf nodo 0x559e79c3de70
parent: USA(2.82843,-1)
USA (-1,0)   -    Slovenia (-1,1.41421)   -    Slovakia (-1,2.82843)   -


Address leaf nodo 0x559e79c3e640
parent: San_Marino(2.82843,-1)
San_Marino (-1,0)   -    Romania (-1,1.41421)   -    Peru (-1,2.82843)   -
```

```
Address leaf nodo 0x559e79c3ec50
parent: North_Macedonia(2.82843,-1)
North_Macedonia (-1,0)   -    Montenegro (-1,1.41421)   -    Lithuania (-1,2.
82843)   -


Address leaf nodo 0x559e79c3ef40
parent: Hungary(2.82843,-1)
Hungary (-1,0)   -    Gibraltar (-1,1.41421)   -    Georgia (-1,2.82843)   -



Address leaf nodo 0x559e79c3f200
parent: Czechia(2.82843,-1)
Czechia (-1,0)   -    Croatia (-1,1.41421)   -    Colombia (-1,2.82843)   -



Address leaf nodo 0x559e79c3f6d0
parent: Bosnia_and_Herzegovina(2.82843,-1)
Bulgaria (-1,2.82843)   -    Brazil (-1,1.41421)   -    Bosnia_and_Herzegovin
a (-1,0)   -    Armenia (-1,1.41421)   -    Argentina (-1,2.82843)   -
```

```
ConsultaRango:
[ 1 2  ]
[ 2 3  ]
[ 3 4  ]
[ 4 5  ]


ConsultaRango Radio:
North_Macedonia    San_Marino    Slovakia    Slovenia    USA
```

6. Conclusión

Dentro del procedimiento clásico de construcción del árbol M se utilizó un procedimiento de clustering personalizado cuando era necesario dividir un nodo. El procedimiento de clustering está diseñado de forma que produce una agrupación óptima de los estudiantes en relación con las "distancias" de conocimiento entre ellos.

REFERENCIAS

Wikipedia contributors.. *M-tree*. Wikipedia. https://en.wikipedia.org/wiki/M-tree

The M-tree Project. http://www-db.deis.unibo.it/Mtree/

Using M Tree data structure as unsupervised classification method. - Free Online Library. https://www.thefreelibrary.com/Using+M+Tree+data+structure+as+unsupervised+classification+method-a0298056451

M-tree Implementation. s. R http://www-db.deis.unibo.it/Mtree/impl.html