Pràctica 3 - CS:LS

Arbres

Adrià Pajares adrian.pajares
Lídia Figueras lidia.figueras
Martí Ejarque marti.ejarque
Victor Xirau victor.xirau
Enginyeria Informàtica
La Salle, URL

Índex

1 Explicació detallada dels algoritmes

1.1 Arbres BST

Al fer la cerca d'un objecte pel preu a la botiga hem implementat l'arbre BST. De manera que introduint un preu o un nom, aquest farà la cerca de l'objecte corresponent. Per tal de crear l'arbre, s'ha fet ús d'una funció anomenada "inserir", la qual funciona de la següent manera: Per saber a quina posició s'ha d'inserir el node, aquest compara el seu preu amb el de l'arrel de l'arbre. Depenent de si aquest preu és més petit o més gran, es col·locarà a l'esquerra o la dreta d'aquest. Si l'arrel no presenta cap fill dret ni esquerre, aquest node és directament inserit. Però, en el cas de que ja presenti algun fill, es torna a cridar la funció "inserir" recursivament, per mirar a quina posició del fill de l'arrel s'ha de col·locar. Per tal d'eliminar un node de l'arbre, s'ha fet ús d'una funció anomenada "deleteKey", la qual funciona de la següent manera: Aquesta funció rep el node a eliminar, i aquest és passat per paràmetre a una funció anomenada deleteRec, juntament amb l'arrel. Quan s'ha eliminat un node, es resta una unitat de la nostre variable "totalNodes". La funció deleteRec s'encarregarà d'esborrar el node de l'arbre recursivament. Primer rebrà l'arrel i el valor del preu del node que volem esborrar. Es compararà aquest valor amb el de l'arrel i es tornarà a cridar de forma recursiva en funció de si el valor del preu és més gran o més petit que el de l'arrel. D'aquesta manera, s'anirà cridant de forma recursiva a la funció, fins que el node trobat tingui un valor de preu igual al valor que li hem passat. En aquest cas, voldrà dir que hem trobat el node que volem esborrar, i aquest serà esborrat, tot tornant l'arrel actual. Finalment, si el node a esborrar tenia fills, s'haurà de seleccionar un per tal de substituir al pare; fent ús de la funció minValue, la qual va buscant a través dels fills esquerres del fill dret del node esborrat, fins trobar un valor adequat. Per últim, s'ha implementat la funció "inordre", el qual en aquesta s'ha creat una variable anomenada result, la qual guarda l'ordre dels nodes en els que s'ha visitat. Primer recorrent el subarbre esquerre, i seguidament el subarbre dret.

1.2 Arbres R

El plantejament per realitzar aquest arbre ha seguit el següent procés: Primerament s'han descarregat tots els nodes del Json, per tal de tenir-los en un array. Un cop s'ha creat aquest array del tipus ObjectesMapa, procedim a la creació de l'arbre amb aquests objectes.

Abans de veure el procediment de creació, cla mencionar que l'arbre estarà guardat en un array del tipus RectanglePare, on aquesta classe té una variable que representa el teu pare, és a dir, l'array de rectangles que t'inclou, un array de nodes, en el cas que tinguis nodes com a fills, un array de pares, en el cas que tinguis pares com a fills, i coordenades, per saber com és de gran aquest rectangle en qüestió.

Un cop vist aquesta clase, procedim a la creació de l'arbre. A continuació, assignarem a cada node un pare, el qual serà ell mateix. Després de fer que tots els nodes (els objectes) tinguin un pare (ells mateixos), començarà el procediment d'unió entre objectes. Buscarem per cada pare de cada node un altre pare (el que sigui més proper) i els unirem, per tal de formar grups de dos pares. En cas de que la quantitat d'objectes que tinguem sigui parell, tots els objectes (els nodes) agrupats en grups de dos, i procedirem a formar grups més grans.

En el cas que els nodes siguin imparells, tindrem tot parelles menys un node. Doncs el que farem serà buscar de quins rectangles que hem creat (que tenen dos nodes, per tant és un rectangle nou més gran) és el més pròxim a aquest node solitari. Un cop s'ha trobat aquesta parella més propera, li afegim aquest node i creem un array de tres nodes. Això ens genera un seguit de parelles (del tamany de l'array d'objectes tende menys un) i un trio.

Un cop tenim els pares agrupats per parelles, i un trio si es dóna el cas, comencem a ajuntar aquests grups, fins que ens queden com a màxim 3 agrupacions. Aquest procés es fa igual que amb nodes, amb l'única diferència que ara estarem buscant pares més propers entre si. Ens trobarem amb la mateixa casuística, i és que si tenim pares (agrupacions de nodes) imparells, s'aniran agrupant per parelles i quedarà un pare sol. Aquest cas es soluciona com hem vist anteriorment, és a dir, buscant l'agrupació de pares més propera al pare solitari.

Aquest procés s'anirà repetint fins que les agrupacions d'agrupacions de pares quedin com a molt de mida 3 i com a mínim de mida 2. Un cop s'ha completat aquest procés, se li diu a aquesta agrupació resultant que té com a pare una variable anomenada root, que és el RectanglePare que ho engloba tot. Aquest rectangle pare serà el que utilitzarem per explorar tot l'arbre en el moment que vulguem trobar els punts.

Un cop esta l'arbre muntat, per tal de trobar si en el punt que ens passa l'usuari es troba un objecte, anem recorrent agrupació a agrupació, fins trobar un objecte que contingui el punt

trobat. Si l'ha trobat, fem el hit i s'elimina, sinó fem miss i tornem a demanar un punt Un cop l'arbre ha estat creat, haurem de comprovar si el punt introduït per l'usuari ha fet un HIT o no. El que farem per tal de detectar això és recórrer totes les estructures de rectangles, fins arribar al rectangle que conté el punt introduït.

Per tal d'eliminar l'objecte trobat, el que es farà és eliminar l'objecte del conjunt d'objectes que tenim inicialment i es tornarà a crear l'arbre per tal de que els nodes estiguin agrupats de forma correcta. Si el HIT trobés més d'un objecte solapat, s'eliminarà el primer objecte que es trobi al recórrer l'array d'objectes.

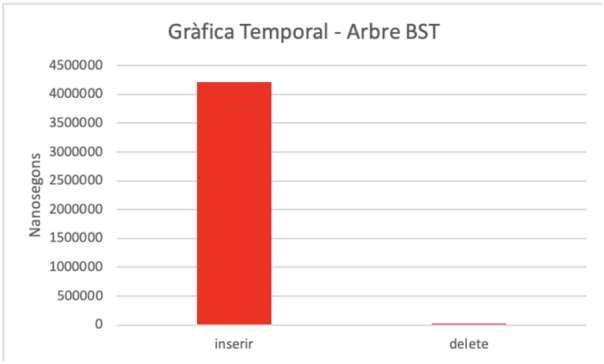
2 Mètode de proves utilitzat

Per tal de poder trobar els errors que anaven apareixent en el codi, hem emprat el debugger que ens ofereix IntelliJ IDEA. Ja que hem pogut anar seguint el codi linia a linia, tot mirant com canvien els valors de les nostres variables. Per exemple, ha estat molt útil en el cas de trobar si els nodes s'havien inserit de forma correcta a l'arbre.

També s'han utilitzat diferents datasets, sobretot per l'arbre R, ja que volíem comprovar el correcte funcionament del codi si el nombre d'objectes era tant parell com imparell. Això ens ha ajudat a detectar varies errades a l'hora de considerar un número imparell de nodes.

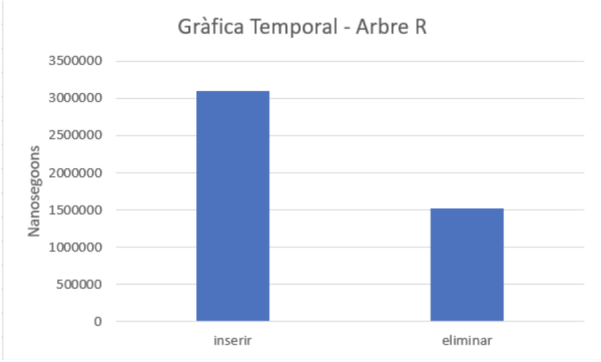
3 Comportament dels algoritmes

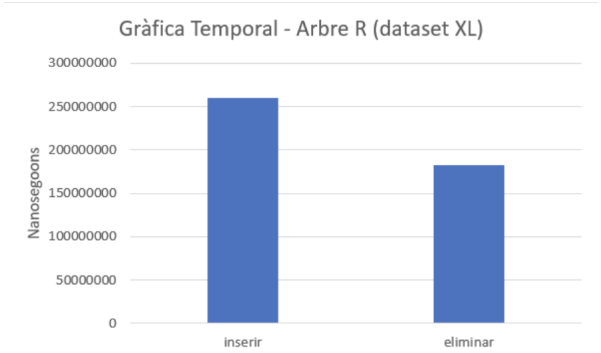
3.1 Arbre BST



Com podem comprovar, el temps d'execució de la funció inserir és molt més alt que el del temps de la funció delete. Com és lògic, ja que per tal d'inserir s'ha de recórrer tot l'arbre per tal d'estudiar a on s'ha d'inserir. En canvi, per tal d'eliminar només s'ha de fer una cerca fins a trobar el node que volem eliminar, però no es fa una cerca total, és a dir, no es recorre tot l'arbre.

3.2 Arbre R



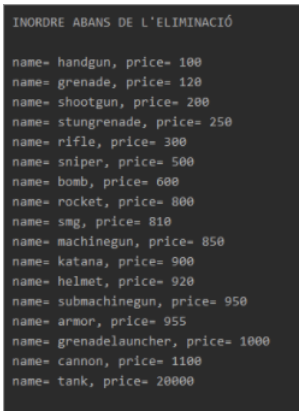


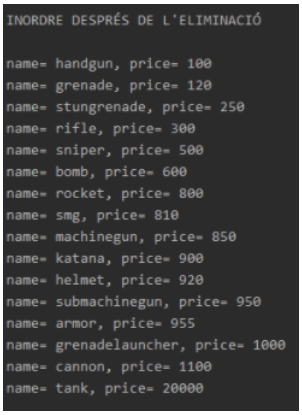
En aquest cas, observem que el temps d'inserció és aproximadament del doble que del d'eliminació. Aquest temps variarà en funció del que es trigui en trobar el node, i quant més temps es trigui a trobar el node a eliminar, més trigarà la funció d'eliminació. A les captures observem que això es compleix tant el dataset proporcionat a la pràctica com el dataset creat per nosaltres, el qual contenia 457 nodes.

4 Anàlisis dels resultats

Com podem comprovar a les següents captures, tant el procés d'eliminació d'un objecte mitjançant l'arbre BST amb un arbre R funcionen de manera correcta.

4.1 Eliminació mitjançant arbres BST





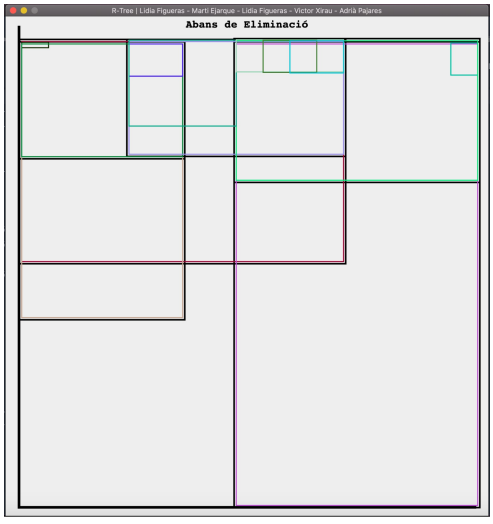
Com podem comprovar, en la primera imatge observem tot l'arbre desde l'arrel. Per tal de crear aquest arbre s'ha dut a terme un procés d'inserció de nodes, tal i com hem explicat anteriorment. Després de buscar l'objecte per nom "shotgun", podem comprovar com en la imatge 2, aquest objecte ha estat eliminat correctament. Pel que podem deduir, que la funció inserir i delete funcionen de forma correcta.

4.2 Crear un objecte al mapa (Arbre R)

Com es pot observar, al introduir un punt al nostre programa, es recorrerà el nostre arbre R per tal de comprovar si hem fet HIT o MISS. En el primer exemple, podem observar que el punt introduït ha fet HIT i, per tant, ha trobat un objecte a eliminar del nostre arbre. Si observem l'arbre abans i després, podrem comprovar que efectivament s'ha eliminat l'objecte. El segon exemple mostra com el punt que hem introduït no ha tocat cap objecte, i per tant la llista d'elements mostrada no presenta cap modificació.

4.3 Abans d'eliminació

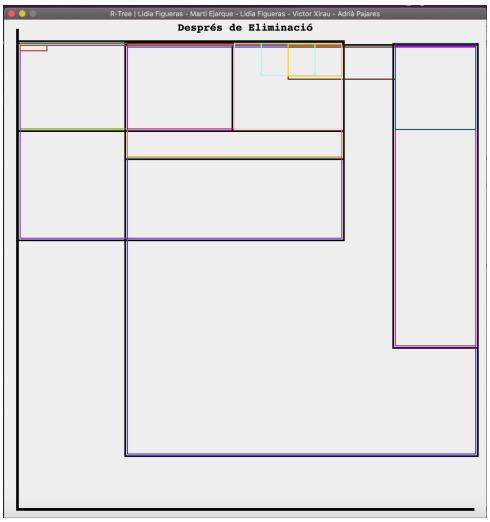
```
------------------------ R-TREE -----------------------------
        Pare 1
        Fills:
        Pare 2
            id: 1 - X1: 0 - Y1:800 - X2:100 - Y2:700
            id: 10 - X1: 200 - Y1:900 - X2:400 - Y2:700
            id: 12 - X1: 450 - Y1:800 - X2:550 - Y2:700
        Pare 3
            id: 8 - X1: 200 - Y1:550 - X2:500 - Y2:450
            id: 13 - X1: 500 - Y1:700 - X2:600 - Y2:600
        Pare 4
        Fills:
        Pare 5
            id: 2 - X1: 0 - Y1:550 - X2:50 - Y2:450
            id: 3 - X1: 0 - Y1:350 - X2:50 - Y2:300
        Pare 6
            id: 4 - X1: 0 - Y1:250 - X2:50 - Y2:200
            id: 5 - X1: 200 - Y1:100 - X2:300 - Y2:0
        Pare 7
        Fills:
        Pare 8
            id: 6 - X1: 500 - Y1:100 - X2:700 - Y2:0
            id: 7 - X1: 800 - Y1:300 - X2:850 - Y2:200
        Pare 9
            id: 9 - X1: 700 - Y1:800 - X2:850 - Y2:600
            id: 11 - X1: 400 - Y1:900 - X2:600 - Y2:800
------------------------------------------------------------
```



4.4 Després d'eliminació

```
S'ha eliminat l'objecte: 4

------------------------- R-TREE -----------------------------
        Pare 1
        Fills:
        Pare 2
            id: 1 - X1: 0 - Y1:800 - X2:100 - Y2:700
            id: 10 - X1: 200 - Y1:900 - X2:400 - Y2:700
        Pare 3
            id: 11 - X1: 400 - Y1:900 - X2:600 - Y2:800
            id: 12 - X1: 450 - Y1:800 - X2:550 - Y2:700
        Pare 4
        Fills:
        Pare 5
            id: 2 - X1: 0 - Y1:550 - X2:50 - Y2:450
            id: 3 - X1: 0 - Y1:350 - X2:50 - Y2:300
        Pare 6
            id: 8 - X1: 200 - Y1:550 - X2:500 - Y2:450
            id: 13 - X1: 500 - Y1:700 - X2:600 - Y2:600
        Pare 7
        Fills:
        Pare 8
            id: 5 - X1: 200 - Y1:100 - X2:300 - Y2:0
            id: 6 - X1: 500 - Y1:100 - X2:700 - Y2:0
        Pare 9
            id: 7 - X1: 800 - Y1:300 - X2:850 - Y2:200
            id: 9 - X1: 700 - Y1:800 - X2:850 - Y2:600
-----------------------------------------------------------------
```



5 Problemes observats

Respecte l'arbre BST, no ens ha suposat un gran repte, si més no ha estat dificultós a l'hora d'interpretar el pseudo-codi dels apunts i pasar-ho a llenguatge Java. Un cop s'ha tingut l'arbre muntat i s'ha procedit a l'eliminació, ens vam adonar que només havíem eliminat el nom de l'objecte, i no pas el node complert. Per tant el preu dels productes es canviava entre productes. En quant a l'arbre R, el principal problema ha estat raonar la implementació d'aquest, ja que l'ús de rectangles i punts feia el codi més tediós del normal. Després de raonar la implementació, hem trobat diverses dificultats per tal de poder crear les diferents agrupacions de rectangles, ja que hem hagut de controlar moltes variables per tal de poder generalitzar el codi per poder obtenir rectangles cada cop més grans i que agrupéssin més rectangles. Trobar el rectangle adequat per crear els grups més òptims també ha estat difícil, ja que, tal i com hem comentat a l'apartat d'explicació del codi, hem hagut de treballar amb les àrees i distàncies entre els rectangles i els punts que els formaven.

6 Conclusions

L'objectiu d'aquesta pràctica ha estat, mitjançant la implementació d'un arbre BST i un arbre R, implementar algorismes de cerca d'objectes en funció d'uns criteris. Cal destacar que la realització d'aquesta pràctica ha estat satisfactòria,i els conceptes clau sobre arbres han quedat molt clars degut a la implementació detallada d'aquests algorismes.

A l'arbre BST, trobem un cost asimptòtic de O(n) tant per inserir, eliminar i buscar un node, ja que hem de recórrer tots els elements de l'arbre per arribar a una posició. Aquesta implementació no presenta costs massa elevats.

Daltra banda, l'arbre R, en relació als costs asimptòtics, trobem que per la creació dels primers rectangles que envolten els objectes el cost asimptòtic és de O(n), ja que recorrem el conjunt d'ojectes un sol cop i anem atribuint directament els rectangles a cada objecte. A l'hora de crear els rectangles pare pels rectangles creats a cada objecte (la primera part de la lògica de creació de rectangles de la nostra implementació) i els rectangles generals, trobem un cost aproximat de O(n^2), ja que trobem els dos bucles principals per crear els pares i el while que controla si l'assignació és correcte. En cas que quedi un objecte O(n^2), per tal de trobar el rectangle òptim amb què s'hauria d'agrupar.

7 Bibliografia

Arbre BST

`https://es.wikipedia.org/wiki/`

Arbre BST

`https://www.geeksforgeeks.org/binary-search-tree-data-structure/`

Rtree

`https://github.com/davidmoten/rtree`

Rtree

`https://stackoverflow.com/questions/8456240/r-tree-implementation-java`

Rtree

`http://www.spf4j.org/jacoco/org.spf4j.base/RTree.java.html`

Rtree

`https://www.programcreek.com/java-api-examples/?code=gegy1000`