Instasalle

Programació avançada i estructures de dades

By

Alex Almansa (alex.almansa), Marc Llort (marc.llort), Paula Rodriguez (paula.rodriguez), Javier Gaig (javier.gaig)

Departament d'enginyeria

La Salle URL

Maig 2019

Contents

1 Funcionament del Sistema

La finalitat del programa és, que a partir de les diferents estructures de dades, les implementem, juntament amb els seus modes de inserció, eliminació i busqueda. D'aquesta manera, el programa ens permetrà importar les dades de dos fitxers JSON (users i posts) a les diferents estructures de dades.
Un cop importades, tindrem la possibilitat de afegir/borrar informació, visualitzar el estat de l'estructura i fer diferents tipus de cerques.
El programa també ofereix la possibilitat de exportar el estat actual de la estructura com a fitxers json, per posteriorment importar-los, i continuar des de l'estat guardat de l'estructura.

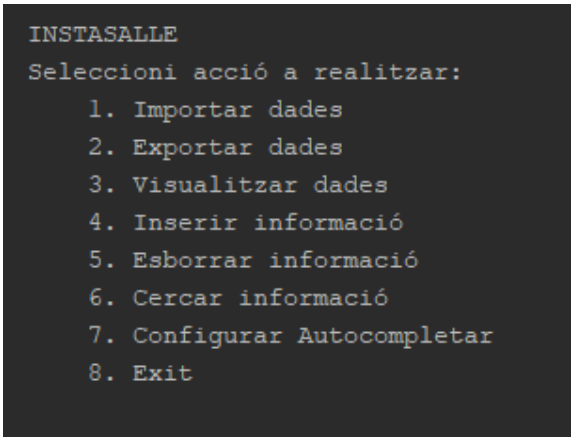


Figure 1: Menú inicial

1.1 Funció 1: Importació

Aquesta primera funcionalitat, tracta de, donats dos fitxer json de Users i Posts, importar-los a la estructura de dades seleccionada.
Un cop clickem a quina estructura de dades volem importar la informació, el programa ens demana la ruta on està el fitxer json. En cas de trobar el fitxer, fem una lectura del fitxer amb "gson".
Posteriorment, realitzem insercions de la informació del json a la estructura de dades, fent us de la funció de inserció que hem programat per cada una de les estructures.

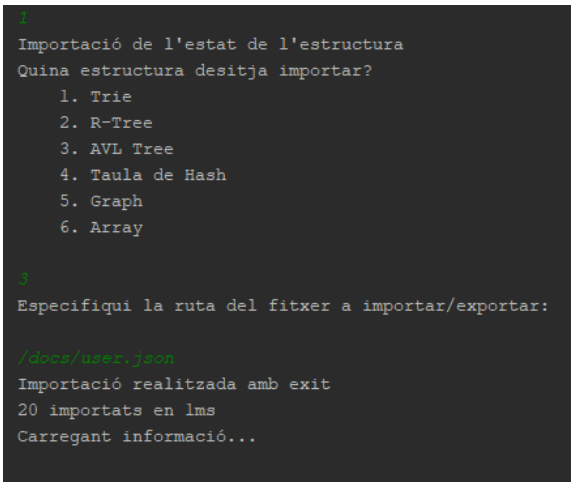


Figure 2: Menú importació

1.2 Funció 2: Exportació

De igual forma que amb la importació, el programa ens demanarà la ruta on volem exportar la estructura de dades. Un cop comprovat que es tracta d'una ruta correcte, començarem a fer una "visualització" de el estat de la estructura de dades, i a partir d'aquesta visualització anirem agafant el objecte de cada "node" i inserint-lo al fitxer json que anem creant.

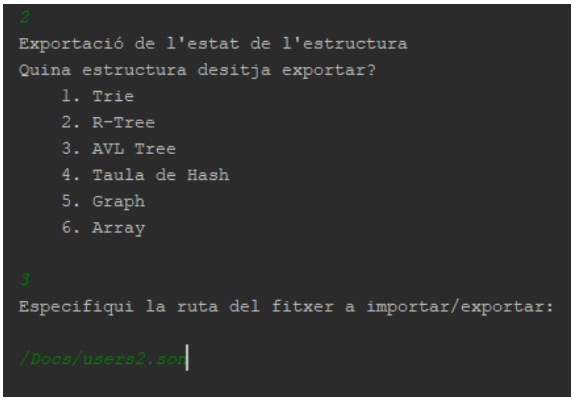


Figure 3: Menú exportació

1.3 Funció 3: Visualització

En aquest mode podrem veure el estat de les diferents estructures. Per dur a terme aquesta funcionalitat simplement anem iterant per les diferents estruc-

tures i printant el nom del objecte que emmagatzemen, sempre seguint certa lògica, segons quina estructura estiguem visualitzant. En el cas de RBT i AVL ho veurem en forma de in-ordre.

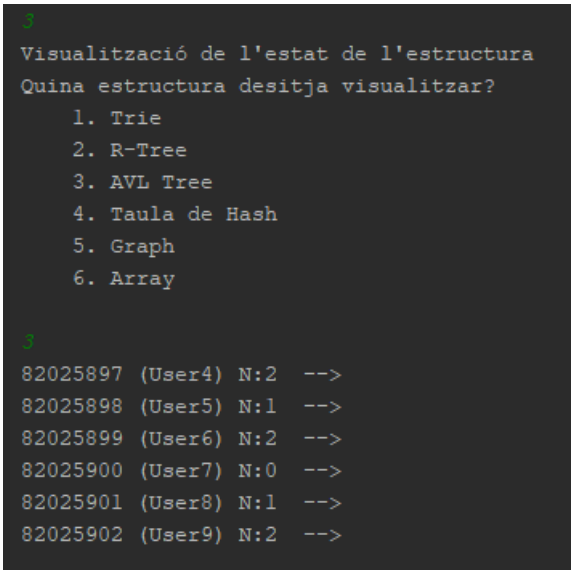


Figure 4: Visualització estructura

1.4 Funció 4: Inserir informació

Per dur a terme la inserció de informació (posts/users), tenim una classe anomenada Funcions, on tenim una funció generalitzada per fer la creació del objecte (user/post), que va demanant els diferents camps necessaris. Dins d'aquesta funció, tenim altres que són específiques realitzar diferents accions segons la estructura (checkUserExists, checkPostExists...).
Per tant, per dur a terme aquestes funcionalitats, necessitarem sempre passar a la funció en quina estructura estem treballant. Finalment aquesta funció ens retorna el objecte, el qual inserirem a la estructura que toca.

A les captures, podem veure el funcionament. Recalcar que com veiem a la figura 6, si ja hem afegit un usuari ens mostrarà el missatge de que ja ha estat inserit.

```
4
A quina estructura desitja realitzar la operció?
    1. Trie
    2. R-Tree
    3. AVL Tree
    4. Taula de Hash
    5. Graph
    6. Array

3
Inserció d'informació
Quin tipus d'informació vol inserir?
    1. Nou Usuari
    2. Nou Post
```

Figure 5: Inserir user o post

```
1
Nom d'usuari:
Test1
Node no trobat
Data creació: (yyyy-mm-dd)
2019-04-20
Usuaris que seguirà (Y/N]:
y
User3
Node no trobat
Usuaris que seguirà (Y/N]:
y
User7
User7 trobat
Usuaris que seguirà (Y/N]:
y
User7
User7 trobat
Usuari User7 ja seguit!
Usuaris que seguirà (Y/N]:
n
```

Figure 6: Menú inserir User

```
2
Id post:
23
Node no trobat
Data creació:
2012-02-02
Usuari del post:
User7
User7 trobat
Localització post X:
23
Localització post Y:
24
Hashtags (Y/N]:
y
#proval
Hashtags (Y/N]:
y
#proval
Hashtag: #proval ja afegit a la llista de hashtags!
Hashtags (Y/N]:
n
Usuaris que han donat like (Y/N]:
y
User8
User8 trobat
Usuaris que han donat like (Y/N]:
n
```

Figure 7: Menú inserir Post

```
2
Id post:
23
Node no trobat
Data creació:
2012-02-02
Usuari del post:
User3
Node no trobat
Usuari inexistent!
```

Figure 8: Error inserir post

1.5 Funció 5: Eliminar informació

De igual forma, tenim una funció a la classe Funcions per "estandarditzar" el procés de eliminació, on també fem us de les funcions de checkUserExists i checkPostExists específiques per cada estructura de dades.
El procés d'eliminar és molt més senzill que el de inserció, ja que només ens caldrà demanar el "username" en el cas dels usuaris i el "id" per els posts.

```
5
A quina estructura desitja realitzar la operció?
    1. Trie
    2. R-Tree
    3. AVL Tree
    4. Taula de Hash
    5. Graph
    6. Array

3
Eliminació d'informació
Quin tipus d'informació vol esborrar?
    1. Usuari
    2. Post
```

Figure 9: Menú eliminació

```
3
Eliminació d'informació
Quin tipus d'informació vol esborrar?
    1. Usuari
    2. Post
1
Nom d'usuari que s'esborrarà:
User7
Processant petició...
L'usuari [User7] s'ha esborrat correctament del sistema
```

Figure 10: Eliminació user

1.6 Funció 6: Buscar informació

Alhora de cercar un post o un usuari, tindrem diferents maneres de fer-ho. La més bàsica seria buscant segons el "username" o el id del post.
En el cas de buscar el username, cada cop que introduïm un tros del username, s'executa el algorisme de cerca de tries, i ens mostra algunes suggerències de usuaris, ordenades per el nombre de vegades que les hem utilitzat, així si busquem moltes vegades un usuari, ens apareixerà abans que el que no hem buscat mai. Si seleccionem alguna de les suggerències que apareixen,es carregarà l'usuari.
Un cop carreguem un usuari o post, cridem la funció mostraInformacio, la qual fa un seguit de "sout's" amb la informació pertinent.

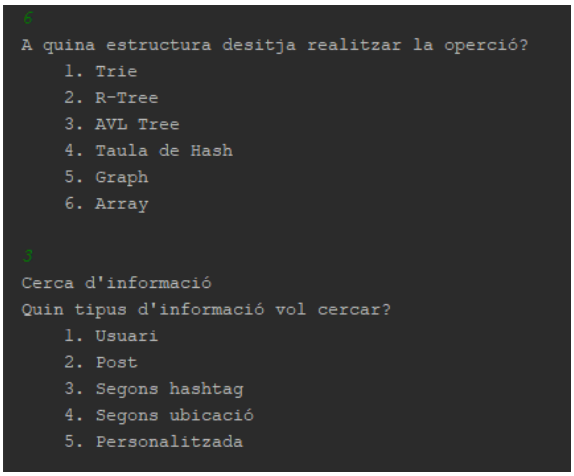


Figure 11: Menú cerca

1.7 Funció 7: Configuració Auto-completar

Per configurar el auto-completar, es dona l'opció de limitar el nombre màxim de paraules que es volen tenir als tries. D'aquesta manera, per defecte, no es carregaran tots els usuaris que hi ha, sino que es carregaran els n primers. En cas de voler reduir la mida dels usuaris, si la estructura esta plena, s'eliminaran primer les paraules que menys s'hagin utilitzat.

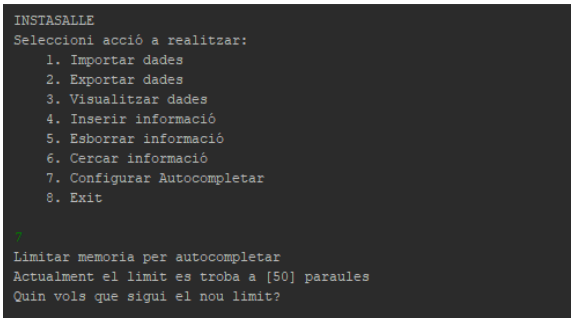


Figure 12: Menú configuració auto-completar

2 Estructures de dades

En aquest apartat ens centrarem en explicar cadascuna de les diferents estructures de dades, com les hem programat i quins són els seus avantatges i inconvenients.

2.1 Trie

Els tries son una estructura de dades basada en la idea dels arbres. La seva principal funcionalitat és la de emmagatzemar paraules. En aquesta estructura es guarda l'alfabet i es recuperen les paraules recorrent les diferents branques de l'arbre. és realment útil per funcions com la del auto-completar que és la que hem implementat en el nostre cas.

El primer que cal fer per arribar a aconseguir que els tries realitzin aquesta funció d'auto-completar és implementar els tries amb les seves funcionalitats bàsiques. És per això que s'explicarà com es va implementar els tries en primer lloc i després s'aniran introduint els canvis que es van realitzar per tal d'arribar a l'estructura de dades que hem utilitzat finalment.

Per començar, la estructura esta formada per uns nodes, que tenen un boolean per indicar si son final de paraula i un array de nodes que son els fills, que simbolitzen totes les lletres del abecedari. A partir d'aquests nodes tan senzills, es poden crear aquestes estructures. Les operacions bàsiques per utilitzar-les son:

Inserció Es parteix de un node pare, que té tants fills lletres hi ha al abecedari. Aquests fills inicialment existiran però estaran inicialitzats a null de manera que quan es vulgui crear la primera paraula, caldrà anar al fill que es trobi en la posició de la lletra per la que comença la paraula i crear un nou node. Per exemple si la paraula comença per 'a', caldrà anar al fill que es troba en la posició 1 i inicialitzar-lo. Un cop inicialitzat, aquest fill tindrà, igual que el pare, tants fills com lletres té l'abecedari. Si la següent lletra fos una 'b', des d'aquest node fill, caldria inicialitzar el seu segon fill per així tenir la combinació 'ab'.
Realitzant aquesta mateixa operació tantes vegades com lletres té la paraula, es generen les paraules. En el node final de cadascuna d'elles s'activa el boolean que indica que son final de paraula. Això s'utilitza perquè en cas que hi hagi dues paraules que estiguin una dins l'altre, les puguem detectar, com podrien ser 'el' i 'electricitat'.

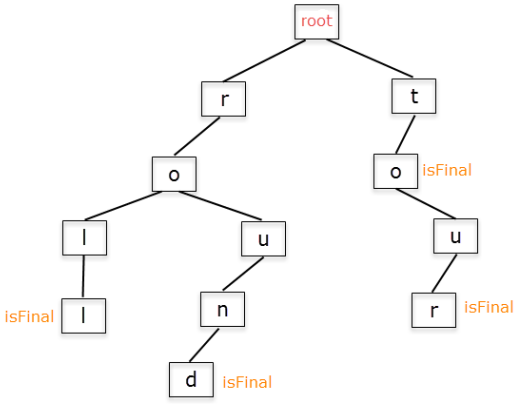


Figure 13: Exemple Trie

A aquesta imatge es pot observar l'estructura que segueixen els tries. A més a més estan indicats amb el text isFinal, els nodes que son final de paraula.

Eliminació Per dur a terme l'eliminació, el primer que cal fer és arribar al últim node de la paraula que es vol eliminar, és a dir l'ultima lletra. Un cop aquí, cal mirar si aquest node té més fills i en cas que no en tingui, es pot eliminar i pujar al seu node pare. Aquesta operació es va repetint fins que es troba algun node que té algun fill i per tant depèn d'una altre paraula o fins que s'arriba al node root.

Cerca Partint de l'estructura de tries, la cerca resulta bastant ràpida i fàcil de fer ja que per cercar una paraula, el que cal fer es anar node per node, mirant si el fill en la posició corresponent esta a null o esta inicialitzat. En el moment que s'arriba a la ultima lletra de la paraula sense trobar-se amb cap filla null, vol dir que s'ha trobat la paraula.

2.1.1 Implementació auto-completar

Ara que ja ha quedat clar el funcionament bàsic dels tries, s'explicaran les modificacions realitzades a aquesta estructura per aconseguir les funcions d'auto-completar. En primer lloc, una de les coses que s'ha canviat és el boolean que indica si es final de paraula, aquest boolean s'ha canviat per un int, que si es 0 indica que no és final de paraula i a més ens indica quantes vegades s'ha utilitzat aquesta paraula, per tal de donar les opcions més utilitzades per l'usuari primer.

Un altre canvi important té a veure amb la funció de cercar paraules. Aquesta funció ha quedat pràcticament igual però a part de buscar la paraula que ens han entrat, l'estructura busca tots els finals de paraula que hi ha a partir d'aquesta paraula i els retorna en un arraylist de paraules, ordenades segons el

nombre de vegades que han estat utilitzades. Per tal d'ordenar aquest arraylist, s'ha utilitzat un QuickSort, que és molt més ràpid que una ordenació normal.

L'ultim canvi important respecte als tries habituals, s'ha fet mentre es provava l'estructura, però a priori no estava previst. Aquest canvi ha estat ampliar el nombre de fills possibles a tots els possibles caràcters de la taula ascii. Això s'ha implementat degut a que els datasets per a realitzar les proves tenien números a buscar i l'estructura base dels tries no estava preparada per emmagatzemar números. Posant tots els caràcters de la taula ascii, tot i que també augmenta la memòria que ocupa la estructura, assegura poder emmagatzemar qualsevol nom que l'usuari vulgui emmagatzemar.

2.2 R-Tree

2.2.1 Estructures

La nostra estructura del RTree està formada per un conjunt de llistes que tenen com a número màxim de entrades 3(M), ja sigui un fill fulla o no, i que pot contenir elements de dos tipus: NodoRTree o un LeafNode.

1. NodoRtree: es tracta d'un node que no és fulla, és a dir, d'un element rectangle que engloba als seus fills que poden ser altres rectangles (un altre conjunt de NodoRtree), o un conjunt de nodes fulla que formarien els diferents (i com màxim 3) posts. La informació que emmagatzemen aquests nodes, a part dels seus fills RTree o fills Leaf, és la informació que referència al rectangle en si, és a dir, la seva àrea i el seu X i Y màxima i mínima (els seus límits).

2. NodoLeaf: es tracta dels nodes fulla, és a dir, els nodes en l'últim nivell de l'arbre, sense cap tipus de fill possible, i que referencien al Post en si i, mitjançant un booleà, si aquest està activat o no (per en un futur "eliminar-ho").

Aquests nodes poden pertànyer a un array anomenat NodeRTreeArray o a un LeafNodeArray respectivament, que en ambdós casos guarden el mateix tipus d'informació: un punter cap al que és el seu pare (el rectangle de menor grau / més proper que els engloba) i un punter cap a l'array del seu pare, és a dir, aquells (mínim 2 i com a màxim 3) rectangles que són englobats pel mateix NodoRTree pare.

2.2.2 Funcionament

Anotacions inserció:

1. Cada Post que inserim en el sistema el convertim, abans de res, en un nou NodoLeaf amb el seu booleà per comprovar si està actiu o no a "TRUE".

2. Cada pas en el que s'editen nodes (ja sigui insercions, splits o reestructuracions del root) es modifiquen els punters al pare, a l'array del pare i els fills segons convingui perquè així l'arbre no trenqui la seva estructura

3. Cada possible acció està representada en l'apartat de mètodes de prova realitzats amb un índex específic (representat d'aquesta manera "casX") perquè s'entengui de forma més fàcil.

Un cop aclarit això, quan es disposa a posar aquest nou element en l'estructura comprovem si l'arrel és un NodeRTreeArray o LeafNodeArray. En el cas que sigui l'última opció es mirarà si aquest té menys de 3 elements introduïts i, en cas positiu s'introduirà mentre que en cas negatiu (es vol introduir el 4t element) es realitzarà un Split per baix (s'explicarà a continuació, cas1). En cas que l'array arran de l'estructura sigui del tipus NodeRTreeArray, se segueixen els següents passos:

1. Com sabem que, en ser un array format per mínim 2 i màxim 3 nodes tipus NodoRTree, i no sabem la quantitat de rectangles que emmagatzema cada un d'aquests nodes, cridem a la funció findBestSplitNode. Aquesta s'encarrega, de forma recursiva i segons si el post a posar es troba dins del rectangle mirant, de trobar el NodoRTree amb fills tipus Leaf en el qual poder posar el nou Node (Post) creat.

2. Mirem si aquest nou post a posar en l'estructura ca com a fill del millor NodoRTree trobat, és a dir, si aquest té 2 o 3 fills. En cas que es pugui posar (tingui 2 fills, caso2), s'insereix el post en l'última posició, s'actualitza el rectangle (la seva àrea, X i Y màxima i mínima), és a dir, el NodoRTree pare del LeafNodeArray al que hem posat aquest node, i diríem a l'funció actualizaRoot. Aquesta consisteix en, partint del NodeRTreeArray del millor NodoRTree que s'acaba d'actualitzar, anar restablint recursivament els valors d'àrea, X i Y màxima i mínima dels nodes pares fins a arribar a l'arrel.

3. En cas que l'estructura en la qual es va a posar el nou NodoLeaf (el post) estigui plena, ¡s'haurà de fer un Split! I per a això primer cridem a la funció lookForAnyEmptySpace, que mira des del node en el qual està (el millor NodeRTree en què inserir el Post) fins al node arrel, passant d'array pare en array pare, comprovant si en alguna posició d'aquests hi ha algun espai lliure. Retornant un "TRUE" en cas que trobi algun i un "FALSE" em cas contrari.

4. Tot seguit es crida al procediment splitTree, que s'encarrega de la reestructuració de l'arbre segons el que li torni la funció esmentada anteriorment. Abans de res i mitjançant la funció getBestPoints, agafem els 4 NodosLeaf (dels 3 que ja estaven posats + el que hem de posar), i els ordenem de manera que els 2 primers i els 2 últims formen, independentment, dos rectangles d'àrea menor possible entre ells. Seguidament, i mitjançant la funció createNewRs, es creen els respectius rectangles amb els 4 punts anteriorment ordenats i formem un nou array amb aquests juntament amb els de l'array del pare del node del que hem fet Split.

5. Tot seguit, i en cas que no hi hagi cap espai lliure en el qual posar el Post, es faria un "Split per baix" (cas3)mitjançant la crida al procediment balanceRoot. Aquest s'encarrega, primer de tot i mitjançant la funció getLowerAreaNodes, d'agafar els 4 NodosRTree anteriorment creats (mitjançant createNewRs) i de ordenar-los de manera que els 2 primers i els 2 últims formen, independentment, dos rectangles d'àrea menor possible entre ells i posant, com els 2 primers, aquells la àrea formada és menor que la dels 2 següents. Tot seguit vam crear 2 nous rectangles amb el primer i segon parell de nodes anteriorment creats, ho afegim a un nou array, i ho substituïm per la matriu pare del node del que hem fet Split. Un cop realitzat aquest procés de "Split per baix" l'arbre queda desequilibrat, de manera que s'agafen els 2 nodes de l'arrel (no utilitzem el node arrel al

que se li ha fet Split), i vam crear un nou rectangle amb aquests afegint un nivell més i balancejant l'estructura.

6. En el cas que sí que hagués espai en algun dels possibles arrays pare del node del que hem fet Split (cas4), es realitza un bucle fins que s'introdueixi. Per a això ens agafem un array pare (que inicialment és el del node del que s'ha fet un Split) que anomenarem "arrayX" i es mira si té espai per introduir-se.

   (a) Si no es pot introduir, és a dir, la matriu ja té les 3 posicions ocupades, cridem a la funció getLowerAreaNodes (explicada anteriorment) i, els 2 primers rectangles (els d'àrea menor) es queden al seu lloc actualitzant el seu node (rectangle) pare, mentre que amb els 2 segons (els d'àrea major) es forma un nou rectangle amb ells i se li afegeix a un nou array juntament amb els de l'array del pare del arrayX. S'actualitza el arrayX posant com el seu array pare.

   (b) Si es pot introduir vol dir que aquest arrayX té espai, és a dir, que el seu array pare té únicament 2 nodes. Pel que l'única cosa que cal fer és canviar aquest array pel anteriorment creat al no poder-se introduir, és a dir, la matriu format pels nodes d'aquest arrayX juntament amb el node que s'ha anat pujant.

Cerca: Pel que fa la cerca, l'unic que s'ha de fer es crear-se un nou rectangle sent la seva latitud i longitud màxima i mínima respectivament la latitud i longitud que ens introdueixen +/- el radi. Un cop es té aquest rectangle es va mirant, des de l'arrel, si aquest està dins de cada un dels seus NodesRTree. En cas afirmatiu es mirará el sues fills repetint el procediment de forma recursiva fins que el array que s'estigui mirant sigui amb nodes de tipus NodeLeaf.

Eliminació: Mateix procediment que la cerca però únicament posant el radi a 0, és a dir, buscant una latitud i longitud específica (la del post). Una vegada es troba el post (el NodeLeaf), es posa el seu booleà que indica si està activat o no a "FALSE".

2.2.3 Costos i usos

El RTree és un molt bon algoritme d'ordenació pel que fa a relacions d'Objectes, és a dir, si s'ha de fer una ordenació la necessitat és la de gestionar dades de diferent o igual naturalesa segons algun tipus d'element en comú. Si per exemple es necessita fer una recerca de la qual després no es necessitarà únicament un Objecte en específic, sinó que es necessitarà el conjunt d'objectes del seu voltant, ja sigui trobar bars propers a la meva localització, indexació en bases de dades relacionals ... En cas d'una bona indexació el cost d'inserció a la estructura es logarítmic

2.3 AVL Tree

El arbre AVL, és un arbre de cerca binari pseudo-balancejat a partir de la diferència d'alçada entre el subarbre dret i esquerre del node inicial o pare.
Per dur a terme el auto-balancejat, es fa us de rotacions a partir del Balance Factor, del qual parlarem posteriorment. Els costos tant per inserir, eliminar o cercar són de O(log(n)).

Vam decidir fer us de AVL i no de RBT perquè tenint en compte de que es tracta d'una xarxa social, és realitzen moltes més cerques que no insercions. Cada cop que es mostra una imatge a un usuari, aquesta ha tingut que ser buscada. Com RBT és superior en insercions, i AVL en cerques vam decantar-nos per el segon.

A continuació passem a explicar cadascuna de les operacions de l'arbre AVL:

Rotacions Cada node té un balance factor, el qual podem calcular restant l'alçada entre el subarbre dret i esquerre. En cas de que aquest balance factor sigui superior de 1, caldrà aplicar algun dels següents casos per d'aquesta forma balancejar el arbre.

1. Cas LL:

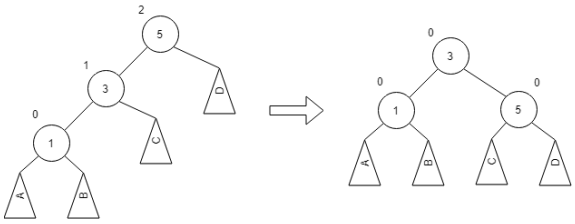


Figure 14: Cas LL

2. Cas RR:

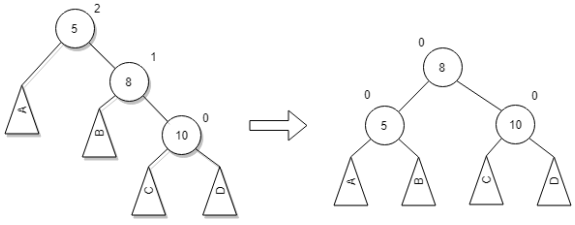


Figure 15: Cas RR

3. Cas LR:

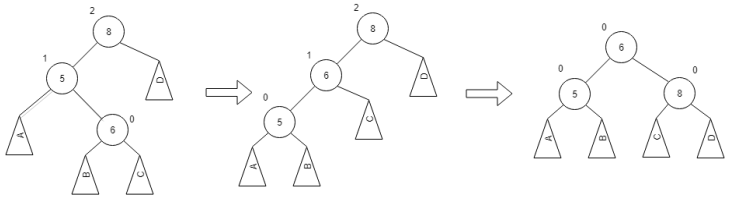


Figure 16: Cas LR

4. Cas RL:

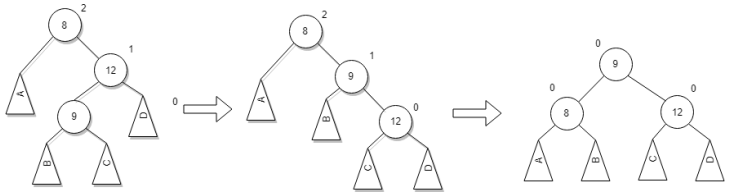


Figure 17: Cas RL

Inserció La inserció és idèntica a la de un arbre binari. Primer, buscarem el lloc on haurem de inserir el node. Anirem comparant el valor a inserir amb el del node on estem situats actualment. Si és menor, baixarem un nivell del arbre per el subarbre esquerre, si es major per el subarbre dret. Quan no puguem baixar més, degut a que ens trobem que el node fulla és "null", allà serà on inserirem el node. En cas de que trobéssim un node amb un valor igual al que nosaltres volem inserir, cal mostrar un error, degut a que no poden haver dos nodes amb el mateix identificador.
Però, de igual forma com veurem a la eliminació i cerca, segons la inserció que fem, el arbre ens pot quedar desbalancejat. En aquest cas ens caldrà aplicar rotacions per ajustar el arbre de forma que la diferencia/balance factor entre el subarbre dret i esquerre del node que estem tractant, sigui inferior a 2.

Eliminació Alhora de eliminar, seguirem el mateix procediment explicat a la inserció, per arribar a trobar el node a eliminar. En cas de no trobar un node que coincidís amb el que volem eliminar, mostraríem un missatge d'error indicant que aquell node no existeix al nostre arbre. En cas de trobar-lo, l'eliminarem. Quan eliminem un node ens podem trobar amb diferents casos:

1. Node no tenia fills:

Aquest és el cas més simple. Un cop borrat només caldria controlar els desbalancejos, com hem explicat anteriorment al apartat de rotacions.

2. Node tenia un fill:
   En cas de només tenir un fill, aquest substituirà al node borrat, es a dir, pujarà una posició, on es trobava el seu pare. Posteriorment caldrà tractar els desbalancejos.

3. Node tenia dos fills:
   Quan un node eliminat té dos fills, a la posició on es trobava el eliminat, posarem el node de menys valor del subarbre dret.Posteriorment caldrà tractar els desbalancejos.

Cerca Funcionament igual al de un BST. Anem comparant el valor que busquem, amb el del node en el que estem. En cas de que el valor sigui menor, continuarem per el subarbre esquerre, en el cas de que sigui major per el dret.
Arribarà un punt on ens trobarem amb el node que estàvem buscant, o un node "null". En cas de que sigui "null" voldrà dir que el valor/node que estem buscant no està emmagatzemat al nostre arbre.

2.4 Taula de Hash

Per realitzar la taula de Hash s'ha utilitzat una llista de màxim 50 elements tipus HashNode. Aquest està format per un enter que identifica el hashtag, una llista del conjunt de posts que utilitzen aquest hashtag, el hashtag en si, i un punter cap a un altre HashNode al qual anomenarem nextElement (que s'explicarà més endavant). Per similar al funcionament d'una taula de Hash, havíem de fer que cada paraula (la string en si), d'alguna manera, ens retornés la posició en la que està en la llista (en la nostra taula de hash). I és aquí on entra la funció hashCode, que ens retorna un identificador de 32 bits al qual li assignem el id del HashNode, és a dir, si tenim, per exemple, el sushi, el seu hashCode ens tornaria alguna cosa tal que 553.106.741, que seria l'identificador del HashNode del sushi. El problema és que no podem tenir una llista de més de 10e9 elements, és per això pel que posem aquest node en la posició resultant de realitzar el mòdul d'aquest HashCode amb el nombre de posicions màximes de la nostra taula de hash (50) . Amb això es solucionaria el problema de l'espai, però sorgeix un altre que és la possibilitat que dos strings (hashtags) diferents donin el mateix resultat, indexant en la mateixa posició de la taula de hash 2 strings diferents. I és aquí on entra el nostre nextElement, un punter d'un HashNode a un altre HashNode que indica que, encara que els dos pertanyen i estiguin en la mateixa fila de la taula de hash (posició de la llista), són independents entre ells (ja que no comparteixen ni ID ni és el mateix hashtag).

ANOTACIÓ: hem escollit la mida màxima de 50 posicions per a la taula de hash ja que, per al nostre mètode de proves utilitzat, és una mida en què no s'omple massa la taula, però hi ha solapacions de posicions, podent provar tots els casos possibles en els quals l'estructura pugui fallar. Les funcions d'Inserció i esborrat d'aquesta estructura són O (n) i O (log n) respectivament.

Inserció Per realitzar la inserció d'un post a la taula de hash, hem de recórrer tots els hashtags que té aquest post guardat i per cada un d'ells, afegir en la posició corresponent al hashtag a la taula de hash, el post. En el procés d'afegir aquest post, cal tenir en compte si la taula ja té un node que faci referència a aquest hashtag o no. En el cas que aquell hashtag encara no hagi estat utilitzat per cap altre post, es crea un nou node que faci referència al hashtag i s'afegeix el post; mentre que en el cas que el hashtag ja existeixi, simplement s'afegeix el post.

Eliminació Per dur a terme l'eliminació d'un post a la taula de hash, s'ha fet servir la mateixa base que a la inserció, només que al revés. Quan s'elimina un post, cal recórrer tots els hashtags que hagi del post, i un cop trobat el node en el qual es troba el hashtag a la taula de hash, es procedeix a eliminar el post. Es repeteix aquest procediment fins que s'hagin recorregut tots els hashtags.

Cerca Per realitzar la recerca d'un hashtag, el primer que es necessita és el hashCode d'aquest hashtag. Una vegada que ja es tingui aquesta clau, es pot començar amb la cerca. Es calcula en què fila de la taula de hash es troba indexat el hashtag. Un cop s'hagi recuperat la fila, s'ha de buscar pels nodes d'aquesta fila fins a trobar el node que tingui la mateixa clau que el hashCode del hashtag introduït.

El cost de fer una cerca en aquesta taula de hash és d'un cost O (n) on n és el nombre de nodes indexats en aquesta fila. En el millor dels casos el cost és O (1).

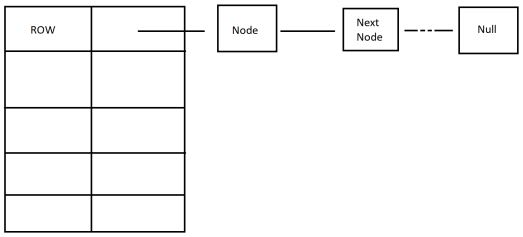


Figure 18: Exemple Taula de Hash

2.5 Graph

Un graf consisteix en un conjunt de nodes i arestes, les quals estableixen les relacions entre els nodes. El graf amb el qual es tracta en aquest projecte és un graf dirigit. En aquest cas, en cada node del graf es troba un usuari diferent, i les arestes fan relació als usuaris als quals un usuari segueix.

Per poder fer el graf possible, cal crear la classe GraphNode. Aquesta classe conté una sèrie d'atributs, els quals són:

1. Key: Aquest element fa referència al hashcode del nom d'usuari del propi usuari.

2. Element: El propi usuari

3. Connections: Array dels usuaris a els quals segeix.

A la classe graph, es troba una llista de nodes graph, en la qual es trobar tots els usuaris amb totes les seves connexions, el nombre de nodes o vèrtexs amb què consta el graph, i, finalment però no menys important, una taula de hash. S'utilitza una taula de hash per tenir guardat la posició en què es troba cada usuari indexat en el graf, de manera s'aconsegueix reduir el temps de recerca i accés a un usuari.

Inserció A l'hora d'inserir un nou usuari, el graf el que fa primer de tot és crear-se un nou GrafNode que faci referència a l'usuari. Un cop fet això, es passa a buscar on i en quina posició es guardarà aquest node en el graf, per després, poder guardar aquest índex a la taula de hash. Un cop inserit el la taula de hash s'augmenta el nombre de nodes que conté el graf.

Eliminació Quan es vulgui eliminar un usuari, el primer que es farà serà buscar a la taula de hash, quin és l'índex en el qual es troba l'element indexat en el graf. Si no rep cap vol dir que l'usuari que es vol eliminar no existeix. En el cas que si existeixi, a la casella del graf on es troba l'usuari es desactiva, i seguidament també s'esborra aquest node de la taula de hash.

Cerca Per realitzar la recerca d'un usuari simplement cal accedir a la hash table per extreure on es troba indexat a l'usuari

2.6 ArrayList

L'arraylist és una estructura de dades que ens permet emmagatzemar dades de forma lineal. És una de les estructures més bàsiques de la programació i s'utilitza per tot tipus de coses diferents. En primer lloc, aquest arraylist esta basat en una estructura molt semblant en quant a la forma, però més senzilla, que es l'array. Utilitzant aquest array, una de les funcions principals de l'arraylist, és fer que sigui dinàmic, és a dir, que puguis anar afegint elements sense haver-te de preocupar de si has demanat prou memòria per emmagatzemar-los tots. A part d'aquesta,hi ha altres funcions que s'expliquen a continuació:

Inserció El que es fa per defecte és crear un array de 10 posicions. Un cop l'usuari ha afegit 10 elements, i per tant, té l'array ple, cada cop que es vol afegir un element nou, es crea un array amb una posició més que l'array anterior i es copia tota la informació a aquest array nou.

Eliminació Per tal de dur a terme l'eliminació, el que es fa es eliminar l'element que es troba a la posició que l'usuari ens indiqui. Per realitzar aquest procés, en primer lloc es crea un array amb una posició menys que l'actual i seguidament es copien tots els elements menys el que ens han indicat al nou array.

Cerca Es tracta d'un procés molt senzill tot i que no resulta ni molt menys el més òptim. El que es fa és començar per el primer element d'aquest array i anar passant per tots els elements fins a trobar el element desitjat.

Aquesta estructura s'ha utilitzat com a substituta de l'arraylist implementat per java a tota la pràctica i per tant ha calgut fer-la com una estructura genèrica, que permet emmagatzemar qualsevol tipus d'objecte.

3 Comparació Estructures no optimitzades

En quant a la comparació de les estructures utilitzades al llarg del projecte, amb la més trivial i poc òptima com és el arraylist, hem decidit subdividir les nostres proves en 2 datasets, i totes les funcionalitats del programa.

3.1 Importació

Dataset Large:

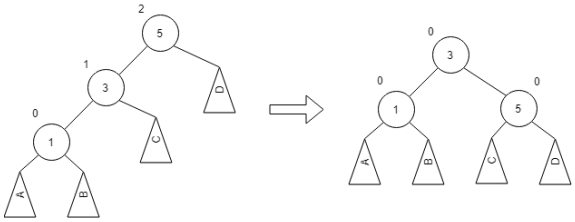


Figure 19: Cas LL

3.2 Exportació

Dataset Large:

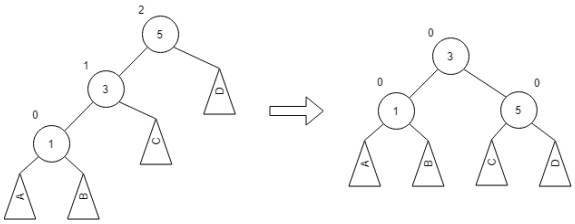


Figure 20: Cas LL

3.3 Inserció

Dataset Large:

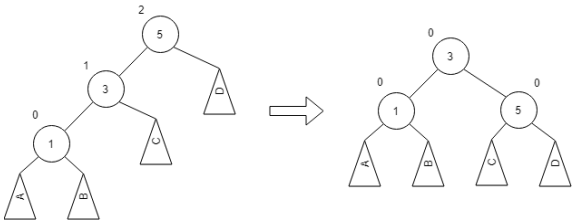


Figure 21: Cas LL

3.4 Eliminació

Dataset Large:

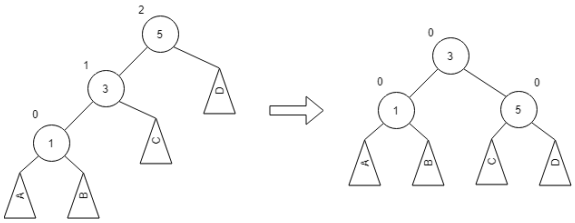


Figure 22: Cas LL

3.5 Cerca

Dataset Large:

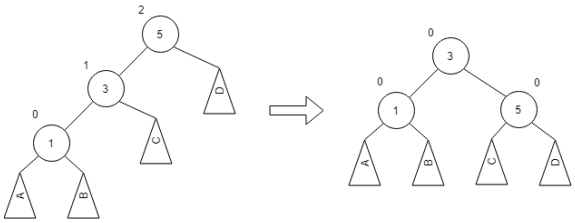


Figure 23: Cas LL

4 Mètode de proves

Per poder provar de forma més eficient el nostre programa ja des de un principi, vam crear el menú que hauríem de tenir a la versió final, per així poder introduir les dades de origen destí en el cas del primer problema i anar fent provatures del nostre codi més ràpid.
També vam programar les diferents funcions "genèriques" de cerca/inserció... de forma que dins de cadascuna només ens calgués posar en un switch, segons la estructura sobre la que estem treballant, la funció de cerca/inserció... específica.

Posteriorment, un cop anàvem tenint implementades les diferents funcionalitats de cada estructura de dades, hem anat realitzant proves a partir de usuaris i posts generats aleatòriament dins el propi programa.

4.1 AVL Tree

En el cas del AVL no va caldre fer us de cap dataset personalitzat, simplement, en un principi provàvem afegir i eliminar manualment, alhora que ho fèiem a paper per comprovar que les rotacions fossin correctes.
Posteriorment, gràcies a la funció de visualització, vam poder comprovar com un cop fèiem servir el AVL amb un dataset, les diferencies de distancia eren sempre les correctes, i tot hi que no podíem comprova-ho degut al gran nombre de posts, tot semblava indicar el seu correcte funcionament.

4.2 R-Tree

Per saber si el RTree estava correctament implementat o no ens realitzem un dataset de 16 posts amb tots els casos possibles que abasta l'estructura. Nota: la indexació a mà està al final per entendre millor la reestructuració de l'arbre en cada pas Comencem amb 3 posts ja introduïts, que estarien indexats en un LeafNodeArray P1 = (1, 1), P2 = (3,2), P3 = (1, 13)

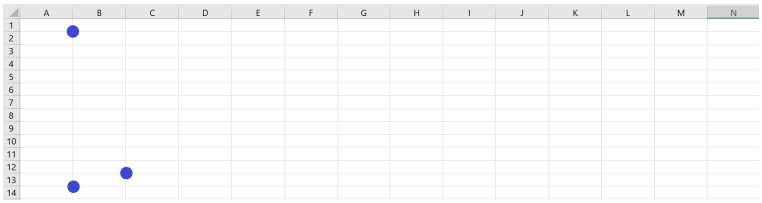


Figure 24: indexació amb el 3 primers Posts

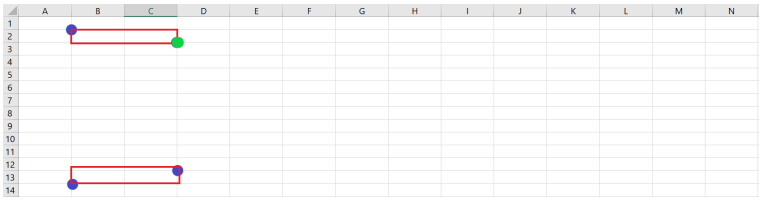


Figure 25: indexació del P4=(10, 5) -> Split per abaix, cas1

```
 ---------------------- PONEMOS EL POST CON ID -> 4 ----------------------

Elemento 0 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 4
            leafNode con ID = 3
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 2.0
        --> posicion 1 con area = 2.0

  su padre == null

Elemento 1 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 1
            leafNode con ID = 2
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 2.0
        --> posicion 1 con area = 2.0

  su padre == null
```

Figure 26: souts de la indexació del Post 4

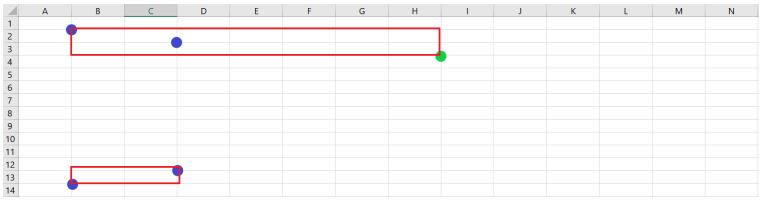


Figure 27: Introduïm el P5=(8, 11), ampliant la regió del rectangle amb fills P3 i P4 (caso2)

```
  ---------------------- PONEMOS EL POST CON ID -> 5 ----------------------

Elemento 0 es un NODERTREE
    su AREA = 14.0
    y sus HIJOS son -->
            leafNode con ID = 4
            leafNode con ID = 3
            leafNode con ID = 5
            PADRE LEAFNODES = aquel con área -> 14.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 14.0
        --> posicion 1 con area = 2.0

  su padre == null

Elemento 1 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 1
            leafNode con ID = 2
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 14.0
        --> posicion 1 con area = 2.0

  su padre == null
```

Figure 28: souts de la indexació del Post 5

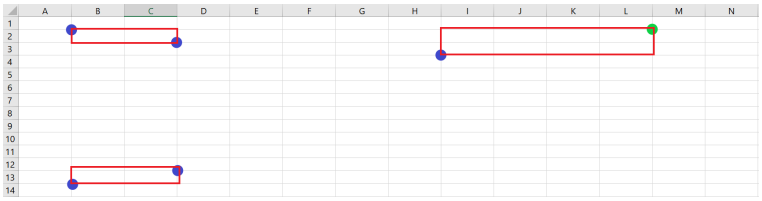


Figure 29: Introducimos el P6 (12, 13), realizando un Split por arriba del nodo de área mayor (caso3)

```
------------------------ PONEMOS EL POST CON ID -> 6 --

Elemento 0 es un NODERTREE
    su AREA = 8.0
    y sus HIJOS son -->
            leafNode con ID = 6
            leafNode con ID = 5
            PADRE LEAFNODES = aquel con área -> 8.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 8.0
        --> posicion 1 con area = 2.0
        --> posicion 2 con area = 2.0

  su padre == null

Elemento 1 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 3
            leafNode con ID = 4
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 8.0
        --> posicion 1 con area = 2.0
        --> posicion 2 con area = 2.0

  su padre == null
```

Figure 30: souts de la indexació del Post 6

```
Elemento 2 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 1
            leafNode con ID = 2
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 8.0
        --> posicion 1 con area = 2.0
        --> posicion 2 con area = 2.0

  su padre == null
```

Figure 31: souts de la indexació del Post 6

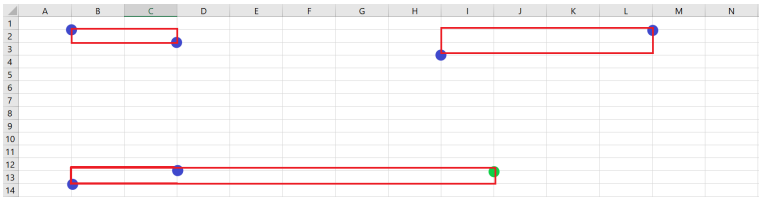


Figure 32: Indexació del P7=(9, 2), ampliant la regió del rectangle amb fill P1 i P2 (cas2)

```
 ------------------------ PONEMOS EL POST CON ID -> 7 ----

Elemento 0 es un NODERTREE
    su AREA = 8.0
    y sus HIJOS son -->
            leafNode con ID = 6
            leafNode con ID = 5
            PADRE LEAFNODES = aquel con área -> 8.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 8.0
        --> posicion 1 con area = 2.0
        --> posicion 2 con area = 8.0

  su padre == null

Elemento 1 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 3
            leafNode con ID = 4
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 8.0
        --> posicion 1 con area = 2.0
        --> posicion 2 con area = 8.0

  su padre == null
```

Figure 33: souts de la indexació del Post 7

```
Elemento 2 es un NODERTREE
    su AREA = 8.0
    y sus HIJOS son -->
            leafNode con ID = 1
            leafNode con ID = 2
            leafNode con ID = 7
            PADRE LEAFNODES = aquel con área -> 8.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 8.0
        --> posicion 1 con area = 2.0
        --> posicion 2 con area = 8.0

  su padre == null
```

Figure 34: souts de la indexació del Post 7

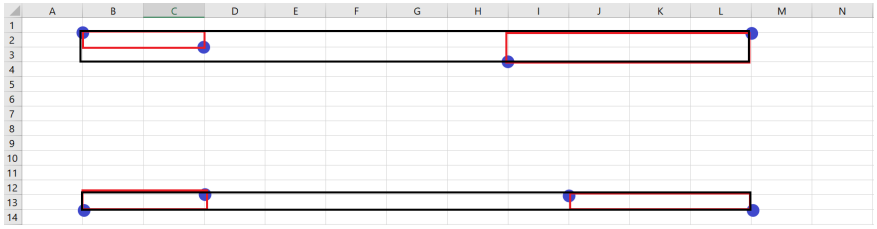


Figure 35: Indexació del P8=(12, 1), realització d'un split per abaix! (cas3)

```
------------------------ PONEMOS EL POST CON ID -> 8 ------------

Elemento 0 es un NODERTREE
    su AREA = 11.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 3.0
    y sus HIJOS son -->
            leafNode con ID = 8
            leafNode con ID = 7
            PADRE LEAFNODES = aquel con área -> 3.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 3.0
        --> posicion 1 con area = 2.0

      su padre es aquel con área -> 11.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 11.0
            --> hijo NODERTREE de la posicion 1 con area 22.0
```

Figure 36: souts de la indexació del Post 8

```
Elemento 1 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 1
            leafNode con ID = 2
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 3.0
        --> posicion 1 con area = 2.0

      su padre es aquel con área -> 11.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 11.0
            --> hijo NODERTREE de la posicion 1 con area 22.0

  su padre == null

Elemento 1 es un NODERTREE
    su AREA = 22.0
    y sus HIJOS son -->
```

Figure 37: souts de la indexació del Post 8

```
Elemento 1 es un NODERTREE
    su AREA = 22.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 3
            leafNode con ID = 4
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 2.0
        --> posicion 1 con area = 8.0

      su padre es aquel con área -> 22.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 11.0
            --> hijo NODERTREE de la posicion 1 con area 22.0
```

Figure 38: souts de la indexació del Post 8

```
Elemento 1 es un NODERTREE
    su AREA = 8.0
    y sus HIJOS son -->
            leafNode con ID = 6
            leafNode con ID = 5
            PADRE LEAFNODES = aquel con área -> 8.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 2.0
        --> posicion 1 con area = 8.0

      su padre es aquel con área -> 22.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 11.0
            --> hijo NODERTREE de la posicion 1 con area 22.0

  su padre == null
```

Figure 39: souts de la indexació del Post 8

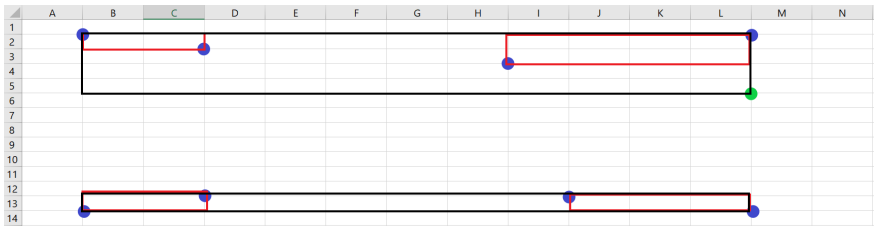


Figure 40: Indexació del P9=(12, 9), ampliant la regió del rectangle amb fill P5 i P6 (cas2)

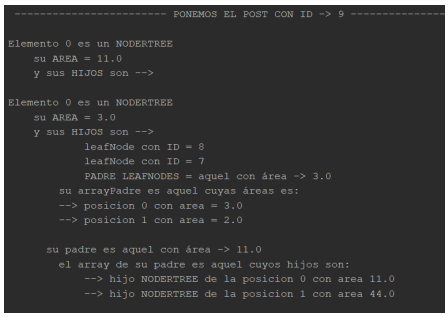


Figure 41: souts de la indexació del Post 9

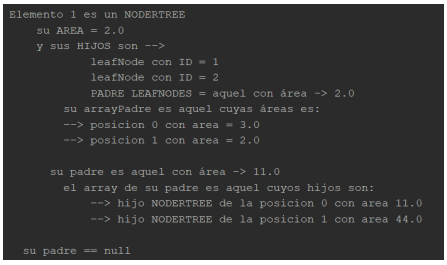


Figure 42: souts de la indexació del Post 9

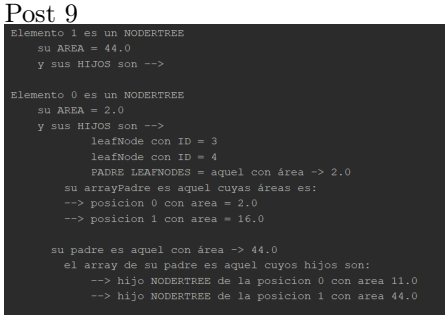


Figure 43: souts de la indexació del Post 9

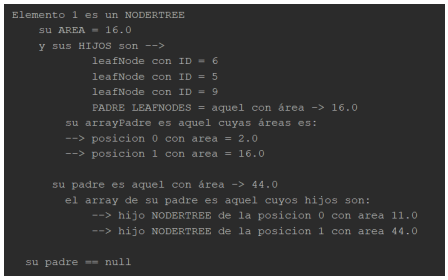


Figure 44: souts de la indexació del Post 9

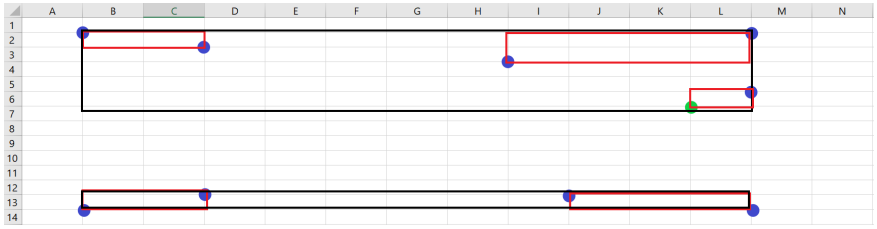


Figure 45: Indexació del P10=(11, 8), realitzant un split cap a dalt! (cas4)

```
 ----------------------- PONEMOS EL POST CON ID -> 10 -----------

Elemento 0 es un NODERTREE
    su AREA = 11.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 3.0
    y sus HIJOS son -->
            leafNode con ID = 8
            leafNode con ID = 7
            PADRE LEAFNODES = aquel con área -> 3.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 3.0
        --> posicion 1 con area = 2.0

      su padre es aquel con área -> 11.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 11.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 46: souts de la indexació del Post 10

```
Elemento 1 es un NODERTREE
    su AREA = 55.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 1.0
    y sus HIJOS son -->
            leafNode con ID = 10
            leafNode con ID = 9
            PADRE LEAFNODES = aquel con área -> 1.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 8.0
        --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 11.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 48: souts de la indexació del Post 10

```
Elemento 2 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 3
            leafNode con ID = 4
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 8.0
        --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 11.0
            --> hijo NODERTREE de la posicion 1 con area 55.0

  su padre == null
```

Figure 50: souts de la indexació del Post 10

```
Elemento 1 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 1
            leafNode con ID = 2
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 3.0
        --> posicion 1 con area = 2.0

      su padre es aquel con área -> 11.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 11.0
            --> hijo NODERTREE de la posicion 1 con area 55.0

  su padre == null
```

Figure 47: souts de la indexació del Post 10

```
Elemento 1 es un NODERTREE
    su AREA = 8.0
    y sus HIJOS son -->
            leafNode con ID = 6
            leafNode con ID = 5
            PADRE LEAFNODES = aquel con área -> 8.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 8.0
        --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 11.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 49: souts de la indexació del Post 10

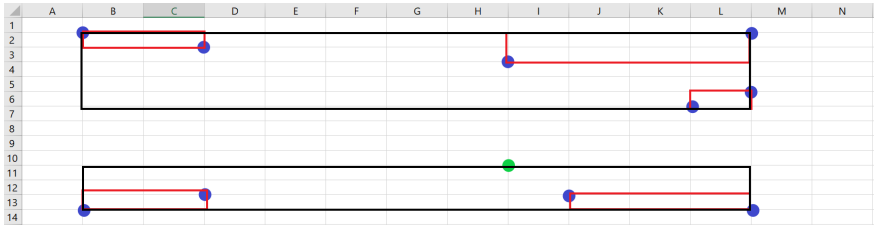


Figure 51: Indexació del P11=(8, 4), ampliant la regió del rectangle amb fill P7 i P8 (cas2)

```
 ---------------------- PONEMOS EL POST CON ID -> 11 ----------

Elemento 0 es un NODERTREE
    su AREA = 33.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 12.0
    y sus HIJOS son -->
            leafNode con ID = 8
            leafNode con ID = 7
            leafNode con ID = 11
            PADRE LEAFNODES = aquel con área -> 12.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 12.0
        --> posicion 1 con area = 2.0

      su padre es aquel con área -> 33.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 33.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 52: souts de la indexació del Post 11

```
Elemento 1 es un NODERTREE
    su AREA = 55.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 1.0
    y sus HIJOS son -->
            leafNode con ID = 10
            leafNode con ID = 9
            PADRE LEAFNODES = aquel con área -> 1.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 8.0
        --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 33.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 54: souts de la indexació del Post 11

```
Elemento 2 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 3
            leafNode con ID = 4
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 8.0
        --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 33.0
            --> hijo NODERTREE de la posicion 1 con area 55.0

  su padre == null
```

Figure 56: souts de la indexació del Post 11

```
Elemento 1 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 1
            leafNode con ID = 2
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 12.0
        --> posicion 1 con area = 2.0

      su padre es aquel con área -> 33.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 33.0
            --> hijo NODERTREE de la posicion 1 con area 55.0

  su padre == null
```

Figure 53: souts de la indexació del Post 11

```
Elemento 1 es un NODERTREE
    su AREA = 8.0
    y sus HIJOS son -->
            leafNode con ID = 6
            leafNode con ID = 5
            PADRE LEAFNODES = aquel con área -> 8.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 8.0
        --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 33.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 55: souts de la indexació del Post 11

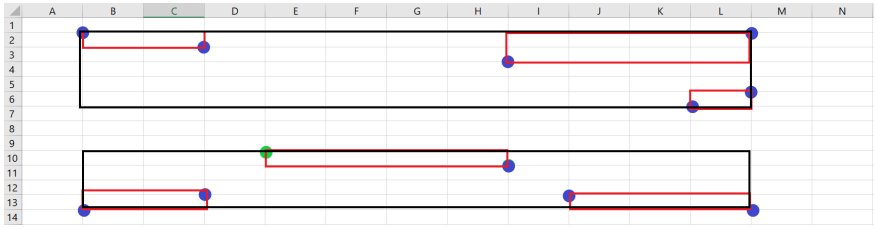


Figure 57: Indexació del P12=(4, 5), ampliant la regió del rectangle amb fill P1 i P2 (cas2)

```
 ---------------------- PONEMOS EL POST CON ID -> 12 ----------

Elemento 0 es un NODERTREE
    su AREA = 44.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 12.0
    y sus HIJOS son -->
            leafNode con ID = 8
            leafNode con ID = 7
            leafNode con ID = 11
            PADRE LEAFNODES = aquel con área -> 12.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 12.0
        --> posicion 1 con area = 12.0

      su padre es aquel con área -> 44.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 58: souts de la indexació del Post 12

```
Elemento 1 es un NODERTREE
    su AREA = 55.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 1.0
    y sus HIJOS son -->
            leafNode con ID = 10
            leafNode con ID = 9
            PADRE LEAFNODES = aquel con área -> 1.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 8.0
        --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 60: souts de la indexació del Post 12

```
Elemento 2 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 3
            leafNode con ID = 4
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 8.0
        --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0

  su padre == null
```

Figure 62: souts de la indexació del Post 12

```
Elemento 1 es un NODERTREE
    su AREA = 12.0
    y sus HIJOS son -->
            leafNode con ID = 1
            leafNode con ID = 2
            leafNode con ID = 12
            PADRE LEAFNODES = aquel con área -> 12.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 12.0
        --> posicion 1 con area = 12.0

      su padre es aquel con área -> 44.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0

  su padre == null
```

Figure 59: souts de la indexació del Post 12

```
Elemento 1 es un NODERTREE
    su AREA = 8.0
    y sus HIJOS son -->
            leafNode con ID = 6
            leafNode con ID = 5
            PADRE LEAFNODES = aquel con área -> 8.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 8.0
        --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 61: souts de la indexació del Post 12

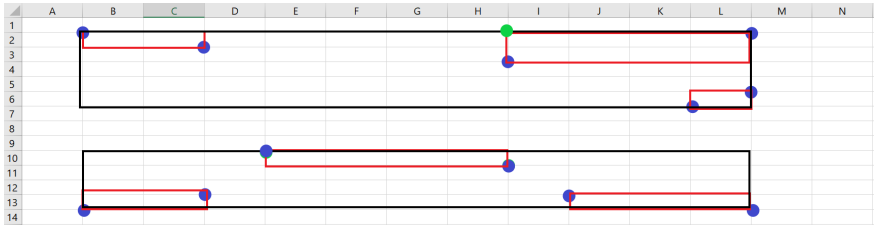


Figure 63: Indexació del P13=(8, 13), ampliant la regió del rectangle amb fill P5 i P6 (cas2)

```
 ------------------------ PONEMOS EL POST CON ID -> 13 -------------

Elemento 0 es un NODERTREE
    su AREA = 44.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 12.0
    y sus HIJOS son -->
            leafNode con ID = 8
            leafNode con ID = 7
            leafNode con ID = 11
            PADRE LEAFNODES = aquel con área -> 12.0
       su arrayPadre es aquel cuyas áreas es:
       --> posicion 0 con area = 12.0
       --> posicion 1 con area = 12.0

      su padre es aquel con área -> 44.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 64: souts de la indexació del Post 13

```
Elemento 1 es un NODERTREE
    su AREA = 55.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 1.0
    y sus HIJOS son -->
            leafNode con ID = 10
            leafNode con ID = 9
            PADRE LEAFNODES = aquel con área -> 1.0
       su arrayPadre es aquel cuyas áreas es:
       --> posicion 0 con area = 1.0
       --> posicion 1 con area = 8.0
       --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 66: souts de la indexació del Post 13

```
Elemento 2 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 3
            leafNode con ID = 4
            PADRE LEAFNODES = aquel con área -> 2.0
       su arrayPadre es aquel cuyas áreas es:
       --> posicion 0 con area = 1.0
       --> posicion 1 con area = 8.0
       --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0

  su padre == null
```

Figure 68: souts de la indexació del Post 13

```
Elemento 1 es un NODERTREE
    su AREA = 12.0
    y sus HIJOS son -->
            leafNode con ID = 1
            leafNode con ID = 2
            leafNode con ID = 12
            PADRE LEAFNODES = aquel con área -> 12.0
       su arrayPadre es aquel cuyas áreas es:
       --> posicion 0 con area = 12.0
       --> posicion 1 con area = 12.0

      su padre es aquel con área -> 44.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0

  su padre == null
```

Figure 65: souts de la indexació del Post 13

```
Elemento 1 es un NODERTREE
    su AREA = 8.0
    y sus HIJOS son -->
            leafNode con ID = 6
            leafNode con ID = 5
            leafNode con ID = 13
            PADRE LEAFNODES = aquel con área -> 8.0
       su arrayPadre es aquel cuyas áreas es:
       --> posicion 0 con area = 1.0
       --> posicion 1 con area = 8.0
       --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 67: souts de la indexació del Post 13

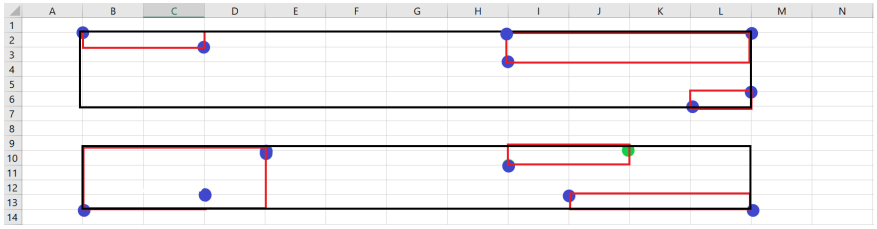


Figure 69: Indexació del P14=(10, 5), realitzant un split cap a dalt! (cas4)

```
---------------------- PONEMOS EL POST CON ID -> 14 ---------

Elemento 0 es un NODERTREE
    su AREA = 44.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 14
            leafNode con ID = 11
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 2.0
        --> posicion 1 con area = 3.0
        --> posicion 2 con area = 12.0

      su padre es aquel con área -> 44.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 70: souts de la indexació del Post 14

```
Elemento 1 es un NODERTREE
    su AREA = 3.0
    y sus HIJOS son -->
            leafNode con ID = 8
            leafNode con ID = 7
            PADRE LEAFNODES = aquel con área -> 3.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 2.0
        --> posicion 1 con area = 3.0
        --> posicion 2 con area = 12.0

      su padre es aquel con área -> 44.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 71: souts de la indexació del Post 14

```
Elemento 2 es un NODERTREE
    su AREA = 12.0
    y sus HIJOS son -->
            leafNode con ID = 1
            leafNode con ID = 2
            leafNode con ID = 12
            PADRE LEAFNODES = aquel con área -> 12.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 2.0
        --> posicion 1 con area = 3.0
        --> posicion 2 con area = 12.0

      su padre es aquel con área -> 44.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0

  su padre == null
```

Figure 72: souts de la indexació del Post 14

```
Elemento 1 es un NODERTREE
    su AREA = 55.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 1.0
    y sus HIJOS son -->
            leafNode con ID = 10
            leafNode con ID = 9
            PADRE LEAFNODES = aquel con área -> 1.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 8.0
        --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 73: souts de la indexació del Post 14

```
Elemento 1 es un NODERTREE
    su AREA = 8.0
    y sus HIJOS son -->
            leafNode con ID = 6
            leafNode con ID = 5
            leafNode con ID = 13
            PADRE LEAFNODES = aquel con área -> 8.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 8.0
        --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
```

Figure 74: souts de la indexació del Post 14

```
Elemento 2 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 3
            leafNode con ID = 4
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 8.0
        --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 44.0
            --> hijo NODERTREE de la posicion 1 con area 55.0

  su padre == null
```

Figure 75: souts de la indexació del Post 14

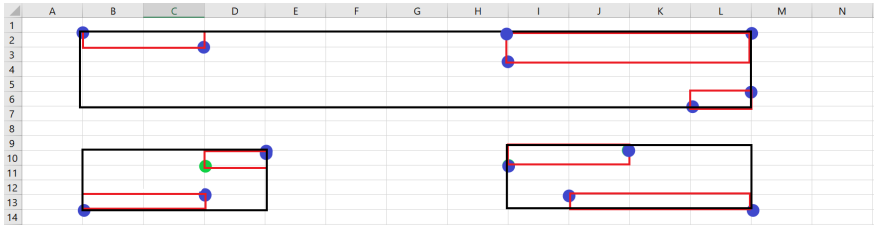


Figure 76: Indexació del P15=(3, 4), realitzant un split cap a dalt! (cas4)

```
 ----------------------- PONEMOS EL POST CON ID -> 15 -----------

Elemento 0 es un NODERTREE
    su AREA = 7.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 1.0
    y sus HIJOS son -->
            leafNode con ID = 15
            leafNode con ID = 12
            PADRE LEAFNODES = aquel con área -> 1.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 2.0

      su padre es aquel con área -> 7.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 7.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
            --> hijo NODERTREE de la posicion 2 con area 11.0
```

Figure 77: souts de la indexació del Post 15

```
Elemento 1 es un NODERTREE
    su AREA = 55.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 1.0
    y sus HIJOS son -->
            leafNode con ID = 10
            leafNode con ID = 9
            PADRE LEAFNODES = aquel con área -> 1.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 8.0
        --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 7.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
            --> hijo NODERTREE de la posicion 2 con area 11.0
```

Figure 79: souts de la indexació del Post 15

```
Elemento 2 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 3
            leafNode con ID = 4
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 8.0
        --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 7.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
            --> hijo NODERTREE de la posicion 2 con area 11.0

  su padre == null
```

Figure 81: souts de la indexació del Post 15

```
Elemento 1 es un NODERTREE
    su AREA = 3.0
    y sus HIJOS son -->
            leafNode con ID = 8
            leafNode con ID = 7
            PADRE LEAFNODES = aquel con área -> 3.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 2.0
        --> posicion 1 con area = 3.0

      su padre es aquel con área -> 11.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 7.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
            --> hijo NODERTREE de la posicion 2 con area 11.0

  su padre == null
```

Figure 83: souts de la indexació del Post 15

```
Elemento 1 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 14
            leafNode con ID = 11
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 2.0

      su padre es aquel con área -> 7.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 7.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
            --> hijo NODERTREE de la posicion 2 con area 11.0

  su padre == null
```

Figure 78: souts de la indexació del Post 15

```
Elemento 1 es un NODERTREE
    su AREA = 8.0
    y sus HIJOS son -->
            leafNode con ID = 6
            leafNode con ID = 5
            leafNode con ID = 13
            PADRE LEAFNODES = aquel con área -> 8.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 8.0
        --> posicion 2 con area = 2.0

      su padre es aquel con área -> 55.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 7.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
            --> hijo NODERTREE de la posicion 2 con area 11.0
```

Figure 80: souts de la indexació del Post 15

```
Elemento 2 es un NODERTREE
    su AREA = 11.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 1
            leafNode con ID = 2
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 2.0
        --> posicion 1 con area = 3.0

      su padre es aquel con área -> 11.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 7.0
            --> hijo NODERTREE de la posicion 1 con area 55.0
            --> hijo NODERTREE de la posicion 2 con area 11.0
```

Figure 82: souts de la indexació del Post 15

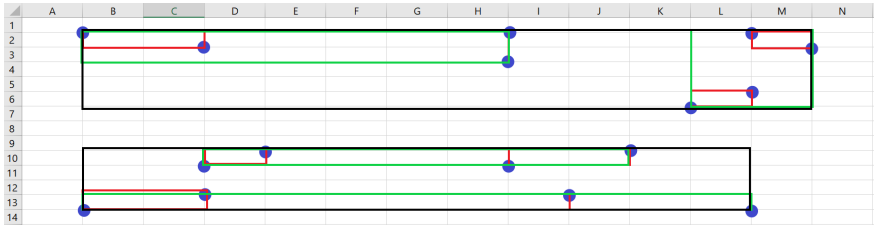


Figure 84: Indexació del P16=(13, 12), realització d'un split per abaix! (cas3)

```
---------------------- PONEMOS EL POST CON ID -> 16 ----------

Elemento 0 es un NODERTREE
    su AREA = 60.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 10.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 16
            leafNode con ID = 6
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 2.0
        --> posicion 1 con area = 1.0

      su padre es aquel con área -> 10.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 10.0
            --> hijo NODERTREE de la posicion 1 con area 14.0
```

Figure 85: souts de la indexació del Post 16

```
Elemento 1 es un NODERTREE
    su AREA = 14.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 3
            leafNode con ID = 4
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 2.0
        --> posicion 1 con area = 0.0

      su padre es aquel con área -> 14.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 10.0
            --> hijo NODERTREE de la posicion 1 con area 14.0
```

Figure 87: souts de la indexació del Post 16

```
Elemento 1 es un NODERTREE
    su AREA = 44.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 7.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 1.0
    y sus HIJOS son -->
            leafNode con ID = 15
            leafNode con ID = 12
            PADRE LEAFNODES = aquel con área -> 1.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 2.0

      su padre es aquel con área -> 7.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 7.0
            --> hijo NODERTREE de la posicion 1 con area 11.0
```

Figure 89: souts de la indexació del Post 16

```
Elemento 1 es un NODERTREE
    su AREA = 11.0
    y sus HIJOS son -->

Elemento 0 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 1
            leafNode con ID = 0
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 2.0
        --> posicion 1 con area = 3.0

      su padre es aquel con área -> 11.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 7.0
            --> hijo NODERTREE de la posicion 1 con area 11.0
```

Figure 91: souts de la indexació del Post 16


```
Elemento 1 es un NODERTREE
    su AREA = 1.0
    y sus HIJOS son -->
            leafNode con ID = 10
            leafNode con ID = 9
            PADRE LEAFNODES = aquel con área -> 1.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 2.0
        --> posicion 1 con area = 1.0

      su padre es aquel con área -> 10.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 10.0
            --> hijo NODERTREE de la posicion 1 con area 14.0

      su padre es aquel con área -> 60.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 60.0
            --> hijo NODERTREE de la posicion 1 con area 44.0
```

Figure 86: souts de la indexació del Post 16

```
Elemento 1 es un NODERTREE
    su AREA = 0.0
    y sus HIJOS son -->
            leafNode con ID = 5
            leafNode con ID = 13
            PADRE LEAFNODES = aquel con área -> 0.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 2.0
        --> posicion 1 con area = 0.0

      su padre es aquel con área -> 14.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 10.0
            --> hijo NODERTREE de la posicion 1 con area 14.0

      su padre es aquel con área -> 60.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 60.0
            --> hijo NODERTREE de la posicion 1 con area 44.0

  su padre == null
```

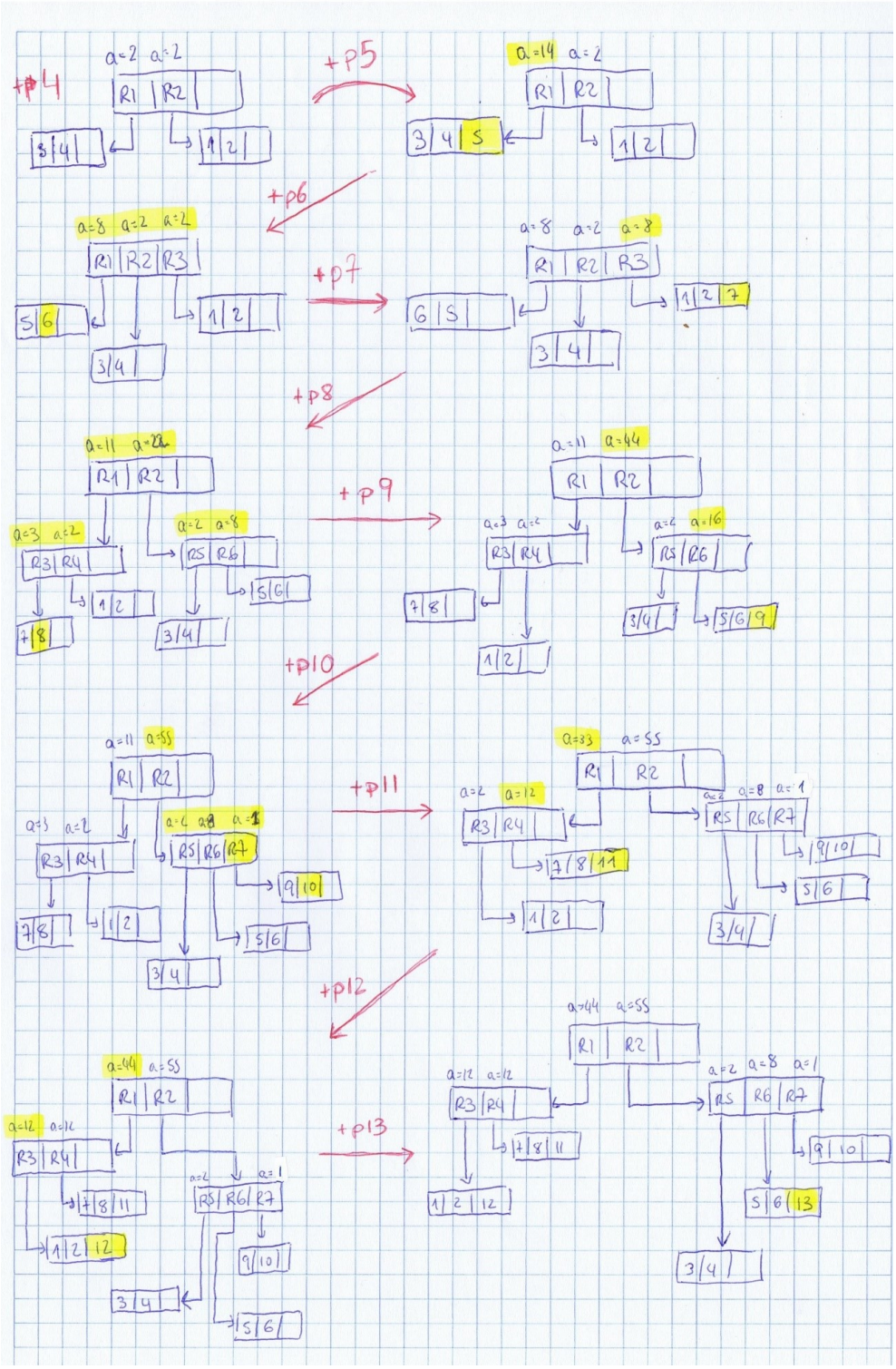
Figure 88: souts de la indexació del Post 16

```
Elemento 1 es un NODERTREE
    su AREA = 2.0
    y sus HIJOS son -->
            leafNode con ID = 14
            leafNode con ID = 11
            PADRE LEAFNODES = aquel con área -> 2.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 1.0
        --> posicion 1 con area = 2.0

      su padre es aquel con área -> 7.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 7.0
            --> hijo NODERTREE de la posicion 1 con area 11.0

      su padre es aquel con área -> 44.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 60.0
            --> hijo NODERTREE de la posicion 1 con area 44.0
```

Figure 90: souts de la indexació del Post 16

```
Elemento 1 es un NODERTREE
    su AREA = 3.0
    y sus HIJOS son -->
            leafNode con ID = 8
            leafNode con ID = 7
            PADRE LEAFNODES = aquel con área -> 3.0
        su arrayPadre es aquel cuyas áreas es:
        --> posicion 0 con area = 2.0
        --> posicion 1 con area = 3.0

      su padre es aquel con área -> 11.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 7.0
            --> hijo NODERTREE de la posicion 1 con area 11.0

      su padre es aquel con área -> 44.0
        el array de su padre es aquel cuyos hijos son:
            --> hijo NODERTREE de la posicion 0 con area 60.0
            --> hijo NODERTREE de la posicion 1 con area 44.0

  su padre == null

Process finished with exit code 0
```

Figure 92: souts de la indexació del



Programació avançada i estructures de dades, 2019

Figure 93: arbre escrit de la indexació dels posts a la estructura

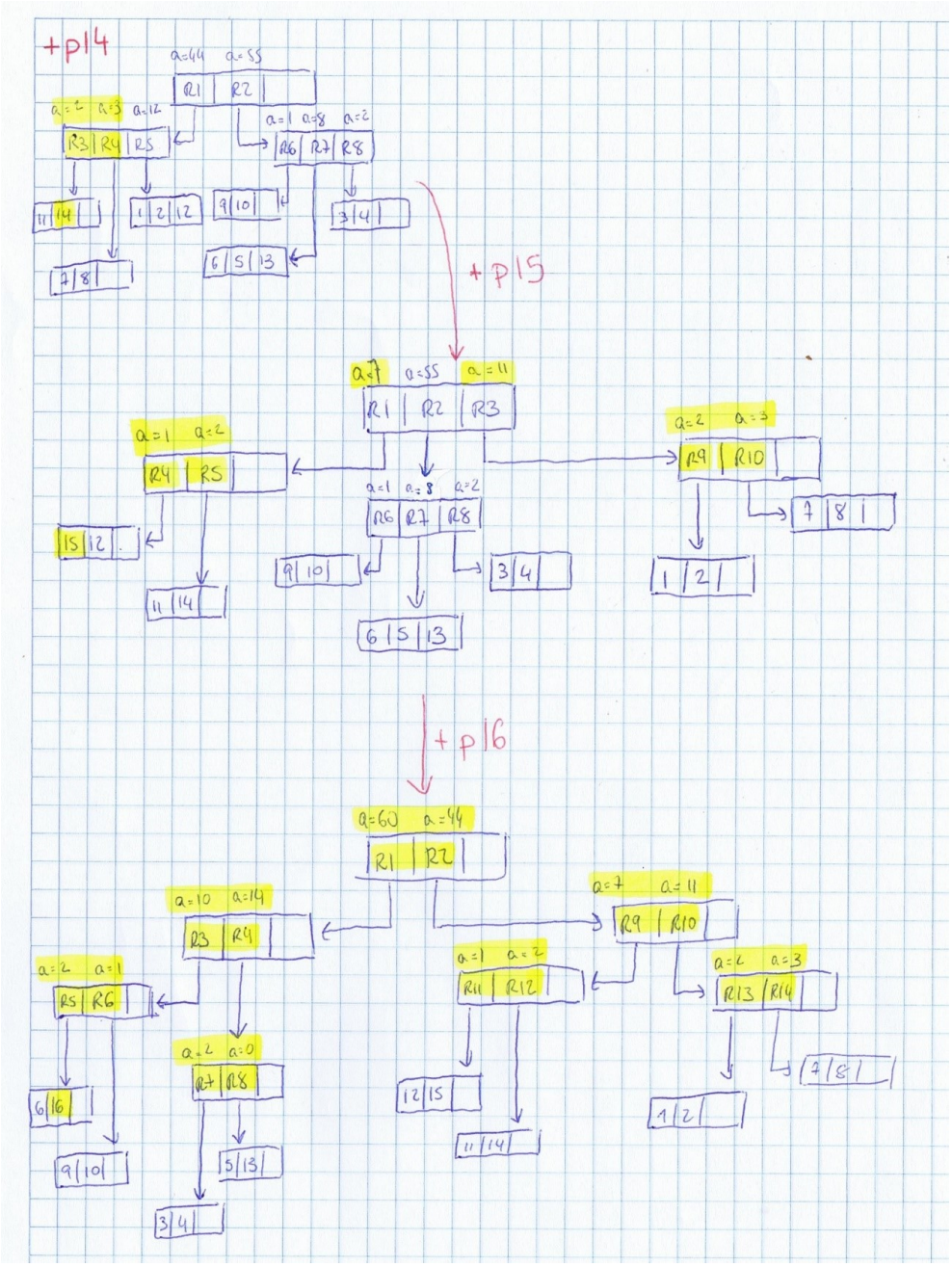


Figure 94: arbre escrit de la indexació dels posts a la estructura

4.3 HashTable

Per comprovar si la inserció en la taula de hash funcionava correctament ens vam crear un dataset amb 4 posts i diversos hashtags amb alguns iguals entre ells.

I com podem observar, la fa correctament:

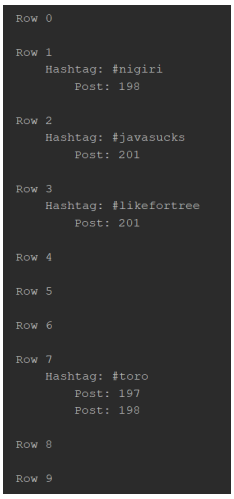


Figure 95: captura hash 1

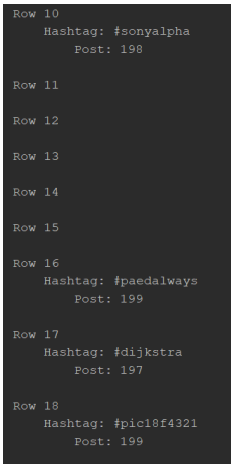


Figure 96: captura hash 2

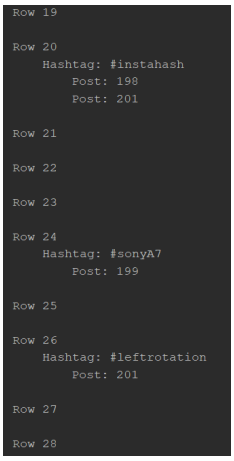


Figure 97: captura hash 3

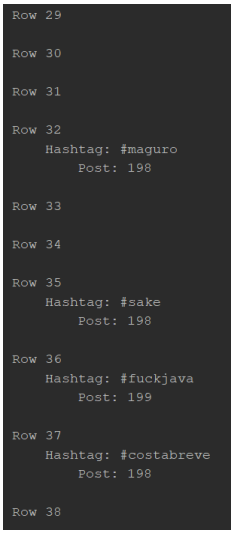


Figure 98: captura hash 4

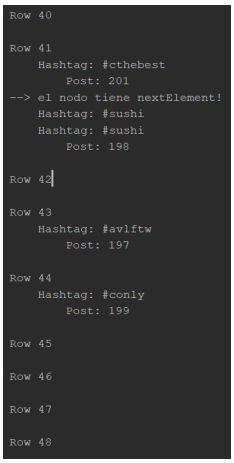


Figure 99: captura hash 5

4.4 Arraylist

Com a mètode de proves per l'arraylist es va implementar una main que realitzava les funcions bàsiques implementades i es van anar fent algunes petites modificacions fins que va funcionar del tot.

4.5 Graph

Com a mètode de proves per el graph, es van crear diferents datasets més petits per facilitar la lectura del resultats donada per la estructura. A priori, per comprovar que una funció funcionaba, vam crear diferents usuaris per poder veure si l'inserció, la cerca i l'eliminació funcionaven de forma correcta.

5 Comparativa

Podem observar clarament tant per a usuaris com per users quant d'òptims en alguns aspectes són unes estructures, i com per altres funcionalitats no ho són. Posem l'exemple de l'Graph. El graph és l'estructura que més triga a importar Usuaris, però després a l'hora de realitzar una cerca, eliminació o inserir un sol usuari nou, té un temps temps d'execució menor al ArrayList. Tries en comparació amb les estucturas Graph i array és sens dubte qui que realitza els canvis en l'estructura de forma més ràpida i eficaç.
Parlant dels posts, la taula de hash es la estructura que més triga en inserir els elements. Això té sentit ja que per cada hashtag que conté un post, la taula de hash haurà d'inserir el post, es a dir, l'inserirà tantes vegades com hashtags tingui el post, augmentant el temps de importació de les dades. No obstant això, podem observar també que a l'hora de realitzar cerques, eliminacions o insercions, el temps que triga l'estructura decreix notablement. L'estructura d'Array no ha sigut capaç de acabar la seva funció, degut a la inmensitat de les dades. Les estructures més òptimes a l'hora de realitzar les tasques dels usuaris podem observar que es sense dubte l'Arbre AVL. Cal marcar d'aquesta estructura el balanç, la qual cosa és el que ajuda a que la recerca en un d'aquests arbres es mantingui sempre amb una complexitat O (log n)

5.1 Posts

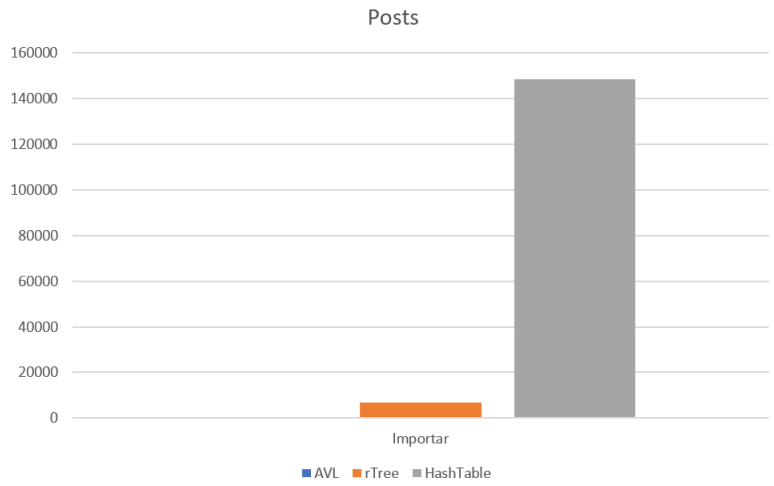


Figure 100: Importació Posts Dataset Large

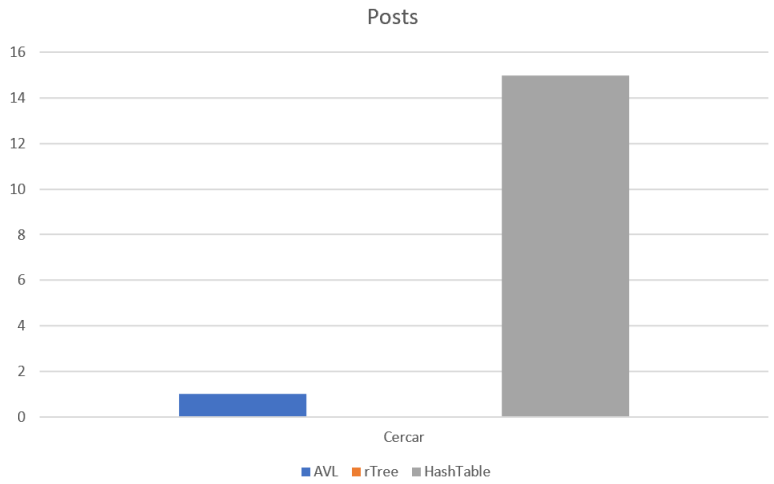


Figure 101: Cerca Posts Dataset Large

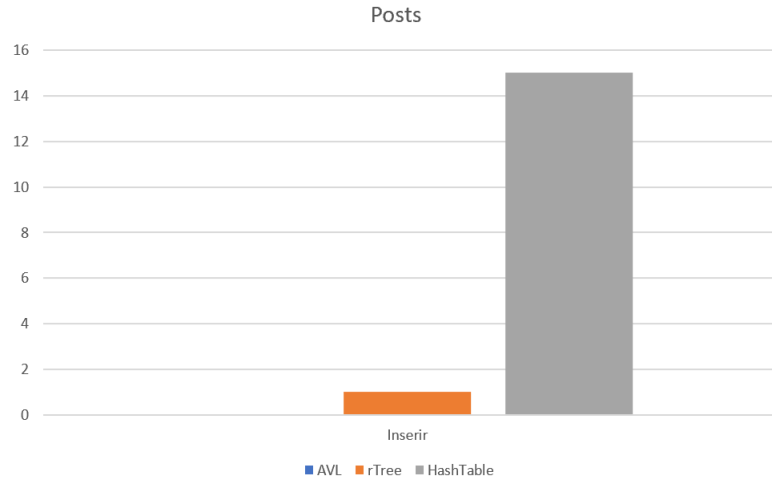


Figure 102: Inserció Post Dataset Large

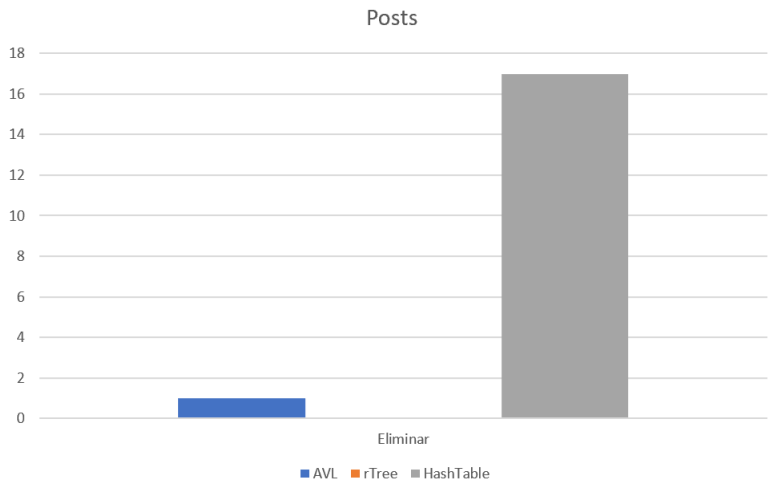


Figure 103: Eliminació Post Dataset Large

5.2 Users

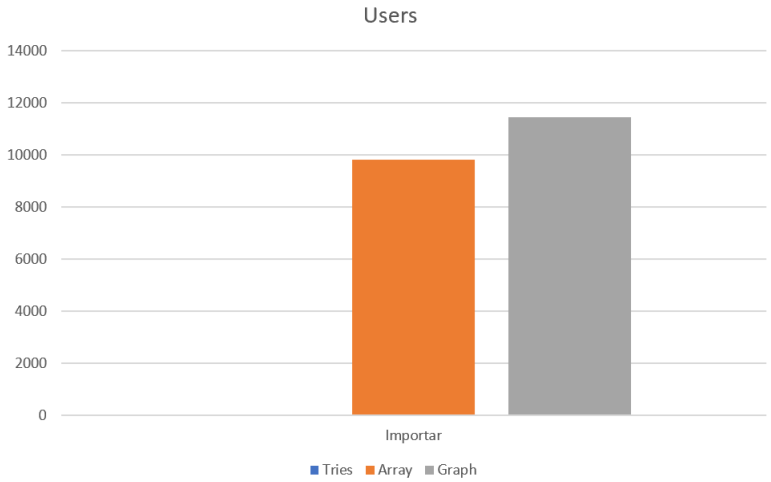


Figure 104: Importació Users Dataset Large

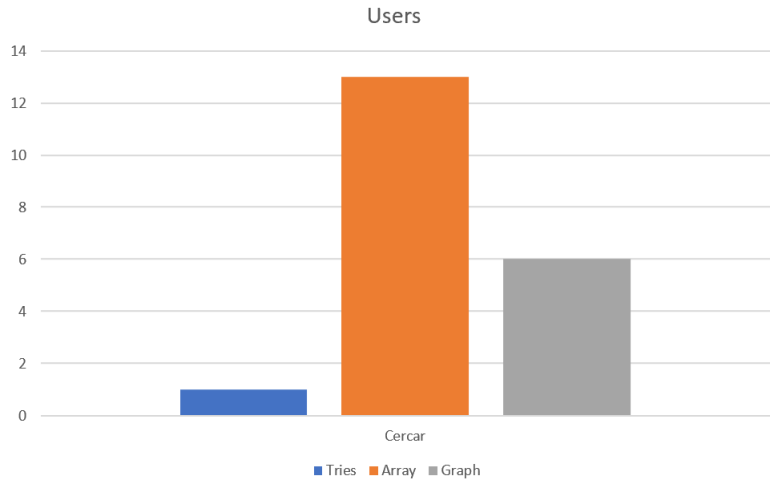


Figure 105: Cerca User Dataset Large

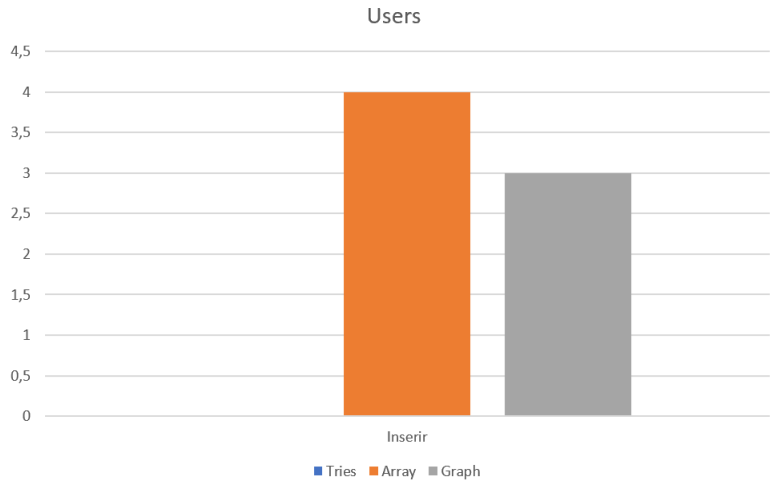


Figure 106: Inserció User Dataset Large

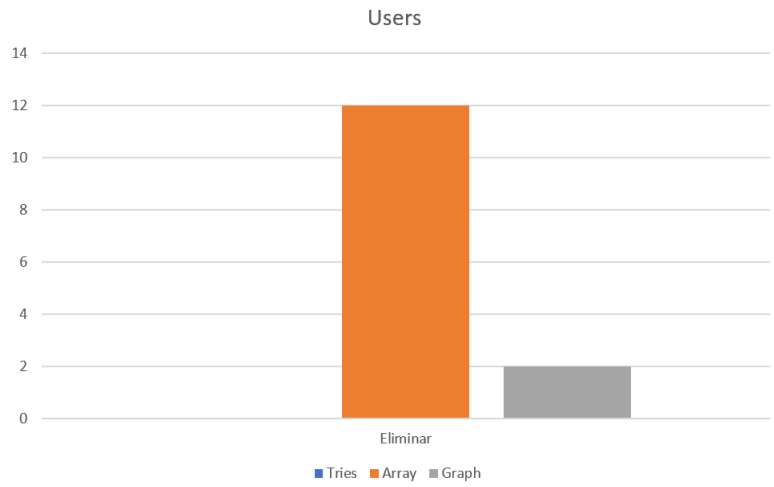


Figure 107: Eliminació User Dataset Large

6 Problemes Observats

Alhora de realitzar la implementació del AVL Tree, tot hi que la gran majoria varen ser problemes simples relacionats amb errors de programació, al programar la eliminació inicialment, el arbre no ens quedava correctament balancejat quan el node eliminat tenia dos fills. Després de moltes provatures i debugging, varem adonar-nos de que el principal error era que tot hi que un cop substituït el node, re-calculàvem la altura, no estàvem re-calculant la alçada total del arbre, de forma que el algorisme no s'adonava de que havia un desbalanceig.
Per solucionar-ho, vam programar una nova funció de calcul de balance factor específica per les eliminacions, que tenia en compte el problema anterior.

En el moment de realitzar el graph, en afegir una taula de hash com a atribut, es va haver de tornar a pensar i distribuir l'estructura de hash, és a dir, tornar a realitzar noves funcions i eliminar altres que després no s'utilitzen.

En quant a problemes que hem tingut per a la implementació del arraylist principalment van ser problemes relacionats amb el tipus object, ja que segons quines funcions es volen realitzar, dona problemes a l'hora de fer el cast al objecte que s'estigui utilitzant.

Per altra banda, si parlem dels problemes que hem tingut a l'hora de fer els tries, principalment han estat dos. En primer lloc, el fet d'haver pensat l'estructura perquè funcionés únicament amb caràcters de la 'a' a la 'z' va fer que al trobar-nos amb el dataset amb números i majúscules, haguéssim de canviar l'estructura per permetre emmagatzemar qualsevol caràcter de la taula ascii. En segon lloc, el fet de no tenir una relació entre el fill i el seu pare, ens va suposar una petita complicació que a priori no havíem previst a l'hora de moure'ns per aquesta estructura.

En quant als problemes trobats a l'hora de la implementació del rTree, es podria fer una memòria sencera amb aquests. Hi han hagut 3 versions del codi. La primera de tot va ser una pèrdua total de temps degut a no acabar de entendre bé el funcionament l'estructura, des de quin index màxim d'elements possibles permetíem (M), passant a com realitzar la estructura que diferencii un NodeRtree d'un NodeLeaf, fins els Splits. La segona també vàrem tindre problemes a l'hora d'organitzar el codi, ja que hi havien moltes funcions que eren més o menys similars però que variaven en uns punts específics, fent que aquesta versió del codi tingués més de 1100 línies de codi, un temps d'execució de més de 10 cops l'actual i una inserció que no funcionava en tots els cassos possibles. Finalment, en la tercera versió, es va canviar la estructura del programa, utilitzant 1 classe diferent per a cada tipus de node i fent que únicament els array tinguessin un punter al pares i als seus arrays (en comptes de que cada node ho tingués com a la v2). Com a error més general s'ha tingut molta paciència per comprovar que cada cop que s'actualitza la estructura d'alguna forma, tots els nodes peres, fills, avis, tiets... s'actualitzi.

7 Conclusions

Aquesta segona part de la pràctica, tot hi que ens ha semblat que era una carrega de treball bastant major a la primera, ha sigut una mica més entretinguda, ja que es tracta d'un concepte més interessant que els mètodes d'ordenació.
Apart de servir-nos molt per ajudar a consolidar els mètodes apresos a classe, ens ha semblat bastant entretingut el anar jugant amb diferents tipus de poda per aconseguir minimitzar els temps de solució del problema.
Ajuda també el fet de veure que amb un mateix algorisme pots solucionar problemes molt diferents, ja que t'obre els ulls en quant a les moltes funcions que podries arribar a donar-li en un futur, que no tinguin perquè ser relacionades amb trobar un camí o omplir una motxilla. En relació amb aquest fet, es interessant veure com per un problema o un altre els diferents algorismes es comporten millor o pitjor.

Un altre punt interessant, ha sigut veure com en alguns casos algun algorisme pot ser molt difícil, o més difícil que d'altres d'aplicar en determinats problemes o situacions, el qual ens acostuma a portar a fer una codificació no tant bona i acaba repercutint directament com hem pogut veure als gràfics.
Ens hagués agradat poder dedicar més temps a la memòria, però degut a certs problemes alhora de dur a terme el algorisme de hashtable, i a moltes altres entregues no ha sigut possible.

8 Bibliografia

AVL Tree. (n.d.).
www.cs.usfca.edu/ galles/visualization/AVLtree.html

AVL Tree | Set 1 (Insertion). (2019, March 18).
www.geeksforgeeks.org/avl-tree-set-1-insertion/

Definition of an AVL tree. (n.d.).
www.cs.auckland.ac.nz/software/AlgAnim/AVL.html

Tutorialspoint.com. (n.d.). Data Structures and Algorithms AVL Trees.
www.tutorialspoint.com/data$_s tructures_a lgorithms/avl_t ree_a lgorithm.htm$

Zhao, S., Zhao, S. (2018, January 27). AVL Trees Where To Find.
medium.com/@sarahzhao25/avl-trees-where-to-find-rotate-them-7b062e0a30f8

Graph - Linked Implementation,
www.cs.bu.edu/teaching/c/graph/linked/.
"ArrayList in Java." GeeksforGeeks, 26 Nov. 2018,
www.geeksforgeeks.org/arraylist-in-java/.

"Basics of Hash Tables Tutorials Notes | Data Structures." HackerEarth,
www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/.
Chandrakant, Kumar.

"Graphs in Java." Baeldung, 7 May 2019,
www.baeldung.com/java-graphs. gboeing, Author.

"R-Tree Spatial Indexing with Python." Geoff Boeing, 22 July 2017,
geoffboeing.com/2016/10/r-tree-spatial-index-python/. Gootooru, Nataraja.

"Program: Write a Program to Implement ArrayList." Java2Novice,
www.java2novice.com/java-interview-programs/arraylist-implementation/.

"Graph and Its Representations." GeeksforGeeks, 4 Oct. 2018,
www.geeksforgeeks.org/graph-and-its-representations/.

"Hashtable in Java." GeeksforGeeks, 28 Mar. 2019,
www.geeksforgeeks.org/hashtable-in-java/.

"How ArrayList Works Internally in Java." CodeNuclear, 9 Apr. 2019,
www.codenuclear.com/how-arraylist-works-internally-java/.

"Implementing Our Own Hash Table with Separate Chaining in Java." Geeks-
forGeeks, 9 Feb. 2018,

www.geeksforgeeks.org/implementing-our-own-hash-table-with-separate-chaining-in-java/. Morina, Fatos.

"Trie Data Structure in Java." Baeldung, 20 Dec. 2018, www.baeldung.com/trie-java.

"Online Training." Vogella.com, www.vogella.com/tutorials/JavaDatastructureList/article.html.

"R-Tree." Wikipedia, Wikimedia Foundation, 29 Apr. 2019, en.wikipedia.org/wiki/R-tree. Techie Delight.

"Implement Graph Data Structure in C." Techie Delight, 15 Oct. 2018, www.techiedelight.com/implement-graph-data-structure-c/.

"Trie | (Insert and Search)." GeeksforGeeks, 1 May 2019, www.geeksforgeeks.org/trie-insert-and-search/. V, Alexander, and Alexander V.

"Tries - Javascript Simple Implementation." Medium, Medium, 17 July 2015, medium.com/@alexanderv/tries-javascript-simple-implementation-e2a4e54e4330.

"QuickSort - GeeksforGeeks." Medium, Medium, 21 May 2015, https://www.geeksforgeeks.org/quick-sort/