Instasalle

Programació avançada i estructures de dades

By

Alex Almansa, Marc Llort

Departament d'enginyeria

La Salle URL

Desembre 2018

Contents

1 Codificació dels algorismes d'ordenació

En aquest apartat, s'explicarà una mica com ha estat la codificació en general dels diferents algorismes, i com nosaltres hem adaptat aquesta codificació per la pràctica. En primer lloc, s'explicarà la organització de la informació en les diferents classes.

1.1 Lector Json

En primer lloc, hem utilitzat la classe lectorJson per llegir el arxiu on hi ha la informació a ordenar. Aquesta classe llegirà l'arxiu especificat al principi de la mateixa i el posarà a un array de usuaris.

Aquest array contindrà cada un dels usuaris que hi ha al arxiu json que hem especificat anteriorment. Dins de cada usuari hi haurà la informació següent:

- Username [String] : Nom d'usuari
- Followers [int] : Numero de seguidors
- Follows [int] : numero de persones seguides
- Connections [Arraylist <Connections>] : Llista de totes les connections (persones seguides)

  Username [String] : Nom del usuari seguit

  Visits [int] : Numero de visites al seu perfil

  Likes [int] : Numero de likes a les seves publicacions

  Comments [int] : Numero de comentaris a les seves publicacions

  Order [double] : Variable auxiliar per ordenar les connexions
- Posts [Arraylist <Posts>]: Llista de tots els posts publicats

  Id [int]: Id de la publicació

  Published [int] : Data en format timestamp

  Location Arraylist <Float>: Coordenades de la localització

  Liked by [Arraylist <String>]: Llista de usuaris que han posat like

  Commented by [Arraylist <String>] : Llista de usuaris que han comentat

  Category [String] : Categoria a la que pertany la publicació

  Ordre [Double] : Variable auxiliar per ordenar els posts

  Point [Point] : Classe per gestionar les distancies entre coordenades

  User [String] : Variable auxiliar que contindrà el usuari al que pertany la publicació

1.1.1 Connections

En quant a les connections, cada connection tindrà la informació de la relació entre dos usuaris. Aquesta informació només serà utilitzada quan ens calgui ordenar segons la combinació de prioritats. En aquest cas, segons el usuari que ens indiquin, caldrà buscar el arraylist de connections d'aquell usuari. Un cop tenim el arraylist, donarem un valor a cada una de les connections i l'emmagatzemarem a la variable auxiliar order. En el nostre cas el valor de les connections l'hem calculat com:

$$Ordre = Likes * 0.1 + Comments * 0.2 + Visits * 0.3 \quad (1)$$

Hem decidit aquests valor en funció del que creiem que té més pes en una relació de Instagram.
Un cop tenim aquest valor, quan hem acabat de calcular tots els valors de totes les relacions dels usuaris, dividirem tots els valors entre el valor màxim per tenir-los tots en base 1.

1.1.2 Posts

Els posts serà el que farem servir per emmagatzemar els diferents posts i en el cas del array de cada usuari, tindrà els posts publicats per el mateix. Per les opcions de ordenar segons data i segons localització, es mostraran tots els posts de tots els usuaris, independentment de si es segueix al usuari en qüestió o no. Per fer això es recorrerà tots els usuaris i es posaran tots els posts de cada un a un array que posteriorment s'ordenarà. Es per això, que al utilitzar els datasets més grans, tarda una estona al principi, ja que ha de carregar tots els posts inicialment al arraylist.

En el cas de ser ordenat per la posició es farà us de la classe point, que conté dues coordenades i una funció que es "calcularDistanciaDesde" a la que se li passara un punt i ens retornarà la distància de la publicació a aquell punt.

Un cop tenim això, emmagatzemarem la distància en el cas que calgui ordenar la distància, o directament la data en format timestamp en el cas de la data, a la variable ordre per posteriorment ordenar els posts segons aquesta variable.

El User s'utilitzarà per al cas de ordenació per ordre de prioritats, en el que només es mostraran les publicacions dels usuaris que segueix el usuari en qüestió, i per tant, el que es farà serà crear un array amb totes les publicacions dels usuaris als que es segueix i gràcies al camp usuari, sabrem quina es la relació entre els dos, que la tindrem emmagatzemada a l'array de connections.

1.1.3 Point

En aquest cas, la classe point es tracta d'una classe molt simple que emmagatzema les coordenades x i y i té una única funcionalitat que és la de calcular la distància des de un punt a les coordenades del point. Això es fa mitjançant la següent formula:

$$Distncia = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (2)$$

1.2 Funcionalitats

Per tal d'ordenar les publicacions i poder reutilitzar les classes amb els diferents mètodes d'ordenació, la forma més fàcil amb la que hem pensat que podíem fer-ho és creant una variable auxiliar, dins els posts, que es diu ordre i a la que assignarem un valor o un altre en funció de segons què vulguem ordenar, d'aquesta manera, sempre ordenarem segons ordre.Per gestionar tot això hem creat la classe ordenador, que bàsicament s'encarrega de omplir la variable ordre i posteriorment cridar al mètode d'ordenació que convingui.Degut a aquesta forma de fer-ho prèviament a la ordenació en si, amb els datasets més grans, el programa tarda una estona en començar a ordenar ja que ha d'omplir totes les variables del arraylist de posts amb el ordre que correspongui. A continuació s'explicarà els processos per les diferents funcionalitats d'ordenació.

1.2.1 Temporalitat

Per dur a terme aquest mètode simplement s'ha agafat el numero del format timestamp, s'ha assignat la variable ordre a aquest numero i s'ha ordenat segons això.

1.2.2 Ubicació

Per dur a terme aquest mètode, en primer lloc es llegeix la ubicació des de la que es vol ordenar, un cop fet això, s'itera per totes les publicacions passant a cada publicació el punt entrat per l'usuari i assignant la distància a la variable ordre. Un cop fet això s'ordena per el mètode que sigui convenient segons la variable ordre.

1.2.3 Combinació de prioritats

En aquest cas, el procés és una mica més complex, ja que s'han de tenir en compte bastants factors per dur-ho a terme. Per tal de que es pugui balancejar fàcilment entre els 3 valors que s'han de tenir en compte, hem fet que els tres estiguin en base 1, es a dir que el més gran és 1 i el més petit 0, i hem afegit al inici de la classe ordenador tres constants que defineixen quin percentatge d'importància donem a cada un.

Relació En primer lloc, es busca el usuari per el que s'hauran d'ordenar les publicacions dins l'array d'usuaris. Un cop trobat l'usuari, es crea un arraylist on hi haurà tots els posts dels usuaris que el usuari segueix. Ja amb aquest array, es començaran a calcular el pes del primer dels tres factors a tenir en compte, la relació que té amb l'usuari del post. Aquest valor es calcula amb la fórmula esmentada anteriorment equació (1) , aquest valor serà el que marqui el ordre de la relació. Per tal d'incloure això agafarem el valor màxim i dividirem el valor de la relació entre el màxim de totes les relacions per fer que la relació més forta valgui 1, i les altres un valor proporcional entre 0 i 1.

Likehood El següent camp que es té en compte és el likehood que té el usuari amb la categoria, per fer això s'utilitza la classe Categories, que bàsicament té el nom de la categoria i el numero de likes que té. Amb un arraylist de Categories, es van afegint tots els likes que ha donat el usuari a totes les publicacions de manera que queda un arraylist amb el nom de les categories a les que el usuari ha donat like i el nombre de likes que ha donat a cada una. Amb aquesta informació, dividida un altre cop pel nombre màxim de likes donats a una categoria per tenir la informació en base 1, es suma la importància del likehood a la variable de ordre global.

Temporalitat Per últim, el darrer aspecte a tenir en compte es la temporalitat, es a dir quant fa que el post va ser publicat. Per calcular aquest valor, també en base 1 agafem directament el valor de "since" que té el post i el dividim entre el timestamp actual, així aconseguim que si fa molt temps que cap dels usuaris penja cap publicació, aquest camp perdrà importància automàticament ja que al dividir-se per el timestamp actual, els valors seran més petits.

1.2.4 Mètodes d'ordenació

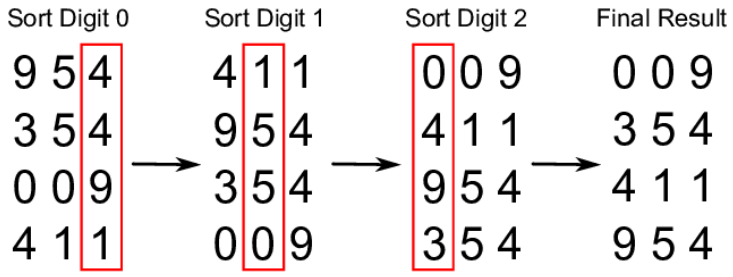
En aquest apartat explicarem com hem codificat cadascun dels diferents mètodes d'ordenació que hem fet servir. Ens centrarem sobretot en el radixSort, ja que es tracta de l'únic que no hem après a classe.

Selection Sort Aquest primer mètode és el més bàsic i ineficient (en la majoria d'escenaris amb cert nombre de informació) de tots.
El seu funcionament és: Itera sobre el array fins trobar el valor màxim o mínim, segons estigui configurat, i un cop trobat el situa al principi del array. El fet de que cada cop iteri tot el array és el que provoca que aquest sistema sigui molt lent per aquells casos on hi ha moltes dades a ordenar.

Quick Sort Es tracta d'un algoritme de dividir i conquerir. Agafa un element com el seu pivot i a partir d'aquest divideix el array. La finalitat d'aquest mètode és aconseguir posar tots els valors menors al pivot a l'"esquerra" del pivot i els més grans a la dreta.

Merge Sort Un algorisme de divide and conquer. Va dividint per la meitat el array fins tenir els elements sols. Un cop cada element esta sol, els anem ajuntant (merge) fent que cada cop que fem un merge entre els dos elements, quedi ordenat. Anirem ajuntant tots els elements de dos en dos fins arribar al array ordenat.

Radix Sort El radix sort, s'encarrega de ordenar digit a digit els diferents elements de l'array. Primer ordenarà segons el digit de menys valor, de més a menys o menys a més, segons com ho tinguem configurat. Posteriorment ordenarà el mateix array un alte cop, però segons el segon digit, i repetint així fins arribar al últim digit, on ja hu tindrem ordenat tot correctament.
El que és important entendre, és que en cada ordenació, en el cas de que dos digits siguin iguals, el primer que es trobi, és el que guardarem primer al array. D'aquesta manera conseguirem tenir, a la última volta, tot correctament ordenat.



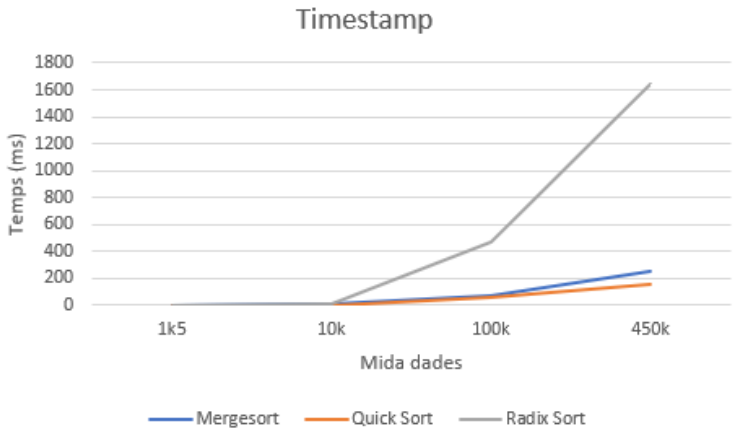
2 Mètodes de testeig

Un cop codificat cada mètode d'ordenació, per testejar cada un d'ells hem fet us tant dels datasets proporcionats, com un parell de datasets modificats que hem fet servir per testejar algun cas en específic.
En el cas de la temporalitat, miràvem que un cop ordenats els post, ens mostrés una serie de posts, juntament amb el seu timestamp per comprovar que estaven correctament ordenats.
En el cas de ordenar segons la ubicació hem fet servir els datasets estàndards, i anavem comprovant, a partir de la formula de la distància, si estaven ordenats de forma correcta.
En el últim cas, el dels interessos, connexions i temporalitat barrejats segons els criteris explicats anteriorment, hem fet us del dataset proporcionat especialitzat per aquest cas, ja que tot hi que vam intentar modificar un dataset per crear un cas específic nosaltres, no vam arribar a aconseguir crear-lo correctament.

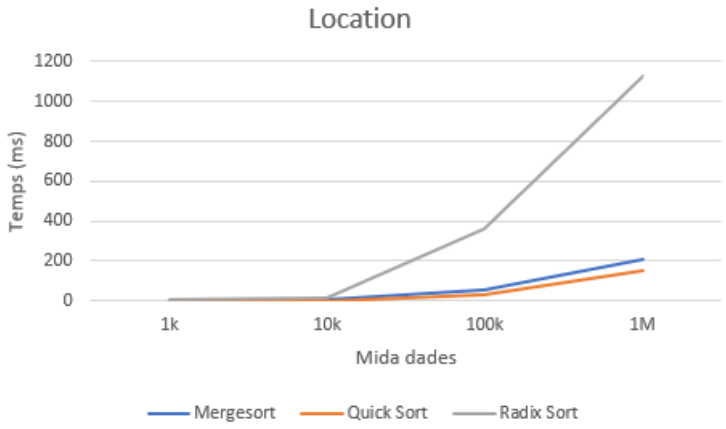
De forma més general, per ajudar-nos alhora de poder testejar de forma més ràpida i eficient, vam crear-nos un menú. Per tant podem ordenar per línia de comandes, o en cas de no escriure-hi res a la línia de comandes ens apareixerà el menú, on podrem escollir el mètode de ordenació, i que ordenar. Un cop seleccionats els dos paràmetres anteriors, ordenarà els posts, i ens preguntarà quants posts mostrar, per facilitar-nos així la visualització d'aquests.

3 Comparativa d'algorismes

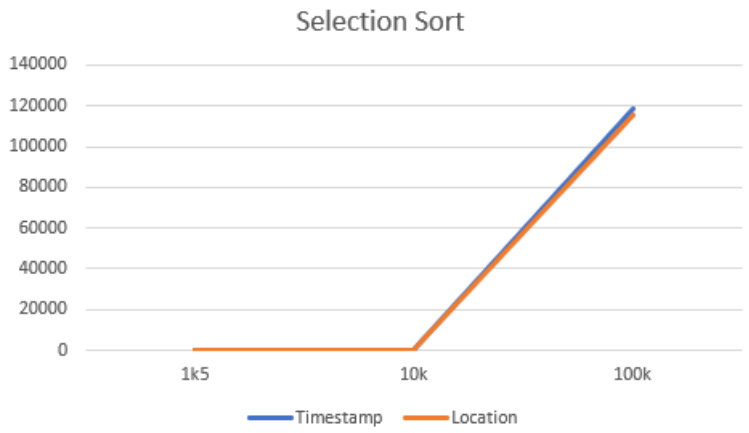
En aquesta secció tractarem de mostrar acompanyat de gràfiques la diferència entre L'eficiència d'uns i altres algorismes.



A la gràfica anterior es pot observar com, si ens fixem en el temps d'ordenació dels algorismes, hi ha una tendencia, que amb l'increment de les dades es fa cada cop més gran. El Radix Sort, va bastant per sobre del Merge Sort i el Quicksort i entre els dos el Quick Sort va una mica per sota de l'altre. El cas del selection sort, l'hem fet a una gràfica a part ja que el temps augmentava tant, que no permetia apreciar la diferència entre els altres 3. Aquesta primera gràfica mostra el que han tardat els diferents mètodes a ordenar segons el timestamp.



Aquesta gràfica representa exactament el mateix que la anterior però en aquest cas estem ordenant segons la localització en comptes de segons el temps, i es pot veure que les tendències son exactament les mateixes.



Tal com he comentat abans, el selection sort l'hem fet en un gràfic a part i només hem arribat fins la mida de 100 k ja que el temps ha passat de ser molt petit a pràcticament ser 2 minuts en els dos casos, tant en la ordenació per temps com en la ordenació per location.

Gràcies a aquestes gràfiques, podem observar la utilitat dels mètodes apresos a classe en quant a la eficiència dels mateixos amb dades grans, ja que tot i que amb dades petites poden comportar-se de forma semblant, al usar dades grans, veiem clarament que la diferencia creix molt.

4 Problemes Observats

En un principi vam tenir uns quants problemes amb el json, ja que els tipus de les diferents classes/tipus que estaven fer servir no coincidien amb els del json, concretament el timestamp.
Tampoc sabíem com configurar intelliJ per poder executar el programa passant-li els diferents arguments necessaris per córrer el programa en mode "línia comandes".
Els problemes més importants van ser com plantejar la ordenació segons els diferents tipus (temporalitat, localització, interessos) amb els diferents mètodes de ordenació (selection, merge, quick i radix). Finalment vam optar per posar una variable a cada post, anomenada ordre, la qual el mètode de ordenació llegirà i ordenarà a partir de la mateixa. Aquesta variable estarà calculada per cada post just abans de ordenar, per així ordenar segons el que calgui, ja sigui temporalitat, localització o interessos.

5 Conclusions

Aquesta pràctica ens ha servit per consolidar els diferents mètodes de ordenació apresos a classe, i agafar una mica més de agilitat alhora de programar en Java. Ens ha semblat molt interessant veure en quins casos funcionen millor o pitjor els diferents mètodes de ordenació, ja que trobem que ens pot ser molt útil en un futur professional. També ens ha obert molt els ulls, ja que fins ara no pensàvem que el "nostre" mètode de sempre, el selection, fos tant ineficient quan hi ha tantes dades, tot i que un cop vist com funcionen els altres mètodes ens sembla molt evident.
Per altra banda ha fet que canviem una mica la manera de pensar alhora de programar i ara, intentem esforçar-nos més per crear un codi més eficient.

Respecte a la memòria, hem fet us de Latex i ens ha semblat un gran programa el qual intentarem fer servir a partir d'ara. Tot hi no haver-hi aprofundit gaire ens ha semblat molt intuïtiu i fàcil d'aprendre a fer servir. Els resultats són molt millors que si estiguéssim fent servir word, ja que té un look molt més net i professional.

Respecte la pràctica, pensem que tot i que els mètodes de ordenació puguin ser no gaire entretinguts, al dur-los a terme en un context pràctic i real com és el de instagram, que tots fem servir a diari, ens ha ajudat molt a mantenir el interès i intentar millorar els temps que anàvem aconseguint.
Ens ha agradat el tenir la opció de escollir el llenguatge de programació a utilitzar, perquè ens permet millorar o investigar algunes funcionalitats que potser encara no havíem fet servir

6 Bibliografia

BAELDUNG. Quicksort Algorithm Implementation in Java. Baeldung [online]. 3 November 2018. [Accessed 11 December 2018].
Available from: https://www.baeldung.com/java-quicksort

Latex Expresiones Matematicas. Demostraciones Matematicas problemas ejercicios preguntas consultas dudas ayuda apoyo, tareas. Foros. Tex, Latex Editor. MathJax. Math help [online]. [Accessed 11 December 2018].
Available from: http://www.rinconmatematico.com/instructivolatex/formulas.htm

LaTeX Fácil. LaTeX Fácil: Guía rápida de LaTeX [online]. [Accessed 11 December 2018].
Available from: http://nokyotsu.com/latex/curso.html

Line breaks and blank spaces. Overleaf [online]. [Accessed 11 December 2018].
Available from: https://www.overleaf.com/learn/latex/Line$_b$$reaks_a$$nd_b$$lank_s$$paces$

Merge Sort. GeeksforGeeks [online]. 31 October 2018. [Accessed 11 December 2018].
Available from: https://www.geeksforgeeks.org/merge-sort/

Ordenamiento Radix. Algoritmos de Ordenamiento [online]. [Accessed 11 December 2018].
Available from: http://ict.udlap.mx/people/ingrid/Clases/IS211/Radix.html

QuickSort. GeeksforGeeks [online]. 4 December 2018. [Accessed 11 December 2018].
Available from: https://www.geeksforgeeks.org/quick-sort/

Radix Sort. GeeksforGeeks [online]. 5 September 2018. [Accessed 11 December 2018].
Available from: https://www.geeksforgeeks.org/radix-sort/

Radix sort. Wikipedia [online]. 4 December 2018. [Accessed 11 December 2018].
Available from: https://en.wikipedia.org/wiki/Radix$_s$$ort$

VOGEL, Lars. MergeSort how to. vogella.com [online]. [Accessed 11 December 2018].
Available from: http://www.vogella.com/tutorials/JavaAlgorithmsMergesort/article.html