Primera Entrega PROP
GRUP 42.2

Nora Caballero de Llanos
Lluc Hospital Escat
Richard Pie Sánchez
Ruben Rivera Villoldo

14/11/2022

Index

Clase Carpeta

Operacions:

```
//pre: null
//post: Retorna la ruta on està situada la carpeta.
public String getPath()

//pre: null
//post: La constructora ha creat un objecte carpeta amb el nom =
nombre_asignado, path = path_asignado.
public Carpeta(String nombre_asignado, String path_asignado)

//pre: null
//post: El mètode de la clase ha importat un document doc des de
la ruta proporcionada, old_string a la localització de la carpeta.
public void importa_document(Document doc, String old_string)

//pre: null
//post: El mètode ha afegit el objecte Document d al TreeMap
atribut de la carpeta.
private void doc_en_listado(Document d)

//pre:null
//post: El document amb el títol i autor específic s'elimina de la
carpeta estàndar i s'afegeix a la carpeta d'eliminats.
public void baixa_Document(String autor, String titol)

//pre: null
//post Retorna el nom de tots els títols que es troben al Treemap
public String get_titols (String autor)

//pre: null
//post: Retorna el nom de tots els autors que es troben al Treemap
public String get_autors (String prefix)

//pre:null
//post: Retorna el contingut del document amb l'autor i títol
indicats
public String get_contenido (String autor, String titol)


//pre: El document d existeix.
//post: Borra el document d de la posting list de cada una de les
paraules existents en el contingut de d.

private void eliminar_paraules_posting(Document d)
```

```
//pre: null
//post: S'ha modificat el contingut del document amb el autor i el
títol indicat.

public void modifica_document_en_carpeta (String autor, String
titol, String str)
```

Clase Document:

Operacions:

```
//pre: null
//post: La constructora ha creat un objecte de la classe document
amb atributs buits.
public Document()

//pre: null
//post: La constructora ha creat un objecte de la classe document
amb atributs títol = t,
autor = a, contingut = c, HashMap tf = tf.
public Document(String t, String a, String c, HashMap<String,
Integer> tf)

//pre:null
//post: Retorna la ruta del document a la funció des de la qual es
fa la crida.
public String getPath()

//pre:null
//post: Retorna el títol del document a la funció des de la qual
es fa la crida.
public String getTitol()

//pre:null
//post: Retorna el autor del document a la funció des de la qual
es fa la crida.
public String getAutor()


//pre:null
//post: Retorna el contingut del document al qual se li fa la
crida.
public String getContingut()
```

```
//pre:null
//post: Retorna el contingut del document al qual se li fa la
crida.
public void carregar_doc(String path_doc)

//pre:null
//post: Retorna el contingut del document al qual se li fa la
crida.
private HashMap<String,Integer> sort_hashmap(HashMap<String,Integer>
hashMap)


//pre: posting_list no buida i carpeta no buida
//post: El HashMap s'ha omplert amb les paraulas contingudes al
document traient el nombres, signes de puntuació i passant tot a
minúscules.

public void omple_TFHash(HashMap<String, List<Document>> pos_list,
Carpeta carpeta)

//pre: null
//post: En el cas de que la paraula es trobi al hashmap el
document s'ha afegit a la llista de documents.
En cas de que no es trobi la paraula, s'afegeix al hashmap i a mes
private void afegir_PostingList(HashMap<String, List<Document>>
posting_list, String paraula, String nom, String autor, Carpeta
carpet)

//pre: El autor i el document es troben a carpeta.
//post: Reemplaça el contingut del document.
public void modificacio_document(String str)
```

Clase Expressio:

Operacions:

```
//pre: null
//post: Creadora
public Expressions(){};
```

```
//pre: null
//post: Si el String x no existeix previament, s'afegeix al
vector<String> de la classe
public  void altaExpressio(String x)

//pre:null
//post:Si el String x existeix previamente, s'elimina del
vector<String> de la classe.

public void baixaExpressio(String x)

//pre:null
//post: En el cas de que el String vell existeix prèviament a la
crida i el string nou no existeix. Llavors, el string vell
s'elimina del vector i s'afegeix el String nou.
public void modifExpressio(String vell, String nou)

//pre:null
//post: Retorna el conjunt de documents després d'haver fet una
AND o OR entre el conjunt de documents de contingut i el conjunt
de documents de fin. Depen d les variables booleanas de and i or,
retornarà la unió(OR) dels documents o la intersecció(AND).
Set<Document> juntadoc(Boolean and, Boolean or,Set<Document>
contingut, Set<Document>fin)

//pre:null
//post:Retorna el conjunt de documents que compleixen amb els
paràmetres del Expressió booleana (String x)
public Set<Document> doc_expressions(String x,Carpeta carpeta)

//pre:null
//post:Retorna True si el vector<String> de la classe esta buit
public boolean isEmpty()
```

Classe Algorisme

```
//pre: existeix tw1 i tw2
//post: retorna un Double el qual és el resultat d'igualar els dos
hashmaps i els seus pesos segons les paraules iguals(similitud de
cosinus a partir de tf-idf).
Double similitud_per_cosinus(HashMap<String, Double> tw1,
HashMap<String, Double> tw2)

//pre:Document "doc" existent i carpeta "carpeta" existent
//post:retrona un Hashmap amb Key "paraula", que és un String del
conjunt de paraules de la "carpeta", i Value el pes de la paraula
sobre el document "doc".
HashMap<String, Double> tf_idf(Document doc, Carpeta carpeta)
```

```
//pre: HashMap “tw” existent i no buit
//post:Retorna un HashMap normalitzat
HashMap<String, Double> normalitzar(HashMap<String, Double> tw)

//pre: “carpeta” i “d” existents i no buits
//post:Retorna un TreeMap Key, double = similitud per cosinus de
document no repetit, Value, List<Document> = tots els documents
que tenen similitud per cosinus igual a la Key.
TreeMap<Double, List<Document>> similitud_doc(Carpeta carpet,
Document d)

//pre: “paraules” no buit i “carpeta” existent i no buida
//post: Retorna un set de Documents els quals contenen totes les
paraules de l’Array paraules.Set<Document>
expresions_boleanes(String[] paraules, Carpeta carpeta)
```

ESTRUCTURA DE DADES

Per la realització de l'algorisme hem utilitzat HashMaps ja que ens ha semblat la manera més eficient de realitzar l'algoritme. Vam utilitzar aquesta estructura en comptes d'un TreeMap(estructura més propera) per les següents raons:

- No necessitat de moure una gran massa de dades, estem parlant del contingut d'un sol document(perquè sinó igualment ho converteix en TreeMap de manera interna)
- Les operacions bàsiques son menys costoses que si es realitzen amb un TreeMap ja que les funcions get(), contains() i put() en un HashMap son molt poc costoses 0(1) en comparació amb el TreeMap que el seu cost és de 0(log(n)).
- La no necessitat d'ordenació per defecte que fa l'estructura de dades més costosa i seria un cost repetitiu. És a dir ens trobem amb la necessitat d'ordenar per Valor fet que si ja haguem ordenat per key estarem reordenant i fent més costosa l'operació.
- El rendiment és clau a l'hora de fer els algorismes més ràpids

Els hem utilitzat de dues formes. Primer de tot la trobem en cada document per guarda la informació del contingut type-freq i d'altra banda en la classe Carpeta per tenir un repositori de paraula-documents que la contenen. És a dir:
TF:

➔ Key: String paraula

➔ Value: Integer freq(en aquell document)

POSTING LIST:

➔ Key: String paraula

➔ Value: List<Documents> documents que contenen "paraula" al seu contingut

Per guardar les els nostres documents en la classe Carpeta_estandard i Carpeta_eliminats en aquest cas utilitzem un TreeMap de TreeMap(TreeMap<String,TreeMap<String, Document>>, les raons per les quals en aquest cas hem decidit utilitzar aquesta estructura és la següent:

- Tenim ordenats els autors i títols en pordre natural d'String, és a dir, en ordre alfabètic, de manera que a l'hora de treure títols, autors,... ho tenim ordenat com es veu millor per l'usuari.
- Tenim la possibilitat de fer una búsqueda de l'autor i una búsqueda del títol fet que els fa la búsqueda més ràpida que buscar en un vector de Documents, en el nostre cas la búsqueda seria de 0(log(n))+0(log(m)) i en cas de un vector deria de 0(n).
- La possibilitat de tenir el document identificat per autor i títol gràcies als get() i no de tenir un identificador i i després comprovar que sigui aquell document
- La possibilitat de tenir tanta memòria com necessitem

La utilització en les dos carpetes ens serveix com a estructura per guardar els documents creats i funciona de tal manera.

LLISTAT_DOCS:

➔ Key: String autor del/s document/s

➔ Value:

◆ String títol del document

◆ Value: Objecte Document

ALGORISMES

En aquest cas el primer algoritme d'espai vectorial juntament amb el la similitud de cosinus ho hem realitzat de la seguent manera:

1. Cada document omple una taula de Hash pròpia on es guarda: paraula, frequència amb un cost 0(1). Al mateix temps s'emplena un HashMap global (Inverted File, explicat en el següent algorisme)on guardem: paraula, documents que contenen la paraula amb un cost 0(1)
2. Per el document seleccionat es realitza el tf-idf. Aquest el que fa és calcular els pesos de les paraules(per veure si són rellevants en el text o no) del document segons el conjunt de documents que tenim a la nostra carpeta a través del tf (term frequency) i el idf (inverse document frequency, per veure si un document és molt comú o estrany en el conjunt de documents).
3. Es realitza el tf-idf un per un de cada document de la carpeta i realitzant la similitud per cosinus, que segons si la dirección del vector de pesos del document escollit primerament i el document actual son més o menys semblants, el text també serà més o menys semblant.

En el segon algorisme, un algorisme de búsqueda de paraules en el conjunt de documents realitzem les següents tasques:

1. Gràcies a l'Inverted File, un HashMap construit a partir de: (paraula, llista de documents d'aquella paraula) podem fer, per cada llistat de paraules entre AND, OR, !, {},realitzem una búsqueda d'una paraula "p" en l'Inverted File i retornar un set de documents en els quals es troba aquella paraula.

2. Segons aquest set i l'organització de l'expressió booleana donada, realitzem una unió del set de documents actual amb l'anterior, en cas de ser una OR, o una resta de la intersecció de documents del set de documents actual amb l'anterior, en cas de ser una AND.
3. Realitzem un altre cop el mateix fins a obtenir un set de documents final en el qual compleix tota l'expressió.

Aquesta últim algoritme el realitzem de tal manera ja que las búsquedes en els HashMaps son de cost 0(1) + els cost de recorre la llista de documents 0(x) sempre hi quan no hi hagi moltes col·lisions, en canvi si miressim de recorre tot el quan un string amb el contains()... ens tardaria 0(n) per cada un dels documents m.

Repartiment de classes

- Classe Document: Lluc
- Classe Algoritme: Lluc
- Classe Expressions: Nora
- Classe Carpeta: Richard, Ruben i Lluc
- Classe ControladorDomini: Richard, Ruben i Lluc
- Classe Main: Richard, Ruben i Lluc