Πανεπιστήμιο Πειραιώς

Ανάκτηση Πληροφοριών

Search Engine

Σπυρόπουλος Ιωάννης e14175
6ο εξάμηνο

Περιεχόμενα

1.Περιγραφή

Η εφαρμογή είναι γραμμένη σε Java και υλοποιεί μια μηχανή αναζήτησης λέξεων σε δοθέντα αρχεία. Δίνει επίσης την δυνατότητα σύνθετης αναζήτησης καθώς επιτρέπει την χρήση του AND και του OR για τον συνδυασμό αποτελεσμάτων

2.Πηγαίος Κώδικας

2.1.Γενική εικόνα

Η εφαρμογή αποτελείται από τέσσερις κλάσεις.

1. InfoRetrieval οπου περιέχει την μέθοδο main
2. Indexer όπου υλοποιεί το ανεστραμμένο ευρετήριο
3. Searcher όπου υλοποιεί την αναζήτηση
4. BoolQuery όπου υλοποιεί την αναζήτηση με AND και OR

2.2.Ανάληση κώδικα

2.2.1.InfoRetrieval

Αρχικά έχουμε την μέθοδο main. Στην συνέχεια δημιουργούμε ένα αντικείμενο Indexer και του δίνουμε ως παράμετρο τον πίνακα args όπου αντιστοιχεί στα ονόματα των αρχείων που έχουμε περάσει με την μορφή παραμέτρων κατά το τρέξιμο της εφαρμογής.

```
public static void main(String[] args) {

            /*Index the given files*/
            Indexer indexer = new Indexer(args);
```

Έτσι μετά το πέρας της εντολής θα φτιαχτεί το ανεστραμμένο ευρετήριο με βάση τα αρχεία που έχουμε δώσει.

Στην συνέχεια ταξινομούμε το ευρετήριο.

```
/*Sort the inverted index*/
            indexer.index = indexer.sortIndex();
```

Ακολουθούν οι εντολές για την αναζήτηση.

```
/*Search*/
            System.out.print("Search:");

            Searcher searcher = new Searcher(indexer);
```

Η πρώτη εντολή τυπώνει στο τερματικό την προτροπή “Search:” ώστε ο χρήστης να δώσει τα στοιχεία της αναζήτησης.
Η δεύτερη εντολή δημιουργεί ένα αντικείμενο τύπου Search και περνάει ως παράμετρο το ανεστραμμένο ευρετήριο που μόλις φτιάχτηκε.

Με τις εντολές που ακολουθούν διαβάζονται οι παράμετροι αναζήτησης που θα δώσει ο χρήστης και αποθηκεύονται ανά λέξη σε έναν πίνακα χαρακτήρων.

```
/*Read querys from user*/
            Scanner s = new Scanner(System.in);
            String[] querys = s.nextLine().split("\\W+");
            s.close();
```

Αμέσως μετά διατρέχουμε τον πίνακα και μετατρέπουμε όλες τις λέξεις σε μικρά γράμματα εκτός των AND και OR.

```
/*Convert querys to lower case*/
            for(int i=0;i<querys.length;i++){
                  if(!(querys[i].equals("AND") || querys[i].equals("OR")))
                        querys[i] = querys[i].toLowerCase();
            }
```

Τέλος δημιουργούμε μια μεταβλητή answer όπου εκεί αποθηκεύεται η απάντηση στο ερώτημα του χρήστη. Στην συνέχεια βλέπουμε αν το ερώτημα περιέχει τις εντολές AND ή OR. Εάν δεν τις περιέχει καλούμε την απλή αναζήτηση για το ερώτημα ενώ αν τις περιέχει τότε καλούμε την αναζήτηση που υπολογίζει και την σύζευξη ή την διάζευξη.

```
/*Simple search or with boolean*/
List<Integer> answer = new ArrayList<Integer>();
if(searcher.hasBool(querys)){
      answer = searcher.boolSearch(querys);
      for(String query : querys){
            System.out.print(query + " ");
      }
      System.out.print(":" + answer);
}else{
      for(String query : querys){
            answer = searcher.search(query);
```

```
            if(answer == null){
                  System.out.println("Word " + "'" + query + "' " + "not
found!");
            }else{
                  System.out.println(query + ":" + answer);
            }
      }
}
```

2.2.2.Indexer

Στην κλάση Indexer αρχικά δημιουργύμε μια μεταβλητή index όπου εκεί θα αποθηκεύσουμε το ανεστραμμένο ευρετήριο.

```
public class Indexer {
      public Map< String,List<Integer> > index = new HashMap<
String,List<Integer> >();
```

Το ευρετήριο είναι τύπου Map<Stirng , List<Integer>> καθώς αποθηκεύει σε αλφαριθμητική μορφή την κάθε λέξη και σε λίστα ακεραίων τα id των αρχείων στα οποία βρίσκεται.

Έπειτα έχουμε τον κατασκευαστή της κλάσης ο οποίος καλεί και την συνάρτηση parseFiles.

```
public Indexer(String[] files) {
            parseFiles(files);
      }
```

Η συνάρτηση parseFiles είναι τύπου void και ο ρόλος της είναι να διατρέχει τα αρχεία που δόθηκαν από σαν όρισμα και με βάση την συνάρτηση index να δημιουργεί το ανεστραμμένο ευρετήριο.

```
private void parseFiles(String[] files){
            String line = null;
            int docId = 0;
            //read all the files one by one
            for(String file : files){
                  try{
                        //read the file
                        BufferedReader bf = new BufferedReader(new
FileReader(file));
                        System.out.println("Indexing " + file + " ...");
```

```
                        docId++;

                        while((line = bf.readLine()) != null){

                              String[] words = line.split("\\W+");//take
only the words
                              for(String word : words){
                                    word = word.toLowerCase();//make all
word lower case in the dictionary
                                    index(word,docId);
                              }
                        }
                        bf.close();
                  }catch (FileNotFoundException e) {
                        System.out.println("Cant open file "+file);
                        e.printStackTrace();
                  } catch (IOException e) {
                        System.out.println("Error reading file "+file);
                        e.printStackTrace();
                  }
            }
      }
```

Στην συνέχει ακολουθεί η συνάρτηση index όπου παίρνει μια λέξη σαν όρισμα κάθε φορά και την αρχειοθετεί κατάλληλα στο ανεστραμμένο ευρετήριο Αν η λέξη υπάρχει προσθέτει στην λίστα με τα id των αρχείο το id του αρχείου από το οποίο προέρχεται η λέξη. Αν η λέξη δεν υπάρχει στο ευρετήριο τότε την τοποθετεί και την συνδέει με μια λίστα όπου περιέχει μόνο το id του αρχείου από το οποίο προήλθε.

```
private void index(String word, int docId){
            //TODO add position of word
            if(index.get(word) == null){//if word isn't in dictionary
                  List<Integer> docs = new ArrayList<Integer>();
                  docs.add(docId);
                  index.put(word, docs);//add the word in the dictionary
            }else{
                  index.get(word).add(docId);//add the document in the list
of documents
            }
      }
```

Τέλος έχουμε την μέθοδο sortIndex όπου μετατρέπει το ευρετήρια από τύπου Map σε TreeMap με αποτέλεσμα να ταξινομηθεί καθώς το TreeMap είναι ταξινομημένο

```
public Map<String,List<Integer>> sortIndex(){
            return new TreeMap<String, List<Integer> >(index);
}
```

2.2.3.Searcher

Αρχικά έχουμε τον κατασκευαστή της κλάσης.

```
private Indexer indexer;

/*Constructor*/
public Searcher(Indexer _indexer) {
      this.indexer = _indexer;
}
```

Στην συνέχεια έχουμε την μέθοδο hasBool όπου διατρέχοντας τον πίνακα με τα ερωτήματα του χρήστη ελέγχει αν περιέχονται οι παράμετροι AND ή OR.

```
public Boolean hasBool(String[] querys){
      for(String query : querys){
            if(query.equals("AND") || query.equals("OR")){
                  return true;
            }
      }
      return false;
}
```

Έπειτα ακολουθεί η συνάρτηση search. Η search παίρνει σαν όρισμα μία λέξη και ψάχνει στο ευρετήριο αν υπάρχει. Αν υπάρχει επιστρέφει την λίστα των id των εγγράφων αλλιώς επιστρέφει null.

```
public List<Integer> search(String query){
      if(indexer.index.get(query) == null){
            return null;
      }else{
            return indexer.index.get(query);
      }
}
```

Τέλος έχουμε την μέθοδο boolSearch όπου υλοποιεί την αναζήτηση στην περίπτωση που υπάρχουν και οι τελεστές AND ή OR.
Παίρνει ως όρισμα τον πίνακα με τα ρωτήματα που έχει δώσει ο χρήστης. Στην συνέχεια διατρέχει τον πίνακα ανά στοιχείο. Αν το παρόν στοιχείο είναι ένας από τους τελεστές τότε δημιουργεί ένα νέο αντικείμενο τύπου BoolQuery. Ως παραμέτρους στο αντικείμενο ορίζουμε το αποτέλεσμα την μεθόδου search για το ερώτημα που προηγείται και για το ερώτημα που έπεται του τελεστή. Τέλος η συνάρτηση επιστρέφει το αποτέλεσμα της μεθόδου calcAND αν υπάρχει ο τελεστής AND και calcOR αν υπάρχει ο τελεστής OR. Αν τα ερωτήματα δεν υπάρχουν τότε επιστρέφεται null.

```
public List<Integer> boolSearch(String[] querys) {
            for(int i=0;i<querys.length;i++){
                  if(querys[i].equals("AND")){
                        BoolQuery bq = new
BoolQuery(search(querys[i-1]),search(querys[i+1]));
                        return bq.calcAND();
                  }else if (querys[i].equals("OR")){
                        BoolQuery bq = new
BoolQuery(search(querys[i-1]),search(querys[i+1]));
                        return bq.calcOR();
                  }

            }
            return null;
      }
```

2.2.4.BoolQuery

Τέλος έχουμε την κλάση BoolQuery όπου αρχικά περιέχει δύο μεταβλητές list1 και list2 όπου αντιστοιχούν στις λίστες με τα id των εγγράφων , και τον κατασκευαστή

```
private List<Integer> list1;
private List<Integer> list2;

public BoolQuery(List<Integer> list1, List<Integer> list2) {
      super();
      this.list1 = list1;
      this.list2 = list2;
}
```

Στην συνέχεια έχουμε τις δύο μεθόδους calcAND και calcOR όπου υπολογίζουν την σύζευξη ή την διάζευξη των δύο λιστών αντίστοιχα.

```
public List<Integer> calcAND(){
	int i = 0,j = 0;
	List<Integer> answer = new ArrayList<Integer>();
	while(i < list1.size() && j < list2.size()){
		if(list1.get(i) == list2.get(j)){
			answer.add(list1.get(i));
			i++;
			j++;
		}else{
			if(i < j){
				i++;
			}else{
				j++;
			}
		}
	}
	return answer;
}
```

Η calcAnd αποτελεί υλοποίηση του αλγορίθμου σύζευξης.

INTERSECT(p_1, p_2)
1 *answer* ← ⟨ ⟩
2 **while** $p_1 \neq$ NIL and $p_2 \neq$ NIL
3 **do if** $docID(p_1) = docID(p_2)$
4 **then** ADD($answer, docID(p_1)$)
5 $p_1 \leftarrow next(p_1)$
6 $p_2 \leftarrow next(p_2)$
7 **else if** $docID(p_1) < docID(p_2)$
8 **then** $p_1 \leftarrow next(p_1)$
9 **else** $p_2 \leftarrow next(p_2)$
10 **return** *answer*

```
public List<Integer> calcOR(){
int i=0,j=0;
List<Integer> answer = new ArrayList<Integer>();
while(i< list1.size() && j < list2.size()){
            if(list1.get(i) == list2.get(j)){
                  answer.add(list1.get(i));
            }else{
                  answer.add(list1.get(i));
                  answer.add(list2.get(j));
            }
            i++;
            J++;
}
return answer;
}
```

3.Χρήση Εφαρμογής

3.1.Τρέξιμο της εφαρμογής

Για να τρέξουμε την εφαρμογή αρκεί να δώσουμε την εντολή :

```
java InfoRetrieval <paths/to/files>
```

Το αρχείο InfoRetrieval βρήασκεται μέσα στον φάκελο bin.

3.2.Παράδειγμα χρήσης

Για τα πλαίσια δοκιμής της εφαρμογής χρησιμοποιούμε δύο αρχεία που περιέχουν απλό κείμενο.

```
• • •
• • • cat ~/tmp/file1.txt
Hello this is a new file for testing.
• • • cat ~/tmp/file2.txt
ok another file for testing too!.
• • •
• • •
```

Τρέχουμε την εφαρμογή δίνοντας ως ορίσματα τα δύο αυτά αρχεία.

```
• • • ls
BoolQuery.class  Indexer.class  InfoRetrieval.class  Searcher.class
• • • java InfoRetrieval ~/tmp/file1.txt ~/tmp/file2.txt
Indexing /home/cre8/tmp/file1.txt ...
Indexing /home/cre8/tmp/file2.txt ...
Search:
```

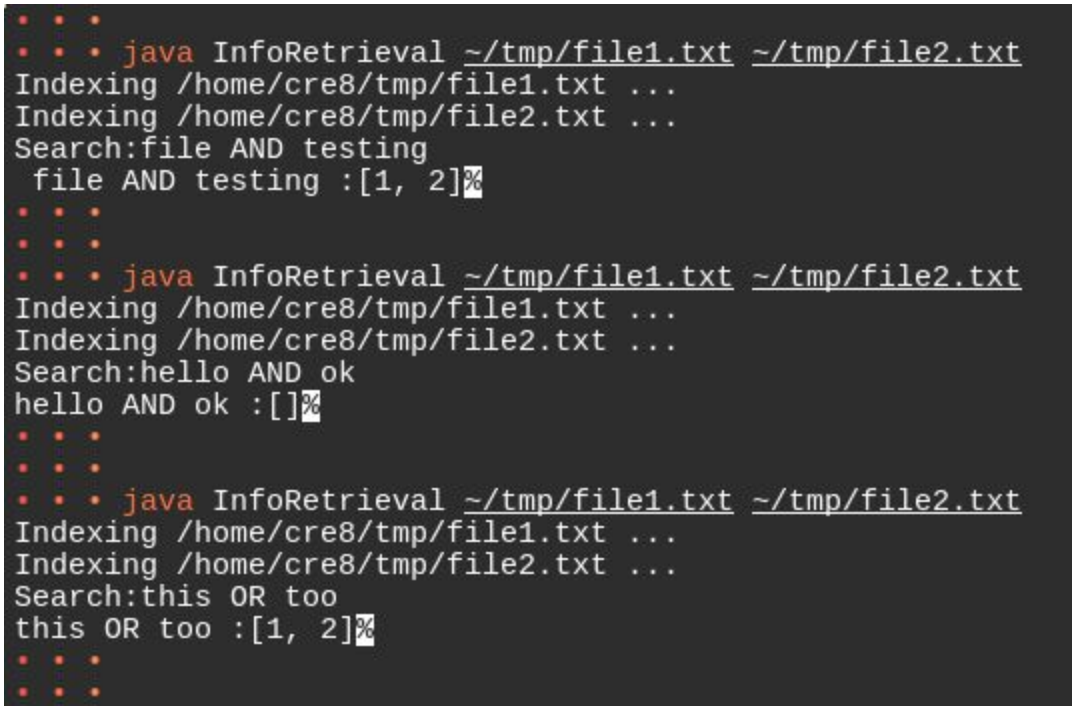
Όπως βλέπουμε δημιουργήθηκε το ανεστραμμένο ευρετήριο με βάση τα δύο αυτά αρχεία και μπορούμε να πληκτρολογήσουμε το ερώτημα μας.

Αρχικά μπορούμε να δώσουμε σαν ερώτημα τις λέξεις που θέλουμε να ψάξουμε μέσα στα αρχεία.

```
• • •
• • • java InfoRetrieval ~/tmp/file1.txt ~/tmp/file2.txt
Indexing /home/cre8/tmp/file1.txt ...
Indexing /home/cre8/tmp/file2.txt ...
Search:hello file too
hello:[1]
file:[1, 2]
too:[2]
• • •
```

Η απάντηση που παίρνουμε είναι η κάθε λέξη που ψάξαμε και η λίστα με τα αρχεία στα οποία βρίσκεται.
Εδώ για παράδειγμα η λέξη hello βρίσκεται στο πρώτο αρχείο η λέξη file βρίσκεται και στα δύο αρχεία και τέλος η λέξη too βρίσκεται στο δεύτερο αρχείο.

Τέλος μπορούμε στο ερώτημα μας να συμπεριλάβουμε και τους τελεστές AND ή OR.



4.Επίλογος

Η εφαρμογή υλοποιεί τα βασικά στοιχεία μιας μηχανής αναζήτησης. Σε καμία περίπτωση η λειτουργικότητα της δεν θεωρείται πλήρης. Έχοντας όμως τον βασικό κορμό μπορούμε να επεκτείνουμε την λειτουργικότητά της υλοποιώντας την υποστήριξη πιο σύνθετων αναζητήσεων, ranking, ορθογραφικό έλεγχο και άλλα.