UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

M2 ILSEN – 2019/20

UE Ingénierie du document et de l'information

UCE Indexation & recherche

Vincent Labatut

TP 3 | Correction

1 Opérateur ET

Exercice 1

```
1  private void splitQuery(String query, List<List<Posting>> result)
2  {  // on tokénize la requête
3     Tokenizer tokenizer = index.getTokenizer();
4     List<String> types = tokenizer.tokenizeString(query);
5
6     // on normalise chaque type
7     Normalizer normalizer = index.getNormalizer();
8     for(String type: types)
9     {  // la normalisation du type donne le terme (ou null)
10       String term = normalizer.normalizeType(type);
11       if(term!=null)
12       {  // on récupére l'entrée associée au terme dans l'index
13          IndexEntry entry = index.getEntry(term);
14          // si pas dans l'index, on utilise une liste vide
15          if(entry==null)
16             result.add(new ArrayList<Posting>());
17          // sinon, on prend sa liste de postings
18          else
19          {  List<Posting> postings = entry.getPostings();
20             result.add(postings);
21          }
22       }
23    }
24 }
```

Exercice 2

```
1  private List<Posting> processConjunction(List<Posting> list1, List<Posting> list2)
2  {  List<Posting> result = new LinkedList<Posting>();
3     Iterator<Posting> it1 = list1.iterator();
4     Iterator<Posting> it2 = list2.iterator();
5
6     // on fusionne le début des listes
7     Posting posting1 = null;
8     Posting posting2 = null;
9     while((it1.hasNext() || posting1!=null) && (it2.hasNext() || posting2!=null))
10    {  if(posting1==null)
11          posting1 = it1.next();
12       if(posting2==null)
13          posting2 = it2.next();
14       int comp = posting1.compareTo(posting2);
15       // posting1 < posting2
16       if(comp<0)
17          posting1 = null;
18       // posting1 == posting2
19       else if(comp==0)
```

```
20          {  result.add(posting1);
21             posting1 = null;
22             posting2 = null;
23          }
24          // posting1 > posting2
25          else if(comp>0)
26             posting2 = null;
27       }
28
29       return result;
30    }
```

Exercice 3

```
1  public int compare(List<Posting> l1, List<Posting> l2)
2  {  int result = l1.size() - l2.size();
3     return result;
4  }
```

Exercice 4

```
1  private List<Posting> processConjunctions(List<List<Posting>> lists)
2  {  // on ordonne la liste de postings
3     Collections.sort(lists, COMPARATOR);
4
5     // on traite les deux premières
6     List<Posting> list1 = lists.get(0);
7     lists.remove(0);
8     List<Posting> list2 = lists.get(0);
9     lists.remove(0);
10    List<Posting> result = processConjunction(list1, list2);
11
12    // on traite chaque liste restante une par une
13    Iterator<List<Posting>> it = lists.iterator();
14    while(it.hasNext() && !result.isEmpty())
15    {  List<Posting> list = it.next();
16       result = processConjunction(result, list);
17    }
18
19    return result;
20 }
```

Exercice 5

```
1  public List<Posting> processQuery(String query)
2  {  System.out.println("Processing query \""+query+"\"");
3     long start = System.currentTimeMillis();
4
5     // on décompose la requête et identifie les termes
6     List<List<Posting>> postings = new LinkedList<List<Posting>>();
7     splitQuery(query,postings);
8     //System.out.println(postings);
9
10    // on traite les opérateurs ET
11    List<Posting> result;
12    if(postings.size()==1)
13       result = postings.get(0);
14    else
15       result = processConjunctions(postings);
16
17    long end = System.currentTimeMillis();
```

```
18      System.out.println("Query processed, returned "+result.size()+"postings,
            duration="+(end-start)+" ms");
19      return result;
20  }
```

Exercice 6

```
1   private static void testQuery() throws IOException, ClassNotFoundException
2   {  List<String> queries = Arrays.asList(
3         "recherche",
4         "recherche INFORMATION",
5         "recherche INFORMATION Web"
6      );
7
8      // chargement de l'index
9      AbstractIndex index = AbstractIndex.read();
10
11     // résolution de la requête
12     AndQueryEngine engine = new AndQueryEngine(index);
13     for(String query: queries)
14     {  List<Posting> result = engine.processQuery(query);
15        System.out.println("Files:
              \n"+FileTools.getFileNamesFromPostings(result)+"\n");
16     }
17  }
```

Exercice 7

On obtient les temps suivants :

- Tableau : 2 016 ms
- Table de hachage : 1 110 ms
- Arbre : 1 508 ms

L'accès à la table de hachage se fait en temps constant (en moyenne), alors que pour les deux autres structures de données, il est logarithmique, ce qui explique les différences observées.

2 Opérateur OU

Exercice 9

```
1   private void splitOrQuery(String query, List<List<List<Posting>>> result)
2   {  String[] strings = query.split(OR);
3
4      // on nettoie chaque sous-chaîne obtenue
5      for(String string: strings)
6      {  List<List<Posting>> list = new LinkedList<List<Posting>>();
7         splitAndQuery(string,list);
8         result.add(list);
9      }
10  }
```

Exercice 10

```
1   private List<Posting> processDisjunction(List<Posting> list1, List<Posting> list2)
2   {  List<Posting> result = new LinkedList<Posting>();
3      Iterator<Posting> it1 = list1.iterator();
4      Iterator<Posting> it2 = list2.iterator();
5
6      // on fusionne le début des listes
7      Posting posting1 = null;
8      Posting posting2 = null;
9      while((it1.hasNext() || posting1!=null) && (it2.hasNext() || posting2!=null))
```

```
10      {  if(posting1==null)
11            posting1 = it1.next();
12         if(posting2==null)
13            posting2 = it2.next();
14         int comp = posting1.compareTo(posting2);
15         // posting1 < posting2
16         if(comp<0)
17         {  result.add(posting1);
18            posting1 = null;
19         }
20         // posting1 == posting2
21         else if(comp==0)
22         {  result.add(posting1);
23            posting1 = null;
24            posting2 = null;
25         }
26         // posting1 > posting2
27         else if(comp>0)
28         {  result.add(posting2);
29            posting2 = null;
30         }
31      }
32
33      // on rajoute la valeur éventuellement présente en tampon
34      if(posting1!=null)
35         result.add(posting1);
36      else if(posting2!=null)
37         result.add(posting2);
38
39      // on rajoute la fin de la liste restante
40      Iterator<Posting> it = null;
41      if(it1.hasNext())
42         it = it1;
43      else if(it2.hasNext())
44         it = it2;
45      if(it!=null)
46      {  while(it.hasNext())
47         {  Posting posting = it.next();
48            result.add(posting);
49         }
50      }
51
52      return result;
53   }
```

Exercice 11

```
1    private List<Posting> processDisjunctions(List<List<Posting>> postings)
2    {  // on ordonne la liste de listes de postings
3       Collections.sort(postings, COMPARATOR);
4
5       // on traite les deux premières
6       List<Posting> list1 = postings.get(0);
7       postings.remove(0);
8       List<Posting> list2 = postings.get(0);
9       postings.remove(0);
10      List<Posting> result = processDisjunction(list1, list2);
11
12      // on traite chaque liste restante une par une
```

```
13      for(List<Posting> list: postings)
14         result = processDisjunction(result, list);
15
16      return result;
17   }
```

Exercice 12

```
1    public List<Posting> processQuery(String query)
2    {  System.out.println("Processing query \""+query+"\"");
3       long start = System.currentTimeMillis();
4
5       // on décompose la requête et identifie les termes
6       List<List<List<Posting>>> postings = new LinkedList<List<List<Posting>>>();
7       splitOrQuery(query,postings);
8       //System.out.println(postings);
9
10      // on traite les opérateurs ET
11      List<List<Posting>> partialResults = new LinkedList<List<Posting>>();
12      for(List<List<Posting>> list: postings)
13      {  List<Posting> partialResult;
14         if(list.size()==1)
15            partialResult = list.get(0);
16         else
17            partialResult = processConjunctions(list);
18         partialResults.add(partialResult);
19      }
20
21      // on traite les opérateurs OU
22      List<Posting> result;
23      if(postings.isEmpty())
24         result = new ArrayList<Posting>();
25      else if(partialResults.size()==1)
26         result = partialResults.get(0);
27      else
28         result = processDisjunctions(partialResults);
29
30      long end = System.currentTimeMillis();
31      System.out.println("Query processed, duration="+(end-start)+" ms");
32      return result;
33   }
```

Exercice 13

```
1    private static void testQuery() throws IOException, ClassNotFoundException
2    {  List<String> queries = Arrays.asList(
3          "project,SOFTWARE,Web,pattern,computer",
4          "project SOFTWARE Web pattern computer",
5          "project SOFTWARE Web,pattern computer"
6       );
7
8       // chargement de l'index
9       AbstractIndex index = AbstractIndex.read();
10
11      // résolution de la requête
12      AndOrQueryEngine engine = new AndOrQueryEngine(index);
13      for(String query: queries)
14      {  List<Posting> result = engine.processQuery(query);
15         System.out.println("Result: "+result.size()+" document(s)\n"+result);
16         System.out.println("Files: \n" +
```

```
            FileTools.getFileNamesFromPostings(result)+"\n");
17      }
18  }
```