M2 ILSEN – 2019/20
UE Ingénierie du document et de l'information
UCE Indexation & recherche
Vincent Labatut

TP 2 | Correction

1 Construction

Exercice 1

```java
private int filterTokens(List<Token> tokens)
{  int result = 0;
   // on passe chaque paire de tokens consécutifs en revue
   Iterator<Token> it = tokens.iterator();
   Token t1 = null;
   while(it.hasNext())
   {  Token t2 = it.next();
      if(t1==null)
         result = 1;
      else
      {  // si deux tokens consécutifs sont identiques,
         // on supprime le second
         if(t1.equals(t2))
            it.remove();
         // on compte les termes
         String type1 = t1.getType();
         String type2 = t2.getType();
         if(!type1.equals(type2))
            result++;
      }
      t1 = t2;
   }
   return result;
}
```

Exercice 2

```java
private int buildPostings(List<Token> tokens, AbstractIndex index)
{  int result = 0;
   int i = 0;
   IndexEntry entry = null;
   // on traite chaque token séparément
   for(Token token: tokens)
   {  String type = token.getType();
      // si besoin, on crée une nouvelle entrée
      if(entry==null || !entry.getTerm().equals(type))
      {  entry = new IndexEntry(type);
         index.addEntry(entry,i);
         i++;
      }

      // dans tous les cas, on met àjour la liste de postings
      int docId = token.getDocId();
      Posting posting = new Posting(docId);
      entry.addPosting(posting);
      result++;
```

```java
    }

    return result;
}
```

Exercice 3

```java
public AbstractIndex buildIndex(List<Token> tokens, LexiconType lexiconType)
{  int indexSize;
   AbstractIndex result = null;

   {  System.out.println(" Sorting tokens...");
      Collections.sort(tokens);
      System.out.println(" "+tokens.size()+" tokens sorted\n");
   }

   {  System.out.println(" Filtering tokens...");
      indexSize = filterTokens(tokens);
      System.out.println(" "+tokens.size()+" tokens remaining, corresponding
                                             to "+indexSize+" terms\n");
   }

   {  System.out.println(" Building posting lists...");
      switch(lexiconType)
      {  case ARRAY:
            result = new ArrayIndex(indexSize);
            break;
         case HASH:
            result = new HashIndex(indexSize);
            break;
         case TREE:
            result = new TreeIndex();
            break;
      }
      int postingNumber = buildPostings(tokens,result);
      System.out.println(" "+postingNumber+" postings listed, lexicon="
                                                     +lexiconType);
   }

   return result;
}
```

2 Indexation

Exercice 4

```java
public static AbstractIndex indexCorpus(TokenListType tokenListType,
                LexiconType lexiconType) throws UnsupportedEncodingException
{  AbstractIndex result;
   Tokenizer tokenizer;
   Normalizer normalizer;
   List<Token> tokens = null;
   switch(tokenListType)
   {  case ARRAY:
         tokens = new ArrayList<Token>();
         break;
      case LINKED:
         tokens = new LinkedList<Token>();
         break;
   }
```

```java
      int docNbr;

      // tokénization
      {  System.out.println("Tokenizing corpus...");
         tokenizer = new Tokenizer();
         docNbr = tokenizer.tokenizeCorpus(tokens);
         System.out.println(tokens.size()+" tokens were found\n");
      }

      // normalisation
      {  System.out.println("Normalizing tokens...");
         normalizer = new Normalizer();
         normalizer.normalizeTokens(tokens);
         System.out.println(tokens.size()+" tokens remaining after
                                          normalization\n");
      }

      // construction
      {  System.out.println("Building index...");
         Builder builder = new Builder();
         result = builder.buildIndex(tokens, lexiconType);
         System.out.println("There are "+result.getSize()+" entries in the
                          index, token list="+tokenListType+"\n");
      }

      // affichage
      System.out.println("Content of the final index :");
      result.print();

      // finalisation de l'index
      result.tokenizer = tokenizer;
      result.normalizer = normalizer;
      result.docNbr = docNbr;
      return result;
   }
```

Exercice 5

```java
public static void main(String[] args) throws Exception
{  // configutation de l'outil
   Configuration.setCorpusName("wp");

   // test du processus d'indexation
   testIndexation();
}

private static void testIndexation() throws IOException
{  // construction de l'index
   AbstractIndex index = AbstractIndex.indexCorpus(TokenListType.LINKED,
                                          LexiconType.ARRAY);
}
```

Exercice 6

```java
public static List<String> getFileNamesFromPostings(List<Posting> postings)
{  List<String> result = new LinkedList<String>();
   File folder = new File(getCorpusFolder());
   String fn[] = folder.list();
   Arrays.sort(fn);

```

```java
      for(Posting posting: postings)
      {  int docId = posting.getDocId();
         String name = fn[docId];
         result.add(name);
      }

      return result;
   }
```

3 Durée de traitement

Exercice 8

Modification de la méthode `buildIndex` :

```java
   public AbstractIndex buildIndex(List<Token> tokens, LexiconType lexiconType)
   {  int indexSize;
      AbstractIndex result = null;

      {  System.out.println(" Sorting tokens...");
         long start = System.currentTimeMillis();
         Collections.sort(tokens);
         long end = System.currentTimeMillis();
         System.out.println(" "+tokens.size()+" tokens sorted,
                                             duration="+(end-start)+" ms\n");
      }

      {  System.out.println(" Filtering tokens...");
         long start = System.currentTimeMillis();
         indexSize = filterTokens(tokens);
         long end = System.currentTimeMillis();
         System.out.println(" "+tokens.size()+"tokens remaining, corresponding
                    to "+indexSize+" terms, duration="+(end-start)+" ms\n");
      }

      {  System.out.println(" Building posting lists...");
         long start = System.currentTimeMillis();
         switch(lexiconType)
         {  case ARRAY:
               result = new ArrayIndex(indexSize);
               break;
            case HASH:
               result = new HashIndex(indexSize);
               break;
            case TREE:
               result = new TreeIndex();
               break;
         }
         int postingNumber = buildPostings(tokens,result);
         long end = System.currentTimeMillis();
         System.out.println(" "+postingNumber+" postings listed,
                       lexicon="+lexiconType+",  duration="+(end-start)+" ms");
      }

      return result;
   }
```

Modification de la méthode `indexCorpus` :

```java
   public static AbstractIndex indexCorpus(TokenListType tokenListType,
                  LexiconType lexiconType) throws UnsupportedEncodingException
```

```
{  AbstractIndex result;
   Tokenizer tokenizer;
   Normalizer normalizer;
   List<Token> tokens = null;
   switch(tokenListType)
   {  case ARRAY:
         tokens = new ArrayList<Token>();
         break;
      case LINKED:
         tokens = new LinkedList<Token>();
         break;
   }
   int docNbr;
   long startTotal = System.currentTimeMillis();

   // tokénization
   {  System.out.println("Tokenizing corpus...");
      long start = System.currentTimeMillis();
      tokenizer = new Tokenizer();
      docNbr = tokenizer.tokenizeCorpus(tokens);
      long end = System.currentTimeMillis();
      System.out.println(tokens.size()+" tokens were found,
                                    duration="+(end-start)+" ms\n");
   }

   // normalisation
   {  System.out.println("Normalizing tokens...");
      long start = System.currentTimeMillis();
      normalizer = new Normalizer();
      normalizer.normalizeTokens(tokens);
      long end = System.currentTimeMillis();
      System.out.println(tokens.size()+" tokens remaining after
                           normalization, duration="+(end-start)+" ms\n");
   }

   // construction
   {  System.out.println("Building index...");
      long start = System.currentTimeMillis();
      Builder builder = new Builder();
      result = builder.buildIndex(tokens);
      long end = System.currentTimeMillis();
      System.out.println("There are "+result.getSize()+" entries in the
      index, token list="+tokenListType+", duration="+(end-start)+" ms\n");
   }

   long endTotal = System.currentTimeMillis();
   System.out.println("Total duration="+(endTotal-startTotal)+" ms\n");

   // affichage
   System.out.println("Content of the final index :");
   result.print();

   // finalisation de l'index
   result.normalizer = normalizer;
   result.tokenizer = tokenizer;
   result.docNbr = docNbr;
   return result;
}
```

Exercice 9

Remarque : Tous les résultats présentés ici sont obtenus avec le corpus `wp`.

Utilisation d'une liste à base de tableau pour stocker les tokens :

```
Tokenizing corpus...
7612885 tokens were found, duration=10498 ms

Normalizing tokens...
7612885 tokens remaining after normalization, duration=11648 ms

Building index...
Sorting tokens...
7612885 tokens sorted, duration=5322 ms

Filtering tokens...
2177994 tokens remaining, corresponding to 142792 terms, duration=16504892 ms

Building posting lists...
2177994 postings listed, lexicon=ARRAY, duration=415 ms
There are 142792 entries in the index, token list=ARRAY, duration=16510630 ms

Total duration=16532776 ms
```

Utilisation d'une liste chaînée pour stocker les tokens :

```
Tokenizing corpus...
7612885 tokens were found, duration=21879 ms

Normalizing tokens...
7612885 tokens remaining after normalization, duration=15493 ms

Building index...
Sorting tokens...
7612885 tokens sorted, duration=6362 ms

Filtering tokens...
2177994 tokens remaining, corresponding to 142792 terms, duration=1029 ms

Building posting lists...
2177994 postings listed, lexicon=ARRAY, duration=559 ms
There are 142792 entries in the index, token list=LINKED, duration=7954 ms

Total duration=45326 ms
```

Conclusion. pour le cas traité ici, les listes chaînées sont beaucoup plus efficaces. La différence se fait surtout lors du filtrage des tokens. En effet, `ArrayList` est basée sur un tableau et nécessite des redimensionnements, ce qui est régulièrement le cas lors du filtrage. Or, le redimensionnement nécessite de recopier l'intégralité du tableau à une adresse différente, ce qui est coûteux en temps.

Comparons maintenant les structures de données proposées pour le lexique, en utilisant des listes chaînées pour stocker les tokens durant l'indexation (puisque c'est le type de liste qui semble être le plus adapté à ce cas-là).

Table de hachage :

```
Tokenizing corpus...
7612885 tokens were found, duration=18166 ms
```

```
Normalizing tokens...
7612885 tokens remaining after normalization, duration=20696 ms

Building index...
Sorting tokens...
7612885 tokens sorted, duration=6372 ms

Filtering tokens...
2177994 tokens remaining, corresponding to 142792 terms, duration=1012 ms

Building posting lists...
2177994 postings listed, lexicon=HASH, duration=512 ms
There are 142792 entries in the index, token list=LINKED, duration=7902 ms

Total duration=46764 ms
```

Arbre de recherche :

```
Tokenizing corpus...
7612885 tokens were found, duration=17959 ms

Normalizing tokens...
7612885 tokens remaining after normalization, duration=27539 ms

Building index...
Sorting tokens...
7612885 tokens sorted, duration=6136 ms

Filtering tokens...
2177994 tokens remaining, corresponding to 142792 terms, duration=1065 ms

Building posting lists...
2177994 postings listed, lexicon=TREE, duration=568 ms
There are 142792 entries in the index, token list=LINKED, duration=7772 ms

Total duration=53272 ms
```

Pour le tableau, l'insertion se fait en temps constant, puisque on reçoit directement la position concernée. Pour la table de hachage, en théorie l'insertion s'y fait (en moyenne) en $O(1)$, c'est-à-dire là aussi en temps constant : ceci explique la proximité des temps mesurés pour ces deux structures de données. Pour l'arbre de recherche, l'insertion est (en moyenne) en $O(\log n)$, ce qui peut expliquer le temps plus élevé que pour les deux autres structures de données.

4 Stockage

Exercice 10

```java
public void write() throws IOException
{  System.out.println("Writing the index");
   long start = System.currentTimeMillis();

   String fileName = FileTools.getIndexFile();
   File file = new File(fileName);
   FileOutputStream fos = new FileOutputStream(file);
   ObjectOutputStream oos = new ObjectOutputStream(fos);

   oos.writeObject(this);

   oos.close();

   long end = System.currentTimeMillis();
   System.out.println("Index written, duration="+(end-start)+" ms\n");
}
```

Exercice 11

```java
public static AbstractIndex read() throws IOException, ClassNotFoundException
{  System.out.println("Loading the index");
   long start = System.currentTimeMillis();

   String fileName = FileTools.getIndexFile();
   File file = new File(fileName);
   FileInputStream fis = new FileInputStream(file);
   ObjectInputStream ois = new ObjectInputStream(fis);

   AbstractIndex result = (AbstractIndex) ois.readObject();

   ois.close();

   long end = System.currentTimeMillis();
   System.out.println("Index loaded, duration="+(end-start)+" ms\n");
   return result;
}
```