2 Construction de l'index

Exercice 1

- Le champ `data` modifié :

```
public TreeSet<IndexEntry> data;
```

- Le constructeur modifié :

```
public Index(~~int size~~)
{  data = new TreeSet<IndexEntry>();
}
```

- La méthode d'affichage modifiée :

```
public void print()
{  int i = 0;
   for(IndexEntry indexEntry: data)
   {  System.out.println(i+".\t"+indexEntry);
      i++;
   }
}
```

Exercice 2

- La méthode `buildPostings` modifiée :

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

      // dans tous les cas, on met à jour la liste de postings
      Posting posting = new Posting(token.docId);
      entry.postings.add(posting);
      entry.frequency++; //TODO rajouté
      result++;
   }

   return result;
}
```

- La méthode `buildIndex` modifiée :

```
public Index buildIndex(List<Token> tokens)
{  ~~int indexSize;~~
   Index result = new Index();
   // triage
   {  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");
      }
      // filtrage
      {  System.out.println("  Filtering tokens...");
         long start = System.currentTimeMillis();
         int indexSize = filterTokens(tokens);
         long end = System.currentTimeMillis();
         System.out.println("  "+tokens.size()+" tokens remaining,corresponding to"
                            +indexSize+" terms, duration="+(end-start)+" ms\n");
      }
      // construction
      {  System.out.println("  Building posting lists...");
         long start = System.currentTimeMillis();
         result = new Index(indexSize);
         int postingNumber = buildPostings(tokens,result);
         long end = System.currentTimeMillis();
         System.out.println("  "+postingNumber+" postings listed, duration="
                                                      +(end-start)+" ms");
      }

      return result;
   }
```

Exercice 3

Modifications de la méthode `getEntry` :

```
   public IndexEntry getEntry(String term)
   {  IndexEntry result = null;

      // on a besoin d'un objet IndexEntry bidon
      IndexEntry entry = new IndexEntry(term);
      // on prend toutes les entrées qui lui sont supérieures ou égales
      SortedSet<IndexEntry> temp = data.tailSet(entry);
      // si l'élément est contenu dans la liste, c'est donc forcément le premier
      if(!temp.isEmpty())
      {  IndexEntry tmp = temp.first();
         if(tmp.term.equals(term))
            result = tmp;
      }

      return result;
   }
```

Exercice 4

On obtient avec un arbre des temps de calcul comparables à ceux qu'on avait en utilisant un tableau trié pour l'index (cf. la correction du TP4 pour comparer) :

```
   Tokenizing corpus...
   5295713 tokens were found, duration=11516 ms

   Normalizing tokens...
   3275276 tokens remaining after normalization, duration=8294 ms

   Building index...
     Sorting tokens...
     3275276 tokens sorted, duration=3278 ms

     Filtering tokens...
     1780937 tokens remaining, corresponding to 114118 terms, duration=670 ms

     Building posting lists...
     1780937 postings listed, duration=465 ms
   There are 114118 entries in the index, duration=4415 ms

   Total duration=24225 ms
```

3 Recherche approchée

Exercice 5

- Modification de `tokenizeString` :

```
public List<String> tokenizeString(String string, boolean removeJokers)
{  List<String> result = new LinkedList<String>();

   // on segmente la chaîne
   String temp[] = string.split("[^\\pL\\pN]");
   if(removeJokers)
      temp = string.split("[^\\pL\\pN\\*]");
   else
      temp = string.split("[^\\pL\\pN]");

   // on traite chaque segment
   ...
}
```

- Modification de `tokenizeDocument` :

```
public void tokenizeDocument(File document, int docId, List<Token> tokens)
{  ...
         List<String> types = tokenizeString(line,true);
   ...
}
```

Exercice 6

- Modifications apportées à `normalizeType` :

```
public String normalizeType(String string, boolean removeStopWords)
{  ...
   // si la chaîne est vide ou est un mot vide, on renvoie null
   if(result.isEmpty() || (removeStopWords && stopWords.contains(result)))
      result = null;

      return result;
}
```

- Modification apportée à `normalizeTokens` :

```
   public void normalizeTokens(List<Token> tokens)
   {  ...
         // on normalise le type
         token.type = normalizeType(token.type,true);
      ...
   }
```

Exercice 7

- Approche vue en cours :

```
public List<IndexEntry> getEntriesStartingWith(String prefix)
{  List<IndexEntry> result = new LinkedList<IndexEntry>();

   // on a besoin d'un objet IndexEntry bidon
   IndexEntry entry = new IndexEntry(prefix);
   // on prend toutes les entrées qui lui sont supérieures ou égales
   SortedSet<IndexEntry> temp = data.tailSet(entry);

   // on les parcourt et les garde tant qu'elles commencent par le prefixe
   boolean goOn = true;
   Iterator<IndexEntry> it = temp.iterator();
   while(goOn && it.hasNext())
   {  IndexEntry indexEntry = it.next();
      goOn = indexEntry.term.startsWith(prefix);
      if(goOn)
         result.add(indexEntry);
   }

   return result;
```

```
}
```

- Approche plus rapide (computationnellement) :

```
public List<IndexEntry> getEntriesStartingWith(String prefix)
{  // on a besoin d'un objet IndexEntry bidon
   IndexEntry entry = new IndexEntry(prefix);
   // on prend toutes les entrées qui lui sont supérieures ou égales
   SortedSet<IndexEntry> temp = data.tailSet(entry);

   // on peut même faire plus simple en gardant les entrées inférieures ou
   // égales à la première chaine possible ne commençant pas comme le préfixe
   if(prefix.isEmpty())
      entry = new IndexEntry("\u0000"); // tout 1er caractère unicode
   else
   {  char lastChar = prefix.charAt(prefix.length()-1);
      entry = new IndexEntry(prefix.substring(0,
                                    prefix.length()-1) + (char)(lastChar+1));
   }
   temp = temp.headSet(entry);
   List<IndexEntry> result = new LinkedList<IndexEntry>(temp);

   return result;
}
```

Exercice 8

Il s'agit exactement de la même méthode que `processDisjunction`, développée lors du TP3 pour traiter l'opérateur OU (cf. sa correction en ligne sur e-uapv).

Exercice 9

```
private void splitQuery(String query, List<List<Posting>> result)
{  // on tokénize la requête
   Tokenizer tokenizer = index.tokenizer;
   List<String> types = tokenizer.tokenizeString(query,false);

   // on normalise chaque type
   Normalizer normalizer = index.normalizer;
   for(String type: types)
   {  // s'il n'y a pas de joker à la fin (cas normal)
      if(!type.endsWith("*"))
      {  // la normalisation du type donne le terme (ou null)
         String term = normalizer.normalizeType(type,true);
         if(term!=null)
         {  // on récupère l'entrée associée au terme dans l'index
            IndexEntry entry = index.getEntry(term);
            // si pas dans l'index, on utilise une liste vide
            if(entry==null)
               result.add(new ArrayList<Posting>());
            // sinon, on prend sa liste de postings
            else
               result.add(entry.postings);
         }
      }
      // s'il y a un joker à la fin
      else
      {  // on enlève le joker puis on normalise (partiellement)
         type = type.substring(0, type.length()-1);
         String prefix = normalizer.normalizeType(type,false);
         if(prefix!=null)
         {  List<Posting> list = new LinkedList<Posting>();
            // on récupère les entrées associées au préfixe dans l'index
            List<IndexEntry> entries = index.getEntriesStartingWith(prefix);
            // on prend leur union
            for(IndexEntry entry: entries)
               list = processUnion(list,entry.postings);
            result.add(list);
         }
      }
```

```
        }
    }
```