lab1

1. 目录结构

2. 实现细节

   2.1. task1

   2.2. task2

3. 创新点

1. 目录结构

```Plain Text
lab1/
└─src
    └─main
        ├─java
        │      task1.java                    // 执行文本分词任务
        │      task2.java                    // 执行分词高频项输出任务
        │
        └─resources
            ├─data                           // 存放实验数据
            │      cn_stopwords.txt
            │      corpus.dict.txt
            │      corpus.sentence.txt
            │
            └─result                         // 存放实验结果
                    high_frequency.txt       // 高频项及其频率
                    word_segmentation.txt    // 分词结果
```

2. 实现细节

2.1. task1

- `List<String> readTextFile(String filePath)`：**读取文本**

  逐行读取文本，去除句子两端的空格。

- `List<Set<String>> readDictFile(String filePath, int maxLength)`：**读取词典**

  维护一个子词典列表，各部子词典分别存储不同词长的词，`maxLength` 为词长的最大长度，也为子词典数量。逐行读取词典，每行去除两端空格后即为一个词，根据词长保存至对应的子词典。

- `List<String> segmentWord(String text, List<Set<String>> dicts, int maxLength)`：**分词**

  自主实现最大正向匹配分词算法。左指针初始化指向句子的首部。每次匹配开始时，右指针指向的位置为左指针指向位置后面的 `maxLength` 长度处（句尾处可能长度不足）。将两指针之间的字符串与对应词长的词典中的词逐一进行匹配。若成功匹配，则左指针指向的位置改为右指针指向位置后面的 `1` 长度处；若未能匹配，则右指针左移 `1` 长度，继续进行匹配；若当前字符串长度为 `1`，则判断为单字。

2.2. task2

- `Set<String> readStopWordsFile(String filePath)`：**读取停用词词典**
- `List<List<String>> readSegmentedWordsFile(String filePath, Set<String> stopWords)`：**读取分词结果**
- `List<String> processWords(String[] words, Set<String> stopWords)`：**去除标点及停用词**

  将词与停用词词典中的词逐一比较。

- `List<Map.Entry<String, Double>> countWords(List<List<String>> words)`：**统计词频**

  遍历分词结果中的每个词，统计其出现次数，计算其占据全文分词总量的概率，并按概率降序排列。

3. 创新点

通过将原有词典的词按照词长分别存入单独的子词典，在几乎不增加空间复杂度的前提下，大大降低了分词任务的时间复杂度。

利用 HashMap 类保存词典，提升了后续执行任务时词的匹配效率。

（未实现）执行词匹配任务时，首先根据标点符号进行划分。（可能与词典不匹配）