实验一：基于正向最大匹配算法的分词

课程名称:	自然语言处理	指导教师:	张蓉
姓　　名:	王海生	学　　号:	10235101559

代码仓库：https://github.com/Hanson-Wang-chn/ECNU-NLP-WHS.git

1 实验结果

```
/Users/wanghaisheng/Desktop/Coding/Courses/NLP/lab1/code/cmake-build-debug/code
Top 10高频词（概率降序）：
1. 民族（出现次数: 38, 概率: 4.7619%）
2. 中华文明（出现次数: 28, 概率: 3.50877%）
3. 文化（出现次数: 27, 概率: 3.38346%）
4. 统一（出现次数: 23, 概率: 2.88221%）
5. 统一性（出现次数: 20, 概率: 2.50627%）
6. 中华民族（出现次数: 17, 概率: 2.13033%）
7. 发展（出现次数: 17, 概率: 2.13033%）
8. 一体（出现次数: 17, 概率: 2.13033%）
9. 形成（出现次数: 15, 概率: 1.8797%）
10. 多元（出现次数: 13, 概率: 1.62907%）

Process finished with exit code 0
```

图 1: 实验结果

2 正向匹配算法

最大正向匹配分词算法（Forward Maximum Matching, FMM）是一种基于**词典**的自然语言处理方法，核心思想是从文本的**开头**开始逐步向前推进，尝试寻找**最长**且位于预定义词汇表中的词组来进行切割。

2.1 算法步骤

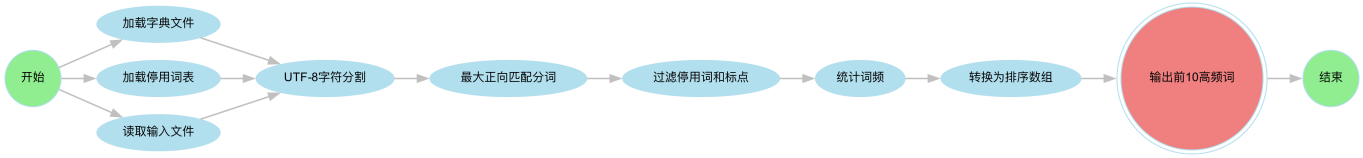

图 2: 最大正向匹配分词算法示意图

1. 加载字典文件、停用词表: 读取词典文件、停用词表，分别放入一个无需集合（代码中使用哈希表实现）中。
2. 读取输入文件: 读取待分析的`.txt` 文件。
3. UTF-8 字符分割: 由于需要处理中文汉字，有必要把所有的字符统一为 UTF-8 编码。在实践中，忽略这一步会导致乱码。
4. 最大正向匹配分词: 内外两个循环。内循环从最大长度开始递减，直至匹配；外循环遍历待处理文本。
5. 过滤停用词和标点: 移除分词结果中的停用词及标点符号。
6. 统计词频: 计算每种词汇在文本中出现的次数，并记录总数。
7. 转换为排序数组: 将统计得到的词频数据转换成一个数组，并按照出现频率从高到低进行排序。
8. 输出高频词: 显示出现频率最高的前 10 个词汇及其对应的出现概率。

2.2 算法分析

1. **优点**：实现简单；非常适合**处理长词**。
2. **缺点**：难以处理词典中不存在的词（每个汉字均作为一个词），性能依赖于词典，灵活性较差。
3. **其他算法**：
   - **最大逆向匹配**：实现复杂度稍高，能更好地处理某些歧义情况。
   - **双向最大匹配**：计算量稍大，结合正向匹配和逆向匹配的优点。
   - **基于统计或深度学习的算法**：不再依赖于词典，效果较好，但需要较大的算力和数据集。

3 C++ 代码创新点

由于本次实验的代码使用 C++ 编写，没有 Python 中的一些分词库，所以必须从头开始设计算法和数据结构。我认为，我的代码中的亮点是，使用了**哈希表**作为保存词表的数据结构，大幅降低了算法的时间复杂度。

具体来说，我使用 unordered_set 作为存储字典文件和停用词表的无序集合，使用 unordered_map 来一一对应词和该词的频率。

```
1    unordered_set<string> loadDictionary(const string& path); // 字典
2    unordered_set<string> loadStopwords(const string& path); // 停用词表
3    unordered_map<string, int> freq; // 词和频率
```

借这次实验的机会，除最大正向匹配分词算法之外，我还熟悉了 C++ STL 的原理和使用。这算是比较大的收获。