# lab5

## 1. 目录结构

```
lab5/
└─src
    └─main
        ├─java
        │     task.java              // 通用函数
        │     task1.java             // 使用“长字符串+词项长度”方法
        │     task2.java             // 使用“按块存储+前端编码”方法
        │
        └─resources
            │  dict.txt              // 原词典
            │
            └─result                 // 存放实验结果
                  new_dict_1.txt     // 使用第一种方法压缩后的词典
                  new_dict_2.txt     // 使用第一种方法解压后的词典
                  raw_dict_1.txt     // 使用第二种方法压缩后的词典
                  raw_dict_2.txt     // 使用第二种方法解压后的词典
```

## 2. 实现细节

### 2.1. task

- `Map<String, String> readTermData(String filePath, Integer taskID)`：**读取原词典**

  读取原词典获取词项和其对应的倒排索引表指针。对于 task1，仅需使用 HashMap 存储，而对于 task2，需要使用 TreeMap 按照键的字典序存储。具体而言，依照长度 107 截取得到词项并去除左端用于补齐缺省的空格，依照长度 8 截取得到倒排索引表指针并去除左端用于补齐缺省的 0。特别的，存在指针指向地址为 0 的情况，需要在去除用于补齐缺省的 0 之后将其地址赋为“0”。

- `void saveDict(String dict, String filePath)`：**保存词典**

  将一个表示词典的字符串保存为 .txt 文件。

- `void evaluate(String rawDictPath, String newDictPath, Integer taskID)`：**评估压缩效果**

  利用压缩后的词典所占空间大小与原词典所占空间大小，计算空间节省百分比，从而评估压缩效果，空间节省百分比越大，压缩效果越好。

## 2.2. task1

“长字符串+词项长度”方法存储结构：

| 词项1长度的位数 | 词项1长度 | 词项1 | 词项1倒排索引表指针长度 | 词项1倒排索引表指针 |
|---|---|---|---|---|

| 词项2长度的位数 | 词项2长度 | 词项2 | 词项2倒排索引表指针长度 | 词项2倒排索引表指针 | …… |
|---|---|---|---|---|---|

- `String compress(Map<String, String> termData)`：**压缩词典**

  遍历读取的“词项–倒排索引表指针”对。对于一个“词项–倒排索引表指针”对，在新词典中需要依次存储以下信息：词项长度的位数、词项长度、词项、倒排索引表指针长度、倒排索引表指针。存储词项长度的位数是因为词项长度可能为两位数或三位数，需要位数指示来引导读取，而倒排索引表指针长度均为个位数，所以无需为其提供位数指示。最终将所有词项的所有信息连结成一个长字符串。

- `String decompress(String newDict)`：**解压词典**

  遍历表示词典的长字符串。首先读取到当前词项长度的位数，据此继续读取到当前词项长度，据此继续读取到当前词项；再读取到当前词项对应的倒排索引表指针的长度，据此继续读取到当前词项对应的倒排索引表指针。然后保存为长度为 107 的词项（缺省用空格从左端补齐）+ 长度为 8 的倒排索引表指针（缺省用 0 从左端补齐）。继续读取下一词项信息，重复上述过程。

## 2.3. task2

“按块存储+前端编码”方法存储结构：

| 块1长度位数 | 块1长度 | 前缀长度位数 | 前缀长度 | 前缀 | 后缀1长度位数 | 后缀1长度 | 后缀1 | 后缀1倒排索引表指针长度 |
|---|---|---|---|---|---|---|---|---|

| 后缀1倒排索引表指针 | 后缀2长度位数 | 后缀2长度 | 后缀2 | 后缀2倒排索引表指针长度 | 后缀2倒排索引表指针 | … … |
|---|---|---|---|---|---|---|

- `int findCommonPrefix(String s1, String s2)`：**寻找公共前缀**

  寻找两个字符串的公共前缀，返回公共前缀的长度。

- `String compress(Map<String, String> termData)`：**压缩词典**

  遍历读取的“词项–倒排索引表指针”对。对于存储至同一个块中的 $k$ 个“词项–倒排索引表指针”对，在新词典中需要依次存储以下信息：块长度的位数、块长度、前缀长度的位数、前缀长度、前缀、{后缀长度的位数、后缀长度、后缀、倒排索引表指针长度、倒排索引表指针} $\times k$ 。存储块长度的位数是因为块长度可能为两位数或三位数，需要位数指示来引导读取，存储前缀和后缀长度的位数同理。特别的，当一个块中只存储有一个词项时，该块中的前缀即此词项，并将该块中的后缀长度的位数设置为 0，且该块中不含有后缀长度、后缀这两个信息。最终将所有词项的所有信息连结成一个长字符串。

- `String decompress(String newDict)`：**解压词典**

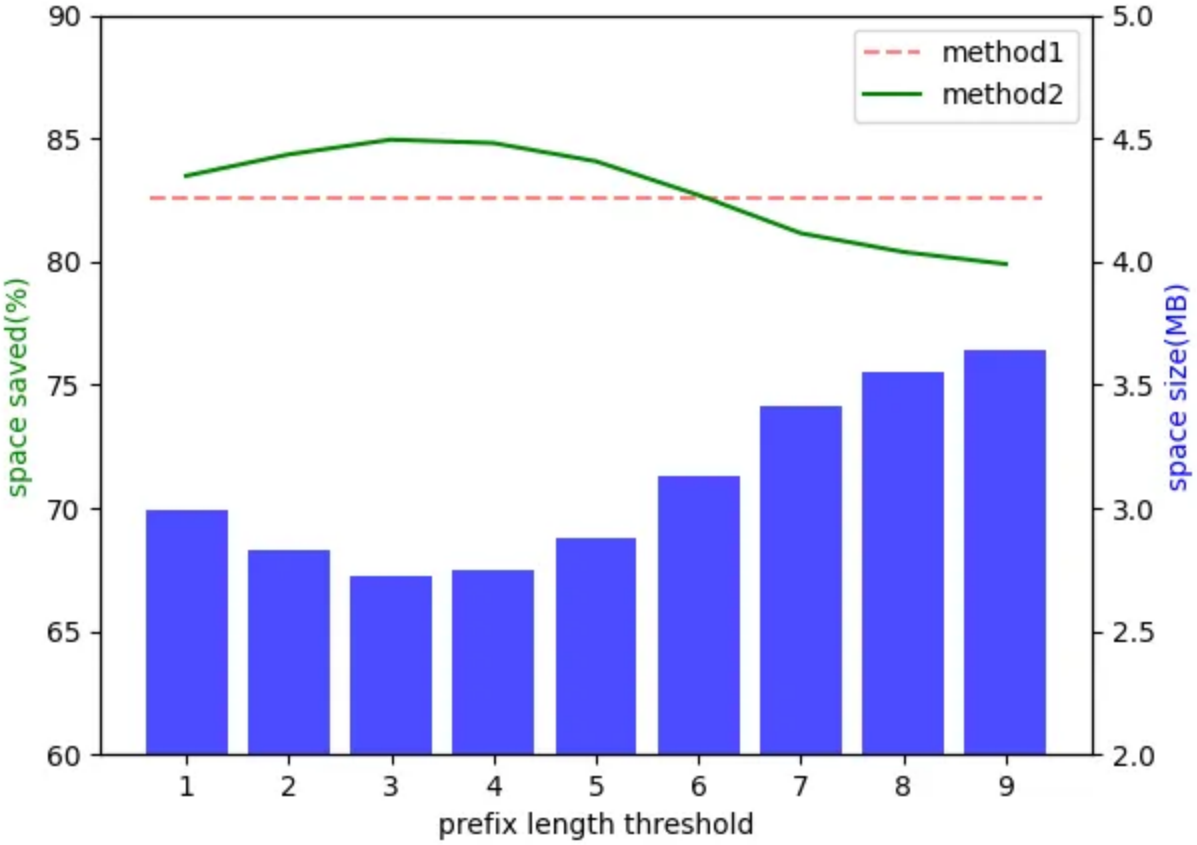
  遍历表示词典的长字符串。首先读取到当前块长度的位数，据此继续读取到当前块长度，据此继续读取到当前块；再读取到当前块的前缀长度的位数，据此继续读取到当前块的前缀长度，据此继续读取到当前块的前缀；再读取到当前后缀长度的位数，据此继续读取到当前后缀长度，据此继续读取到当前后缀；再读取到当前后缀对应的倒排索引表指针的长度，据此继续读取到当前后缀对应的倒排索引表指针。再将前缀与后缀连接成词项，然后保存为长度为 107 的词项（缺省用空格从左端补齐）+ 长度为 8 的倒排索引表指针（缺省用 0 从左端补齐）。继续读取下一后缀信息，重复上述过程。读取完当前块信息后，继续读取下一块信息，重复上述过程。

# 3. 实验结果

在实现“按块存储+前端编码”压缩方法时，前缀长度下限是一个超参数，需要进行调参。经过实验寻找到最优参数取值为 3。当取值大于等于 7 时，压缩效果反而劣于“长字符串+词项长度”方法，这是因为很少有词项共有这么长的前缀，此时绝大多数块中只存储有一个词项，而“按块存储+前端编码”方法需要向词典中额外存储块长度的位数、块长度和后缀长度的位数这些信息，造成了空间开销上的浪费。

| 长字符串+词项长度 | 3.16 | 82.56 |
|---|---|---|
| **前缀长度下限** | **压缩后空间大小(MB)** | **空间节省百分比(%)** |
| 1 | 2.99 | 83.48 |
| 2 | 2.83 | 84.35 |

| | | |
|---|---|---|
| 3 | 2.72 | 84.95 |
| 4 | 2.75 | 84.81 |
| 5 | 2.88 | 84.07 |
| 6 | 3.13 | 82.70 |
| 7 | 3.41 | 81.15 |
| 8 | 3.55 | 80.39 |
| 9 | 3.64 | 79.90 |




# 4. 创新点

- 在实现“按块存储+前端编码”压缩方法时，对于前缀长度下限进行了调参，寻找到了最优参数取值