# 信息检索实验一实验报告

## 项目简介

该项目用于构建倒排索引，并实现了布尔联合检索。最后对倒排索引的编码方式进行优化，实现了对于文件间隔的VB编码和γ编码。

项目完成后的文件目录如下：

```
air2-hw1-2021
├─dev_output                ----- 测试查询的输出
├─dev_queries               ----- 测试的查询
├─output_dir                ----- 未使用压缩编码的索引
├─output_dir_compressed      ----- 使用gap-encoding和VB编码的索引
├─output_dir_eccompressed    ----- 使用γ编码的索引
├─pa1-data                  ----- 数据集
├─tmp                       ----- 测试程序的输出目录
├─toy-data                  ----- 测试使用的数据集
└─toy_output_dir            ----- 测试集上的索引
```

## 倒排索引的实现

### `IdMap`类

该类用于实现`id`和`str`之间的映射关系。在初始化中定义了两个属性：

- `str_to_id`：字典，用于存储`str:id`形式的键值对
- `id_to_str`：列表，`id`为列表的下标，可以通过下表查询对应的`str`

该类中有魔术方法`__getitem__(self, key)`，用于对以`[]`形式查找元素是进行处理，对不同类型的`key`调用不同的方法

该类中需要完善两个方法：

- `_get_str(self, i)`：通过`id`查找对应的`str`，直接从`id_to_str`中取出即可
- `_get_id(self, s)`：通过`str`查找`id`，如果该`str`不在字典中，需要在两个变量中添加对应的映射关系，之后直接从`str_to_id`取出即可

在后续的代码中，需要使用该类实现文档名(doc)和文档id(docID)、单词(term)和单词id(termID)之间的映射。

### `UncompressedPostings`类

该类没有对数据进行压缩，包含两个静态方法：

- `encode(postings_list)`：对列表进行编码
- `decode(encoded_postings_list)`：对字节序列进行解码

## `InvertedIndex`类

该类作为基类，实现了对文件读写的一些基本功能。

该类的构造方法为`InvertedIndex(index_name, postings_encoding=None, directory='')`，参数的介绍如下：

- `index_name`：索引的名称，后续生成索引文件和字典文件时作为文件名
- `postings_encoding`：编码方式，如果没有该项，后续会默认为上面的`UncompressedPostings`
- `directory`：生成文件的目录，会在当前文件夹下生成该目录，并在该目录中写入索引文件和字典文件

该类在`__init__`和`__enter__`中定义了以下属性：

- `index_file_path`：索引文件的路径，形如`<directory>/<index_name>.index`
- `metadata_file_path`，字典文件的路径，形如`<directory>/<index_name>.dict`
- `postings_encoding`：编码方式
- `directory`：生成文件所在目录
- `postings_dict`：字典，存储格式为`termID:(该词的索引在索引文件中的起始位置, 该词的索引中元素个数, 该词的索引的长度)`
- `terms`：列表，用于存储该索引中所有的`term`
- `index_file`：已打开的索引文件，可以从中读取索引信息
- `term_iter`：`terms`的迭代器

在使用`with`打开`InvertedIndex`对象时，会在给定的目录下打开名为`<index_name>.index`的文件，读取名为`<index_name>.dict`的文件，将存储的`postings_dict`和`terms`读入到对应的属性中。在`with`语句结束时，会关闭`<index_name>.index`文件，并将`postings_dict`和`terms`保存到`<index_name>.dict`文件中。

## `InvertedIndexWriter`类

继承自`InvertedIndex`类，该类重新实现了`__enter__`方法，使得在使用`with`打开时只需要打开指定的`.index`文件，通过`append`方法写入索引，在`with`语句结束时会关闭`.index`文件，并将`postings_dict`和`terms`保存到指定的`.dict`文件中。

该类的属性与`InvertedIndex`类相同，额外实现了一个方法`append(term, postings_list)`，实现方法如下：

- `term`为给定的`term`或`termID`，直接添加到`self.terms`的末尾
- `postings_list`为该`term`对应的索引列表，调用`self.postings_encoding.encode`对该列表进行编码
- 获取`index_file`的文件大小，作为起始位置，获取元素个数、索引长度，再写入`postings_dict`中
- 将编码后的序列写入`index_file`中，调用`self.index_file.flush()`刷新写缓冲区，将索引写入磁盘中

## `InvertedIndexIterator`类

继承自`InvertedIndex`类，该类为索引的迭代器，用于对磁盘上的索引进行迭代，每次迭代返回`(term,postings_list)`。

该类重写了`InvertedIndex`类的`__enter__`方法，在打开`.index`文件、读取`.dict`文件并写入属性之后，需要调用`_initialization_hook()`额外初始化一些迭代器中自行定义的属性。该类还提供了一个方法`delete_from_disk()`，调用该方法会在`with`语句结束时删除磁盘上的`.index`和`.dict`文件。如果没有调用该方法，会将两属性写入`.dict`文件中。

该类中需要完善两个方法：

- `_initialization_hook(self)`：在这里我们定义一个`self.curr_pos = 0`用于初始化并记录迭代到的位置
- `__next__(self)`：用于获取下一个元素。首先判断`self.curr_pos`与`len(self.terms)`的大小，如果前者小，则可以从`self.terms`中获取对应下标的`term`，并根据`self.postings_dict`查询该索引在`.index`文件中的起始位置与大小，从文件中读取索引并解码。最后使`self.curr_pos`自增1并返回`(term,postings_list)`。如果超出遍历范围，则抛出异常`StopIteration`

## `BSBIIndex`类

该类的构造方法为`BSBIIndex(data_dir, output_dir, index_name = "BSBI",  postings_encoding = None)`，参数的介绍如下：

- `data_dir`：数据所在目录，比如在本项目中的`toy-data`、`pa1-data`
- `output_dir`：输出目录，比如在本项目中的`tmp`，`output_dir`，`toy_output_dir`
- `index_name`：作为合并后的索引的名称
- `postings_encoding`：编码方式

在`__init__`方法中，定义了以下属性：

- `term_id_map`：`term`与`termID`的映射
- `doc_id_map`：`doc`与`docID`的映射
- `data_dir`：数据所在目录
- `output_dir`：输出目录
- `index_name`：索引名称
- `postings_encoding`：编码方式
- `intermediate_indices`：分块生成的索引名的列表，形如`['index_0', 'index_1']`

该类还提供了保存文件的方法`save()`和读取文件的方法`load()`，用于读取或将`term_id_map`和`doc_id_map`保存为`terms.dict`和`docs.dict`

我们下面对其他的方法进行详细的解释。

## `parse_block`方法

该方法用于实现对一块中的所有文件建立索引。

在数据集`pa1-data`中，数据被分别放置在编号为0~9这十个文件夹中，我们将每一个文件夹视为每一块，需要在这每一块上分别建立索引。

该方法的原型为`parse_block(block_dir_relative)`，传入的参数为相对于`data_dir`路径，例如`'0'`，`'1'`。

具体实现方法如下：

- 为了能得到该文件夹下的文件，合并`data_dir`和`block_dir_relative`，读取合并后的路径下的文件并排序
- 遍历文件列表
    - 通过`self.doc_id_map`获取为该文件分配的`id`，这里传入的`key`为`block_dir_relative/file_name`，例如`0/fine.txt`
    - 读取文件，分割单词，遍历单词，获取单词`id`，构成形如`(termID,docID)`的`td_pair`，添加到`td_pairs`中
- 返回`ts_pairs`

## `inverte_write`方法

该方法用于将`parse_block`得到的`td_pairs`转换为倒排表，并写入磁盘中。

该方法的原型为`invert_write(td_pairs, index)`，其中`td_pairs`为`term-id`对，`index`为`InvertedIndexWriter`对象，通过该对象将得到的倒排表写入磁盘中。

具体的实现方法如下：

- 维护一个字典，用于存储`term:set`键值对，这里的`set`为该`term`对应的倒排表
- 遍历`td_pairs`，按照每一个的内容添加到字典里对应的`set`中
- 对排序过后的字典进行遍历，对每一个倒排表进行排序，转换为`list`后调用`index.append`方法写入磁盘

## `merge`方法

该方法用于合并每一块得到的倒排表，将合并后的倒排表分别将写入磁盘中。

该方法的原型为`merge(indices, merged_index)`，其中`indices`的类型为`List[InvertedIndexIterator]`，可以简单地将其视为`List[List[Tuple(Int,List)]]`，这里的`tuple`为`(term,postings_list)`；`merged_index`为`InvertedIndexWriter`对象，用于将合并后的倒排表写入磁盘中。

具体的实现方法如下：

- 通过使用`heapq.merge`对`indices`里的`tuple`按照`term`进行排序
- 遍历排序后的对象，将`term`相同列表加入同一个`List`中，对每一个`List`通过`heapq.merge`排序，得到倒排表
- 对每一个倒排表通过调用`merged_index.append`写入磁盘中

### `index`方法

这是一个已经写好的方法，用于对数据集建立倒排索引。具体的实现方法如下：

- 使用`sorted(next(os.walk(self.data_dir))[1])`得到数据目录下的所有子文件夹的名称
- 遍历子文件夹，将这里的每一个子文件夹视为一块
  - 调用`self.parse_block`解析为`td_pairs`
  - 将该块的索引名称`index_id`命名为`index_<文件夹名>`，例如`index_0`，并将该名称加入`self.intermediate_indices`属性中
  - 通过构造`InvertedIndexWriter`对象，调用`inverte_write`方法，在输出目录中生成`<index_id>.index`和`<index_id>.dict`文件
- 调用`save`方法保存`terms.dict`和`docs.dict`
- 构造`InvertedIndexWriter`对象，获取所有块的索引并合并，在输出目录中生成`<index_name>.index`、`<index_name>.dict`文件，前者作为该数据集里所有块上的倒排索引。

## 索引压缩-VB编码

在本项目中主要对倒排表进行压缩，通过对文件间隔使用gap-encoding，对间隔表进行VB编码。

VB编码又名可变长字节编码，将数字的二进制序列按照7位分割，去掉全是0的部分，在其余部分左边添加一位0（该部分不是最后一个部分）或1（该部分是最后一个部分），作为该数的编码。

编码步骤如下：

- 使用`zip(postings_list,postings_list[1:])`，通过列表的推导式获得文件的间隔序列，在列表第一项添加第一个文件的`id`
- 对得到的列表中的每个元素进行VB编码，得到字节序列
- 合并字节序列

解码的思想就是比较每个字节对应的整数与128的大小，如果小于128，则继续读取；如果大于128，则说明首位为1，该数字结束，将该数加上前一个数后写入列表中。直到字节序列遍历完毕。

## 额外的编码方式-γ编码

γ编码是将数字变为二进制序列，去掉左边的0和第一个1，根据剩下二进制位数在左边添加相同个数的1和一个0，作为该数的编码。

编码步骤如下：

- 使用相同的方式得到文件间隔列表，将第一个文件的`id`加上1之后添加到列表头部（为了防止有0）
- 对得到的列表中的每个元素进行γ编码，得到01字符串序列
- 合并字符串，对于字符串序列末端不足8位的部分补`'1'`
- 将字符串转换为字节序列

解码的思想就是将字节序列转换为01字符串序列后，遍历字符串，遇到0即根据之前1的个数向后取对应位的子串，在子串的开头加上`'1'`，转换为整数。如果这是列表第一个元素，需要减一；如果不是，需要加上前一个元素。直到遍历完整个字符串结束。

# 实验结果

下表展示了使用三种编码方式在 `pa1-data` 数据集上构建索引时所用时长以及得到的索引文件大小。

| 编码 | 时间 | 索引大小 | 时间比例 | 压缩率 |
| --- | --- | --- | --- | --- |
| 未压缩 | 1m30s | 53991KB | - | - |
| VB编码 | 1m35s | 16002KB | 1.056 | 70.36% |
| γ编码 | 1m49s | 11179KB | 1.211 | 79.29% |

可以看到γ编码压缩率最高，达到了79.29%，但是构建时间为不使用压缩算法的1.211倍。这说明编码的时间较长，可以通过修改编码实现方式，例如直接处理字节序列而非字符串，从而提高编码速度。