信息检索系统实验1

1911590 周安琪

1 数据集

2 索引构建与检索 (40%)

作业的第一部分是使用**blocked sort-based indexing (BSBI)** 算法来构建倒排索引并实现布尔检索。关于BSBI算法可以参考老师课件或者斯坦福教材 Section 4.2 。以下摘自教材内容

2.1 IdMap

把term映射到int类型的termID

所以在往map中添加的时候，肯定是以term为key。

```
def _get_str(self, i):
    return self.id_to_str[i]

def _get_id(self, s):
    # 如果参数中的str在map中没有，那么加进去
    if s not in self.str_to_id:
        self.str_to_id[s] = len(self.id_to_str)
        self.id_to_str.append(s)
        return self.str_to_id.get(s)
```

2.2 将倒排列表编码成字节数组

2.3 磁盘上的倒排索引

2.4 索引

2.4.1 解析

`parse_block` 对文件做语法分析，生成 `term-ID->docID` 的对，并且把这些对存储在内存里，直到收集一个block停止。

```
#参数是str类型，是要处理的文件的地址。
def parse_block(self, block_dir_relative):

    td_pairs = []

    #首先把两个路径连起来，然后把里面的文档排序，按次序打开file
    for file_dir in sorted(os.listdir(os.path.join(self.data_dir, block_dir_relative))):

        # 'r': 以只读方式打开文件
        with open(os.path.join(self.data_dir, block_dir_relative, file_dir), 'r') as f:
            # strip()删除空白符（包括'\n', '\r', '\t',  ' ')
            # split()以空格和换行为分隔符分开，返回分割后的字符串列表。
            content = f.read().strip().split() # list of tokens

            # 把doc映射到doc_id
            doc_id = self.doc_id_map[os.path.join(block_dir_relative, file_dir)]

            # 现在content是由一堆token组成的东西
            for token in content:
                # 把token映射到id，然后把pair加入到列表中
                term_id = self.term_id_map[token]
                td_pairs.append([term_id, doc_id])
    return td_pairs
```

2.4.2 倒排表

2.4.2.1 append()

把做好的 `term->posting_list` 加到索引文件的末尾

- 用posting_encoding来给postings_list编码
- 将metadata 以 `self.terms` 和 `self.postings_dict` 的形式储存，`self.postings_dict` 把 `termID` 映射到一个三元组
- 加入磁盘里的索引文件

```
def append(self, term, postings_list):
    # 编码
    encoded_postings_list =
self.postings_encoding.encode(postings_list)
    # 开始地点是文件末尾，目前的偏移量是0
    start_position_in_index_file = self.index_file.seek(0, 2)
    # 返回写入的字符长度。
    length_in_bytes_of_postings_list = self.index_file.write(
        encoded_postings_list)

    self.terms.append(term)
    # termID映射到一个三元组，这个三元组其实就是倒排索引的列表，包括开始位置，
几条条目，按字节有多长
    self.postings_dict[term] = (start_position_in_index_file,
                                len(postings_list),
                                length_in_bytes_of_postings_list)
```

2.4.2.2 invert_write()

将解析得到的td_pairs转换成倒排表，并使用 `InvertedIndexWriter` 类将其写入磁盘。

```
def invert_write(self, td_pairs, index):
    td_dict = collections.defaultdict(list)
    for t, d in td_pairs:
        td_dict[t].append(d)
        # 以键值排序
        for t in sorted(td_dict.keys()):
            # 每一个term对应着一个posting_list
            p_list = sorted(td_dict[t])
            # 在索引中加入这个tuple
            index.append(t, sorted(p_list))
```

2.4.3 合并

2.4.3.1 基础函数

```
def _initialization_hook(self):
    # 初始化指针，一开始在文件的开头
    self.curr_term_pos = 0

def __next__(self):

    if self.curr_term_pos >= len(self.terms):
        raise StopIteration

```

```
        # term
        term = self.terms[self.curr_term_pos]
        self.curr_term_pos += 1
        # 对于每个term，都有一块index的空间。
        start_position, n_postings, length_in_bytes =
    self.postings_dict[term]
        # offset=start_position
        self.index_file.seek(start_position)
        # 读取相应长度的数据，就是postings_list
        postings_list = self.postings_encoding.decode(
            self.index_file.read(length_in_bytes))
        return term, postings_list
```

2.4.3.2 merge()

把相同term后面跟着的postings_list整合到一起。

- indices：指针列表，每个指针指向其中一个小的索引
- merged_index：结果

```
def merge(self, indices, merged_index):
    last_term = last_posting = None

    # 先把indices合并排序，得到一个 term & postings 对的列表（排好序的）。
    for curr_term, curr_postings in heapq.merge(*indices):
        # 如果和之前的不相等也就是说之前的那个没有可以合并得了。
        if curr_term != last_term:
            # 如果不是第一个条目
            if last_term:
                # 在结果中加上这一term及其条目，接着循环。
                last_posting = list(sorted(set(last_posting)))
                merged_index.append(last_term, last_posting)
            last_term = curr_term
            last_posting = curr_postings
        else:
            # 如果这两个需要合并，就把这个条目加进去。
            last_posting += curr_postings
    if last_term:
        last_posting = list(sorted(set(last_posting)))
        merged_index.append(last_term, last_posting)
```

3 布尔联合检索 (10%)

3.1 _get_postings_list()

给定一个term，返回对应的条目列表。（不会遍历整个index文件）

```
def _get_postings_list(self, term):
    # 开始地址，posting的条数，字节数
    start_position, n_postings, length_in_bytes =
self.postings_dict[term]
    self.index_file.seek(start_position)
    return self.postings_encoding.decode(
        self.index_file.read(length_in_bytes))
```

3.2 sorted_intersect()

对两个排序好的list做Intersect，最后返回排好序的结果。

```
def sorted_intersect(list1, list2):
    idx1 = idx2 = 0
    intersect = []
    while idx1 < len(list1) and idx2 < len(list2):
        if list1[idx1] < list2[idx2]:
            idx1 += 1
        elif list2[idx2] < list1[idx1]:
            idx2 += 1
        else:
            intersect.append(list1[idx1])
            idx1 += 1
            idx2 += 1
    return intersect
```

3.3 retrieve()

查找同时符合几个条件的文件

- query：用空格分隔的token list (string 类型)

返回doc list

```
def retrieve(self, query):
    if len(self.term_id_map) == 0 or len(self.doc_id_map) == 0:
        self.load()

    with InvertedIndexMapper(self.index_name,
directory=self.output_dir,
```

```
                                postings_encoding=
                                self.postings_encoding) as mapper:
            result = None
            # 先把query以空格为分界分成list，然后遍历
            for term in query.split():
                term_id = self.term_id_map.str_to_id.get(term)
                if not term_id:
                    return []
                r = mapper[term_id]
                if result is None:
                    result = r
                else:
                    result = sorted_intersect(result, r)
        return [self.doc_id_map[r] for r in result]
```

4 索引压缩 (30%)

```
class CompressedPostings:
    @staticmethod
    def encode_int(gap):
        ret = [(gap & 0x7f) | 0x80]
        gap >> = 7
        while gap != 0:
            ret.insert(0, gap & 0x7f)
            gap >> = 7
        return ret

    @staticmethod
    def encode(postings_list):
        # 初始化结果列表
        encoded_postings_list = []
        encoded_postings_list += CompressedPostings.encode_int(
            postings_list[0])
        # 对每一个都做压缩
        for i in range(1, len(postings_list)):
            encoded_postings_list += CompressedPostings.encode_int(
                postings_list[i] - postings_list[i-1])
        return array.array('B', encoded_postings_list).tobytes()


    @staticmethod
    def decode(encoded_postings_list):
        decoded_postings_list = array.array('B')
        decoded_postings_list.frombytes(encoded_postings_list)

        postings_list = []
        base, n = 0, len(decoded_postings_list)
```

```
        idx = 0
        while idx < n:
            gap = 0
            while idx < n and (decoded_postings_list[idx] & 0x80) == 0:
                gap = (gap << 7) | (decoded_postings_list[idx] & 0x7f)
                idx += 1
            gap = (gap << 7) | (decoded_postings_list[idx] & 0x7f)
            idx += 1

            posting = base + gap
            postings_list.append(posting)
            base = posting
        return postings_list
```

5 额外的编码方式 (10%)

gamma-encoding :先去掉1，然后还剩n bit，然后求出n的一元码，然后把这俩拼在一起。

```
class ECCompressedPostings:
    def encode_int(gap):
        if gap == 0 or gap == 1:
            return '0'
        ret = '1' * int(log(gap, 2)) + '0' + bin(gap)[3:]
        print(ret)
        return ret

    @staticmethod
    def encode(postings_list):
        encoded_postings_list = ''
        encoded_postings_list += ECCompressedPostings.encode_int(
            postings_list[0] - (-1))
        for i in range(1, len(postings_list)):
            encoded_postings_list += ECCompressedPostings.encode_int(
                postings_list[i] - postings_list[i - 1])
        print(encoded_postings_list)
        return array.array('B', [int(encoded_postings_list[x:x+8], 2) for x in range(0, len(encoded_postings_list), 8)]).tobytes()


    @staticmethod
    def decode(encoded_postings_list):
        decoded_bytes_list = array.array('B')
        decoded_bytes_list.frombytes(encoded_postings_list)
```

```
25
26          decoded_postings_list = ''.join([bin(x)[2:].zfill(8)
27                                           for x in
    decoded_bytes_list])
28          decoded_postings_list = decoded_postings_list[:-7] +
    bin(decoded_bytes_list[-1])[2:]
29          print(decoded_postings_list)
30
31          postings_list = []
32          base, idx, n = -1, 0, len(decoded_postings_list)
33          while idx < n:
34              length = 0
35              while idx < n and decoded_postings_list[idx] == '1':
36                  length += 1
37                  idx += 1
38              if idx < n:
39                  # '111 ... 1(length)0xxx ... x(length)', length maybe 0
40                  idx = idx + 1 + length
41                  gap = int('1' + decoded_postings_list[idx-length :
    idx], 2)
42                  print(idx, gap)
43                  posting = base + gap
44                  postings_list.append(posting)
45                  base = posting
46          return postings_list
```