实验思路

管昀玫 2013750

索引构建

辅助类IdMap函数

目的是得到词~id和文档~id的映射关系

`id_to_str`和`str_to_id`是list，直接存储id和str相关的信息。在`_get_id()`中，如果当前的str不在list中，则需要append。id就是按照str加入list的先后顺序确定的。

```python
def _get_str(self, i):
        """Returns the string corresponding to a given id (`i`)."""
        ### Begin your code
        return self.id_to_str[i]
        ### End your code

def _get_id(self, s):
        """Returns the id corresponding to a string (`s`).
        If `s` is not in the IdMap yet, then assigns a new id and returns the new id.
        """
        ### Begin your code
        idx = self.str_to_id.get(s, len(self.id_to_str))
        if s not in self.str_to_id.keys():
            self.str_to_id[s] = len(self.id_to_str)
            self.id_to_str.append(s)
        return idx
        ### End your code
```

BSBI索引-解析

得到全局的映射关系后，开始分块处理，使用BSBIIndex类下的**parse_block**方法得到termID-docID对。 然后为每个块创建小的索引，使用BSBIIndex类下的**invert_write**方法由termID-docID对构建倒排表。

将每一个子目录当做一个块，`parse_block`接收子目录路径作为参数。

需要使用`os.path.join()`函数来找到当前的块，遍历这个块中的每一个文件，使用`doc_id_map`函数得到`doc_id`；遍历每一个文件的内容，使用`term_id_map`得到`term_id`，这样就可以得到每块所有的`[term_id, doc_id]`

```
        td_pairs = []
        for file_dir in sorted(os.listdir(os.path.join(self.data_dir,
block_dir_relative))):
            with open(os.path.join(self.data_dir, block_dir_relative, file_dir),
'r') as f:
                content = f.read().strip().split() # list of tokens
                #strip函数：去除空格
                doc_id = self.doc_id_map[os.path.join(block_dir_relative,
file_dir)]
                for token in content:
                    term_id = self.term_id_map[token]
                    td_pairs.append([term_id, doc_id])
        return td_pairs
```

倒排表

使用**InvertedIndexWriter**类的append方法记录postings_dict，并将posting_list写入磁盘。

目的是实现由`termID-docID`对构建倒排表。

在`InvertedIndexWriter`中，倒排表不会存储在内存中而是直接写入磁盘中。需要先将`postings_list`变成字节数组的形式，然后从文件末尾处找到该`term`应写入的起始位置，并完成写入。需要将`term`三元组信息加入`postings_dict`中。

```
# InvertedIndexWriter
    encoded_postings_list = self.postings_encoding.encode(postings_list)
    start_position_in_index_file = self.index_file.seek(0, 2)
    length_in_bytes_of_postings_list =
self.index_file.write(encoded_postings_list)
    self.terms.append(term)
    self.postings_dict[term] = (start_position_in_index_file,
len(postings_list),           length_in_bytes_of_postings_list)
```

`invert_writer`中，将解析得到的td_pairs转换成倒排表，并加入已有的倒排表中。

```
# invert_writer
td_dict = defaultdict(list)
    for t, d in td_pairs:
        td_dict[t].append(d)
    for t in sorted(td_dict.keys()):
        p_list = sorted(td_dict[t])
        index.append(t, sorted(p_list))
```

合并

`InvertedIndexIterator`主要完成一个小的读缓存的要求。这里需要利用`postings_dict`中的三元组信息从`index_file`中读出`postings_list`，与`term`一起返回。

```
        self.curr_term_pos = 0
        if self.curr_term_pos < len(self.terms):
            term = self.terms[self.curr_term_pos]
            self.curr_term_pos += 1
            start_position, n_postings, length_in_bytes =
self.postings_dict[term]
            self.index_file.seek(start_position)
            postings_list =
self.postings_encoding.decode(self.index_file.read(length_in_bytes))
            return term, postings_list
        else:
            raise StopIteration
```

在`merge()`中实现合并。合并的思路类似于归并排序。

```
last_term = None
        last_posting = None
        for curr_term, curr_postings in heapq.merge(*indices):
            if curr_term != last_term: #两个指针,比较,如果不同则append新的term
                if last_term:
                    last_posting = list(sorted(set(last_posting)))
                    merged_index.append(last_term, last_posting)
                last_term = curr_term
                last_posting = curr_postings
            else:#两个term相同的情况
                last_posting += curr_postings
        if last_term:
            last_posting = list(sorted(set(last_posting)))
            merged_index.append(last_term,last_posting)
```

布尔联合检索

实现`InvertedIndexMapper`，找到对应terms在索引文件中位置并取出它的倒排记录表。

```
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))
```

实现`sorted_intersect`函数，求交集。遍历两个有序列表并在线性时间内合并。整体思路和`merge`类似，都是利用归并排序的思想，两个指针相互，较小者后移，相同则将元素加入`intersect_list`，且两个指针同时后移。

```
    while idx1 < len(list1) and idx2 < len(list2):
        if list1[idx1] < list2[idx2]:
            idx1 += 1
        elif list1[idx1] > list2[idx2]:
            idx2 += 1
        else:
            intersect_list.append(list1[idx1])
            idx1 += 1
            idx2 += 1
    return intersect_list
```

实现`retrieve`函数，实现查询。思路为两两先做intersect，中间结果再与第三个term做intersect，以此类推。

```
        with InvertedIndexMapper(self.index_name, directory=self.output_dir,
                                 postings_encoding=
                                 self.postings_encoding) as mapper:
            result = None
            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]
```

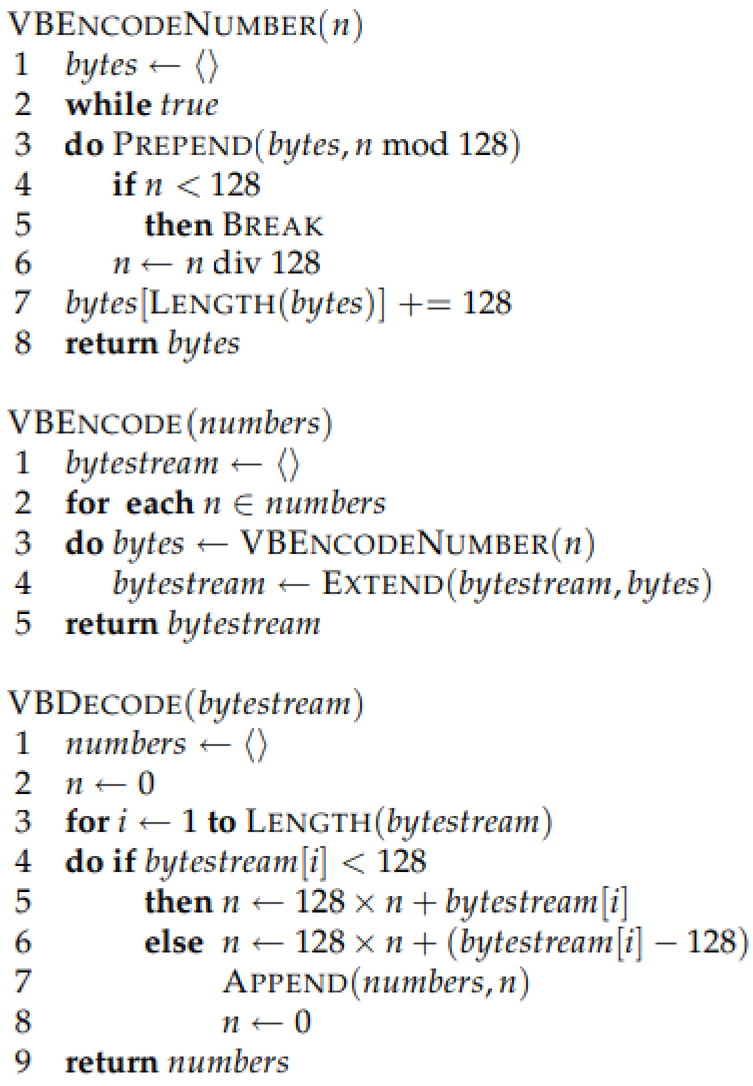
索引压缩

编码方式:

可变长字节编码：每个字节的低7位是二进制数，高位是一个决定位。编码的最后一个字节高位位置为1，位置为0。处理器一般是以字节为处理单位，所以Variable ByteCode速度快，但是处理大数据压缩比不高。

D-gaps: 对有序编号(如docid)进行差值(d-gaps)编码。（处理小数据需要小代码量，处理时间短）编码并没有定义存储数据的比特模式，所以他自身不节省任何空间。

伪代码如下所示:

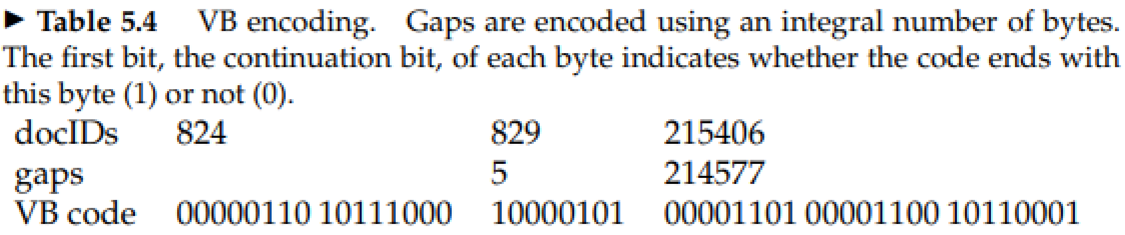


```
VBENCODENUMBER($n$)
  $bytes \leftarrow \langle\rangle$
  while $true$
  do PREPEND($bytes, n$ mod 128)
      if $n < 128$
        then BREAK
      $n \leftarrow n$ div 128
  $bytes$[LENGTH($bytes$)] += 128
  return $bytes$

VBENCODE($numbers$)
  $bytestream \leftarrow \langle\rangle$
  for each $n \in numbers$
  do $bytes \leftarrow$ VBENCODENUMBER($n$)
      $bytestream \leftarrow$ EXTEND($bytestream, bytes$)
  return $bytestream$

VBDECODE($bytestream$)
  $numbers \leftarrow \langle\rangle$
  $n \leftarrow 0$
  for $i \leftarrow 1$ to LENGTH($bytestream$)
  do if $bytestream[i] < 128$
        then $n \leftarrow 128 \times n + bytestream[i]$
        else $n \leftarrow 128 \times n + (bytestream[i] - 128)$
             APPEND($numbers, n$)
             $n \leftarrow 0$
  return $numbers$
```

范例如下：



► **Table 5.4** VB encoding. Gaps are encoded using an integral number of bytes. The first bit, the continuation bit, of each byte indicates whether the code ends with this byte (1) or not (0).

docIDs	824	829	215406
gaps		5	214577
VB code	00000110 10111000	10000101	00001101 00001100 10110001

先计算gap，然后再使用可变长字节编码，代码如下所示：

```
    @staticmethod
    def VBEncodeNum(n):
        byte = []
        while True:
            byte.append(n % 128)
            if n < 128:
                break
            n //= 128
        byte[0] += 128
        return byte[::-1]

    @staticmethod
    def VBEncode(n_list):
        b = []
        for n in n_list:
            b.extend(CompressedPostings.VBEncodeNum(n))
        return b

    @staticmethod
    def VBDecode(bs):
        n_list = []
        n = 0
        for b in bs:
            if b < 128:
                n = 128*n + b
            else:
                n = 128*n + b - 128
                n_list.append(n)
                n = 0
        return n_list

    # encode
        p = postings_list.copy()
        for i in range(1, len(p))[::-1]:
            p[i] -= p[i-1]
        vb = CompressedPostings.VBEncode(p)
        return array.array('B', vb).tobytes()

    #decode
        vb = array.array('B')
        vb.frombytes(encoded_postings_list)
        postings_list = CompressedPostings.VBDecode(vb.tolist())
        for i in range(1, len(postings_list)):
            postings_list[i] += postings_list[i-1]
```

额外的编码方式

选择Gamma编码

Elias-γ Code：结合了一元编码和二进制编码。编码数字k需要计算两个值：

$$k_d = \lfloor log_2 k \rfloor$$

$$k_r = k - 2^{\lfloor log_2 k \rfloor}$$

Number	Binary	γ encoding	Implied probability
1 = 20 + 0	`1`	`1`	1/2
2 = 21 + *0*	`10`	`010`	1/8
3 = 21 + *1*	`11`	`011`	1/8
4 = 22 + *0*	`100`	`00100`	1/32
5 = 22 + *1*	`101`	`00101`	1/32
6 = 22 + *2*	`110`	`00110`	1/32
7 = 22 + *3*	`111`	`00111`	1/32
8 = 23 + *0*	`1000`	`0001000`	1/128
9 = 23 + *1*	`1001`	`0001001`	1/128
10 = 23 + *2*	`1010`	`0001010`	1/128
11 = 23 + *3*	`1011`	`0001011`	1/128
12 = 23 + *4*	`1100`	`0001100`	1/128
13 = 23 + *5*	`1101`	`0001101`	1/128
14 = 23 + *6*	`1110`	`0001110`	1/128
15 = 23 + *7*	`1111`	`0001111`	1/128
16 = 24 + *0*	`10000`	`000010000`	1/512
17 = 24 + *1*	`10001`	`000010001`	1/512

Steps in Encoding/Decoding:

To encode a number X,

- Find the largest N, with $2^N \leq X$(greater power of 2).
- Encode N using Unary coding(i.e N zeroes followed by a one).
- Append the integer $(X - 2^N)$ using N digits in Binary.

To decode an Elias gamma-coded integer:

1. Read and count 0s from the stream until you reach the first 1. Call this count of zeroes *N*.
2. Considering the one that was reached to be the first digit of the integer, with a value of 2*N*, read the remaining *N* digits of the integer.

伪代码如下：

```
# encoding
S: input-stream of bits, C: output-stream
b <- S.top(); C.append(b)
while S is non-empty do
    k=1
    // get length of run
    while(S is non-empty and S.top()==b) do
        ++k;S.pop()
    // compute and append Elias gamma code
    K <- empty string
    while k>1
        C.append(0)
        K.append(k mod 2)
        k <- k/2
    K.prepend(1) // K is binary encoding of k.
    C.append(K)
    b <- 1-b

# decoding
C: input-stream of bits, S: output-stream
b <- C.pop()
while C is non-empty
    len <- 0
    while C.pop() == 0 do ++len
    k <- 1
    for(j<-1 to len) do k <- k*2 + C.pop()
    for(j<-1 to k) do S.append(b)
    b <- 1-b
```

`ECCompressedPostings` 实现（同时使用gap-encoding):

```
class ECCompressedPostings:
    #If you need any extra helper methods you can add them here
    ### Begin your code
    @staticmethod
    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
    ### End your code

    @staticmethod
    def encode(postings_list):

        ### Begin your code
        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 [int(encoded_postings_list[x:x + 8], 2) for x in range(0, len(encoded_postings_list), 8)]
        return array.array('B', [int(encoded_postings_list[x:x+8], 2) for x in range(0, len(encoded_postings_list), 8)]).tobytes()
        ### End your code


    @staticmethod
    def decode(encoded_postings_list):

        ### Begin your code
        decoded_bytes_list = array.array('B')
        decoded_bytes_list.frombytes(encoded_postings_list)

        decoded_postings_list = ''.join([bin(x)[2:].zfill(8) for x in decoded_bytes_list])
        decoded_postings_list = decoded_postings_list[:-7] + bin(decoded_bytes_list[-1])[2:]
        print(decoded_postings_list)

        postings_list = []
        base, idx, n = -1, 0, len(decoded_postings_list)
        while idx < n:
            length = 0
            while idx < n and decoded_postings_list[idx] == '1':
                length += 1
                idx += 1
            if idx < n:
                # '111...1(length)0xxx...x(length)', length maybe 0
                idx = idx + 1 + length
                gap = int('1' + decoded_postings_list[idx-length : idx], 2)
                print(idx, gap)
                posting = base + gap
                postings_list.append(posting)
                base = posting
        return postings_list
        ### End your code 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
```

思考感悟

在第一次实验中还是遇到挺多问题的，主要有两大块：

第一是要要把思路理顺才能完成作业整体。我当时一直没懂postings_dict, posting_list, index这三者的关系以及各自存储在哪里，理了好多遍才发现，查询是应该先插term_id_map，通过postings_dict找到记录在三元组中的信息，根据三元组的信息再找index，最后求交集。

另一个问题就是构建倒排表时读写文件的操作。由于我对python语言读写文件所用的函数不太熟悉，废了好大劲才找到合适的函数使用，emo了好久才写出来。

编码过程倒比较顺利，因为跟着伪代码编写思路就比较顺畅。

总体来说第一次实验是一次较为艰难的挑战，通过第一次实验，我对倒排索引的构建、索引的查询、压缩有了更深层次的理解，也对python语言掌握得更为熟练。

References:

1. VB编码, Gamma编码, Delta编码 https://blog.csdn.net/starvapour/article/details/111415936
2. Elias Gamma Encoding in Python https://www.geeksforgeeks.org/elias-gamma-encoding-in-python/