HW1

一、 索引构建与检索

ID map

- 补全字符串转换为数字
    - 当不存在对应的文档和id对时，需要创建并加入到字典中，同时更新列表的内容

```python
def _get_id(self, s):
    if s not in self.str_to_id:
        self.str_to_id[s] = len(self.id_to_str) # 创建新的<str, id>对
        self.id_to_str.append(s)
    return self.str_to_id[s]    # 返回str对应的id
```

- 补全数字转换为字符串

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

将倒排列表编码成字节数组

- 这里没有要补全的部分

磁盘上的倒排索引

- 这里没有要补全的部分

索引

解析

- 实现索引中的`parse_block`函数，输入数据集分块的目录，构建其中文件对应的`(term_id,doc_id)`对
    - 根据验证文件中对文档的命名方式，文档名称需要以数据块的目录开始

```
def parse_block(self, block_dir_relative):
    block_dir_path = os.path.join(self.data_dir, block_dir_relative)
    file_name_list = os.listdir(block_dir_path)
    td_pairs = []
    for file_name in file_name_list:
        doc_id = self.doc_id_map[os.path.join(block_dir_relative, file_name)]
   # 这里文件名使用相对数据集路径下的文件名，否则后续测试不能通过
        with open(os.path.join(block_dir_path, file_name), 'r') as file:
            for term in file.read().split():
                term_id = self.term_id_map[term]
                td_pairs.append((term_id, doc_id))
    return td_pairs
```

- 测试结果中，可以看到词项`you`在文档0，1中均有出现，函数正确。

```
[31]: ### Begin your code
BSBI_instance = BSBIIndex(data_dir=toy_dir, output_dir = 'tmp/', index_name = 'toy')
test_td_pairs = BSBI_instance.parse_block('0')
for td_pair in test_td_pairs:
    term = BSBI_instance.term_id_map[td_pair[0]]
    doc = BSBI_instance.doc_id_map[td_pair[1]]
    print(td_pair[0], term, doc, td_pair[1])
### End your code
```

```
0 i'm 0/fine.txt 0
1 fine 0/fine.txt 0
2 , 0/fine.txt 0
3 thank 0/fine.txt 0
4 you 0/fine.txt 0
5 hi 0/hello.txt 1
5 hi 0/hello.txt 1
6 how 0/hello.txt 1
7 are 0/hello.txt 1
4 you 0/hello.txt 1
8 ? 0/hello.txt 1
```

倒排表

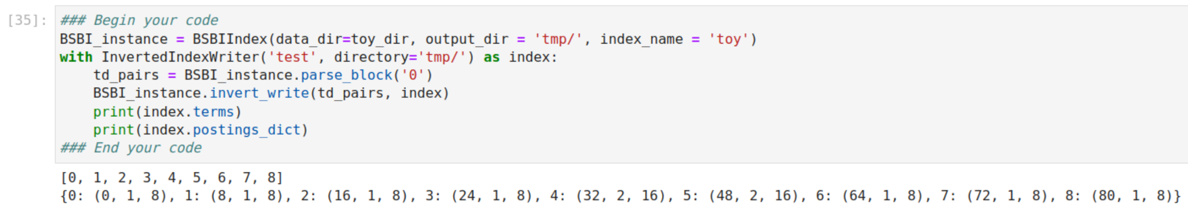
- 实现`InvertedIndexWriter`中的`append`函数
  - 对文档序列编码
  - 构建词项与索引的词典：词项对应索引在二进制文件中的偏移量、包含文档的数量、二进制索引的长度

```
def append(self, term, postings_list):
    encoded_postings_list = self.postings_encoding.encode(postings_list)
    start_pos = self.index_file.seek(0,2)    # 开始位置是文件末尾
    num_postings = len(postings_list)    # 该条索引包含的文档数
    length_in_bytes = self.index_file.write(encoded_postings_list)  # 写入文件，返回写入的字节数
    self.terms.append(term) # 更新词项
    self.postings_dict[term] = (start_pos, num_postings, length_in_bytes)   # 更新词项对应的索引信息
```

- 实现`BSBIIndex` 中的`invert_write`函数

```python
def invert_write(self, td_pairs, index):
    td_dict = defaultdict(list)
    for term, docID in td_pairs:    # 创建词项-文档ID序列的字典
        td_dict[term].append(docID)
    for term in sorted(td_dict.keys()): # 按词项排序
        postings_list = sorted(td_dict[term])   # 对应词项的按文档ID序列排序
        index.append(term, postings_list)
```

    - 测试结果中，注意到词项和字典的存储是正确的



```
[35]: ### Begin your code
BSBI_instance = BSBIIndex(data_dir=toy_dir, output_dir = 'tmp/', index_name = 'toy')
with InvertedIndexWriter('test', directory='tmp/') as index:
    td_pairs = BSBI_instance.parse_block('0')
    BSBI_instance.invert_write(td_pairs, index)
    print(index.terms)
    print(index.postings_dict)
### End your code
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
{0: (0, 1, 8), 1: (8, 1, 8), 2: (16, 1, 8), 3: (24, 1, 8), 4: (32, 2, 16), 5: (48, 2, 16), 6: (64, 1, 8), 7: (72, 1, 8), 8: (80, 1, 8)}
```

合并

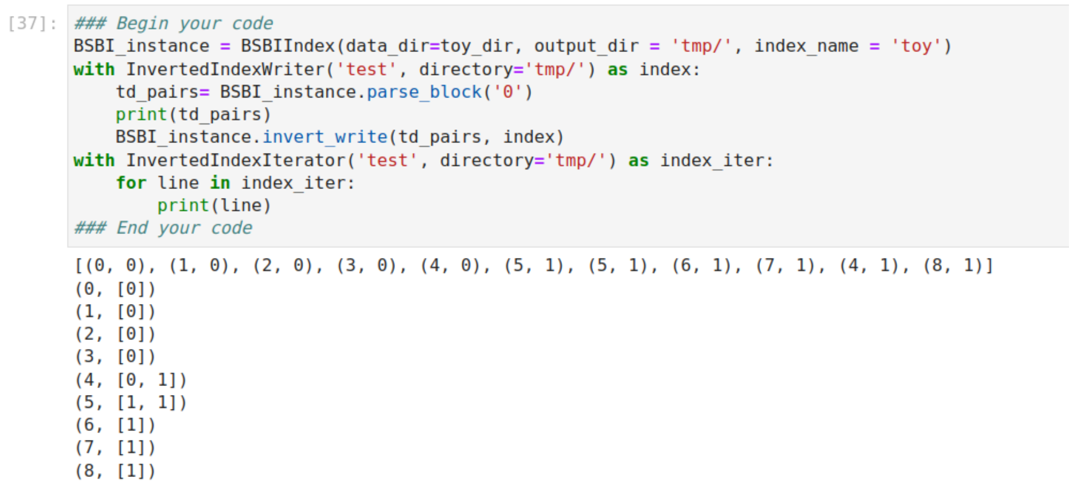
- 完成`InvertedIndexIterator`中的`_initialization_hook`函数
    - 初始化指向词项的迭代器，并记录总词项数

```python
def _initialization_hook(self):
    self.curr_term_pos = 0   # 初始化为指向第0个词项
    self.acc_term_num = len(self.terms)
```

- 完成`InvertedIndexIterator`中的`__next__`函数
    - 遍历词项，读取对应索引，并decode
    - 需要设置`StopIteration`停止读取

```python
def __next__(self):
    if self.curr_term_pos < self.acc_term_num:    # 词项未读完
        term = self.terms[self.curr_term_pos]   # 找到下一个词项
        start_pos, num_postings, length_in_bytes = self.postings_dict[term]
# 找到该词项对应的索引信息
        self.index_file.seek(0, 1)  # 从当前位置开始
        postings_list =
self.postings_encoding.decode(self.index_file.read(length_in_bytes))    # 读
取该词项对应的索引
        self.curr_term_pos += 1
        return term, postings_list
    else:
        raise StopIteration # 否则不能停下来啊（雾）
```

    - 测试结果中，注意到由`td_pairs`到词项和对应索引的建立是正确的



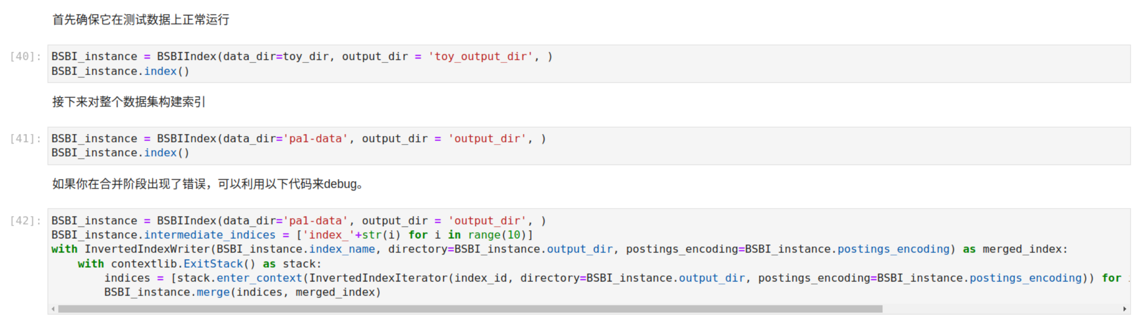
```
[37]:
### Begin your code
BSBI_instance = BSBIIndex(data_dir=toy_dir, output_dir = 'tmp/', index_name = 'toy')
with InvertedIndexWriter('test', directory='tmp/') as index:
    td_pairs= BSBI_instance.parse_block('0')
    print(td_pairs)
    BSBI_instance.invert_write(td_pairs, index)
with InvertedIndexIterator('test', directory='tmp/') as index_iter:
    for line in index_iter:
        print(line)
### End your code
```

```
[(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 1), (5, 1), (6, 1), (7, 1), (4, 1), (8, 1)]
(0, [0])
(1, [0])
(2, [0])
(3, [0])
(4, [0, 1])
(5, [1, 1])
(6, [1])
(7, [1])
(8, [1])
```

- 完成`BSBIIndex`中的`merge`函数
    - 将不同`block`得到的倒排索引合并，需要记录上一个词项，以和当前词项进行对比

```
def merge(self, indices, merged_index):
    last_term = last_postings_list = None
    for term, postings_list in heapq.merge(*indices):
        if term != last_term:   # 出现新的词项
            if last_term != None:   # 不是第一次出现新的词项
                last_postings_list = list(sorted(set(last_postings_list)))
# 去重
                merged_index.append(last_term, last_postings_list)  # 写入上一
个词项和对应的索引
            last_term = term    # 更新词项
            last_postings_list = postings_list   # 更新索引
        else:
            last_postings_list += postings_list  # 词项相同合并索引
    if last_term:   # 最后一个词项时，写入它和对应的索引
    last_postings_list = list(sorted(set(last_postings_list)))
    merged_index.append(last_term, last_postings_list)
```

    - 在一系列测试中均表现正常



首先确保它在测试数据上正常运行

```
[40]:
BSBI_instance = BSBIIndex(data_dir=toy_dir, output_dir = 'toy_output_dir', )
BSBI_instance.index()
```

接下来对整个数据集构建索引

```
[41]:
BSBI_instance = BSBIIndex(data_dir='pa1-data', output_dir = 'output_dir', )
BSBI_instance.index()
```

如果你在合并阶段出现了错误，可以利用以下代码来debug。

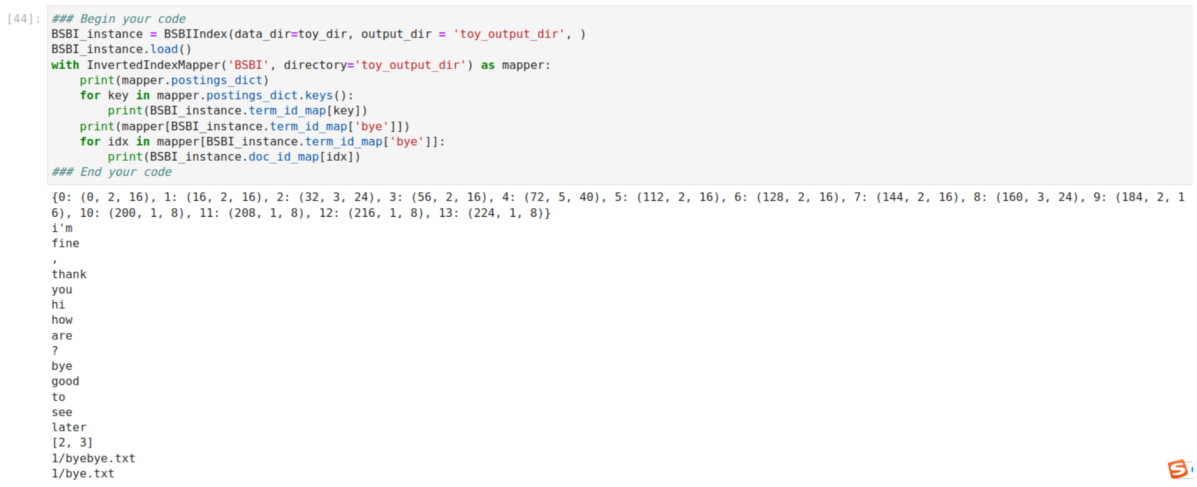
```
[42]:
BSBI_instance = BSBIIndex(data_dir='pa1-data', output_dir = 'output_dir', )
BSBI_instance.intermediate_indices = ['index_'+str(i) for i in range(10)]
with InvertedIndexWriter(BSBI_instance.index_name, directory=BSBI_instance.output_dir, postings_encoding=BSBI_instance.postings_encoding) as merged_index:
    with contextlib.ExitStack() as stack:
        indices = [stack.enter_context(InvertedIndexIterator(index_id, directory=BSBI_instance.output_dir, postings_encoding=BSBI_instance.postings_encoding)) for
        BSBI_instance.merge(indices, merged_index)
```

布尔联合检索

- 实现`InvertedIndexMapper`中的`_get_postings_list`
    - 实现从二进制文件特定位置读取词项对应的倒排索引并`decode`

```python
def _get_postings_list(self, term):
    """Gets a postings list (of docIds) for `term`.

    This function should not iterate through the index file.
    I.e., it should only have to read the bytes from the index file
    corresponding to the postings list for the requested term.
    """
    ### Begin your code
    start_pos, num_postings, length_in_bytes = self.postings_dict[term]   #
找到该词项对应的索引信息
    self.index_file.seek(start_pos) # 从索引文件的该位置开始读取，偏移量为start_pos
    return
self.postings_encoding.decode(self.index_file.read(length_in_bytes)) # 读取
length_in_bytes个字节
    ### End your code
```

- 测试词项`bye`对应的倒排索引和文档名称



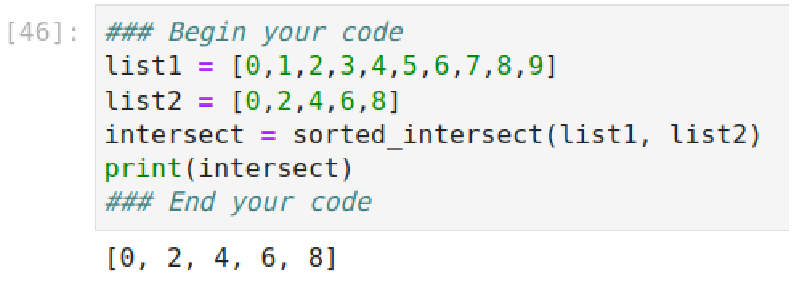
```
[44]: ### Begin your code
BSBI_instance = BSBIIndex(data_dir=toy_dir, output_dir = 'toy_output_dir', )
BSBI_instance.load()
with InvertedIndexMapper('BSBI', directory='toy_output_dir') as mapper:
    print(mapper.postings_dict)
    for key in mapper.postings_dict.keys():
        print(BSBI_instance.term_id_map[key])
    print(mapper[BSBI_instance.term_id_map['bye']])
    for idx in mapper[BSBI_instance.term_id_map['bye']]:
        print(BSBI_instance.doc_id_map[idx])
### End your code
```

```
{0: (0, 2, 16), 1: (16, 2, 16), 2: (32, 3, 24), 3: (56, 2, 16), 4: (72, 5, 40), 5: (112, 2, 16), 6: (128, 2, 16), 7: (144, 2, 16), 8: (160, 3, 24), 9: (184, 2, 1
6), 10: (200, 1, 8), 11: (208, 1, 8), 12: (216, 1, 8), 13: (224, 1, 8)}
i'm
fine
,
thank
you
hi
how
are
?
bye
good
to
see
later
[2, 3]
1/byebye.txt
1/bye.txt
```

- 实现`sorted_intersect`函数
  - 简单归并，时间复杂度为线性

```python
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(list1[idx1] > list2[idx2]):
            idx2 += 1
        else:
            intersect.append(list1[idx1])
            idx1 += 1
            idx2 += 1
    return intersect
```

  - 测试结果如下，求交正确

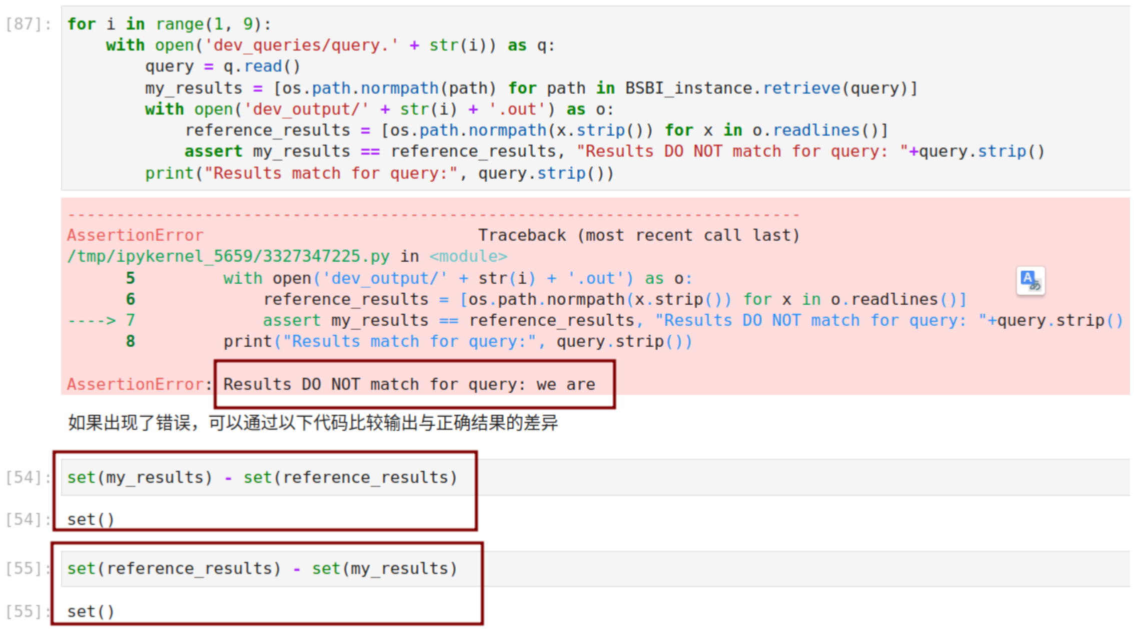


```
[46]: ### Begin your code
      list1 = [0,1,2,3,4,5,6,7,8,9]
      list2 = [0,2,4,6,8]
      intersect = sorted_intersect(list1, list2)
      print(intersect)
      ### End your code

      [0, 2, 4, 6, 8]
```

- 实现`BSBIIndex`中的`retrieve`函数
  - 对由空格隔开的词项，首先获得词项对应`id`，如果一个词项不存在，则它的倒排索引为空，最终求交一定为空。依次将新找到的倒排索引与之前求交结果继续求交

```
def retrieve(self, query):
    if len(self.term_id_map) == 0 or len(self.doc_id_map) == 0:
            self.load()
    with InvertedIndexMapper(index_name = self.index_name,
postings_encoding=self.postings_encoding, directory=self.output_dir) as
mapper:
        result = None
        for term in query.split():
            term_id = self.term_id_map[term]
            if not term_id:
                return []   # 有一个词项不在词典中，求交一定为空
            if result is None:
                result = mapper[term_id]
            else:
                result = sorted_intersect(result, mapper[term_id])  # 求交
        return [self.doc_id_map[idx] for idx in result]
```

  - 测试结果没有通过，但最终结果和查询结果交换求差的集合为空，也就是两个集合等价。这里可能是查询结果中没有滤过词项重复的情况导致的



```
[87]: for i in range(1, 9):
          with open('dev_queries/query.' + str(i)) as q:
              query = q.read()
              my_results = [os.path.normpath(path) for path in BSBI_instance.retrieve(query)]
              with open('dev_output/' + str(i) + '.out') as o:
                  reference_results = [os.path.normpath(x.strip()) for x in o.readlines()]
                  assert my_results == reference_results, "Results DO NOT match for query: "+query.strip()
              print("Results match for query:", query.strip())

      ---------------------------------------------------------------------------
      AssertionError                            Traceback (most recent call last)
      /tmp/ipykernel_5659/3327347225.py in <module>
            5         with open('dev_output/' + str(i) + '.out') as o:
            6             reference_results = [os.path.normpath(x.strip()) for x in o.readlines()]
      ----> 7             assert my_results == reference_results, "Results DO NOT match for query: "+query.strip()
            8         print("Results match for query:", query.strip())

      AssertionError: Results DO NOT match for query: we are
```

如果出现了错误，可以通过以下代码比较输出与正确结果的差异

```
[54]: set(my_results) - set(reference_results)

[54]: set()

[55]: set(reference_results) - set(my_results)

[55]: set()
```

二、 索引压缩

- `VB`编码和解码
  - 分别实现`VB`对单个数和数列进行编码

- 实现`VB`对字节序列解码
- 编码过程需要首先计算`gap`，实现进一步压缩

```python
class CompressedPostings:
    @staticmethod
    def VBEncodeNum(num):    # 对单个数进行VB编码
        byte_list = []
        while True:
            byte_list.append(num % 128)    # 一次取7个bit
            if num < 128:
                break
            num //= 128
        byte_list[0] += 128   # 最后一个字节的首位变1，表示结束
        return byte_list[::-1]    # 反转

    @staticmethod
    def VBEncode(num_list):    # 对数列进行VB编码
        byte_list = []
        for num in num_list:
            byte_list.extend(CompressedPostings.VBEncodeNum(num))
        return byte_list

    @staticmethod
    def VBDecode(byte_list):    # 对字节序列进行解码
        num_list = []
        num = 0
        for byte in byte_list:    # 字节首位为0，表示连续
            if byte < 128:
                num = 128*num + byte    # 左移7位
            else:
                num = 128*num + byte - 128    # 字节首位为1，表示到此一个数结束
                num_list.append(num)
                num = 0
        return num_list

    @staticmethod
    def encode(postings_list):
        gap_list = postings_list.copy()
        for i in range(1, len(gap_list))[::-1]:
            gap_list[i] -= gap_list[i-1]
        VBEncoded_list = CompressedPostings.VBEncode(gap_list)
        return array.array('B', VBEncoded_list).tobytes()

    @staticmethod
    def decode(encoded_postings_list):
        VBEncoded_list = array.array('B')
        VBEncoded_list.frombytes(encoded_postings_list)
        postings_list = CompressedPostings.VBDecode(VBEncoded_list.tolist())
        for i in range(1, len(postings_list)):
            postings_list[i] += postings_list[i-1]
        return postings_list
```

- 在测试中，分别输出原序列、编码后序列、解码后序列，结果正确：

[90]:
```
def test_encode_decode(l):
    print(l)
    e = CompressedPostings.encode(l)
    print(e)
    d = CompressedPostings.decode(e)
    print(d)
    # assert d == l
    # print(l, e)
```

写一些测试样例来确保文档列表的压缩和解压正常运行

[91]:
```
### Begin your code
postings_list = [100, 200, 300]
test_encode_decode(postings_list)
### End your code
```

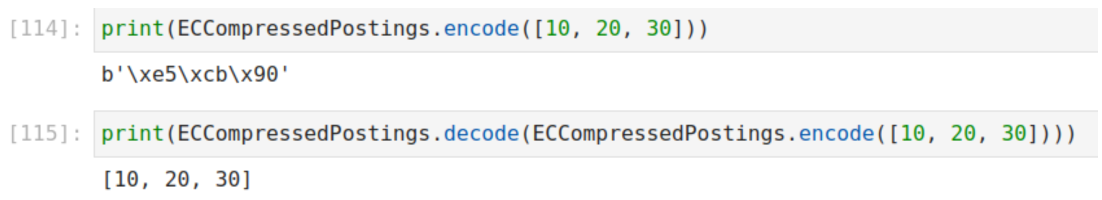
```
[100, 200, 300]
b'\xe4\xe4\xe4'
[100, 200, 300]
```

三、 额外的编码方式

- `gamma` 编码
    - 同样是先计算`gap`再进行编码
    - 编码过程是先得到`gamma`编码下二进制数组成的字符串，再将其尾部用`‘0’`把长度补为8的倍数（用于字节存储），转换为`int`后用字节存储
    - 解码过程需要特别考虑结尾引入的`'0'`，因为文档不重复，`gap`为0意味着遍历完毕
    - 该部分编程实现参考了gamma encoding

```
class ECCompressedPostings:
    # input: decimal number list
    # output: binary string gamma_encoded
    @staticmethod
    def gamma_encode(num_list):
        binary_string = ''
        for num in num_list:
            binary_num_left = bin(num)[3:]  # 删除0b和开始的1
            binary_string += 
ECCompressedPostings.unary_encode(len(binary_num_left)) + binary_num_left
        return binary_string


    # input: binary_string
    # output: list of number
    @staticmethod
    def gamma_decode(binary_string):
        num_list = []
        while binary_string != "" :
            first_zero_pos = binary_string.find('0')
            binary_num_left_length_unary = binary_string[:first_zero_pos+1]
            binary_num_left_length = 
ECCompressedPostings.unary_decode(binary_num_left_length_unary)
            if binary_num_left_length == 0:
                num_list.append(0)
            else:
                binary_num = '1' + binary_string[first_zero_pos + 1 :
first_zero_pos + 1 + binary_num_left_length]
                num = int(binary_num, 2)
                num_list.append(num)
            binary_string = binary_string[first_zero_pos + 1 +
binary_num_left_length :]
        return num_list


    # input: a decimal number
    # output: number unary_encoded
    @staticmethod
    def unary_encode(num):
        return ''.join('1'*num + '0')


    # input: num_string unary_encoded
    # output: corresponding number
    def unary_decode(unary_num_string):
        return len(unary_num_string) - 1


    @staticmethod
    def encode(postings_list):
        bytes = []
        gap_list = postings_list.copy()
        for i in range(1, len(gap_list))[::-1]:
            gap_list[i] -= gap_list[i-1]
        gap_list_binary_string = ECCompressedPostings.gamma_encode(gap_list)
        if len(gap_list_binary_string) % 8 != 0:
            gap_list_binary_string += '0' * (8 - len(gap_list_binary_string)
% 8)    # 末尾补0成为8的倍数
        bytes = []
        while gap_list_binary_string != '':
            bytes.append(int(gap_list_binary_string[ : 8], 2))
            gap_list_binary_string = gap_list_binary_string[8 : ]
        return array.array('B', bytes).tobytes()
        ### End your code


    @staticmethod
    def decode(encoded_postings_list):
        gamma_encoded_list = array.array('B')
        gamma_encoded_list.frombytes(encoded_postings_list)
        gamma_encoded_list = gamma_encoded_list.tolist()
        binary_string = ''
        for num in gamma_encoded_list:
            binary_string += bin(num)[2:]
        postings_list = ECCompressedPostings.gamma_decode(binary_string)
        end_pos = 1
        for i in range(1, len(postings_list)):
            if postings_list[i] == 0:     # gap为0时，表示当前指向末尾补0引入的0号
文档，不计入
                end_pos = i
                break
            postings_list[i] += postings_list[i-1]
        return postings_list[:end_pos]
```

- 测试结果中，先编码后解码，结果正确



```
[114]: print(ECCompressedPostings.encode([10, 20, 30]))

       b'\xe5\xcb\x90'

[115]: print(ECCompressedPostings.decode(ECCompressedPostings.encode([10, 20, 30])))

       [10, 20, 30]
```