Assignment 1 - Inverted Index Construction

ANANNYA MATHUR
2019TT10953

Implementation:

1. **Preprocessing The Data**:
   BeautifulSoup library in python was used for parsing XML.
   1. Parsing:
      A file xml-tags-info was provided, which contained document identifier and the tags carrying the indexable portion. Using the file, relevant text was extracted.
   2. Removal of punctuation marks:
      Punctuation marks and symbols were removed from the text.
      punctuation = '"!`()|-[]{};:'"\,<>./?@#$%^&*_~'"
      Symbols from the above list were replaced with a space(“ ”).
   3. Splitting of text:
      Text was further split into words using split().
   4. Stemming of words:
      Used the Porter stemmer (from https://tartarus.org/martin/PorterStemmer) to stem words before indexing.
   5. Lower-casing and Stop Words:
      A list of stop words was provided, which were removed. All the tokens were then lower-cased.

2. Building The Index Files:

The document id was split using split(“-”) to generate two terms-document name(a string) and document identifier(an integer).

1. **Dictionary(.dict):**
   For every word in the list of tokens, document names and pointers to the respective postings list were stored.
   The stopwords and compression techniques used were also stored in the dictionary.
2. **Posting List(.idx)**:
   It consisted of document identifiers for every word. It was then compressed to a bytes object and stored as a binary file.

Compression techniques used-

Compression 0: The posting list was converted to a bytes object using **pack('%dB'%len(posting_list), *posting_list).**

Compression 1:

The algorithm used-

For compression:

```
def vb_encoding(n):
    byte_stream=[]
    while True:
        byte_stream.insert(0,n%128)
        if n<128:
            break
        n=n//128
    byte_stream[-1]=byte_stream[-1]+128
    return pack('%dB' % len(byte_stream), *byte_stream)
   # return byte_stream


def compression1_encode(list_of_ids):
    byte_stream=[]
    for id in list_of_ids:
        byte_stream.append(vb_encoding(id))
    return b"".join(byte_stream)
```

For decompression:

```
def compression1_decode(byte):
    decode=0
    ids=[]
    byte= unpack('%dB' % len(byte), byte)
    for bit in byte:
        if bit<128:
            decode=decode*128+bit
        else:
```

```
            decode=decode*128+(bit-128)
            ids.append(decode)
            decode=0
    return ids
```

Compression 2:

The algorithm used-

For compression-

```
def binary_encoding(n):
    byte_stream=[]
    while True:
        byte_stream.insert(0,n%16)
        if n<16:
            break
        n=n//16
    byte_stream[-1]=byte_stream[-1]+16
    return pack('%dB' % len(byte_stream), *byte_stream)
   # return byte_stream


def compression2_encode(list_of_ids):
    byte_stream=[]
    for id in list_of_ids:
        byte_stream.append(binary_encoding(id))
    return b"".join(byte_stream)
```

For decompression-

```
def compression2_decode(byte):
    decode=0
    ids=[]
    byte= unpack('%dB' % len(byte), byte)
    for bit in byte:
        if bit<16:
            decode=decode*16+bit
        else:
```

```
            decode=decode*16+(bit-16)
            ids.append(decode)
            decode=0
    return ids
```

Compression 3:
I applied compression technique 1 on the postings list and then used snappy.compress() to further compress.
For decompression,
I first used snappy.decompress() and then used the decompression technique of the compression-1 to retrieve the original postings list.

CALCULATION OF INDEX SIZE RATIO(ISR):
|C|= 1.28MB
|D|=19.3MB

Before compression,
|P|=4.23MB
ISR=18.3

After compression-1,
|P|=2.83MB
ISR=17.3

After compression-2,
|P|=2.22MB
ISR=16.8

After compression-3,
|P|=0.138MB
ISR=15.2

COMPRESSION SPEED:
Compression 1: 1000ms
Compression 2: 424ms
Compression 3: 673ms

3. BOOLEAN RETRIEVAL:

Both the .dict and .idx files were loaded. Stopwords, which were stored in the .dict file, were removed from the query terms after stemming. The .idx file was uncompressed according to the compression technique used while building it. Punctuation marks and symbols were also removed from the query terms and replaced with " ". Using the .dict file, pointers to the postings list were fetched. For multi-word queries, I implemented the following function to compute the list of common documents-

```
def intersect(docs1, docs2):
    if docs1 is not None and docs2 is not None:
        return list(set(docs1)&set(docs2))
    else:
        return []
```

Query Speed Calculation:

$|Q|=56$

$|T_Q|=751068$ microseconds

Query speed= 13411.9 microseconds/query