Assignment 1

Inverted Index

2018CS10388
Sharique Shamim

Directory Structure

Inside the folder 2018CS10388

Source Files

- `utils.py`: Contains functions to read, write file, tokenize and stem text, encode gap and decode gap in posting list and compress and decompress posting lists.
- `inv_idx.py`: Code for creating the inverted index and compressing the posting list using given method, using the corresponding compression function from utils.
- `boolsearch.py`: Code to solve queries.
- `PorterStemmer.py`: Code for Porter Stemmer.

Bash Scripts

- `build.sh`: Empty (Since Python)
- `invidx.sh`: Creates inverted index using given arguments
- `boolsearch.sh`: Solves the queries using given arguments.

Report

- `2018CS10388.pdf`: Report containing Implementation Details
- `README`: README

How to run the Code?

To create inverted index do:

bash inivdx.sh [coll-path] [indexfile] [stopwordfile] {0|1|2|3|4|5} [xml-tags-info]

To solve queries do:

bash boolsearch.py [queryfile] [resultfile] [indexfile] [dictfile]

Document Processing

BeautifulSoup is used to parse the xml file. The content within every given tag is processed by first splitting using a regex expression, followed by removing stopwords and then stemming using Porter Stemmer.

Metrics

Size of the entire collection = 515.3 MB

Inverted Index - C0

See create_invidx function in invidx_cons.py for inverted index construction.

Metrics

- Size of dictionary file = 3.3 MB
- Size of posting lists file = 64.9 MB
- Time for inverted index construction = 701 seconds
- Index Size Ratio (ISR) = (3.3 + 64.9)/515.3 = 0.132
- Compression Speed : No Compression C0
- Query Processing Speed = 0.001 seconds

Gap Encoding

```
def encode_gap(pl):
    pl = sorted(list(pl))
    gap_pl = [pl[0]]
    for i in range(1, len(pl)):
        gap_pl += [pl[i] - pl[i-1]]
    return gap_pl
```

Here pl is the posting list which is first sorted and then gaps are introduced between consecutive document ids. Decode works just the opposite

C1: Variable Byte Encoding

```
def compress_1(nlist):
    def helper(n):
        bl = []
        while True:
            bl.insert(0, n % 128)
            if n < 128:
            break
            n = n // 128
        bl[-1] += 128
        return pack('%dB' % len(bl), *bl)
    bytes_list = []
    for n in nlist:
        bytes_list.append(helper(n))
    return b"".join(bytes_list)
```

Here nlist is a gap-encoded posting list. All the gaps are encoded using helper function in which the 7 bits are pocessed using n % 128 (since $2^7 = 128$), and n is reduced to n // 128. Once all the 8 bits chunks are obtained it is packed using struct. Finally the bytes for all the gaps are appended. During decoding first struct.unpack is used followed by reading bytes from bytestream while present which is converted to integer by multiplying by 128. (See decompress_1 in utils for more details).

Metrics

- Size of dictionary file = 3.3 MB
- Size of posting lists file = 38.4 MB
- Time for inverted index construction = 701 seconds
- Index Size Ratio (ISR) = (3.3 + 38.4)/515.3 = 0.0809
- Compression Speed : 11.2 seconds
- Query Processing Speed = 0.0001 seconds

C2: Elias Delta Encoding

```
def l(x): return len(bin(x)[2:])
def U(x): return '1'*(x-1) + '0'
def lsb(x, y): return bin(x)[2:][-y:]
v1 = l(n)
v2 = l(v1)
s = U(v2) + lsb(v2, v2-1) + lsb(n, v1-1)
s = bytes(s, 'utf-8')
return pack("I%ds" % (len(s),), len(s), s)
```

Here n is the integer that is to be compressed and l(x), U(x), lsb(x, y) are corresponding required functions. See utils for more details.

Metrics

- Size of dictionary file = 3.3 MB
- Size of posting lists file = 94.4 MB
- Time for inverted index construction = 701 seconds
- Index Size Ratio (ISR) = (3.3 + 94.4)/515.3 = 0.189
- Compression Speed : 21.3 seconds
- Query Processing Speed = 0.0001

C3: Snappy Encoding

```
def compress_3(nlist):
    a = [str(n) for n in nlist]
    return snappy.compress(' '.join(a))
```

Here, nlist is the gap encoded posting lists. Used snappy.compress to compress the posting list by converting it to a string of integers. Used snappy.uncompress to decompress it. (See utils for more details)

Metrics

- Size of dictionary file = 3.3 MB
- Size of posting lists file = 60.4 MB
- Time for inverted index construction = 701 seconds
- Index Size Ratio (ISR) = (3.3 + 60.4)/515.3 = 0.123
- Compression Speed : 9.4 seconds
- Query Processing Speed = 0.0001 seconds

C4: Golomb Encoding

```
k = math.floor(math.log(n, 2))
q = n // 2**k
r = n - q*(2**k)
```

Metrics

- Size of dictionary file = 3.3 MB
- Size of posting lists file = 91.4 MB
- Time for inverted index construction = 701 seconds
- Index Size Ratio (ISR) = (3.3 + 91.4)/515.3 = 0.183
- Compression Speed : 13.1 seconds
- Query Processing Speed = 0.0001 seconds

C5: Rice Encoding

Not implemented

Note: All compression algorithms can be found in `utils.py`

Boolean Retrieval

Retrieve the compression method, stopwords and document mapping from dictionary file.

Process the query file. For each query first parse it by tokenization and stemming. For each term in the query retrieve the posting list, decompress it and keep taking the intersection of all the posting list for each term. Map the final posting list to corresponding document ids and return it.