README

1.Inverted Index Construction

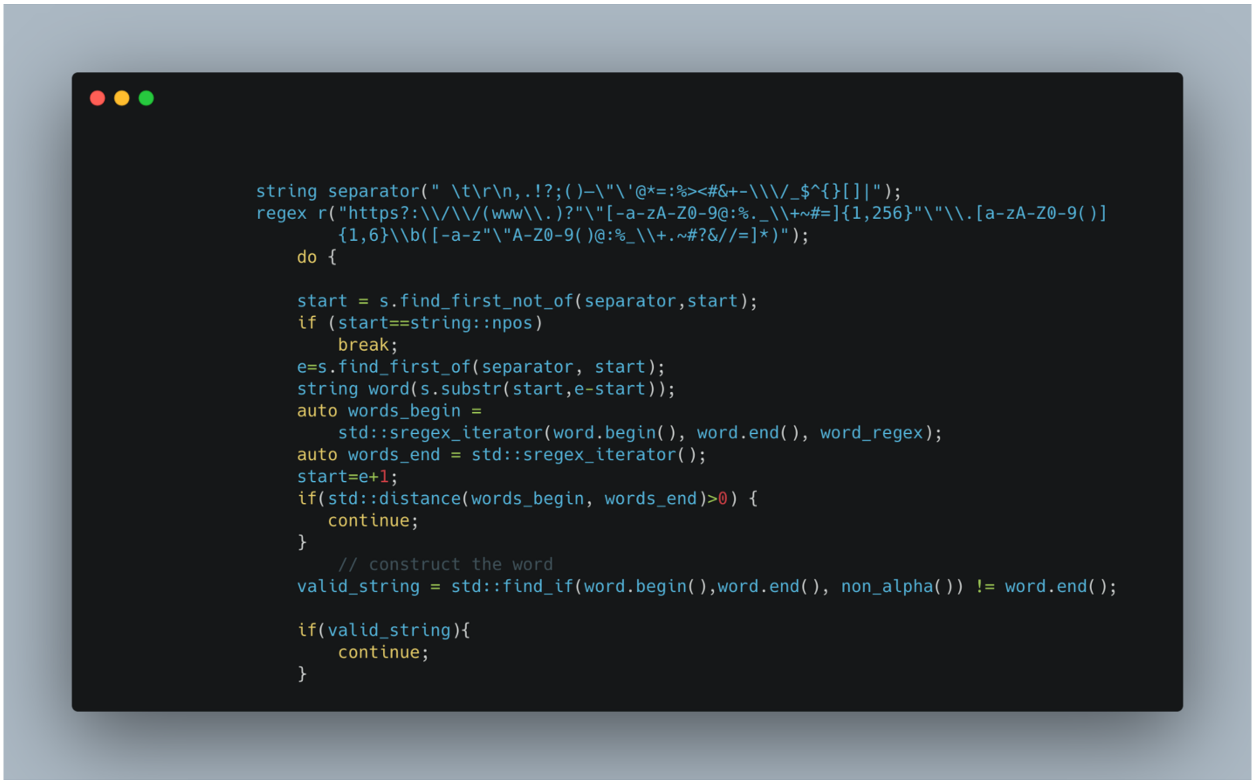
Stages involved in constructing index for the given data set follows as,

1. Parsing file to create intermediate postings.
2. Merge and sort into one file used to construct actual index.
3. Construct actual postings to generate inverted list for the word term.
4. Generate page table to retrieve URL of the given docID.
5. Build Lexicon that stores metadata for each word term found in the data set.

2.Design

2.1. Creating Intermediate posting

- Used map container in C++ to count the occurrences of word term present in dataset `msmarco-docs.trec.`
- And, used map container to map between docid and URLs to output into page_table.txt.
- Utilized Regex to filter non-English characters and extract URL for the document.



```
string separator(" \t\r\n,.!?;()—\"\'@*=:%><#&+-\\\/_$^{}[]|");
regex r("https?:\\/\\/(www\\.)?"\"[-a-zA-Z0-9@:%._\\+~#=]{1,256}"\"\\.[a-zA-Z0-9()]
        {1,6}\\b([-a-z"\"A-Z0-9()@:%_\\+.~#?&//=]*)");
    do {

    start = s.find_first_not_of(separator,start);
    if (start==string::npos)
        break;
    e=s.find_first_of(separator, start);
    string word(s.substr(start,e-start));
    auto words_begin =
        std::sregex_iterator(word.begin(), word.end(), word_regex);
    auto words_end = std::sregex_iterator();
    start=e+1;
    if(std::distance(words_begin, words_end)>0) {
      continue;
    }
        // construct the word
    valid_string = std::find_if(word.begin(),word.end(), non_alpha()) != word.end();

    if(valid_string){
        continue;
    }
```

.

2.2. Creating Intermediate posting

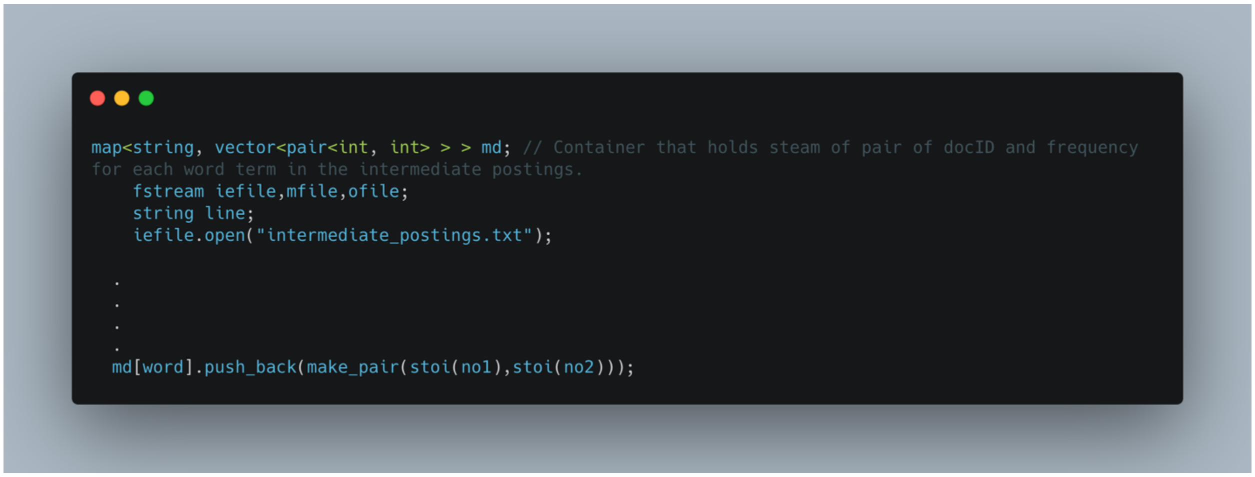
Used linux sort to sort each misc_* files generated by generate_postings.cpp.
First, sorted in batch size of 100 and then piped into single file (s_misc1.txt ….smisc9.txt)with below operation,
Sort –parallel=15 misc1.txt>s_misc1.txt.
This executed 2 mins for each file. Hence, cumulatively it took 20 mins for initial first phase sort.

Then merged all s_misc files into one using,
Sort -m smisc_* > intermediate_postings.txt.

2.3. Construct Inverted List

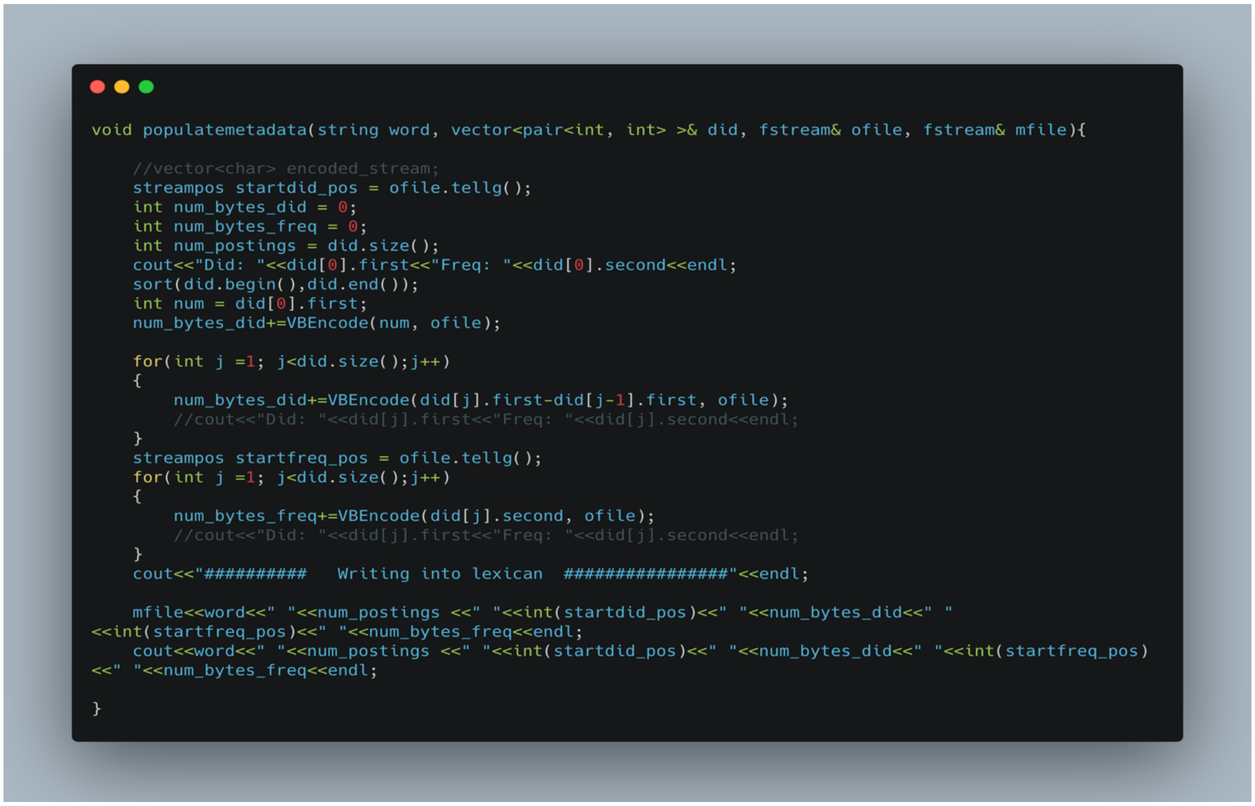
- For each word term in the intermediate_posting, all DocID and frequency is pushed as pair into the vector. This is method was helpful, to encode all the DOCID.

- If buffer size exceeds, for all the items in the map containing posting in the form (Word, <DocID, Freq>), metdata entry is created for all terms and the container is reset.



```
map<string, vector<pair<int, int> > > md; // Container that holds steam of pair of docID and frequency
for each word term in the intermediate postings.
    fstream iefile,mfile,ofile;
    string line;
    iefile.open("intermediate_postings.txt");

  .
  .
  .
  .
  md[word].push_back(make_pair(stoi(no1),stoi(no2)));
```

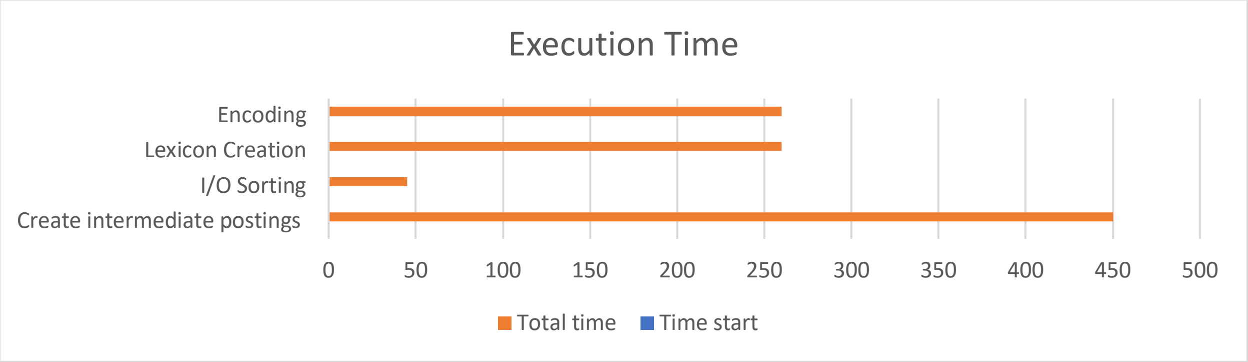
- For each word term in the container, respective docID stream is encoded using varbyte compression as it is suitable for this implementation that makes use of vector of docID.



```
void populatemetadata(string word, vector<pair<int, int> >& did, fstream& ofile, fstream& mfile){

    //vector<char> encoded_stream;
    streampos startdid_pos = ofile.tellg();
    int num_bytes_did = 0;
    int num_bytes_freq = 0;
    int num_postings = did.size();
    cout<<"Did: "<<did[0].first<<"Freq: "<<did[0].second<<endl;
    sort(did.begin(),did.end());
    int num = did[0].first;
    num_bytes_did+=VBEncode(num, ofile);

    for(int j =1; j<did.size();j++)
    {
        num_bytes_did+=VBEncode(did[j].first-did[j-1].first, ofile);
        //cout<<"Did: "<<did[j].first<<"Freq: "<<did[j].second<<endl;
    }
    streampos startfreq_pos = ofile.tellg();
    for(int j =1; j<did.size();j++)
    {
        num_bytes_freq+=VBEncode(did[j].second, ofile);
        //cout<<"Did: "<<did[j].first<<"Freq: "<<did[j].second<<endl;
    }
    cout<<"##########   Writing into lexican  ################"<<endl;

    mfile<<word<<" "<<num_postings <<" "<<int(startdid_pos)<<" "<<num_bytes_did<<" "
<<int(startfreq_pos)<<" "<<num_bytes_freq<<endl;
    cout<<word<<" "<<num_postings <<" "<<int(startdid_pos)<<" "<<num_bytes_did<<" "<<int(startfreq_pos)
<<" "<<num_bytes_freq<<endl;

}
```

3. Execution

For 22GB dataset(3.5 million docs) given, program able to parse about (3 million docs)20GB in 6.5 hours.



Linux sort completed in about 45 mins to sort and merge into single (intermediate_postings.txt) file.
Finally, meta.cpp constructed inverted list and lexicon in about 5 hours.

FILE	SIZE	DESCRIPTION
inverted_list.dat	~2.3GB	Encoded docid, freq stream for each word term
lexican.txt	~450MB	Contains metadata of each word term for the inverted list
Page_table.txt	~225MB	Contains mapping of docid and URL.

4. Limitations.

- Inverted list is constructed only for English characters.
- Any numeric term is ignored to create intermediate postings.
- Performance of program to create intermediate postings is not satisfactory due to delay in file read/write, and insertion of element into container to store the occurrences.
- Inbuilt sorting is used to sort the docIDs for preparing input to Varbyte encoder which takes hit in the performance of meta.cpp to construct inverted_list.