IR PROJECT REPORT

Swathi Kedarasetty - S20180010172

TASKS DONE:

- Searching for a query in the set of documents
- Ranking the obtained search results and providing the top 10 relevant documents to the user.

ALGORITHM / IMPLEMENTATION DETAILS:

- Constructed inverted index
- Implemented tf-idf (vector space model) for ranking the documents.

DATASET:

Movie reviews news dataset containing 2026 documents.

LANGUAGE:

C++

LIBRARIES/ HEADER FILES USED:

`bits/stdc++.h` → to include the basic STL like vector, map in C++

INVERTED INDEX IMPLEMENTATION:

- Trie data structure for storing the terms/ vocabulary.
- The leaf node of each word in the trie points to the corresponding word's posting list.
- The posting list is implemented using Binary Search Tree (BST).
- The posting list contains all the document IDs in which the corresponding term occurs.
- Prototype of a node in the BST:

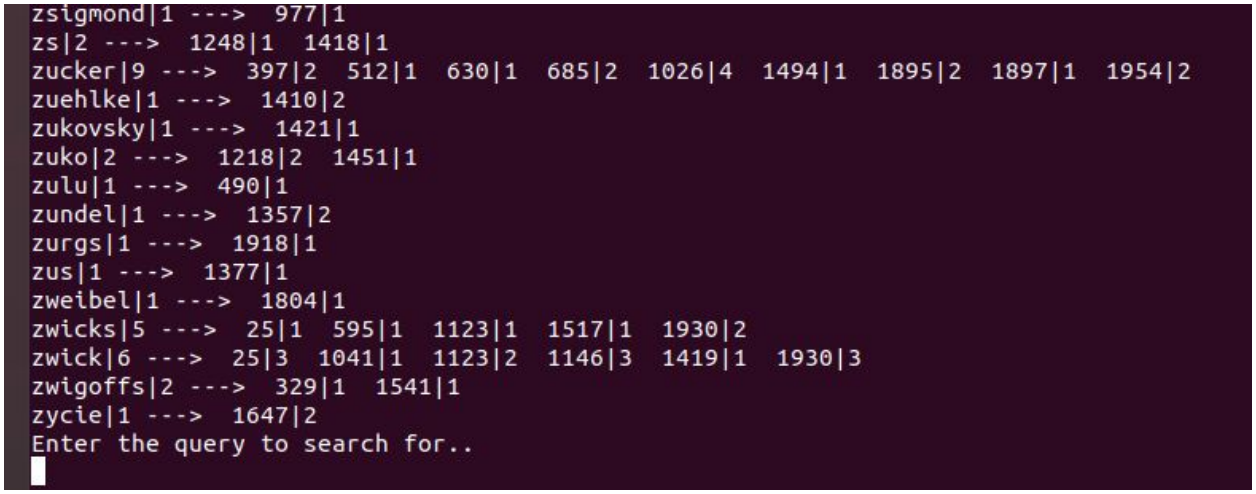
```
class bstNode{
public:
      int data;   //docID
      int freq;   //number of times that the word occurs in this doc..
      bstNode* left, *right;
};
```

- The nodes in the BST are ordered by the docIDs and each node stores `freq`, which is the number of occurrences of that word in the current doc. This is the term frequency of the word in the given doc.
- Prototype of a node in the trie:

```
class TrieNode {

public:

      TrieNode* next[26];
      bool isWordEnd;
      int docFreq;
      BST* postingList;
};

class BST{

private:
      bstNode* root;

};
```

- For the leaf nodes in the trie, `isWordEnd` will be true and the `docFreq` will be non-zero. `docFreq` is the number of documents in which the corresponding word occurs, this is nothing but the size of the posting list or, the number of nodes in the BST.
- `docFreq` is populated by incrementing it's value by 1, everytime when a node is inserted in the BST. For all the non leaf nodes in the trie, the `docFreq` is zero and `isWordEnd` is false.

SAMPLE OUTPUT FROM THE INVERTED INDEX:



So, the inverted index obtained will give us the following information:

- List of all terms in the corpus
- Number of docs in which each term occurs (doc frequency)
- The documents in which each term occurs (docIDs of the corresponding documents from the posting list)
- Number of times each term occurs in those documents (term frequency)

This information obtained from the inverted index, is of great use in the implementation of tf-idf for ranked retrieval.

TF-IDF IMPLEMENTATION:

- Firstly, the possible set of relevant documents for the given query is obtained. This is obtained by taking the union of all docIDs in which each query term occurs.
- The query and the relevant documents are represented as vectors in ‘n’ dimensional space where ‘n’ is the number of terms in the query.
- The tf-weight (which is taken as 1+log(tf) here) and the idf-weight of each term in the query are multiplied and the result is represented as a vector.

Eg: if the query is

“Zwick zulu zwick”

For the query,

word	tf-raw	Tf-wght = 1+log(tf-raw)	df	Idf = log(N/df)	weight = tf-wght * idf
zwick	2	1.69315	6	5.82008	9.85426
zulu	1	1	1	7.61431	7.61431

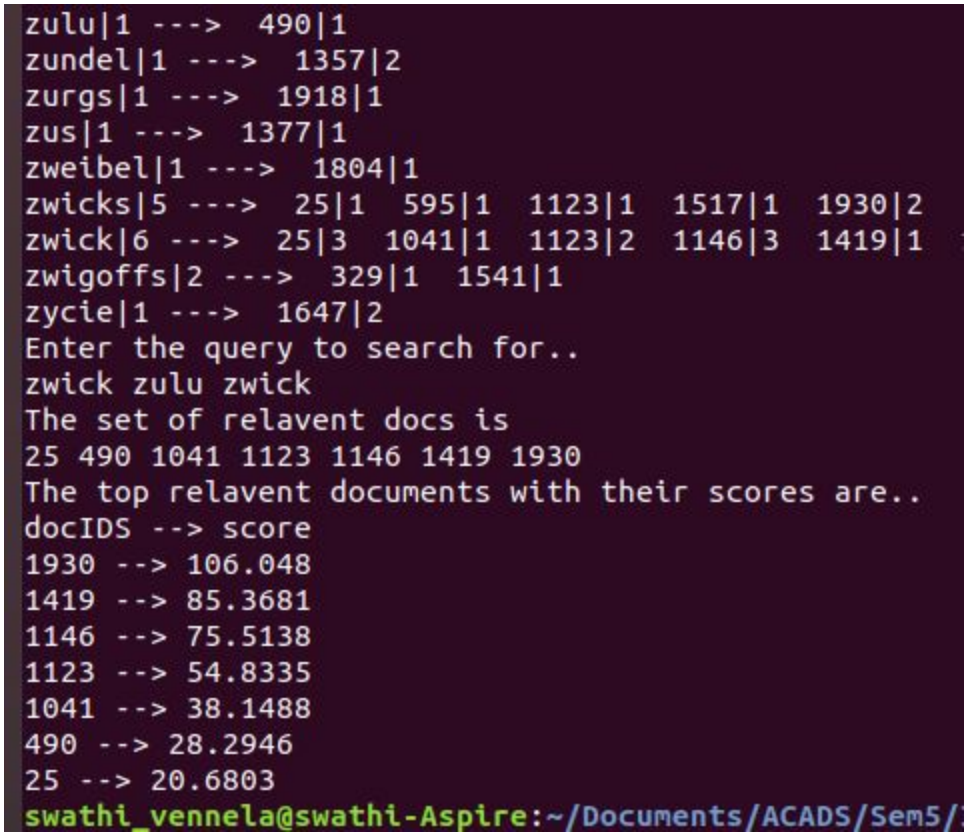
Let `v1` be `[9.85426 7.61431]`
For the relevant documents, we obtain the normalized weights of each term as shown

	Zwick	Zulu
docID 25		
docID 490		
docID 1041		
docID 1123		
docID 1419		
docID 1930		

Each entry in the matrix `mat[i][j]` corresponds to the weight of the term `j` in the doc `i`. I've taken the normalized log frequency weight for this.

Now, multiplying the `ith` row in `mat` with the vector `v1` transpose (dot product) gives the relevance score of `i th` document for the given query.

Based on the scores obtained, the top 10 relevant documents are shown to the user as follows:



This way, the user is given the set of relevant documents for the query entered, with the results ranked based on the tf-idf score obtained.