

EC504 Project Report

Meghna Murali, Nirmal Patel, Jaskaran Gahunia

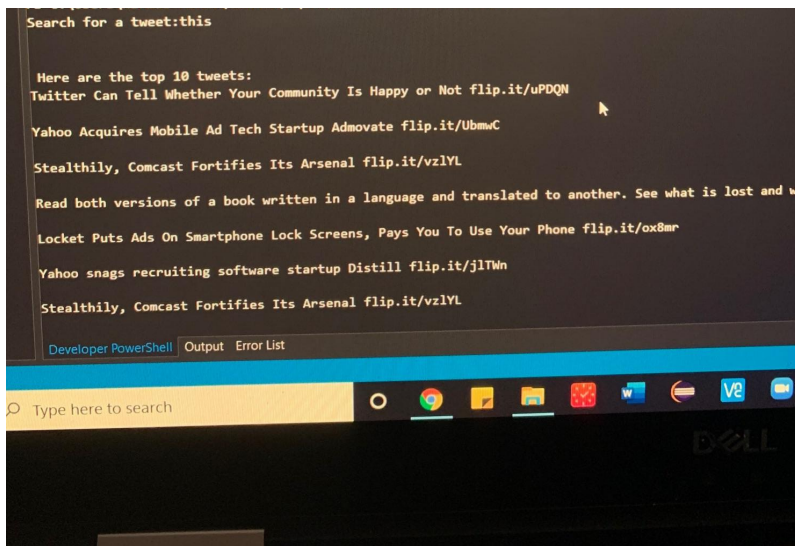
For this semester our group, consisting of Meghna, Nirmal and Jaskaran, have chosen to work on the Twitter Keyword Search project. The goal with the Twitter Keyword Search project is to write a keyword search engine that accepts a text file containing a number of tweets as input (each line is one tweet), and provides a simple user interface for querying the list of tweets against keywords, the system replies to keyword queries with the top 10 tweets that are most relevant to user query keyword(s). We are given a practice text file that contains a limited amount of tweets, along with a csv file that will be used and implemented into our project. The project should result as such:

1. Find the list of tweets that contains at least one word in the given query
2. Compute the similarity of the tweets in the list with the query keywords according to the number of times they pass in the tweets
3. Select the top 10 most similar tweets
4. Display the tweet texts to the querying user in the relevance order

For this project we are given the freedom to choose any data structure that we see fit, with efficiency and functionality being the top priority; the more efficient the algorithm, the better. Our group has decided to store the database of tweets in an unordered map. We chose to do so as in reality, a Twitter search engine receives a large number of search queries. As the main objective of our design is to perform efficiently, we chose the unordered_map as it has a lookup time complexity of $O(1)$. The database of tweets was stored in a csv file to be read into the program. This was done so by reading each line and feeding it into our countFrequency()

function which built a “dictionary” of tweets and their corresponding location in the csv file. This is then passed into an unordered map. An unordered map is implemented using a hash table, which uses the aid of a hash function to index into a value. The database of unique words in a particular tweet is then stored in a vector. The way this works is that each key will store unique words and values will go into vectors (vector<int>). With every search query, each word is looked up in the vector of unordered maps and a similarity score is given to each tweet depending on how many words were identified. This similarity score will determine the top 10 tweets based on the current search query by indexing through the file to access the tweets associated with their respective tweet numbers.

Taking into account the maximum number of words per tweet and number of tweets, we get a time complexity of $O(n + m)$. When storing and accessing data, we need to parse the data by separation of the tweet, which is represented by n . Whereas m represents strings which are stored into an unordered_map for each tweet. What should also be noted is that the quicksort for getting the top 10 tweets will be denoted by $O(\log(n))$. An example of the program is pictured below:



```
Search for a tweet:this

Here are the top 10 tweets:
Twitter Can Tell Whether Your Community Is Happy or Not flip.it/uPDQN
Yahoo Acquires Mobile Ad Tech Startup Admivate flip.it/UbmwC
Stealthily, Comcast Fortifies Its Arsenal flip.it/vzLYL
Read both versions of a book written in a language and translated to another. See what is lost and wh
Locket Puts Ads On Smartphone Lock Screens, Pays You To Use Your Phone flip.it/ox8mr
Yahoo snags recruiting software startup Distill flip.it/jlTWn
Stealthily, Comcast Fortifies Its Arsenal flip.it/vzLYL

Developer PowerShell: Output Error List
```

The main challenge in building the word search was cross referencing in an efficient way. The user enters an input and references each tweet and increments the occurrence of a word from the input. We needed to find a method to store occurrence values with each tweet and efficiently access the given string. Our initial plan was to make each occurrence a node that would include and integer an element. However, the issue with making a linked list is the need to traverse through the nodes instead of dynamically accessing a given string. Potential improvements we would have made include keeping track of the top 10 tweets that are most relevant to the search query while simultaneously matching strings of the input and tweets instead of sorting the sums after the matching is done to improve the algorithm's efficiency. Another, more ambitious, improvement that could have been made is for the program to recognize words that may not be exactly the same, yet fall under the same category. For example, if a search query included the word "car," tweets pertaining to various car models such as "Jeep" or "Honda" would be included while the program gave a similarity score. Although we were not able to include these features due to time constraints, they would have been an efficient addition to the project.