EC 504 Twitter Keyword Search Report

Yuan Wang

Xiang Liu

Tian Wang

Github: https://github.com/GreatTwang/EC504project

Abstract

This paper claims how to construct a good keyword search engine based on Twitter database. We use inverted index to store twitter data and compared four data structures and two kinds of sorting algorithms for building the search engine.

1. Introduction

A search engine which is a computer software used to search data such as text or a database for specified information [1]. This paper mainly claims how to construct a keyword engine based on the database.

In our search engine, the main function is to find the sentences in the database that contain at least one word we input. Then, compute the similarity of the sentences we found with our input. Finally, it can show the top 10 most similar tweets.

This paper is organized as follows. In Section 2, data structures and algorithms for data storage will be explained in detail. Then, the search engine will include two parts, where similarity map is introduced in Section 3.1 and top 10 searching algorithms are introduced in Section 3.2. After that, Section 4 will be complexity analysis and test results are shown in Section 5. Finally, Section 6 will be conclusion.

2. Data storage

(1) Algorithm

The key idea of implementation for this project is inverted index. Inverted index is a database index, which can store a mapping in a table. The mapping is from the words to their location. For example, if we have the documents as follow:

document number	**document content**
1	I am a student in BU
2	I study engineering in BU
3	My major is engineering
4	I want to study computer science

By using inverted index, we have another table:（List only some keywords）

Keywords	**Document number**
I	1,2
am	1
student	1
BU	1,2

major	3
engineering	2,3
computer	4
science	4
study	2,4

If we want to search a word: engineering, we use the second table to know the keyword ‘engineering’ is in document 2 and document 3. Then, in the first table, we can find document 2 and document 3.

By using inverted index, we can get each keyword’s location, which shorten our search time. Since we do not need to search the first list completely.

(2) Data structure

For inverted index mapping, we have two choices: map and unordered_map.

The differences between map and unordered_map are shown in figure 1.

```
                  | map             | unordered_map
-------------------------------------------------------
Ordering          | increasing  order   | no ordering
                  | (by default)        |

Implementation    | Self balancing BST  | Hash Table
                  | like Red-Black Tree |

search time       | log(n)              | O(1) -> Average
                  |                     | O(n) -> Worst Case

Insertion time    | log(n) + Rebalance  | Same as search

Deletion time     | log(n) + Rebalance  | Same as search
```

Figure 1: the difference between map and unordered_map[6]

Unordered_map internally uses hash table. hashing is a technique that is used to uniquely identify a specific object from a group of similar objects. In the hashing large keys are converted into small keys by using hash function. The values are stored in a data structure called hash table[2].

Map internally uses red-black tree. Red-Black Tree is a self-balancing Binary Search with three rules. 1) each node has a color either red or black, 2) Root of tree is always black, 3) there are two adjacent red node which means the red node can not have a red parent or child[3].

For value representation in inverted index map, we also have two choices: vector and map. Vector is used to record keyword occurrence places while map is used to count number of keyword occurrences.

According their properties, vector appends new entries fast, while Map saves space in following search and possibly reduces time when one sentence has more than one same keyword

(3) Implementation

```
// data structure 1
unordered_map<string,vector<word_position> > dictionary;

// data structure 2
unordered_map<string,unordered_map<string,int> > dictionary2;

// data structure 3
map<string,vector<word_position> > dictionary3;

// data structure 4
map<string,map<string,int> > dictionary4;
```

Figure 2: the data structure we used in these project

Like the figure 2 shown above, we combine these options and have 4 combinations.

3. Search engine

(1) Update similarity

We use a map to record similarity between the input and information in the database. The similarity is defined as the amount of words is the same between the input and tweet in the database. For instance, if there are six words as input and there is tweet in database conclude all of these five words then the similarity is 6.

For each keyword, we search it in inverted index map and iterate over all lines containing this word.

Line content	**Number of occurrences (Search:** I am a student in BU)
I am a student in BU	6
I study engineering in BU	3
I want to study computer science	1

(2) Find top 10 similar tweets

As the project instructions mentioned that top 10 similar tweets should be shown up after input the keyword. Two algorithms were applied here, bucket sort and priority queue to implement of selecting top ten tweets.

Priority queue is an extension of queue with three properties: 1) Every item has a priority associated with it. 2) An element with high priority is dequeued before an element with low priority. 3) if two elements have the same priority, they are served according to their order in the queue.

We maintain a min heap to implement priority queue. Here we have two choices. One is a heap with size N, which is like heap sort. The other is a heap with size 10, which can reduce time complexity to O(N log 10).

On the other hand, as the input keywords is given, we can also use bucket sort to improve efficiency further. Bucket sort is a comparison sort algorithm that operates on elements by dividing them into different bucket and then sorting these buckets individually. Each bucket is sorted individually using a separate sorting algorithm or by applying the bucket sort algorithm recursively [4].

4. Complexity analysis

(1) Load data

Suppose there are N lines in the csv file and average M words in each line, then the time complexity will be O(MN) .

The invert index map takes O(1) complexity for put and append operations.

(2) Compute similarity

Suppose the length of keywords we input is N, and each keyword relates to M lines in the invert index map, then the time complexity for building similarity map is O(MN).

(3) Get top 10 results

Suppose there are N records in our similarity map：

Algorithm 1: Priority queue(size N) Time complexity: O(N log N)

Algorithm 2: Priority queue(size 10) Time complexity: O(N log 10)

Algorithm 3: Bucket sort Time complexity: O(N)

5.Test results

(1) Load data

Data structure	time(s)
map + vector	13.45
unordered_map + vector	5.96
map + map	28.70
unordered_map + unordered_map	11.82

We can find that unordered_map is much faster than map and vector is much faster than map.

(2) Search

We choose two test cases. One is with some highly frequent words, and the other is not.

test case 1: beattie vows not to walk away from schoolies

unordered_map+vector

Data structure & Algorithm	time(s)
Priority queue size N	0.48
Priority queue size 10	0.45
Bucket sort	0.3

unordered_map+unordered_map

Data structure & Algorithm	time(s)
Priority queue size N	0.34
Priority queue size 10	0.34
Bucket sort	0.27

test case 2: australia is locked into war timetable opp

unordered_map+vector

Data structure & Algorithm	time(s)
Priority queue size N	0.04
Priority queue size 10	0.03
Bucket sort	0.02

unordered_map+unordered_map

Data structure & Algorithm	time(s)
Priority queue size N	0.03
Priority queue size 10	0.04
Bucket sort	0.02

6. Conclusion

In this paper, we successfully build the twitter keyword search system and research on system performance with different data structures and algorithms. Our findings are as follows:

(1) In building database and search the keyword, unordered_map is better than map

(2) In storing keyword's location and frequency, vector is better than map

(3) In sorting the top 10 results: bucket sort is better than priority queue

Reference

[1]:https://www.merriam-webster.com/dictionary/search%20engine

[2]:https://www.hackerearth.com/zh/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/

[3]https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/

[4]https://www.hackerearth.com/zh/practice/algorithms/sorting/bucket-sort/tutorial/

[5]https://www.geeksforgeeks.org/unordered_map-in-cpp-stl/