CS300 – Spring 2023-2024 - Sabancı University

Homework 3 – Search Engine V2

Due May 15 Wednesday at 22:00

Brief Description

In this homework, you will write a search engine and compare the speed of two different data structures for your search engine architecture. The search engines in real life search hundreds of millions web pages to see if they have the words that you have typed, and they can do this for thousands of users at a given time. In order to do these searches really fast, search engines such as Google do a lot of what we call preprocessing of the pages they search; that is, they transform the contents of a web page (which for the purposes of this homework, we will assume it consists of only strings) into a structure that can be searched very fast.

In this homework, you are going to preprocess the documents provided to you. For each unique word, you will insert a node into both of your data structures, i.e. Binary Search Tree (BST) and Hash Table. Of course, you will also keep track of the details such as the document name which the word appears in and the number of times the word appears in each document. So, you need to implement a templated BST and Hash Table.

Using both of the data structures, you need to show the number of times that every queried word appears in. After that, you will compare the speed of them. You may use and modify the data structures and classes you implemented in Homework2.

BST (Binary Search Tree)

You may use the BST class that you have implemented in Homework2.

HashTable

You are going to implement a hash table data structure adopting any of the collision handling strategies, such as separate chaining, linear or quadratic probing, etc. You may use any strategy which may accelerate the HashTable operations.

You need to rehash every time when the load of the table exceeds the load factor that you have determined for your HashTable. For each rehash operation you need to print the internal information of the HashTable such as the previous capacity, current capacity, previous load, and the current load. The load will be calculated as (item count / capacity).

Additionally, you need to develop a hash function aiming to have collision free hash values for different WordItem structs.

Program Flow

You need to get the number of the documents that you need to preprocess first. Then, after getting the names of all documents from the console, you need to preprocess all the text in the documents. Only the alphabetical characters are considered, and the unique words need to be case insensitive (for example; “izmir” and “Izmir” are the same). **The rest (such as digits or punctuation characters) are processed as the separator for the words.** For each unique word appearing in the text, you need to insert a new node into the tree and the hashtable.

If the node is already there, you need to update the node with the information about this document. For example; you have preprocessed the “a.txt” document first and there is only one “the” word in this document. You need to insert “the” word into the data structures. Then, while preprocessing the “b.txt” document, “the” word appears 4 times. So, you need to add this information ({“b.txt”, 4}) into the existing item in your data structures and the item of “the” word becomes {{“a.txt”, 1}, {“b.txt”, 4}}.

After you preprocess all the documents, you need to get a query from the console which consists of a line of strings (HINT: You may use `getline(cin, line)`). This line might consist of more than one word. Then, **you need to output the document names in which all words in the query appear including the number of times that each word appears in that document.**

For the comparison of the speed of using different data structures, you need to perform the same operation, processing the query and building the result, without printing the result 20 times and taking the average timing.

Hint

```
struct DocumentItem {
	string documentName;
	int count;
};

struct WordItem {
	string word;
	// List of DocumentItem’s.
};
```

For timing

```
int k = 20;
auto start = std::chrono::high_resolution_clock::now();
for (int i = 0; i < k; i++)
{
      // QueryDocuments(with BST);
}
auto BSTTime = std::chrono::duration_cast<std::chrono::nanoseconds>
                        (std::chrono::high_resolution_clock::now() -
                  start);
cout << "\nTime: " << BSTTime.count() / k << "\n";
start = std::chrono::high_resolution_clock::now();
for (int i = 0; i < k; i++)
{
      // QueryDocuments (with hashtable);
}
auto HTTime = std::chrono::duration_cast<std::chrono::nanoseconds>
                  (std::chrono::high_resolution_clock::now() - start);
cout << "\nTime: " << HTTime.count() / k << "\n";
```

Rules

- HashTable implementation **must be faster** than the BST implementation. Otherwise your grade will be 0.
- Every non-alphabetical character is considered as a separator (**different than HW2**). This rule applies for both the text in input files and queried words.

  For example; "face2face" = "face.face" = "face:facE" = "face face"

- The resulting HashTable load factor (λ) **must be larger than 0.25** and **lower than 0.9** (0.25 < λ < 0.9) whatever the input size is. Otherwise your grade will be 0. Pick the upper bound of load value that gives you best results in terms of speed.

Input

First, input is the number of documents that is going to be preprocessed. Then the file names will be received from the console. Then the queried words will be taken from the user as one line of words.

Output

While preprocessing the files, you need to print the information about the HashTable every time when the rehashing occurs, i.e. display the previous capacity, the previous load, current capacity, current load, and the number of items in the HashTable. After all preprocessing is finished, you need to print the final item count in the HashTable and the current load factor λ.

After the user enters the query words, you need to print the document names and the number of times each queried word appears in that document.

Lastly you compute the time it takes to do the query with each data structure you've implemented (BST, Hash Table) and display them with the speedup ratio of these timing values compared to each other. In addition you display the ranking of the times of each sort algorithm.

Sample Run 1

```
Enter number of input files: 1
Enter 1. file name: a.txt
rehashed...
previous table size:53, new table size: 107, current unique word count 49,
current load factor: 0.448598
rehashed...
previous table size:107, new table size: 223, current unique word count 98,
current load factor: 0.434978
rehashed...
previous table size:223, new table size: 449, current unique word count 202,
current load factor: 0.447661
rehashed...
previous table size:449, new table size: 907, current unique word count 406,
current load factor: 0.446527
rehashed...
previous table size:907, new table size: 1823, current unique word count 818,
current load factor: 0.448162
rehashed...
previous table size:1823, new table size: 3659, current unique word count
1642, current load factor: 0.448483
rehashed...
previous table size:3659, new table size: 7321, current unique word count
3295, current load factor: 0.449939
rehashed...
previous table size:7321, new table size: 14653, current unique word count
6590, current load factor: 0.449669

After preprocessing, the unique word count is 8475. Current load ratio is
0.57838
Enter queried words in one line: above
in Document a.txt, above found 12 times.
in Document a.txt, above found 12 times.

Time: 9896
Time: 1797
Speed Up: 5.50544
```

Sample Run 2

```
Enter number of input files: 2
Enter 1. file name: a.txt
Enter 2. file name: b.txt
rehashed...
previous table size:53, new table size: 107, current unique word count 49,
current load factor: 0.448598
rehashed...
previous table size:107, new table size: 223, current unique word count 98,
current load factor: 0.434978
rehashed...
previous table size:223, new table size: 449, current unique word count 202,
current load factor: 0.447661
rehashed...
previous table size:449, new table size: 907, current unique word count 406,
current load factor: 0.446527
rehashed...
previous table size:907, new table size: 1823, current unique word count 818,
current load factor: 0.448162
rehashed...
previous table size:1823, new table size: 3659, current unique word count
1642, current load factor: 0.448483
rehashed...
previous table size:3659, new table size: 7321, current unique word count
3295, current load factor: 0.449939
rehashed...
previous table size:7321, new table size: 14653, current unique word count
6590, current load factor: 0.449669
rehashed...
previous table size:14653, new table size: 29311, current unique word count
13189, current load factor: 0.449933


After preprocessing, the unique word count is 15965. Current load ratio is
0.544676
Enter queried words in one line: above:aberration
in Document a.txt, above found 12 times, aberration found 1 times.
in Document a.txt, above found 12 times, aberration found 1 times.

Time: 14933
Time: 3081
Speed Up: 4.84613
```

Sample Run 3

```
Enter number of input files: 3
Enter 1. file name: a.txt
Enter 2. file name: b.txt
Enter 3. file name: c.txt
rehashed...
previous table size:53, new table size: 107, current unique word count 49,
current load factor: 0.448598
rehashed...
previous table size:107, new table size: 223, current unique word count 98,
current load factor: 0.434978
rehashed...
previous table size:223, new table size: 449, current unique word count 202,
current load factor: 0.447661
rehashed...
previous table size:449, new table size: 907, current unique word count 406,
current load factor: 0.446527
rehashed...
previous table size:907, new table size: 1823, current unique word count 818,
current load factor: 0.448162
rehashed...
previous table size:1823, new table size: 3659, current unique word count
1642, current load factor: 0.448483
rehashed...
previous table size:3659, new table size: 7321, current unique word count
3295, current load factor: 0.449939
rehashed...
previous table size:7321, new table size: 14653, current unique word count
6590, current load factor: 0.449669
rehashed...
previous table size:14653, new table size: 29311, current unique word count
13189, current load factor: 0.449933


After preprocessing, the unique word count is 22320. Current load ratio is
0.761489
Enter queried words in one line: a the-has been
No document contains the given query
No document contains the given query

Time: 5669
Time: 5076
Speed Up: 1.11673
```

Sample Run 4

```
Enter number of input files: 2
Enter 1. file name: a.txt
Enter 2. file name: d.txt
rehashed...
previous table size:53, new table size: 107, current unique word count 49,
current load factor: 0.448598
rehashed...
previous table size:107, new table size: 223, current unique word count 98,
current load factor: 0.434978
rehashed...
previous table size:223, new table size: 449, current unique word count 202,
current load factor: 0.447661
rehashed...
previous table size:449, new table size: 907, current unique word count 406,
current load factor: 0.446527
rehashed...
previous table size:907, new table size: 1823, current unique word count 818,
current load factor: 0.448162
rehashed...
previous table size:1823, new table size: 3659, current unique word count
1642, current load factor: 0.448483
rehashed...
previous table size:3659, new table size: 7321, current unique word count
3295, current load factor: 0.449939
rehashed...
previous table size:7321, new table size: 14653, current unique word count
6590, current load factor: 0.449669


After preprocessing, the unique word count is 8575. Current load ratio is
0.585204
Enter queried words in one line: A:
in Document a.txt, a found 87 times.
in Document d.txt, a found 4 times.
in Document a.txt, a found 87 times.
in Document d.txt, a found 4 times.

Time: 2390
Time: 1975
Speed Up: 1.21002
```

General Rules and Guidelines about Homeworks

The following rules and guidelines will be applicable to all homeworks, unless otherwise noted.

How to get help?

You may ask questions to TAs (Teaching Assistants) of CS300. Office hours of TAs can be found at SUCourse. Recitations will partially be dedicated to clarify the issues related to homework, so it is to your benefit to attend recitations.

What and Where to Submit

Please see the detailed instructions below/in the next page. The submission steps will get natural/easy for later homeworks.

Grading and Objections

Your programs should follow the guidelines about input and output order; moreover, you should also use the exact same prompts as given in the Sample Runs. Otherwise the grading process will fail for your homework, and you may get a zero, or in the best scenario you will lose points.

Grading:

- ❑ Late penalty is 10% off the full grade and only one late day is allowed.
- ❑ **Having a correct program is necessary, but not sufficient to get the full grade. Comments, indentation, meaningful and understandable identifier names, informative introduction and prompts, and especially proper use of required functions, unnecessarily long programs (which is bad) and unnecessary code duplications will also affect your grade.**
- ❑ Please submit your own work only (even if it is not working). It is really easy to find "similar" programs!
- ❑ For detailed rules and course policy on plagiarism, please check out http://myweb.sabanciuniv.edu/gulsend/courses/cs201/plagiarism/

Plagiarism will not be tolerated!

Grade announcements: Grades will be posted in SUCourse, and you will get an Announcement at the same time. You will find the grading policy and test cases in that announcement.

Grade objections: Since we will grade your homeworks with a demo session, there will be very likely no further objection to your grade once determined during the demo.

What and where to submit (IMPORTANT)

Submission guidelines are below. Most parts of the grading process are automatic. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.
Add your name to the program: It is a good practice to write your name and last name somewhere in the beginning program (as a comment line of course).

Name your submission file:

- ❑ Use only English alphabet letters, digits, dot or underscore in the file names. Do not use blank, Turkish characters or any other special symbols or characters.
- ❑ Name your cpp file that contains your program as follows.

  "SUCourseUserName_yourLastname_yourName_HWnumber.cpp"
- ❑ Your SUCourse user name is actually your SUNet username which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the file name must be:

  cago_ozbugsizkodyazaroglu_caglayan_hw3.cpp
- ❑ Do not add any other character or phrase to the file name.
- ❑ Make sure that this file is the latest version of your homework program.
- ❑ You need to submit ALL .cpp and .h files including the data structure files in addition to your main.cpp in your VS solution.

Submission:

Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!
Gülşen Demiröz