CS300 – Summer 2019-2020 - Sabancı University

Homework #4 – Search Engine V3

Due 15 August Saturday at 23:55

Brief Description

In this homework, you will write a search engine as in the 2nd and 3rd Homework. Additionally, you will compare the speed of 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 read the input files, and store them into the vector. You will then sort this vector with several sorting algorithms:

1. Quick Sort (selection of pivot=first, pivot=random, and pivot=median),
2. Insertion Sort,
3. Merge Sort (in-place: without using extra memory storage),
4. Heap Sort.

You will need to keep four different copies of the vector, since we want to see which of the sorting algorithms is more efficient. Next, after getting a query from the user, you should search for words in vector with the binary search algorithm (that's why we sorted them).

Program Flow

You need to get the number of the documents that you need to process first. Then, after getting the names of all documents from the console, you need to store them in a vector. 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 vector.

If the node is already there, you need to update the node with the information about this document. For example; you have 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 processing 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 processed 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 which all words in the query appear including the number of times that each word appears in that document.

For the timing output, you need to print three different results;

1. Time for search using binary search tree, hash table and binary search on sorted vectors.
2. Time for all sorting algorithms.
3. Speedups for comparing sorting algorithms and comparing search techniques.

For the comparison of the speed of using different search techniques, you need to perform the same operation, processing the query and building the result, without printing the result 100 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 = 100;
clock_t begin_time = clock();
for (int i = 0; i < k; i++)
    // QueryDocuments with Binary Search Tree
```

```
double endBST = float(clock() - begin_time);

cout << "\nBinary Search Tree Time: " << endBST / k << "\n";

begin_time = clock();

for (int i = 0; i < k; i++)

    // QueryDocuments with Hash Table

double endHT = float(clock() - begin_time);

cout << "\nHash Table Time: " << endHT / k << "\n";

begin_time = clock();

for (int i = 0; i < k; i++)

    // QueryDocuments with Binary Search

double endBS = float(clock() - begin_time);

cout << "\nBinary Search Time: " << endBS / k << "\n";
```

Rules

- Every non-alphabetical character is considered as a separator. This rule applies for both the text in input files and queried words.

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

Input

First, input is the number of documents that is going to be processed. 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

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 measurements for each sorting algorithm and print them with the ratio of these timing values.

Sample Run 1

```
Enter number of input files: 1

Enter 1. file name: a.txt

After preprocessing, the unique word count is 8475. Current load ratio is
0.521571

Enter queried words in one line: above

in Document a.txt, above found 12 times.

in Document a.txt, above found 12 times.

in Document a.txt, above found 12 times.


Binary Search Tree Time: 199420

Hash Table Time: 49840

Binary Search Time: 3652


Quick Sort(median) Time: 7052375

Quick Sort(random) Time: 7067562

Quick Sort(first) Time:  7067666

Merge Sort Time: 89310980

Heap Sort Time: 34407965

Insertion Sort Time: 59291425


Speed Up BST/HST: 4.0012

Speed Up Merge Sort/Quick Sort(Median): 12.6125

Speed Up Heap Sort/Quick Sort(Median): 4.859

Speed Up Insertion Sort/Quick Sort(Median): 8.37314
```

```
Speed Up Binary Search / Binary Search Tree Time: 10.926

Speed Up Binary Search / Hash Table Time: 2.711
```

Sample Run 2

```
Enter number of input files: 2

Enter 1. file name: a.txt

Enter 2. file name: b.txt

After preprocessing, the unique word count is 15965. Current load ratio is
0.574219

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.

in Document a.txt, above found 12 times, aberration found 1 times.


Binary Search Tree Time: 249490

Hash Table Time: 49975

Binary Search Time: 17787


Quick Sort(random) Time: 13987222

Quick Sort(median) Time: 15261330

Quick Sort(first) Time: 17879211

Merge Sort Time: 188395480

Heap Sort Time: 72005655

Insertion Sort Time: 281349125

Speed Up BST/HST: 4.9923
```

```
Speed Up Merge Sort/Quick Sort(Median): 12.3446

Speed Up Heap Sort/Quick Sort(Median): 4.71818

Speed Up Insertion Sort/Quick Sort(Median): 18.4354


Speed Up Binary Search Tree Time / Binary Search: 14.02

Speed Up Binary Search / Hash Table Time: 2.8
```

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

After preprocessing, the unique word count is 22320. Current load ratio is
0.802791

Enter queried words in one line: a the-has been

No document contains the given query

No document contains the given query

No document contains the given query


Binary Search Tree Time: 440676

Hash Table Time: 149620

Binary Search Time: 24482


Quick Sort(median) Time: 22140735

Quick Sort(first) Time:  22511732
```

```
Quick Sort(random) Time: 25140735

Merge Sort Time: 267334760

Heap Sort Time: 102276335

Insertion Sort Time: 3027406830


Speed Up BST/HST: 2.94063

Speed Up Merge Sort/Quick Sort(Median): 12.0743

Speed Up Heap Sort/Quick Sort(Median): 4.61937

Speed Up Insertion Sort/Quick Sort(Median): 136.735


Speed Up Binary Search Tree Time / Binary Search: 18.02

Speed Up Hash Table Time / Binary Search Tree Time: 6.1
```

Sample Run 4

```
Enter number of input files: 2

Enter 1. file name: a.txt

Enter 2. file name: d.txt


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.

in Document a.txt, a found 87 times.
```

```
in Document d.txt, a found 4 times.


Binary Search Tree Time: 99720

Hash Table Time: 49820

Binary Search Time: 9772


Quick Sort(first) Time: 7010690

Quick Sort(random) Time: 7111210

Quick Sort(median) Time: 7230590

Merge Sort Time: 94050125

Heap Sort Time: 36254350

Insertion Sort Time: 60584885


Speed Up BST/HST: 2.00161

Speed Up Merge Sort/Quick Sort(Median): 13.0073

Speed Up Heap Sort/Quick Sort(Median): 5.01402

Speed Up Insertion Sort/Quick Sort(Median): 8.37897


Speed Up Binary Search Tree Time / Binary Search: 10

Speed Up Hash Table Time / Binary Search: 5.02
```

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 are at the syllabus. 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

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: It is your right to object to your grade if you think there is a problem, but before making an objection please try the steps below and if you still think there is a problem, contact the TA that graded your homework from the email address provided in the comment section of your announced homework grade or attend the specified objection hour in your grade announcement.

- Check the comment section in the homework tab to see the problem with your homework.
- Download the .zip file you submitted to SUCourse and try to compile it.
- Check the test cases in the announcement and try them with your code.
- Compare your results with the given results in the announcement.

What and where to submit (IMPORTANT)

Submissions 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 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_hw4.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 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.).
  1. Click on "Assignments" at CS300 SUCourse.
  2. Click Homework 4 in the assignments list.
  3. Click on "Add Attachments" button.
  4. Click on "Browse" button and select the zip file that you generated.
  5. Now, you have to see your zip file in the "Items to attach" list.
  6. Click on "Continue" button.
  7. Click on "Submit" button. We cannot see your homework if you do not perform this step even if you upload your file.

Resubmission:

- After submission, you will be able to take your homework back and resubmit. In order to resubmit, follow the following steps.
  1. Click on "Assignments" at CS300 SUCourse.
  2. Click Homework 4 in the assignments list.
  3. Click on "Re-submit" button.
  4. Click on "Add/remove Attachments" button
  5. Remove the existing cpp file by clicking on "remove" link. This step is very important. If you don't delete the old cpp file, we get both files and the old one may be graded.
  6. Click on "Browse" button and select the new zip file that you want to resubmit.
  7. Now, you have to see your new zip file in the "Items to attach" list.
  8. Click on "Continue" button.
  9. Click on "Submit" button. We cannot see your homework if you do not perform this step even if you upload your file.

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