Department of Computer Science

CS260
Data Structures

Fall 2021

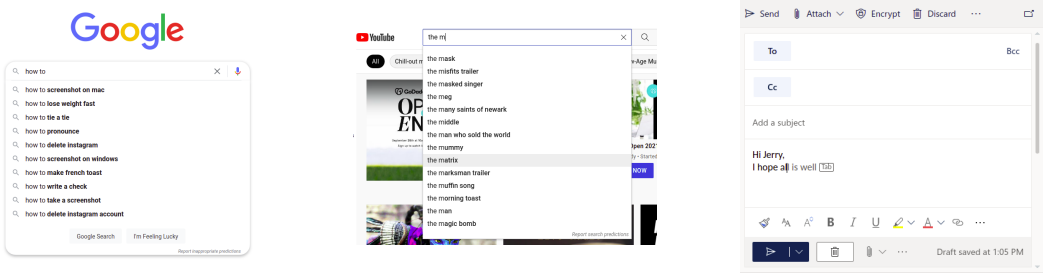
Programming Assignment 1

1 Regulations

Due Date : Friday, 10/8/2021 - 11:59pm
Late Submission: 10% off for each late day, with at most 2 day late submission
Submission : via Gradescope. See details below.
Team : The homework is to be done and turned in individually. No teaming up.
Cheating : All parties involved in cheating get zero from assignment and will be reported to the University.

2 Autocomplete

Autocomplete, or word completion, is a common feature that we interact with in our everyday life in which an application predicts the rest of a word a user is typing. It is in Google when you start typing a search query, in Youtube that would suggest you a video relevant to the initials of what you have started typing, or even in email apps where the rest of the sentence is being suggested as you keep typing a few letters.



Autocomplete is everywhere!

One key aspect of an autocomplete program would be to suggest the most relevant query, which can reasonably be defined as the most frequently encountered, as the user starts typing in initial parts of a query. In typing to the Google search bar, for example, it is very likely that the underlying algorithm has already seen a good amount of queries so far (billions of?) and knows how frequently each query is typed in. Then, as a user starts typing in, the algorithm searches for the most relevant queries from its knowledge base and lists the top most, say 10, as a suggestion.

The performance of autocomplete is crucial in several aspects. First, it needs to provide suggestions faster than a user would type it in, so that the suggestion should make sense. For sure, no one would like to wait for seconds for Google to suggest some autocomplete, while they themselves can type it much quicker! Second, we also need to consider that, this algorithm will be running afresh for each keystroke of every user! Thus, there will be a huge computational burden on the server side if the algorithm is not efficient. Third, suggestions provided by the algorithm should be relevant, which will require ordering the output based on frequency of the queries in the already existing knowledge base.

3 Problem

In this assignment, you will write a C program that will implement autocomplete. Your program will read a text file that contains query words in string along with a weight represented with a floating point number, where larger weights indicate higher likelihood of observing each query.

You should take your input query (which will be a partially written word) from command line and print out all query words from your knowledge base sorted in the order of relevence. For this assignment, you will need to consider three problems:

- When initially loading the knowledge base file, you should sort and store it internally in the memory in a way that it will allow efficient search later on.
- You will need a search algorithm that will efficiently search for the given query. Keep in mind that, your query will most probably be incomplete words. Thus, you will need to search for terms in the knowledge base for entries whose initial characters match the query word. (it can be an exact match as well)
- If your search leads to more than one entry as output, you should sort them according to their weights, printing suggestions from most frequently encountered to the least frequent.

3.1 Specifications

- Your code should be running on tux.
- **Materials to submit:** Your submission package will have your **source code**, a **makefile**, and a **report file written in pdf** that explains your approach for solving the problem. Put all these material in a folder named with your userid. (example: abc123)
- You should zip and submit this folder through Gradescope.
- Your **makefile** needs to produce an executable named **"auto"**, once we type "./make" in command line.
- Your **executable** "auto" is going to take **two command line arguments**, first being the name of the input knowledge base file and the second being the input query that you will autocomplete. Thus, an example call to your program will be as follows: **./auto movieScripts.txt complet**
- **Input knowledge base file** that shows the frequency of English words used in movies is provided as an example for you along this assignment description. We will test your program with other input knowledge base files as well, which will have the same structure as below:

```
/pa1] more movieScripts.txt
you 28787591
i 27086011
the 22761659
to 17099834
a 14484562
's 14291013
it 13631703
and 10572938
that 10203742
't 9628970
of 8915110
is 7400675
in 7337058
what 6900164
we 6755687
```

```
/pa1] more simpsons.txt
the     107946
you     98068
i       91502
a       79241
to      70362
and     47916
of      42175
it      36497
in      32503
my      32254
that    32083
is      31533
this    29902
me      28208
your    26781
```

  - each line represents a single query in the knowledge base
  - first column contains the query in string format. (Don't worry about the punctuation marks in the queries. Search for whatever is in those strings, which might contain apostrophe etc.)
  - second column contains the weight of the query word. You can read it as a floating point number (rather than an integer), just to be on the safe side.
- Your program is going to search for the partial query provided by the user from inside the knowledge base.
  - if the query is found, your output will be the list of terms that start with the query item, in **sorted order**.

  - If the item is not found in the knowledge base, your output will be -1.

- **Output** will be printed to standard output and needs to strictly satisfy the following structure for a test input "complet":
  pa1/> ./auto movieScripts.txt complet
  completely,72777
  complete,40210
  completed,8357
  completion,1314
  completing,1058
  completes,750

  - If query is found, each line should be in <query>,<weight> format, without any space between query and weight, but simply a comma.
  - If there are multiple outputs, put a new line at the end of each line.
  - If query is not found, only print -1.

- **Report file** should explain the algorithm you used for sorting and searching, how you stored the knowledge base in memory (i.e., data structure), and contemplate about its running time, your observation about its behavior for different inputs, what is good/bad about your algorithm and also your implementation, how much it might be improved if you used different methods, etc. These are just sample questions that you might (and at least you should) address in your report. You are more than welcome to write a more comprehensive report.

3.2 Grading

- Assignment is going to be graded out of 100 points.
- Submitting a code that compiles and runs gets 10 points.
- Next 60 points will be for the test cases (there will be multiple of them), which you will either pass or fail individually. Since we will do black box testing, make sure you adhere to input/output specifications.
- Final 30 points is for your report file. The purpose of the report file is to make sure that you fully understood what you have been doing while attempting to solve the problem. Try your best to convince the grader!
- **Regrading:** If you end up with a grade that you did not expect, possibly due to compilation or input/output errors, you can ask for a regrading for the first 70 point portion of the assignment. Keep in mind that there will be a 20 point penalty. And your corrections should not be more than 5 lines of code.