Department of Computer Science

CS260
Data Structures

Winter 2022

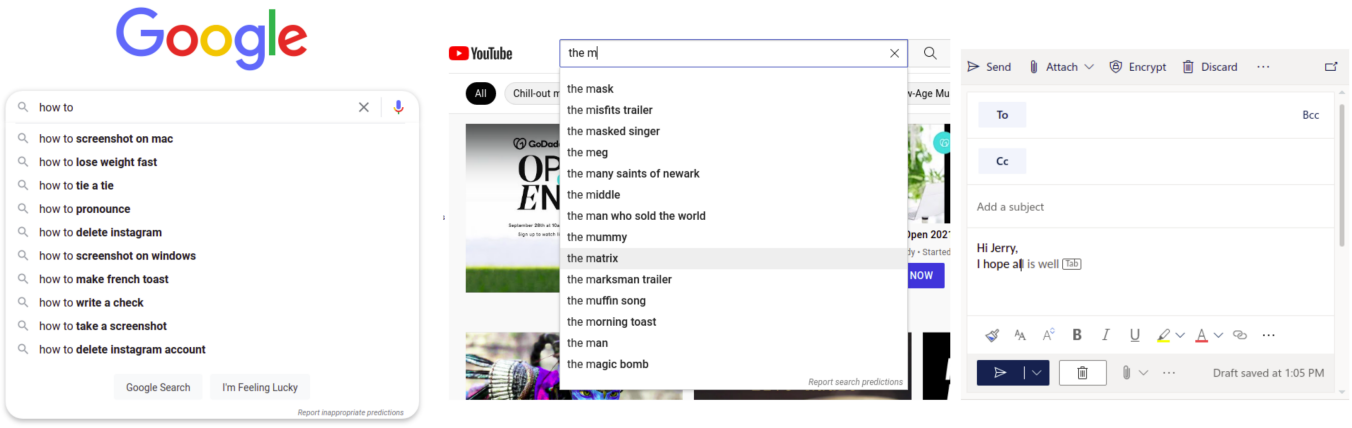
Programming Assignment 2

1 Regulations

Due Date : 1/21/2022, Friday, 11:59pm
Late Submission: 10% off for each late day, with at most 2 day late submission
Submission : via Gradescope.
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 every- day 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 by an integer value, where larger weights indicate higher likelihood of observing each query.
You should take your input query (which will be single partially written word) from command line and print out **all** query words from your knowledge base sorted in the order of relevance. For this assignment, you will need to consider three problems:

1. 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.
2. 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)
3. 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.

4 Starter Package

- Along with this assignment instructions document, you will be provided with:
  - a **starter code** that handles reading the input file and taking command line arguments
  - a simple **makefile**, that compiles the code to generate an executable
  - an **executable autocomplete program**, which you can compare against your program to match input/output specifications.
  - a starter **report file in LaTeX**, which you will fill with the analysis of your assignment.

5 Input/Output Specifications

- **Your executable program** (which should be named as "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:

  $pa2/> ./auto movieScripts.txt complet

- **Input knowledge base file** that contains 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:

```
CCI-FGG0ZQQ6LX:pa2 yo42$ 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
me 6444985
this 5739788
he 5516364
for 5174060
```

```
CCI-FGG0ZQQ6LX:pa2 yo42$ 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
for 25634
oh 25053
```

First few lines of sample input files!

  - 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 an integer. The two columns are separated from each other by a single whitespace.

- 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 program should print the list of terms that start with the query item, in sorted order. If there are many output terms, you should print **top 10** terms with highest weight in your output.
  - If the item is not found in the knowledge base, your output should be "No suggestion!".

- **Output** will be printed to standard output and needs to strictly satisfy the following structure: (following is an example over the test input "complet")

  $pa2/> ./auto movieScripts.txt complet
  completely,72777
  complete,40210
  completed,8357
  completion,1314
  completing,1058
  completes,750

  That is to say:
  - 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 "No suggestion!".

- **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. You will be provided with a starter LaTeXfile for your report. You should answer the questions in there, and can add more information if you like.

5.1 Submission

- Your code must be running on tux. So, make sure that it compiles and runs on tux before you submit, even if you develop it elsewhere.
- Your submission package **must include** 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), compress it into a zip file named as your user id (example: abc123.zip). Submit this zip file through Gradescope.
- Your makefile needs to produce an executable named "auto", once we type "make" in command line.

5.2 Grading

- Assignment is going to be graded out of 100 points.
- You will provide your self assessment for your effort for the 40 point portion of the assignment.
- 40 point portion of the remaining part will be for the test cases of the assignment.
- The final 20 point will be for the report file.