CSCE 2014 – Programming Project 6

Midpoint Due Date – 11/14/2022 at 11:59pm
Final Due Date – 11/21/2022 at 11:59pm

1. Problem Statement:

The purpose of this project is to give students experience with hash tables. In particular, you will be creating a hash table to store the 10,000 most common English words in order of frequency, as determined by n-gram frequency analysis of the Google's Trillion Word Corpus. Your task is to optimize this hash table so words can be looked up as quickly as possible (i.e. the fewest collisions).

In this project, you will be given a fully implemented HashTable class that stores (string, integer) pairs using the "linear probing" technique for handling collisions. In this case the string is the "key" (the words you want to look up) and the integer is the "value" (the word's position in the frequency sorted input file). You will also be given a text file "google10000.txt" that contains the most common 10,000 English words sorted by frequency with the most common word "the" at the top of the file.

Task 1: Read and process input file

Your first task will be to create a main program to open and read "google10000.txt" file and store (string, integer) pairs in the HashTable. Since there are 10,000 words, your hash table should be 2-4 times larger than 10,000. The integer value should correspond to the position of the word in the file. For example, "the" is the first word, so you should store ("the", 1) in the hash table.

Task 2: Count the number of collisions

The current HashTable class handles collisions, but it does not keep track of how many collisions occur. Your second task is to add a collision counter to the class and a method to "get" this value in the main program. Once you have done this, modify your main program to print out the number of collisions that took place when inserting 10,000 values into the HashTable.

Task 3: Improve the Hash function

The current HashTable class has a private "Hash" method that returns the hash table index given an input string. This hash formula has not been optimized in any way. Your next task is to modify this "hash" method to find three hash formulas that

outperform the current hash formula. This will require some trial and error looking at the number of collisions that occur with each formula.

Task 4: Improve the Hash2 function

The current HashTable class has a private "Hash2" method that implements the "linear probing" phase by going to the next table location. Your final task is to adapt this function to use "secondary hashing" instead. You are welcome to use any formula you like in place of the simple "linear probing" formula. You may also need to change the size of the HashTable to ensure that the hashing formula wraps around properly.

2. Design:

Your main design tasks for this project are in tasks 3 and 4 above where you must find hash formulas to minimize the number of collisions. You should keep track of your experiments as you go so you can document the formulas you tried and how they performed in your project report. You should leave your best formulas in your final version of the HashTable class.

3. Implementation:

To implement your project, you should break your code down into **three** files: hash_table2.h, hash_table2.cpp, and main.cpp. As always, it is a good idea to add code incrementally writing comments, adding code, compiling, debugging, a little bit at a time.

Remember to use good programming style when creating your program (good names for variables and constants, proper indenting for loops and conditionals, clear comments). Be sure to save backup copies of your program somewhere safe. Otherwise, you may end up retyping your whole program if something goes wrong.

4. Testing:

For this project students are required to compile and test their program on Linux. Once you have logged into turing.csce.uark.edu, copy your source code into a folder called "project6". Go into this folder and type in the command "script". This will create a "typescript" file that records all user input and program output until you type "exit".

Compile your program using "g++ -Wall *.cpp -o main.exe". Then run your program by typing "./main.exe" to verify that your program operates correctly for all of the requirements listed above.

When you are finished testing your program, type "exit". Copy the "typescript file back to your personal computer to be included with your code and project report when you submit your completed project into Blackboard.

5. Documentation:

When you have completed your C++ program, write a short report using the project report template describing what the objectives were, what you did, and the status of the program. Does it work properly for all test cases? Are there any known problems? Save this report to be submitted electronically.

6. Midpoint Submission:

For the midpoint submission, you should upload your code after you are approximately 1/3 to 1/2 way through this implementation process. It is OK to have skeleton methods that simply print debugging messages at this stage. The main thing we want to see is code that compiles and is part of your final solution.

7. Project Submission:

In this class, we will be using electronic project submission to make sure that all students hand their programming projects and labs on time, and to perform automatic plagiarism analysis of all programs that are submitted.

When you have completed the tasks above, copy all of your source code, your typescript file, and your project report into a folder called "project6". Compress this directory into a single ZIP file called "project6.zip", and upload this ZIP file into Blackboard. The GTAs will download and unzip your ZIP file and compile your code using "g++ -Wall *.cpp" and run your program to verify correctness.

The dates on your electronic submission will be used to verify that you met the due date above. All late projects will receive reduced credit:

- 10% off if less than 1 day late,
- 20% off if less than 2 days late,
- 30% off if less than 3 days late,
- no credit if more than 3 days late.

You will receive partial credit for all programs that compile even if they do not meet all program requirements, so handing projects in on time is highly recommended.

8. Academic Honesty Statement:

Students are expected to submit their own work on all programming projects, unless group projects have been explicitly assigned. Students are **not** allowed to distribute code to each other, or copy code from another individual or website.

Students **are** allowed to use any materials on the class website, or in the textbook, or ask the instructor and/or GTAs for assistance. This course will be using highly effective program comparison software to calculate the similarity of all programs to each other, and to homework assignments from previous semesters. Please do not be tempted to plagiarize from another student.

Violations of the policies above will be reported to the Provost's office and may result in a zero on the programming project, an XF in the class, or suspension from the university, depending on the severity of the violation and any history of prior violations.