CS/IT 200

Lab 7: Hash Tables

Recommended Due Date: Thursday, October 29, 11:59pm

Submission

- Submit all code and other materials in a single zip file to the Lab 7 assignment folder on Kodiak.

Goal: Analyze hash table efficiency by varying collision-handling schemes.

Part I

You have been provided with the code from the textbook for `HashMapBase`, `ChainHashMap`, and `ProbeHashMap`. Implement two additional subclasses of `HashMapBase`, based on two alternative collision handling techniques described on page 419 of your textbook:

- `QuadHashMap` – Handles collisions using quadratic probing
- `DoubleHashMap` – Handles collisions using double hashing

Part II

You will now perform experiments on your four hash table implementations. You will do this by disabling size increases and rehashing and testing the "speed" of insertions to the hash table by measuring the total number of probes required to fill in every 5% of the table.

There are three tasks to this part:

(a) Modify the code in the `HashMap` classes to disable size increases and rehashing, as well as keep track of total probes.

Since we want to see the effect of letting a table get increasingly full, we need to disable the code that causes size increases and re-hashing.

Each insertion has a certain number of probes (attempts to use an index). If there is no collision, then there is only 1 probe. If there are collisions, there will be additional probes. The total number of probes is the sum of all probes for all insertions. Modify the code in the `HashMap` classes to keep track of the total number of probes carried out.

(b) Write a function outside of the classes that runs the analysis described below.

For your simulation, you will pick a capacity for your hash tables. This capacity must be a prime number. (As an example, I'll use 4099). Use the same capacity for all four hash tables. You will add words from `words.txt` (a file containing unique words) until your table is full. Note that the file contains a large number (10K+) of words, and you may not need to read them all. If your capacity is 4099, you will only need to read and insert 4099 words. Since insertion requires a key and a value, use the word as the key and the value.

Acknowledgement: This assignment is adapted from one created by Tommy Färnqvist, Linköpings Universitet, Sweden.

As you insert data into your tables, record the total number of probes needed until the table is 5% filled, 10% filled, etc., until you reach 100% of the input data. Note that the table will probably never be exactly 5% full (or any exact percentage except 100%), so you should record the number of probes once the table is at least 5% full, 10% full, etc.

Note for Mac Users: Open the words.txt file using the command `open('words.txt', 'r', encoding='latin-1')`

Note about quadratic hashing: Due to clustering, it is possible that even if the table is not full, the hash function may not be able to find a place to insert. If this happens, your program will be in an infinite loop. Terminate your program.

Note about double hashing: Make sure the value of `q` selected for your secondary hash function is smaller than the size of your table.

(c) Submit a graph and explanation of the analysis described above.

Submit a graph comparing the four hashing methods. On the x-axis, put the loading factor (5%, 10%, …, 100%). On the y-axis, put the total number of probes needed to insert the data until that moment. Explain what your data and graph show about the number of probes needed and the effectiveness of each collision strategy. Are some strategies more effective at different load factors? If you had to terminate your program for quadratic hashing, fill in as much data as you have.

What to Submit:

- All code (modified code, new classes, and code that runs your analysis)
- Your graph and explanation of the graph in a Word file or PDF.

Rubric

Grade	Overall	Part I	Part II
4	A 4 in one part and at least a 3 in the other part.	Both classes use inheritance correctly, implementing only the necessary functions. Calculation of slots are correct for both approaches.	Correctly disables resizing and rehashing code. Correctly counts probes in all 4 methods. Experiment code is correct, including correct syntax for adding to tables. Graph and analysis reflect understanding of material.
3	At least a 3 in one part, and at least a 2 in the other part.	Not correctly using inheritance; or minor errors in calculating slots for one or both approaches; or a major	No more than 2 of the following: Resize/rehash code not correctly removed

		error calculating slots in one approach.	Flaw in probe counting in 1 table class. Experiment code uses improper syntax. Written analysis is flawed or incomplete.
2	At least a 2 in both parts; or at least a 3 in one part, and at least a 1 in the other part.	Two of the issues above. No errors cause code to break.	No more than 3 of the above items, or a bug in the experiment code
1	At least a 1 in both parts.	One class works largely as expected, with few or no issues mentioned above. Using the other class generates errors.	Many of the issues above, or no graph/analysis.
0		Neither class is usable.	Experiment code cannot run.