Data Structures (2028C) -- Spring 2019 – Homework 4

Topics covered: Hashing and Unit Testing

Homework due: ***Monday, April 16 at 6:00PM***

Objective:

The objective of this homework is to compare the performance of two different implementations of a Hash Table.

Scenario:

We learned about creating Hash tables that use probing to solve the problem of collisions, use 2D arrays, and use linked lists. This assignment is intended to determine which approach produces better performance. This needs to be written using C++.

Requirements:

1. Create a hash table that uses a 1D array and linear probing to handle collisions. This table should have 500 available slots to hold items. The hashing function can be as simple as taking the modulus of the input value for the number of available slots. The hash table class should have the following members:
    a. Constructor
    b. Destructor
    c. Insert – accepts a value (integer), runs a hash function on the integer, and places it in the hash table. This function should return the number of spots it checks before inserting the item in the hash table so if it has no collisions, it should return 1 and if it has 3 collisions and then finds a spot on the 4^{th} try, it return 4.
    d. Find – accepts a value (integer), locates the value in the hash table and returns the number of spots it checked to find the item or determine it is not in the hash table. This uses the same procedure as Insert to determine number of spots checked.
    e. Remove – accepts a value (integer), locates the value in the hash table and removes it from the hash table. It returns the number of spots it checked to find the item or determine it is not in the hash table. This uses the same procedure as Insert to determine number of spots checked.
    f. Print – prints all items in the hash table including an indication of which spots are not occupied.
2. Derive a 2D hash table of 100 slots with a depth of 5 for each slot. Modify all the methods of the 1D hash table accordingly. In the event that the depth for a slot is full, it should increment to the next slot and proceed until it finds an open spot.
3. Create a main function that will be used to test your 2 hash tables. This should generate a list of 100 random, unique integers that will be used to test both hash tables (both tables should get the same set of integers in the same order. Then perform the following tasks:
    a. Insert the first 50 values in both tables, keeping a running sum of spots checked to insert those items. This value after inserting 50 should be recorded.
    b. Remove all the items from the hash tables where the index of the value in the main function % 7 == 0. Record the running sum of spots checked to remove those items.
    c. Insert the remaining 50 values from both tables. Record the running sum of

spots checked to remove those items.

   d. Attempt to find all the items in the hash table where the index value in the main function % 9 == 0 and the item wasn't removed previously.

4. Write unit tests for your two classes. You can learn more about unit tests for Visual Studio at https://docs.microsoft.com/en-us/visualstudio/test/writing-unit-tests-for-c-cpp?view=vs-2019.

Submission:

Submit all source code files and any required data files in a zip file. Include a write up as a PDF including:

- The name of all group members (minimum 2 members, maximum 4 members).
- Instructions for compiling and running the program including any files or folders that must exist.
- What each group member contributed. If the contributions are not equitable, what portion of the grade each group member should receive.
- An analysis of the results from counting attempts to insert, remove, and find items in the hash table for both classes. You may run additional scenarios beyond those described in #3 above.

Submission should be submitted via BlackBoard.

Grading:

1. 25% - 1D hash class functions Insert, Find, Remove, and Print work correctly.
2. 25% - 2D hash class functions Insert, Find, Remove, and Print work correctly.
3. longest path to a leaf and the shortest path to a leaf no greater than 2.
4. 15% - Main function correctly performs required tasks.
5. 10% - Your analysis of the results correctly interprets the strengths and weaknesses of each approach and describes areas for further research.
6. 15% - Unit tests correctly test in an automated fashion common scenarios for your classes.
7. 10% - Code is well formatted, well commented and follows a reasonable style.

If program fails to compile, the grade will be limited to a max grade of 50%.