Data Structures (2028C) -- Fall 2024 – Homework 3

Topics covered: Hashing and Unit Testing

Homework due: ***Friday, Nov 22 at 11:59PM***

Objective:

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

Scenario:

In this assignment, we explored creating hash tables using different collision resolution techniques: linear probing and 2D array. The goal is to evaluate which approach offers the best performance. The implementation should be written in C++.

Requirements:

1. Create a **hash table using a 1D array with linear probing** for collision handling. The table should have 500 slots available to store items. **The hashing function can be as simple as taking the modulus of the input value for the number of available slots, i.e.,** $\boldsymbol{h(x) = x \,\%\, 500}$. The hash table class should include the following members:
    a. **Constructor**
    b. **Destructor**
    c. **Insert**
        - It accepts an integer value, applies the hash function, and inserts the value into the hash table.
        - This function should return the number of slots checked before inserting the item. For example, if there are no collisions, it returns 1; if there are 3 collisions and a spot is found on the 4th try, it returns 4.
        - If all slots are full, print out that 'The value x cannot be inserted.' and return the number of spots it checked.
    d. **Search**
        - It accepts an integer value, locates the value in the hash table.
        - If the value is in the hash table, print out that 'The value x can be found.' and return the number of spots it checked.
        - If the value is not in the hash table, print out that 'The value x cannot be found.' and return the number of spots it checked.
        - This uses the same procedure as Insert to determine the number of spots checked.
    e. **Remove**
        - It accepts an integer value, locates it in the hash table, removes it, and repositions any other values in the hash table that have the same hash value.
        - If the value is in the hash table, print out that 'The value x was removed.' and return the number of spots it checked.
        - If the value is not in the hash table, print out that 'The value x cannot be removed.' and return the number of spots it checked.

f. **Print**
   - Outputs all items in the hash table, indicating which slots are unoccupied. The output format is up to you.

2. Design a hash table using a 2D array structure with 100 slots, each capable of holding 5 items (i.e., int arr[100][5]). **The hashing function is $h(x) = x \% 100$.** This hash table should have the same functions as described in requirement 1.

3. Create a main function that will be used to test your 2 data structures.
   a. Start by generating a list of 100 random, unique integers between 1 and 1000. Store these integers in an array called dataset. This dataset will be used to test both data structures you created.
   b. Then perform the following tasks:
      - **Initial Insertion**: Insert the first 50 values from the dataset into both hash tables, tracking the cumulative number of spots checked during each insertion for each structure. Record the total number of spots checked for both data structures after all 50 values have been inserted and output the results. For example, by the end of this stage, you might record that 1D hash table checked 1000 spots during the initial insertion, and 2D hash table checked 500 spots during the initial insertion. (The numbers 1,000 and 500 are provided as examples only and do not represent actual results.)
      - **Selective Removal**: Check the first 50 values from the dataset, and if a value satisfies the condition $x \% 7 == 0$, remove that value from both hash tables. For example, if the first 50 values of dataset include {0, 1, 7, 17, 49, 109, 217, …}, we remove {0, 7,49, 217, …} from other hash tables. Record the running sum of spots checked to remove those items and output the results.
      - **Second Insertion**: Insert the remaining 50 values from the dataset into both hash tables. Record the running total of spots checked during this insertion phase and output the results.
      - **Search Operation**: Check all values from the dataset, and if a value satisfies the condition $x \% 9 == 0$, find that value from both hash tables. For example, if the dataset includes {0, 1, 7, 17, 49, 109, 217, …}, we find {0, 49, …} from other hash tables. Record the running sum of spots checked and output the results.

4. Analyze the results (number of spots checked for both structures) and explain your findings. Did it meet your expectations? Which performed best? What situations make each ideal and least ideal?

5. 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.

Submission should be submitted via Canvas.

Grading:

1. 30% - 1D hash class functions Insert, Find, Remove, and Print work correctly.
2. 15% - 2D hash class functions Insert, Find, Remove, and Print work correctly.
3. 15% - Main function correctly performs required tasks.
4. 15% - Your analysis of the results correctly interprets the strengths and weaknesses of each approach and describes areas for further research.
5. 15% - Unit tests correctly test in an automated fashion common scenarios for your classes.
6. 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%.