CSDS 233 Assignment #4

Due November 10, 2024, before 11:59 pm. 100 points

Submission Instructions

- The submissions will be evaluated on completeness, correctness, and clarity.
- Please provide sufficient comments in your source code to help the TAs read it.
- Please generate a single zip file containing all your *.java files needed for this assignment (not .class) and optionally a README.txt file with an explanation about added classes and extra changes you may have done.
- Name your file P1_YourCaseID_YourLastName.zip for your **coding exercises**. Submit your zip file electronically to Canvas.
- Please submit a **PDF** for your answers to the written exercise **separate from the zip file**.

Office Hours for This Assignment

- Written Assignment:
- Programming Assignment:

Contact Information

If you have questions outside of the scheduled office hours:

- For written questions, email: Derrick qqn@case.edu
- For coding problems, email: Michael Boelens mvb59@case.edu

Written Exercise [50 points]

Problem 1: Collision Resolution Using Linear Probing and Double Hashing (20 points)

The keys 14, 17, 18, 3, 8, 1, 18, 11, 13, and 20 are inserted into an initially empty hash table of size 10 using the hash function h(x) = x mod 10.

a) (10 points) Show the resulting hash table using linear probing to resolve collisions. Include intermediate steps.

b) (10 points) Now, double the hash table size to 20 and reinsert the same keys using double hashing. Use the following hash functions:

- First hash function: h1(x) = x mod 20
- Second hash function: h2(x) = 7 - (x mod 7)

Show the final hash table after double hashing.

Problem 2: Rehashing Example (15 points)

You have a hash table that exceeds the load factor threshold = 0.7. After inserting the following keys [12, 18, 25, 42, 38, 17], the table size is doubled from 10 to 20.

a) Show the initial hash table after inserting the keys using the hash function h(x) = x mod 10.

b) Rehash the table after doubling the size, and show the final hash table after rehashing with h(x) = x mod 20.

Problem 3: Comparison of Collision Resolution Techniques (15 points)
Explain and compare the following collision resolution techniques:

- Linear Probing
- Quadratic Probing
- Separate Chaining

In your answer, discuss the pros and cons of each method, focusing on their performance under high load factors and how they handle clustering.

Programming Exercise [50 points]

In this assignment, you will implement a hash table that resolves collisions using both linear probing and double hashing. You are required to manually implement these collision resolution techniques without using built-in hash table libraries. The goal is to understand how open addressing works in hash tables.

You will implement the following methods in a class named **HW4_abc123** (where abc123 is your CaseID):

1. **Constructor** [5 pts]
    - **public HW4_abc123(int size):**
        - Initializes a hash table with the specified size.
        - The table should be empty upon initialization.
2. **Linear Probing Insertion** [15 pts]
    - **void linearProbingInsert(int key)**:
        - Inserts the given key into the hash table using linear probing for collision resolution.
        - If a collision occurs, the function should probe linearly (i.e., move to the next available slot) until an empty position is found.
        - function should print the state of the table after each insertion
3. **Double Hashing Insertion** [20 pts]
    - **void doubleHashingInsert(int key, int prime)**:
        - Inserts the given key into the hash table using double hashing for collision resolution.
        - The first hash function is key % size, and the second hash function is prime - (key % prime).
        - If a collision occurs, the function should probe using the result of the second hash function to resolve it.
        - The function should print the state of the table after each insertion
4. **Table Printing** [5 pts]
    - **void printTable()**:
        - Prints the current state of the hash table. The table should be printed as a comma-separated list of elements, where null represents an empty slot.
5. **Testing** [5 pts]

- Linear Probing: Test the linearProbingInsert method using the array:
    - [10, 3, 17, 14, 18, 3, 8, 1, 18, 11]
- Double Hashing: Test the doubleHashingInsert method using the array:
    - [7, 17, 27, 37, 47, 57, 67, 77, 87, 97] with prime = 7