ECE250 - Project 2
Hashing
Design Document

Leah Burgess, UW UserID:l2burges
November 8th, 2022

Overview of Classes

Class: Student
Description: Represents each individual student as an object in the dynamically allocated array. Each student is defined by their student number and last name.
Member Variables (all private): string name (their last name), unsigned int x (their student number)
Member Functions (public):

- Setters for the string name, and unsigned int x
- Getters for the string name, and unsigned int x

Class: doubleHash
Description: Represents the students as a dynamically allocated array and contains the methods for managing the hash table (insert student, search student, delete student).
Member Variables (private):

- Int size: the max size of the array
- Int currentSize: the current size of the array (the number of students in the hashtable)
- Student *student: dynamic array as a pointer filled with the student objects that acts as a hash table

Member Functions (public):

- String insert(unsigned int x, string name) - If the number of students in the hash table are less than the max size, check if the space from the first hash function is empty. If it is empty, insert and return success. Otherwise, check if the student already filling the spot is the same student number. If it is, return failure. Otherwise, calculate (h1 + i*h2)modm and repeat the process from that spot in the hash table. If the table is full, return failure.
- String search(unsigned int x) - While the number of iterations is less than the size of the table, if the student number from the first hash function is equal to x, return “found last name in position”. Otherwise, calculate (h1 + i*h2)modm and repeat the process from that spot in the hash table. If you haven’t found the student number and the while loop ends, return failure.
- String deleteHash(unsigned int x) - While the number of iterations is less than the size of the table, if the student number from the first hash function is equal to x, set the name of the student to an empty string and the student number to 0 and return success. Otherwise, calculate (h1 + i*h2)modm and repeat the process from that spot in the hash table. If you haven’t found the student number and the while loop ends, return failure.

Class: chainingHash
Description: Represents the students as a dynamically allocated array of vectors and contains the methods for managing the hash table (insert student, search student, delete student, print chain).
Member Variables (private):

- Int size: the max size of the array
- vector<Student> *student: dynamic array as a pointer filled with vectors that hold the student objects that acts as a hash table

Member Functions (public):

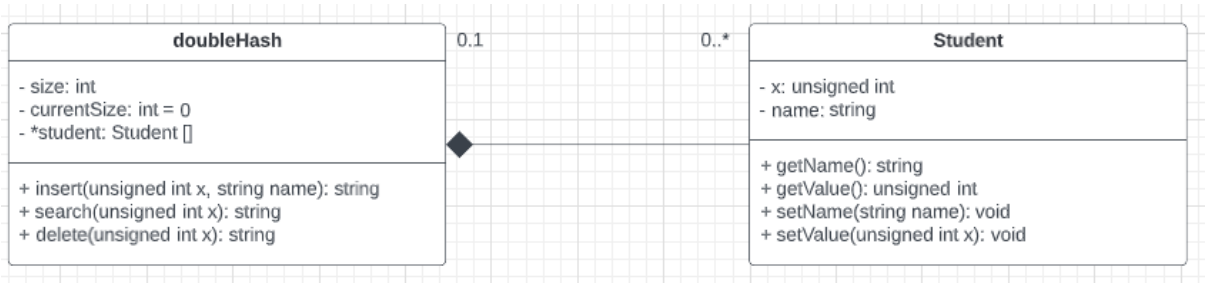
- String insert(unsigned int x, string name) - Check if the vector’s size in the position of the array from the first hash function is 0. If it is 0, insert and return success. Otherwise, check if

the student already filling the spot is the same student number. If it is, failure. Otherwise, loop through the vector and check for when x is greater than the next index. Insert at this position and return success.

- String search(unsigned int x) - Loop through the vector in the position of the array from the first hash function. If x is equal to the student number, return found last name in position. If you reach the end of the vector and the number is not found, return not found.
- String deleteHash(unsigned int x) - Loop through the vector in the position of the array from the first hash function. If x is equal to the student number, erase the student and return success. If you reach the end of the vector and the number is not found, return failure.
- Void print(int x) - If the vector's size at position x in the array is 0, print chain is empty. Otherwise, loop through the vector at position x in the array, printing each student number at each index.

UML Class Diagrams

Class: doubleHash



Class: chainingHash

Design Decisions

Student Class: I have a default constructor, and a constructor that takes a name and student number and sets name and value respectively. The destructor is empty.

doubleHash Class: I have a constructor that takes in an integer x. It then creates a new dynamically allocated array of student objects and sets max size of the array to x. The destructor deletes the array of student objects.

chainingHash Class: I have a constructor that takes in an integer x. It then creates a new dynamically allocated array of vectors that hold student objects and sets max size of the array to x. The destructor deletes the array of vectors holding the student objects.

***** I did not override any operators and I did not have to pass each parameter by reference or use of the keyword "const".

Test Cases

*For each test file, I tested instances where the hash table was empty, partially filled, and completely filled.

Double Hashing:

- Tested to create a hash table of size x
- Tested inserting multiple elements, including inserting when the hash table is full (expect failure), inserting when there is a collision using first hash function and collisions using secondary hash function, and inserting the same student number that has already been inserted (expect failure).
- Tested searching for students that were in the hashtable (expect success) and those that weren't (expect failure). As well, tested searching for a student that was inserted and has since been deleted (expect failure).
- Tested deleting students that were in the hashtable (expect success) and those that weren't (expect failure). As well, I tested deleting the same student twice (expect success on the first attempt and failure on the second).

Chaining:

- Tested to create a hash table of size x.
- Tested inserting multiple elements, including inserting when there is a collision (expect chaining) and inserting the same student number that has already been inserted (expect failure).
- Tested searching for students that were in the hashtable (expect success) and those that weren't (expect failure). As well, tested searching for a student that was inserted and has since been deleted (expect failure).
- Tested deleting students that were in the hashtable (expect success) and those that weren't (expect failure). As well, I tested deleting the same student twice (expect success on the first attempt and failure on the second).
- Tested print, including indexes that have no students, and ones that have multiple. Tested to make sure they are printing in descending order. Tested if you delete a student that it prints out the chain without that student.

Performance Consideration

Double Hashing: For insert, delete, and search, in the best case no collisions occur, therefore, the time complexity will be O(1). Worst case, we have n iterations of the hash function, therefore the time complexity would be O(n). However, the worst case is rare, therefore the average insertion time is O(1).

Chaining: For insert, delete, and search, 1 is the constant running time to reach the slot location, m is the number of elements stored, and n is the number of slots available in the hash table. Measuring when to decide to increase the hash table size is m/n. Therefore, the time complexity is O(1+m/n) -> O(1+1) -> O(1).