ECE 250 - Project 2
Hashing
Design Document

Jamie Yen, UW UserID: j6yen

Nov. 7th, 2022

Method 1: Double Hashing

Overview of Classes

Class 1: hash

Description:

Represents a dynamically allocated array to store the data and provides operations such as inserting student numbers with the students' name, searching the student number, deleting, etc.

Member variables(all private):

1. integer m: number of slots the hash table has
2. integer num: number of keys currently stored in the table
3. node type pointer list: address of the hash table

Member functions(all public):

1. **set_size**: create a new hash table with size N
2. **insert_number**: i. the table full or key already in the list? Y- fail to insert and return/ N-do ii, ii. is slot=hash1 empty? Y-do iv/ N-do iii, iii. not empty, offset=hash2 and keep incrementing offset until an empty slot is found, do iv, iv. insert SN and the associated last name
3. **searches**: i. calls found function ii. returns a positive integer? Y-do iii/ N-do iv iii. the SN is found and stored in that index, prints the SN and name iv. prints not found
4. **deletes**: i. calls found function ii. returns a positive integer? Y-do iii/ N-do iv iii. the SN is found and stored in that index, set the SN and name to null iv. deletion fails
5. **hash1**: returns the integer of prob = SN mod m
6. **hash2**: returns the integer of offset = floor(k/m)mod m, +1 to the offset if the resulting value is even
7. **empty**: i. name at index n is empty? Y-return ture/ N-return false
8. **found**: i. key is in prob=hash1? Y-do iii/ N-do ii, ii. keep searching with every time incrementing by offset=hash2 until finding the element or one complete traversal of the hash table is completed iii. found the key? Y-returns it's index/ N- returns -1

Constructor: not used since set_size function has been used to initialize the hash table

Desctructor: deletes the hash table as the program executes to ensure there will not be any memory leaks

Class 2: node

Description:

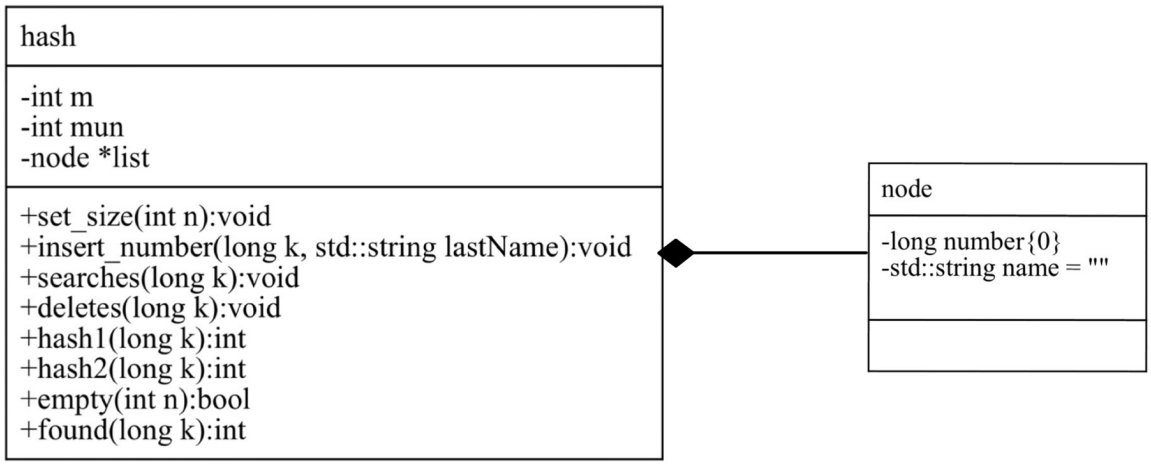
A class representing a node, which can store two entries in it. One of them is a student number, the other is the student's name.

Member variables(all privates):

1. long number
2. string name

Constructor: not used since both entries has already been set to have a null value when each nodes are created
Desctructor: not used since the hash class will be in charge of deleting all the nodes in the table once the program ends

UML Class Diagram



Test Cases
1. Capacity test: the size of the array is “m” and I will have more than “m” times of the “i” command. then I have to see the insertion fails since the table is already full.
2. Finding test: the student number n is stored in slot prob + 2*offset, I will have “s” command and the found function should first check if the student number is stored at prob, then at prob + offset, then prob + 2*offset and find it in the index. I will also have “s” command that the student number is not stored in the list and then the found function will fail to find such number in the table
3. Deletion and insert test: the size of the array is “m” and I will have several times “i” command, call “s” to check they are inserted in the correct spot, have “d” command to delete all the keys inserted, call “s” to check all of the keys have indeed all been deleted, then try have “i” commands m times, all the insertion will be successful since I have clear all the keys before inserting new keys
4. Duplicated test: the student number n is already inserted in the list, and I will insert it again to the table then I have to see insertion fails since it is already in the table

Method 2: Chaining

<u>Class 1: hash</u>
Description:
Represents a vector of type linked list to store the data and provides operations such as inserting student numbers with the students’ name, searching the student number, deleting, etc.
Member variables(all private):
1. integer m: number of slots the hash table has
2. integer num: number of keys currently stored in the table
3. a vector list type table: contains the linked-list head pointer in each vector entry

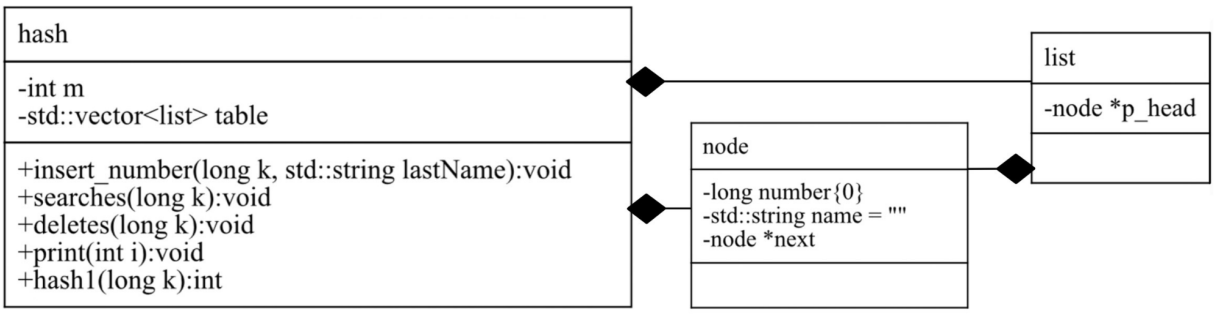
Member functions:
1. **prints**: i. the linked-list at position i of vector an empty list? Y-prints chain is empty and returns/ N-do ii, ii. iterate the whole linked-list and prints all the nodes from the head of the linked-list to tail
2. **insert_number**: i. hash table at index=hash1 empty? insert it ii. key already in the list? fails to insert iii. traverse through the linked list till an appropriate place and insert SN in descending order
3. **searches**: i. searches the linked list at index=hash1 ii. iterating through the linked list and check each node's student number until a null node
4. **deletes**: i. search the linked list at index=hash1 and find if the node we want to delete is in the linked list Y-do ii/ N-fails and return ii. the node is at head? set the new head to the next node, delete it and iii. the node is at the linked-list's tail or in the middle of the linked-list? set the let the previous node points to the next node of the node we want to delete and delete the node
5. **hash1**: returns the integer of prob = SN mod m

Constructor: initialize the size of the vector hash table
Desctructor: deletes the hash table vector when the program executes to ensure there will not be any memory leaks

Class 2: node
Description:
A class representing a node, which can store two entries and also a node type pointer pointing to the next node. One of the entries is a student number, the other is the student's name.
Member variables(all private):
1. Node type pointer p_next
2. long number
3. string name

Class 3: list
Description:
A class that contains the head pointer of a linked-list
Member variables:
1. private node type pointer p_head

II. UML Class Diagram



IV. Test Cases

1. Deletion test: I will have the program to delete a key that is not in the linked-list, it will fail to delete the key and I will see the linked-list remains the same when I print the same linked-list. I will have command “d” to delete a key that is the head of the linked list, I will have to see the rest of the linked list still get printed when I call the “p” command of the same linked-list.
2. Duplicated test: insert the same student numbers twice and I will have to see the second insertion fail since the student number is already in the list
3. Print test: print a linked-list that is empty, I will see the program says the chain is empty
4. Ordered test: insert numbers of student numbers that go to the same table slot, linked list, they will be inserted in descending order. When a largest number is inserted to the linked-list, it should replace the head of the linked list; other numbers will also be inserted in its relative position. When I call the ‘p’ command to print that linked list from its head, I have to see the list being ordered

Performance

The expected runtime of all operations for both methods is constant. To ensure this, I make sure that there is no loop that goes through the whole hash table in any of my functions. For the found function that I used to help implement search and delete, instead of going through the whole hash table, I start checking index prob=hash1 then check for index+offset each time until one traversal of the hash table is complete. Therefore, the average runtime for insert, search, and delete operation can be done in constant.