ECE 250 - Project 2
Phone Directory as Hash Table
Design Document

Harshit Manchanda, UW UserID: h2mancha

Feb 14th, 2020

1. Overview of Classes

Method 1: Double Hashing Hash Table

Class 1: NumCaller

Description:
Represents the phone number and caller ID as an object of this, stored inside the hash table vector.

Member Variables:
1. Phone number
2. Caller ID

For example, a NumCaller object inside the vector will hold values 5374892482 as number and Dave as caller.

Member Functions (operations):
Doesn't have any member functions because it has HashTable as its friend class so the private member variables can be directly accessed and used inside the HashTable class.

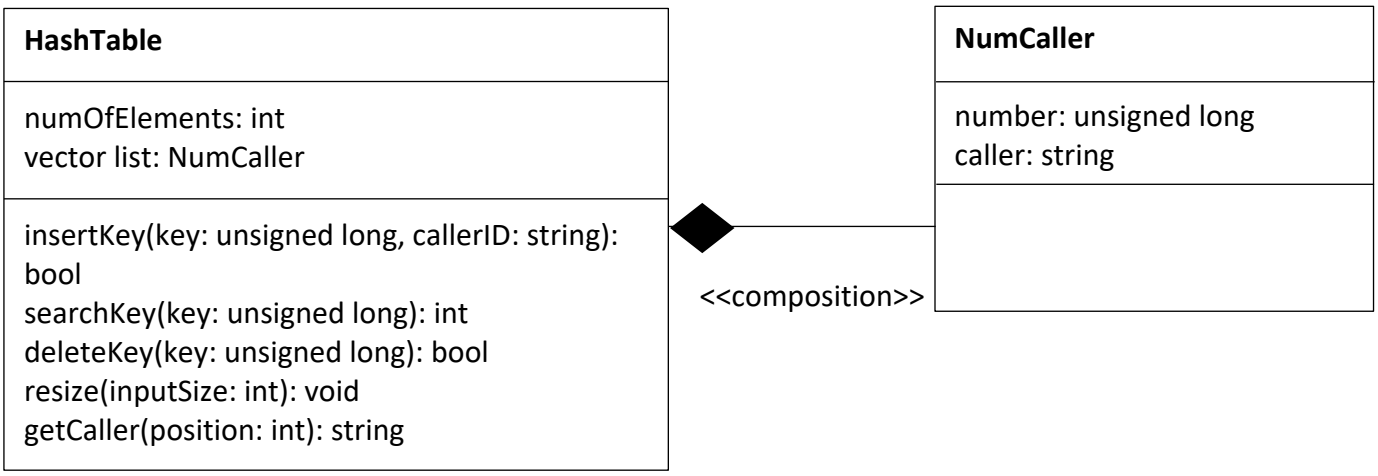
Class 2: HashTable

Description:
Represents the directory of numbers as a vector of type NumCaller and provides operations as adding numbers and callers, deleting the records, searching for them in the table, etc.

Member Variables:
1. Number of records in the table
2. Vector of type NumCaller

Member Functions (operations):
1. insertKey: adds the number and caller ID passed as parameters at the index found by using double hashing.
2. searchKey: searches for the key passed as parameter in the table and prints the caller ID and index if present.
3. deleteKey: removes the key passed as parameter and its respective caller ID if it exists in the table.
4. resize: if the new table command is passed again, it resizes the table with the new capacity passed as parameter.
5. getCaller: it returns the caller ID stored at the position passed as parameter.

UML Class Diagram -



Class NumCaller (Constructor & Destructor):

The constructor and destructor for this class assigns the number and caller with the value 0 and empty string.

Class HashTable (Constructor):
The parameterized constructor takes in the size and reserves space in memory for that size.

Class HashTable (Destructor):
The destructor erases all the elements of the table and shrinks it to nothing.

Method 2: Chaining Hash Table

Class 1: Node

Description:
Represents a node of the single linked list stored in the hash table containing number, caller ID and pointer to next node.

Member Variables:
1. Phone number
2. Caller ID
3. Node type pointer next

For example, a node inside the vector will hold values 5374892482 as number, Dave as caller and will point to next node.

Member Functions (operations):
Doesn't have any member functions because it has HashTable as its friend class so the private member variables can be directly accessed and used inside the HashTable class.

Class 2: LinkedList

Description:
This class helps make the table as the vector is of type LinkedList and nodes are added in front of it.

Member Variables:
1. Node type pointer head

Member Functions (operations):
Doesn't have any member functions because it has HashTable as its friend class so the private member variables can be directly accessed and used inside the HashTable class.

Class 3: HashTable

Member Functions (operations):
1. insertKey: adds the number and caller ID passed as parameters at the index found by using chaining.
2. searchKey: searches for the key passed as parameter in the table and prints the caller ID and index if present.
3. deleteKey: removes the key passed as parameter and its respective caller ID if it exists in the table.
4. resize: if the new table command is passed again, it resizes the table with the new capacity passed as parameter.
5. printTable: it prints all the numbers in the linked list at the position passed as parameter.
6. numOfElements: it returns the number of nodes at the position passed as parameter.

2. Constructors/Destructors

Class Node (Constructor):
The constructor for this class assigns the number, caller and next pointer with the values passed as parameter.

Class Node (Destructor):
The destructor makes the number and caller 0 and empty and also makes the next pointer null.

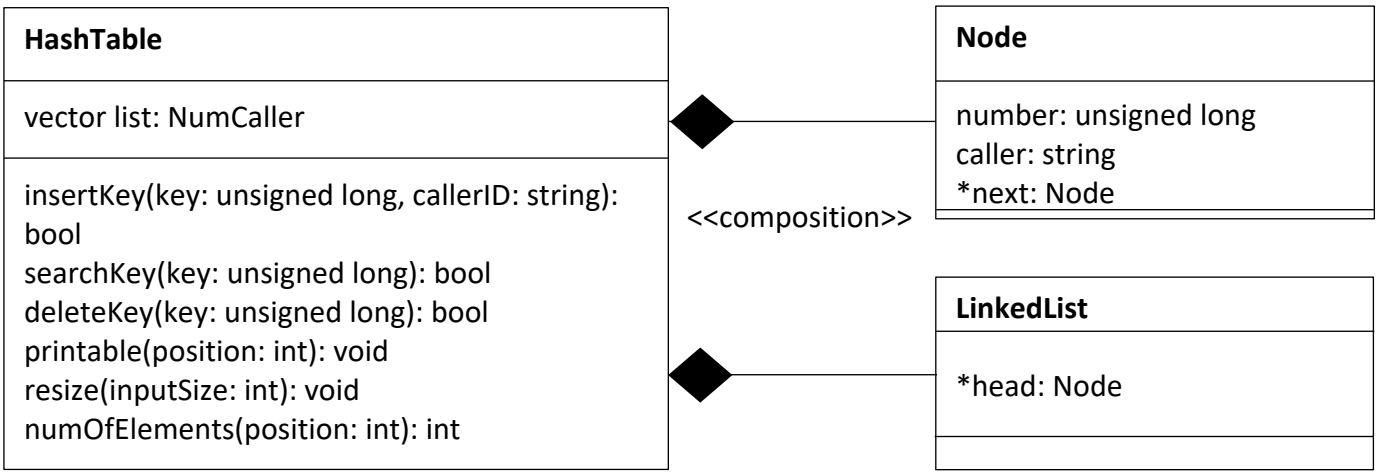
Class LinkedList (Constructor & Destructor):
The constructor and destructor both make the head pointer null.

Class HashTable (Constructor):

The parameterized constructor takes in the size and reserves space in memory for that size.

Class HashTable (Destructor):
The destructor erases all the elements of the table and shrinks it to nothing.

UML Class Diagram -



Test Cases

Test 1(Double Hashing): Create a new table by adding records to check if they are added at the right place or not, delete them, search for deleted items, insert new ones.

Test 2(Chaining): Create a few nodes in the table and then delete some, print them to check if they are ordered or not, search for a number and create a new table and insert new nodes.

Test File Example 1(Double Hashing):

```
n 53
i 5195840179;A
i 8327492181;B
i 5196750336;C
i 6135990591;D
s 5196750336
d 8327492181
n 45
s 5195840179
s 6135990591
i 5623740242;E
i 4252234241;F
s 5623740242
```

Test File Example 2(Chaining):

```
n 10
i 7901308491;A
i 8327492181;B
i 6313718331;C
p 1
d 6313718331
p 1
s 7901308491
s 7484823928
i 5623740242;D
p 4
p 2
```

Performance

Given a Hash Table with m slots holding n values, the average number of keys per slot would be $\alpha = n/m$. The time it would take to compute the hash function would be O(1). Now, assuming slots are proportional to number of elements, n = O(m) and $\alpha = n/m = O(m)/m = O(1)$. The search function would be O(1+ α) which would on average be constant time O(1).
Now insertion would either involve searching if the element doesn't already exist in the list and that would take constant time as seen above. Thus, insertion would also take constant time O(1).
Deletion would also require to search for the element in the table and thus it would also take constant time O(1).