BLG 335E, Analysis of Algorithms I, Fall 2014
Project 3
Handed out: 20.11.2014
Due: 12.12.2012, until 23.00
Hashing

A company wants to store the records of their customers in a hash table. In order to decide on a suitable hashing strategy, they want to analyze the performance of different hashing strategies. The company wants to operate four operations, *insert, delete, update* and *search*.

The following rules should be satisfied with your implementation:

- Each customer is hashed with their national IDs.
- In the beginning of the execution, all the hash slots are empty.
- ***insert(key)*** method should find a suitable slot for the given key value and add the key to the table.
- ***delete(key)*** method should find the given key in the table and delete it from the table. In order to delete a key from a hash table slot, you should just mark the corresponding slot as *deleted* (Lazy deletion).
- Remember that during insertion, slots marked as *deleted* are treated as *empty.*
- ***update(key,new_value)*** method should find the given key in the table and update the key value with the *new_value*. Note that after the update, you may need to update the index of the slot as well (You may delete *key* from the table and insert *new_value*).
- ***search(key)*** method should find the given key in the hash table and return its index. If the key is not in the table, it should return -1.
- During searching a key, if a slot marked with *deleted* is encountered, the search continues. Therefore, *deleted* slots are treated as *occupied* during search.
- It is not allowed to have more than 50 *deleted* marked slot in the table. In such a situation, you should organize the hash table in such a way that there are no *deleted* marked slots.
- You should use corresponding hash functions during all the operations.
- You are given a text file namely “operations.txt”. Every line of the file indicates an operation and the corresponding key values. For example, when “insert 714589763” is encountered, you should add 714589763 key value to the table. When “update 714589763 568745236” is encountered, you should update the corresponding key with its new value 568745236.

a. **(10 points)** Implement an m-sized hash table that uses ***linear probing*** strategy in order to store customer information by using the following hash function. The table should support *insert, delete, update* and *search* operations.

$$h(k,i) = (k+i)\ mod\ m \text{ and } i \in [0, m-1]$$

b. **(15 points)** Implement an m-sized hash table that uses ***double hashing*** strategy in order to store customer information by using the following hash function. The table should support *insert, delete, update* and *search* operations.

$$h(k,i) = (h_1(k) + i * h_2(k))\ mod\ m \text{ and } i \in [0, m-1]$$

$$h_1(k) = k\ mod\ m \text{ and } h_2(k) = 1 + k\ mod\ (m-1)$$

c. **(15 points)** Implement an m-sized hash table that uses ***quadratic hashing*** strategy in order to store customer information by using the following hash function. The table should support *insert, delete, update* and *search* operations. You may choose $c_1 = 1, c_2 = 1$.

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2)\ mod\ m \text{ and } i \in [0, m-1]$$

$$h'(k) = k\ mod\ m$$

d. **(20 points)** Implement an m-sized hash table that uses ***universal hashing*** strategy in order to store customer information. Details of the universal hashing strategy is given below. Decompose key *k* into *r+1* (for your case, r = 2) digits each with a value in the set $[0, m-1]$. Therefore, $k = [k_0, k_1, \dots, k_r]$ where $0 \leq k_i < m$. By using the randomized strategy, pick $a = [a_0, a_1, \dots, a_r]$ where each a_i is chosen randomly from $[0, m-1]$.

Then define the hash function as follows.

$$h_a(k) = \sum_{i=0}^{r} a_i k_i \mod m$$

In case of a collusion, use one of the open addressing methods in order to probe. Example decompositions of some keys for *r = 2* are given as follows:

$k = 154896356 \rightarrow \mathrm{k}_0 = 154, k_1 = 896, k_2 = 356$

$k = 4896356 \rightarrow \mathrm{k}_0 = 4, k_1 = 896, k_2 = 356$

$k = 96356 \rightarrow \mathrm{k}_0 = 0, k_1 = 96, k_2 = 356$

e. **(10 points)** Read the input file "operations.txt" and apply the defined operations on the hash table.

f. (30 points) Number of collisions when inserting an item k is defined as follows:
Initialize *collisions = 0.*
Compute *h(k,i)*, if *h(k,i)* is occupied, then increment *collisions* by 1. During the execution, compute the number of collisions for each strategy and fill in the table with the number of collisions occurred for each strategy and m value.
Comment on the results.

	Insertion			
	Linear	**Quadratic**	**Double**	**Universal**
m = 1327				
m = 2657				

Detailed Instructions:

- All your code must be written in C++ using ***object oriented approach*** and able to compile and run on Linux using ***g++***.
- Do not use additional libraries such as STL.
- Since this is an individual project, you are required to work ***individually***.
- Submissions will be done through the Ninova server. You must submit your source code files (.h and .cpp) and a softcopy report (.pdf).
- In your report, explain your classes and methods briefly.
- During the execution, delete, search and update methods should give explanatory messages to the screen. At the end of the execution, total number of collisions for each strategy and for each m should be printed out. An example output can be seen below: (Note that the numbers in the output are imaginary).

```
.....
Deletion: Key 637789932 is deleted from the table.
Search: Key 256478523 is found at index 548.
Search: Key 124579863 is not found in the table.
Update: Key 124578956 is updated with key 145236789. Its new slot is 478.
.......
Linear Probing: 300 collusions have been occurred (m = 1327).
.......
```