**Assignment 3**

Hashes are used to ensure data and message integrity, password validity, and are the basis of many other cryptographic systems. Hashes are also used in a data structure called a hash table, widely used in computer software for rapid data lookup. Hash functions also accelerate table or database lookup by detecting duplicated records in a large file. Hashes are also useful in cryptography. A cryptographic hash function allows one to easily verify that some input data maps to a given hash value.

Suppose we want to design a system for storing employee records keyed using phone numbers. And we want following queries to be performed efficiently:

1. Insert a phone number and corresponding information.
2. Search a phone number and fetch the information.
3. Delete a phone number and related information.

We can think of using the following data structures to maintain information about different phone numbers.

1. Array of phone numbers and records.
2. Linked List of phone numbers and records.
3. Balanced binary search tree with phone numbers as keys.
4. Direct Access Table.

-

For **arrays and linked lists**, we need to search in a linear fashion, which can be costly in practice. If we use arrays and keep the data sorted, then a phone number can be searched in O(log n) time using Binary Search, but insert and delete operations become costly as we have to maintain sorted order.

With**balanced binary search tree**, we get moderate search, insert, and delete times. All of these operations can be guaranteed to be in O(log n) time.

Another solution that one can think of is to use a **Direct Access Table** where we make a big array and use phone numbers as index in the array. An entry in array is NIL if phone number is not present, else the array entry stores pointer to records corresponding to phone number. Time complexity wise this solution is the best among all, we can do all operations in O(1) time. For example, to insert a phone number, we create a record with details of given phone number, use phone number as index and store the pointer to the created record in table. This solution has many practical limitations. First problem with this solution is extra space required is huge. For example if phone number is n digits, we need O( **m \* 10n** ) space for table where m is size of a pointer to record. Another problem is an integer in a programming language may not store **n** digits.

Due to above limitations Direct Access Table cannot always be used. **Hashing** is the solution that can be used in almost all such situations and performs extremely well compared to above data structures like Array, Linked List, Balanced BST in practice. With hashing we get **O(1)** search time on average (under reasonable assumptions) and O(n) in worst case. **Hashing** is an improvement over Direct Access Table. The idea is to use hash function that converts a given phone number. or any other key to a smaller number and uses the small number as index in a table called hash table.

[**Hash Function**](http://en.wikipedia.org/wiki/Hash_function)**:** A function that converts a given big phone number to a small practical integer value. The mapped integer value is used as an index in hash table. In simple terms, a hash function maps a big number or string to a small integer that can be used as index in hash table. A good hash function should have following properties:

1. Efficiently computable.
2. Should uniformly distribute the keys (Each table position equally likely for each key)

For example, for phone numbers a bad hash function is to take first three digits. A better function is to consider the last four digits. Please note that this may not be the best hash function. There may be better ways.

[**Hash Table**](http://en.wikipedia.org/wiki/Hash_table)**:** An **array** that stores pointers to records corresponding to a given phone number. An entry in hash table is NIL if no existing phone number has hash function value equal to the index for the entry.

**Collision Handling**: Since a hash function gets us a small number for a big key, there is possibility that two keys result in same value. The situation where a newly inserted key maps to an already occupied slot in hash table is called **collision** and must be handled using some collision handling technique. Following are the ways to handle collisions:

* **Chaining:** The idea is to make each cell of hash table point to a linked list of records that have same hash function value. Chaining is simple but requires additional memory outside the table.
* **Open Addressing:**In open addressing, all elements are stored in the hash table itself. Each table entry contains either a record or NIL. When searching for an element, we one by one examine table slots until the desired element is found or it is clear that the element is not in the table.

For this assignment, use the concept of **Hashing** to store customers’ record, which comprises of their telephone number, name, and postal address. Use the last four digits of the telephone number as the hash key, and use the collision resolution method of  **Chaining** to handle any possible collision.

Your program should be able to:

1. Build a hash table by inserting a phone number and the corresponding information.
2. Search a phone number and fetch the information.
3. Delete a phone number and related information.

Supply your own data and provide your own output format. The following incomplete classes may be used as a guide.

|  |  |
| --- | --- |
| class Customer // The data class  {  String name, address, phone;    Customer(String name, String address, String phone)  {      }  } | class CustomerNode // Node for the linked list  {  Customer cust;  CustomerNode next;    CustomerNode( Customer cust)  {  this.cust = cust;  next = null;  }  } |
| class CustomerHashing // Builds table, etc  {  CustomerNode table;    CustomerHashing( )  {    }  } | class TestHashing  {  public static void main(String[] arg)  {      }  } |

You are encouraged to visit these sources, as they may offer some guide.

1. <https://www.youtube.com/watch?v=B4vqVDeERhI&app=desktop>
2. <https://www.youtube.com/watch?v=QWVBu_GlUgI>
3. <https://www.youtube.com/watch?v=SVsT7oG4ap8>
4. Section 5.3 of the textbook