National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Fall 2022, Lab Manual – 12

Course Code: CL-2001	**Course : Data Structures - Lab**
Instructor(s) :	**Abeer Gauher, Sobia Iftikhar**

LAB - 12

Hashing with it's Collision Techniques & Rehashing

Objective: Hashing is a technique or process of mapping keys, values into the hash table by using a hash function. It is done for faster access to elements. Modulo Division is the easiest method to create a hash function.

Hash Function:

1. Division Method
2. Mid Square Method
3. Folding Method

1. Division Method

$$h(k) = k \bmod m$$

Here, h(k) is the hash value obtained by dividing the key value k by size of hash table m using the remainder.

Example: Elements to be placed in a hash table are 42,78,89,64 and let's take table size as 10.

Hash (key) = Elements % table size;

h(k)= k mod m

h(42) = 42 % 10 = 2

h(78) = 78 % 10 = 8

h(89) = 89 % 10 = 9

h(64) = 64 % 10 = 4

Key	Value
0	
1	
2	42
3	
4	64
5	
6	
7	
8	78
9	89

Algorithm

```
Suppose array name is A and n is the size of array.
Step 1: Initialize all array values with -1.
Step 2: Specify the values which needs to be inserted.
Step 3: Calculate key address using modulo division method.
 Set key= value % size
Step 4: If A[key] = = -1
 Set A[key] = value // Insert the value at calculated key or address
28
 Else
 Print: Unable to insert
 [End If]
Step 5: If A[key] = = value
 Print: Search found
 Else
 Print: Search not found
 [End If]
Step 6: Repeat while key< n
 Print: A[key]
 [End while]
Step 7: End
```

2. Mid Square Method

Example:
Suppose the size of the Hash Table (m) = 10 (0 - 9) *maximum digits required for the index is 1*
Element (x) = 12 $\Rightarrow x^2 = 144$
Mid 1 digit of 1**4**4 is 4, so the element x=12 will be stored at the index=4 in the hash table with the size of 10 slots.
Another Example:
Suppose the size of the Hash Table (m) = 1000 (0 - 999) maximum digits required for the index is 3
Element (x) = 87431 $\Rightarrow x^2 = 7644179761$
The possible 3 digit mids of 764***4179***761 are 417 or 179, we can pick any of those mids. If we pick 419 then the element x=87431 will be stored at the index=419 in the hash table with the size of 1000 slots.

3. Folding Shift

In fold shift the key value is divided into parts whose size matches the size of the required address.
Example:
Suppose to calculate hash value for X = 5678 and hash table size 100, we need to follow below steps:
Step 1: The X will be divided into two parts each having two digits i.e.
k1=56 and k2 = 78
Step 2: Adding all key parts
k1 + k2 i.e.
Key= 56 + 78 = 134
After ignoring the carry 1 (because here only two digits are required as hash value) the resulting hash value for 5678 is 34**.**

Algorithm

Algorithm:
Step 1: The folding method is used for creating hash functions starts with the item being divided into equal-sized pieces i.e., the last piece may not be of equal size.
Step 2:The outcome of adding these bits together is the hash value, H(x) = (a + b + c) mod M, where a, b, and c represent the preconditioned key broken down into three parts and M is the table size, and mod stands for modulo.
Step 3:In other words, the sum of three parts of the preconditioned key is divided by the table size. The remainder is the hash key.

Hash Table

Task-1:

Write a Program to implement Hash table and implement the following task.

a. Insert element into the table
b. Search element from the key
c. Delete element at a key
d. Use any simple approach for hash function.
e. Use the values given in the example for **Division Method**.

Implement program for Linear and Quadratic probe for Collision Resolution.

Example:

Let us consider a simple hash function as "key mod 7" and a sequence of keys as 50, 700, 76, 85, 92. Hash them and if collision occurs resolve it with linear probing

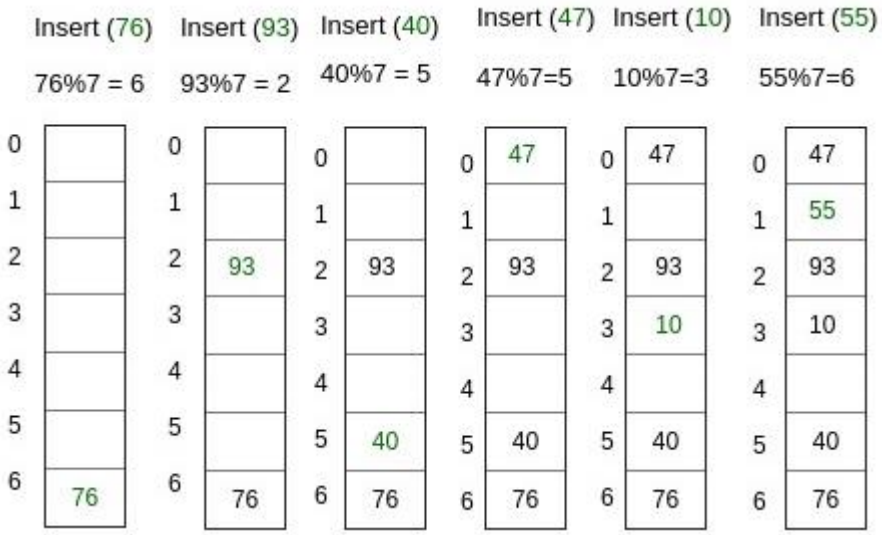
Key	700	50	85	92			76
Location	0	1	2	3	4	5	6

Algorithm

Steps for collision resolution using linear probing are as follows.

Step 1: Calculate the hash key.
address = key % size;

Step 2: If hashTable[key] is empty, store the value directly.
hashTable[key] = data.
If the hash index already has some value, check for next index.
Next address = (key+1) % size;
If the next index is available hashTable[key], store the value.
Otherwise try for next index.
h(k) = (key+i) % size; where i= 0,1,2,3,…
Step 3:Do the above process till we find the space.



Task – 2:

Implement a program using linear and quadratic probing:
Following are the values to be inserted in the hashtable:
3, 2, 9, 6, 11, 13, 7, 12.

Task-3:

Write a Java program to implement the contact book of student using hashing algorithm.
Contact Book is a collection of pairs of the form (K,V). Where 'K' is a Key (student-ID) and 'V' (Contact Number) is the value associated with the key 'k'. Two or more pairs in the contact book have a same key. The following operations are performed on contact book:
Use the key (student – ID) for the hash function to calculate the hash value.

1. Determine whether or not the contact book is empty
2. Determine the contact book Size
3. Find the pair with a specified key
4. Insert a pair into the dictionary
5. Delete or erase the pair into the specified key

Task 04:

The task is to implement all functions of a phone directory:

1. **create_record**
2. **display_record**
3. **delete_record**
4. **search_record**
5. **update_record**

Following data will be taken from the client:

ID, Name, Telephone number

```
1.CREATE record
Enter id :                      4
Enter name :                    XYZ Gupta
Enter telephone number :        23451234



2.DISPLAY record
        ID      NAME            TELEPHONE
        4       XYZ Gupta       23451234



3.SEARCH record
Enter record id to search : 4
Record found:
        ID      NAME            TELEPHONE
        4       XYZ Gupta       23451234



4.UPDATE record
Enter record id to update : 4
Enter name:                     XYZ Agarwal
Enter telephone number:         23413421
Details updated:
        ID      NAME            TELEPHONE
        4       XYZ Agarwal     23413421



5.DELETE record
Enter record id to delete : 4
```

Chaining with Singly Linked Lists

Task 5:

Write a program that will take 5 different user entries (each entry contains name and passwords) consider password for calculating the hash value and store in hash table using any Hash function. If collision occurs, then resolve by Chaining Concept.
For calculating the hashvalue for the password, use the ASCII values for the characters.
Perform all the following operations:

1. Insert the values in the hashtable.
2. Update the password for a user.
3. Delete an entry from the table.
4. Search for a value in the table.

Rehashing

Rehashing is the process of re-calculating the hashcode of already stored entries (Key-Value pairs), to move them to another bigger size hashmap when the threshold is reached/crossed.

Rehashing of a hash map is done when the number of elements in the map reaches the maximum threshold value. Java specification suggests that the Good load factor value is .75 and the default initial capacity of HashMap is 16. Once the number of elements reaches or crosses 0.75 times the capacity, the complexity increases, So to overcome this the size of the array is increased by double and all the values are hashed again and stored in the new array of double size. This is done to reduce the complexity and maintain low load factor. In this case, when the number of elements is 12, rehashing occurs. (0.75 * 16 = 12).

Rehashing – How it's done?

Steps for Rehashing as follows:

- For every new entry into the map, check the load factor.
- If the load factor is greater than its threshold value (default 0.75 for HashMap), then start Rehash.
- For Rehashing, initialize a new array of double the size of the previous one.
- Copy all elements into a new array and make it the new bucket array.

Task 6:

You need to insert 4 Keys: 100, 101, 102, 103. The Hash function used is division method: Key % ArraySize
With the insertion of 3 elements, the load on Hash Table = ¾ = 0.74.
Add the 4th element to this Hash table and increase its size to 6 now (Rehash). Rehash the hash of existing elements using the new size of the table.