K. J. Somaiya College of Engineering, Mumbai-77
(A constituent College of Somaiya Vidyavihar University)

Batch: B2	**Roll. No.: 16010122151**
Experiment: 09	
Grade: AA / AB / BB / BC / CC / CD /DD	

Title: Implementation of hashing concept

Objective: To understand various hashing methods

Expected Outcome of Experiment:

CO	Outcome
CO4	Demonstrate sorting and searching methods.

Websites/books referred:

Abstract: -

(Define Hashing ,hash function, list collision handling methods)

Hashing is a technique or process of mapping keys, and values into the hash table by using a hash function. It is done for faster access to elements. The efficiency ofmapping depends on the efficiency of the hash function used.

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. So, in simple terms we can say that a hash function is used to transform a given key into a specific slot index. Its main job is to map each and every possible key into a unique slot index. If every key is mapped into a unique slot index, then the hash function is known as a perfect hash function. It is very difficult to create a perfect hash function but our job as a programmer is to create such a hash function with the help of which the number of collisions are as few as possible. Collision is discussed ahead.

A good hash function should have following properties:

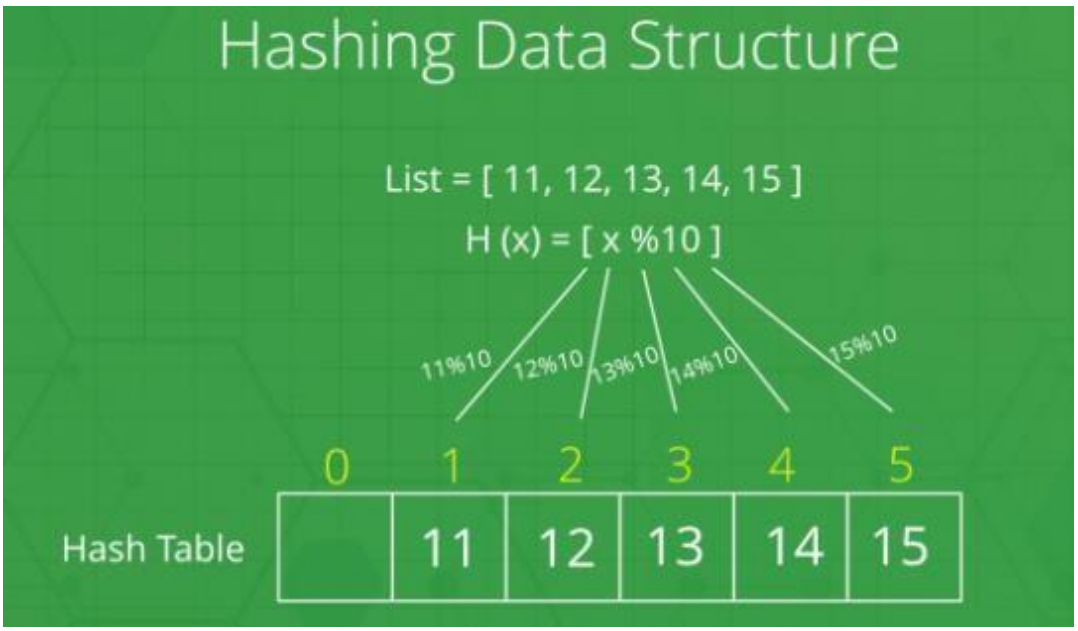
1. Efficiently computable.
2. Should uniformly distribute the keys (Each table position equally likely for each).
3. Should minimize collisions.
4. Should have a low load factor(number of items in table divided by size of the table).

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 examine the table slots one by one until the desired element is found or it is clear that the element is not in the table.

Example:

Let a hash function H(x) maps the value x at the index x%10 in an Array. For example if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.



Code and output screenshots:

```c
#include <stdio.h>
#include <stdlib.h>

struct HashNode {
    int key;
    int value;
};

const int capacity = 20;
int size = 0;

struct HashNode** arr;
struct HashNode* dummy;

void insert(int key, int value) {
    struct HashNode* temp = (struct HashNode*)malloc(sizeof(struct HashNode));
    temp->key = key;
    temp->value = value;

    int hashIndex = key % capacity;

    while (arr[hashIndex] != NULL && arr[hashIndex]->key != key &&
arr[hashIndex]->key != -1) {
        hashIndex++;
        hashIndex %= capacity;
    }

    if (arr[hashIndex] == NULL || arr[hashIndex]->key == -1) {
        size++;
```

```
28          size++;
29      }
30
31      arr[hashIndex] = temp;
32  }
33
34  int delete(int key) {
35      int hashIndex = key % capacity;
36
37      while (arr[hashIndex] != NULL) {
38          if (arr[hashIndex]->key == key) {
39              arr[hashIndex] = dummy;
40              size--;
41              return 1;
42          }
43          hashIndex++;
44          hashIndex %= capacity;
45      }
46
47      return 0;
48  }
49
50  int find(int key) {
51      int hashIndex = key % capacity;
52      int counter = 0;
53
54      while (arr[hashIndex] != NULL) {
55          if (counter++ > capacity) {
56              break;
```

```
57          }
58
59          if (arr[hashIndex]->key == key) {
60              return arr[hashIndex]->value;
61          }
62
63          hashIndex++;
64          hashIndex %= capacity;
65      }
66
67      return -1;
68  }
69
70  int main() {
71      arr = (struct HashNode**)malloc(sizeof(struct HashNode*) * capacity);
72
73      for (int i = 0; i < capacity; i++) {
74          arr[i] = NULL;
75      }
76
77      dummy = (struct HashNode*)malloc(sizeof(struct HashNode));
78      dummy->key = -1;
79      dummy->value = -1;
80
81      insert(1, 5);
82      insert(2, 15);
83      insert(3, 20);
84      insert(4, 7);
85
86      int result = find(4);
```

```
    int result = find(4);
    if (result != -1) {
        printf("Value of Key 4 = %d\n", result);
    } else {
        printf("Key 4 does not exist\n");
    }

    if (delete(4)) {
        printf("Node value of key 4 is deleted successfully\n");
    } else {
        printf("Key does not exist\n");
    }

    result = find(4);
    if (result != -1) {
        printf("Value of Key 4 = %d\n", result);
    } else {
        printf("Key 4 does not exist\n");
    }

    return 0;
}

```

Output:

```
Value of Key 4 = 7
Node value of key 4 is deleted successfully
Key 4 does not exist
```

Conclusion: -

The given experiment was successfully completed and implemented. In this experiment we learnt about Hashing and algorithms related to hashing and also coded for the same in C language.

Post lab questions-

a. **Compare and contrast various collision handling methods.**

Separate Chaining is a hashing technique in which there is a list to handle collisions. So there are many elements at the same position and they are in a list. The sequences are maintained in a linked list.

Linear Probing is a simple collision resolution technique for resolving collisions in hash tables, data structures for maintaining collection of values in a hash table. If there is a collision for the position of the key value then the linear probing technique assigns the next free space to the value.

Quadratic probing also is a collision resolution mechanism which takes in the initial hash which is generated by the hashing function and goes on adding a successive value of an arbitrary quadratic polynomial from a function generated until an open slot is found in which a value is placed.

Double hashing is also a collision resolution technique when two different values to be searched for produce the same hash key.It uses one hash value generated by the hash function as the starting point and then increments the position by an interval which is decided using a second, independent hash function. Thus here there are 2 different hash functions.

b. Store the given numbers in bucket of size 16, resolve the collisions if any with

a. Linear probing

20, 33, 65, 23, 11, 32, 78, 64, 3, 87, 10, 7

Index	Value
0	32
1	33
2	65
3	64
4	20
5	3
6	0
7	87
8	7
9	0
10	10
11	11
12	0
13	0
14	78
15	0