Name: Advait Naik

Roll No: 21354

DSAL - ASSIGNMENT 4

* **TITLE:** HASH-TABLE IMPLEMENTATION

* **PROBLEM STATEMENT:**

Consider telephone database of N clients. Make use of a hash table implementation to quickly look up client's telephone number. Make use of 2 collision handling techniques & compare them using using number of comparisons required to find a set of telephone numbers. (Use linear probing with & without replacement)

* **OBJECTIVE:**

To understand practical implementation & usage of hash table for solving the problems.

* **REQUIREMENTS:**

Eclipse C++ - IDE

* **THEORY:**

- Hash Table is one of the most important data structures that uses a special function known as a hash function that maps a given value with a key to access the elements faster.
- A hash table is a data structure that stores some information, & the information has basically 2 main components i.e. key & value. The hash table

can be implemented with the help of an associative array. The efficiency of mapping depends upon the efficiency of the hash function used for mapping.

- Hashing :-

Hashing is one of the searching techniques that uses a constant time. The time complexity in hashing is O(1). In hashing technique, the hash table & hash function are used. Using the hash function, we can calculate the address at which the value can be stored. The main idea behind the hashing is to create the (key/value) pairs. If the key is given, then the algorithm computes the index at which the value would be stored. It can be written as,

index = hash (key)

- Collision :-

- When 2 different values have the same hash value, then the problem occurs between the 2 values, known as collision. To resolve the collisions, we have some techniques known as collision techniques.
- The following are collision techniques :-

i) Open Hashing ⇒ It is also known as closed addressing

ii) Closed Hashing ⇒ It is also known as open addressing

- Linear Probing :-

When the hash function causes a collision by mapping a new key to a cell of the hash table that is already occupied by another key. Linear probing searches the table for the closest following free location & inserts the new key there. Lookups are performed in the same way, by searching the ~~ta~~ table sequentially starting at the ~~new~~ position given by the hash function, until finding a cell with a matching key or an empty cell. Here, array or hash table is considered circular because when the last slot reached an empty location not found then the search proceeds to the first location of the array

* ALGORITHMS / PSEUDOCODES :

- Class DataItem :-

```
class DataItem {
        int data;
        int key;
};
```

- Hash Method :-

```
int hashCode(int key) {
        return key % size;
}
```

- Search Operation ;-

```
Algorithm * search (key) {
        // get the hash
        hashIndex = hashCode (key) ;
        // move in array until an empty slot is found.
        while ( hashArray [hashIndex] != NULL) {

                if (hashArray [hashIndex] → key == key)
                        return hashArray [ hashIndex]
                ++ hashIndex ;
                hashIndex %= size ;
        }
        return NULL ;
}
```

- Insert Operation ;-

```
Algorithm Insert ( key, Data) {
    item = new DataItem ;
    item → data = Data ;
    item → key = key ;
    hashIndex = hashCode (key) ;
    while (hashArray [hashIndex] != null && hashArray [hashIndex]
                                        → key != -1) {
            ++ hashIndex ;
            hashIndex %=size ;
    }
    hashArray [hashIndex] = item ;
}
```

- Delete Operation :-

```
DataItem * delete (item) {
        key = ~~k~~ item → key ;
        hashIndex = hashCode (key) ;
        while (hashArray [hashIndex != null) {

            if (.hashArray [hashIndex] → key == key) {
                ~~stru~~
                struct DataItem * temp = hashArray [hashIndex];
                hashArray [hashIndex] = dummyItem ;
                return temp ;
            }

            ++ hashIndex ;
            hashIndex %= size ;
        }
        return null ;
}
```

* Test Cases :

No	Description	Input	Expected Output	Actual Output	Result
1)	Insert Operation.	Entry ⇒ (~~ab~~ bac, 784) Table ⇒ 0 - - 1 abc 83 2 - -	Table ⇒ 0 - - 1 abc 83 2 bac 784	Table ⇒ 0 - - 1 abc 83 2 bac 784	Pass

2) Search Operation	Data ⇒ ("ggg") Table ⇒ 0 - - 1 abc 83 2 bac 784	Data not Found	Data not found	Pass
3) Delete Operation	Data ⇒ ("bac") Table ⇒ 0 - - 1 abc 83 2 bac 784	Table ⇒ 0 - - 1 abc 83 2 - -	Table ⇒ 0 - - 1 abc 83 2 - -	Pass

* <u>CONCLUSION</u>:

- Students have understood & implemented the hashing technique.
- Able to use the hash table for efficient solution in searching problems.