ASSINGMENT NO.	4
TITLE	Hash Table implementation
PROBLEM STATEMENT /DEFINITION	Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client's telephone number. Make use of two collision handling techniques and compare them using number of comparisons required to find a set of telephone numbers (Note: Use linear probing with replacement and without replacement)
OBJECTIVE	To understand practical implementation and usage of hash table for solving the problems.
OUTCOME	After successful completion of this assignment, students will be able to implement and use hashing for efficient solution to problems
S/W PACKAGES AND HARDWARE APPARATUS USED	• (64-bit)64-BIT Fedora 17 or latest 64-bit update of equivalent Open-source OS • Programming Tools (64-bit) Latest Open-source update of Eclipse Programming frame work
REFERENCES	1. E. Horowitz S. Sahani, D. Mehata, "Fundamentals of data structures in C++", Galgotia Book Source, New Delhi, 1995, ISBN: 1678298 2. Sartaj Sahani, ―Data Structures, Algorithms andApplications in C++‖, Second Edition, University Press, ISBN:81-7371522 X.
INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. and title 3. Problem definition 4. Learning Objective 5. Learning Outcome 6. Software / Hardware requirement 7. Concepts related Theory 8. Algorithms/ Pseudo Code 9. Class ADT 10. Test cases 11. Conclusion/Analysis

Prerequisites:

- Basic knowledge of linked list, its operations and its implementation, searching techniques
- Object oriented programming features

Learning Objectives:

- To understand practical implementation and usage of hashing for solving the problems

Learning Outcomes:

- After successful completion of this assignment, students will be able to implement and use hashing for efficient solution to problems

Concepts related 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, and the information has basically two main components, i.e., key and 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.

For example, suppose the key value is XYZ and the value is the roll number, so when we pass the key value in the hash function shown as below:

Hash(key)= index;

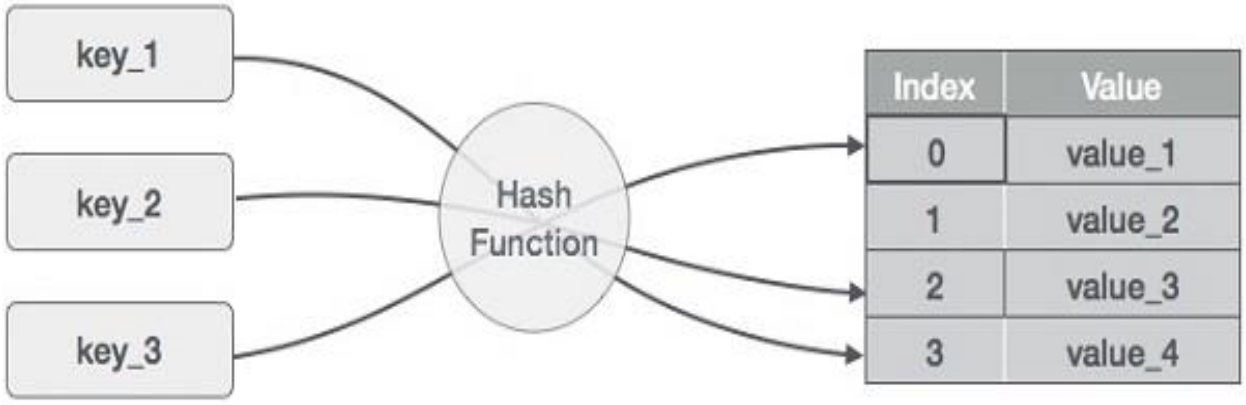
When we pass the key in the hash function, then it gives the index.

Hash(XYZ) = 3;

Hence, the above example adds the john at the index 3.

Hashing

Hashing is one of the searching techniques that uses a constant time. The time complexity in hashing is O(1). We are already familiar with the various techniques for searching, i.e., linear search and binary search. The worst time complexity in linear search is O(n), and O(logn) in binary search. In both the searching techniques, the searching depends upon the number of elements but we want a technique that takes a constant time. So, we use hashing technique that provides a constant time.



In Hashing technique, the hash table and 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:

$$\text{Index} = \text{hash(key)}$$

In the division method, the hash function can be defined as:

$$h(ki) = ki \% m$$

where m is the size of the hash table.

Collision

When the two different values have the same hash value, then the problem occurs between the two values, known as a collision. For example, the key value 6 is stored at index 6. If the key value is 26, then the index would be:

$$h(26) = 26\%10 = 6$$

Therefore, two values are stored at the same index, i.e., 6, and this leads to the collision problem. To resolve these collisions, we have some techniques known as collision techniques.

The following are the collision techniques:

Open Hashing: It is also known as closed addressing.

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 and inserts the new key there. Lookups are performed in the same way, by searching the table sequentially starting at the 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.

Search

To search for a given key x, the cells of T are examined, beginning with the cell at index h(x) (where h is the hash function) and continuing to the adjacent cells h(x) + 1, h(x) + 2, ..., until finding either an empty cell or a cell whose stored key is x. If a cell containing the key is found, the search returns the value from that cell. Otherwise, if an empty cell is found, the key cannot be in the table, because it would have been placed in that cell in preference to any later cell that has not yet been searched. In this case, the search returns as its result that the key is not present in the dictionary

Insertion

To insert a key–value pair (x,v) into the table (possibly replacing any existing pair with the same key), the insertion algorithm follows the same sequence of cells that would be followed for a search, until finding either an empty cell or a cell whose stored key is x. The new key–value pair is then placed into that cell.

Basic Operations

Following are the basic primary operations of a hash table.

- **Search** − Searches an element in a hash table.
- **Insert** − inserts an element in a hash table.
- **Delete** − Deletes an element from a hash table.

ADT:

Class DataItem

Define a data item having some data and key, based on which the search is to be conducted in a hash table.

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

Hash Method

Define a hashing method to compute the hash code of the key of the data item.

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

Search Operation

Whenever an element is to be searched, compute the hash code of the key passed and locate the element using that hash code as index in the array. Use linear probing to get the element ahead if the element is not found at the computed hash code.

Example

```
algorithm *search (key)
{
//get the hash
hashIndex = hashCode(key);
//move in array until an empty
while(hashArray[hashIndex] != NULL) {
if(hashArray[hashIndex]->key == key)
return hashArray[hashIndex];
//go to next cell
++hashIndex;
//wrap around the table
hashIndex %= SIZE;
}
return NULL;
}
```

Insert Operation

Whenever an element is to be inserted, compute the hash code of the key passed and locate the index using that hash code as an index in the array. Use linear probing for empty location, if an element is found at the computed hash code.

Example

```
algorithm insert( key, data)
{
item= new DataItem;
item->data = data;
item->key = key;
//get the hash
hashIndex = hashCode(key);
//move in array until an empty or deleted cell
while(hashArray[hashIndex] != NULL && hashArray[hashIndex]->key != -1) { //go to next cell
++hashIndex;
//wrap around the table
hashIndex %= SIZE;
}
hashArray[hashIndex] = item;
}
```

Delete Operation

Whenever an element is to be deleted, compute the hash code of the key passed and locate the index using that hash code as an index in the array. Use linear probing to get the element ahead if an element is not found at the computed hash code. When found, store a dummy item there to keep the performance of the hash table intact.

Example

```
DataItem* delete (item) {
key = item->key;
//get the hash
hashIndex = hashCode(key);
//move in array until an empty
while(hashArray[hashIndex] !=NULL) {
if(hashArray[hashIndex]->key == key) {
struct DataItem* temp = hashArray[hashIndex];
//assign a dummy item at deleted position
hashArray[hashIndex] = dummyItem;
return temp;
}
//go to next cell
```

```
++hashIndex;
//wrap around the table
hashIndex %= SIZE;
}
return NULL;
}
```

Conclusion:

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

Sample Questions:

1. What is meant by hashing?
2. Why do we use hashing?
3. Explain the concept of hash table and hash function
4. Explain the types of hashing techniques
5. What is meant by collision?
6. How do you perform collision resolution.