Page No.____ Date: / /

Assignment - 6.

Title : The Dictionary ADT.

Problem : Implement all the functions of a dictionary using hashing. Data: set of (key, value) pairs, keys are mapped to values, keys must be comparable, keys must be unique standard operations : Insert (key, value), find(key), Delete(key)

Objective : To understand the implementation of all the functions of a dictionary and standard operations on Dictionary.

Outcome : At the end of ~~their~~ this assignment students will be able to perform standard operations on Dictionary ADT.

Theory :

The Dictionary ADT : A dictionary is an ordered or unordered list of key-element pairs, where keys are used to locate elements in the list

Dictionary is a data structure, which is generally an association of unique keys with some values. One may find a value to a key delete a key (and naturally an associated value) and look up for a value by the key. Values are not required to be unique.
A Dictionary can be implemented in various

ways: using a list, binary search tree, hash table etc.

* Hashing: Hashing is a technique to convert a range of key values into a range of indexes of an array.
In hashing, large keys are converted into small keys by using hash functions. The values are then stored in a data structure called hash table. The idea of hashing is to distribute entries uniformly across an array. Each element is assigned a key. By using that key you can access the element in O(1) time.

* Hash function: A hash function is any function that can be used to map a data set of an arbitrary size to a data set of a fixed size which falls into the hash table. The values returned by a hash values, hash code, hash sum or simply hashes.

* Hash Table: A hash table is a data structure that is used to store keys/value pairs. It uses a hash function to compute an index into an array in which an element will be inserted or searched.

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.

Page No.______
Date: / /

The situation where a newly inserted key maps to an already occupied slot in hash table is called collision.

Following are the ways to handle collision:

1) Chaining: The idea is to make each cell of hash table point to a linked list of records that have same hash functions value. Chaining is simple, but requires additional memory outside the table.

2) Open Addressing: In open addressing, all elements are stored in the hash table it self. 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.

Implementation of Hash Table:
Consider a dictionary, where keys are integers in the range [0, N-1]. Then an array of size N can be used to represent the dictionary. Each entry in this array is thought of as a "bucket". An element 'e' with key 'k' is inserted in A[k]. Bucket entries associated with keys not present in dictionary contains a special NO-SUCH-KEY object. If the dictionary contains elements with the same key then two or more different elements may be mapped to the same bucket of A. In this case,

Page No.
Date: / /

we say that a collision between these elements has occured. One easy way to deal with collisions is to allow a sequence of elements with the same key, k, to be stored in A[k].

Assuming that an arbitrary element with key k satisfies queries find item(k) and remove item(k), these operations are now performed in O(1) time, while insert item(k,e) needs only to find where on the existing list & n(k) to insert the new item, e. The drawback of this is that the size of the bucket array is the size of the set from which key are drawn, which may be huge.

Algorithm:

Hash Node class Declaration:

```
class HashNode {
public:
 int key;
 int value;
 HashNode *next;
 HashNode (int key, int value) {
    this -> key = key;
    this -> value = value;
    this -> next = NULL;
  };
```

Page No.______ Date: / /

Insertions :

```
void insert (int key , string value)
{
    int k = key-hash (key);
    if (arr[k] != 1) {
    int t = traverse (k);
    }
  arr[t] -> key = key ;
    arr[t] -> meaning = value; }
 else {
   arr[k] -> key = key;
  arr[k] -> meaning = value;
   }
```

Deletion :

```
void remove (int key) {
 int k = key-hash (key);
 if (arr[k] -> key == key) {
    arr [k] -> key = -1;
     arr [k] -> meaning = " ";
  }
else {
    int t = traverse (k);
     if (arr[t] -> key == key) {
          arr [t] -> key = -1;
          arr [t] -> meaning = " "; }
       else {
              cout << "key not found";
       }
    }
```

Page No.
Date: / /

Search :

```
void search(int key) {
int k = keyhash(key);
   if (arr[k] -> key == key)
      cout << "key found";
   else {
      int t = traverse[k];
         if (arr[t] -> key == key)
            cout << "key found";
         else
            cout << "key not found";
      }
   }
```

Test Cases :

	Test Case	Expected	Outcome	Result.
1)	Insert - 25, 26, 35, 30 36, 40, 41, 49, 39, 33	0 - 30	As expected	Pass.
		1 - 40		
		2 - 41		
	Hash function = key%10	3 - 39		
		4 - 33		
		5 - 25		
		6 - 26		
		7 - 35		
		8 - 36		
		9 - 49		

Page No.______
Date: / /

TestCase	Expected	Outcome	Result
Insert -21, 24, 30 09, 04, 14, 28, 18, 15, 12	0 - 30 1 - 21 2 - 18 3 - 12 4 - 24 5 - 4 6 - 14 7 - 15 8 - 28 9 - 9	As expected	Pass

Conclusion: After successfully completing this assignment we have learned implementation of Dictionary (ADT) using hashing and various ~~Stard~~ standard operations on Dictionary ADT.