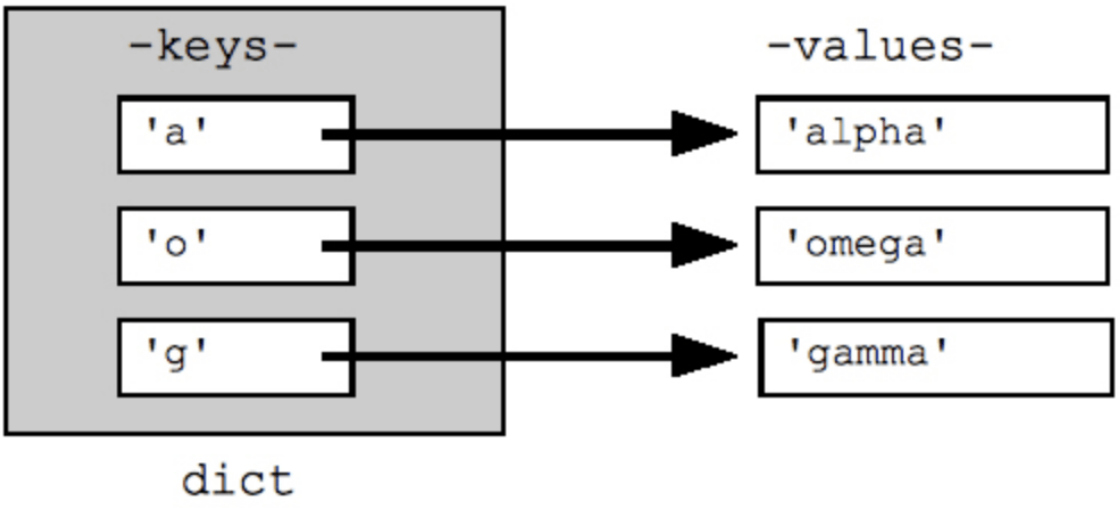
ASSINGMENT NO.	5
TITLE	The Dictionary ADT implementation using open hashing technique
PROBLEM STATEMENT /DEFINITION	Implement all the functions of a dictionary (ADT) using open hashing technique: separate chaining using linked list Data: Set of (key, value) pairs, Keys are mapped to values, Keys must be comparable, and Keys must be unique. Standard Operations: Insert (key, value), Find(key), Delete(key)
OBJECTIVE	To understand implementation of all the functions of a dictionary (ADT) and standard operations on Dictionary using open hashing technique.
OUTCOME	At the end of this assignment, students will able to perform standard operations on Dictionary ADT using open hashing technique.
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, TC++, GTK++.
REFERENCES	• E. Horowitz S. Sahani, D. Mehata, "Fundamentals of data structures in C++", Galgotia Book Source, New Delhi, 1995, ISBN: 1678298 • Sartaj Sahani, ―Data Structures, Algorithms and Applications in C++‖, Second Edition, University Press, ISBN:81-7371522 X.
INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning Outcome 6. Concepts related Theory 7. Algorithm 8. Test cases 9. Conclusion/Analysis

Prerequisites:

- Basic knowledge of Dictionary and Hashing.
- Object oriented programming, features and basic concepts of link list data structures.

Concepts related 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 bind 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.

Example: Consider a data structure that stores bank accounts; it can be viewed as a dictionary, where account numbers serve as keys for identification of account objects.

A Dictionary (also known as Table or Map) can be implemented in various ways: using a list, binary search tree, hash table, etc.

In each case the data structure has to be able to hold key-value pairs and able to do insert, find, and delete operations according to the key.

Hashing

Hashing is a method for directly referencing an element in a table by performing arithmetic transformations on keys into table addresses. This is carried out in two steps:

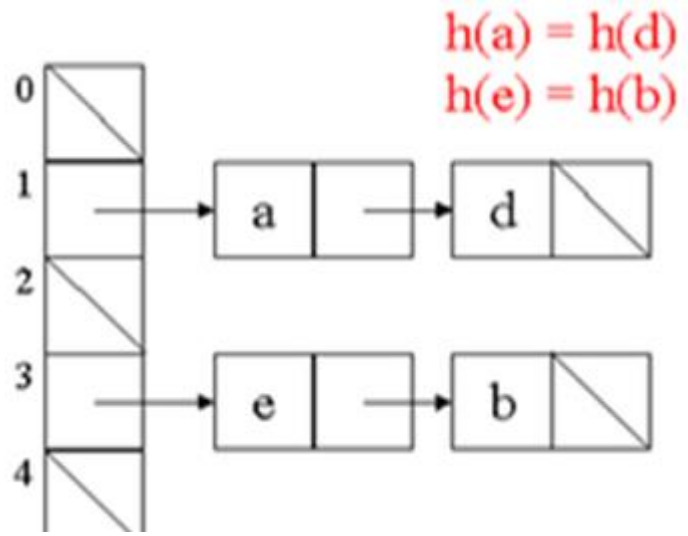
1. Computing the hash function H: K -> A.
2. Collision resolution, which handles cases where two or more different keys hash to the same table address.

Hashing is a popular technique for quickly storing and retrieving data. The primary reason for using hashing is that it produces optimal results by performing optimal searches.

Components of Hashing

There are majorly three components of hashing:

1. Key: A Key can be any string or integer which is fed as input in the hash function that determines an index or location for storage of an item in a data structure.
2. Hash Function: The hash function receives the input key and returns the index of an element in an array called a hash table. The index is known as the hash index.
3. Hash Table: Hash table is a data structure that maps keys to values using a special function called a hash function. Hash stores the data in an associative manner in an array where each data value has its own unique index.



Uses array of linked list to resolve the collision.

Algorithm:

1. Declare an array of a linked list with the hash table size.
2. Initialize an array of a linked list to NULL.
3. Find hash key.
4. If chain[key] == NULL

    Make chain[key] points to the key node.

  Otherwise(collision),

    Insert the key node at the end of the chain[key].

ADT

```
class HashNode

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

Pseudocode for Insertion Operation:

algorithm Insert(int key, int value)

```
{
hash_val = HashFunc(key);
HashNode* prev = NULL;
HashNode* entry = htable[hash_val];
while (entry != NULL)
{
prev =entry;
entry = entry->next;
}
if (entry == NULL)
{
entry = new HashNode(key, value);
if (prev == NULL)
{
htable[hash_val] = entry;
}
else
{
prev->next = entry;
}
}
else
{
entry->value = value;
}
}
```

Pseudocode for Deletion Operation:

```
Algorithm remove(int key)
{
 hash_val = HashFunc(key);
HashNode* entry = htable[hash_val];
HashNode* prev = NULL;
if (entry == NULL || entry->key != key)
{
Print("No Element found at key ",key);
return;
}
while (entry->next != NULL)
{
prev = entry;
entry = entry->next;
}
if (prev != NULL)
```

```
{
prev->next = entry->next;
}
delete entry;
print("Element Deleted");
}
```

Pseudocode for Search Operation:

```
algorithm Search(int key)
{
bool flag = false;
hash_val = HashFunc(key);
HashNode* entry = htable[hash_val];
while (entry != NULL)
{
if (entry->key == key)
{
Print("entry found");
flag = true;
return;
}
entry = entry->next;
}
if (!flag)
return -1;
}
};
```

Conclusion: The Dictionary (ADT) using open Hashing and various standard operations on Dictionary ADT are successfully implemented.

Review Questions:

1. In what ways is a dictionary similar to an array? In what ways are they different?
2. What does it mean to hash a value?
3. What is a hash function?
4. What is a perfect hash function?
5. What is meant by collision of two values in hashing?
6. What does it mean to probe for a free location in an open address hash table?
7. What is the load factor for a hash table?
8. Why do you not want the load factor to become too large?
9. Can you come up with a perfect hash function for the names of the week? The names of the months? The names of the planets?