St. Francis Institute of Technology, Mumbai-400 103
Department of Information Technology

A.Y. 2020-2021
Class: SE-ITA/B, Semester: III
Subject: DATA STRUCTURE LAB

Experiment – 7 Hashing and collision resolution techniques

1. **Aim:** Write a C program to construct a hash table using hashing and collision resolution techniques.

2. **Objectives:** After study of this experiment, the student will be able to
   - Understand the concept of hashing and its application.

3. **Outcomes:** After study of this experiment, the student will be able to
   - Implement different collision resolution techniques.
   - Understand the concepts and apply the techniques of hashing

4. **Prerequisite:** Hashing and collision.

5. **Requirements:** PC and Turbo C compiler version 3.0

6. **Pre-Experiment Exercise:**
   Brief Theory:

A. What is Hash function ?

A **hash function** is any function that can be used to map data of arbitrary size to fixed-size values. The values returned by a hash function are called *hash values*, *hash codes*, *digests*, or simply *hashes*. The values are used to index a fixed-size table called a *hash table*. Use of a hash function to index a hash table is called *hashing* or *scatter storage addressing*.

Hash functions and their associated hash tables are used in data storage and retrieval applications to access data in a small and nearly constant time per retrieval, and storage space only fractionally greater than the total space required for the data or records themselves. Hashing is a computationally and storage space efficient form of data access which avoids the non-linear access time of ordered and unordered lists and structured trees, and the often-exponential storage requirements of direct access of state spaces of large or variable-length keys.

Use of hash functions relies on statistical properties of key and function interaction: worst case behavior is intolerably bad with a vanishingly small probability, and average case behavior can be nearly optimal (minimal collisions).

Hash functions are related to (and often confused with) checksums, check digits, fingerprints, lossy compression, randomization functions, error-correcting codes, and ciphers. Although the concepts overlap to some extent, each one has its own uses and requirements and is designed and optimized differently.

B. Explain different types of hash function?

Open addressing, or **closed hashing**, is a method of collision resolution in hash tables. With this method a hash collision is resolved by **probing**, or searching through alternate locations in the array (the *probe sequence*) until either the target record is found, or an unused array slot is found, which indicates that there is no such key in the table. Well-known probe sequences include:

- **Linear probing** in which the interval between probes is fixed — often set to 1.
- **Quadratic probing** in which the interval between probes increases quadratically (hence, the indices are described by a quadratic function).
- **Double hashing** in which the interval between probes is fixed for each record but is computed by another hash function.

The main tradeoffs between these methods are that linear probing has the best cache performance but is most sensitive to clustering, while double hashing has poor cache performance but exhibits virtually no clustering; quadratic probing falls in-between in both areas. Double hashing can also require more computation than other forms of probing.

C. Explain collision resolution techniques?

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

7. Laboratory Exercise

A. Procedure

Write a C program to construct hash table using hashing and collision resolution techniques.

```
// C program to implement hash table and collision resolution

#include <stdio.h>
#define MAX 10
int hashtbl[MAX];
int a[MAX];

// main function
int main()
{
```

```
    int i,j,m,hash;
    // load the array from user
    printf("\n Enter size of array:");
    scanf("%d", &m);
    printf("\n Enter the values in array:");
    for (i=0;i<m;i++)
        scanf("%d", &a[i]);

    // initialise the hash table
    //with values 999
    for (int i = 0; i < m; i++)
    {
        hashtbl[i]=999;
    }

    // apply hashing with linear probing
    for (i=0;i<m;i++)
    {
        j=0;
        do
        {
            hash=(a[i]%m)+j;
            j++;
            if (hash>=m)
            {
                hash=hash%m;
            }

        }
        while(hashtbl[hash]!=999);
        hashtbl[hash] = a[i];
    }

    // display the hashtable
    //Display array hashtbl
    printf("\nDisplaying Hashtable:");
    printf("\n Index \t Key");
    for (int i = 0; i < m; i++)
    {
        printf("\n %d \t %d",i,hashtbl[i]);
    }
    return 0;
}
```

B. Result/Observation/Program code:

Observe the output for the above code and print it.

```
D:\College\DSA\Experiments\Exp7>Exp7

 Enter size of array:10

 Enter the values in array:98 20 94 27 67 99 41 0 4 17

Displaying Hashtable:
 Index   Key
 0       20
 1       99
 2       41
 3       0
 4       94
 5       4
 6       17
 7       27
 8       98
 9       67
D:\College\DSA\Experiments\Exp7>
```

8. Post-Experiments Exercise

A. Questions:

1. Hash the following in a table of size 12. Use any two-collision resolution technique 98, 20, 94, 27, 67, 99, 41, 0, 4, 17, 2, 15

B. Conclusion:

1. Summary of Experiment
2. Importance of Experiment

C. References:

1. S. K Srivastava, Deepali Srivastava; Data Structures through C in Depth; BPB Publications; 2011.
2. Reema Thareja; Data Structures using C; Oxford.
3. Data Structures A Pseudocode Approach with C, Richard F. Gilberg & Behrouz A. Forouzan, second edition, CENGAGE Learning.

~~Yas~~ Yash Mahajan SE ITB 04 191061

8. Post-Experiment Exercise:-

1. Hash the following in a Table of size 12 use any two collision resolution technique
98, 20, 94, 27, 67, 99, 91, 0, 4, 17, 2, 15

i) Using linear probing:-

Let $h'(k) = k \bmod m$, $m = 12$

Initially,

0	1	2	3	4	5	6	7	8	9	10	11
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Step 1:-
key = 98
$h(98,0) = (98 \bmod 12 + 0) \bmod 12$
$= (2 \bmod 12)$
$= 2$
Inserting 98 in T[2]

Step 2:-
key = 20.
$h(20,0) = (20 \bmod 12 + 0) \bmod 12$
$= 8 \bmod 12$
$= 8$
Inserting 20 at T[8]

Yash Mahajan SE IT B 04 191061

Page:
Date: / /

~~to~~ Step 3

key = 94

$h(94,0) = (94 \bmod 12 + 0) \bmod 12$

$= 10$

Inserting 94 at T[10]

Step 4 :-

key = 27

$h(27,0) = (\cancel{84}\ 27 \bmod 12 + 0) \bmod 12$

$= 3$

Inserting 27, at T[3]

Step 5 :-

key = 67

$h(67,0) = (67 \bmod 12 + 0) \bmod 12$

$= 7$

Inserting 67 at T[7]

Step 6 :-

key = 99

$h(99,0) = (99 \bmod 12 + 0) \bmod 12$

$= 3$

But T[3] is occupied

$\therefore i = 1$

$h(99,1) = (99 \bmod 12 + 1) \bmod 12$

$= 4$

Inserting 99 at T[4]

Yash Mahajan SE IT B 04 191061

Page:
Date: / /

Step 7 :-

key = 41

$h(41,0) = (41 \bmod 12 + 0) \bmod 12$

$= 5$

Inserting 41 at T[5]

Step 8 :-

key = 0

$h(0,0) = (0 \bmod 12 + 0) \bmod 12$

$= 0$

Inserting 0 at T[0]

Step 9 :-

key = 4

$h(4,0) = (4 \bmod 12 + 0) \bmod 12$

$= 4$

T[4] is occupied

$i = 1$

$h(4,1) = (4 \bmod 12 + 1) \bmod 12$

$= 5$

T[5] is occupied

$i = 2$

$h(4,2) = (4 \bmod 12 + 2) \bmod 12$

$= 6$

Inserting 4 at T[6]

Step 10 :-

key = 17

$h(17,0) = (17 \bmod 12 + 0) \bmod 12$

$= 5$

Yash Mahajan SE IT B 04 191061

Page:
Date: / /

T[5] is occupied.

i = 1

h(17, 1) = (17mod 12 + 1) mod 12

= 6

T[6] is occupied

similarly performing for

i = 2, i = 3, i = 4

we get h(17, 4) = (17mod 12 + 4) mod 12

= 9

Inserting 17 at T[9]

Step 11

key = 2

h(2, 0) = (2mod 12 + 0) mod 12

= 2

T[2] is occupied

i = 1

h(2, 1) = (2 mod 12 + 1) mod 12

= 3

T[3] is occupied

similarly solving for i = 2, 3, 4, 7, 8, 9

h(2, 10) = (2mod 12 + 9) mod 12

= 11

Inserting 2 at T[11]

Yash Mahajan SE IT B 04 191061

Page :
Date : / /

Step 12 :-

key = 15

$h(15,0) = (15 \bmod 12 + 0) \bmod 12$

$= 3$

T[3] is occupied

$i = 1$

$h(15,1) = (15 \bmod 12 + 1) \bmod 12$

$= 4$

T[4] is occupied

Similarly trying for i = 2, 3, 4, 5, ~~6~~, 7, 8, 9, 10

$h(15,10) = (15 \bmod 12 + 10) \bmod 12$

$= 1$

Inserting 15 at T[1]

Final hash table is

0	15	98	27	99	41	4	67	20	17	94	2

Yash Mahajan SE IT B 04191061

Page:
Date: / /

2) Collision Resolution by chaining.

98, 20, 94, 207, 67, 99, 41, 0, 4, 17, 2, 15

$h(k) = k \bmod m, \quad m = 12$

Initially Hash table is given as.

0	Null
1	Null
2	Null
3	Null
4	Null
5	Null
6	Null
7	Null
8	Null
9	Null
10	Null
11	Null

Step 1:-

key = 98

$h(k) = 98 \bmod 12 = 2$

Storing 98 in a linked list at position 2.

Step 2:-

key = 20

$h(k) = 20 \bmod 12 = 8$

Storing 20 in a linked list at position 8

Yash Mahajan SE IT B 04 191061

Step 3:-
key = 94
h (k) = 94 mod 12 = 10
Storing 98 in a linked list at position 10.

Step 4
Key = 27
h(k) = 27 mod 12 = 3
Storing 27 in a linked list at position 3.

Step 5:-
key = 67
h(k) = ~~8~~67 mod 12 = 7
Storing 67 ~~at~~ in a linked list at position 7

Step 6:-
key = 99
h (k) = 99 mod 12 = 3
Storing 99 at the end of linked list at position 3.

Step 7:-
Key = 41
h(k) = 41 mod 12 = 5
Storing 41 ~~at~~ in a linked list at position 5.

Step 8:-
key = 0
h (k) = 0 mod 12 = 0
Storing 0 in linked list at position 0.

Yash Mahajan SE IT B 04 191061

Step ~~8~~ 9 :-
key = 4
h (k) = 4 mod 12 = 4
Storing 4 at ~~position~~ linked list at position 4.

Step 10 :-
key = 17
h (k) = 17 mod 12 = 5
Storing 17 at end of linked list at position 5.

Step 11 :-
key = 2
h (k) = 2 mod 12 = 2
Storing 2 at the end of linked list at position 2

Step 12 :-
key = 15
~~2to~~ h (k) = 15 mod 12 = 3
Storing 15 at the end of linked list at position 3.
Final Hash table :-

0 → | 0 | X |
1 Null
2 → | 98 | | → | 2 | X |
3 → | 27 | | → | 99 | | → | 15 | X |
4 → | 4 | X |
5 → | 41 | | → | 17 | X |
6 Null
7 → | 67 | X |
8 → | 20 | X |
9 Null
10 → | 94 | X |
11 Null

Yash Mahajan SE IT B 04 191061

Page:
Date: / /

B. Conclusion :-

In this experiment we have written programs in C to implement hashing and collision resolution techniques. The program takes an integer array as input and uses division method to hash the key and store it in the hash table and uses linear probing for collision resolution.

Hash Table is a data structure in which keys are mapped to array positions by a hash function. A value stored in hash table can be searched in O(1) time. Hash Tables are widely used in situations where enormous amounts of data have to be accessed to quickly search and retrieve information. It is also used for database indexing.