ASSIGNMENT NO.7.

Aim :- Insert the keys into a hash table of length m using open addressing using double hashing with h(k)=1+(k mod(m-1)).

Objective:- to study the hashing concept and its techniques.

Theory:-

Open Addressing—

Like separate chaining, open addressing is a method for handling collisions. In Open Addressing, all elements are stored in the hash table itself. So at any point, size of the table must be greater than or equal to the total number of keys (Note that we can increase table size by copying old data if needed).
Insert(k): Keep probing until an empty slot is found. Once an empty slot is found, insert k.

Search(k): Keep probing until slot's key doesn't become equal to k or an empty slot is reached.

Delete(k): ***Delete operation is interesting***. If we simply delete a key, then search may fail. So slots of deleted keys are marked specially as "deleted".
Insert can insert an item in a deleted slot, but the search doesn't stop at a deleted slot.

Open Addressing is done following ways:

a) Linear Probing: In linear probing, we linearly probe for next slot. For example, typical gap between two probes is 1 as taken in below example also.
let **hash(x)** be the slot index computed using hash function and **S** be the table size

```
If slot hash(x) % S is full, then we try (hash(x) + 1) % S

If (hash(x) + 1) % S is also full, then we try (hash(x) + 2) % S

If (hash(x) + 2) % S is also full, then we try (hash(x) + 3) % S
```

Double Hashing:-

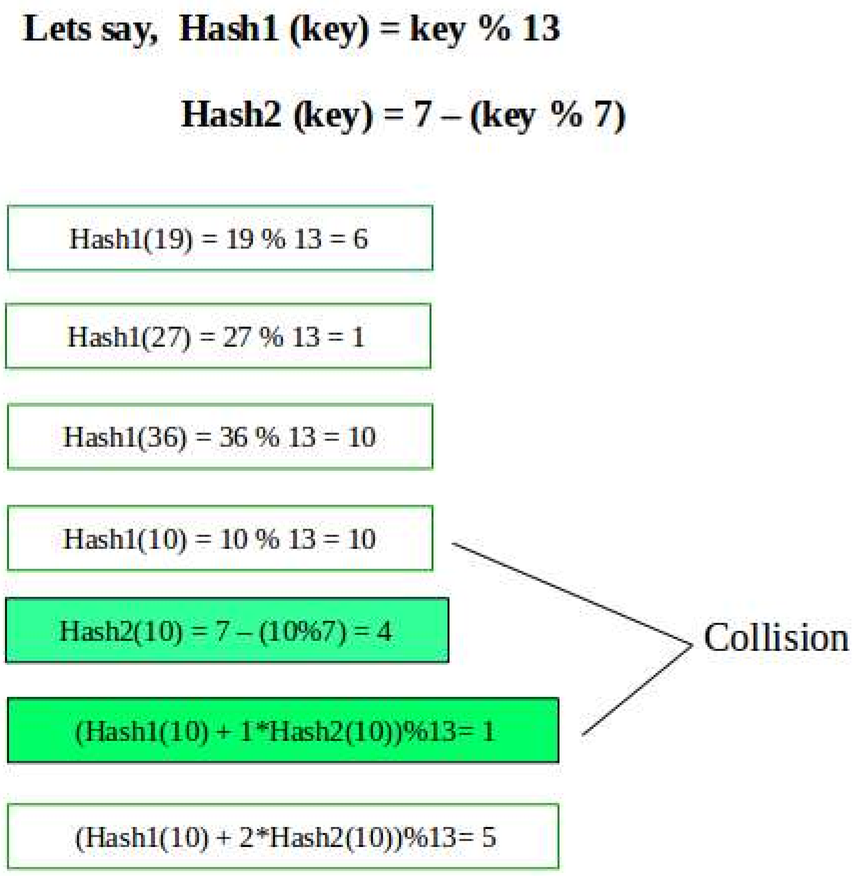
Double hashing is a collision resolving technique in **Open Addressed** Hash tables. Double hashing uses the idea of applying a second hash function to key when a collision occurs.
Double hashing can be done using :
(hash1(key) + i * hash2(key)) % TABLE_SIZE
Here hash1() and hash2() are hash functions and TABLE_SIZE

is size of hash table.
(We repeat by increasing i when collision occurs)

First hash function is typically hash1(key) = key % TABLE_SIZE

A popular second hash function is : **hash2(key) = PRIME – (key % PRIME)** where PRIME is a prime smaller than the TABLE_SIZE.
A good second Hash function is:

- It must never evaluate to zero
- Must make sure that all cells can be probed


Lets say, Hash1 (key) = key % 13
Hash2 (key) = 7 – (key % 7)
Hash1(19) = 19 % 13 = 6
Hash1(27) = 27 % 13 = 1
Hash1(36) = 36 % 13 = 10
Hash1(10) = 10 % 13 = 10
Hash2(10) = 7 – (10%7) = 4
(Hash1(10) + 1*Hash2(10))%13= 1
(Hash1(10) + 2*Hash2(10))%13= 5
Collision

Algorithm:-

1.Start.

2. Accept the size of the table.

3. Initialize the hash table array to any negative integer value say "-111"(Provided negative keys are not accepted in the table).

4. Map the key to it’s value, using first hash function: hash1(key) = key % Table_size. Skill Development Lab-2 ,2018-19 S.Y.-C,Department of Computer Engineering,VIIT,2018-19 3

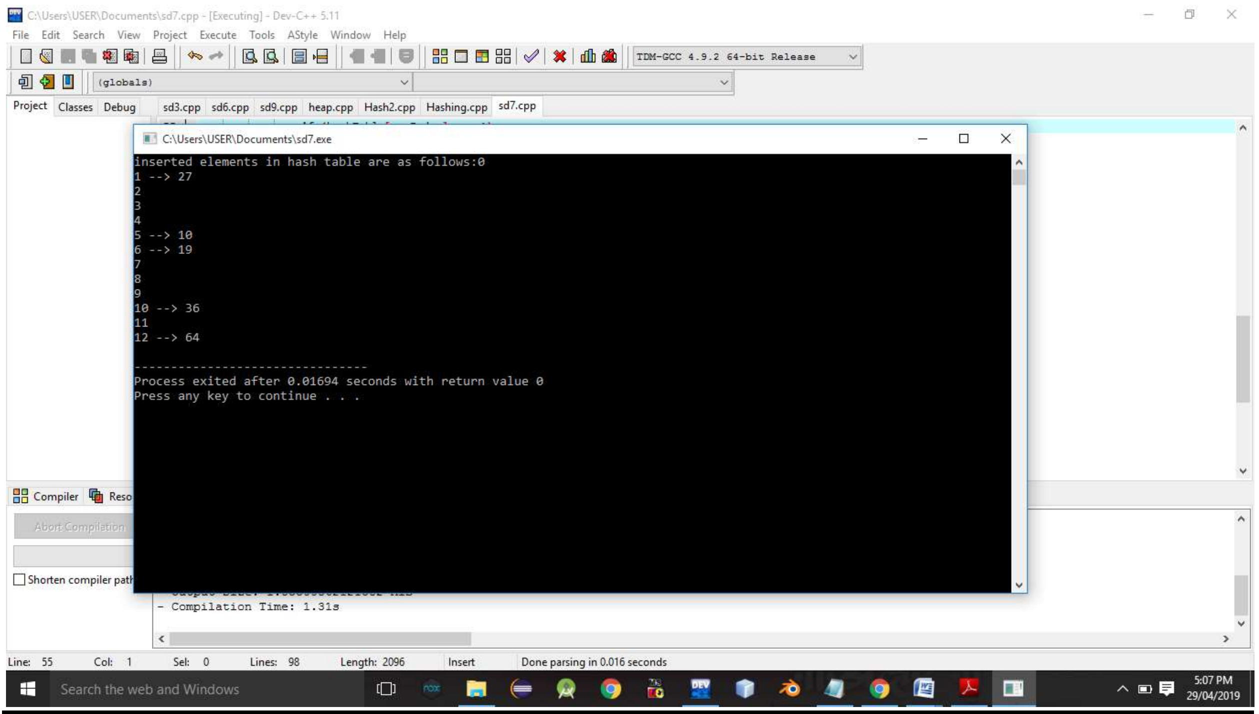
5. If collision occurs use the second hash function: hash2(key) = 1+(key mod (size1)).

6. Do: Hi(key)=((Hash(key) + i * hash2(key)) mod size) , using a for loop, for i from 1 to (size-1), untill the key gets mapped to it’s appropriate value. 7. Stop.

Program Code:-

```
#include <bits/stdc++.h>

using namespace std;


#define TABLE_SIZE 13


#define PRIME 7


class DoubleHash

{

   int *hashTable;

   int curr_size;


public:


   bool isFull()

   {
       return (curr_size == TABLE_SIZE);
   }
   int hash1(int key)
   {
       return (key % TABLE_SIZE);
   }

   int hash2(int key)
   {
       return (PRIME - (key % PRIME));
   }

   DoubleHash()
   {
       hashTable = new int[TABLE_SIZE];
       curr_size = 0;
       for (int i=0; i<TABLE_SIZE; i++)
           hashTable[i] = -1;
   }

   void insertHash(int key)
   {
       if (isFull())
           return;
       int index = hash1(key);


       if (hashTable[index] != -1)

       {

           int index2 = hash2(key);

           int i = 1;

           while (1)

           {

               int newIndex = (index + i * index2) %

                                       TABLE_SIZE;


               if (hashTable[newIndex] == -1)

               {

                   hashTable[newIndex] = key;

                   break;

               }

               i++;

           }

       }


       else

           hashTable[index] = key;

       curr_size++;

   }
   // function to display the hash table
   void displayHash()
   {
       for (int i = 0; i < TABLE_SIZE; i++)
       {
           if (hashTable[i] != -1)
               cout << i << " --> "
                    << hashTable[i] << endl;
           else
               cout << i << endl;
       }
   }
};

// Driver's code
int main()
{

   int a[] = {19, 27, 36, 10, 64};
   int n = sizeof(a)/sizeof(a[0]);
 cout<<"inserted elements in hash table are as follows:";
   // inserting keys into hash table
   DoubleHash h;
   for (int i = 0; i < n; i++)
       h.insertHash(a[i]);
   // display the hash Table

   h.displayHash();

   return 0;

}
```

Output Screenshots:-



```
inserted elements in hash table are as follows:0
1 --> 27
2
3
4
5 --> 10
6 --> 19
7
8
9
10 --> 36
11
12 --> 64

--------------------------------
Process exited after 0.01694 seconds with return value 0
Press any key to continue . . .
```

```
- Compilation Time: 1.31s
```

Conclusion:- Thus,we have studied double hashing and hashing technique.