ASSIGNMENT 7

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

OBJECTIVE: To study and learn the concepts of double hashing.

THEORY: **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:

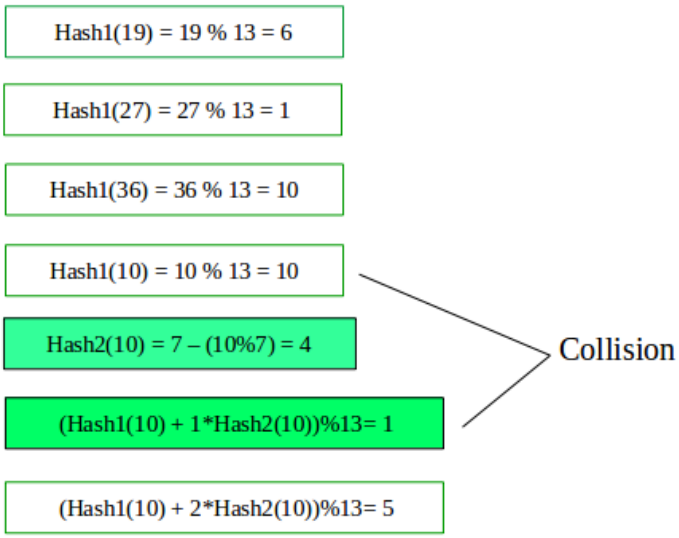
- It must never evaluate to zero
- Must make sure that all cells can be probed

ALGORITHM:

[Date]

Lets say, Hash1 (key) = key % 13

Hash2 (key) = 7 – (key % 7)



PROGRAM:

```
#include<iostream>
using namespace std;

const int size = 10;
int HashTable[size] = {0};

void addElement()
{
      int key,hash;
      bool isPlaced = false ;
      cout << "\tEnter the value to be inserted" << endl;
      cin >> key;

      hash = key%size ;
      if(HashTable[hash] == 0)
      {
            HashTable[hash] = key;
```

```
            isPlaced = true;
            return;
      }
      else if(HashTable[hash]!=0)
      {
            hash = 7 - (key%7);
            if(HashTable[hash]==0)
            {
                  HashTable[hash] = key;
                  isPlaced = true;
                  return;
            }
            else
            {
                  for(int i =0;i<10;i++)
                  {
                        hash = ((key%size) + i*(7 - (key%7)))%size;
                        if(HashTable[hash]==0)
                        {
                              HashTable[hash] = key;
                              isPlaced = true;
                              return;
                        }
                  }
            }
      }
      else
      {
            if(!isPlaced)
            {
                  cout << "\tArray is full" << endl;
            }
      }
```

```
}

void display()
{
      for(int i =0 ;i<10;i++)
      {
            cout << " " << HashTable[i];
      }
}

int main()
{
      int choice;
      char ch='y';
      int HashTable[size]={0};
      while(ch == 'y')
      {
            cout << "\tMENU"<< endl;
            cout << "\t1.Insert A Element" << endl;
            cout << "\t2.Display HashTable" << endl;
            cin >> choice;
            switch(choice)
            {
                  case 1:
                        addElement();
                        break;
                  case 2:
                        display();
                        break;
                  default:
                        cout << "\tINVALID CHOICE" << endl;
```

```
            }
            cout << "\tDo you wish to continue" << endl;
            cout << "\tenter y if yes" << endl;
            cin >> ch;
      }

}
```

OUTPUT:

```
jugal@ubuntu:~/17u183/sem2/SD$ g++ Hashing.cpp
jugal@ubuntu:~/17u183/sem2/SD$ ./a.out
        MENU
        1.Insert A Element
        2.Display HashTable
1
        Enter the value to be inserted
43
        Do you wish to continue
        enter y if yes
y
        MENU
        1.Insert A Element
        2.Display HashTable
1
        Enter the value to be inserted
56
        Do you wish to continue
        enter y if yes
y
        MENU
        1.Insert A Element
        2.Display HashTable
2
 0 0 0 43 0 0 56 0 0 0  Do you wish to continue
        enter y if yes
```

CONCLUSION:

We successfully implemented open addressing using double hashing.