

ASSIGNMENT 7

AIM: Insert the keys into a hash table of length m using open addressing using double hashing with $h(k)=(1+k\text{mod}(m-1))$.

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:

$$(\text{hash1}(\text{key}) + i * \text{hash2}(\text{key})) \% \text{TABLE_SIZE}$$

Here $\text{hash1}()$ and $\text{hash2}()$ are hash functions and TABLE_SIZE is size of hash table.

(We repeat by increasing i when collision occurs)

First hash function is typically $\text{hash1}(\text{key}) = \text{key} \% \text{TABLE_SIZE}$

A popular second hash function is:

$\text{hash2}(\text{key}) = \text{PRIME} - (\text{key} \% \text{PRIME})$ 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:

Lets say, **$\text{Hash1}(\text{key}) = \text{key} \% 13$**

$\text{Hash2}(\text{key}) = 7 - (\text{key} \% 7)$

$$\text{Hash1}(19) = 19 \% 13 = 6$$

$$\text{Hash1}(27) = 27 \% 13 = 1$$

$$\text{Hash1}(36) = 36 \% 13 = 10$$

$$\text{Hash1}(10) = 10 \% 13 = 10$$

$$\text{Hash2}(10) = 7 - (10 \% 7) = 4$$

$$(\text{Hash1}(10) + 1 * \text{Hash2}(10)) \% 13 = 1$$

$$(\text{Hash1}(10) + 2 * \text{Hash2}(10)) \% 13 = 5$$

Collision

PROGRAM:

```
#include <iostream>
using namespace std;
class dr
{
    int n=10;
    int arr[100][3];
    int c;
public:
    dr()
    {
        cout<<"Table of size "<<n<<" created\n";
        for(int i=0;i<n;i++)
        {
            arr[i][0]=0;
            arr[i][1]=-1;
            arr[i][2]=-1;
        }
        c=0;
    }
    void add(int,int);
    int find_key(int);
    void display();
    void update_val(int,int);
};

void dr::add(int key,int value)
{
    int new_hash_addr1,new_hash_addr2,main_hash_addr=-1,j=0;
    if(this->find_key(key)!=-1)
    {
        cout<<"Key already exists\n";
        return;
    }
    if(c==(n-1))
    {
        cout<<"Table full, request denied\n";
    }
    new_hash_addr1=(key)%n;
    new_hash_addr1=1+(key%(n-1));
    if(arr[new_hash_addr1][1]==-1)
    {
        arr[new_hash_addr1][0]=key;
        arr[new_hash_addr1][1]=value;
    }
    else if(arr[new_hash_addr2][1]==-1)
    {
        arr[new_hash_addr2][0]=key;
        arr[new_hash_addr2][1]=value;
    }
    else
    {
        while(arr[new_hash_addr2][2]!=-1)
```

```

{
    main_hash_addr=new_hash_addr2;
    new_hash_addr2=arr[main_hash_addr][2];
}
main_hash_addr=new_hash_addr2;
for(int i=0;i<n;i++)
{
    new_hash_addr2=(main_hash_addr+i)%n;
    if(arr[new_hash_addr2][1]==-1)
    {
        arr[new_hash_addr2][0]=key;
        arr[new_hash_addr2][1]=value;
        arr[main_hash_addr][2]=new_hash_addr2;
        c++;
        break;
    }
}
}
}
void dr::display()
{
    cout<<"Key\t\tValue\t\tChain\n";
    for(int i=0;i<n;i++)
    {
        cout<<arr[i][0]<<"\t\t"<<arr[i][1]<<"\t\t"<<arr[i][2]<<endl;
    }
}
int dr::find_key(int key)
{
    int search_addr=key%n,f=0;
    while(arr[search_addr][0]!=key && arr[search_addr][2]!=-1)
    {
        search_addr=arr[search_addr][2];
    }
    if(arr[search_addr][0]==key)
    {
        return arr[search_addr][1];
    }
    else if(arr[search_addr][2]==-1)
    {
        return -1;
    }
}
int main()
{
    char r;
    do
    {
        char op;
        dr table;
        int c;
        do

```

```

{
    cout<<"-----Menu-----\n";
    cout<<"1] Insert value\n2] Display\n";
    cout<<"_____ \n";
    cout<<"Enter your choice: ";
    cin>>c;
    switch(c)
    {
        case 1: {
            int key,val;
            cout<<"Enter key: ";
            cin>>key;
            cout<<"Enter value: ";
            cin>>val;
            table.add(key,val);
        }
        break;
        case 2: table.display();
        break;
        default:cout<<"Invalid\n";
    }
    cout<<"\nDo you wish to go again? ";
    cin>>op;
}while(op=='y' || op=='Y');
cout << "Test pass?(y/n): " << endl;
cin>>r;
}while(r=='n' || r=='N');
cout<<"*****\n";
cout<<"*   Thank You!   *\n";
cout<<"*****\n";
return 0;
}

```

OUTPUT:

```
"C:\Users\USER\Downloads\BinaryTree (1).exe"
Table of size 10 created
-----Menu-----
1] Insert value
2] Display

Enter your choice: 1
Enter key: 1
Enter value: 12

Do you wish to go again? y
-----Menu-----
1] Insert value
2] Display

Enter your choice: 1
Enter key: 5
Enter value: 34

Do you wish to go again? y
-----Menu-----
1] Insert value
2] Display

Enter your choice: 3
Invalid

Do you wish to go again? y
-----Menu-----
1] Insert value
2] Display

Enter your choice: 1
Enter key: 3
Enter value: 45

Do you wish to go again? y
-----Menu-----
1] Insert value
2] Display

Enter your choice: 2
Key      Value      Chain
0        -1         -1
0        -1         -1
1        12         -1
0        -1         -1
3        45         -1
0        -1         -1
5        34         -1
0        -1         -1
```

CONCLUSION: We successfully implemented open addressing using double hashing.