DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJ19ITL502 **DATE:3/12/22**
COURSE NAME: Advanced Data Structures Laboratory **CLASS/Div: A**
SAP ID:60003200076 **Academic Year 2022-23**

EXPERIMENT NO. 7

CO/LO: LO1

AIM: To Implement Perfect hashing

DESCRIPTION OF EXPERIMENT:

We say a hash function is perfect for S if all lookups involve O(1) work

This is used when the keys stored in the hash table are expected to be static. In this case perfect hashing guarantees excellent average as well as worst-case performance. This is how real dictionaries work. You most of the time just need to read from a dictionary.

This hashing technique is called perfect hashing because it takes constant time, O(1) to search for an item in the table.

Perfect hashing is implemented using two hash tables, one at each level. Each of the table uses universal hashing. The first level is the same a hashing with chaining such that n elements is hashed into m slots in the hash table. This is done using a has function selected from a universal family of hash functions.

The second level of uses a second hash table (instead of a linked list used in chaining). Elements that hash to the same slot j in the first hash table are stored in a second hash table. This second hash table is known as secondary hash table . The hash function h_j is carefully chosen such that there are no collision in the secondary table.

To ensure there are no collisions in the secondary hash table S_j, we need to make the size m_j of the secondary table equal to the square of the number of keys hashing into slot j in the first table. That is:

$m_j = n_j^2$

The hash functions for the primary hash table is carefully chosen so that we limit the expected total amount of space used to be O(n)

Technology stack used: Python

Program:

```
import random

m=5
p=101
keys=[]
for i in range(0,m):
    keys.append(random.randint(1,2000))
collision_frequency={}
d={}
A=random.randint(1,2000)
B=random.randint(1,2000)

def gethash(a,b,p,e,m):
    #print(a,b,p,e,m)
    #print(((a*e+b)%p)%m)
    return ((a*e+b)%p)%m



for i in keys:
    h=gethash(A,B,p,i,m)
    try:
        collision_frequency[h]=collision_frequency[h]+1
    except:
        collision_frequency[h]=1

print(collision_frequency)
for i in collision_frequency.keys():
    M=collision_frequency[i]

    size=M*M
    l=[]
    for k in range(0,size+3):
        l.append(None)
    if size==1:
        a=0
        b=0
    else:
        a=random.randint(1,2000)
        b=random.randint(1,2000)
    l[0]=a
    l[1]=b
    l[2]=M
```

```
    d[i]=l
#print(d)
for i in keys:
    h=gethash(A,B,p,i,m)
    a=d[h]
    h1=gethash(a[0],a[1],p,i,a[2]*a[2])
    print(i,h,h1+3)
    #print(h1)
    d[h][h1+3]=i

print(d)
```

Output:

```
PS C:\Users\SHREE RAM\Desktop\ads> python -u "c:\Users\SHREE RAM\Desktop\ads\try.py"
{4: 2, 2: 2, 3: 1}
28 4 4
1490 4 6
650 2 3
1354 3 3
602 2 4
{4: [1208, 371, 2, None, 28, None, 1490], 2: [449, 294, 2, 650, 602, None, None], 3: [0, 0, 1, 1354]}
PS C:\Users\SHREE RAM\Desktop\ads>
```

APPLICATIONS:

Some application of perfect hashing includes:

- data storage on a CD ROM
- set of reserved words in a programming language

Conclusion:

Thus we have implemented perfect hashing.