K. J. Somaiya College of Engineering, Mumbai-77
(A constituent College of Somaiya Vidyavihar University)

Batch: B2	Roll. No.: 110
Experiment:	
Grade: AA / AB / BB / BC / CC / CD /DD	

Title: Implementation of hashing concept

Objective: To understand various hashing methods

Expected Outcome of Experiment:

CO	Outcome
CO4	Demonstrate sorting and searching methods.

Websites/books referred:

1. https://www.tutorialspoint.com/state-the-advantages-and-disadvantages-of-collision-resolution-strategies
2. https://github.com/Aatmaj-Zephyr/Data-Structures/tree/main/Lab/Day%208
3. https://www.techtarget.com/searchdatamanagement/definition/hashing#:~:text=Hashing%20is%20the%20process%20of,the%20implementation%20of%20hash%20tables.

Abstract: -

(Define Hashing ,hash function, list collision handling methods)

Hashing is the process of transforming any given key or a string of characters into another value. This is usually represented by a shorter, fixed-length value or key that represents and makes it easier to find or employ the original string. 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 usually used to index a fixed-size table called a hash table.What are the methods for collision handling in hashing?

Collision Resolution Techniques

1) Linear Probing.

2) Quadratic Probing.

3) Double Hashing.

Example:

20, 33, 65, 23, 11, 32, 78, 64, 3, 87, 10, 7

Linear - 20,11,32,33,64,65,3,87,78,10,7,0,0,0,0,0,

Quadratic - 20,11,32,33,64,65,0,87,78,7,10,0,0,3,0,0,

Chaining - 20,10 0 0 33,23,3 64 65 0 87,7 78 0 0 0 0 0 0 0 0 0

Code and output screenshots:

```
/**************************************************************************
**
Hashing

**************************************************************************
**/
#include <iostream>
const int len=7;
using namespace std;
class hashtable{
   public: int arr[len];
   public: int collision = 0;
   public: void print(){
      for(int i=0;i<len;i++){
```

```
        cout<<arr[i]<<",";
    }
}
public: void addquad(int a){
    int temp =hash(a);
    if(arr[temp]!=0){
        collision++;
        temp++;
        //overflow not considered
          for(int i=0;i<len;i++){
              if(arr[(i*i+temp)%len]==0){
               arr[(i*i+temp)%len]=a;
               break;
              }
           }

        }
        else{
           arr[temp]=a;
        }
    }
public: void addlinear(int a){
    int temp =hash(a);
    if(arr[temp]!=0){
        collision++;
```

```
    temp++;
    //overflow not considered
      for(int i=0;i<len;i++){
         if(arr[(i+temp)%len]==0){
          arr[(i+temp)%len]=a;
          break;
         }
       }

    }
    else{
       arr[temp]=a;
    }
  }
public: bool searchlinear(int a){
   int temp =hash(a);
  if(arr[temp]!=a){

    for(int i=0;i<len;i++){
       if(arr[(i+temp)%len]==a){
          return true;
       }

    }
  return false;
```

```
    }
    else{
        return true;
    }
}
public: bool searchquad(int a){
     int temp =hash(a);
    if(arr[temp]!=a){

        for(int i=0;i<len;i++){
            if(arr[(i*i+temp)%len]==a){
                return true;
            }

        }
    return false;
    }
    else{
        return true;
    }
}
public: int hash(int a){
return a%10;

}
```

```
};
int main()
{
   hashtable *linear = new hashtable();
   hashtable *quad = new hashtable();
   linear->addlinear(1002);
   linear->addlinear(12);
   linear->addlinear(212);
   linear->addlinear(123);
   linear->addlinear(125);
   cout<<linear->searchlinear(1002)<<"\n";
   cout<<linear->searchlinear(100)<<"\n";
   cout<<linear->searchlinear(123)<<"\n";
   linear->print();
   cout<<"\n"<<linear->collision<<"\n";

   quad->addquad(1002);
   quad->addquad(12);
   quad->addquad(212);
   quad->addquad(123);
   quad->addquad(125);
   cout<<quad->searchquad(1002)<<"\n";
   cout<<quad->searchquad(100)<<"\n";
   cout<<quad->searchquad(123)<<"\n";
   quad->print();
```

```
    cout<<"\n"<<linear->collision<<"\n";

    cout<<"Hello World";


    return 0;

}
```

1

0

1

0,0,1002,12,212,123,125,

4

1

0

1

0,0,1002,12,212,123,125,

4

Hello World

Conclusion: -
Number of collisions for linear hashing - 4.

Number of collisions for Quadratic hashing - 4

In this experiment we understood what hashing is and how to implement it. We tried both ways of resolution of collisions, that is the linear method and quadratic method.

Post lab questions-

a. **Compare and contrast various collision handling methods.**

Linear Probing

Linear probing is a simple collision resolution technique for resolving collisions in hash tables, data structures for maintaining collection of values in a hash table. If there is a collision for the position of the key value then the linear probing technique assigns the next free space to the value.

The advantages of linear probing are as follows −

- Linear probing requires very less memory.
- It is less complex and is simpler to implement.

The disadvantages of linear probing are as follows −

- Linear probing causes a scenario called "primary clustering" in which there are large blocks of occupied cells within the hash table.
- The values in linear probing tend to cluster which makes the probe sequence longer and lengthier.

Quadratic probing

Quadratic probing also is a collision resolution mechanism which takes in the initial hash which is generated by the hashing function and goes on adding a successive value of an arbitrary quadratic polynomial from a function generated until an open slot is found in which a value is placed.

The advantages of quadratic probing is as follows −

- Quadratic probing is less likely to have the problem of primary clustering and is easier to implement than Double Hashing.

The disadvantages of quadratic probing are as follows −

- Quadratic probing has secondary clustering. This occurs when 2 keys hash to the same location, they have the same probe sequence. So, it may take many attempts before an insertion is being made.
- Also probe sequences do not probe all locations in the table.

Double hashing

Double hashing is also a collision resolution technique when two different values to be searched for produce the same hash key.

It uses one hash value generated by the hash function as the starting point and then increments the position by an interval which is decided using a second, independent hash function. Thus here there are 2 different hash functions.

The advantage of double hashing is as follows −

- Double hashing finally overcomes the problems of the clustering issue.

The disadvantages of double hashing are as follows:

1) Double hashing is more difficult to implement than any other.

2) Double hashing can cause thrashing.

Source - **https://www.tutorialspoint.com/state-the-advantages-and-disadvantages-of-collision-resolution-strategies**

b. Store the given numbers in bucket of size 16, resolve the collisions if any with
   a. Linear probing
   b. Quadratic hashing
   c. Chaining

20, 33, 65, 23, 11, 32, 78, 64, 3, 87, 10, 7

Linear - 20,11,32,33,64,65,3,87,78,10,7,0,0,0,0,0,

Quadratic - 20,11,32,33,64,65,0,87,78,7,10,0,0,3,0,0,

Chaining - 20,10 0 0 33,23,3 64 65 0 87,7 78 0 0 0 0 0 0 0 0 0