**Practical - 3**

**Aim:** Implementation of different hashing techniques.

**Description:**

1. **Modulo Hashing with Linear Probe**

This is the most simple and most straightforward method to generate a hash value. The hash function divides the value k by M and then uses the remainder obtained.

Formula:

h(K) = k mod M

Here,

k is the key value, and

M is the size of the hash table.

It is best suited that M is a prime number as that can make sure the keys are more uniformly distributed. The hash function is dependent upon the remainder of a division. sometimes we get a key where an element already is present. We re-implement it using a linear probe that is the next empty index assigned to it for storing.

Code:

#include <iostream>

#include <stdlib.h>

int \*arr, size, capacity = 0;

void setSize(){

arr = (int\*)malloc(sizeof(int)\*size);

std::cout << "Size Allocated!" << std::endl;

return;

}

// to get linear index which is empty

int linearProbe(int index, int data){

int i, k;

for(i = 0; i < size; i++){

k = (index+i)%size;

if(arr[k] == 0){

capacity++;

break;

}else if(arr[k] == data){

break;

}

}

return k;

}

void hashingWithLinearProbe(int data){

int index =data % size;

if(arr[index] == 0){

capacity++;

arr[index] = data;

}else if(arr[index] == data){

return;

}else{

arr[linearProbe(index+1, data)] = data;

}

}

void hashingWithLinearProbeSearch(int data){

int i, k;

for(i = 0; i < size; i++){

k = ((data % size) + i) % size;

if(arr[k] == data)

break;

}

if(i != size)

std::cout<<data << " found at index "<< k << std::endl;

else

std::cout<<data << " element not found!"<< std::endl;

return;

}

int main() {

int data;

std::cout << "Enter Size of Array : ";

std::cin >> size;

setSize();

for(int i = 0; i < size; i++){

arr[i] = 0;

}

while(capacity < size){

std::cout << "Enter elements : ";

std::cin >> data;

hashingWithLinearProbe(data);

}

std::cout << "Array element stored using hashing element are : ";

for(int i = 0; i < size; i++){

std::cout << " " << arr[i];

}

std::cout << std::endl;

std::cout << "Enter element to find : ";

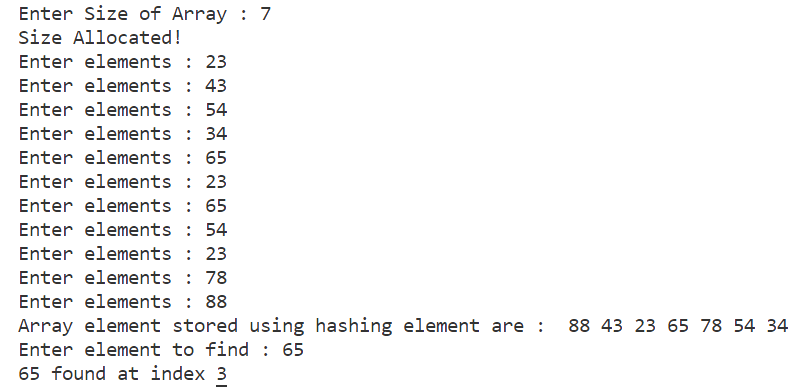
std::cin >> data;

hashingWithLinearProbeSearch(data);

return 0;

}

Output:



1. **Mid-square Hashing Method**

Description:

Mid-Square hashing is a hashing technique in which unique keys are generated. In this technique, a seed value is taken and it is squared. Then, some digits from the middle are extracted. These extracted digits form a number which is taken as the new seed. This technique can generate keys with high randomness if a big enough seed value is taken. However, it has a limitation. As the seed is squared, if a 6-digit number is taken, then the square will have 12-digits. This exceeds the range of int data types. So, overflow must be taken care of. In case of overflow, use long long int data type or use string as multiplication if overflow still occurs. The chances of a collision in mid-square hashing are low, not obsolete. So, in the chances, if a collision occurs, it is handled using some hash map

Code:

#include <iostream>

using namespace std;

//Array Bucket

int arr[100] = {0};

int size = 100;

int getHashCode(long long data){

if(data < 100){

return data;

}

long long k = data \* data;

string n = to\_string(k);

int spliter = n.length()/4;

n = n.substr(spliter, spliter\*2);

return getHashCode(stoi(n));

}

int linearProbe(int index, int data){

int i, k = -1;

for(i = 0; i < size; i++){

k = (index+i)%size;

if(arr[k] == 0){

break;

}

else if(arr[k] == data){

break;

}

}

return k;

}

void midSquareHashing(int data){

int index, k = data;

index = getHashCode(k);

if(arr[index] == 0);

else if(arr[index] == data)

return;

else

index = linearProbe(index, data);

if(index == -1)

printf("Bucket is full\n");

else{

arr[index] = data;

printf("%d added at index %d in bucket\n", data, index);

}

}

void searchElement(int data){

int index = getHashCode(data);

if(arr[index] == data){

printf("%d found at index %d in bucket\n", data, index);

}else{

index = linearProbe(index, data);

if(index != -1)

printf("%d found at index %d in bucket\n", data, index);

else

printf("%d not found at any index in bucket\n", data);

}

}

int main(int argc, char const \*argv[])

{

int data[] = {7644,379, 5487, 3456, 3234, 4534, 3234, 99, 99};

for(int i = 0; i < 9; i++)

midSquareHashing(data[i]);

searchElement(5487);

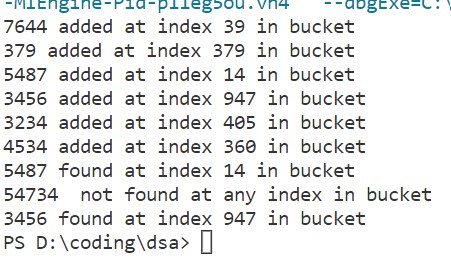
searchElement(54734);

searchElement(3456);

return 0;

}

Output:



**Conclusion:** Hashing implemented using division and mid square method successfully.