**Batch: C3 Roll No.: 16010123217**

**Experiment / assignment / tutorial No. 10**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:**  Implementation of Hashing  - Linear and quadratic hashing |

**Objective:** To Understand and Implement Linear and Quadratic Hashing

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| 4 | Demonstrate sorting and searching methods. |

**Books/ Journals/ Websites referred:**

1. *Fundamentals Of Data Structures In C –* Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications –* Jean Paul Tremblay,

Paul G. Sorenson

1. *Data Structures A Pseudo Approach with C –* Richard F. Gilberg & Behrouz A. Forouzan
2. *https://www.tutorialspoint.com/state-the-advantages-and-disadvantages-of-collision-resolution-strategies*

**Abstract**:

Linear and quadratic hashing are two methods used in hash table implementations to manage collisions—situations where two keys hash to the same index in a table. These techniques are crucial for ensuring efficient data retrieval, storage, and overall performance of hash-based data structures.

### Linear Hashing

In linear hashing, when a collision occurs, the algorithm searches for the next available slot in a sequential manner. This means that if a key hashes to a position that is already occupied, the algorithm checks the next index, and continues this process until an empty slot is found. This approach is simple and easy to implement, but it can lead to a phenomenon known as "clustering," where groups of filled slots form. This clustering can degrade performance, especially as the load factor (the ratio of filled slots to total slots) increases, resulting in longer search times.

### Quadratic Hashing

Quadratic hashing offers a solution to the clustering problem associated with linear hashing. Instead of searching for the next available slot linearly, quadratic hashing uses a quadratic function to probe for an open slot. For instance, if a collision occurs at index h(k)h(k)h(k), the algorithm will check h(k)+12,h(k)+22,h(k)+32h(k) + 1^2, h(k) + 2^2, h(k) + 3^2h(k)+12,h(k)+22,h(k)+32, and so on, effectively spreading out the probe sequence. This reduces the chances of clustering and generally leads to better performance, particularly in scenarios with higher load factors.

**Algorithm for assigned hashing method:**

Linear Hashing

1. Start
2. Take the size of Hash Table from user and number of elements to inserted

from user

1. Declare a vector of size of Hash table and assign -1 to all index
2. Take the elements using for loop
3. While(hash[idx] != -1){

index = (index + 1)%size of Hash Table;

}

hash[index] = x;

1. Print the hash Table
2. Exit

**Program:**

**Linear Hashing**

#include <bits/stdc++.h>

using namespace std;

#define int long long

#define endl "\n"

const int MOD = 1e9 + 7;

const int INF = LLONG\_MAX >> 1;

signed main(){

    cout<<"Enter the size of the hash table: ";

    int n; cin>>n;

    if (n <= 0) {

        cout << "Hash table size must be greater than 0!" << endl;

        return 0;

    }

    vector<int> hash(n, -1);

    cout<<"Enter the number of elements to be inserted: ";

    int m; cin>>m;

    if (m > n) {

        cout << "Number of elements exceeds the size of the hash table!" << endl;

        return 0;

    }

    cout<<"Enter the elements: ";

    for(int i=0; i<m; i++){

        int x; cin>>x;

        int index = x % n;

        while(hash[index] != -1){

            index = (index + 1) % n;

        }

        hash[index] = x;

    }

    cout<<"Hash Table: ";

    for(int i=0; i<n; i++){

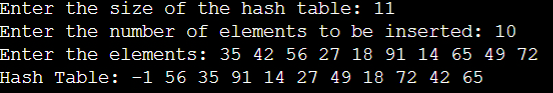
        cout<<hash[i]<<" ";

    }

    return 0;

}

**Output Screenshots:**

****

**Conclusion:-**

From this experiment we learnt about hashing techniques : Linear and Quadratic Hashing and the difference between them.

**Post Lab Questions:**

1. Explain how linear hashing resolves collisions. What are the potential drawbacks of this method?

Ans. Linear Hashing is a collision resolution technique used in hash tables to handle collisions that occur when two keys map to the same index.

Procedure to resolve collision

1. When a collision occurs, the hash table searches for the next available slot.

2. The search starts from the collided index and moves linearly through the table, probing each slot.

3. If an empty slot is found, the key-value pair is inserted.

4. If the table is full, the hash table is resized, and the process repeats.

Drawbacks

* 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.

1. Describe the probing sequence used in quadratic hashing. How does this sequence differ from that of linear hashing?

Ans. Quadratic Hashing uses a probing sequence that probes the hash table at indices determined by a quadratic function.

Quadratic Hashing Probing Sequence:

1. Initial index: h(k) = k mod m (where k is the key, m is the table size)

2. Probing sequence: h(k, i) = (h(k) + i^2) mod m (where i is the probe number, starting from 1)

Key differences from Linear Hashing:

1. Probe step: Quadratic Hashing uses i^2 instead of i (linear probing).

2. Probe distribution: Quadratic probing spreads probes more evenly throughout the table, reducing clustering.

3. Probe pattern: Quadratic probing has a parabolic pattern, whereas linear probing has a linear pattern.

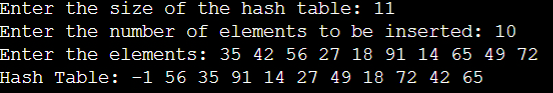
1. Consider a hash table of size 11 and the following sequence of 10 elements to insert:

35, 42, 56, 27, 18, 91, 14, 65, 49, 72

* 1. Insert all 10 elements into the hash table using Linear Probing.
  2. Insert the same elements into the hash table using Quadratic Probing.

Assume your own hash function(s).

a)



b)

