**Assignment No. 07**

Name:- Anirudha Shripad Kale.

Roll No. :- 223029

PRN No :- 22210288

Batch: C1

**Problem Statement:** Implement Hash table and perform collision resolution using Linear Probing method.

**Objectives:**

1. To understand the concept of Hash Table.
2. To study the different type of collision resolution methods.
3. To implement the linear probing method.
4. To understand how hash table is good over others methods to store the key and value.

**Theory:**

Hash Table:

A hash table is a data structure that stores key-value pairs. It uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found. Hash tables offer efficient insertion, deletion, and retrieval operations in average constant time complexity (O(1)), making them suitable for scenarios where fast access to data is crucial.

Components of a Hash Table:

Hash Function: A hash function maps keys to indices in the hash table's array. It should distribute keys uniformly across the array to minimize collisions.

Array of Buckets: The hash table consists of an array of buckets or slots, where each bucket stores key-value pairs.

Collision Resolution Strategy: Collisions occur when two different keys map to the same index. A collision resolution strategy is used to handle these collisions efficiently.

Collision Resolution Methods:

Separate Chaining:

In separate chaining, each bucket in the hash table maintains a linked list of key-value pairs. When a collision occurs, the new key-value pair is appended to the linked list at the corresponding bucket.

This method ensures that all key-value pairs with the same hash value are stored together. It requires additional memory for maintaining linked lists but can handle a large number of collisions efficiently.

Open Addressing:

Open addressing resolves collisions by probing the hash table for an empty slot when a collision occurs. Various probing techniques can be used, such as linear probing, quadratic probing, and double hashing.

Linear Probing: If a collision occurs at index 𝑖, the algorithm searches for the next available slot by probing 𝑖 + 1, 𝑖 + 2, 𝑖 + 3, and so on, wrapping around to the beginning if necessary.

Quadratic Probing: The algorithm probes at positions that are quadratic offsets away from the original hash index.

Double Hashing: The algorithm uses a secondary hash function to determine the step size for probing.

Open addressing reduces memory overhead compared to separate chaining but may suffer from clustering and degraded performance under high load factors.

**Algorithm:**

1. Initialization:

Initialize arrays keys and values to store keys (roll numbers) and values (names) respectively.

Set the size of the hash table.

Initialize size counter.

1. Hash Function:

Implement a hash function hash(key) to compute the index for a given key.

1. Insertion:

Implement insertPair(key, value) method:

Compute the hash index for the key.

Handle collisions using linear probing until an empty slot is found.

Insert the key-value pair into the hash table.

1. Removal:

Implement removePair(key) method:

Compute the hash index for the key.

Search for the key using linear probing.

If found, mark the slot as empty by setting key to -1 and value to null.

1. Search:

Implement searchPair(key) method:

Compute the hash index for the key.

Search for the key using linear probing.

Print whether the key is present or absent.

1. Print Hash Table:

Implement printHashTable() method to print the contents of the hash table.

1. Main Function:

Initialize a Scanner object to take user input.

Create an instance of the HashContents class.

Prompt the user to enter the size of the hash table.

Enter a loop to allow user interaction:

Display menu options for operations: insert, remove, search, display, and exit.

Based on user input, call corresponding methods of the HashContents class.

Terminate the loop when the user chooses to exit.

1. Input/Output:

Prompt the user for input when necessary (e.g., roll number, name, operation choice).

Display appropriate messages for successful or failed operations.

Print the contents of the hash table as requested by the user.

**Input:**

import java.util.\*;

class HashContents {

    // implementing the hash table to store the roll no and the name of the students

    // in department

    // 2 or more students may have same roll no. from different division

    int[] keys; // for roll no

    String[] values; // for name

    int size;

    HashContents(int maxSize) {

        keys = new int[maxSize];

        values = new String[maxSize];

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

            keys[i] = -1;

            values[i] = null;

        }

        int size = 0;

    }

    public int hash(int key) {

        return key % keys.length;

    }

    public void insertPair(int key, String value) {

        if (size == keys.length)

            System.out.println("Hash Table is full\nNot able to insert the pair");

        int index = hash(key);

        while (keys[index] != -1) { // util the next empty space get

            index = (index + 1) % keys.length;

        }

        keys[index] = key;

        values[index] = value;

        size++;

        System.out.println("Pair get inserted successfully !!");

    }

    public void removePair(int key) {

        if (size == 0)

            System.out.println("Hash Table is empty\nCann't remove the pair");

        int index = hash(key);

        while (keys[index] != key && keys[index] != -1) {

            index = (index + 1) % keys.length;

        }

        if (keys[index] == key) {

            keys[index] = -1;

            values[index] = null;

            size--;

            System.out.println("Pair get deleted successfully !!");

        } else {

            System.out.println("Key not present cann't remove it");

        }

    }

    public void searchPair(int key) {

        if (size == 0)

            System.out.println("Empty Hash Table");

        int index = hash(key);

        while (keys[index] != key && keys[index] != -1) {

            index = (index + 1) % keys.length;

        }

        if (keys[index] == key)

            System.out.println("Roll No. " + key + " is present in the class.");

        else

            System.out.println("Roll No. " + key + " is absent.");

    }

    // printing hashTable

    public void printHashTable() {

        for (int i = 0; i < keys.length; i++) {

            System.out.println("Roll No. : " + keys[i] + " Name : " + values[i]);

        }

    }

}

public class Assignment7 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println();

        System.out.println("\t\*\*\*\*\* Hash Table Implementation Using Linear Probing \*\*\*\*\*\t");

        System.out.println("\t--- Hash Table to store the roll no. and name of the student--- \t");

        boolean x = true;

        System.out.println("----------------------------------------------");

        System.out.print("First enter the size of the hash table: ");

        int size = sc.nextInt();

        System.out.println("----------------------------------------------");

        HashContents table1 = new HashContents(size);

        while (x) {

            System.out.println("\t \*\*\* Operations Available \*\*\*\t");

            System.out.println(

                    "1. Insert the pair.\n2. Remove the pair.\n3. Search the pair.\n4. Display Hash Table.\n5. Exit");

            System.out.println("----------------------------------------------");

            System.out.print("Enter your choice : ");

            int choice = sc.nextInt();

            switch (choice) {

                case 1:

                    System.out.print("Enter the roll no. : ");

                    int roll = sc.nextInt();

                    System.out.println("Enter the name : ");

                    String name = sc.next();

                    table1.insertPair(roll, name);

                    System.out.println("----------------------------------------------");

                    break;

                case 2:

                    System.out.print("Enter the roll no. which you want to delete : ");

                    int roll1 = sc.nextInt();

                    table1.removePair(roll1);

                    System.out.println("----------------------------------------------");

                    break;

                case 3:

                    System.out.print("Enter the roll no. which you want to search : ");

                    int roll2 = sc.nextInt();

                    table1.searchPair(roll2);

                    System.out.println("----------------------------------------------");

                    break;

                case 4:

                    table1.printHashTable();

                    System.out.println("----------------------------------------------");

                    break;

                case 5:

                    x = false;

                    System.out.println("----------------------------------------------");

                    break;

            }

        }

    }

}

**Output:**

\*\*\*\*\* Hash Table Implementation Using Linear Probing \*\*\*\*\*

--- Hash Table to store the roll no. and name of the student---

----------------------------------------------

First enter the size of the hash table: 5

----------------------------------------------

\*\*\* Operations Available \*\*\*

1. Insert the pair.

2. Remove the pair.

3. Search the pair.

4. Display Hash Table.

5. Exit

----------------------------------------------

Enter your choice : 1

Enter the roll no. : 29

Enter the name :

Anirudha

Pair get inserted successfully !!

----------------------------------------------

\*\*\* Operations Available \*\*\*

1. Insert the pair.

2. Remove the pair.

3. Search the pair.

4. Display Hash Table.

5. Exit

----------------------------------------------

Enter your choice : 1

Enter the roll no. : 16

Enter the name :

Yash

Pair get inserted successfully !!

----------------------------------------------

\*\*\* Operations Available \*\*\*

1. Insert the pair.

2. Remove the pair.

3. Search the pair.

4. Display Hash Table.

5. Exit

----------------------------------------------

Enter your choice : 1

Enter the roll no. : 11

Enter the name :

Sanchit

Pair get inserted successfully !!

----------------------------------------------

\*\*\* Operations Available \*\*\*

1. Insert the pair.

2. Remove the pair.

3. Search the pair.

4. Display Hash Table.

5. Exit

----------------------------------------------

Enter your choice : 1

Enter the roll no. : 25

Enter the name :

Abhishek

Pair get inserted successfully !!

----------------------------------------------

\*\*\* Operations Available \*\*\*

1. Insert the pair.

2. Remove the pair.

3. Search the pair.

4. Display Hash Table.

5. Exit

----------------------------------------------

Enter your choice : 100

\*\*\* Operations Available \*\*\*

1. Insert the pair.

2. Remove the pair.

3. Search the pair.

4. Display Hash Table.

5. Exit

----------------------------------------------

Enter your choice : 1

Enter the roll no. : 100

Enter the name :

ABC

Pair get inserted successfully !!

----------------------------------------------

\*\*\* Operations Available \*\*\*

1. Insert the pair.

2. Remove the pair.

3. Search the pair.

4. Display Hash Table.

5. Exit

----------------------------------------------

Enter your choice : 2

Enter the roll no. which you want to delete : 100

Pair get deleted successfully !!

----------------------------------------------

\*\*\* Operations Available \*\*\*

1. Insert the pair.

2. Remove the pair.

3. Search the pair.

4. Display Hash Table.

5. Exit

----------------------------------------------

Enter your choice : 3

Enter the roll no. which you want to search : 29

Roll No. 29 is present in the class.

----------------------------------------------

\*\*\* Operations Available \*\*\*

1. Insert the pair.

2. Remove the pair.

3. Search the pair.

4. Display Hash Table.

5. Exit

----------------------------------------------

Enter your choice : 4

Roll No. : 25 Name : Abhishek

Roll No. : 16 Name : Yash

Roll No. : 11 Name : Sanchit

Roll No. : -1 Name : null

Roll No. : 29 Name : Anirudha

----------------------------------------------

\*\*\* Operations Available \*\*\*

1. Insert the pair.

2. Remove the pair.

3. Search the pair.

4. Display Hash Table.

5. Exit

----------------------------------------------

Enter your choice : 5

----------------------------------------------