**Practical 2 – Advanced Data Structures Lab**

**1. Write a java program to hash various keys using Modulo Division hashing methods and use linear probe for collision resolution. Program:**

**HashTable.java**

package mypack;

import java.util.Arrays;

public class HashTable {

private static final int SIZE = 10;

private int[] hashTable;

// Constructor to initialize hash table

public HashTable() {

hashTable = new int[SIZE];

Arrays.fill(hashTable, -1); // -1 indicates an empty slot for probing

}

// Insert key using Linear Probing

public void insertLinearProbing(int key) {

int index = key % SIZE;

while (hashTable[index] != -1) {

System.out.println("\nCollision found at address " + index + " for " + key);

System.out.println("Searching next empty slot using linear probing!");

index = (index + 1) % SIZE; // Find the next available slot

}

System.out.println("\nNo collision at address " + index + " for " + key);

hashTable[index] = key;

}

// Display the hash table for probing methods

public void displayProbing() {

System.out.println("\n\n");

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

if (hashTable[i] != -1) {

System.out.println(" Index " + i + " : " + hashTable[i]);

} else {

System.out.println(" Index " + i + " : NULL");

}

}

}

public static void main(String[] args) {

int[] keys = {25, 36, 47, 55, 63, 75, 88, 92}; // keys to insert

HashTable ht = new HashTable();

System.out.println("\nHashing using Linear Probing:\n");

for (int key : keys) {

ht.insertLinearProbing(key);

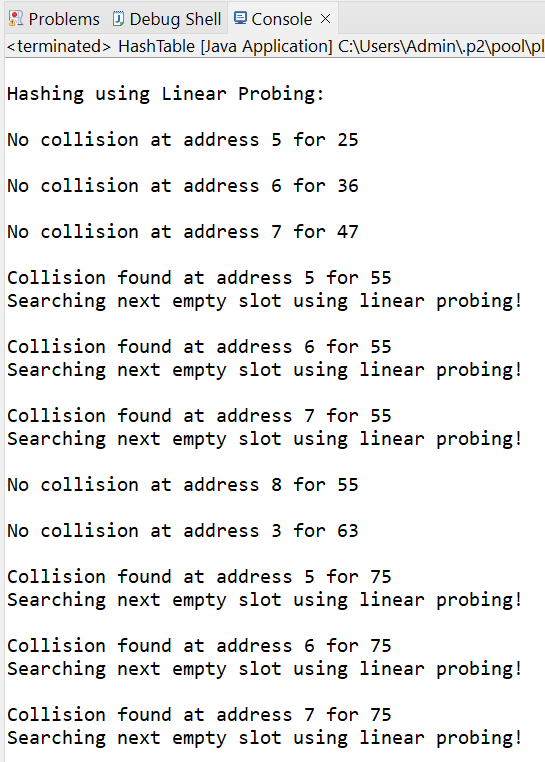
}

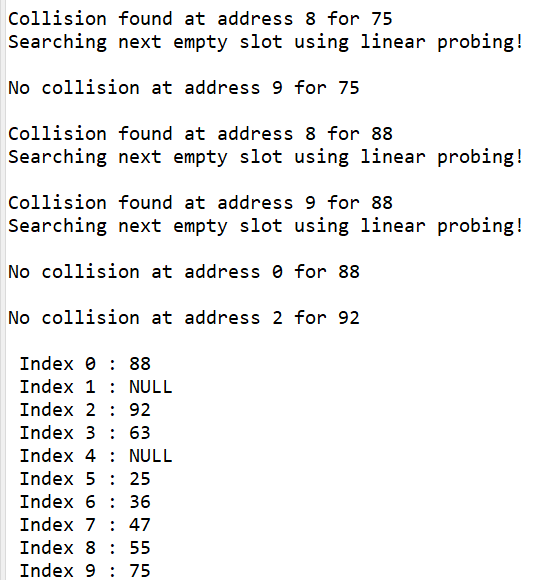
ht.displayProbing();

}

}

**Output:**





**2. Write a java program to hash various keys using Digit-Extraction hashing methods and use linear probe for collision resolution.**

**Program:**

**HashTableDigitExt.java:**

package mypack;

import java.util.Arrays;

public class HashTableDigitExt {

private static final int SIZE = 10; // size of the hash table

private int[] hashTable; // Array to store the hash table

// Constructor to initialize hash table

public HashTableDigitExt() {

hashTable = new int[SIZE];

Arrays.fill(hashTable, -1); // -1 indicates an empty slot for probing

}

// Insert key into hash table

public void insert(int key, int digitPosition) {

int numDigits = (int) Math.log10(key) + 1; // Count total number of digits in the key

if (digitPosition > numDigits || digitPosition <= 0) {

System.out.println(" Invalid digit position!");

return;

}

// Extract the digit at the desired position and assign to index (key/10^(numDigits-digitPosition))%10

int index = (key / (int) Math.pow(10, numDigits - digitPosition)) % 10;

while (hashTable[index] != -1) {

System.out.println("\nCollision found at address " + index + " for " + key);

System.out.println("Searching next empty slot using linear probing!");

index = (index + 1) % SIZE; // Find the next available slot

}

System.out.println("\nNo collision at address " + index + " for " + key);

hashTable[index] = key;

}

// Display the hash table

public void display() {

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

if (hashTable[i] != -1) {

System.out.println(" Index " + i + " : " + hashTable[i]);

} else {

System.out.println(" Index " + i + " : NULL");

}

}

}

public static void main(String[] args) {

int[] keys = {12345, 67890, 13579, 24680, 98765, 43210, 56789, 10234}; // keys to insert

int n = keys.length;

int digitPosition = 3;

System.out.print("Keys: ");

for (int key : keys) {

System.out.print(key+" " );

}

System.out.println();

HashTableDigitExt ht = new HashTableDigitExt();

for (int key : keys) {

ht.insert(key, digitPosition);

}

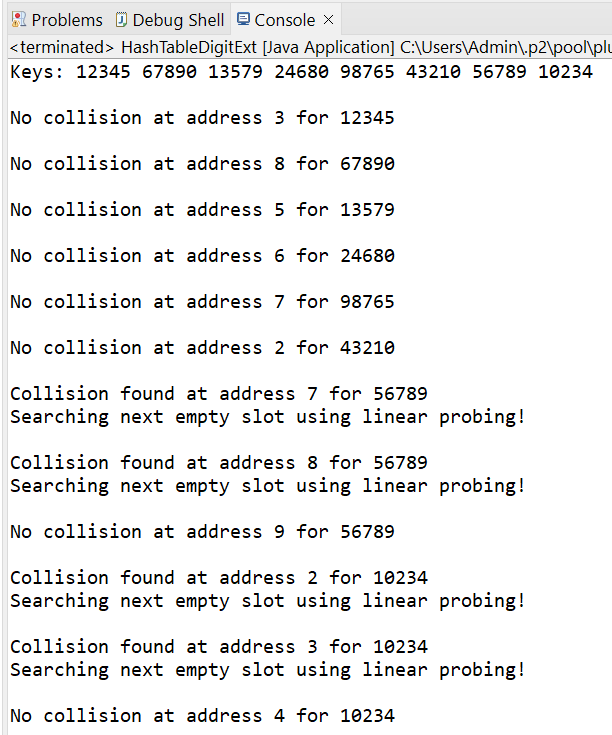
System.out.println("\n\nHash Table using Digit Extraction Hashing (place of digit is " + digitPosition + " )\n");

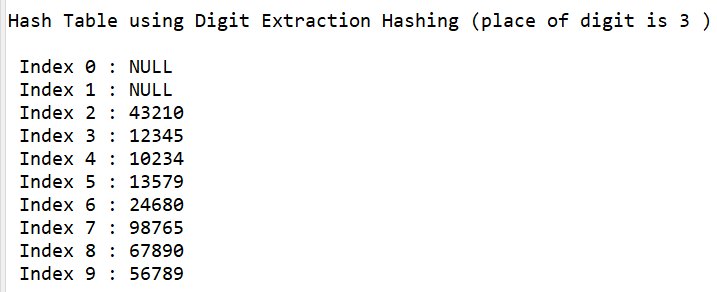
ht.display();

}

}

**Output:**





**Conclusion: Implemented hashing technique with collision resolution successfully.**