**Name: Shantanu Sethi**

**Roll no: 163**

**Aim: To implement hashing methods and collision resolution techniques**

1. Modulo Division

2. Digit Extraction

3. Fold shift

4. Fold Boundary

5. Linear Probe for Collision Resolution

**Objectives:**

1. Learn how to map a large amount of data to a smaller table using a “hash function”
2. Learn to how to solve collision using Linear Probing

**Theory:**

**Modulo – Division Method:**

Address = key MODULO list size + 1

**CODE:-**

**package** hashing;

**import** java.util.Scanner;

**public** **class** modulodeviation {

**public** **static** **void** main(String[] args) {

ModularDivisionHashing hashing = **new** ModularDivisionHashing();

hashing.modularDivision();

}

**public** **static** **class** ModularDivisionHashing {

**private** **int**[] list;

**private** **int**[] givenKeys;

**private** **static** **final** **int** ***SIZE*** = 100;

**public** ModularDivisionHashing() {

list = **new** **int**[***SIZE***];

givenKeys = **new** **int**[5];

}

**public** **void** accept() {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.println("184 - Abhinav Singh");

System.***out***.println("Enter 4-Digit Keys (4 keys):");

**for** (**int** i = 0; i < 4; i++) {

givenKeys[i] = scanner.nextInt();

}

}

**public** **void** modularDivision() {

accept();

**for** (**int** i = 0; i < 5; i++) {

**int** index = (givenKeys[i] % ***SIZE***);

**if** (list[index] == 0) {

list[index] = givenKeys[i];

} **else** {

**int** temp = index;

**while** (list[temp] != 0) {

temp++;

**if** (temp >= ***SIZE***) {

temp = 0;

}

}

list[temp] = givenKeys[i];

}

}

print();

}

**public** **void** print() {

System.***out***.println("Hash Table:");

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

**if** (list[i] != 0) {

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

}

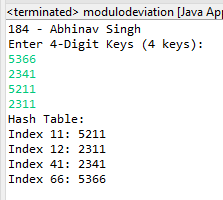
}

}

}

}

**OUTPUT:-**



**Digit Extraction:**

Selected digits are extracted from the key and used as the address.

**CODE:-**

**package** hashing;

**import** java.util.Scanner;

**public** **class** Digit\_ex{

**public** **static** **void** main(String[] args) {

**int**[] list = **new** **int**[100];

**int**[] givenKeys = **new** **int**[5];

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.println(" 184 - Abhinav Singh");

System.***out***.println("Enter 3-Digit Keys (4 keys):");

**for** (**int** i = 0; i < 4; i++) {

givenKeys[i] = scanner.nextInt();

}

System.***out***.println("Enter 2 positions (0, 1, or 2): ");

**int** pos1 = scanner.nextInt();

**int** pos2 = scanner.nextInt();

**for** (**int** i = 0; i < 5; i++) {

**int** d1 = (givenKeys[i] / 100) % 10;

**int** d2 = (givenKeys[i] / 10) % 10;

**int** d3 = givenKeys[i] % 10;

**int** add = 0;

**if** (pos1 == 0) add += d1;

**if** (pos1 == 1) add += d2;

**if** (pos1 == 2) add += d3;

**if** (pos2 == 0) add += d1;

**if** (pos2 == 1) add += d2;

**if** (pos2 == 2) add += d3;

**int** index = add % 100;

**if** (list[index] == 0) {

list[index] = givenKeys[i];

} **else** {

**int** temp = index;

**while** (list[temp] != 0) {

temp++;

**if** (temp >= 100) {

temp = 0;

}

}

list[temp] = givenKeys[i];

}

}

System.***out***.println("Hash Table:");

**for** (**int** i = 0; i < 100; i++) {

**if** (list[i] != 0) {

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

}

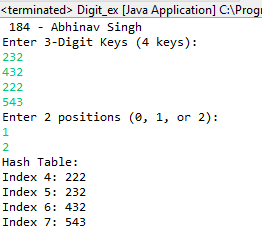
}

scanner.close();

}

}

**OUTPUT:-**



**Fold Shift :**

key is divided into number of parts say k1,k2,…..,kn where each parts has the same number of digits except the last part , which can have lesser digits.Add all these parts and ignore last carry.

For eg. If Key = 123456789

**123  
 + 456  
 789  
 1368** Discard 1 so the address is **368**

**CODE:-**

**package** hashing;

**import** java.util.Scanner;

**public** **class** foldshift {

**public** **static** **void** main(String[] args) {

System.***out***.println("184 - Abhinav Singh");

**long** key = 123456789;

**int** partSize = 3;

**int** address = *calculateAddress*(key, partSize);

System.***out***.println("Key: " + key);

System.***out***.println("Final Address: " + address);

}

**public** **static** **int** calculateAddress(**long** key, **int** partSize) {

**int** sum = 0;

String keyString = Long.*toString*(key);

System.***out***.println("Parts:");

**for** (**int** i = 0; i < keyString.length(); i += partSize) {

String part = keyString.substring(i, Math.*min*(i + partSize, keyString.length()));

System.***out***.println(part);

sum += Integer.*parseInt*(part);

}

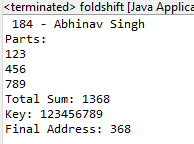
System.***out***.println("Total Sum: " + sum);

**return** sum % 1000;

}

}

**OUTPUT:-**



**Fold Boundary:**

left and right numbers are folded on a fixed boundary between them and the center number. These results in two outside values are being reversed.

For eg. If Key = 123-456-789

**321  
 456  
 + 987  
 1764**  Discard 1 so the address is : **764**

**CODE:-**

**package** hashing;

**import** java.util.Scanner;

**public** **class** foldBoundary {

**public** **static** **void** main(String[] args) {

System.***out***.println(" 184 - Abhinav Singh");

**final** **int** SIZE = 100;

String[] hashTable = **new** String[SIZE];

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter the numbers (format: XXX XXX XXX): ");

String input = scanner.nextLine();

String[] parts = input.split(" ");

**if** (parts.length != 3) {

**throw** **new** IllegalArgumentException("Input must be in the format 'XXX XXX XXX'");

}

String leftReversed = **new** StringBuilder(parts[0]).reverse().toString();

String centerValue = parts[1];

String rightReversed = **new** StringBuilder(parts[2]).reverse().toString();

**int** leftValue = Integer.*parseInt*(leftReversed);

**int** centerNum = Integer.*parseInt*(centerValue);

**int** rightValue = Integer.*parseInt*(rightReversed);

**int** sum = leftValue + centerNum + rightValue;

String sumString = String.*valueOf*(sum);

String address = sumString.substring(1);

**int** index = Integer.*parseInt*(address) % SIZE;

**if** (hashTable[index] == **null**) {

hashTable[index] = address;

} **else** {

**int** temp = index;

**while** (hashTable[temp] != **null**) {

temp = (temp + 1) % SIZE;

}

hashTable[temp] = address;

}

System.***out***.println("The address is: " + address);

System.***out***.println("Hash Table:");

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

**if** (hashTable[i] != **null**) {

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

}

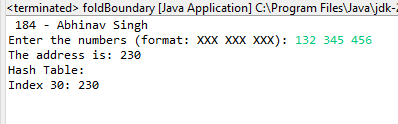
}

scanner.close();

}

}

**OUTPUT:-**



5. **Linear Probe for Collision Resolution**

**CODE:-**

**package** hashing;

**import** java.util.Scanner;

**public** **class** collision {

**public** **static** **void** main(String[] args) {

System.***out***.println("184 - ABHINAV SINGH");

**final** **int** SIZE = 100;

String[] hashTable = **new** String[SIZE];

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter the numbers (format: XXX XXX XXX): ");

String input = scanner.nextLine();

String[] parts = input.split(" ");

**if** (parts.length != 3) {

**throw** **new** IllegalArgumentException("Input must be in the format 'XXX XXX XXX'");

}

String leftReversed = **new** StringBuilder(parts[0]).reverse().toString();

String centerValue = parts[1];

String rightReversed = **new** StringBuilder(parts[2]).reverse().toString();

**int** leftValue = Integer.*parseInt*(leftReversed);

**int** centerNum = Integer.*parseInt*(centerValue);

**int** rightValue = Integer.*parseInt*(rightReversed);

**int** sum = leftValue + centerNum + rightValue;

String sumString = String.*valueOf*(sum);

String address = sumString.substring(1);

**int** index = Integer.*parseInt*(address) % SIZE;

**if** (hashTable[index] == **null**) {

hashTable[index] = address;

} **else** {

**int** temp = index;

**while** (hashTable[temp] != **null**) {

temp = (temp + 1) % SIZE;

}

hashTable[temp] = address;

}

System.***out***.println("The address is: " + address);

System.***out***.println("Hash Table:");

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

**if** (hashTable[i] != **null**) {

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

}

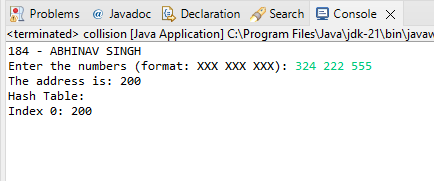
}

scanner.close();

}

}

**OUTPUT:-**

****