Experiment 9

Student Name: Zatch	**UID:**
Branch: BE-CSE	**Section:**
Semester: 5	**Date of Performance**
Subject Name: AP LAB -1	**Subject Code: 22CSP-314**

1. **Aim:** Sub string game.

2. **Objective:** The Samantha and Sam are playing a numbers game. Given a number as a string, no leading zeros, determine the sum of all integer values of substrings of the string. Given an integer as a string, sum all of its substrings cast as integers. As the number may become large, return the value modulo 109+7.

3. **Algorithm:**

a) Initialization:
- Initialize a variable totalSum = 0 to store the cumulative sum of all substring integers.
- Let currentSum = 0 to store the sum for the substrings ending at each position.
- Set MOD = 10^9 + 7 to handle large numbers.

b) Iterate Through the String:
- For each character at position i in the string (from left to right):
  - Convert the character to its integer value.
  - Update the currentSum as: currentSum=(currentSum×10+(i+1)×digit)%MODcurrentSum = (currentSum \times 10 + (i+1) \times \text{digit}) \% MODcurrentSum=(currentSum×10+(i+1)×digit)%MOD
  - Add currentSum to totalSum and take modulo MOD.

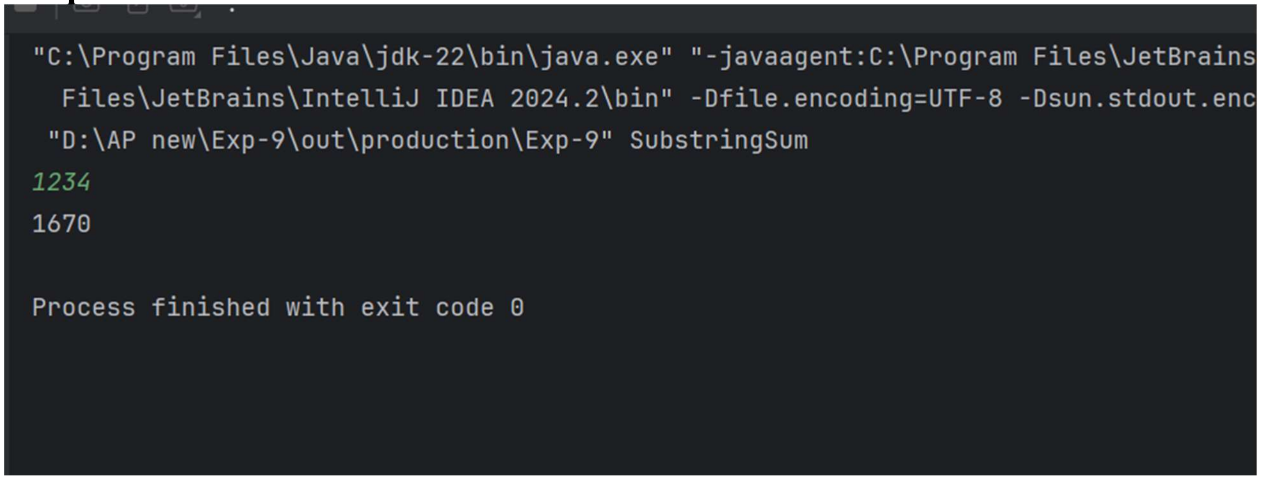
c) Return the Result:
- After processing all characters, return totalSum \% MOD.

4. **Code :**

```
import java.util.Scanner;
public class SubstringSum {
    public static int sumOfSubstrings(String number) {
        int n = number.length();
        long totalSum = 0;
        long currentSum = 0;
        long MOD = 1000000007;
```

```
            for (int i = 0; i < n; i++) {
                int digit = number.charAt(i) - '0';
                currentSum = (currentSum * 10 + (i + 1) * digit) % MOD;
                totalSum = (totalSum + currentSum) % MOD;
            }
            return (int) totalSum;
        }
        public static void main(String[] args) {
            Scanner sc = new Scanner(System.in);
            String number = sc.next();
            System.out.println(sumOfSubstrings(number));
        }
    }
```

5. **Output:**



```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains
  Files\JetBrains\IntelliJ IDEA 2024.2\bin" -Dfile.encoding=UTF-8 -Dsun.stdout.enc
 "D:\AP new\Exp-9\out\production\Exp-9" SubstringSum
1234
1670

Process finished with exit code 0
```

6. **(a) Time Complexity :** O(n)
   (b) Space Complexity : O(1)

7. **Learning Outcomes :**

a) Understood how to work with large numbers using the modulo operator to avoid overflow issues, especially in competitive programming.
b) Learnt an optimized way to calculate sums of all substrings of a number string by iterating over each digit once, updating the sum progressively.
c) Practiced managing large integers and input strings in Java.

Problem -2

1. **Aim :** Kingdom Division

2. **Objective :** King Arthur has a large kingdom that can be represented as a tree, where nodes correspond to cities and edges correspond to the roads between cities. The kingdom has a total of n cities numbered from 1 to n. The King wants to divide his kingdom between his two children, Reggie and Betty, by giving each of them 0 or more cities; however, they don't get along so he must divide the kingdom in such a way that they will not invade each other's cities. The first sibling will invade the second sibling's city if the second sibling has no other cities directly connected to it.

3. **Algorithm :**

a) Input:
- n: Number of cities.
- An array of edges representing the roads between the cities.

b) Tree Construction:
- Construct an adjacency list representing the tree using the given edges.

c) Subtree Sizes:
- Perform a DFS (Depth-First Search) traversal to calculate the size of the subtree for each node. The subtree size of a node is the number of cities (nodes) in the subtree rooted at that node.

d) Optimal Edge Removal:
- For each edge, calculate the size of the subtree it would create if the edge were cut.
- Calculate the difference between the sizes of the two resulting subtrees.
- Find the edge that minimizes this difference.

e) Return the result:
- Return the best edge to cut.

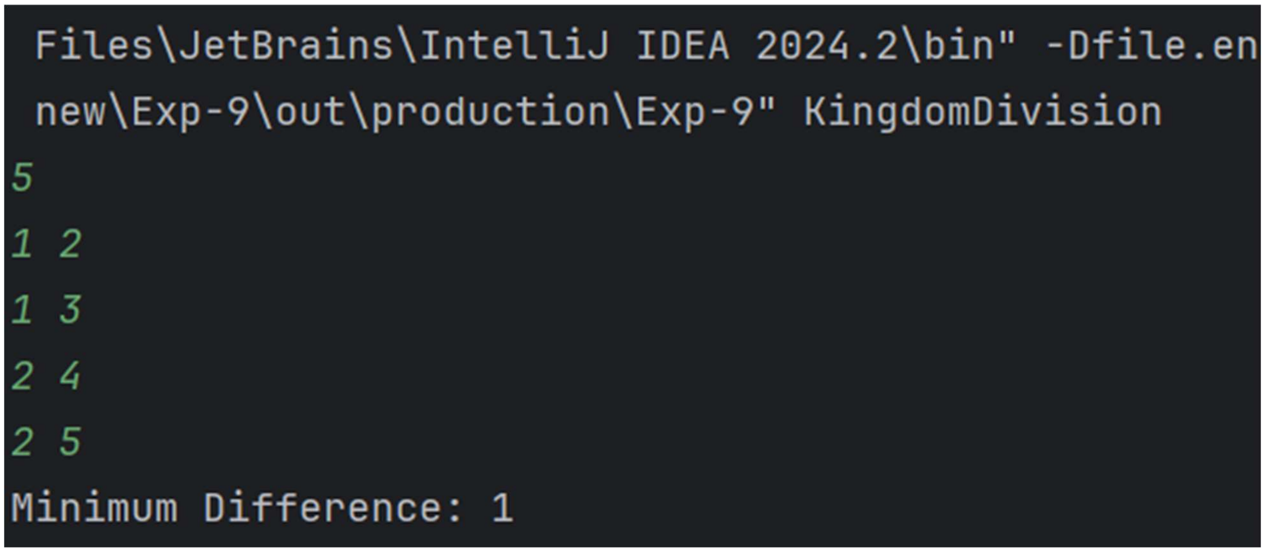
4. **Code :**

```
import java.util.*;
public class KingdomDivision {
    static List<List<Integer>> tree = new ArrayList<>();
    static int[] subtreeSize;
    static int totalCities;
    static int minDifference = Integer.MAX_VALUE;
    public static int calculateSubtreeSizes(int node, int parent) {
        subtreeSize[node] = 1;
        for (int neighbor : tree.get(node)) {
            if (neighbor != parent) {
                subtreeSize[node] += calculateSubtreeSizes(neighbor, node);
            }
        }
        return subtreeSize[node];
    }
    public static void findBestEdgeToCut(int node, int parent) {
        for (int neighbor : tree.get(node)) {
            if (neighbor != parent) {
                findBestEdgeToCut(neighbor, node);
                int size1 = subtreeSize[neighbor];
                int size2 = totalCities - size1;
                minDifference = Math.min(minDifference, Math.abs(size1 - size2));
            }
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        totalCities = sc.nextInt();
        subtreeSize = new int[totalCities + 1];

        for (int i = 0; i <= totalCities; i++) {
            tree.add(new ArrayList<>());
        }
        for (int i = 0; i < totalCities - 1; i++) {
            int u = sc.nextInt();
            int v = sc.nextInt();
            tree.get(u).add(v);
            tree.get(v).add(u);
        }
        calculateSubtreeSizes(1, -1);
        findBestEdgeToCut(1, -1);
        System.out.println("Minimum Difference: " + minDifference);
    }
}
```

5. **a)Time Complexity :** O(n)
   b)Space Complexity : O(n)

6. **Output :**



```
 Files\JetBrains\IntelliJ IDEA 2024.2\bin" -Dfile.en
 new\Exp-9\out\production\Exp-9" KingdomDivision
5
1 2
1 3
2 4
2 5
Minimum Difference: 1
```

7. **Learning Outcomes :**

a) Applying DFS to calculate subtree sizes and find optimal edge removal in a tree structure.
b) Learning to optimize division by minimizing differences in the sizes of divided parts.
c) Applying graph-theoretical concepts (trees, edge cuts) to solve practical problems like kingdom division.