DSA Practice Solutions – 12.11.24

NAME: N J Meenakshi

DEPARTMENT: CSBS

ROLL NO.: 22CB028

1. **Check if two Strings are Anagrams of each other**
   Given two strings s1 and s2 consisting of lowercase characters, the task is to check whether the two given strings are anagrams of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different.
   Code:

```
import java.util.Scanner;

public class Anagrams {
    public static boolean solution(String s, String t){
        if (s.length()!=t.length()) return false;
        int[] freq=new int[26];
        for(char ch:s.toCharArray()){
            freq[ch-'a']++;
        }
        for(char ch:t.toCharArray()){
            freq[ch-'a']--;
        }
        for(int count:freq){
            if (count!=0) return false;
        }
        return true;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter s: ");
        String s=sc.next();
        System.out.print("Enter t: ");
        String t=sc.next();
        System.out.print("Result: "+solution(s, t));
        sc.close();
    }
}
```

Output:

```
Enter s: geeks
Enter t: kseeg
Result: true

Enter s: allergy
Enter t: allergic
```

Result: false

Enter s: g
Enter t: g
Result: true

Time Complexity: O(N)

2. **Find the row with maximum number of 1s**
   Given a binary 2D array, where each row is sorted. Find the row with the maximum number of 1s.
   Input: mat = [[1, 1, 1, 1], [1, 1, 0, 0], [0, 0, 1, 1], [1, 1, 1, 1]]
   Output: 0
   Input: mat = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
   Output: -1

   Code:

```
import java.util.*;
public class Row_with_max1s {
    public static int solution(int[] arr,int n){
        int low=0,high=n-1,res=n;
        while(low<=high){
            int mid=(low+high)/2;
            if (arr[mid]>=1){
                res=mid;
                high=mid-1;
            }
            else{
                low=mid+1;
            }
        }
        return res;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        System.out.print("Enter m: ");
        int m=sc.nextInt();
        int[][] mat=new int[n][m];
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                mat[i][j]=sc.nextInt();
            }
        }
        int res=0,ans=-1;
        for(int i=0;i<n;i++){
            int curr=m-solution(mat[i], m);
            if (curr>res){
                res=curr;
                ans=i;
            }
        }
        System.out.print("Row with max 1s: "+ans);
        sc.close();
    }
}
```

Output:

```
Enter n: 4
Enter m: 4
1 1 1 1
1 1 0 0
0 0 1 1
1 1 1 1
Row with max 1s: 0

Enter n: 3
Enter m: 3
0 0 0
0 0 0
0 0 0
Row with max 1s: -1
```

Time Complexity: O(N)

3. **Longest Consecutive Subsequence**

Given an array arr of non-negative integers. Find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers, the consecutive numbers can be in any order.

Input: arr[] = [2, 6, 1, 9, 4, 5, 3]
Output: 6
Explanation: The consecutive numbers here are 1, 2, 3, 4, 5, 6. These 6 numbers form the longest consecutive subsquence.
Input: arr[] = [1, 9, 3, 10, 4, 20, 2]
Output: 4
Explanation: 1, 2, 3, 4 is the longest consecutive subsequence.
Input: arr[] = [15, 13, 12, 14, 11, 10, 9]
Output: 7
Explanation: The longest consecutive subsequence is 9, 10, 11, 12, 13, 14, 15, which has a length of 7.

Code:

```java
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;

public class Longest_Consecutive_subseq {
    public static int solution(int arr[]){
        int n=arr.length;
        if (n==0) return 0;
        int res=1;
        Set<Integer> set=new HashSet<>();
        for(int i=0;i<n;i++){
            set.add(arr[i]);
        }
        for(int i:set){
            if (!set.contains(i-1)){
                int count=1;
                int x=i;
                while(set.contains(x+1)){
                    count+=1;
                    x=x+1;
                }
                res=Math.max(res, count);
            }
        }
        return res;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        int arr[]=new int[n];
        for(int i=0;i<n;i++){
            arr[i]=sc.nextInt();
        }
        System.out.print("Longest subsequence: "+solution(arr));
        sc.close();
    }
}
```

Output:

```
Enter n: 7
2 6 1 9 4 5 3
Longest subsequence: 6

Enter n: 7
1 9 3 10 4 20 2
Longest subsequence: 4
```

Time Complexity: O(N)

4. **Longest Palindromic Substring**

Given a string S, find the longest palindromic substring in S. **Substring of string S:** S[i . . . . j] where 0 ≤ i ≤ j < len(S)**.**

Input:

S = "aaaabbaa"

Output:

aabbaa

Explanation:

The longest palindrome string present in the given string is "aabbaa".

Code:

```java
import java.util.Scanner;

public class Longest_Pal_Substring {
    public static String solution(String s){
        if (s==null || s.length()<1) return "";
        String res="";
        for (int i=0;i<s.length();i++){
            String sub1=expandAtCenter(s,i,i);
            if(sub1.length()>res.length()) res=sub1;
            String sub2=expandAtCenter(s,i,i+1);
            if(sub2.length()>res.length()) res=sub2;
        }
        return res;
    }
    public static String expandAtCenter(String s, int left, int right){
        while((left>=0 && right<s.length()) && (s.charAt(left)==s.charAt(right))){
            left--;
            right++;
        }
        return s.substring(left+1,right);
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter s: ");
        String s=sc.next();
        System.out.print("Longest Palindromic Substring: "+solution(s));
        sc.close();
    }
}
```

Output:

```
Enter s: forgeeksskeegfor
Longest Palindromic Substring: geeksskeeg

Enter s: Geeks
Longest Palindromic Substring: ee

Enter s: abc
Longest Palindromic Substring: a

Enter s:
""
Longest Palindromic Substring: ""
```

Time Complexity: O(N^2)

5. Rat in a Maze Problem

Consider a rat placed at (0, 0) in a square matrix of order N * N. It has to reach the destination at (N – 1, N – 1). Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are 'U'(up), 'D'(down), 'L' (left), 'R' (right). Value 0 at a cell in the matrix represents that it is blocked and rat cannot move to it while value 1 at a cell in the matrix represents that rat can be travel through it. Return the list of paths in lexicographically increasing order.

Note: In a path, no cell can be visited more than one time. If the source cell is 0, the rat cannot move to any other cell.

Input: mat[][] = [[1, 0, 0, 0],
            [1, 1, 0, 1],

[1, 1, 0, 0],
[0, 1, 1, 1]]

Output: DDRDRR DRDDRR

Explanation: The rat can reach the destination at (3, 3) from (0, 0) by two paths - DRDDRR and DDRDRR, when printed in sorted order we get DDRDRR DRDDRR.

Input: mat[][] = [[1, 0],
[1, 0]]

Output: -1

Explanation: No path exists and destination cell is blocked.

Code:

```java
import java.util.ArrayList;
import java.util.Scanner;

public class Rat_in_a_Maze {
    public static void solve(int i, int j,int n, int m, ArrayList<String> ans, int[][]
vis, String move, int[][] a){
        if(i==n-1 && j==m-1){
            ans.add(move);
            return;
        }
        if(i+1<n && vis[i+1][j]==0 && a[i+1][j]==1){
            vis[i][j]=1;
            solve(i+1, j, n, m, ans, vis, move+'D', a);
            vis[i][j]=0;
        }
        if(j-1>=0 && vis[i][j-1]==0 && a[i][j-1]==1){
            vis[i][j]=1;
            solve(i, j-1, n, m, ans, vis, move+'L', a);
            vis[i][j]=0;
        }
        if(j+1<m && vis[i][j+1]==0 && a[i][j+1]==1){
            vis[i][j]=1;
            solve(i, j+1, n, m, ans, vis, move+'R', a);
            vis[i][j]=0;
        }
        if(i-1>=0 && vis[i-1][j]==0 && a[i-1][j]==1){
            vis[i][j]=1;
            solve(i-1, j, n, m, ans, vis, move+'U', a);
            vis[i][j]=0;
        }
    }
    public static ArrayList<String> path(int[][] a, int n, int m){
        int vis[][] =new int[n][m];
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                vis[i][j]=0;
            }
        }
        ArrayList<String> ans=new ArrayList<>();
        solve(0,0,n,m,ans,vis,"",a);
        return ans;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        System.out.print("Enter m: ");
        int m=sc.nextInt();
        int[][] mat=new int[n][m];
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                mat[i][j]=sc.nextInt();
            }
        }
        ArrayList<String> res=path(mat, n, m);
        if(res.size()>0){
            for(int i=0;i<res.size();i++){
                System.out.print(res.get(i)+" ");
            }
        }
        else{
            System.out.print(-1);
        }
        sc.close();
    }
}
```

Output:

```
Enter n: 4
Enter m: 4
1 0 0 0
1 1 0 1
1 1 0 0
0 1 1 1
DDRDRR DRDDRR

Enter n: 2
Enter m: 2
1 0
1 0
-1
```

Time Complexity: O(4^(N*M))