GATE TECHNICAL TRAINING – DSA CODING PRACTICE PROBLEMS 2026

DATE: 12-11-24 **NAME: JASHVARTHINI R – CSBS**

1. ANAGRAM PROGRAM

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.

Examples:

Input: s1 = “geeks” s2 = “kseeg”
Output: true
Explanation: Both the string have same characters with same frequency. So, they are anagrams.

Input: s1 = “allergy” s2 = “allergic”
Output: false
Explanation: Characters in both the strings are not same. s1 has extra character ‘y’ and s2 has extra characters ‘i’ and ‘c’, so they are not anagrams.

Input: s1 = “g”, s2 = “g”
Output: true
Explanation: Characters in both the strings are same, so they are anagrams.

CODE:

```java
import java.util.Arrays;
import java.util.Scanner;

public class AnagramCheck {

    public static boolean areAnagrams(String s1, String s2) {
        if (s1.length() != s2.length()) {
            return false;
        }

        char[] arr1 = s1.toCharArray();
        char[] arr2 = s2.toCharArray();
        Arrays.sort(arr1);
        Arrays.sort(arr2);

        return Arrays.equals(arr1, arr2);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter first string: ");
        String s1 = scanner.nextLine();
```

```
        System.out.print("Enter second string: ");
        String s2 = scanner.nextLine();

        if (areAnagrams(s1, s2)) {
            System.out.println("The strings are anagrams.");
        } else {
            System.out.println("The strings are not anagrams.");
        }

        scanner.close();
    }
}
```

OUTPUT:

```
Enter first string: geeks
Enter second string: kseeg
The strings are anagrams.
```

```
Enter first string: allergy
Enter second string: allergic
The strings are not anagrams.
```

```
Enter first string: racecar
Enter second string: racecar
The strings are anagrams.
```

TIME COMPLEXITY: O(1)

2. **ROW WITH MAX 1S'**

Given a m x n binary matrix mat, find the 0-indexed position of the row that contains the maximum count of ones, and the number of ones in that row.

In case there are multiple rows that have the maximum count of ones, the row with the smallest row number should be selected.

Return an array containing the index of the row, and the number of ones in it.

Example 1:

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

Output: [0,1]

Explanation: Both rows have the same number of 1's. So we return the index of the smaller row, 0, and the maximum count of ones (1). So, the answer is [0,1].

Example 2:

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

Output: [1,2]

Explanation: The row indexed 1 has the maximum count of ones (2). So we return its index, 1, and the count. So, the answer is [1,2].

Example 3:

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

Output: [1,2]

Explanation: The row indexed 1 has the maximum count of ones (2). So the answer is [1,2].

CODE:

```
import java.util.Scanner;

class Solution {
    public int[] rowAndMaximumOnes(int[][] mat) {
        int max = 0;
        int[] result = new int[2];

        for (int i = 0; i < mat.length; i++) {
            int count = 0;
            for (int j = 0; j < mat[i].length; j++) {
                if (mat[i][j] == 1) {
                    count++;
                }
            }
            if (count > max) {
                max = count;
                result[0] = i;
                result[1] = count;
            }
        }

        return result;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of rows in the matrix: ");
        int rows = scanner.nextInt();
        System.out.print("Enter the number of columns in the matrix: ");
        int cols = scanner.nextInt();

        int[][] mat = new int[rows][cols];

        System.out.println("Enter the elements of the matrix (0 or 1):");
```

```
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                mat[i][j] = scanner.nextInt();
            }
        }

        Solution solution = new Solution();
        int[] result = solution.rowAndMaximumOnes(mat);

        System.out.println("Row with maximum number of 1's: " + result[0]);
        System.out.println("Number of 1's in that row: " + result[1]);

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of rows in the matrix: 2
Enter the number of columns in the matrix: 2
Enter the elements of the matrix (0 or 1):
0 1
1 0
Row with maximum number of 1's: 0
Number of 1's in that row: 1
```

TIME COMPLEXITY: O(n*m)

3. LONGEST CONSECUTIVE SUBSEQUENCE

Given an array of 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.

Examples:

Input: arr[] = {1, 9, 3, 10, 4, 20, 2}
Output: 4
Explanation: The subsequence 1, 3, 4, 2 is the longest subsequence of consecutive elements

Input: arr[] = {36, 41, 56, 35, 44, 33, 34, 92, 43, 32, 42}
Output: 5
Explanation: The subsequence 36, 35, 33, 34, 32 is the longest subsequence of consecutive elements.

CODE:

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

class ArrayElements {
    static int findLongestConseqSubseq(int arr[], int n) {
        HashSet<Integer> S = new HashSet<Integer>();
```

```
        int ans = 0;

        for (int i = 0; i < n; ++i)
            S.add(arr[i]);

        for (int i = 0; i < n; ++i) {
            if (!S.contains(arr[i] - 1)) {
                int j = arr[i];
                while (S.contains(j))
                    j++;

                if (ans < j - arr[i])
                    ans = j - arr[i];
            }
        }
        return ans;
    }

    public static void main(String args[]) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();

        int arr[] = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        System.out.println("Length of the Longest consecutive subsequence is "
                + findLongestConseqSubseq(arr, n));

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of elements in the array: 7
Enter the elements of the array:
1
9
3
10
4
20
2
Length of the Longest consecutive subsequence is 4
```

TIME COMPLEXITY: O(n)

4. LONGEST PALINDROME IN A STRING

Given a string S of length N, the task is to find the length of the longest palindromic substring from a given string.

Examples:

Input: S = “abcbab”
Output: 5
Explanation:
string “abcba” is the longest substring that is a palindrome which is of length 5.

CODE:

```
import java.util.HashMap;
import java.util.Scanner;

class Solution {
    public int longestPalindrome(String s) {
        int res = 0;
        int val = 0;
        HashMap<Character, Integer> mp = new HashMap<>();

        for (char ch : s.toCharArray()) {
            mp.put(ch, mp.getOrDefault(ch, 0) + 1);
        }

        for (int count : mp.values()) {
            if (count % 2 != 0) {
                res += count - 1;
                val = 1;
            } else {
                res += count;
            }
        }

        return res + val;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a string: ");
        String s = scanner.nextLine();

        Solution solution = new Solution();
        int result = solution.longestPalindrome(s);
```

```
        System.out.println("Length of longest palindrome that can be built: " +
result);

        scanner.close();
    }
}
```

OUTPUT:

```
Enter a string: assessment
Length of longest palindrome that can be built: 7
```

```
Enter a string: abccccdd
Length of longest palindrome that can be built: 7
```

TIME COMPLEXITY: O(n)

5. RAT IN A MAZE PROBLEM

Consider a rat placed at (0, 0) in a square matrix mat 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.
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. In case of no path, return an empty list. The driver will output "-1" automatically.

Examples:

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.

Expected Time Complexity: O(3n^2)
Expected Auxiliary Space: O(l * x)
Here l = length of the path, x = number of paths.

Constraints:
2 ≤ n ≤ 5
0 ≤ mat[i][j] ≤ 1

CODE:

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

class Solution{
    public ArrayList<String> findPath(int[][] mat) {
        int n = mat.length;
        ArrayList<String> paths = new ArrayList<>();

        if (mat[0][0] == 0 || mat[n - 1][n - 1] == 0) {
            return paths;
        }

        boolean[][] visited = new boolean[n][n];
        solve(mat, 0, 0, n, "", paths, visited);

        paths.sort(null);
        return paths;
    }

    public static void solve(int[][] mat, int x, int y, int n, String path,
ArrayList<String> paths, boolean[][] visited) {
        if(x == n - 1 && y == n - 1) {
            paths.add(path);
            return;
        }

        visited[x][y] = true;

        int[] rowMove = {1, 0, 0, -1};
        int[] colMove = {0, -1, 1, 0};
        char[] direction = {'D', 'L', 'R', 'U'};

        for(int i = 0; i < 4; i++) {
            int newX = x + rowMove[i];
            int newY = y + colMove[i];

            if(isSafe(newX, newY, n, mat, visited)) {
                solve(mat, newX, newY, n, path + direction[i], paths, visited);
            }
        }
        visited[x][y] = false;
```

```
    }

    public static boolean isSafe(int x, int y, int n, int[][] mat, boolean[][]
visited) {
        return (x >= 0 && x < n && y >= 0 && y < n && mat[x][y] == 1 &&
!visited[x][y]);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the size of the matrix (n): ");
        int n = scanner.nextInt();

        int[][] mat = new int[n][n];
        System.out.println("Enter the matrix elements row by row (0 or 1):");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                mat[i][j] = scanner.nextInt();
            }
        }

        Solution solution = new Solution();
        ArrayList<String> paths = solution.findPath(mat);

        if (paths.isEmpty()) {
            System.out.println("No path found.");
        } else {
            System.out.println("Paths from top-left to bottom-right:");
            for (String path : paths) {
                System.out.println(path);
            }
        }

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the size of the matrix (n): 4
Enter the matrix elements row by row (0 or 1):
1 0 0 0
1 1 0 1
1 1 0 0
0 1 1 1
Paths from top-left to bottom-right:
DDRDRR
DRDDRR
```

```
Enter the size of the matrix (n): 2
Enter the matrix elements row by row (0 or 1):
1 0
1 0
No path found.
```

TIME COMPLEXITY: O(3n^2)