DSA PRACTICE PROBLEMS – SET 3

Madhulekha R – AIML - [12/11/2024]

1. **Question:** Given two strings of s1 and s2, return true if s2 is an anagram of s1, and false otherwise.

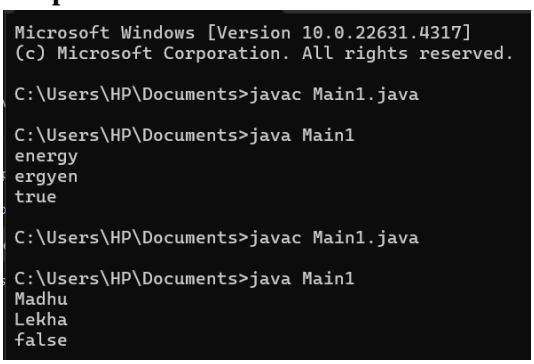
   Code:

```
import java.util.*;
import java.util.Scanner;

class Main1{
  public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    String s1 = sc.next();
    String s2 = sc.next();
    System.out.println(anagram(s1,s2));
  }

  static Boolean anagram(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);
  }
}
```

   Output:



Time Complexity: O(n log n)
Space Complexity: O(n)

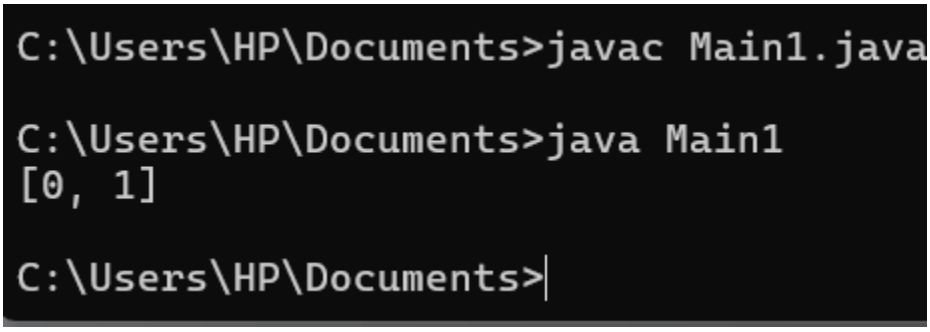
2. **Question:** Rows with maximum one's

Code:

```
import java util.*;

class Main1 {
   public static void main(String[] args){
     int[][] mat = {{0,1},{1,0}};
     System.out.println(Arrays.toString(func(mat)));
   }
   static int[] func(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;
   }
}
```

Output:



Time Complexity: O(1)
Space Complexity: O(m*n)

3. **Question:** 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.

Code:

```
import java.io.*;
import java.util.*;

class Main1{
   public static void main(String[] args){
      int arr[] = { 1, 9, 3, 10, 4, 20, 2 };
      int n = arr.length;
      System.out.println("Length of the Longest subsequence is " +
findLongestConseqSubseq(arr, n));
   }

   static int findLongestConseqSubseq(int arr[], int n){
      Arrays.sort(arr);
      int ans = 0, count = 0;
      ArrayList<Integer> arr1 = new ArrayList<Integer>();
      arr1.add(10);

      for (int i = 1; i < n; i++) {
         if (arr[i] != arr[i - 1])
            arr1.add(arr[i]);
      }

      for (int i = 0; i < arr1.size(); i++) {
         if (i > 0 && arr1.get(i) == arr1.get(i - 1) + 1){
            count++;
         } else {
            count = 1;
         }
         ans = Math.max(ans, count);
      }
      return ans;
   }
}
```

Output:

```
C:\Users\HP\Documents>javac Main1.java

C:\Users\HP\Documents>java Main1
Length of the Longest subsequence is 3
```

Time Complexity: O(n log n)
Space Complexity: O(n)

4. **Question:** Longest palindrome in a string
Given a string **str**, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.

Code:

```
import java.util.*;

public class Main1{
   public static void main(String[] args) {
      String s = "malayalam";
      System.out.println(longestPalSubstr(s));
      String s1 = "ramar";
      System.out.println(longestPalSubstr(s1));
   }

   static String longestPalSubstr(String s) {
      int n = s.length();
      boolean[][] dp = new boolean[n][n];
      int maxLen = 1;
      int start = 0;

      for (int i = 0; i < n; ++i)
         dp[i][i] = true;
      for (int i = 0; i < n - 1; ++i) {
         if (s.charAt(i) == s.charAt(i + 1)) {
            dp[i][i + 1] = true;
            start = i;
            maxLen = 2;
         }
      }
      for (int k = 3; k <= n; ++k) {
         for (int i = 0; i < n - k + 1; ++i) {
```

```
                int j = i + k - 1;
                if (dp[i + 1][j - 1] && s.charAt(i) == s.charAt(j)) {
                    dp[i][j] = true;
                    if (k > maxLen) {
                        start = i;
                        maxLen = k;
                    }
                }
            }
        }

        return s.substring(start, start + maxLen);
    }
}
```

Output:

```
C:\Users\HP\Documents>javac Main1.java

C:\Users\HP\Documents>java Main1
malayalam
ramar
```

Time Complexity: O(n^2)
Space Complexity: O(n^2)

5. **Question:** 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.

Code:

```
import java.util.ArrayList;
import java.util.List;

public class Main1{
    public static List<String> findPath(int[][] mat, int n) {
        List<String> result = new ArrayList<>();
```

```
        if (mat[0][0] == 0 || mat[n - 1][n - 1] == 0) return result;
        boolean[][] visited = new boolean[n][n];
        dfs(mat, n, 0, 0, "", visited, result);
        return result;
    }

    private static void dfs(int[][] mat, int n, int row, int col, String path, boolean[][] visited,
List<String> result) {
        if (row == n - 1 && col == n - 1) {
            result.add(path);
            return;
        }

        visited[row][col] = true;
        int[] dRow = {1, 0, 0, -1};
        int[] dCol = {0, -1, 1, 0};
        char[] moves = {'D', 'L', 'R', 'U'};

        for (int i = 0; i < 4; i++) {
            int newRow = row + dRow[i];
            int newCol = col + dCol[i];
            if (isSafe(mat, n, newRow, newCol, visited)) {
                dfs(mat, n, newRow, newCol, path + moves[i], visited, result);
            }
        }
        visited[row][col] = false;
    }

    private static boolean isSafe(int[][] mat, int n, int row, int col, boolean[][] visited) {
        return row >= 0 && row < n && col >= 0 && col < n && mat[row][col] == 1 &&
!visited[row][col];
    }

    public static void main(String[] args) {
        int[][] mat = {{1, 0, 0, 0}, {1, 1, 0, 1}, {0, 1, 0, 0}, {1, 1, 1, 1}};
        int n = mat.length;
        List<String> paths = findPath(mat, n);
        if (paths.isEmpty()) {
            System.out.println("-1");
        } else {
            System.out.println(paths);
        }
    }
}
```

Output:

```
C:\Users\HP\Documents>javac Main1.java

C:\Users\HP\Documents>java Main1
[DRDDRR]
```

Time Complexity: O(4^(m*n))
Space Complexity: O(n^2)