12-11-2024

DSA Problems (SET - 3)

1. Row with Max Ones

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].

Constraints:

- m == mat.length
- n == mat[i].length
- 1 <= m, n <= 100
- mat[i][j] is either 0 or 1.

SOLUTION:

```
import java.util.*;

class RowWithMaxOne {

   public static int[] rowAndMaximumOnes(int[][] mat) {

      int c;

      int val = 0;

      int ind = 0;
```

```
        for (int i = 0; i < mat.length; i++) {
            c = 0;
            for (int j = 0; j < mat[0].length; j++) {
                if (mat[i][j] == 1) {
                    c++;
                }
            }
            if (c > val) {
                val = c;
                ind = i;
            }
        }
        return new int[]{ind, val};
    }
    public static void main(String[] args) {
        int[][] mat1 = {{0, 1}, {1, 0}};
        int[][] mat2 = {{0, 0, 0}, {0, 1, 1}};
        int[][] mat3 = {{0, 0}, {1, 1}, {0, 0}};
        System.out.println("Output for mat1: " + Arrays.toString(rowAndMaximumOnes(mat1)));
        System.out.println("Output for mat2: " + Arrays.toString(rowAndMaximumOnes(mat2)));
        System.out.println("Output for mat2: " + Arrays.toString(rowAndMaximumOnes(mat3)));
    }
}
```

OUTPUT:

```
D:\OOPS\12-11-2024 practice.java>javac RowWithMaxOne.java

D:\OOPS\12-11-2024 practice.java>java RowWithMaxOne
Output for mat1: [0, 1]
Output for mat2: [1, 2]
Output for mat2: [1, 2]
```

Time Complexity : O(N*M)

Space Complexity : O(1)

2. 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.

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.

SOLUTION:

```
import java.util.*;
class Anagrams {
   static boolean areAnagrams(String s1, String s2) {
      HashMap<Character, Integer> charCount = new HashMap<>();
      for (char ch : s1.toCharArray())
         charCount.put(ch, charCount.getOrDefault(ch, 0) + 1);
      for (char ch : s2.toCharArray())
         charCount.put(ch, charCount.getOrDefault(ch, 0) - 1);
      for (var pair : charCount.entrySet()) {
         if (pair.getValue() != 0) {
            return false;
         }
      }
      return true;
   }
   public static void main(String[] args) {
      String s1 = "geeks";
      String s2 = "kseeg";
      System.out.println(areAnagrams(s1, s2) ? "true" : "false");
      System.out.println();
      String s3 = "allergy";
      String s4 = "allergic";
      System.out.println(areAnagrams(s3, s4) ? "true" : "false");
      System.out.println();
      String s5 = "g";
      String s6 = "g";
      System.out.println(areAnagrams(s5, s6) ? "true" : "false");
   }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac Anagrams.java

D:\OOPS\09-11-2024 practice java>java Anagrams
true

false

true
```

Time Complexity: O(N+M)

Space Complexity: O(K)

3. Longest consequtive subsequence

Given an unsorted array of integers nums, return *the length of the longest consecutive elements sequence.*

You must write an algorithm that runs in O(n) time.

Example 1:

Input: nums = [100,4,200,1,3,2]

Output: 4

Explanation: The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its length is 4.

Example 2:

Input: nums = [0,3,7,2,5,8,4,6,0,1]

Output: 9

Constraints:

- $0 <= nums.length <= 10^5$
- $-10^9 <= nums[i] <= 10^9$

SOLUTION:

```
import java.util.*;
class LongConsecutiveSeq {
   public static int longestConsecutive(int[] nums) {
      Set<Integer> s = new HashSet<>();
      if (nums.length < 1)
         return 0;
      for (int num : nums)
         s.add(num);
      int largest = 0;
      for (int num : nums) {
```

```
            if (!s.contains(num - 1)) {
                int val = num;
                int c = 1;
                while (s.contains(val + 1)) {
                    c++;
                    val = val + 1;
                }
                largest = Math.max(largest, c);
            }
        }
        return largest;
    }
    public static void main(String[] args) {
        int[] nums1 = {100, 4, 200, 1, 3, 2};
        int[] nums2 = {0, 3, 7, 2, 5, 8, 4, 6, 0, 1};
        System.out.println("Output for nums1: " + longestConsecutive(nums1));
        System.out.println("Output for nums2: " + longestConsecutive(nums2));
    }
}
```

OUTPUT:

```
D:\OOPS\12-11-2024 practice.java>javac LongConsecutiveSeq.java

D:\OOPS\12-11-2024 practice.java>java LongConsecutiveSeq
Output for nums1: 4
Output for nums2: 9
```

Time Complexity: O(N)

Space Complexity: O(N)

4. **Longest Palindromic Substring**

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.

Examples:

Input: *str = "forgeeksskeegfor"*
Output: *"geeksskeeg"*
Explanation: *There are several possible palindromic substrings like "kssk", "ss", "eeksskee" etc. But the substring "geeksskeeg" is the longest among all.*

Input: *str = “Geeks”*
Output: *“ee”*

Input: *str = “abc”*
Output: *“a”*

Input: *str = “”*
Output: *“”*

SOLUTION:

```
class LongPalindrome {
    public static String longestPalindrome(String s) {
        String ans = "";
        if (s.length() <= 1)
            return s;
        for (int i = 1; i < s.length(); i++) {
// odd palindrome check
            int l = i;
            int h = i;
            while (l >= 0 && h < s.length() && s.charAt(l) == s.charAt(h)) {
                l--;
                h++;
            }
            String n = s.substring(l + 1, h);
            if (n.length() > ans.length())
                ans = n;
// even palindrome check
            l = i - 1;
            h = i;
            while (l >= 0 && h < s.length() && s.charAt(l) == s.charAt(h)) {
                l--;
                h++;
            }
            n = s.substring(l + 1, h);
            if (n.length() > ans.length())
                ans = n;
        }
```

```
        return ans;
    }
    public static void main(String[] args) {
        String s = "babad";
        System.out.println(longestPalindrome(s));
    }
}
```

OUTPUT:

```
D:\OOPS\12-11-2024 practice.java>javac LongPalindrome.java

D:\OOPS\12-11-2024 practice.java>java LongPalindrome
bab
```

Time Complexity:O(N^2)

Space Complexity:O(1)

5. **Rat in a Maze Problem - I**

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.

SOLUTION:

```
import java.util.*;
class Sub{
  private static void solve(int i, int j, int a[][], int n, ArrayList<String> ans, String move, int vis[][], int di[], int dj[])
{
    if (i == n - 1 && j == n - 1) {
      ans.add(move);
      return;
    }
    String dir = "DLRU";
    for (int ind = 0; ind < 4; ind++) {
      int nexti = i + di[ind];
      int nextj = j + dj[ind];
      if (nexti >= 0 && nextj >= 0 && nexti < n && nextj < n && vis[nexti][nextj] == 0 && a[nexti][nextj] == 1) {
        vis[i][j] = 1;
        solve(nexti, nextj, a, n, ans, move + dir.charAt(ind), vis, di, dj);
        vis[i][j] = 0;
      }
    }
  }
  public static ArrayList<String> findPath(int[][] m, int n) {
    int vis[][] = new int[n][n];
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        vis[i][j] = 0;
      }
    }
    int di[] = {+1, 0, 0, -1};
    int dj[] = {0, -1, 1, 0};
    ArrayList<String> ans = new ArrayList<>();
    if (m[0][0] == 1) solve(0, 0, m, n, ans, "", vis, di, dj);
    return ans;
  }
}
class RatInMaze {
  public static void main(String[] args) {
```

```
    int n = 4;
    int[][] a = {{1,0,0,0},{1,1,0,1},{1,1,0,0},{0,1,1,1}};
    Sub obj = new Sub();
    ArrayList<String> res = obj.findPath(a, n);
    if (res.size() > 0) {
      for (int i = 0; i < res.size(); i++)
        System.out.print(res.get(i) + " ");
      System.out.println();
    } else {
      System.out.println(-1);
    }
  }
}
```

OUTPUT:

```
D:\OOPS\12-11-2024 practice.java>javac RatInMaze.java

D:\OOPS\12-11-2024 practice.java>java RatInMaze
DDRDRR DRDDRR
```

Time Complexity: O(m*n)

Space Complexity: O(m*n)