12.11.2024 **Sangeswaran A [Cyber Security]**

ANAGRAM PROBLEM

CODE

```
import java.util.*;
class Anagram {
    public static boolean equalstring(String str1, String str2) {
        str1 = str1.toLowerCase();
        str2 = str2.toLowerCase();
        char[] s1 = str1.toCharArray();
        char[] s2 = str2.toCharArray();
        Arrays.sort(s1);
        Arrays.sort(s2);
        return Arrays.equals(s1, s2);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of strings: ");
        int n = sc.nextInt();
        sc.nextLine();  // Consume the newline character after the integer input
        String[] arr = new String[n];
        System.out.println("Enter the strings:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextLine();
```

```
        }
        boolean allAnagrams = true;
        for (int i = 0; i < n - 1; i++) {
            if (!equalstring(arr[i], arr[i + 1])) {
                allAnagrams = false;
                break;
            }
        }
        if (allAnagrams) {
            System.out.println("All strings are anagrams.");
        } else {
            System.out.println("Not all strings are anagrams.");
        }
        sc.close();
    }
}
```

OUTPUT

```
G:\java programs\hw>java Anagram
Enter the number of strings: 3
Enter the strings:
sanga
ansag
ganas
All strings are anagrams.
```

TIME COMPLEXITY

O(n)

ROW WITH MAX 1's

CODE

```
import java.util.*;
public class RowMostOnes {
    public static int firstRowWithMostOnes(int[][] arr) {
        int n = arr.length;
        int m = arr[0].length;
        int maxOnes = 0;
        int rowIndex = -1;
        for (int i = 0; i < n; i++) {
            int left = 0, right = m - 1;
            int firstOneIndex = m;
            while (left <= right) {
                int mid = left + (right - left) / 2;
                if (arr[i][mid] == 1) {
                    firstOneIndex = mid;
                    right = mid - 1;
                } else {
                    left = mid + 1;
                }
            }
            int onesCount = m - firstOneIndex;
```

```
            if (onesCount > maxOnes) {
                maxOnes = onesCount;
                rowIndex = i;
            }
        }
        return rowIndex;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of rows: ");
        int n = scanner.nextInt();
        System.out.print("Enter the number of columns: ");
        int m = scanner.nextInt();
        int[][] arr = new int[n][m];
        System.out.println("Enter the elements of the array (0 or 1):");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                arr[i][j] = scanner.nextInt();
            }
        }
        scanner.close();
        System.out.println("Row with most 1's: " + firstRowWithMostOnes(arr));
    }
}
```

OUTPUT

G:\java programs\hw>java RowMostOnes

Enter the number of rows: 4

Enter the number of columns: 4

Enter the elements of the array (0 or 1):

0 1 1 1

0 0 1 1

1 1 1 1

0 0 0 0

Row with most 1's: 2

TIME COMPLEXITY

O(n)

LONGEST CONSECUTIVE SUBSEQUENCE

CODE

```
import java.util.*;
public class LongestConsecutive {
    public static int longestConsecutive(int[] arr) {
        if (arr.length == 0) return 0;
        HashSet<Integer> set = new HashSet<>();
        for (int num : arr) {
            set.add(num);
        }
        int longestStreak = 0;
        for (int num : set) {
            if (!set.contains(num - 1)) {
                int currentNum = num;
                int currentStreak = 1;
                while (set.contains(currentNum + 1)) {
                    currentNum++;
                    currentStreak++;
                }
                longestStreak = Math.max(longestStreak, currentStreak);
            }
        }
        return longestStreak;
    }
    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.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();
       System.out.println(longestConsecutive(arr));
    }
}
```

OUTPUT

```
G:\java programs\hw>java LongestConsecutive
Enter the number of elements in the array: 7
Enter the elements of the array: 1 9 3 10 4 20 2
4
```

TIME COMPLEXITY

O(n)

LONGEST PALINDROME

CODE

```
import java.util.*;
public class LongestPalindrome {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String s = scanner.nextLine();
        scanner.close();
        LongestPalindrome solution = new LongestPalindrome();
        String result = solution.longestPalindrome(s);
        System.out.println("Longest palindromic substring: " + result);
    }
    public String longestPalindrome(String s) {
        if (s == null || s.length() <= 1) {
            return s;
        }
        String longest = "";
        for (int i = 0; i < s.length(); i++) {
            String palindrome1 = expandAroundCenter(s, i, i);
            String palindrome2 = expandAroundCenter(s, i, i + 1);
            if (palindrome1.length() > longest.length()) {
                longest = palindrome1;
            }
            if (palindrome2.length() > longest.length()) {
```

```
                longest = palindrome2;
            }
        }
        return longest;
    }
    private String expandAroundCenter(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);
    }
}
```

OUTPUT

```
G:\java programs\hw>java LongestPalindrome
Enter a string: sangaangsa
Longest palindromic substring: aa
```

TIME COMPLEXITY

$O(n^2)$

RAT IN MAZE PROBLEM

CODE

```
import java.util.*;

public class RatInMaze {

    private static void findPaths(int[][] maze, int x, int y, String path, boolean[][] visited,
List<String> paths) {

        int n = maze.length;

        if (x == n - 1 && y == n - 1) {

            paths.add(path);

            return;

        }

        visited[x][y] = true;

        if (x + 1 < n && maze[x + 1][y] == 1 && !visited[x + 1][y]) {

            findPaths(maze, x + 1, y, path + "D", visited, paths);

        }

        if (y - 1 >= 0 && maze[x][y - 1] == 1 && !visited[x][y - 1]) {

            findPaths(maze, x, y - 1, path + "L", visited, paths);

        }

        if (y + 1 < n && maze[x][y + 1] == 1 && !visited[x][y + 1]) {

            findPaths(maze, x, y + 1, path + "R", visited, paths);

        }

        if (x - 1 >= 0 && maze[x - 1][y] == 1 && !visited[x - 1][y]) {

            findPaths(maze, x - 1, y, path + "U", visited, paths);

        }

        visited[x][y] = false;
```

```
}
public static List<String> getAllPaths(int[][] maze) {
    int n = maze.length;
    List<String> paths = new ArrayList<>();
    boolean[][] visited = new boolean[n][n];
    if (maze[0][0] == 1) {
        findPaths(maze, 0, 0, "", visited, paths);
    }
    return paths;
}
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[][] maze = new int[n][n];
    System.out.println("Enter the matrix (0 for blocked, 1 for open):");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            maze[i][j] = scanner.nextInt();
        }
    }
    List<String> paths = getAllPaths(maze);
    if (paths.isEmpty()) {
        System.out.println("-1");
    } else {
```

```
            paths.sort(null);
            for (String path : paths) {
                System.out.print(path + " ");
            }
        }
    }
}
```

OUTPUT

```
G:\java programs\hw>java RatInMaze
Enter the size of the matrix (n): 4
Enter the matrix (0 for blocked, 1 for open):
1 0 0 0
1 1 0 1
1 1 0 0
0 1 1 1
DDRDRR DRDDRR
```

TIME COMPLEXITY

O(n)