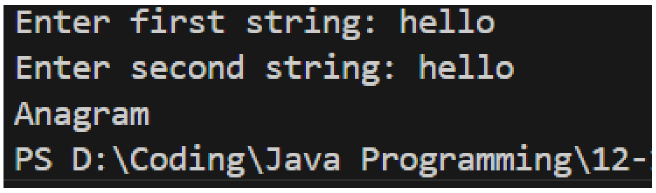
DSA Practice - 12-11-2024 **JENIFA S JOANNA**

1.Anagram Program

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

public class Anagram {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter first string: ");
        String str1 = sc.nextLine();
        System.out.print("Enter second string: ");
        String str2 = sc.nextLine();

        if (str1.length() != str2.length()) {
            System.out.println("Not an Anagram");
        } else {
            char[] arr1 = str1.toCharArray();
            char[] arr2 = str2.toCharArray();
            Arrays.sort(arr1);
            Arrays.sort(arr2);
            if (Arrays.equals(arr1, arr2)) {
                System.out.println("Anagram");
            } else {
                System.out.println("Not an Anagram");
            }
        }
    }
}
```

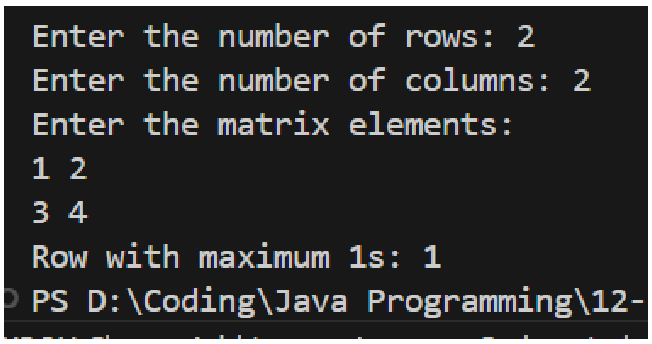


```
Enter first string: hello
Enter second string: hello
Anagram
PS D:\Coding\Java Programming\12-
```

Time Complexity: O(n)

2. Rows with maximum 1's

```
import java.util.Scanner;

public class MaxOnesRow {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of rows: ");
        int n = sc.nextInt();
        System.out.print("Enter the number of columns: ");
        int m = sc.nextInt();
        int[][] matrix = new int[n][m];
        System.out.println("Enter the matrix elements:");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                matrix[i][j] = sc.nextInt();
            }
        }

        int rowIndex = -1;
        int maxCount = 0;

        for (int i = 0; i < n; i++) {
            int count = 0;
            for (int j = 0; j < m; j++) {
                if (matrix[i][j] == 1) {
                    count++;
                }
            }
            if (count > maxCount) {
                maxCount = count;
                rowIndex = i;
            }
        }

        System.out.println("Row with maximum 1s: " + (rowIndex + 1));
    }
}
```

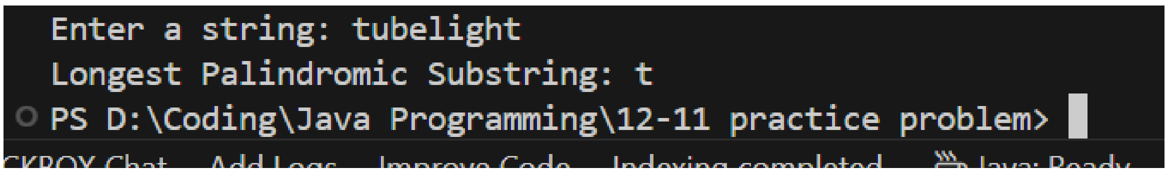


```
Enter the number of rows: 2
Enter the number of columns: 2
Enter the matrix elements:
1 2
3 4
Row with maximum 1s: 1
PS D:\Coding\Java Programming\12-
```

Time Complexity: **O(n+m)**

3.Longest consequtive subsequence

```
import java.util.Arrays;
import java.util.HashSet;

public class LongestConsecutiveSubsequence {
    public static void main(String[] args) {
        int[] arr = {100, 4, 200, 1, 3, 2};
        HashSet<Integer> set = new HashSet<>();
        for (int num : arr) {
            set.add(num);
        }

        int longestStreak = 0;
        for (int num : arr) {
            if (!set.contains(num - 1)) {
                int currentNum = num;
                int currentStreak = 1;
                while (set.contains(currentNum + 1)) {
                    currentNum += 1;
                    currentStreak += 1;
                }
                longestStreak = Math.max(longestStreak, currentStreak);
            }
        }

        System.out.println("Longest Consecutive Subsequence length: " + longestStreak);
    }
}
```

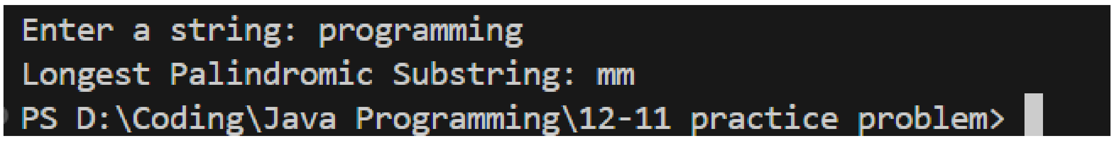


```
Enter a string: tubelight
Longest Palindromic Substring: t
PS D:\Coding\Java Programming\12-11 practice problem>
CKBOX Chat    Add Logs    Improve Code    Indexing completed    Java: Ready
```

Time Complexity: **O(n)**

4.Longest palindrome in a string

```
import java.util.Scanner;

public class LongestPalindrome {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String str = sc.nextLine();
        String result = "";

        for (int i = 0; i < str.length(); i++) {
            for (int j = i + 1; j <= str.length(); j++) {
                String subStr = str.substring(i, j);
                String reversed = new StringBuilder(subStr).reverse().toString();
                if (subStr.equals(reversed) && subStr.length() > result.length()) {
                    result = subStr;
                }
            }
        }

        System.out.println("Longest Palindromic Substring: " + result);
    }
}
```

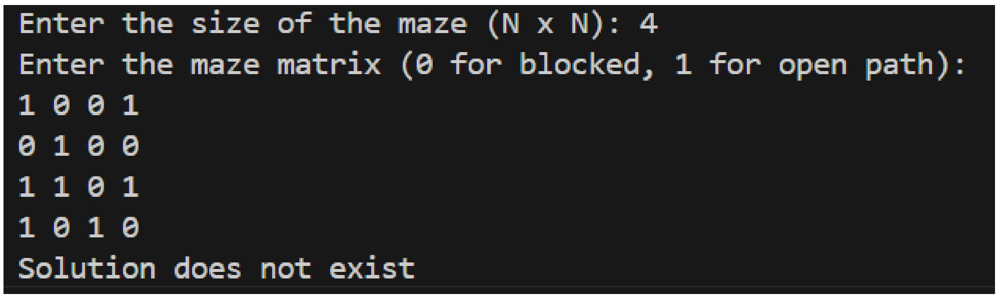


```
Enter a string: programming
Longest Palindromic Substring: mm
PS D:\Coding\Java Programming\12-11 practice problem>
```

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

5. Rat In Maze

```
import java.util.Scanner;

public class RatInMaze {
    static int N;

    public static boolean isSafe(int maze[][], int x, int y) {
        return (x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1);
    }

    public static boolean solveMaze(int maze[][], int x, int y, int sol[][]) {
        if (x == N - 1 && y == N - 1 && maze[x][y] == 1) {
            sol[x][y] = 1;
            return true;
        }
        if (isSafe(maze, x, y)) {
            sol[x][y] = 1;
            if (solveMaze(maze, x + 1, y, sol))
                return true;
            if (solveMaze(maze, x, y + 1, sol))
                return true;
            sol[x][y] = 0;
            return false;
        }
        return false;
    }

    public static void printSolution(int sol[][]) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                System.out.print(" " + sol[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of the maze (N x N): ");
        N = sc.nextInt();
        int maze[][] = new int[N][N];
        int sol[][] = new int[N][N];

        System.out.println("Enter the maze matrix (0 for blocked, 1 for open path):");
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                maze[i][j] = sc.nextInt();
            }
        }

        if (!solveMaze(maze, 0, 0, sol)) {
            System.out.println("Solution does not exist");
        } else {
            printSolution(sol);
        }
    }
}
```



```
Enter the size of the maze (N x N): 4
Enter the maze matrix (0 for blocked, 1 for open path):
1 0 0 1
0 1 0 0
1 1 0 1
1 0 1 0
Solution does not exist
```

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