**1.Anagrams**

**CODE:**

import java.util.\*;

public class Anagram {

    boolean areAnagrams(String s1, String s2) {

        HashMap<Character,Integer> h1=new HashMap<Character,Integer>();

        HashMap<Character,Integer> h2=new HashMap<Character,Integer>();

        int n1=s1.length();

        int n2=s2.length();

        if(n1!=n2){

            return false;

        }

        for(int i=0;i<n1;i++){

            h1.put(s1.charAt(i),h1.getOrDefault(s1.charAt(i),0)+1);

        }

        for(int i=0;i<n2;i++){

            h2.put(s2.charAt(i),h2.getOrDefault(s2.charAt(i),0)+1);

        }

        return h1.equals(h2);

    }

    public static void main(String args[]){

        Anagram an=new Anagram();

        String s1="geeks";

        String s2="keegs";

        String s3="apple";

        String s4="appple";

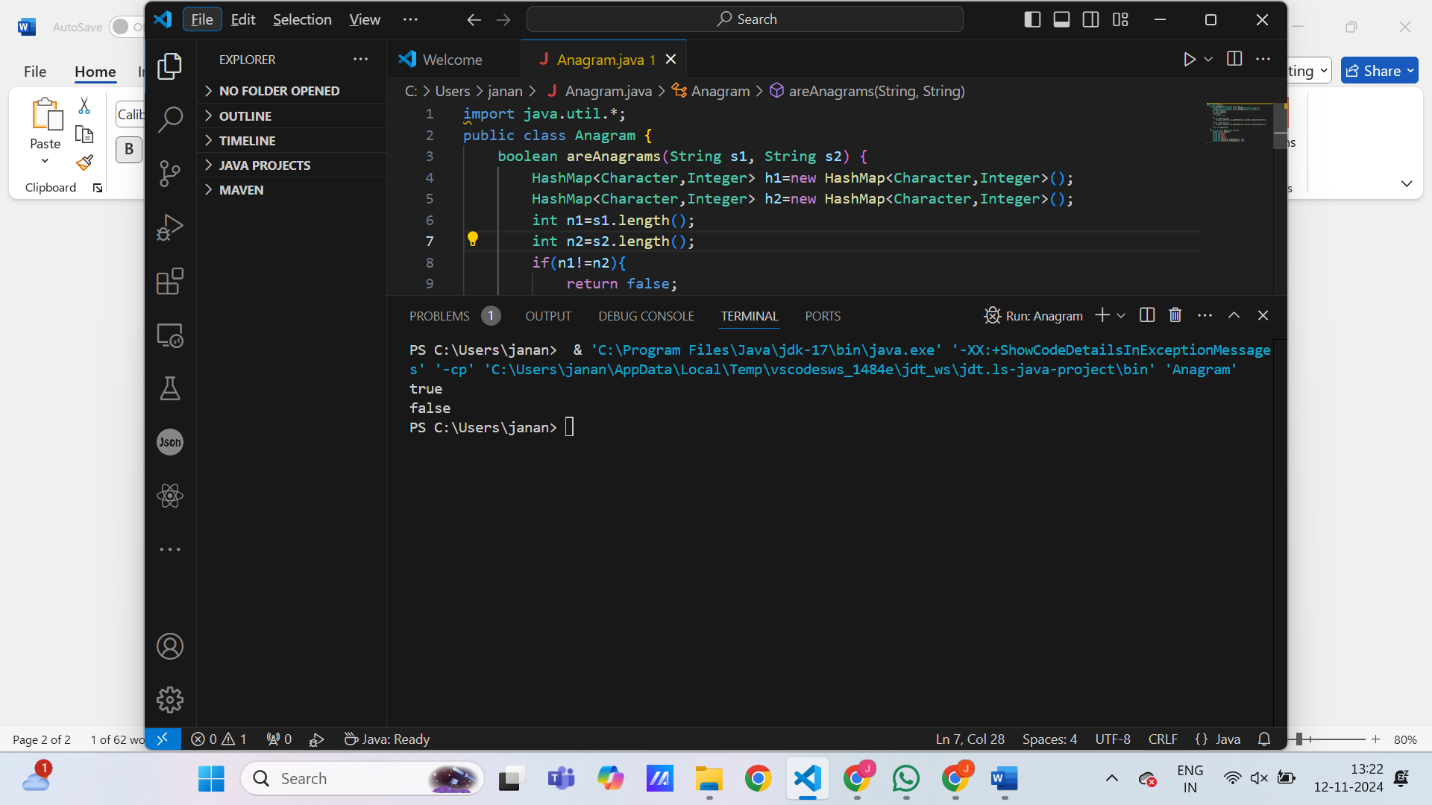
        System.out.println(an.areAnagrams(s1, s2));

        System.out.println(an.areAnagrams(s3, s4));

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(n) **SPACE COMPLEXITY:** O(n)

**2.Row with Max 1s:**

**CODE:**

public class RowsWithMax1s{

    int rowWithMax1s(int arr[][]) {

        int n=arr.length;

        int m=arr[0].length-1;

        int ans=-1;

        for(int i=0;i<n;i++){

            while(m>=0 && arr[i][m]==1){

                m--;

                ans=i;

            }

        }

        return ans;

    }

    public static void main(String[] args) {

        RowsWithMax1s r=new RowsWithMax1s();

        int arr1[][]={{0, 1, 1, 1},{0, 0, 1, 1},{1, 1, 1, 1},{0, 0, 0, 0}};

        int arr2[][]={{0,0},{1,1}};

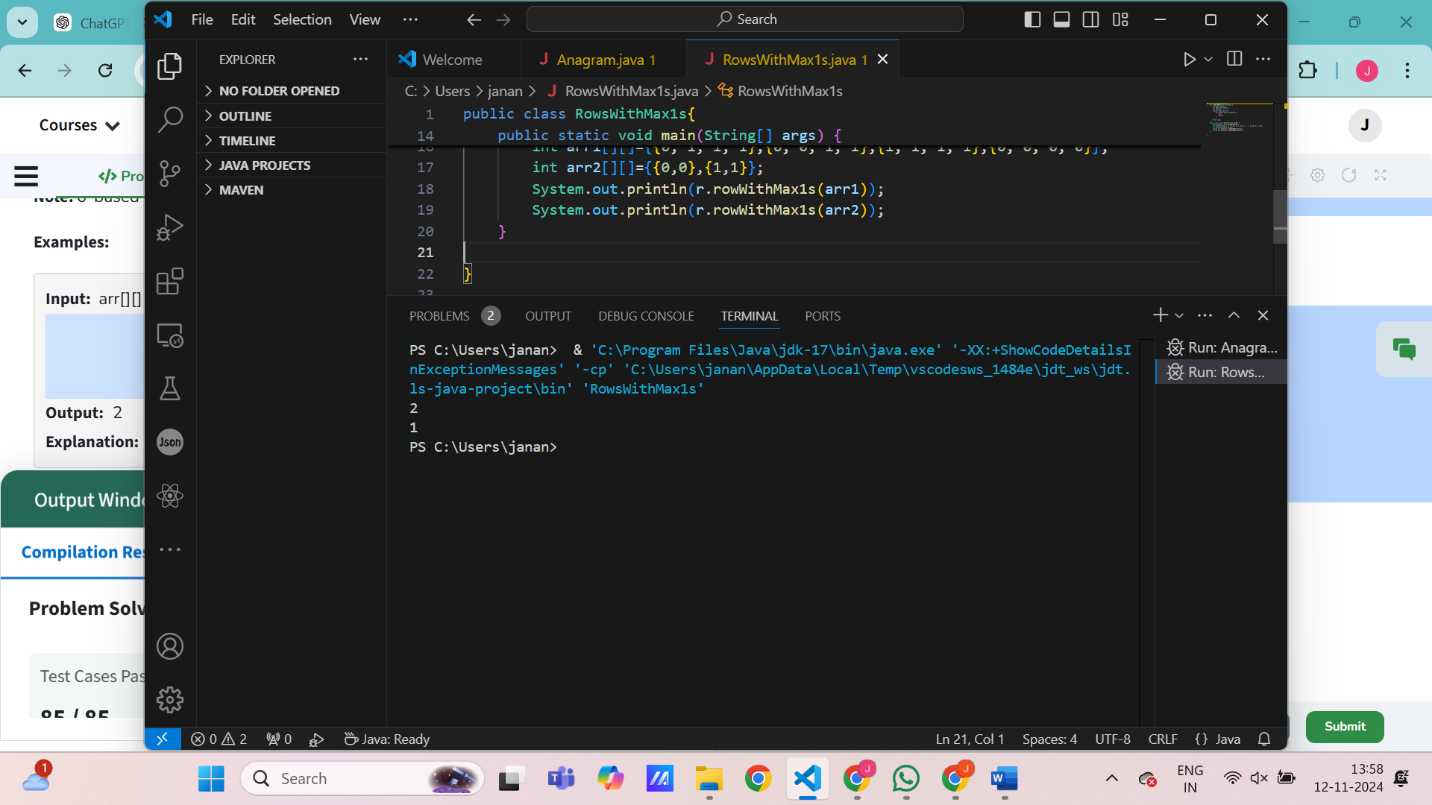
        System.out.println(r.rowWithMax1s(arr1));

        System.out.println(r.rowWithMax1s(arr2));

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(n + m) **SPACE COMPLEXITY:** O(1)

**3. Longest consecutive subsequence:**

**CODE:**

import java.util.\*;

public class ConSeq {

    int findLongestConseqSubseq(int[] arr) {

        HashSet<Integer> h=new HashSet<>();

        int n=arr.length;

        int max=-1;

        for(int i=0;i<n;i++){

            h.add(arr[i]);

        }

        for (int e : h) {

            if (!h.contains(e -1)) {

              int j=e;

              int c=1;

                while(h.contains(j+1))

                {

                j++;

                c++;

                }

                max=Math.max(max,c);

            }

        }

    return max;

    }

    public static void main(String[] args) {

        ConSeq p=new ConSeq();

        int[] arr1={1, 9, 3, 10, 4, 20, 2};

        int[] arr2={15, 13, 12, 14, 11, 10, 9};

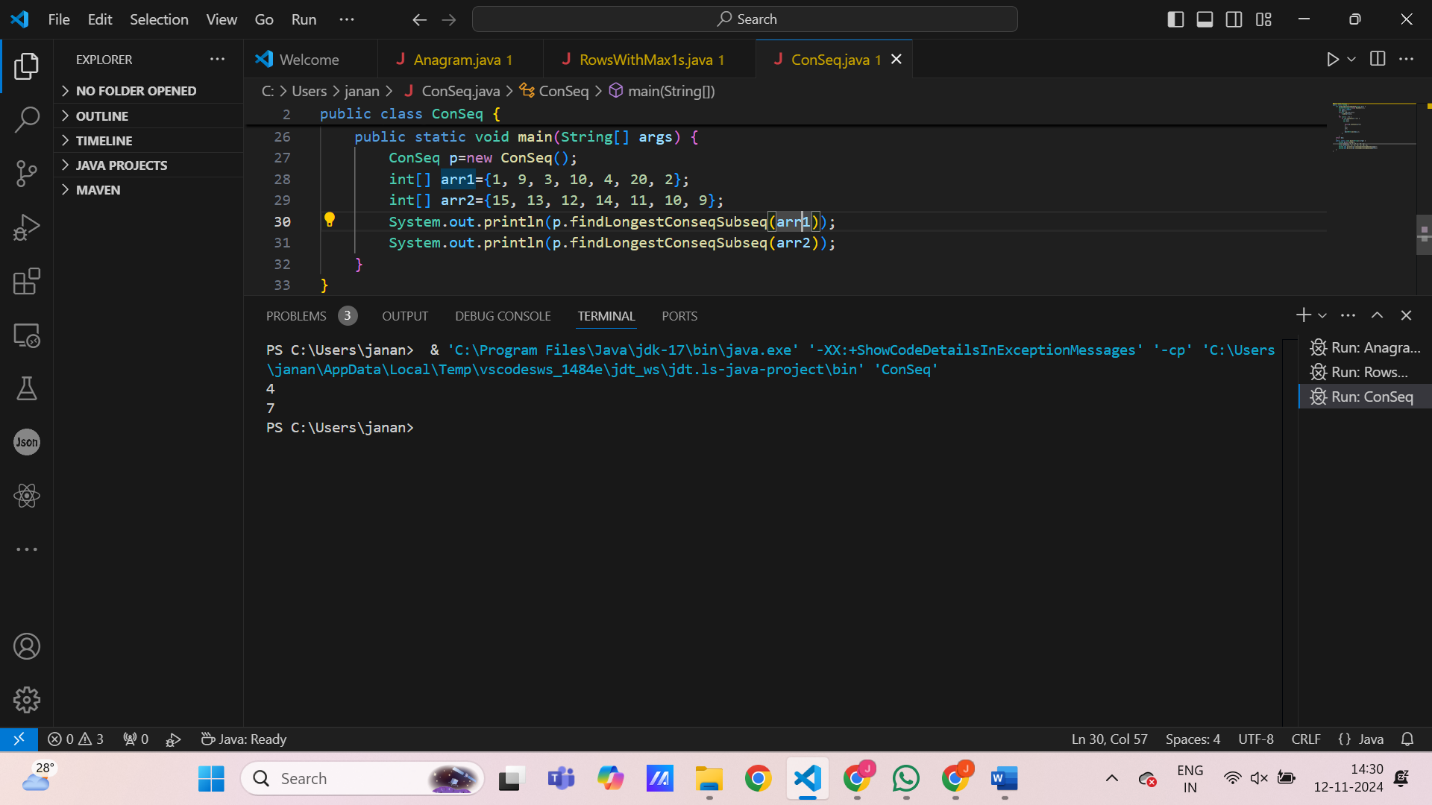
        System.out.println(p.findLongestConseqSubseq(arr1));

        System.out.println(p.findLongestConseqSubseq(arr2));

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(n) **SPACE COMPLEXITY:** O(n)

**4.** **Longest palindrome in a string**

**CODE:**

public class PalindromeString{

    public String longestPalindrome(String s) {

        if (s == null || s.length() < 1) return "";

        String longest = "";

        for (int i = 0; i < s.length(); i++) {

            String oddPalindrome = expandAroundCenter(s, i, i);

            String evenPalindrome = expandAroundCenter(s, i, i + 1);

            String currentLongest = (oddPalindrome.length() > evenPalindrome.length()) ? oddPalindrome : evenPalindrome;

            if (currentLongest.length() > longest.length()) {

                longest = currentLongest;

            }

        }

        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);

    }

    public static void main(String[] args) {

       PalindromeString s = new PalindromeString();

        String s1 = "aaaabbaa";

        String s2 = "abc";

        String s3 = "abacdfgdcaba";

        System.out.println(s.longestPalindrome(s1));

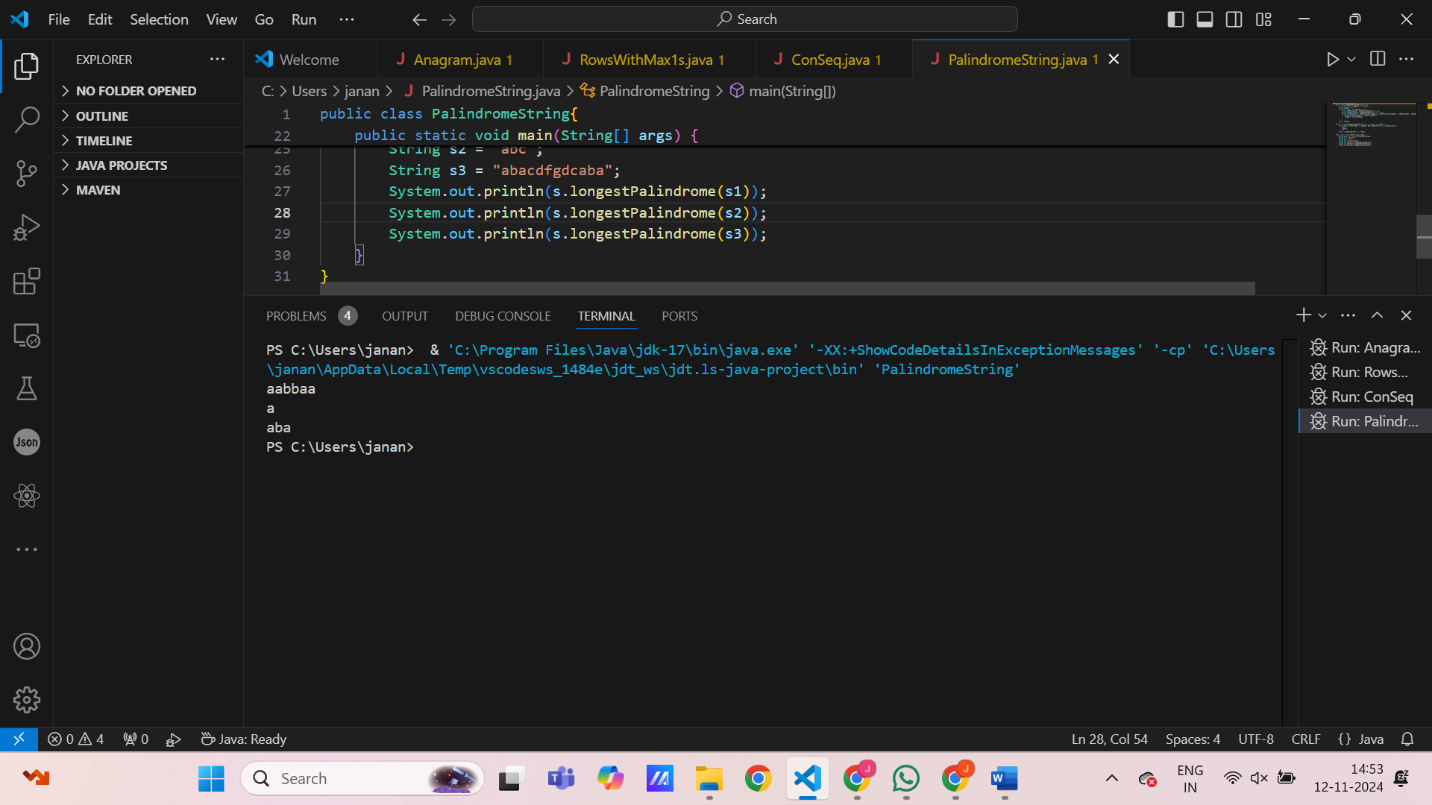
        System.out.println(s.longestPalindrome(s2));

        System.out.println(s.longestPalindrome(s3));

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(n^2) **SPACE COMPLEXITY:** O(1)

**5.Rat in a Maze:**

**CODE:**

import java.util.\*;

public class RatInMaze {

    public static ArrayList<String> findPath(int[][] mat) {

        ArrayList<String> result = new ArrayList<>();

        int n = mat.length;

        if (mat[0][0] == 0 || mat[n - 1][n - 1] == 0) return result;

        boolean[][] visited = new boolean[n][n];

        helper(mat, n, 0, 0, "", result, visited);

        Collections.sort(result);

        return result;

    }

    private static void helper(int[][] mat, int n, int row, int col, String path, ArrayList<String> result, boolean[][] visited) {

        if (row == n - 1 && col == n - 1) {

            result.add(path);

            return;

        }

        if (row < 0 || col < 0 || row >= n || col >= n || mat[row][col] == 0 || visited[row][col]) return;

        visited[row][col] = true;

        helper(mat, n, row + 1, col, path + 'D', result, visited);

        helper(mat, n, row - 1, col, path + 'U', result, visited);

        helper(mat, n, row, col + 1, path + 'R', result, visited);

        helper(mat, n, row, col - 1, path + 'L', result, visited);

        visited[row][col] = false;

    }

    public static void main(String args[]) {

        int[][] mat1 = {{1, 0, 0, 0}, {1, 1, 0, 1}, {1, 1, 0, 0}, {0, 1, 1, 1}};

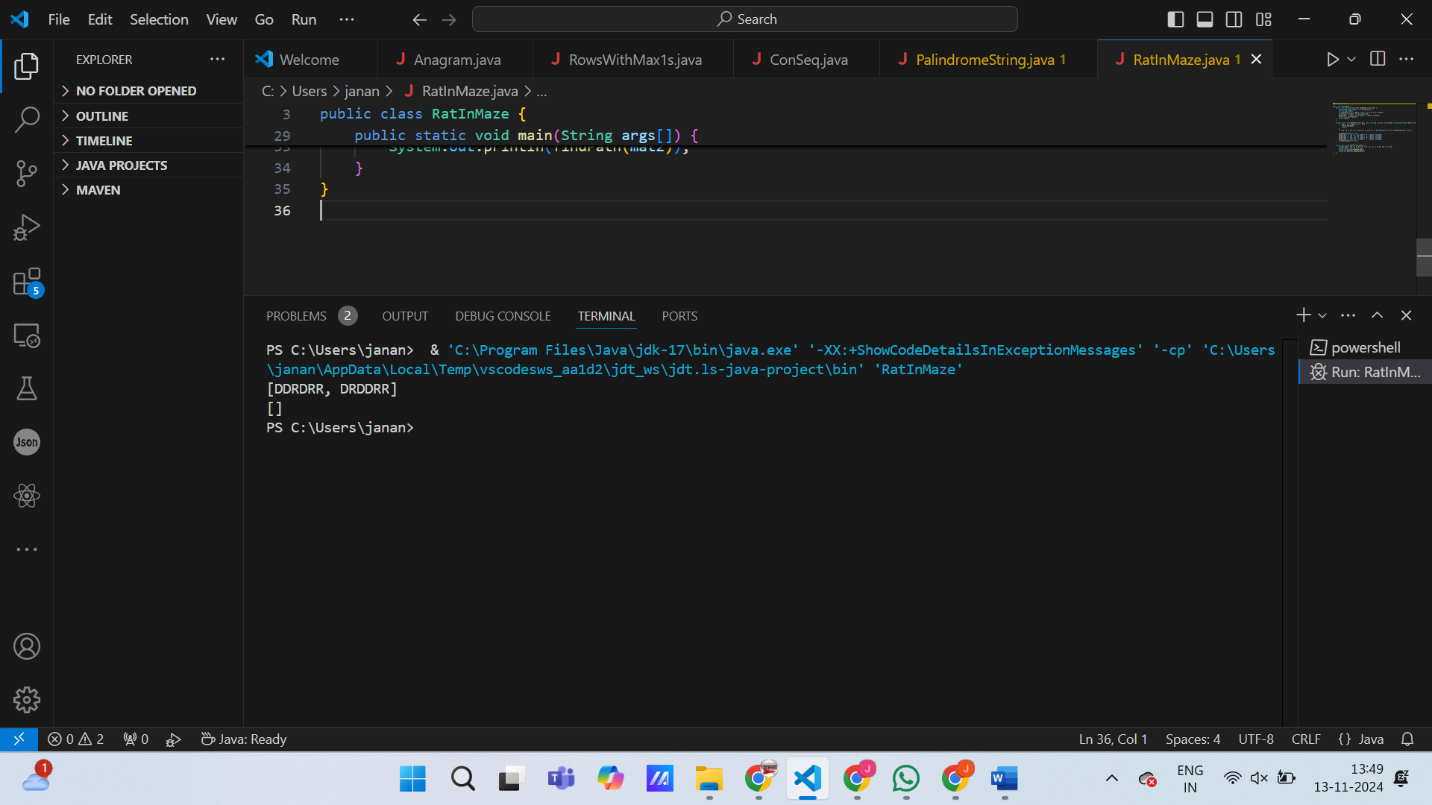
        int[][] mat2 = {{1, 0}, {1, 0}};

        System.out.println(findPath(mat1));

        System.out.println(findPath(mat2));

    }}

**OUTPUT:**



**TIME COMPLEXITY:** O(3n^2)

**SPACE COMPLEXITY:** O(l\*x)