**Name: Abishek S H Date: 12/11/23**

**Dept: CSE**

**1.Anagram Program**

**Code:**

package sample3;

import java.util.\*;

public class anagram {

static boolean check(String s1,String s2) {

int[] arr=new int[26];

for(int i=0;i<s1.length();i++) {

arr[s1.charAt(i)-'a']++;

}

for(int j=0;j<s2.length();j++) {

arr[s2.charAt(j)-'a']--;

}

for(int i:arr) {

if(i==1) {

return false;

}

}

return true;

}

public static void main(String[] args) {

String s1="hello";

String s2="ohell";

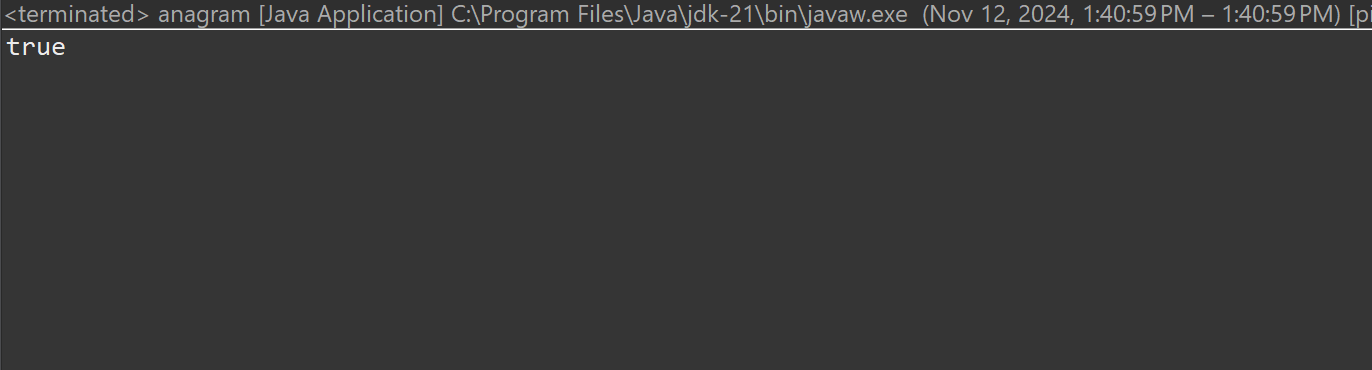
boolean res=*check*(s1,s2);

System.***out***.println(res);

}

}

**Output:**

****

**Time Complexity:** O(n)

**2.Row with Maximum 1’s**

**Code:**

package sample3;

import java.util.\*;

public class max1s {

static int check(int[][] arr) {

int ind=-1;

int max=0;

boolean flag=true;

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

int count=0;

for(int j=0;j<arr[i].length;j++) {

if(arr[i][j]==1) {

count++;

}

}

if(count>max) {

max=count;

ind=i;

}

}

return ind;

}

public static void main(String[] args) {

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

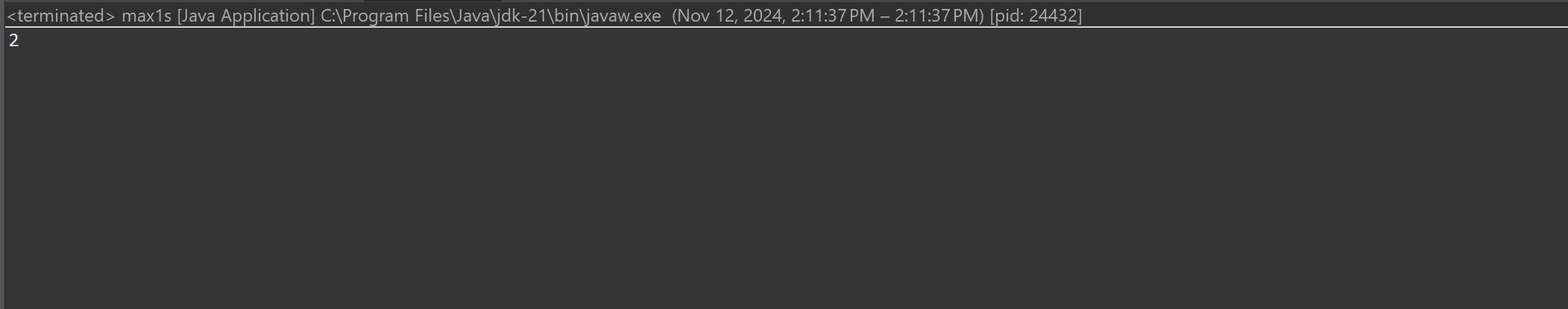
int res=*check*(ar);

System.***out***.println(res);

}

}

**Output:**

****

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

**3. Longest consecutive subsequence**

**Longest Consecutive subsequence:**

package sample3;

import java.util.\*;

public class LongestSubSeq {

static int findLongestConseqSubseq(int arr[], int n)

{

Arrays.*sort*(arr);

int ans = 0, count = 0;

ArrayList<Integer> v = new ArrayList<Integer>();

v.add(10);

for (int i = 1; i < n; i++) {

if (arr[i] != arr[i - 1])

v.add(arr[i]);

}

for (int i = 0; i < v.size(); i++) {

if (i > 0 && v.get(i) == v.get(i - 1) + 1)

count++;

else

count = 1;

ans = Math.*max*(ans, count);

}

return ans;

}

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 "

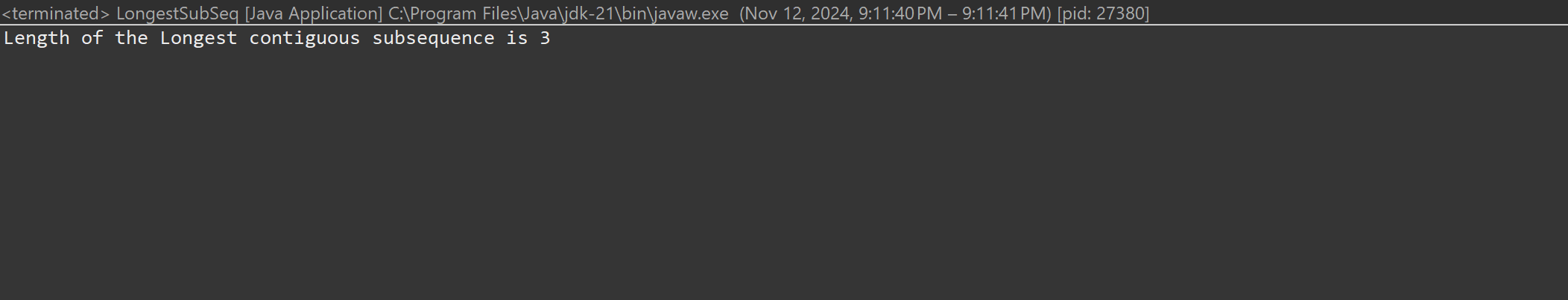
+ "contiguous subsequence is "

+ *findLongestConseqSubseq*(arr, n));

}

}

**Output:**

****

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

**4.** **longest palindrome in a string**

**Code:**

package sample1;

import java.util.\*;

public class longPalindrome {

static String longestPal(String s) {

int n = s.length();

if (n == 0) return "";

int start = 0, maxLen = 1;

for (int i = 0; i < n; i++) {

for (int j = 0; j <= 1; j++) {

int low = i;

int hi = i + j;

while (low >= 0 && hi < n && s.charAt(low) == s.charAt(hi)) {

int currLen = hi - low + 1;

if (currLen > maxLen) {

start = low;

maxLen = currLen;

}

low--;

hi++;

}

}

}

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

}

public static void main(String[] args) {

String str1 = "forgeeksskeegfor";

System.***out***.println("Longest Palindromic Substring: " + *longestPal*(str1));

String str2 = "Geeks";

System.***out***.println("Longest Palindromic Substring: " + *longestPal*(str2));

String str3 = "abc";

System.***out***.println("Longest Palindromic Substring: " + *longestPal*(str3));

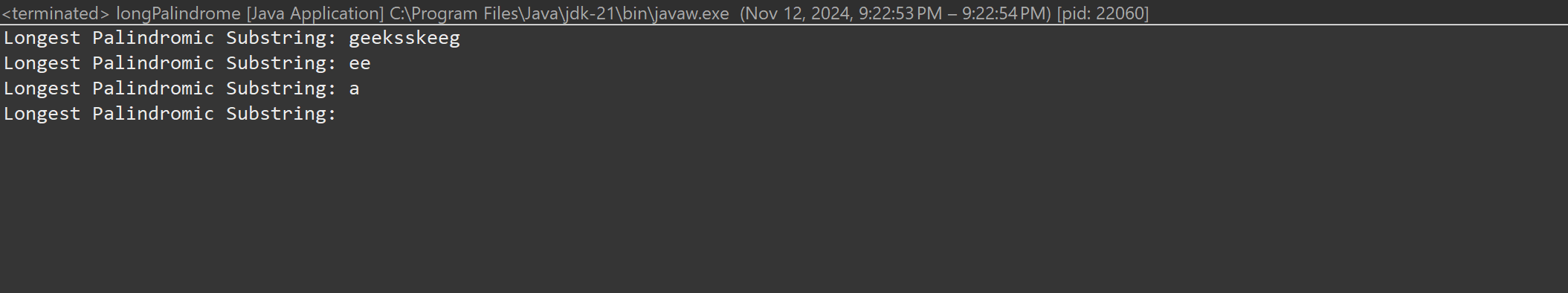
String str4 = "";

System.***out***.println("Longest Palindromic Substring: " + *longestPal*(str4));

}

}

**Output:**



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

**5. rat in a maze problem**

**Code:**

package sample1;

import java.util.\*;

public class maze {

static String *direction* = "DLRU";

static int[] *dr* = { 1, 0, 0, -1 };

static int[] *dc* = { 0, -1, 1, 0 };

static boolean isValid(int row, int col, int n,

int[][] maze)

{

return row >= 0 && col >= 0 && row < n && col < n

&& maze[row][col] == 1;

}

static void findPath(int row, int col, int[][] maze,

int n, ArrayList<String> ans,

StringBuilder currentPath)

{

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

ans.add(currentPath.toString());

return;

}

maze[row][col] = 0;

for (int i = 0; i < 4; i++) {

int nextrow = row + *dr*[i];

int nextcol = col + *dc*[i];

if (*isValid*(nextrow, nextcol, n, maze)) {

currentPath.append(*direction*.charAt(i));

*findPath*(nextrow, nextcol, maze, n, ans,

currentPath);

currentPath.deleteCharAt(

currentPath.length() - 1);

}

}

maze[row][col] = 1;

}

public static void main(String[] args) {

int[][] maze = { { 1, 0, 0, 0 },

{ 1, 1, 0, 1 },

{ 1, 1, 0, 0 },

{ 0, 1, 1, 1 } };

int n = maze.length;

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

StringBuilder currentPath = new StringBuilder();

if (maze[0][0] != 0 && maze[n - 1][n - 1] != 0) {

*findPath*(0, 0, maze, n, result, currentPath);

}

if (result.size() == 0)

System.***out***.println(-1);

else

for (String path : result)

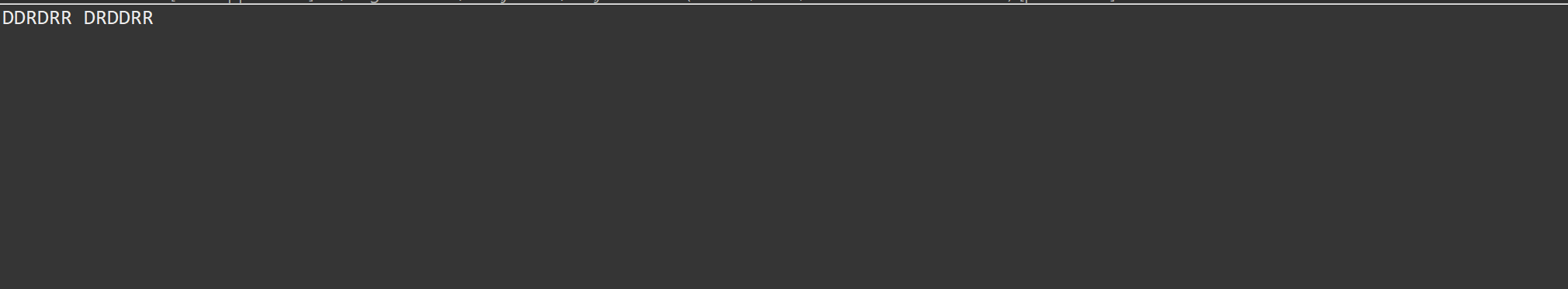
System.***out***.print(path + " ");

System.***out***.println();

}

}

**Output:**

****

**Time Complexity:** O(4^(n\*n))