**PRACTICE SET – 3**

Meena Rhoshini C

Nov 12, 24

1. **Anagram**

class Anagram {

public static void main(String[] args) {

String s1 = "geeks", s2 = "kseeg";

System.out.println(isAnagram(s1, s2));

String s3 = "allergy", s4 = "allergic";

System.out.println(isAnagram(s3, s4));

String s5 = "g", s6 = "g";

System.out.println(isAnagram(s5, s6));

}

static boolean isAnagram(String s1, String s2) {

if (s1.length() != s2.length()) return false;

int[] count = new int[26];

for (char c : s1.toCharArray()) count[c - 'a']++;

for (char c : s2.toCharArray()) {

if (--count[c - 'a'] < 0) return false;

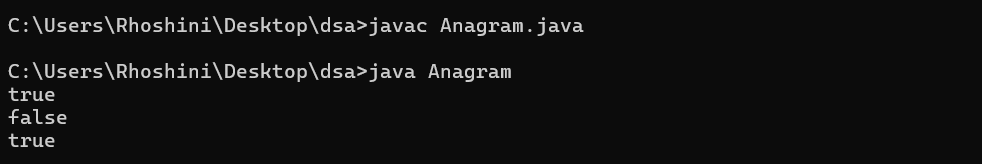
}

return true;

}

}

OUTPUT:



Time Complexity: O(n)  
Space Complexity: O(1)

1. **Row with max 1s**

class RowWithMax1s {

public static void main(String[] args) {

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

System.out.println(rowWithMax1s(arr1, 4, 4));

int[][] arr2 = {{0, 0}, {1, 1}};

System.out.println(rowWithMax1s(arr2, 2, 2));

}

static int rowWithMax1s(int[][] arr, int n, int m) {

int maxRow = -1, j = m - 1;

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

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

j--;

maxRow = i;

}

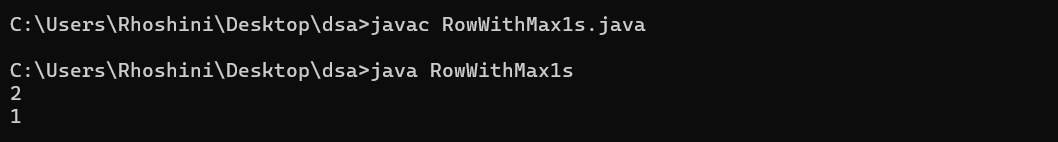
}

return maxRow;

}

}

OUTPUT:



Time Complexity: O(n + m)  
Space Complexity: O(1)

1. **Longest Consecutive Subsequence**

import java.util.HashSet;

class LongestConsecutiveSubsequence {

public static void main(String[] args) {

int[] arr1 = {2, 6, 1, 9, 4, 5, 3};

System.out.println(longestConseqSubseq(arr1, arr1.length));

int[] arr2 = {1, 9, 3, 10, 4, 20, 2};

System.out.println(longestConseqSubseq(arr2, arr2.length));

int[] arr3 = {15, 13, 12, 14, 11, 10, 9};

System.out.println(longestConseqSubseq(arr3, arr3.length));

}

static int longestConseqSubseq(int[] arr, int n) {

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

for (int num : arr) set.add(num);

int maxLen = 0;

for (int num : arr) {

if (!set.contains(num - 1)) {

int currNum = num;

int currLen = 1;

while (set.contains(currNum + 1)) {

currNum++;

currLen++;

}

maxLen = Math.max(maxLen, currLen);

}

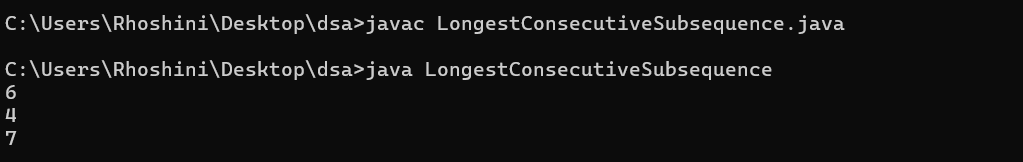
}

return maxLen;

}

}

OUTPUT:



Time Complexity: O(n)  
Space Complexity: O(n)

1. **Longest Palindrome Substring**

import java.util.\*;

class LongestPalindromeSubstring {

public static String longestPalindrome(String s) {

if (s.length() == 0) return "";

int start = 0, maxLength = 1;

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

int len1 = expandAroundCenter(s, i, i);

int len2 = expandAroundCenter(s, i, i + 1);

int len = Math.max(len1, len2);

if (len > maxLength) {

maxLength = len;

start = i - (len - 1) / 2;

}

}

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

}

private static int expandAroundCenter(String s, int left, int right) {

while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {

left--;

right++;

}

return right - left - 1;

}

public static void main(String[] args) {

String s1 = "aaaabbaa";

String s2 = "abc";

String s3 = "abacdfgdcaba";

String s4 = "racecar";

String s5 = "a";

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

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

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

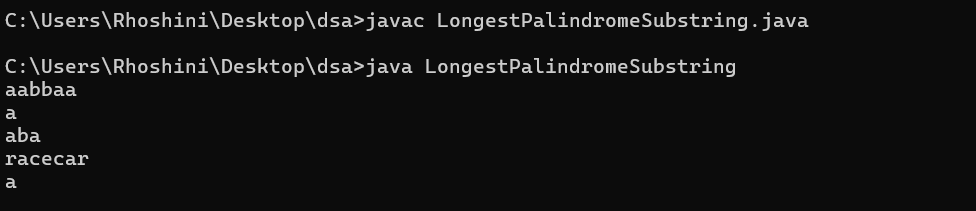
System.out.println(longestPalindrome(s4));

System.out.println(longestPalindrome(s5));

}

}

OUTPUT:

****

Time Complexity: O(n^2)  
Space Complexity: O(1)

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

import java.util.\*;

class RatInMaze {

public static boolean isSafe(int[][] mat, boolean[][] visited, int x, int y, int N) {

return (x >= 0 && x < N && y >= 0 && y < N && mat[x][y] == 1 && !visited[x][y]);

}

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

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

paths.add(path);

return;

}

visited[x][y] = true;

if (isSafe(mat, visited, x + 1, y, N)) {

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

}

if (isSafe(mat, visited, x - 1, y, N)) {

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

}

if (isSafe(mat, visited, x, y + 1, N)) {

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

}

if (isSafe(mat, visited, x, y - 1, N)) {

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

}

visited[x][y] = false;

}

public static List<String> getAllPaths(int[][] mat) {

int N = mat.length; // Dynamically set N based on the matrix size

List<String> paths = new ArrayList<>();

boolean[][] visited = new boolean[N][N];

if (mat[0][0] == 1) {

findPaths(mat, 0, 0, visited, "", paths, N);

}

Collections.sort(paths);

return paths.isEmpty() ? Arrays.asList("-1") : paths;

}

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}};

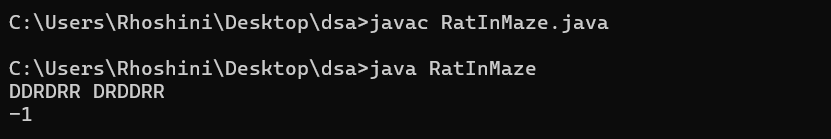
System.out.println(String.join(" ", getAllPaths(mat1)));

System.out.println(String.join(" ", getAllPaths(mat2)));

}

}

OUTPUT:



Time Complexity: O(3^n^2)  
Space Complexity: O(l \* x)