GRAPHS ASSIGNMENT

20BD1A1290-Jakkanagari Harika

1. Given a directed acyclic graph (DAG), implement a function to find the longest path between any two vertices in the graph. Example: Input: graph = [[1,2],[2,3],[3,4],[4,5],[5,6],[7,6]] Output: 6

Solution:

import java.util.\*;

public class Solution{

public static int length = 0;

public static void dfs(int u,int[] vis,int[][] adj,int v,int e,int val){

length = Math.max(length,val);

vis[u] = 1;

for(int i=0;i<adj[u].length;i++){

if(adj[u][i] == 1 && vis[i] == 0) dfs(i,vis,adj,u,e,val+1);

}

}

public static void helper(int v,int e,int[][] edges){

int[][] adj = new int[v+1][v+1];

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

adj[edges[i][0]][edges[i][1]] = 1;

adj[edges[i][1]][edges[i][0]] = 1;

}

int length = 0;

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

int[] vis = new int[v+1];

Arrays.fill(vis,0);

vis[i] = 1;

int val = 0 ;

dfs(i,vis,adj,v,e,val);

// length = Math.max(length,val);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int v = sc.nextInt();

int e = sc.nextInt();

int[][] edges = new int[e][2];

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

edges[i][0] = sc.nextInt();

edges[i][1] = sc.nextInt();

}

helper(v,e,edges);

System.out.println(length);

}

}

PROBLEM-2

import java.util.\*;

public class Solution {

public static int swap(int[] arr,int n){

List<int[]> list = new ArrayList<>();

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

list.add(new int[]{arr[i],i});

}

Collections.sort(list,(a,b)->a[0]-b[0]);

System.out.println(list);

return 0;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] arr = new int[n];

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

arr[i] = sc.nextInt();

}

System.out.println(swap(arr,n));

}

}

PROBLEM-3

class Solution {

public void dfs(Node node , Node copy , Node[] visited){

visited[copy.val] = copy;

for(Node n : node.neighbors){

if(visited[n.val] == null){

Node newNode = new Node(n.val);

copy.neighbors.add(newNode);

dfs(n , newNode , visited);

}else{

copy.neighbors.add(visited[n.val]);

}

}

}

public Node cloneGraph(Node node) {

if(node == null) return null;

Node copy = new Node(node.val);

Node[] visited = new Node[101];

Arrays.fill(visited , null);

dfs(node , copy , visited);

return copy;

}

}

PROBLEM-4

class Islands {

static final int ROW = 5, COL = 5;

boolean isSafe(int M[][], int row, int col,

boolean visited[][])

{

return (row >= 0) && (row < ROW) && (col >= 0)

&& (col < COL)

&& (M[row][col] == 1 && !visited[row][col]);

}

void DFS(int M[][], int row, int col,

boolean visited[][])

{

int rowNbr[]= new int[] { -1, -1, -1, 0, 0, 1, 1, 1 };

int colNbr[]= new int[] { -1, 0, 1, -1, 1, -1, 0, 1 };

visited[row][col] = true;

for (int k = 0; k < 8; ++k)

if (isSafe(M, row + rowNbr[k], col + colNbr[k],

visited))

DFS(M, row + rowNbr[k], col + colNbr[k],

visited);

}

int countIslands(int M[][])

{

boolean visited[][] = new boolean[ROW][COL];

int count = 0;

for (int i = 0; i < ROW; ++i)

for (int j = 0; j < COL; ++j)

if (M[i][j] == 1

&& !visited[i][j]) // If a cell with

{

DFS(M, i, j, visited);

++count;

}

return count;

}

}

PROBLEM-5

class Solution {

public int kthSmallest(int[][] matrix, int k) {

int n = matrix.length;

int [] arr = new int[n\*n];

int idx = 0;

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

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

arr[idx++] = matrix[i][j];

}

}

Arrays.sort(arr);

return arr[k - 1];

}

}

PROBLEM-6

public class GridTraversal {

public int uniquePaths(int[][] grid) {

int m = grid.length;

int n = grid[0].length;

int[][] dp = new int[m][n]; // create a dynamic programming table

dp[0][0] = grid[0][0];

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

dp[0][j] = dp[0][j - 1] + grid[0][j];

}

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

dp[i][0] = dp[i - 1][0] + grid[i][0];

}

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

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

dp[i][j] = Math.min(dp[i - 1][j], dp[i][j - 1]) + grid[i][j];

}

}

return dp[m - 1][n - 1];

}

}

PROBLEM-7

class Solution {

public List<Integer> findMinHeightTrees(int n, int[][] edges) {

List<Integer> result = new ArrayList<>();

if(n <= 0 || edges == null) return result;

if(n == 1){

result.add(0);

return result;

}

Map<Integer, List<Integer>> map = new HashMap<>();

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

map.put(i, new ArrayList<Integer>());

for(int[] arr: edges){

map.get(arr[0]).add(arr[1]);

map.get(arr[1]).add(arr[0]);

}

int min = Integer.MAX\_VALUE;

LABEL:

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

boolean[] visited = new boolean[n];

List<Integer> list = map.get(i);

LinkedList<Integer> queue = new LinkedList<>();

queue.offer(i);

int count = 0;

while(!queue.isEmpty()){

count++;

if(count > min) continue LABEL;

int size = queue.size();

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

int cur = queue.poll();

if(visited[cur]) continue;

visited[cur] = true;

queue.addAll(map.get(cur));

}

}

if(count == min)

result.add(i);

else if(count < min){

min = count;

result.clear();

result.add(i);

}

}

return result;

}

}

PROBLEM-8

class Node {

int data;

Node left, right;

public Node(int item)

{

data = item;

left = right = null;

}

}

class Res {

public int val;

}

class BinaryTree {

Node root;

int findMaxUtil(Node node, Res res)

{

if (node == null)

return 0;

int l = findMaxUtil(node.left, res);

int r = findMaxUtil(node.right, res);

int max\_single = Math.max(

Math.max(l, r) + node.data, node.data);

int max\_top

= Math.max(max\_single, l + r + node.data);

res.val = Math.max(res.val, max\_top);

return max\_single;

}

int findMaxSum() { return findMaxSum(root); }

int findMaxSum(Node node)

{

Res res = new Res();

res.val = Integer.MIN\_VALUE;

findMaxUtil(node, res);

return res.val;

}

}

PROBLEM-9

import java.util.\*;

public class DAGPaths {

public int countPaths(int[][] graph, int source, int destination) {

int numNodes = graph.length;

List<Integer>[] adjacencyList = new List[numNodes];

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

adjacencyList[i] = new ArrayList<>();

}

for (int[] edge : graph) {

int u = edge[0];

int v = edge[1];

adjacencyList[u].add(v);

}

int[] pathCount = new int[numNodes];

pathCount[destination] = 1;

dfs(adjacencyList, source, pathCount);

return pathCount[source];

}

private void dfs(List<Integer>[] adjacencyList, int node, int[] pathCount) {

for (int neighbor : adjacencyList[node]) {

pathCount[neighbor] += pathCount[node];

dfs(adjacencyList, neighbor, pathCount);

}

}

}

PROBLEM-10

public class NumberOfConnectedComponentsInAnUndirectedGraph {

private int count;

public int countComponents(int n, int[][] edges) {

this.count = n;

int[] root = new int[n];

for (int i = 0; i < n; ++i) root[i] = i;

for (int[] edge : edges) {

union(root, edge[0], edge[1]);

}

return this.count;

}

private int find(int[] root, int i) {

if (root[i] == i) return i;

return find(root, root[i]);

}

private void union(int[] root, int src, int dst) {

int srcRoot = find(root, src);

int dstRoot = find(root, dst);

if (srcRoot != dstRoot) {

root[srcRoot] = dstRoot;

--this.count;

}

}

public static void main(String[] args) {

int[][] edges = new int[][] {{0,1},{1,2},{3,4},{2,4},{1,4},{0,4}};

int n = 7;

NumberOfConnectedComponentsInAnUndirectedGraph nc = new NumberOfConnectedComponentsInAnUndirectedGraph();

System.out.println(nc.countComponents(n, edges));

}

}

PROBLEM-11

public class Solution {

public int longestIncreasingPath(int[][] matrix) {

if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {

return 0;

}

int[][] cache = new int[matrix.length][matrix[0].length];

int max = 0;

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

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

int length = findSmallAround(i, j, matrix, cache, Integer.MAX\_VALUE);

max = Math.max(length, max);

}

}

return max;

}

private int findSmallAround(int i, int j, int[][] matrix, int[][] cache, int pre) {

if (i < 0 || i >= matrix.length || j < 0 || j >= matrix[0].length || matrix[i][j] >= pre) {

return 0;

}

if (cache[i][j] > 0) {

return cache[i][j];

} else {

int cur = matrix[i][j];

int tempMax = 0;

tempMax = Math.max(findSmallAround(i - 1, j, matrix, cache, cur), tempMax);

tempMax = Math.max(findSmallAround(i + 1, j, matrix, cache, cur), tempMax);

tempMax = Math.max(findSmallAround(i, j - 1, matrix, cache, cur), tempMax);

tempMax = Math.max(findSmallAround(i, j + 1, matrix, cache, cur), tempMax);

cache[i][j] = ++tempMax;

return tempMax;

}

}

}

PROBLEM-12

class Solution

{

static int maxLen = 10;

static int maskLen = 130;

static int[][] dp = new int[maxLen][maskLen];

static boolean[][] v = new boolean[maxLen][maskLen];

static int minSteps(int arr[], int i, int mask, int n)

{

if (i == n - 1)

{

return 0;

}

if (i > n - 1 || i < 0)

{

return 9999999;

}

if ((mask >> i) % 2 == 1)

{

return 9999999;

}

if (v[i][mask])

{

return dp[i][mask];

}

v[i][mask] = true;

dp[i][mask] = 1 + Math.min(minSteps(arr, i - arr[i], (mask | (1 << i)), n),

minSteps(arr, i + arr[i], (mask | (1 << i)), n));

return dp[i][mask];

}

}

PROBLEM-13

class Solution{

public List<String> findItinerary(List<List<String>> tickets) {

Map<String, List<String>> map = new HashMap<>();

for(List<String> ticket: tickets)

{

String start = ticket.get(0);

String end = ticket.get(1);

List<String> inner = map.getOrDefault(start, new ArrayList<>());

inner.add(end);

map.put(start,inner);

}

for(String s: map.keySet())

Collections.sort(map.get(s));

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

helper(map,list,"JFK");

return list;

}

public void helper(Map<String,List<String>> map, List<String> rst, String cur){

List<String> nexts = map.get(cur);

while(nexts!=null && nexts.size()>0)

{

String next = nexts.get(0);

nexts.remove(0);

helper(map,rst,next);

}

rst.add(0,cur);

}

}

PROBLEM-14

import java.io.\*;

import java.util.\*;

class Solution {

public static int minSwaps(int[] nums)

{

int len = nums.length;

HashMap<Integer, Integer> map = new HashMap<>();

for (int i = 0; i < len; i++)

map.put(nums[i], i);

Arrays.sort(nums);

boolean[] visited = new boolean[len];

Arrays.fill(visited, false);

int ans = 0;

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

if (visited[i] || map.get(nums[i]) == i)

continue;

int j = i, cycle\_size = 0;

while (!visited[j]) {

visited[j] = true;

j = map.get(nums[j]);

cycle\_size++;

}

if (cycle\_size > 0) {

ans += (cycle\_size - 1);

}

}

return ans;

}

}

PROBLEM-15

public class DecodeWays {

public static int numDecodings(String s) {

if (s.length() == 0 || s.charAt(0) == '0') {

return 0;

}

if (s.length() == 1) {

return 1;

}

int n = s.length();

int[] dp = new int[n + 1];

dp[0] = 1;

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

int currentDigit = Character.getNumericValue(s.charAt(i - 1));

int previousDigit = Character.getNumericValue(s.charAt(i - 2));

if (currentDigit >= 1 && currentDigit <= 9) {

dp[i] += dp[i - 1];

}

int twoDigitNumber = previousDigit \* 10 + currentDigit;

if (twoDigitNumber >= 10 && twoDigitNumber <= 26) {

dp[i] += dp[i - 2];

}

}

return dp[n];

}

}