 Text

Description automatically generated

Module Name STW5008CEM: Programming for Developers

MODULE TEACHER: HIKMAT SAUD

SUBMITTED BY: SUBHANJAN BAM(210237)

BATCH:31A

1a. import java.util.\*;

public class Question1A {

static class Edge {

int to, cost, time;

Edge(int to, int cost, int time) {

this.to = to;

this. Cost = cost;

this. Time = time;

}

}

static int n, m;

static List<Edge>[] G;

static int[] dist, cost;

static boolean[] used;

static int dijkstra(int s, int t, int limit) {

Arrays.fill(dist, Integer.MAX\_VALUE / 2);

Arrays.fill(cost, Integer.MAX\_VALUE / 2);

Arrays.fill(used, false);

PriorityQueue<Integer> pq = new PriorityQueue<>((u, v) -> dist[u] - dist[v]);

dist[s] = 0;

cost[s] = 0;

pq.offer(s);

while (!pq.isEmpty()) {

int v = pq.poll();

if (v == t) break;

if (used[v]) continue;

used[v] = true;

for (Edge e : G[v]) {

int u = e.to;

int time = e.time;

int c = e.cost;

if (dist[v] + time > limit) continue;

if (dist[u] > dist[v] + time) {

dist[u] = dist[v] + time;

cost[u] = cost[v] + c;

pq.offer(u);

} else if (dist[u] == dist[v] + time && cost[u] > cost[v] + c) {

cost[u] = cost[v] + c;

pq.offer(u);

}

}

}

return cost[t];

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

m = sc.nextInt();

G = new ArrayList[n];

for (int i = 0; i < n; i++) G[i] = new ArrayList<>();

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

int u = sc.nextInt(), v = sc.nextInt(), t = sc.nextInt(), c = sc.nextInt();

G[u].add(new Edge(v, c, t));

G[v].add(new Edge(u, c, t));

}

int s = sc.nextInt(), t = sc.nextInt(), limit = sc.nextInt();

dist = new int[n];

cost = new int[n];

used = new boolean[n];

System.out.println(dijkstra(s, t, limit));

}

}

1b. import java.util.\*;

public class Question1B {

static List<Integer>[] G;

static boolean[] visited;

static void dfs(int u) {

visited[u] = true;

for (int v : G[u]) {

if (!visited[v]) dfs(v);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int m = sc.nextInt();

G = new ArrayList[n];

for (int i = 0; i < n; i++) G[i] = new ArrayList<>();

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

int u = sc.nextInt(), v = sc.nextInt();

G[u].add(v);

G[v].add(u);

}

int k = sc.nextInt();

visited = new boolean[n];

dfs(0);

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

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

if (visited[i] && !visited[k]) res.add(i);

}

System.out.println(res);

}

}

2a. public class Question2A {

class Node {

Node left;

Node right;

int data;

}

class BinaryTree {

public Node getParent(Node node, int key) {

if (node == null || node.data == key) {

return null;

}

if ((node.left != null && node.left.data == key) || (node.right != null && node.right.data == key)) {

return node;

}

Node l = getParent(node.left, key);

if (l != null) {

return l;

}

l = getParent(node.right, key);

return l;

}

public Node createNewNode(int val) {

Node newNode = new Node();

newNode.data = val;

newNode.left = null;

newNode.right = null;

return newNode;

}

}

public class BinaryTreeApp {

public void main(String[] args) {

BinaryTree a = new BinaryTree();

Node root = a.createNewNode(2);

root.left = a.createNewNode(7);

root.right = a.createNewNode(9);

root.right.right = a.createNewNode(3);

root.left.left = a.createNewNode(10);

root.left.right = a.createNewNode(6);

root.left.right.left = a.createNewNode(5);

root.left.right.right = a.createNewNode(11);

Node parent = a.getParent(root, 6);

if(parent != null) {

System.out.println(parent.data);

} else {

System.out.println("Parent is null");

}

}

}

}

2b. import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class Question2B {

private int[] parents;

private int[] depth;

private List<Integer>[] children;

private int[] subtreeSize;

private int[] serviceCenters;

private int n;

public int minimumServiceCenters(int n, List<Integer>[] children) {

this.n = n;

this.children = children;

parents = new int[n];

depth = new int[n];

subtreeSize = new int[n];

serviceCenters = new int[n];

Arrays.fill(parents, -1);

Arrays.fill(serviceCenters, -1);

dfs1(0, -1);

return dfs2(0, -1);

}

private void dfs1(int node, int parent) {

parents[node] = parent;

depth[node] = parent == -1 ? 0 : depth[parent] + 1;

subtreeSize[node] = 1;

for (int child : children[node]) {

dfs1(child, node);

subtreeSize[node] += subtreeSize[child];

}

}

private int dfs2(int node, int parent) {

if (serviceCenters[node] != -1) return serviceCenters[node];

int count = 0;

for (int child : children[node]) {

count += dfs2(child, node);

}

int option1 = count;

int option2 = n - subtreeSize[node];

serviceCenters[node] = Math.min(option1, option2) + 1;

return serviceCenters[node];

}

public static void main(String[] args) {

List<Integer>[] children = new List[4];

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

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

}

children[0].add(1);

children[0].add(2);

children[0].add(3);

Question2B sc = new Question2B();

System.out.println(sc.minimumServiceCenters(4, children));

}

}

3a. import java.util.Arrays;

public class Question3A {

public static int minProductDifference(int[] arr) {

Arrays.sort(arr);

int minDiff = Integer.MAX\_VALUE;

for (int i = 0; i < arr.length / 2; i++) {

int a = arr[i];

int b = arr[arr.length - i - 1];

int diff = Math.abs(a \* b - (a + b));

minDiff = Math.min(minDiff, diff);

}

return minDiff;

}

public static void main(String[] args) {

int[] arr = {5, 2, 4, 11};

System.out.println(minProductDifference(arr));

}

}

3b.

public class Question3B {

public static boolean match(String a, String pattern) {

int i = 0, j = 0;

while (i < a.length() && j < pattern.length()) {

if (pattern.charAt(j) == '@') {

return i == a.length() - 1 && j == pattern.length() - 1;

} else if (pattern.charAt(j) == '#') {

i++;

j++;

} else if (a.charAt(i) == pattern.charAt(j)) {

i++;

j++;

} else {

return false;

}

}

return i == a.length() && j == pattern.length();

}

public static void main(String[] args) {

System.out.println(match("tt", "@")); // true

System.out.println(match("ta", "t")); // false

System.out.println(match("ta", "t#")); // true

}

}

4a. import java.util.HashMap;

import java.util.Map;

public class Question4A {

class Node {

int key;

int value;

int count;

Node prev;

Node next;

public Node(int key, int value) {

this.key = key;

this.value = value;

this.count = 1;

}

}

int capacity;

Map<Integer, Node> cache;

Map<Integer, Node> head;

Map<Integer, Node> tail;

public Question4A(int capacity) {

this.capacity = capacity;

cache = new HashMap<>();

head = new HashMap<>();

tail = new HashMap<>();

}

public int get(int key) {

if (!cache.containsKey(key)) {

return -1;

}

Node node = cache.get(key);

remove(node);

add(node);

return node.value;

}

public void put(int key, int value) {

if (capacity == 0) {

return;

}

if (cache.containsKey(key)) {

Node node = cache.get(key);

node.value = value;

remove(node);

add(node);

} else {

if (cache.size() == capacity) {

cache.remove(head.get(head.size()).key);

remove(head.get(head.size()));

}

Node node = new Node(key, value);

cache.put(key, node);

add(node);

}

}

private void remove(Node node) {

if (head.get(node.count) == node) {

head.put(node.count, node.next);

}

if (tail.get(node.count) == node) {

tail.put(node.count, node.prev);

}

if (node.prev != null) {

node.prev.next = node.next;

}

if (node.next != null) {

node.next.prev = node.prev;

}

}

private void add(Node node) {

node.count++;

if (!head.containsKey(node.count)) {

head.put(node.count, node);

}

if (!tail.containsKey(node.count)) {

tail.put(node.count, node);

}

if (head.containsKey(node.count - 1)) {

tail.get(node.count - 1).next = node;

node.prev = tail.get(node.count - 1);

}

tail.put(node.count, node);

}

}

4b.

//class LinkedListSort {

// static class ListNode {

// int val;

// ListNode next;

//

// ListNode(int x) {

// val = x;

// }

// }

//

// public int sortList(ListNode head) {

// if (head == null || head.next == null)

// return 0;

//

// int count = 0;

// ListNode current = head;

// while (current.next != null) {

// if (current.val > current.next.val) {

// current.next = current.next.next;

// count++;

// } else {

// current = current.next;

// }

// }

// return count;

// }

//

// public static void main(String[] args) {

// LinkedListSort obj = new LinkedListSort();

// ListNode head = new ListNode(3);

// head.next = new ListNode(2);

// head.next.next = new ListNode(1);

// head.next.next.next = new ListNode(4);

// head.next.next.next.next = new ListNode(7);

// head.next.next.next.next.next = new ListNode(8);

//

// System.out.println("Number of steps required to sort the linked list: " + obj.sortList(head));

// }

//}

5a. import java.util.\*;

class Question5A {

public List<Integer> getBorder(int[][] height) {

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

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

int n = height.length;

int maxHeight = 0;

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

map.put(height[i][0], 0);

map.put(height[i][1], 0);

maxHeight = Math.max(maxHeight, height[i][2]);

}

int currentHeight = 0;

for (Map.Entry<Integer, Integer> entry : map.entrySet()) {

int key = entry.getKey();

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

if (height[i][0] <= key && key <= height[i][1]) {

entry.setValue(Math.max(entry.getValue(), height[i][2]));

}

}

if (entry.getValue() > currentHeight) {

result.add(key);

currentHeight = entry.getValue();

} else if (entry.getValue() < currentHeight) {

result.add(key);

currentHeight = entry.getValue();

}

}

return result;

}

}

5b. import java.util.ArrayList;

public class Question5B {

public int numBatteryReplacements(int[][] serviceCenters, int targetMiles, int startChargeCapacity) {

int count = 0;

int currentMiles = startChargeCapacity;

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

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

for (int[] serviceCenter : serviceCenters) {

distances.add(serviceCenter[0]);

capacities.add(serviceCenter[1]);

}

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

if (distances.get(i) > currentMiles) {

currentMiles = capacities.get(i - 1);

count++;

}

}

if (currentMiles < targetMiles) {

count++;

}

return count;

}

public static void main(String[] args) {

int [][] serviceCenterList={{10,60},{20,30},{30,30},{60,40}};

Question5B question1=new Question5B();

int finalAnswer=question1.numBatteryReplacements(serviceCenterList,100,10);

System.out.println("the car's batteries need to be replaced: "+finalAnswer +"times.");

}

}

6a. import java.util.PriorityQueue;

import java.util.Comparator;

public class Question6A {

// Huffman Coding in Java

class HuffmanNode {

int item;

char c;

HuffmanNode left;

HuffmanNode right;

}

// For comparing the nodes

class ImplementComparator implements Comparator<HuffmanNode> {

public int compare(HuffmanNode x, HuffmanNode y) {

return x.item - y.item;

}

}

// IMplementing the huffman algorithm

public class Huffman {

public static void printCode(HuffmanNode root, String s) {

if (root.left == null && root.right == null && Character.isLetter(root.c)) {

System.out.println(root.c + " | " + s);

return;

}

printCode(root.left, s + "0");

printCode(root.right, s + "1");

}

public void main(String[] args) {

int n = 4;

char[] charArray = { 'A', 'B', 'C', 'D' };

int[] charfreq = { 5, 1, 6, 3 };

PriorityQueue<HuffmanNode> q = new PriorityQueue<HuffmanNode>(n, new ImplementComparator());

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

HuffmanNode hn = new HuffmanNode();

hn.c = charArray[i];

hn.item = charfreq[i];

hn.left = null;

hn.right = null;

q.add(hn);

}

HuffmanNode root = null;

while (q.size() > 1) {

HuffmanNode x = q.peek();

q.poll();

HuffmanNode y = q.peek();

q.poll();

HuffmanNode f = new HuffmanNode();

f.item = x.item + y.item;

f.c = '-';

f.left = x;

f.right = y;

root = f;

q.add(f);

}

System.out.println(" Char | Huffman code ");

System.out.println("--------------------");

printCode(root, "");

}

}

}

6b. import java.util.HashMap;

public class Question6B {

static HashMap<Character, Integer> charToNum = new HashMap<>();

static int[] wordToNum(String word) {

int[] nums = new int[word.length()];

for (int i = 1; i < word.length(); i++) {

nums[i] = charToNum.get(word.charAt(i));

}

return nums;

}

static int toNum(int[] nums) {

int num = 0;

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

num = num \* 10 + nums[i];

}

return num;

}

public static boolean isValid(String[] words, String result) {

charToNum.clear();

int cnt = 0;

for (String word : words) {

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

if (!charToNum.containsKey(c)) {

charToNum.put(c, cnt++);

}

}

}

int[] wordNums = wordToNum(result);

int target = toNum(wordNums);

int sum = 0;

for (String word : words) {

int[] nums = wordToNum(word);

sum += toNum(nums);

}

return sum == target;

}

public static void main(String[] args) {

String[] words = {"SIX", "SEVEN", "SEVEN"};

String result = "TWENTY";

System.out.println(isValid(words, result));

}

}

7a. import java.util.HashMap;

public class Question6B {

static HashMap<Character, Integer> charToNum = new HashMap<>();

static int[] wordToNum(String word) {

int[] nums = new int[word.length()];

for (int i = 1; i < word.length(); i++) {

nums[i] = charToNum.get(word.charAt(i));

}

return nums;

}

static int toNum(int[] nums) {

int num = 0;

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

num = num \* 10 + nums[i];

}

return num;

}

public static boolean isValid(String[] words, String result) {

charToNum.clear();

int cnt = 0;

for (String word : words) {

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

if (!charToNum.containsKey(c)) {

charToNum.put(c, cnt++);

}

}

}

int[] wordNums = wordToNum(result);

int target = toNum(wordNums);

int sum = 0;

for (String word : words) {

int[] nums = wordToNum(word);

sum += toNum(nums);

}

return sum == target;

}

public static void main(String[] args) {

String[] words = {"SIX", "SEVEN", "SEVEN"};

String result = "TWENTY";

System.out.println(isValid(words, result));

}

}

7b. import java.util.ArrayList;

public class Question7B {

public class WebCrawler implements Runnable{

private static final int MAX\_DEPTH=3;

private Thread thread;

private String first\_link;

private ArrayList<String> visitedLinks= new ArrayList<String>();

private int ID;

public WebCrawler(String link,int num){

System.out.println("WebCrawler Created");

first\_link=link;

ID=num;

thread=new Thread(this);

thread.start();

}

@Override

public void run() {

}

}

}

8a.

8b. import java.util.ArrayList;

public class Question7B {

public class WebCrawler implements Runnable{

private static final int MAX\_DEPTH=3;

private Thread thread;

private String first\_link;

private ArrayList<String> visitedLinks= new ArrayList<String>();

private int ID;

public WebCrawler(String link,int num){

System.out.println("WebCrawler Created");

first\_link=link;

ID=num;

thread=new Thread(this);

thread.start();

}

@Override

public void run() {

}

}

}