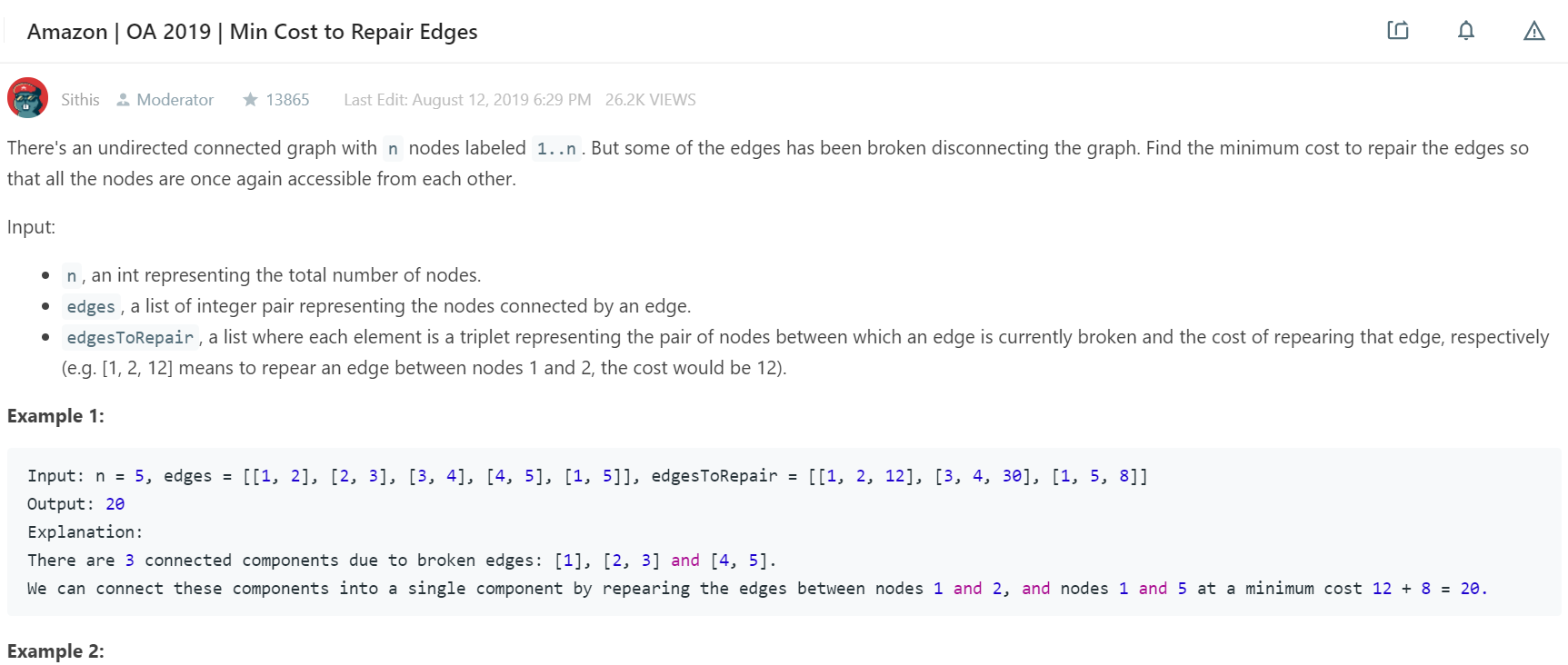
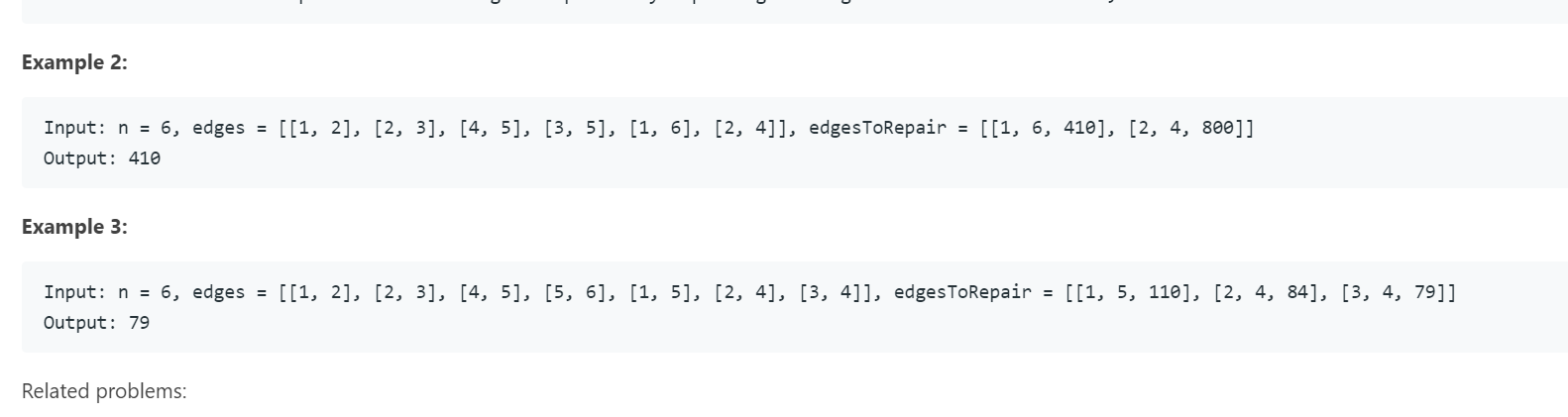
Amazon OA Min Cost To Repair Edges (Minimum Spanning Tree II)

LEET CODE : <https://leetcode.com/discuss/interview-question/357310>





public static void main(String[] args) {

int n1 = 5;

int[][] edges1 = {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {1, 5}}, edgesToRepair1 = {{1, 2, 12}, {3, 4, 30}, {1, 5, 8}};

System.out.println("Min Cost: " + minCost(n1, edges1, edgesToRepair1));

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

System.out.println("Min Cost: " + minCostAllStartNodes(n1, edges1, edgesToRepair1));

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

int n2 = 6;

int[][] edges2 = {{1, 2}, {2, 3}, {4, 5}, {3, 5}, {1, 6}, {2, 4}}, edgesToRepair2 = {{1, 6, 410}, {2, 4, 800}};

System.out.println("Min Cost: " + minCost(n2, edges2, edgesToRepair2));

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

System.out.println("Min Cost: " + minCostAllStartNodes(n2, edges2, edgesToRepair2));

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

int n3 = 6;

int[][] edges3 = {{1, 2}, {2, 3}, {4, 5}, {5, 6}, {1, 5}, {2, 4}, {3, 4}}, edgesToRepair3 = {{1, 5, 110}, {2, 4, 84}, {3, 4, 79}};

System.out.println("Min Cost: " + minCost(n3, edges3, edgesToRepair3));

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

System.out.println("Min Cost: " + minCostAllStartNodes(n3, edges3, edgesToRepair3));

}

private static int minCost(int n, int[][] edges, int[][] newEdges) {

String to = "->";

int res = 0;

Map<String, Integer> graph = new HashMap<>();

for (int[] edge : edges) {

graph.put(edge[0] + to + edge[1], 0);

graph.put(edge[1] + to + edge[0], 0);

}

for (int[] edge : newEdges) {

graph.put(edge[0] + to + edge[1], edge[2]);

graph.put(edge[1] + to + edge[0], edge[2]);

}

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

boolean[] visited = new boolean[n + 1];

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

dist[1] = 0;

visited[1] = true;

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

if (graph.containsKey(1 + to + i)) {

dist[i] = Math.min(dist[i], graph.get(1 + to + i));

}

if (graph.containsKey(i + to + 1)) {

dist[i] = Math.min(dist[i], graph.get(i + to + 1));

}

}

for (int cnt = 0; cnt < n - 1; cnt++) {

int[] next = getMin(dist, visited);

int p = next[0];

int v = next[1];

if (p != -1) {

visited[p] = true;

dist[p] = v;

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

String key = p + to + i;

if (!visited[i] && graph.containsKey(key) && graph.get(key) < dist[i]) {

dist[i] = graph.get(key);

}

}

}

}

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

if (dist[i] == Integer.MAX\_VALUE)

return -1;

res += dist[i];

}

return res;

}

private static int minCostAllStartNodes(int n, int[][] edges, int[][] newEdges) {

String to = "->";

int res = 0;

Map<String, Integer> graph = new HashMap<>();

for (int[] edge : edges) {

graph.put(edge[0] + to + edge[1], 0);

graph.put(edge[1] + to + edge[0], 0);

}

for (int[] edge : newEdges) {

graph.put(edge[0] + to + edge[1], edge[2]);

graph.put(edge[1] + to + edge[0], edge[2]);

}

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

boolean[] visited = new boolean[n + 1];

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

dist = new int[n + 1];

visited = new boolean[n + 1];

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

dist[j] = 0;

visited[j] = true;

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

if (graph.containsKey(j + to + i)) {

dist[i] = Math.min(dist[i], graph.get(j + to + i));

}

if (graph.containsKey(i + to + j)) {

dist[i] = Math.min(dist[i], graph.get(i + to + j));

}

}

for (int cnt = 0; cnt < n - 1; cnt++) {

int[] next = getMin(dist, visited);

int p = next[0];

int v = next[1];

if (p != -1) {

visited[p] = true;

dist[p] = v;

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

String key = p + to + i;

if (!visited[i] && graph.containsKey(key) && graph.get(key) < dist[i]) {

dist[i] = graph.get(key);

}

}

}

}

System.out.println("Dist Array with Start Node: " + j + ", "+ Arrays.toString(dist));

}

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

if (dist[i] == Integer.MAX\_VALUE)

return -1;

res += dist[i];

}

return res;

}

static int[] getMin(int[] dist, boolean[] visited) {

int[] res = new int[2];

res[0] = -1;

res[1] = Integer.MAX\_VALUE;

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

if (!visited[i]) {

if (dist[i] < res[1]) {

res[0] = i;

res[1] = dist[i];

}

}

}

return res;

}