

BAHRIA UNIVERSITY (KARACHI CAMPUS)

|  |  |
| --- | --- |
| **CSC 221 - Data Structures & Algorithms – Assignment 4** | |
| CLO-3 | Deadline: 10-Jan-2023 |
| Class: BSE-3B | Total Marks 5 |

Name: **Muhammad Shoaib Akhter Qadri (individual)**

Enrollment No: **02-131212-009**

Reg No: **79290**

**Question.**

Suppose a company has a wired network within its premises. Users can easy share files from their system to every other system as fast as possible. Unfortunately, the security measures of the company are insufficient. Wires can be monitored by shadowy organization who can intercepts your messages.

After doing some preliminary research, you are able to assign each wire a “risk factor” indicating the likelihood that wire is being monitored. For example, if a wire has a risk factor of zero, it is extremely unlikely to be monitored; if a wire has a risk factor of 10, it is more likely to be monitored. The smallest possible risk factor is 0; the largest possible risk factor is n.

Design and implement an efficient data structure algorithm that selects wires to send your message such that (a) every computer receives the message and (b) you minimize the total risk factor. The total risk factor is defined as the sum of the risks of all the wires you use.

**Answer:**

One possible solution to this problem would be to use a minimum spanning tree (MST) algorithm to find the set of wires that connects all the computers in the network with the minimum total risk.

A minimum spanning tree is a subset of the edges in a graph that forms a tree and connects all the vertices in the graph, without any cycles and with the minimum total weight (in this case, the risk factor). There are several algorithms that can be used to find a minimum spanning tree, such as Prim's algorithm or Kruskal's algorithm.

To use an MST algorithm to solve this problem, you could represent each computer as a vertex in a graph, and each wire as an edge connecting two vertices. The weight of each edge would be the risk factor of the corresponding wire. Then, you could apply an MST algorithm to find the minimum spanning tree of the graph. The edges in the minimum spanning tree would be the wires that you should use to send the message, as they form a tree that connects all the computers in the network with the minimum total risk.

There are several benefits to using an MST algorithm to solve this problem. First, it guarantees that every computer will receive the message, as the minimum spanning tree connects all the vertices in the graph. Second, it minimizes the total risk, as it selects the set of wires with the minimum total risk. Finally, MST algorithms are generally efficient, with a time complexity of O(E log E) or O(E log V) for most MST algorithms, where E is the number of edges and V is the number of vertices in the graph.

**Design & Implementation**

Here is an outline of how you can design and implement an algorithm to select wires to send a message such that every computer receives the message and the total risk factor is minimized:

1. Identify the computers that need to receive the message and the wires that can be used to send the message between them.
2. Create a graph representing the computers and the wires as nodes and edges, respectively.
3. Use a minimum spanning tree (MST) algorithm to find the minimum set of wires that connect all the computers. An MST is a subset of the edges in the graph that form a tree and connect all the nodes in the graph, with the minimum total weight among all possible trees.
4. In this case, the weight of each edge is the risk factor of the corresponding wire. The MST will minimize the total risk factor because it selects the minimum set of wires needed to connect all the computers.
5. To implement the MST algorithm, you can use a greedy algorithm such as Prim's or Kruskal's algorithm. These algorithms build the MST by repeatedly adding the minimum-weight edge that connects an unconnected node to the tree.

**Pseudocode:**

Here is some **pseudocode** for the MST algorithm using Prim's algorithm:

MST(G, w, s):

// G: graph as an adjacency list

// w: weight function for the edges

// s: start node

for each u in G:

dist[u] = INF // Initialize distances to infinity

prev[u] = null // Initialize previous nodes to null

dist[s] = 0 // Distance from start node to itself is 0

Q = G // Initialize priority queue with all nodes in the graph

while Q is not empty:

u = extract\_min(Q) // Get node with minimum distance

for each v in neighbors(u):

if v in Q and w(u, v) < dist[v]:

// Update distance and previous node for v

dist[v] = w(u, v)

prev[v] = u

// Construct the MST by adding the edges (v, prev[v]) for all nodes v

return MST