



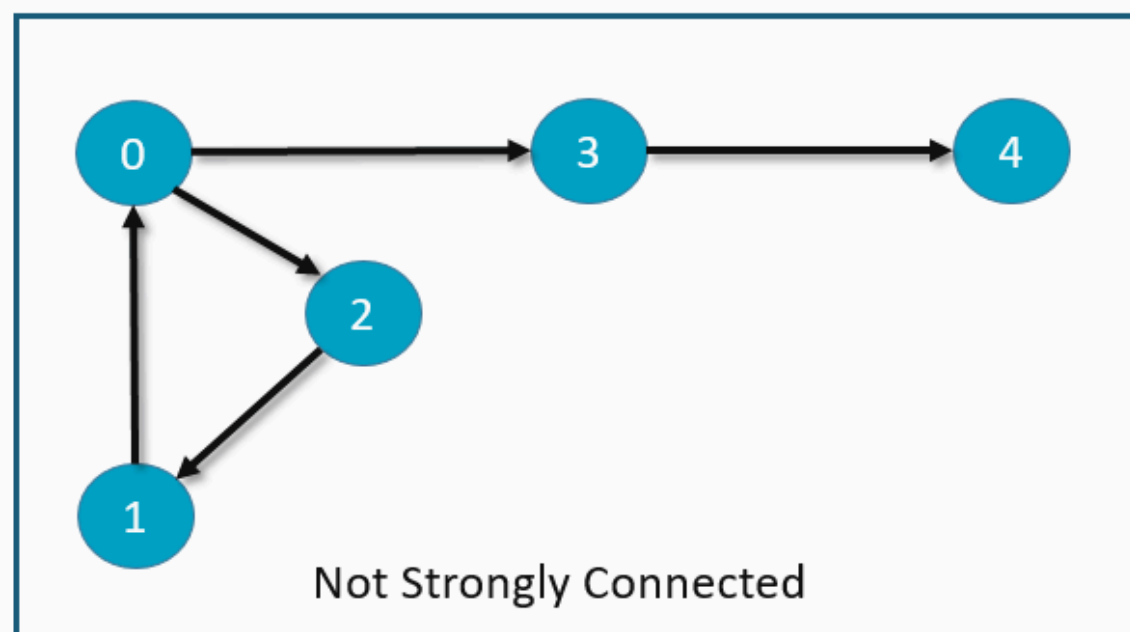
# GRAPHS 2.0

Bharat Singla  
Yiheng

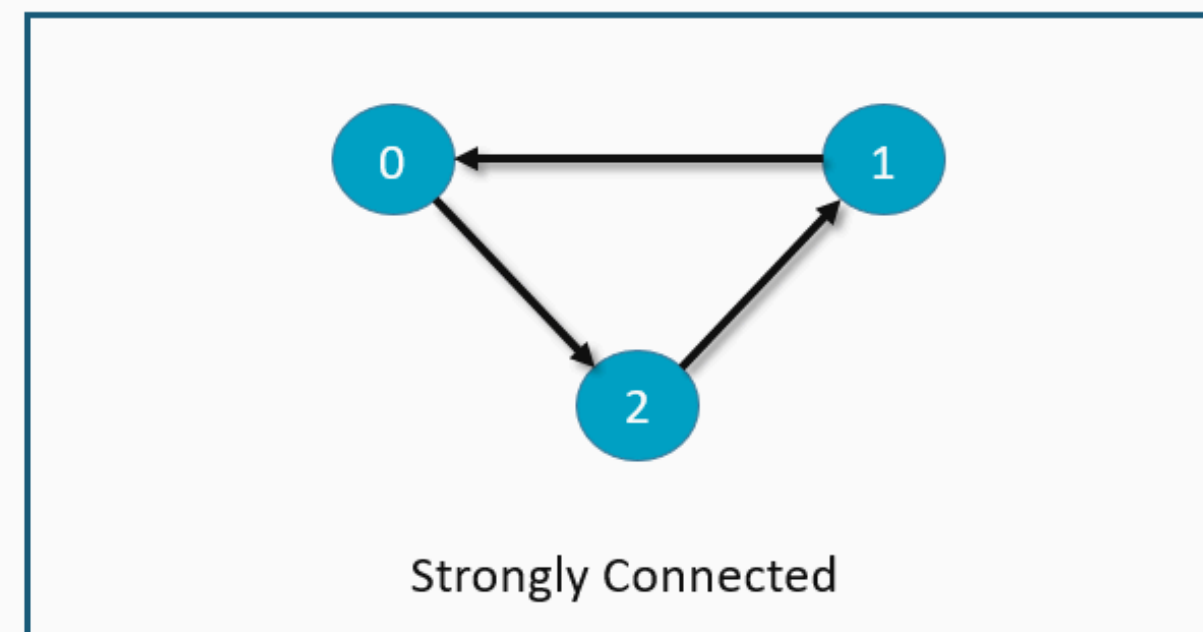
# *Problem: Check if a directed graph is strongly connected*

You are given a directed graph. Check if every vertex is reachable from every other vertex.

Graph 1:



Graph 2:

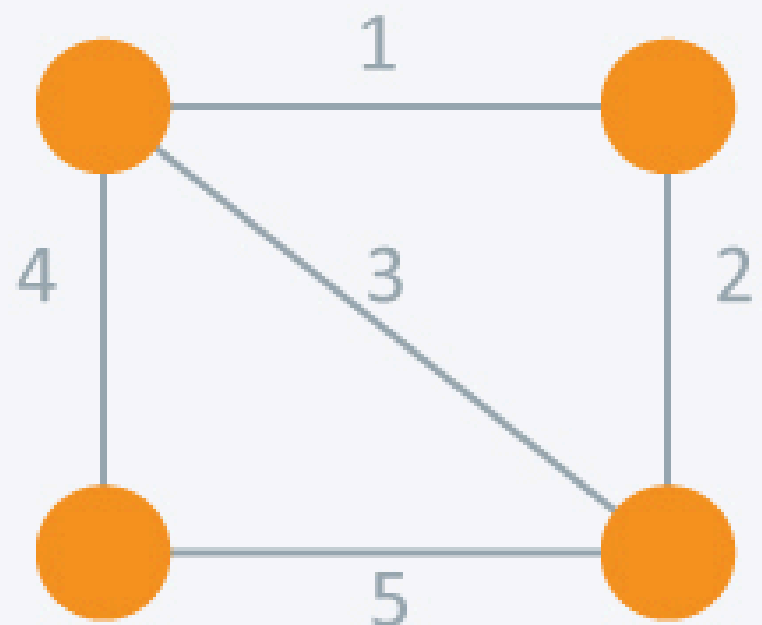


# *Solution:*

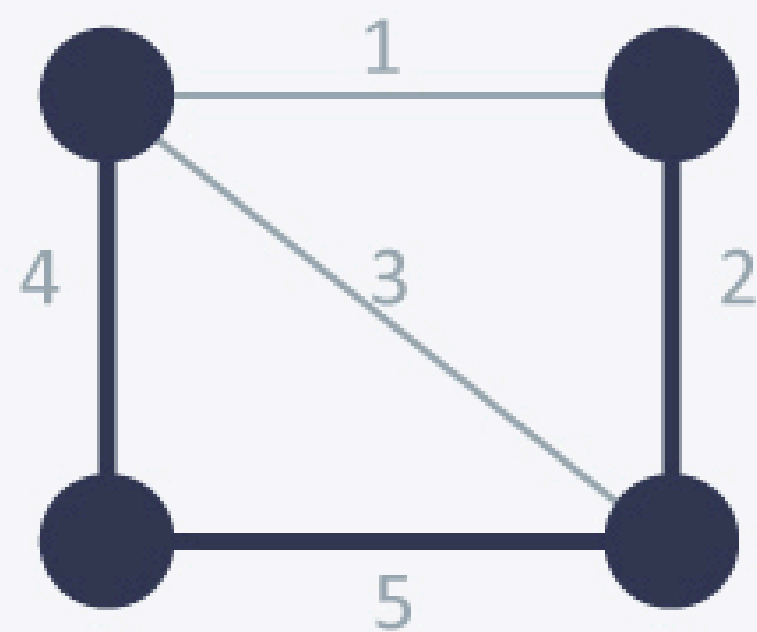
- Perform a DFS starting from an arbitrary vertex ' $v$ '. If any vertex is not visited during this traversal, the graph is not strongly connected.
- Invert the graph (reverse the direction of all edges).
- Perform another DFS starting from the same vertex ' $v$ ' on the transposed graph. If any vertex is not visited, the graph is not strongly connected.

# Minimum Spanning Tree (MST)

An MST of a graph is a sub-graph that is a tree and minimises the sum of edge weights

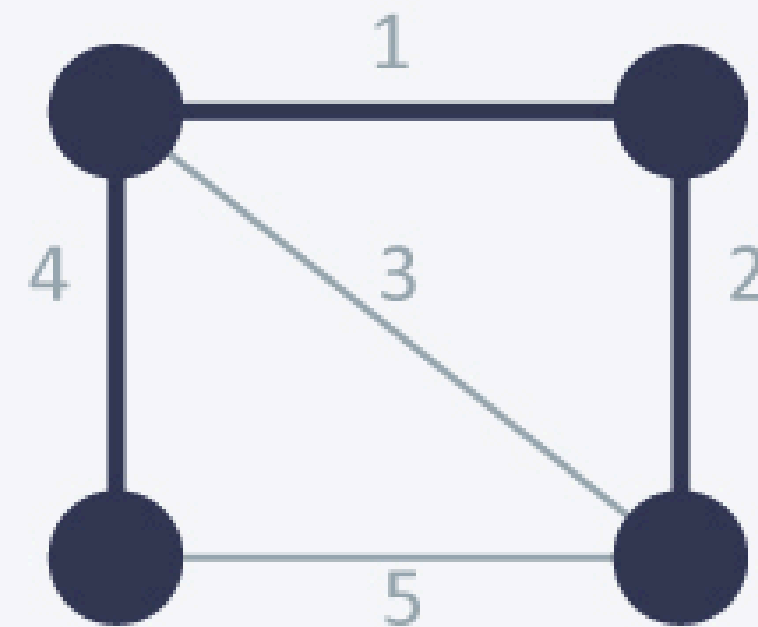


Undirected  
Graph



Spanning  
Tree

Cost = 11(=4+5+2)



Minimum Spanning  
Tree

Cost = 7(=4+1+2)

# *Kruskal's Algorithm*

1. Sort edges of the graph in increasing order of their weights.
2. Initialize an empty list or store the edges of the MST.
3. Initialize a DSU/UFDS to keep track of components.
4. For each edge  $(u, v)$  in the sorted list:
  - a. If  $u$  and  $v$  belong to different components (i.e., adding the edge won't create a cycle):
    - i. Add the edge  $(u, v)$  to the MST.
    - ii. Merge the components of  $u$  and  $v$
5. Repeat step 4 until the MST contains exactly  $(n - 1)$  edges

# *Prim's Algorithm*

1. Initialize a priority queue (min heap) to store edges based on their weights.
2. Start with any arbitrary node as the initial part of the MST and mark it as visited.
3. Add all edges from the starting node to the PQ
4. While the MST does not contain all vertices:
  - a. Extract the edge with the minimum weight from the PQ.
  - b. If the target node of the edge has not been visited:
    - i. Mark the node as visited.
    - ii. Add the edge to the MST.
    - iii. Add all edges from this new node that lead to unvisited nodes to the PQ.
5. Repeat step 5 until all nodes are included in the MST.

# *Problem: Connect Cities to Power Plants*

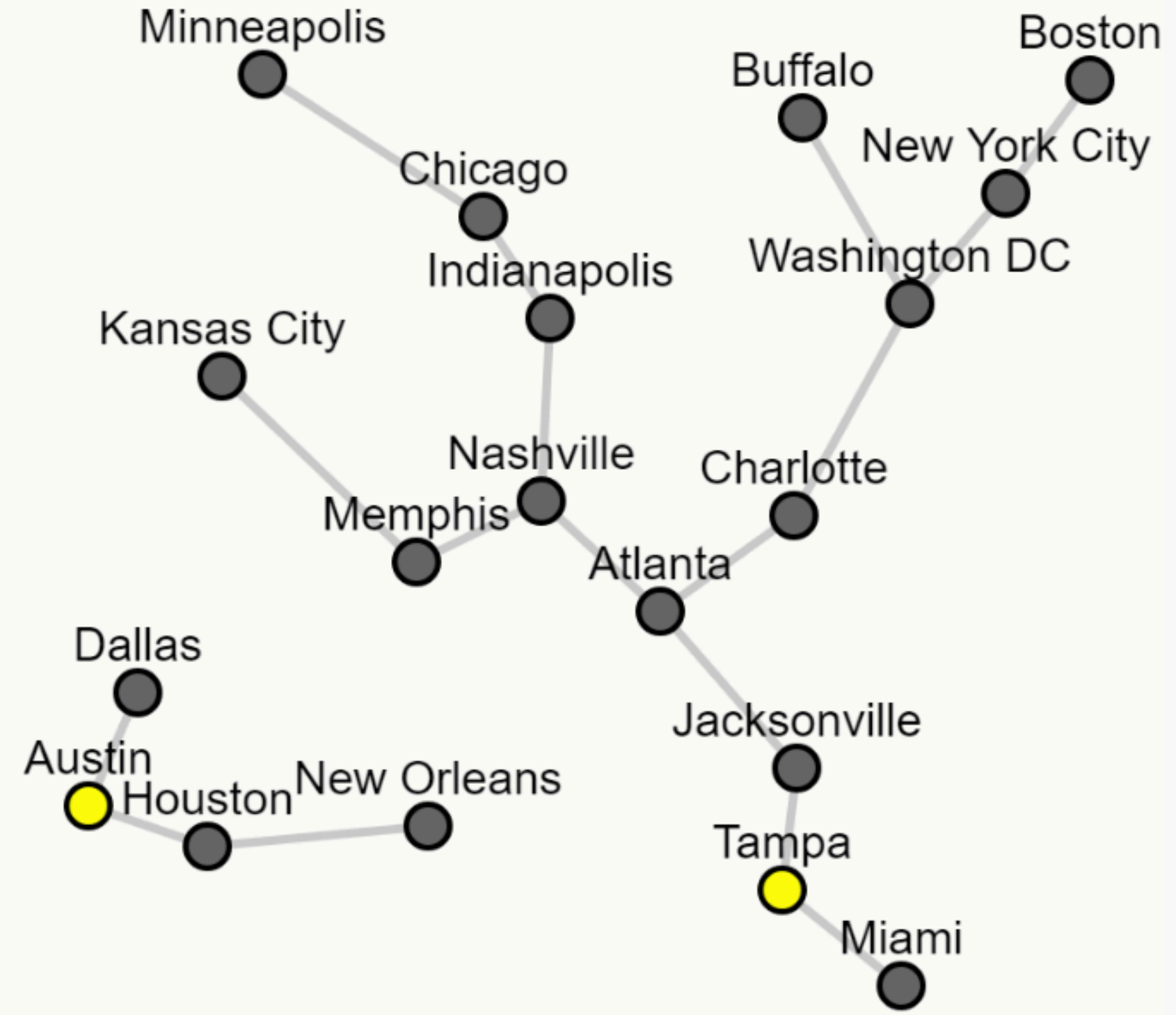
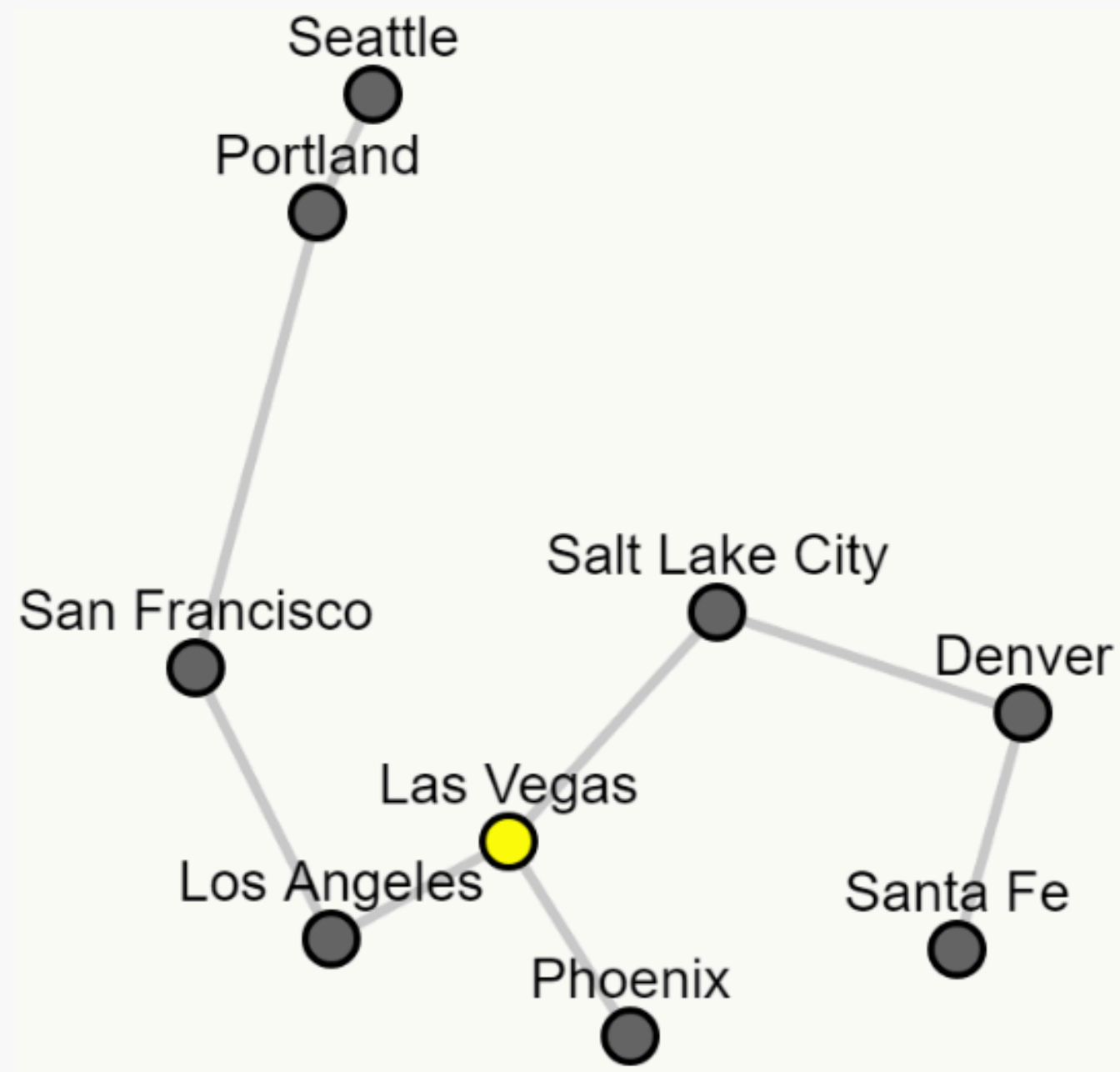
You are given a network of  $n$  cities connected by  $m$  roads.

Each road connects two cities and has an associated cost.

Additionally,  $k$  of these cities have power plants already built.

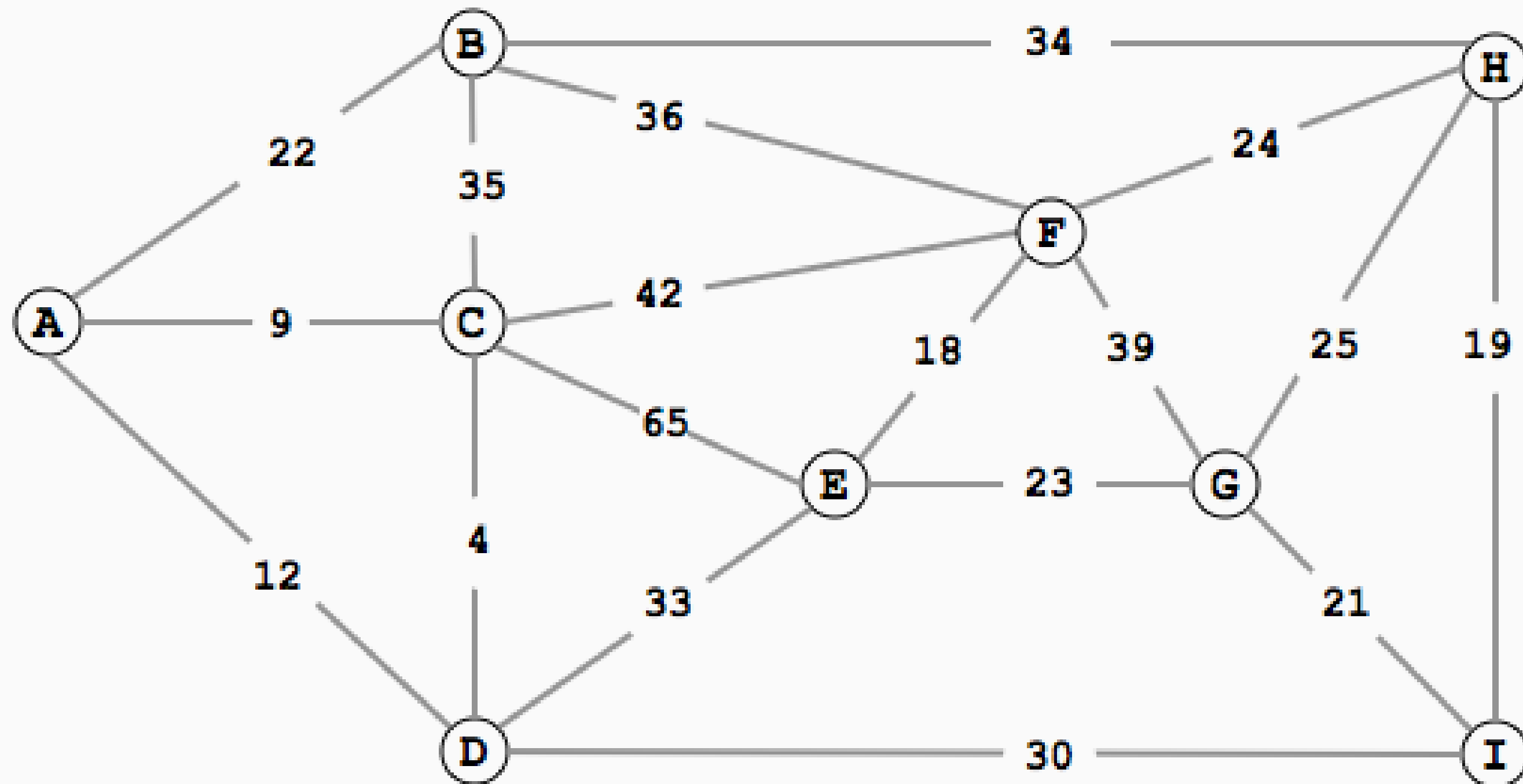
Your task is to connect every city to at least one power plant, directly or indirectly, using the minimum total cost.

Power plants can transmit power to other cities through intermediate cities.






# *Shortesssst Paths!*



# *Dijkstra's Algorithm (SSSP)*

1. Assign a distance of 0 to the source node and  $\infty$  to all other nodes.
2. Initialize a priority queue with the source node.
3. While the priority queue is not empty:
  - a. Extract the node with the smallest tentative distance from the priority queue.
  - b. For each neighbor of the extracted node:
    - i. Calculate the distance to the neighbour through the extracted node.
    - ii. If this new distance is shorter than the neighbor's current shortest distance, update the neighbour's distance and add it to the priority queue.



```
1  int dijkstra(int src, int dest) {
2      priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
3      vector<int> dist(n, INF);
4
5      pq.push({0, src});
6      dist[src] = 0;
7
8      while (!empty(pq)) {
9          auto [d, u] = pq.top();
10         pq.pop();
11         for (auto [v, w] : adj[u]) {
12             if (d + w < dist[v]) {
13                 dist[v] = d + w;
14                 pq.push({dist[v], v});
15             }
16         }
17     }
18
19     return dist[dest];
20 };
```

# *Problem:*

<https://www.geeksforgeeks.org/minimum-possible-modifications-in-the-matrix-to-reach-destination>

# *Floyd-Warshall Algorithm (APSP)*

1. Create a 2D array 'dist' where  $\text{dist}[i][j]$  stores the shortest distance from node  $i$  to node  $j$ .
2. Initialize  $\text{dist}[i][j]$  to:
  - a. 0 if  $i == j$
  - b. edge weight if there is a direct edge from  $i$  to  $j$
  - c.  $\infty$  otherwise
3. For each intermediate node  $k$  from 0 to  $n - 1$ :
  - a. For each pair of nodes  $(i, j)$ :
    - i. Update  $\text{dist}[i][j] = \min(\text{dist}[i][j], \text{dist}[i][k] + \text{dist}[k][j])$
4. After all iterations,  $\text{dist}[i][j]$  contains the shortest path from  $i$  to  $j$ .

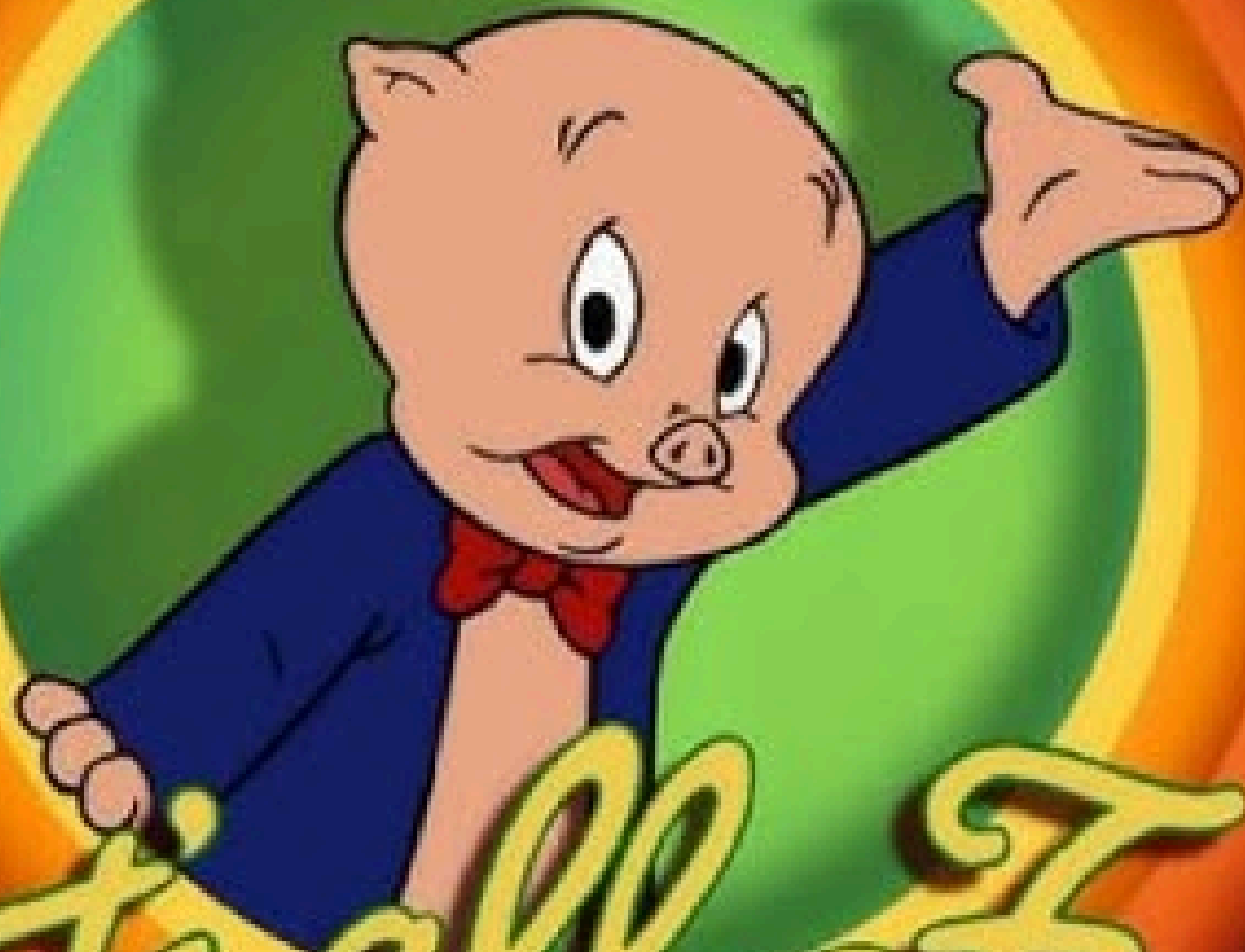


```
1  int floyd_warshall(int src, int dest) {
2      vector<vector<int>> dist = g;
3      for (int k = 0; k < n; k++) {
4          for (int u = 0; u < n; u++) {
5              for (int v = 0; v < n; v++) {
6                  dist[u][v] = min(dist[u][v], dist[u][k] + dist[k][v]);
7              }
8          }
9      }
10     return dist[src][dest];
11 };
```

# *Problem:*

<https://leetcode.com/problems/minimum-cost-to-convert-string-i/description/>

# LOONEY TUNES



*"That's all Folks!"*