Minimum Spanning Tree

Sample Problem: Agri-Net [Russ Cox, Winter 1999 USACO Open]

Farmer John is bringing internet connectivity to all farms in the area. He has ordered a high speed connection for his farm and is going to share his connectivity with the other farmers. To minimize cost, he wants to minimize the length of optical fiber to connect his farm to all the other farms.

Given a list of how much fiber it takes to connect each pair of farms, find the minimum amount of fiber needed to connect them all together. Each farm must connect to some other farm such that a path exists from any farm to any other farm. Some farms might have 1, 2, 3, or more connections to them.

The Abstraction

Given: an undirected, connected graph with weighted edges

A *spanning tree* of a graph is any sub-graph which is a connected tree (i.e., there exists a path between any nodes of the original graph which lies entirely in the sub-graph).

A *minimal* spanning tree is a spanning tree which has minimal `cost' (where cost is the sum of the weights of the edges in the tree).

Prim's algorithm to construct a Minimal Spanning Tree

Given: lists of nodes, edges, and edge costs

The algorithm (greedily) builds the minimal spanning tree by iteratively adding nodes into a working tree.

- Start with a tree which contains only one node. Iteratively find the closest node to that one and add the edge between them.
- Let the distance from each node not in the tree to the tree be the edge (connection) of minimal weight between that node and some node in the tree. If there is no such edge, then assume the distance is infinity (this shouldn't happen).
- At each step, identify a node (outside the tree) which is closest to the tree and add the minimum weight edge from that node to some node in the tree and incorporate the additional node as a part of the tree.

For analysis of why this works, consult Chapter 24 of [Cormen, Leiserson, Rivest].

Here is pseudocode for the algorithm:

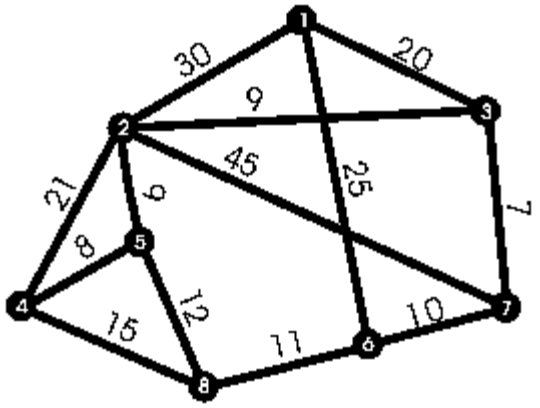
```
  # distance(j) is distance from tree to node j
  # source(j) is which node of so-far connected MST
  #                      is closest to node j
1   For all nodes i
2     distance(i) = infinity        # no connections
3     intree(i) = False             # no nodes in tree
```

```
 4     source(i) = nil

 5   treesize = 1                    # add node 1 to tree
 6   treecost = 0
 7   intree(1) = True
 8   For all neighbors j of node 1   # update distances
 9      distance(j) = weight(1,j)
10     source(j) = 1

11   while (treesize < graphsize)
12     find node with minimum distance to tree; call it node i
13     assert (distance(i) != infinity, "Graph Is Not Connected")

    # add edge source(i),i to MST
14     treesize = treesize + 1
15     treecost = treecost + distance(i)
16     intree(i) = True              # mark node i as in tree

    # update distance after node i added
17     for all neighbors j of node i
18       if (distance(j) > weight(i,j))
19         distance(j) = weight(i,j)
20         source(j) = i
```

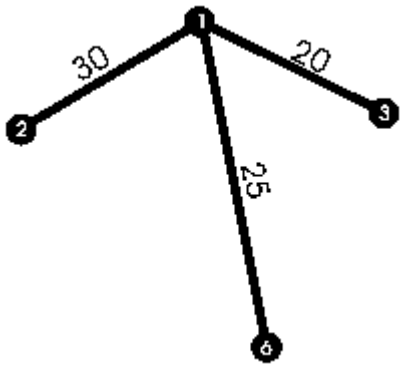
Running time of this formulation is O(N^2). You can obtain O(N log N) for sparse graphs, but it normally isn't worth the extra programming time.

Execution Example

Consider the following graph with weighted edges:



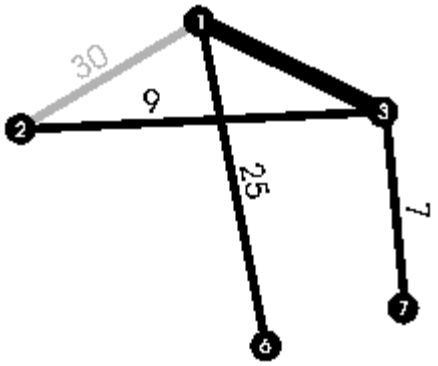
The goal is to find the minimal spanning tree. The algorithm will start at node 1 which connects to nodes 2, 6, and 3 with the weights shown on the edges:



Node	distance	intree	source
1	infinity	True	nil
2	30	False	1
3	20	False	1
6	25	False	1

All nodes not shown have infinite distance, intree=False, and source=nil.

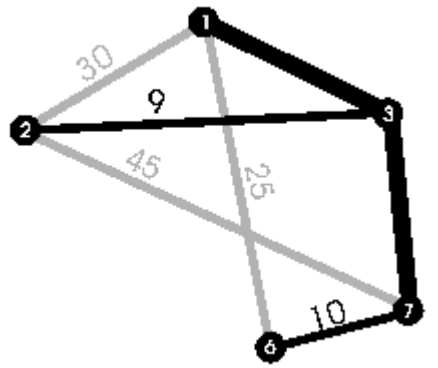
The smallest distance **for a node not in the tree** is 20, so the listed edge to node 3 is added to the tree:



Node	distance	intree	source
1	infinity	True	nil
2	**9**	False	**3**
3	20	**True**	1
6	25	False	1
7	7	False	3

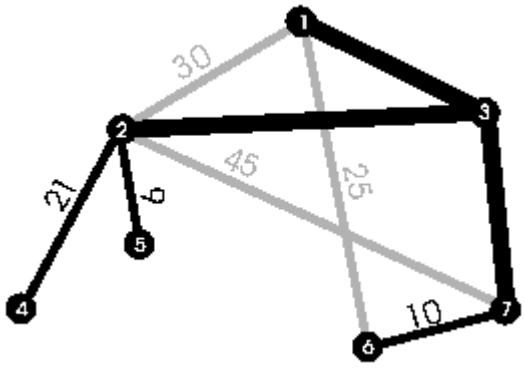
Note that node 3 is now `in the tree'. Node 2's distance changed to 9 while the source changed to 3.

The smallest distance is 7, so the edge from node 3 to node 7 (coincidental name!) is connected:



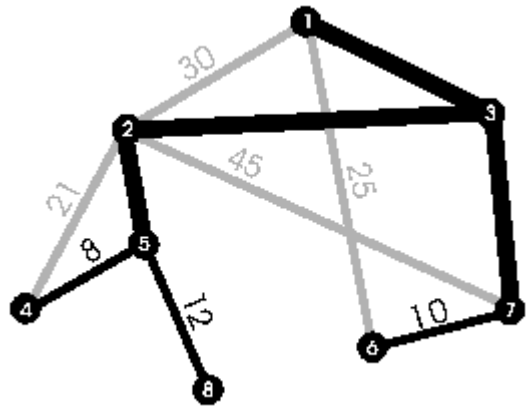
Node	distance	intree	source
1	infinity	True	nil
2	9	False	3
3	20	True	1
6	**10**	False	**7**
7	7	**True**	3

Node 2's distance is 9, the smallest of any node not in the tree. Adding the edge from node 3 to node 2 results in a graph that looks like this:



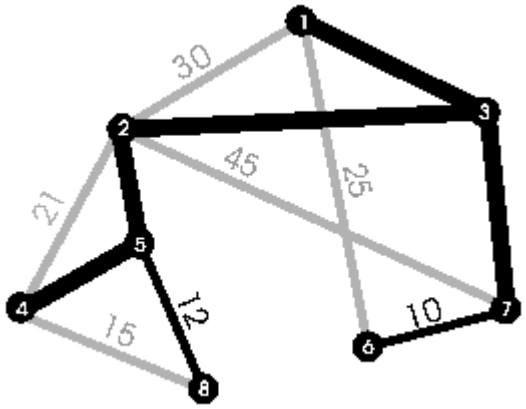
Node	distance	intree	source
1	infinity	True	nil
2	9	**True**	3
3	20	True	1
4	21	False	2
5	9	False	2
6	10	False	7
7	7	True	3

Add the edge from node 2 to node 5:



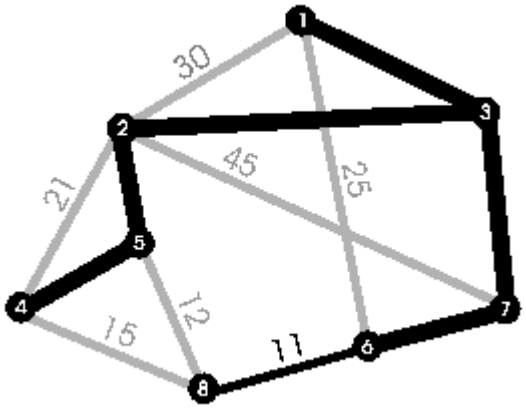
Node	distance	intree	source
1	infinity	True	nil
2	9	**True**	3
3	20	True	1
4	**8**	False	**5**
5	9	**True**	2
6	10	False	7
7	7	True	3
8	12	False	5

Adding the edge from node 5 to node 4 is the next step:



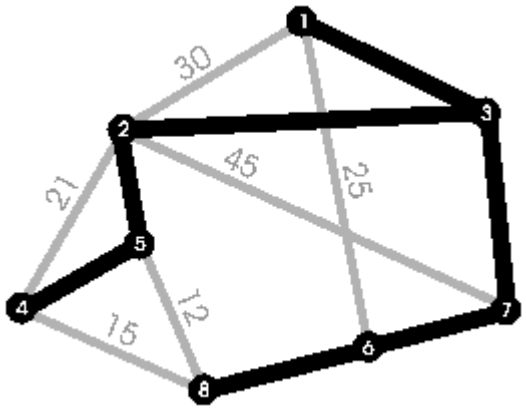
Node	distance	intree	source
1	infinity	True	nil
2	9	True	3
3	20	True	1
4	8	**True**	5
5	9	True	2
6	10	False	7
7	7	True	3
8	12	False	5

Next up: edge connecting nodes 6 and 7:



Node	distance	intree	source
1	infinity	True	nil
2	9	True	3
3	20	True	1
4	8	True	5
5	9	True	2
6	10	**True**	7
7	7	True	3
8	**11**	False	**6**

Finally, add the edge from node 6 to node 8:



Node	distance	intree	source
1	infinity	True	nil
2	9	True	3
3	20	True	1
4	8	True	5
5	9	True	2
6	10	True	7
7	7	True	3
8	11	**True**	6

And the minimal spanning tree is easily seen here.

Dangerous Curve

Understand that changing any element in a tree requires complete recalculation - incremental recalculation of a spanning tree when changing isolated nodes, for example, is not generally possible.

Problem Cues

If the problem mentions wanting an optimal, connected sub-graph, a minimum cost way to connect a system together, or a path between any two parts of the system, it is very likely to be a minimum spanning tree problem.

Extensions

If you subject the tree to any other constraints (no two nodes may be very far away or the average distance must be low), this algorithm breaks down and altering the program to handle such constraints is very difficult.

There is obviously no problem with multiple edges between two nodes (you ignore all but the smallest weight).

Prim's algorithm does not extend to directed graphs (where you want strong connectedness), either.

Sample Problems

Package Routing

Given: a set of locations of cities and the cost of connecting each pair of cities for a shipping company. Find the cheapest set of pairs of cities such that a package can be routed from any city to any other city.

Highway Building

Lower Slobbovia has made the plunge and has decided to connect all their cities with roads. Of course, being cheap, they want to spend as little money as possible. The cost

of a highway is linearly proportional to its length. Given the x,y coordinates of the cities in L.S., find the cheapest way to interconnect the cities.

Bovile Phones (abridged) [USACO Training Camp 1998, Contest 2]

Given: a collection of stationary cows and haystacks in the field along with a cost function for connecting two (arbitrary) locations. Using only the haystacks and cows, calculate which haystacks one should include in a network to minimize the total cost.

Analysis: For each possible set of haystacks (i.e., about 2^n sets), calculate the cost of the minimal spanning tree of the haystacks in that set along with all the cows. Find the combination of haystacks that minimizes the cost.

Back to USACO Training Gateway | Comment or Question