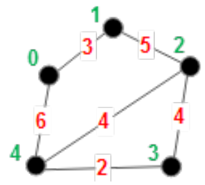
Minimum Spanning Trees

We consider connected undirected graphs that have a weight on each edge. Each weight is a positive integer. In the graph to the right, the nodes are numbered (in green) 0, 1, …, 4. The edge weights are given in red.



A *minimum-cost spanning tree*, or *minimum spanning tree*, is a spanning tree whose sum of the weights on its edges is a minimum over all spanning trees of the graph. The graph to the right has two minimum spanning trees, with cost 14. They contain edges {3, 4}, {0, 1}, one of {2, 3} and {2, 4}, and {1, 2}.

There are many applications of minimum spanning trees. Here's one. A company has to lay cable in a neighborhood, reaching all houses (the nodes of the graph). The cables have to be laid along certain roads (the edges). The cost on an edge could depend on the length of the edge, how deep the cable has to be buried, and other factors. A spanning tree would describe where the cable should be laid. A minimum spanning tree would minimize the cost.

Here are other applications. Telecommunications networks, including the internet, have loops, or cycles, of course. There are "routing protocols" for sending packets from one node on the network to another. These protocols require each router to maintain a spanning tree, and a minimum-cost spanning tree is best.

This application led to the development of an algorithm: Minimize the wire needed to connect pins on the back panel of a computer.

The generation of mazes is another interesting application. We complete a discussion of this later.

Two greedy algorithms

We have already discussed the basic additive algorithm for constructing a spanning tree `G1` of a graph `G`, which is:

```
G1= (the nodes of G and no edges);
Repeat until no longer possible:
    Add to G1 an edge (of G) that connects two unconnected components, i.e. does not form a cycle.
```

We show two abstract algorithms for constructing a minimum-cost spanning tree that are refinements of this basic algorithm. They are both *greedy* algorithms —if you don't know this term, visit JavaHyperText entry "greedy algorithm" and study it before reading further.

Kruskal's algorithm

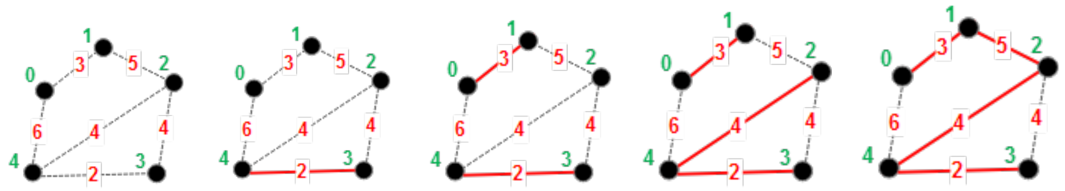
Kruskal published his minimum spanning tree algorithm in 1956. In it, he makes the above abstract algorithm into a greedy one by always adding an edge of minimum weight. Here is his algorithm at an abstract level:

```
G1= (the nodes of G and no edges);
Repeat until no longer possible:
    Add to G1 an edge (of G) of minimum weight that connects two unconnected components of G1.
```

Thus, as the algorithm progresses, `G1` always contains a forest, each tree of which is a subtree of the final spanning tree.

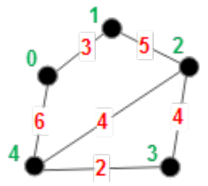
We step through this algorithm using the graph given at the top of the page. Below, the leftmost graph shows the initial graph being constructed. It has the nodes of `G` and that is all. Edges are dotted just to indicate the edges of `G`.

The first iteration adds edge {3, 4} with minimum weight 2. The second iteration adds edge {0, 1} with minimum weight 3. In the third iteration, there is a choice because two edges have weight 4. We arbitrarily chose edge {2, 4}. For the fourth iteration, edge {2, 3} with weight 4 cannot be chosen because its endpoints are in the same component (and a cycle would be formed), so edge {1, 2} with weight 5 is chosen. That's it.

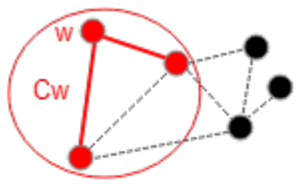


The Jarnik/Prim/Dijkstra (JPD) algorithm

Wikipedia says that this algorithm was first developed in 1930 by Czech mathematician Vojtěch Jarník. It was rediscovered and republished by Robert Prim in 1957. Edsger W. Dijkstra developed it in 1957 and published it in 1959. It is usually known as Prim's algorithm, but in light of its history, we call it the JPD algorithm, as do others.



The JPD algorithm uses the same greedy choice as Kruskals' algorithm, with one major difference. In Kruskal's algorithm, the added edges can belong to different connected components. For example, after edges {0, 1} and {3, 4} are added, they belong to two different connected components. On the other hand, the JDP algorithm requires that all added edges belong to the same component. The algorithm arbitarily chooses one node `w` to start with and adds edges only to the connected component that contains `w`.



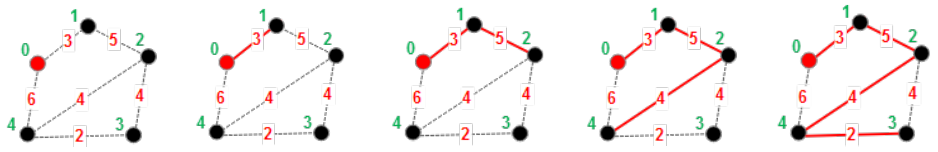
Let's give the name `Cw` to the component that contains `w`. In the diagram to the right, the circled part is `Cw`, and it is a tree. The uncircled part consists of a bunch of 1-node components.

We write the JPD algorithm:

```
G1= (the nodes of G and no edges);
Choose an arbitrary node w;
// invariant: G1 = Cw together with a bunch of 1-node trees, and Cw is a tree
Repeat until no longer possible:
    Add to Cw an edge (of G) of minimum weight that has exactly one endpoint in Cw
```

We step through this algorithm using the graph given at the top of the page. Below, the leftmost graph shows the initial graph `G1`, with node `w` being red node 0. `G1` contains node `w` and no edges. The dotted edges indicate the edges of `G`. Because of the requirement that all added edges must be in component `Cw`:

- Iteration 0 can add only edge {0, 1} or {0, 4}; it greedily adds the one with minimum weight: {0, 1}.
- Similarly iteration 1 adds either {1, 2} or {0, 4}; it greedily adds the one with minimum weight: {1, 2}.
- Iteration 2 can add one of {2, 4}, {2, 3}, and {0, 4}. Two have minimum weight 4, and we have arbitrarily chosen edge {2, 4}.
- Finally, iteration 4 greedily adds edge {3, 4} —the edge with minimum weight over all edges turns out to be the last one added!



It has been proven that both greedy algorithms —Kruskal and JPD— construct a minimal-cost spanning tree. Further, if the edge weights are all different, they construct the same spanning tree.