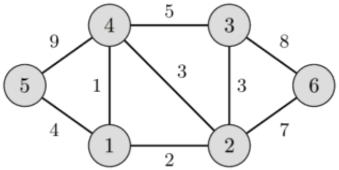
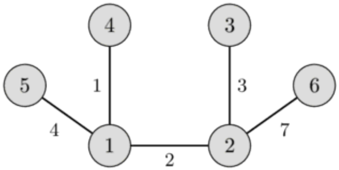
Minimum Spanning Tree

Given a weighted, undirected graph G with n vertices and m edges. We wish to find a spanning tree of this graph which connects all vertices and has the least weight (i.e. the sum of weights of edges is minimal). A spanning tree is a set of edges such that any vertex can reach any other by exactly one simple path. The spanning tree with the least weight is called a minimum spanning tree.

Description

The left image represents a weighted undirected graph, and in the right image we can see the corresponding minimum spanning tree.





Description

It is easy to see that any spanning tree will necessarily contain $n-1$ edges.More generally, any edge-weighted undirected graph (not necessarily connected) has a minimum spanning forest, which is a union of the minimum spanning trees for its connected components.

This problem appears quite naturally in a lot of problems. For instance in the following problem: there are n cities and for each pair of cities we are given the cost to build a road between them (or we know that is physically impossible to build a road between them). We have to build roads, such that we can get from each city to every other city, and the cost for building all roads is minimal.

Properties

1. Possible multiplicity
2. Uniqueness
3. Minimum-cost subgraph
4. Cycle property
5. Cut property
6. Minimum-cost edge
7. Contraction

Possible multiplicity

There may be several minimum spanning trees of the same weight; in particular, if all the edge weights of a given graph are the same, then every spanning tree of that graph is minimum.

Uniqueness

If each edge has a distinct weight then there will be only one, unique minimum spanning tree. This is true in many realistic situations, such as the telecommunications company example above, where it's unlikely any two paths have exactly the same cost. This generalizes to spanning forests as well. More generally, if the edge weights are not all distinct then only the (multi-)set of weights in minimum spanning trees is certain to be unique; it is the same for all minimum spanning trees.

Minimum-cost subgraph

If the weights are positive, then a minimum spanning tree is, in fact, a minimum-cost subgraph connecting all vertices, since if a subgraph contains a cycle, removing any edge along that cycle will decrease its cost and preserve connectivity.

Cycle property

For any cycle C in the graph, if the weight of an edge e of C is larger than any of the individual weights of all other edges of C, then this edge cannot belong to an MST.

Cut property

For any cut C of the graph, if the weight of an edge e in the cut-set of C is strictly smaller than the weights of all other edges of the cut-set of C, then this edge belongs to all MSTs of the graph.

Minimum-cost edge

If the minimum cost edge e of a graph is unique, then this edge is included in any MST.

Contraction

If T is a tree of MST edges, then we can contract T into a single vertex while maintaining the invariant that the MST of the contracted graph plus T gives the MST for the graph before contraction.

Classic algorithms

The first algorithm for finding a minimum spanning tree was developed by Czech scientist Otakar Borůvka in 1926 (see Borůvka's algorithm). Its purpose was an efficient electrical coverage of Moravia. The algorithm proceeds in a sequence of stages. In each stage, called Boruvka step, it identifies a forest F consisting of the minimum-weight edge incident to each vertex in the graph G, then forms the graph $G1 = G \setminus F$ as the input to the next step. Here $G \setminus F$ denotes the graph derived from G by contracting edges in F (by the Cut property, these edges belong to the MST). Each Boruvka step takes linear time. Since the number of vertices is reduced by at least half in each step, Boruvka's algorithm takes $O(mlogn)$ time.

A second algorithm is Prim's algorithm, which was invented by Vojtěch Jarník in 1930 and rediscovered by Prim in 1957 and Dijkstra in 1959. Basically, it grows the $MST(T)$ one edge at a time. Initially, T contains an arbitrary vertex. In each step, T is augmented with a least-weight edge (x,y) such that x is in T and y is not yet in T. By the Cut property, all edges added to T are in the MST. Its run-time is either $O(mlogn)$ or $O(m+nlogn)$, depending on the data-structures used.
A third algorithm commonly in use is Kruskal's algorithm, which also takes $O(mlogn)$ time.
A fourth algorithm, not as commonly used, is the reverse-delete algorithm, which is the reverse of Kruskal's algorithm. Its runtime is $O(mlogn(loglogn)^3)$.
All four of these are greedy algorithms.

Faster algorithms

Several researchers have tried to find more computationally efficient algorithms.
In a comparison model, in which the only allowed operations on edge weights are pairwise comparisons, Karger, Klein and Tarjan (1995) found a linear time randomized algorithm based on a combination of Borůvka's algorithm and the reverse-delete algorithm.
The fastest non-randomized comparison-based algorithm with known complexity, by Bernard Chazelle, is based on the soft heap, an approximate priority queue. Its running time is $O(m\alpha(m,n))$, where α is the classical functional inverse of the Ackermann function. Chazelle's algorithm takes very close to linear time.