Chapter 23. Minimum Spanning Trees

We are given a connected, weighted, undirected graph $G = (V, E; w)$. Each edge $(u, v) \in E$ has a *non-negative weight* (often called *length*) $w(u, v)$.

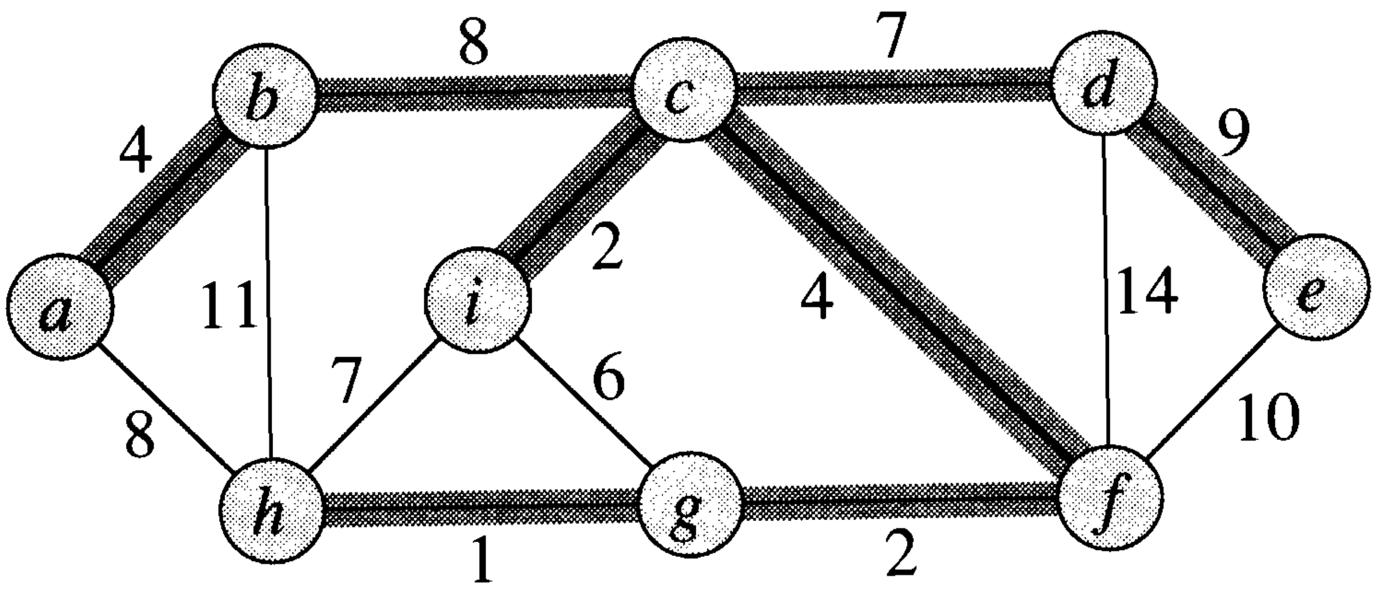
The Minimum Spanning Tree problem (MST) in G is to find a spanning tree $T = (V, E')$ such that the weighted sum of the edges in T is minimised, i.e.,

$$minimise \qquad w(T) = \sum_{(u,v)\in E'} w(u, v),$$

where the minimum is taken over spanning trees T of G, and the sum is taken over all the edges in $T(V, E')$.

Clearly, **if $|V| = n$, then $|E'| = n - 1$.**

Chapter 23. Minimum Spanning Trees



An example, Fig 23.1 (page 625)

23.1 Approaches to Finding MSTs

There are two well-known MST algorithms:

- Kruskal's algorithm
- Prim's algorithm

23.1 Generic Algorithm for MSTs

Both Kruskal's and Prim's algorithms are examples of a generic technique for finding minimum spanning trees.

We have a set of edges, A, which is empty initially. The algorithm adopts the greedy strategy: That is, find tree edges one by one iteratively until the MST is formed. Specifically, at each iteration, we add a new edge to A until it finally is a tree spanning all nodes in the graph.

The idea is to always maintain the following condition:

INVARIANT: Some minimum spanning tree of G contains A.

The initial value $A = \emptyset$ satisfies INVARIANT. So, the problem is to add more edges into A while keeping INVARIANT true.

If INVARIANT is still true when A has become a spanning tree, then the tree must be a minimum spanning tree in the graph.

23.1 Generic MST Algorithm (continued)

A general strategy for choosing edges so that INVARIANT remains true uses cuts.

- A **cut** is a pair $(S, V \setminus S)$, where $S \subseteq V$.
- An edge of G **crosses** the cut $(S, V \setminus S)$ if one endpoint of the edge is in S and the other endpoint of the edge is in $V \setminus S$.
- An edge of G is **light** for the cut $(S, V \setminus S)$ if it crosses the cut and no other edge crossing the cut has lower weight.

Theorem. Suppose $A \subseteq E$ satisfies INVARIANT, and $(S, V \setminus S)$ is a cut such that no edge of A crosses the cut. Let a be a light edge for the cut. Then $A \cup \{a\}$ satisfies INVARIANT.

Note: Choosing an edge $a \in E$ at each step of an algorithm according to this theorem is a greedy strategy.

23.1 Generic MST Algorithm (continued)

Proof of the theorem.

Since a crosses the cut but no edge of A crosses the cut, $A \cup \{a\}$ has no cycles.

Since A satisfies INVARIANT, there is some MST T that includes A.

(a) If a is in T, done, T is an MST that includes $A \cup \{a\}$.

(b) If a is not in T, then $T \cup \{a\}$ has a cycle that includes a and at least one other edge a' crossing the cut. Let $T' = T \cup \{a\} \setminus \{a'\}$. Then, T' is a tree, and

$$w(T') = w(T) + w(a) - w(a') < w(T),$$

since a is a light edge for the cut, this implies $w(a) < w(a')$, which contradicts the fact that T is an MST of G. Therefore, $A \cup \{a\}$ satisfies the INVARIANT.

23.2 Kruskal's Algorithm

For $A \subseteq E$, define $G_A = (V, A)$.

In Kruskal's algorithm, the edge added to A at each step is:

an edge a with least weight that does not create a cycle with the edges in A.

Suppose edge a connects two components CC_1 and CC_2 in G_A. Consider the cut $(S, V \setminus S)$, where S is the vertex set of CC_1. Then, a is light for this cut, and so, by our previous theorem, the algorithm is correct.

The difficult part of implementation is to ensure that we don't create a cycle. For this purpose, we keep track of the connected components of G_A, and only consider edges connecting different connected components. To maintain different connected components (or disjoint vertex sets), we make use of the data structures for disjoint set representations, such data structures include **linked lists** and **inverted trees**.

23.2 Kruskal's Algorithm (continued)

```
Kruskal_MST(G, w)
    A ← ∅;
    for each vertex v ∈ V do
        Make_Set(v);
    Sort the edges in E in increasing order of edge weights w;
        /* most time-consuming step */
    for each edge (u, v) ∈ E in the sorted edge sequence do
        if Find_Set(u) ≠ Find_Set(v)
            A ← A ∪ {(u, v)};
            Union(u, v)
    return A.
```

23.2 Kruskal's Algorithm (continued)

The running time of Kruskal's algorithm:

- Sorting the edges according to weight takes time $O(|E| \log |E|) = O(|E| \log |V|)$. Here, we use the fact that $|E| \leq |V|^2$, which implies $\log |E| \leq 2 \log |V|$.

- If we use different data structures to represent disjoint sets and adopt both the "union by rank" and "path compression" heuristics, the time spent on all the Find_Set, Make_Set, and Union operations in total for the MST construction is $O(|E| \log |V|)$.

- So, the total amount of running time of Kruskal's is $O(|E| \log |V|)$.