15 Minimum Spanning Trees

When a graph is connected, we may ask how many edges we can delete before it stops being connected. Depending on the edges we remove, this may happen sooner or later. The slowest strategy is to remove edges until the graph becomes a tree. Here we study the somewhat more difficult problem of removing edges with a maximum total weight. The remaining graph is then a tree with minimum total weight. Applications that motivate this question can be found in life support systems modeled as graphs or networks, such as telephone, power supply, and sewer systems.

Free trees. An undirected graph (U, T) is a *free tree* if it is connected and contains no cycle. We could impose a hierarchy by declaring any one vertex as the root and thus obtain a *rooted tree*. Here, we have no use for a hierarchical organization and exclusively deal with free trees. The

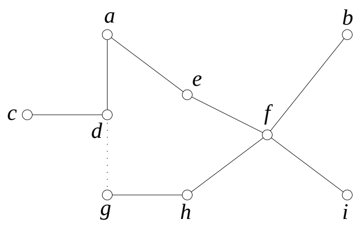

Figure 63: Adding the edge dg to the tree creates a single cycle with vertices d, g, h, f, e, a.

number of edges of a free tree is always one less than the number of vertices. Whenever we add a new edge (connecting two old vertices) we create exactly one cycle. This cycle can be destroyed by deleting any one of its edges, and we get a new free tree, as in Figure 63. Let (V, E) be a connected and undirected graph. A *subgraph* is another graph (U, T) with $U \subseteq V$ and $T \subseteq E$. It is a *spanning tree* if it is a free tree with $U = V$.

Minimum spanning trees. For the remainder of this section, we assume that we also have a weighting function, $w : E \to \mathbb{R}$. The *weight* of subgraph is then the total weight of its edges, $w(T) = \sum_{e \in T} w(e)$. A *minimum spanning tree*, or *MST* of G is a spanning tree that minimizes the weight. The definitions are illustrated in Figure 64 which shows a graph of solid edges with a minimum spanning tree of bold edges. A generic algorithm for constructing an MST grows a tree by adding more and

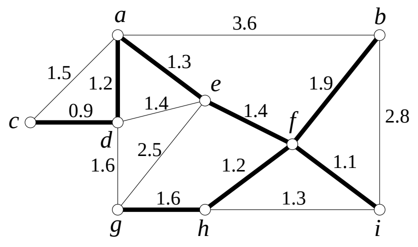

Figure 64: The bold edges form a spanning tree of weight $0.9 + 1.2 + 1.3 + 1.4 + 1.1 + 1.2 + 1.6 + 1.9 = 10.6$.

more edges. Let $A \subseteq E$ be a subset of some MST of a connected graph (V, E). An edge $uv \in E - A$ is *safe for* A if $A \cup \{uv\}$ is also subset of some MST. The generic algorithm adds safe edges until it arrives at an MST.

```
$A = \emptyset$;
while $(V, A)$ is not a spanning tree do
  find a safe edge $uv$;  $A = A \cup \{uv\}$
endwhile.
```

As long as A is a proper subset of an MST there are safe edges. Specifically, if (V, T) is an MST and $A \subseteq T$ then all edges in $T - A$ are safe for A. The algorithm will therefore succeed in constructing an MST. The only thing that is not yet clear is how to find safe edges quickly.

Cuts. To develop a mechanism for identifying safe edges, we define a *cut*, which is a partition of the vertex set into two complementary sets, $V = W \dot{\cup} (V - W)$. It is *crossed* by an edge $uv \in E$ if $u \in W$ and $v \in V - W$, and it *respects* an edge set A if A contains no crossing edge. The definitions are illustrated in Figure 65.

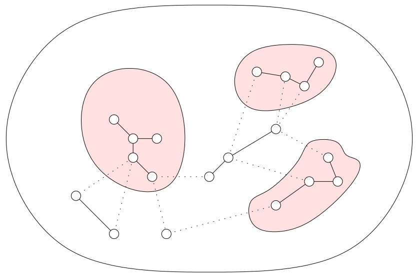

Figure 65: The vertices inside and outside the shaded regions form a cut that respects the collection of solid edges. The dotted edges cross the cut.

CUT LEMMA. Let A be subset of an MST and consider a cut $W \dot\cup (V - W)$ that respects A. If uv is a crossing edge with minimum weight then uv is safe for A.

PROOF. Consider a minimum spanning tree (V, T) with $A \subseteq T$. If $uv \in T$ then we are done. Otherwise, let $T' = T \cup \{uv\}$. Because T is a tree, there is a unique path from u to v in T. We have $u \in W$ and $v \in V - W$, so the path switches at least once between the two sets. Suppose it switches along xy, as in Figure 66. Edge xy

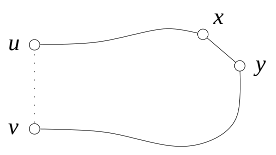

Figure 66: Adding uv creates a cycle and deleting xy destroys the cycle.

crosses the cut, and since A contains no crossing edges we have $xy \notin A$. Because uv has minimum weight among crossing edges we have $w(uv) \leq w(xy)$. Define $T'' = T' - \{xy\}$. Then (V, T'') is a spanning tree and because

$$w(T'') \;=\; w(T) - w(xy) + w(uv) \;\leq\; w(T)$$

it is a minimum spanning tree. The claim follows because $A \cup \{uv\} \subseteq T''$. ⊒

A typical application of the Cut Lemma takes a component of (V, A) and defines W as the set of vertices of that component. The complementary set $V - W$ contains all other vertices, and crossing edges connect the component with its complement.

Prim's algorithm. Prim's algorithm chooses safe edges to grow the tree as a single component from an arbitrary first vertex s. Similar to Dijkstra's algorithm, the vertices that do not yet belong to the tree are stored in a priority queue. For each vertex i outside the tree, we define its priority $V[i].d$ equal to the minimum weight of any edge that connects i to a vertex in the tree. If there is no such edge then $V[i].d = \infty$. In addition to the priority, we store the index of the other endpoint of the minimum weight edge. We first initialize this information.

```
$V[s].d = 0$; $V[s].\pi = -1$; INSERT($s$);
forall vertices $i \neq s$ do
  $V[i].d = \infty$; INSERT($i$)
endfor.
```

The main algorithm expands the tree by one edge at a time. It uses marks to distinguish vertices in the tree from vertices outside the tree.

```
while priority queue is non-empty do
  $i$ = EXTRACTMIN; mark $i$;
  forall neighbors $j$ of $i$ do
    if $j$ is unmarked and $w(ij) < V[j].d$ then
      $V[j].d = w(ij)$; $V[j].\pi = i$
    endif
  endfor
endwhile.
```

After running the algorithm, the MST can be recovered from the π-fields of the vertices. The algorithm together with its initialization phase performs $n = \text{card}\, V$ insertions into the priority queue, n extractmin operations, and at most $m = \text{card}\, E$ decreasekey operations. Using the Fibonacci heap implementation, we get a running time of O($n \log n + m$), which is the same as for constructing the shortest-path tree with Dijkstra's algorithm.

Kruskal's algorithm. Kruskal's algorithm is another implementation of the generic algorithm. It adds edges in a sequence of non-decreasing weight. At any moment, the chosen edges form a collection of trees. These trees merge to form larger and fewer trees, until they eventually combine into a single tree. The algorithm uses a priority queue for the edges and a set system for the vertices. In this context, the term 'system' is just another word for 'set', but we will use it exclusively for sets whose elements are themselves sets. Implementations of the set system will be discussed in the next lecture. Initially, $A = \emptyset$, the priority queue contains all edges, and the system contains a singleton set for each vertex, $C = \{\{u\} \mid u \in V\}$. The algorithm finds an edge with minimum weight that connects two components defined by A. We set W equal to the vertex set of one component and use the Cut Lemma to show that this edge is safe for A. The edge is added to A and the process is repeated. The algorithm halts when only one tree is left, which is the case when A contains $n - 1 = \text{card}\, V - 1$ edges.

```
$A = \emptyset$;
while $\text{card}\, A < n - 1$ do
  $uv$ = EXTRACTMIN;
  find $P, Q \in C$ with $u \in P$ and $v \in Q$;
  if $P \neq Q$ then
    $A = A \cup \{uv\}$; merge $P$ and $Q$
  endif
endwhile.
```

The running time is $O(m \log m)$ for the priority queue operations plus some time for maintaining C. There are two operations for the set system, namely finding the set that contains a given element, and merging two sets into one.

An example. We illustrate Kruskal's algorithm by applying it to the weighted graph in Figure 64. The sequence of edges sorted by weight is cd, fi, fh, ad, ae, hi, de, ef, ac, gh, dg, bf, eg, bi, ab. The evolution of the set system

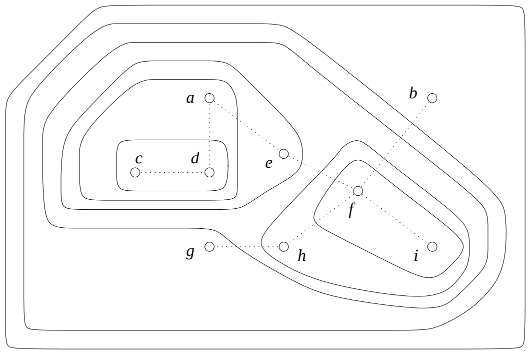

Figure 67: Eight union operations merge the nine singleton sets into one set.

is illustrated in Figure 67, and the MST computed with Kruskal's algorithm and indicated with dotted edges is the same as in Figure 64. The edges cd, fi, fh, ad, ae are all added to the tree. The next two edge, hi and de, are not added because they each have both endpoints in the same component, and adding either edge would create a cycle. Edge ef is added to the tree giving rise to a set in the system that contains all vertices other than g and b. Edge ac is not added, gh is added, dg is not, and finally bf is added to the tree. At this moment the system consists of a single set that contains all vertices of the graph.

As suggested by Figure 67, the evolution of the construction can be interpreted as a hierarchical clustering of the vertices. The specific method that corresponds to the evolution created by Kruskal's algorithm is referred to as single-linkage clustering.