

## Lecture 3

# Minimum Spanning Trees I

*Supplemental reading in CLRS: Chapter 4; Appendix B.4, B.5; Section 16.2*

### 3.1 Greedy Algorithms

As we said above, a **greedy algorithm** is an algorithm which attempts to solve an optimization problem in multiple stages by making a “locally optimal” decision at each stage.

**Example.** We wish to make 99¢ in change using the minimal number of coins. Most people instinctively use a greedy algorithm:

$$\begin{array}{r} 99\text{¢} = (25\text{¢}) \times 3 + \\ \underline{-75\text{¢}} \\ 24\text{¢} = (10\text{¢}) \times 2 + \\ \underline{-20\text{¢}} \\ 4\text{¢} = (1\text{¢}) \times 4 \text{ (no nickels)} \\ \underline{-4\text{¢}} \\ 0\text{¢} \end{array}$$

---

3 quarters + 2 dimes + 4 pennies = 9 coins.

This greedy algorithm is correct: starting with the coin of largest value, take as many as possible without allowing your total to exceed 99¢. However, plenty of greedy algorithms are not correct. For example, suppose we started with the smallest coin rather than the largest coin:

$$99\text{¢} = (1\text{¢}) \times 99 \implies 99 \text{ pennies.}$$

Or, imagine trying to make 15¢ of change if the dime were replaced by an 11¢ piece:

$$\begin{array}{l} \text{Greedy: } 15\text{¢} = (11\text{¢}) \times 1 + (5\text{¢}) \times 0 + (1\text{¢}) \times 4 \implies 5 \text{ coins} \\ \text{Optimal: } 15\text{¢} = (11\text{¢}) \times 0 + (5\text{¢}) \times 3 + (1\text{¢}) \times 0 \implies 3 \text{ coins.} \end{array}$$

**Remark.** Greedy algorithms sometimes give a correct solution (are globally optimal). But even when they’re not correct, greedy algorithms can be useful because they provide “good solutions” efficiently. For example, if you worked as a cashier in a country with the 11¢ piece, it would be perfectly reasonable to use a greedy algorithm to choose which coins to use. Even though you won’t always use the smallest possible number of coins, you will still always make the correct amount of change, and the number of coins used will always be close to optimal.

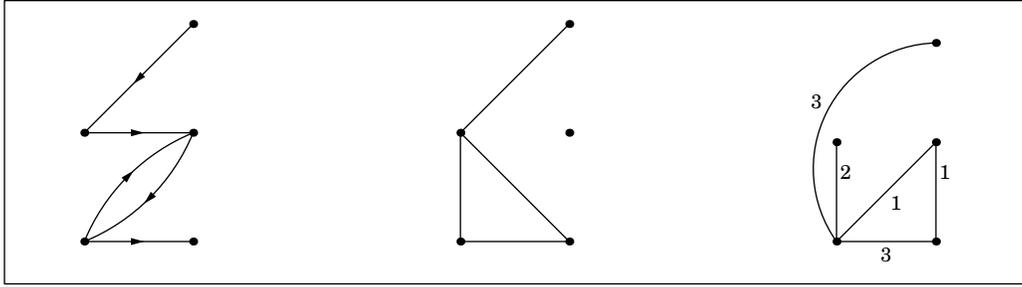


Figure 3.1. From left to right: A directed graph, an undirected graph, and a weighted undirected graph.

The above example shows that the performance of a greedy algorithm (in terms of both correctness and efficiency) depends on both

- the structure of the algorithm (*starting with big coins vs. small coins*)
- the structure of the problem (*coin values*).

### 3.2 Graphs

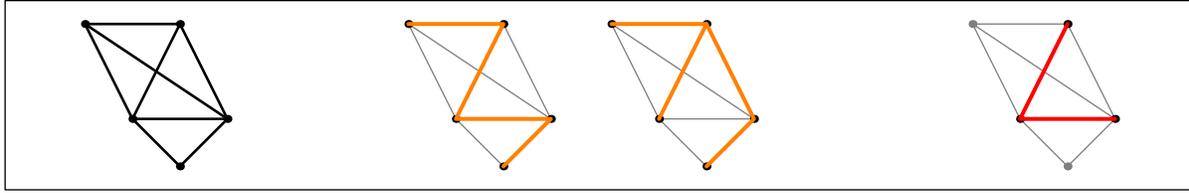
We assume that you have seen some graph theory before. We will give a quick review here; if you are shaky on graph theory then you should review §B.4 of CLRS. A **graph** is a mathematical object which consists of vertices and edges. Thus, we may say that a graph is a pair  $G = (V, E)$ , where  $V$  is a set of vertices and  $E$  is a set of edges. Or, if the vertex set  $V$  has been fixed ahead of time, we may simply say that “ $E$  is a graph on  $V$ .” In an **undirected graph**, the edges are unordered pairs  $e = \{u, v\}$  with  $u, v \in V$ ; whereas in a **directed graph**, the edges are ordered pairs  $e = (u, v)$ . A directed edge  $e = (u, v)$  is said to emanate “from  $u$  to  $v$ ,” and we may use the notation  $e = (u \rightarrow v)$ .<sup>1</sup> Our convention (which is by no means standard convention, since there is no standard) will be that all graphs are assumed finite, and are allowed to have loops (edges  $u \rightarrow u$ ) but may not have multiple edges between the same pair of vertices. Thus  $E$  is just a set, not a multiset (though in a directed graph,  $u \rightarrow v$  and  $v \rightarrow u$  count as different edges).

It is often useful to consider graphs with satellite data attached to the edges or vertices (or both). One common instance of this is a **weighted undirected graph**, which is an undirected graph equipped with a weight function  $w : E \rightarrow \mathbb{R}$ . We say that the edge  $e = \{u, v\}$  has *weight*  $w(e)$ , or sometimes  $w_{uv}$ . For example, let  $V$  be the set of cities in the United States, and let  $E$  be a **complete graph** on  $V$ , meaning that  $E$  contains every pair of vertices  $\{u, v\}$  with  $u \neq v$  (but no loops). Let  $w(u, v)$  be the distance in miles between  $u$  and  $v$ . The graph  $G = (V, E, w)$  is obviously of great importance to airline booking companies, shipping companies, and presidential candidates in the last few months of campaigning.

Consider a fixed set  $V$  of vertices. A **tree**  $T$  on  $V$  is a graph which is acyclic and connected.<sup>2</sup> (See Figure 3.2.) Note that acyclic is a smallness condition, while connectivity is a largeness condition. Thus, trees are “just the right size.” According to the following proposition, that size is  $|V| - 1$ .

<sup>1</sup> In fact, we will probably use several other notations, depending on our whim; in particular, we will sometimes use the notation of directed graphs in discussions of undirected graphs. It is the author’s job to keep clear the meaning of the notation—you should speak up if you are confused by any notation, since it is more likely the author’s fault than yours.

<sup>2</sup> A **path** in a graph  $(V, E)$  is a sequence of vertices  $\langle v_0, v_1, \dots, v_n \rangle$ , where  $(v_j \rightarrow v_{j+1}) \in E$  for  $j = 0, \dots, n - 1$ . The number  $n$  is called the *length* of the path. If  $v_0 = v_n$ , then the path is called a **cycle**. A graph is called **acyclic** if it contains no cycles, and is said to be **connected** if, for every pair  $u, v$  of vertices, there exists a path from  $u$  to  $v$ .



**Figure 3.2.** Left: An undirected graph  $G$ . Center: Two different spanning trees of  $G$ . Right: A subgraph of  $G$  which is a tree but not a spanning tree. The gray vertices and edges are just visual aids; they are not part of the graphs.

**Proposition 3.1** (Theorem B.2 of CLRS, slightly modified). *Let  $V$  be a finite set of vertices, and let  $G = (V, E)$  be an undirected graph on  $V$ . The following conditions are equivalent:*

- (i)  $G$  is acyclic and connected.
- (ii)  $G$  is acyclic and has at least  $|V| - 1$  edges.
- (iii)  $G$  is connected and has at most  $|V| - 1$  edges.
- (iv)  $G$  is acyclic, but if any edge is added to  $T$ , the resulting graph contains a cycle.
- (v)  $G$  is connected, but if any edge is removed from  $T$ , the resulting graph is not connected.
- (vi) For any two vertices  $u, v \in V$ , there exists a unique simple path from  $u$  to  $v$ .<sup>3</sup>

Any one of these six conditions could be taken as the definition of a tree. If  $G$  is a tree, then  $G$  has exactly  $|V| - 1$  edges.

We will not prove the proposition here, but you would do yourself a service to write the proof yourself (or look it up in CLRS), especially if you have not seen proofs in graph theory before. There are certain types of arguments that occur again and again in graph theory, and you will definitely want to know them for the remainder of the course.

**Definition.** Let  $G = (V, E)$  be an undirected graph. A **spanning tree** of  $G$  is a subset  $T \subseteq E$  such that  $T$  is a tree on  $V$ .

The reason for the word “spanning” is that  $T$  must be a tree on all of  $V$ . There are plenty of subgraphs of  $G$  that are trees but not spanning trees: they are graphs of the form  $G' = (V', T')$  where  $V' \subsetneq V$  and  $T'$  is a tree on  $V'$  (but not on  $V$  because it does not touch all the vertices).

**Proposition 3.2.** *Let  $G = (V, E)$  be an undirected graph. Then  $G$  has a spanning tree if and only if  $G$  is connected. Thus, every connected graph on  $V$  has at least  $|V| - 1$  edges.*

*Proof.* If  $G$  has a spanning tree  $T$ , then for any pair of vertices  $u, v$  there is a path from  $u$  to  $v$  in  $T$ . That path also lies in  $E$  since  $T \subseteq E$ . Thus  $G$  is connected.

Conversely, suppose  $G$  is connected but is not a tree. Then, by Proposition 3.1(v), we can remove an edge from  $E$ , obtaining a connected subgraph  $E_1$ . If  $E_1$  is not a tree, then we can repeat the process again. Eventually we will come upon a connected subgraph  $(V, E_k)$  of  $G$  such that it is impossible to remove an edge from  $E_k$ , either because we have run out of edges, or because the resulting graph will be disconnected. In the former case,  $V$  must have only one vertex, so the empty set of edges is a spanning tree. In the latter case,  $E_k$  is a spanning tree by Proposition 3.1(v). It follows from Proposition 3.1 that  $G$  has at least  $|V| - 1$  edges, since any spanning tree has exactly  $|V| - 1$  edges.  $\square$

<sup>3</sup> A path is said to be **simple** if all vertices in the path are distinct. A cycle is said to be **simple** if all vertices are distinct except for the common start/end vertex. Thus, strictly speaking, a simple cycle is not a simple path; extra clarifications will be made in any situation where there might be confusion.

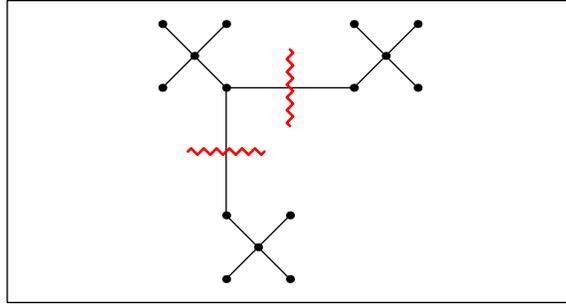


Figure 3.3. Clustering using an MST.

### 3.3 Minimum Spanning Trees

Given a weighted undirected graph  $G = (V, E, w)$ , one often wants to find a **minimum spanning tree** (MST) of  $G$ : a spanning tree  $T$  for which the total weight  $w(T) = \sum_{(u,v) \in T} w(u,v)$  is minimal.

**Input:** A connected, undirected weighted graph  $G = (V, E, w)$

**Output:** A spanning tree  $T$  such that the total weight

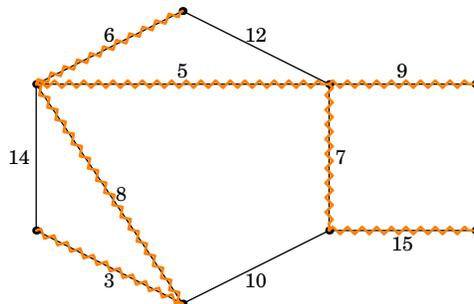
$$w(T) = \sum_{(u,v) \in T} w(u,v)$$

is minimal.

For example, if  $V$  is the set of buildings in a city and  $E$  is the complete graph on  $V$ , and if  $w(u,v)$  is the distance between  $u$  and  $v$ , then a minimum spanning tree would be very useful in building a minimum-length fiber-optic network, pipeline network, or other infrastructure for the city.

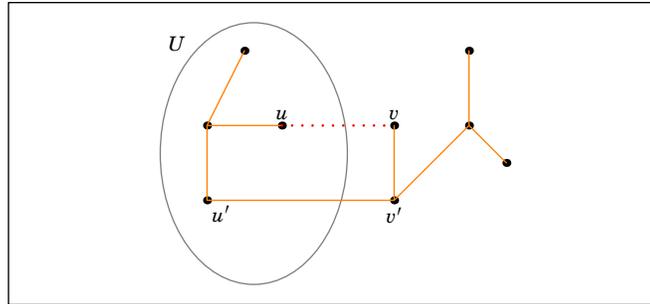
There are many other, less obvious applications of minimum spanning trees. Given a set of data points on which we have defined a metric (i.e., some way of quantifying how close together or similar a pair of vertices are), we can use an MST to cluster the data by starting with an MST for the distance graph and then deleting the heaviest edges. (See Figure 3.3 and Lecture 21.) If  $V$  is a set of data fields and the distance is mutual information, then this clustering can be used to find higher-order correlative relationships in a large data set.

How would we go about finding an MST for this graph?



There are several sensible heuristics:

- *Avoid large weights.* We would like to avoid the 14 and 15 edges if at all possible.



**Figure 3.4.** Illustration of the MST property.

- *Include small weights.* If the 3 and 5 edges provide any benefit to us, we will probably want to include them.
- *Some edges are inevitable.* The 9 and 15 edges must be included in order for the graph to be connected.
- *Avoid cycles,* since an MST is not allowed to have cycles.

Some more thoughts:

- Would a greedy algorithm be likely to work here? Is there something about the structure of the problem which allows locally optimal decisions to be informed enough about the global solution?
- Should we start with all the edges present and then remove some of them, or should we start with no edges present and then add them?
- Is the MST unique? Or are there multiple solutions?

A key observation to make is the following: If we add an edge to an MST, the resulting graph will have a cycle, by Proposition 3.1(iv). If we then remove one of the edges in this cycle, the resulting graph will be connected and therefore (by Proposition 3.1(iii)) will be a spanning tree. If we have some way of knowing that the edge we removed is at least as heavy as the edge we added, then it follows that we will have a minimum spanning tree. This is the idea underlying the so-called “MST property.”

**Theorem 3.3** (MST Property). *Let  $G = (V, E, w)$  be a connected, weighted, undirected graph. Let  $U$  be a proper nonempty subset of  $V$ .<sup>4</sup> Let  $S$  be the set of edges  $(x, y)$  with  $x \in U$  and  $y \in V \setminus U$ . Suppose  $(u, v)$  is the lightest edge (or one of the lightest edges) in  $S$ . Then there exists an MST containing  $(u, v)$ .*

This partition of  $V$  into  $U$  and  $V \setminus U$  is called a “cut.” An edge is said to “cross” the cut if one endpoint is in  $U$  and the other endpoint is in  $V \setminus U$ . Otherwise the edge is said to “respect” the cut. So  $S$  is the set of edges which cross the cut and  $E \setminus S$  is the set of edges that respect the cut. An edge in  $S$  of minimal weight is called a “light edge” for the cut.

<sup>4</sup>That is,  $\emptyset \subsetneq U \subsetneq V$ .

*Proof.* Let  $T \subseteq E$  be an MST for  $G$ , and suppose  $(u, v) \notin T$ . If we add the edge  $(u, v)$  to  $T$ , the resulting graph  $T'$  has a cycle. This cycle must cross the cut  $(U, V \setminus U)$  in an even number of places, so in addition to  $(u, v)$ , there must be some other edge  $(u', v')$  in the cycle, such that  $(u', v')$  crosses the cut. Remove  $(u', v')$  from  $T'$  and call the resulting graph  $T''$ . We showed above that  $T''$  is a spanning tree. Also, since  $(u, v)$  is a light edge, we have  $w(u, v) \leq w(u', v')$ . Thus

$$w(T'') = w(T) + w(u, v) - w(u', v') \leq w(T).$$

Since  $T$  is a minimum spanning tree and  $T''$  is a spanning tree, we also have  $w(T) \leq w(T'')$ . Therefore  $w(T) = w(T'')$  and  $T''$  is a minimum spanning tree which contains  $(u, v)$ .  $\square$

**Corollary 3.4.** *Preserve the setup of the MST property. Let  $T$  be any MST. Then there exists an edge  $(u', v') \in T$  such that  $u' \in U$  and  $v' \in V \setminus U$  and  $(T \setminus \{(u', v')\}) \cup \{(u, v)\}$  is an MST.*

*Proof.* In the proof of the MST property,  $T'' = (T \setminus \{(u', v')\}) \cup \{(u, v)\}$ .  $\square$

**Corollary 3.5.** *Let  $G = (V, E, w)$  be a connected, weighted, undirected graph. Let  $T$  be any MST and let  $(U, V \setminus U)$  be any cut. Then  $T$  contains a light edge for the cut. If the edge weights of  $G$  are distinct, then  $G$  has a unique MST.*

*Proof.* If  $T$  does not contain a light edge for the cut, then the graph  $T''$  constructed in the proof of the MST property weighs strictly less than  $T$ , which is impossible.

Suppose the edge weights in  $G$  are distinct. Let  $M$  be an MST. For each edge  $(u, v) \in M$ , consider the graph  $(V, M \setminus \{(u, v)\})$ . It has two connected components<sup>5</sup>; let  $U$  be one of them. The only edge in  $M$  that crosses the cut  $(U, V \setminus U)$  is  $(u, v)$ . Since  $M$  must contain a light edge, it follows that  $(u, v)$  is a light edge for this cut. Since  $G$  has distinct edge weights,  $(u, v)$  is the unique light edge for the cut, and every MST must contain  $(u, v)$ . Letting  $(u, v)$  vary over all edges of  $M$ , we find that every MST must contain  $M$ . Thus, every MST must equal  $M$ .  $\square$

Now do you imagine that a greedy algorithm might work? The MST property is the crucial idea that allows us to use local information to decide which edges belong in an MST. Below is **Kruskal's algorithm**, which solves the MST problem.

**Algorithm:** KRUSKAL-MST( $V, E, w$ )

Initially, let  $T \leftarrow \emptyset$  be the empty graph on  $V$ .

Examine the edges in  $E$  in increasing order of weight (break ties arbitrarily).

- If an edge connects two unconnected components of  $T$ , then add the edge to  $T$ .\*
- Else, discard the edge and continue.

Terminate when there is only one connected component. (Or, you can continue through all the edges.)

\* By "two unconnected components," we mean two distinct connected components. Please forgive me for this egregious abuse of language.

<sup>5</sup> Given a graph  $G$ , we say a vertex  $v$  is **reachable** from  $u$  if there exists a path from  $u$  to  $v$ . For an undirected graph, reachability is an equivalence relation on the vertices. The restrictions of  $G$  to each equivalence class are called the **connected components** of  $G$ . (The **restriction** of a graph  $G = (V, E)$  to a subset  $V' \subseteq V$  is the subgraph  $(V', E')$ , where  $E'$  is the set of edges whose endpoints are both in  $V'$ .) Thus each connected component is a connected subgraph, and a connected undirected graph has only one connected component.

Try performing Kruskal's algorithm by hand on our example graph.

*Proof of correctness for Kruskal's algorithm.* We will use the following loop invariant:

Prior to each iteration, every edge in  $T$  is a subset of an MST.

- *Initialization.*  $T$  has no edges, so of course it is a subset of an MST.
- *Maintenance.* Suppose  $T \subseteq T^*$  where  $T^*$  is an MST, and suppose the edge  $(u, v)$  gets added to  $T$ . Then, as we will show in the next paragraph,  $(u, v)$  is a light edge for the cut  $(U, V \setminus U)$ , where  $U$  is one of the connected components of  $T$ . Therefore, by Corollary 3.4, there exist vertices  $u' \in U$  and  $v' \in V \setminus U$  with  $(u', v') \in T^*$  such that  $T'$  is an MST, where  $T' = (T^* \setminus \{(u', v')\}) \cup \{(u, v)\}$ . But  $(u', v') \notin T$ , so  $T \cup \{(u, v)\}$  is a subset of  $T'$ .

As promised, we show that  $(u, v)$  is a light edge for the cut  $(U, V \setminus U)$ , where  $U$  is one of the connected components of  $T$ . The reasoning is this: If  $(u, v)$  is added, then it joins two unconnected components of  $T$ . Let  $U$  be one of those components. Now suppose that  $(x, y) \in E$  were some lighter edge crossing the cut  $(U, V \setminus U)$ , say  $x \in U$  and  $y \in V \setminus U$ . Since  $x$  and  $y$  lie in different connected components of  $T$ , the edge  $(x, y)$  is not in  $T$ . Thus, when we examined  $(x, y)$ , we decided not to add it to  $T$ . This means that, when we examined the edge  $(x, y)$ ,  $x$  and  $y$  lay in the same connected component of  $T_{\text{prev}}$ , where  $T_{\text{prev}}$  is what  $T$  used to be at that point in time. But  $T_{\text{prev}} \subseteq T$ , which means that  $x$  and  $y$  lie in the same connected component of  $T$ . This contradicts the fact that  $x \in U$  and  $y \in V \setminus U$ .

- *Termination.* At termination,  $T$  must be connected: If  $T$  were disconnected, then there would exist an edge of  $E$  which joins two components of  $T$ . But that edge would have been added to  $T$  when we examined it, a contradiction. Now, since  $T$  is connected and is a subset of an MST, it follows that  $T$  is an MST.  $\square$

To implement Kruskal's algorithm and analyze its running time, we will first need to think about the data structures that will be used and the running time of operations on them.

**Exercise 3.1.** *What if we wanted a maximum spanning tree rather than a minimum spanning tree?*

MIT OpenCourseWare  
<http://ocw.mit.edu>

6.046J / 18.410J Design and Analysis of Algorithms  
Spring 2012

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.