Lecture 4: Trees, part II

September 10, 2020

Weighted graphs

Definition. A weighted graph is a finite graph G, with a function (length or cost) $c\colon E(G) \to \mathbf{R}$ (usually the image is in $\mathbf{R}_{\geq 0}$).

Problem. If T is a spanning tree of G, we put $c(T) := \sum_{e \in E(T)} c(e)$ and want to find T such that $c(T)$ is minimal.

Remark. It makes sense to restrict to trees: for example, if c takes values in $\mathbf{R}_{\geq 0}$, then for every connected spanning subgraph H of G which is not a tree, we can find another connected spanning subgraph with a lower total cost.

Example. Say G represents a network of cities and for an edge e joining x and y, we have $c(e)$ is the cost of building a telephone line between x and y. Finding a spanning tree of minimal total cost is thus useful in practice.

The Greedy Algorithm

We now discuss an algorithm for finding the cheapest spanning tree. This is a version of the Greedy Algorithm.

Idea: at each step, choose the best possible solution without looking back.

We first need some new terminology.
Definition. A finite graph G is a forest if it contains no polygons; equivalently, all connected components of G are trees.

Recall: we showed last time that if G is a finite connected graph, then $\#E(G) \geq \#V(G) - 1$, with equality if and only if G is a tree. If G is not necessarily connected, by applying this to each connected component of G, we see that

$$\#E(G) \geq \#V(G) - \#\{\text{connected components of } G\},$$

with equality if and only if G is a forest.

The Greedy Algorithm, cont'd

Notation. If G is a graph and S is a subset of $E(G)$, then $(G : S)$ denotes the subgraph of G with the same vertex set as G and with set of edges equal to S.

Definition. If G is a finite graph, a subset $S \subseteq E(G)$ is independent if $(G : S)$ is a forest.

Algorithm. Let G be a finite, connected, weighted graph. Let $n = \#V(G)$.

Step 1. Choose $e_1 \in E(G)$ such that $c(e_1)$ is minimal. Note: the graph $\big(G : \{e_1\}\big)$ is a forest, with $n-1$ connected components.

Step $i \leq n-1$. Suppose that we have chosen $e_1, \ldots, e_{i-1} \in E(G)$ and we know that $\big(G : \{e_1, \ldots, e_{i-1}\}\big)$ is a forest, with $n-i+1$ connected components.

The Greedy Algorithm, cont'd

Step i, cont'd. We choose $e_i \in E(G)$ with ends in two different components of $\big(G : \{e_1, \ldots, e_{i-1}\}\big)$ and such that $c(e_i)$ is minimal among such possibilities.

Note: 1) There are such edges e_i since G is connected and $n - i + 1 \geq 2$.

2) The condition on e_i implies that $\big(G : \{e_1, \ldots, e_i\}\big)$ is a forest with $n - i$ connected components.
Indeed, say e_i has ends in the connected components A and B of $\big(G : \{e_1, \ldots, e_{i-1}\}\big)$. It is then clear that we have a connected component H of $\big(G : \{e_1, \ldots, e_i\}\big)$ with $V(H) = V(A) \sqcup V(B)$. Moreover, H is a tree: any closed path in H (of positive length) has to be contained in either A or B (it can pass between A and B only using e).

We stop after $n - 1$ such steps, when $T = \big(G : \{e_1, \ldots, e_{n-1}\}\big)$ is a forest with 1 connected component, hence a tree. Since it has the same vertex set as G, it is a spanning tree of G.

The Greedy Algorithm, cont'd

Key claim. For every spanning tree T' of G, we have $c(T) \leq c(T')$.

In order to prove the claim, note that T' has $n-1$ edges and let us order them as $a_1, \ldots, a_{n-1}$, such that $c(a_1) \leq c(a_2) \leq \ldots \leq c(a_{n-1})$. In fact, we will prove the stronger

Claim'. For every $1 \leq i \leq n-1$, we have $c(e_i) \leq c(a_i)$.

We argue by contradiction. Suppose that $i \leq n-1$ is such that $c(e_i) > c(a_i)$. Note that we also have

$$c(a_i) \geq c(a_{i-1}) \geq \ldots \geq c(a_1).$$

Since $c(e_i)$ is strictly larger than all $c(a_j)$, with $j \leq i$, we deduce from the fact that at Step i none of the edges $a_1, \ldots, a_i$ was chosen that each of $a_1, \ldots, a_i$ had both ends in the same connected component of $\big(G : \{e_1, \ldots, e_{i-1}\}\big)$.

The Greedy Algorithm, cont'd

This implies that

$$r := \text{number of connected components of } \big(G : \{a_1, \ldots, a_i\}\big)$$

$$\geq \text{number of connected components of } \big(G : \{e_1, \ldots, e_{i-1}\}\big) = n - i + 1.$$

Hence we have $r \geq n - i + 1$. On the other hand, since T' is a tree, it follows that $\big(G : \{a_1, \ldots, a_i\}\big)$ is a forest. Therefore we have

$$i = \#E\big(G : \{a_1, \ldots, a_i\}\big) = \#V\big(G : \{a_1, \ldots, a_i\}\big) - r \leq n - r.$$

This contradicts the previous inequality.
We have thus shown that $c(e_i) \leq c(a_i)$ for all $i \leq n - 1$. In particular, we have $c(T) \leq c(T')$.

Conclusion. The algorithm exhibits a spanning tree of minimal total cost.

Removing an edge from a tree

Let's take a break, to discuss in groups the following

Problem. Show that if T is a tree on $n \geq 2$ vertices and T' is the graph obtained from T by removing one edge, then T' is a forest with two connected components.

Depth-first search tree

We next discuss a certain construction of spanning trees that we will then use to construct orientations.
Let G be a finite connected graph. We discuss an algorithm that produces:
- a numbering of the vertices of G and
- a spanning tree for G (depth-first search tree for G).

Description of the procedure.

Step 0. Pick an arbitrary vertex v_0 of G. We consider the tree T_0 with vertex v_0 and no edges.

Step $k+1$. Suppose that we have already picked distinct vertices $v_0, \ldots, v_k$ of G and a subgraph T_k of G on these vertices, which is a tree.

Depth-first search tree, cont'd

Step $k+1$, cont'd. If $k+1 < \#V(G)$, since G is connected, there is $\ell \leq k$ such that v_ℓ is adjacent to a vertex of G which is not a vertex of T_k. We choose ℓ maximal with this property and one such edge, whose end different from v_ℓ is denoted v_{k+1} (note: while ℓ is uniquely determined by maximality, both v_{k+1} and the edge are not unique).
We let T_{k+1} be the graph obtained from T_k by adding v_{k+1} and this edge. We have already seen (Lemma 3 from Lecture 3) that in this case T_{k+1} is a tree as well.

On the other hand, if $k+1 = \#V(G)$, then the procedure stop. In this case T_k is a spanning tree of G.

Conclusion. We end with a numbering of the vertices of G and a spanning tree of G.

Next: we discuss some properties of this construction, but first, we need some further terminology about trees.

Some terminology about trees

Recall: a path in a graph is a walk of positive length in which all edges are distinct. We have seen that if x and y are distinct vertices of a tree T, then there is a unique path from x to y.

Definition. A root in a tree T is a choice of a vertex x_0 of T. In this case one also says that T is a rooted tree.

Definition. Given a rooted tree T with root x_0, the ancestors of a vertex $x \in V(T)$ are the vertices of T on the unique path from x_0 to x. If y is an ancestor of x, we also say that x is a descendent of y.
Convention. x is both an ancestor and a descendent of himself.

Definition. If $x \neq x_0$, then the closest vertex to x on the unique path from x_0 to x is the parent of x.
Remark. Given an edge joining x and y in a rooted tree T, we see that either x is the parent of y or y is the parent of x.

Properties of depth-first search trees

Suppose now that T is a depth-first search tree in G, constructed as described. We consider T to be rooted, with root v_0.

Remark. By construction, the numbering is increasing along any path starting from v_0; in other words, if v_j is a descendent of v_i, then $j \geq i$. This follows from the fact that, by construction, for every vertex x we have a path with this property from v_0 to x and since T is a tree, this path is unique.

Proposition 1. If (T, v_0) is as above and x, y are adjacent vertices of G, then either x is a descendent of y or y is a descendent of x.

Proof of Proposition 1. Given adjacent vertices x and y, let k and k' be such that $x = v_k$ and $y = v_{k'}$. Suppose, for example, that $k < k'$ (note that if $x = y$, which can be the case if G has a loop at x, the assertion in the proposition is clear).

Properties of depth-first search trees, cont'd

Note first that we have a path in T from v_0 to v_k only involving v_i, with $i \leq k$. We show by induction on q, with $k+1 \leq q \leq k'$, that the unique path in T from v_k to v_q only involves v_i, with $k \leq i \leq q$. This implies that v_q is a descendent of v_k; by taking $q = k'$ we obtain the assertion in the proposition.

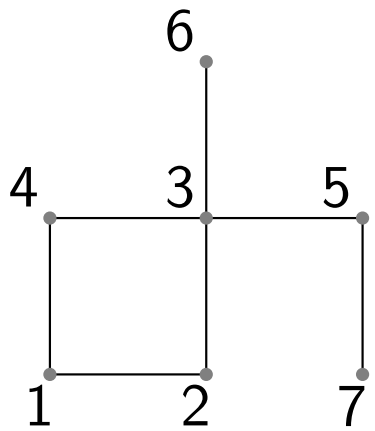
Since $\{v_k, v_{k'}\} \in E(G)$, it follows from the construction of T that when v_q is picked, we have some ℓ with $k \leq \ell \leq q-1$ such that $\{v_\ell, v_q\} \in E(T)$. If $q = k+1$, then $\ell = k$ and $\{v_k, v_q\} \in E(T)$.

If $q > k+1$, then by induction the unique path in T from v_k to v_ℓ only involves v_i, with $k \leq i \leq \ell$. By concatenating this with $\{v_\ell, v_q\}$, we obtain the unique path joining v_k and v_q. This clearly satisfies the requirement.

Definition of isthmus

Definition. If G is a connected graph, an isthmus (or bridge) is an edge of G whose deletion results in a disconnected graph.

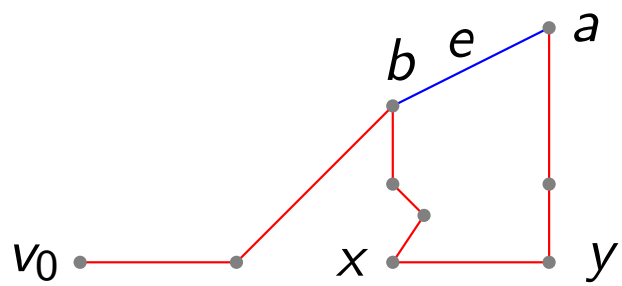
Example. In the following graph



the isthmuses are $\{3,5\}$, $\{3,6\}$, and $\{5,7\}$.

Another property of depth-first search trees

Proposition 2. Let T be a depth-first search tree of G, with root at v_0. If $\{x, y\}$ is an edge in T which is not an isthmus in G, with x the parent of y, then there is an edge e in G, but not in T, joining a descendent a of y with an ancestor b of x.



Here: the red edges are in $E(T)$ and the blue edge is in $E(G) \setminus E(T)$.

Proof. Let D be the set of descendents of y in T. We clearly have $y \in D$, but $x \notin D$ (since x is the parent of y).

Another property of depth-first search trees, cont'd

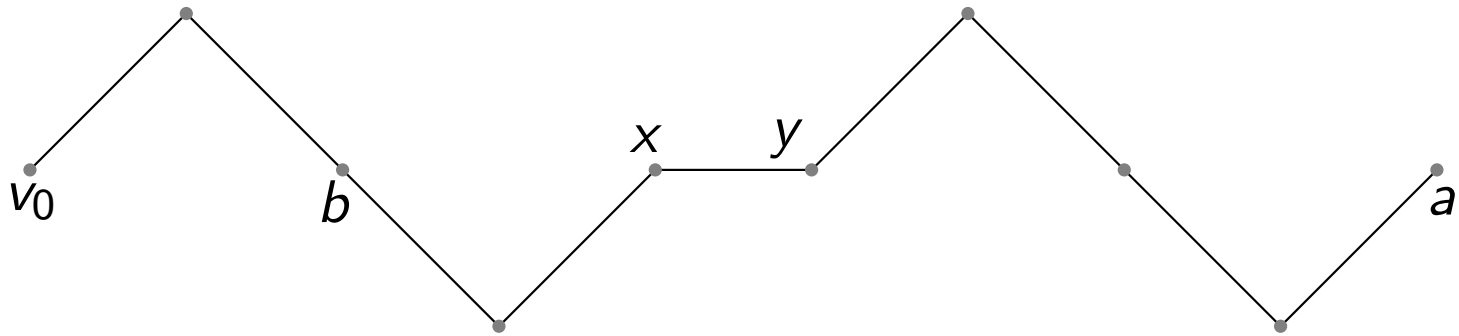
Since $\{x, y\}$ is not an isthmus in G, the graph obtained from G by deleting this edge is connected. Therefore there is an edge $\{a, b\} \in E(G)$ different from $\{x, y\}$, with $a \in D$ and $b \notin D$ (consider the edges on a walk in G from x to y not involving $\{x, y\}$ and use the fact that $y \in D$, but $x \notin D$). Since a, b are adjacent in G, it follows from Prop. 1 that either b is a descendent of a in T or a is a descendent of b in T.

If b is a descendent of a, since a is a descendent of y, it follows that b is a descendent of y, too. This is a contradiction, hence a is a descendent of b.

Another property of depth-first search trees, cont'd

We thus know that a is a descendent of b.

Consider the unique path in T from v_0 to a. Since a is a descendent of y, it follows that y appears on this path. Since x is the parent of y, it follows that x appears on this path right before y. Also, b has to appear on this path and since $b \notin D$, it must appear on the part of the path joining v_0 with x. Therefore b is an ancestor of x.



Since $\{a, b\} \notin E(T)$ (as T contains no polygons), we are done.

An application to constructing orientations

We use the properties of depth-based search trees to prove the following **Theorem**. If G is a finite connected graph with no isthmuses, then G admits an orientation that makes it a strongly connected digraph.

Recall: A digraph consist of a graph, together with an "orientation" of each edge. A walk in a digraph is strong if each edge is traversed according to its direction (from tail to head). A digraph is strongly connected if for every two vertices x and y, there is a strong walk from x to y.

Proof of the theorem. We need to choose an orientation for each edge. We may and will assume that G has no loops: these only have one possible orientation and they do not matter for existence of strong walks. We can thus remove all loops.

An application to constructing orientations

We first construct a depth-first search tree T for G. In particular, we get a numbering $v_0, \ldots, v_n$ of the vertices of G.

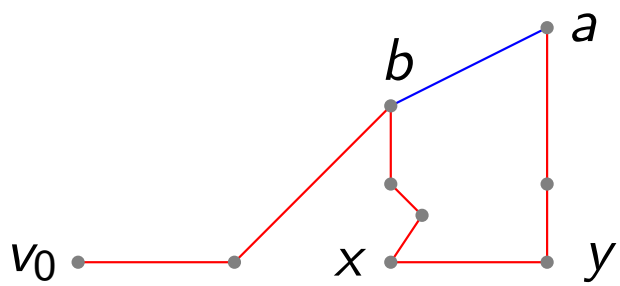
Given an edge $\{v_i, v_j\} \in E(G)$, with $i < j$, we orient it as follows
- we orient it as $v_i \to v_j$ if $\{v_i, v_j\} \in E(T)$, and
- we orient it as $v_j \to v_i$, otherwise.

Let us show that thus oriented, G is strongly connected. Note first that we have a strong walk from v_0 to any vertex of G (we can simply use the path in T). Therefore it is enough to show that for every vertex y of G, there is a strong walk from y to v_0.

Suppose that $y = v_k$. We argue by induction on k, the case $k = 0$ being trivial. If $k \geq 1$, let x be the parent of y with respect to the rooted tree (T, v_0).

An application to constructing orientations

We apply Proposition 2 for x and y to conclude that there is an edge $\{a, b\}$ in G, not in T, such that a is a descendent of y and $b = v_i$ is an ancestor of x (note that $i < k$).



We then have a strong path from y to b, by going from y to a in T and then using $\{a, b\}$. Since $i < k$, it follows by induction that we have a strong path from b to v_0. Concatenating this to the one from y to b, we get a strong path from y to b.

We thus conclude that G is strongly connected.