1 Trees

1.1 Minimum Spanning Trees • 1.2 Breadth-First Search Trees • 1.3 Directed Trees

1.1 Minimum Spanning Trees

In this subsection graphs will be undirected and weighted (by a function $w : E(G) \to \mathbb{R}$).

Definitions
A graph T is a *tree* if it satisfies one of the following equivalent conditions:

- T is connected and acyclic;
- T is minimally connected (removing any edge disconnects it);
- T is maximally acyclic (adding any edge creates a cycle);
- T is connected and $|E(T)| = |V(T)| - 1$;
- for any $u, v \in V(T)$, there is a unique path in T between u and v.

A *spanning tree* of a graph G is a subgraph T with $V(T) = V(G)$ which is a tree.
A *minimum spanning tree* of G is a spanning tree T such that $w(T) = \sum_{e \in E(T)} w(e)$ is minimum among all spanning trees of G.
A graph is a *forest* if it is acyclic; equivalently, it is a disjoint union of trees.
Finally for $S \subset V(G)$ we define $\delta(S) = \{e \in E(G) : e \cap S = 1\}$ (recall that an edge is a set of two vertices, so $\delta(S)$ is the set of edges with exactly one vertex in S).

Greedy Tree-Growing Algorithm
We want an algorithm that, given a weighted undirected graph G, finds a minimum spanning tree, or determines that there is none (which implies the graph is disconnected).
The easiest thing to try is a greedy approach: repeatedly add a smallest edge, while preserving connectedness and acyclicity (i.e. you maintain a tree all along). More precisely:

Tree-Growing Algorithm

1. Start with T being a single vertex;
2. Find $e \in \delta(T)$ with minimum $w(e)$; if $\delta(T) = \emptyset$, go to 4;
3. Add e to T; go to 2;
4. If $|E(T)| = |V(G)| - 1$, return T, else return "disconnected".

This algorithm is usually called *Prim's algorithm.*

Theorem 1.1. *The Tree-Growing Algorithm returns a minimum spanning tree, if it exists.*

Proof 1. Suppose it returns T. Let T^* be a minimum spanning tree that has as many edges as possible in common with T. If $T = T^*$, then we are done, so assume it isn't.
Let e be the first edge not in T^* that is picked by the algorithm, and let T' be the tree that the algorithm had before it picked e. Since $e \notin T^*$, adding it creates a cycle. Some of the edges in that cycle must not be in T', otherwise the algorithm couldn't have picked e. Let f be such an edge with one endpoint in T' and the other not (this is possible because one endpoint of e is in T'). Then since the algorithm chose e over f, we must have $w(e) \leq w(f)$. Now define $T^{**} = T^* - f + e$, so $w(T^{**}) \leq w(T^*)$, hence T^{**} is also a minimum spanning tree. But it has one more edge in common with T than T^*, contradicting the definition of T^*. It follows that $T = T^*$, and T is minimum. □

Proof 2. Call a tree *good* if it is contained in a minimum spanning tree. Clearly the empty graph is good. We will inductively show that all the trees occurring during the algorithm are good, hence so is the final tree T. Since T is clearly a spanning tree, it follows that it is minimum.
So suppose we are in step 2 with a good tree T', contained in a minimum spanning tree T^*. Then we want to show that $T' + e$ is also good, where $e \in \delta(T)$ with minimum $w(e)$. If $e \in T^*$, then this is obvious, so suppose $e \notin T^*$. Adding e to T^* creates a cycle, which must have some edge f that has only one endpoint in T'. Since the algorithm chose e over f, we have $w(f) \geq w(e)$. Then $T^{**} = T^* - f + e$ is a minimum spanning tree containing T' and e, which means $T' + e$ is good. □

Greedy Forest-Growing Algorithm
There are different greedy approaches for finding a minimum spanning tree, where different property is preserved in the process. For instance, one can just preserve acyclicity (i.e. maintain a forest), and in the end the connectedness will follow just from the number of edges. This leads to:

Forest-Growing Algorithm

1. Start with empty graph F, and set $S = E(G)$ (edges still to be done);
2. Find $e \in S$ with minimum $w(e)$; if $S = \emptyset$ or $|E(F)| = |V(G)| - 1$, go to 4;
3. Add e to F, unless that creates a cycle; remove e from S; go back to 2;
4. If $|E(F)| = |V(G)| - 1$, return F, else return "disconnected".

Theorem 1.2. *The Forest-Growing Algorithm returns a minimum spanning tree, if it exists.*

Proof. Is asked for on Problem Set 2. □

Equivalent problems

Two optimization problems are *equivalent* if one can translate one problem into the other, and the other way around, such that an algorithm for one will give an algorithm for the other. Technically the translation should not have too large running time in some sense, but we won't make that precise here (below the translations actually have linear running time).

Theorem 1.3. *Given a weighted undirected graph, the following* 4 *optimization problems are equivalent:*

- Minimum/Maximum Spanning Tree Problem: *Find a spanning tree with minimum/maximum weight, if one exists.*
- Minimum/Maximum Forest Problem: *Find a forest with minimum/maximum weight.*

Proof. The equivalence between the minimum and maximum versions follows simply by multiplying all weights by -1. For instance, to find a maximum spanning tree in a graph with weights $w(e)$, instead look for a minimum spanning tree in the same graph but with weights $-w(e)$. (Note that this trick happens to work for trees, but will not for other objects, for instance for shortest paths).

We will show that the maximum spanning tree and maximum forest problems are also equivalent, which completes the theorem.

Given a graph G in which to find a maximum forest, modify it to G' as follows: remove all edges with negative weight (would never be used in a maximum forest), then add edges with weight 0 between any two vertices that are not connected. Then G' is connected, so has a spanning tree. Find a maximum spanning tree T' in G'. Then removing the edges of T' with weight 0 gives a maximum forest F in G.

Conversely, given a graph G with weights w in which to find a maximum spanning tree, give it new weights w' as follows. Let W be the largest weight of any edge, and define $w'(e) = w(e) + W + 1$, so that all $w(e) > 0$. Find a maximum forest F in G' with w'. Because all weights are positive, F will be connected, so a maximum spanning for w'. Then it is also a minimum spanning tree for w, because $w(T) = w'(T) - (|V(G)| - 1) \cdot (W + 1)$. □

Linear Programming Formulation

The Minimum Spanning Tree Problem has a good LP formulation, but since we didn't really need it to find an algorithm, I will give it here without proof.

It is good in the sense that the polytope described by its inequalities is *integral*, i.e. the linear program and the integer program have the same optimum solutions (regardless of what linear function is to be optimized). In other words, the vertices of that polytope correspond exactly the to the spanning trees of G. That too I won't prove here, but it will follow from a more general theorem that we will very likely see later in the course (the matroid polytope theorem).

$$\begin{array}{ll} \text{minimize } \sum w(e)x_e & \text{subject to} \\ \sum_{e \in E(G[X])} x_e \leq |X| - 1 & \text{for } \emptyset \neq X \subsetneq V(G), \\ \sum_{e \in E(G)} x_e = |V(G)| - 1, & \text{and } 0 \leq x \leq 1. \end{array}$$

1.2 Breadth-First Search Trees

In this subsection the graphs are directed but unweighted.
An edge of a direct graph has a head vertex $h(e)$ (where the arrowhead is), and a tail vertex $t(e)$. I will always write a directed edge with its tail first, so if $e = uv$, then $t(e) = u$, $h(e) = v$.

Definitions
We define $\delta^{\text{in}}(v) = \{e : h(e) = v\}$ and $\delta^{\text{out}}(v) = \{e : t(e) = v\}$, and more generally $\delta^{\text{in}}(S) = \{e : h(e) \in S,\ t(e) \notin S\}$ and $\delta^{\text{in}}(S) = \{e : t(e) \in S,\ h(e) \notin S\}$ for $S \subset V(G)$.
A *directed path* is directed graph whose underlying undirected graph is a path, and $\delta^{\text{in}}(v) \leq 1$ and $\delta^{\text{out}}(v) \leq 1$ for all its vertices v.
A *directed cycle* is a directed path such that the underlying undirected graph is a cycle; equivalently, it is a connected directed graph such that $\delta^{\text{in}}(v) = \delta^{\text{out}}(v) = 1$ for all its vertices v.
A *directed tree with root* r is a directed graph T satisfying one of the following equivalent definitions:

- The underlying undirected graph is a tree, $|\delta^{\text{in}}(r)| = 0$, and $|\delta^{\text{in}}(v)| = 1$ for all $v \in V(T)\backslash\{r\}$.
- For each $v \in V(T)\backslash\{r\}$, there is a unique directed path from r to v.

A *directed forest* is a directed graph F such that the underlying undirected graph is a forest, and $|\delta^{\text{in}}(v)| \leq 1$ for all $v \in V(F)$.
These definitions can be summarized as follows: a *directed* X is an X in the underlying undirected graph with all edge oriented "in the same direction".

Breadth-First Algorithm
This is not an optimization algorithm, but it will be important for finding shortest paths.

BFS-Tree Algorithm (given root r)

1. Start with the graph T consisting just of r;
2. Set $S = \delta^{out}(T)$.
3. For each $e \in S$, add e to T if it does not create a cycle (in the undirected sense).
4. If $\delta^{\text{out}}(T) \neq \emptyset$, go back to 2.
5. If $|E(T)| = |V(G)| - 1$, return T; else return "disconnected".

Note that this is not the standard way of describing this algorithm (usually one would use a *queue*), but it will suit our purposes better.

Theorem 1.4. *Given a graph G and root r, the BFS-Tree Algorithm returns a spanning directed tree T rooted at r, if one exists.*
The unique path from r to a v is a shortest such path; in other words, $dist_T(r, v) = dist_G(r, v)$.

Proof. The algorithm clearly maintains a directed tree rooted at r, and if it returns T it must have $|E(T)| = |V(G)| - 1$, which implies that it is spanning.
A vertex v is added to T (along with an entering edge) at the kth loop of the algorithm if and only if $k = \text{dist}_T(r, v) = \text{dist}_G(r, v)$. Indeed, if there were a shorter path, v would have been added in an earlier loop. □

1.3 Directed Trees

In this subsection the graphs are directed and weighted.
We will write SDT for *spanning directed tree*.

Equivalent problems

Theorem 1.5. *Given a weighted undirected graph, the following* 6 *optimization problems are equivalent:*

- Minimum/Maximum SDT Problem: *Find a spanning directed tree with minimum/maximum weight.*
- Minimum/Maximum SDT Problem with Root r: *Find a spanning directed tree with root r and minimum/maximum weight.*
- Minimum/Maximum Directed Forest Problem: *Find a directed forest with minimum/maximum weight.*

Proof. The equivalence of the directed tree and directed forest problems is analogous to that for undirected trees and forests, and I will omit it here, as well as the equivalence of minimum/maximum versions. Assuming those, I will show that the rooted minimum SDT problem is equivalent to the others.
Given a graph G in which to find a minimum directed forest, add a new root r to it as follows:

$$V(G') = V(G) \cup \{r\}, \quad E(G') = E(G) \cup \{rv : v \in V(G)\}.$$

Then find a minimum SDT with root r in G'. Removing r will give a minimum directed forest.
Given a graph G in which to find a minimum SDT with given root r, add a new vertex as follows:

$$V(G') = V(G) \cup \{s\}, \quad E(G') = E(G) \cup \{sr\}.$$

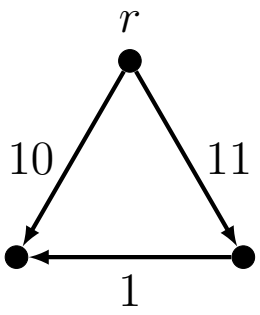
Find a minimum SDT in G'; it must be rooted at s. Removing s will give a minimum SDT with root r. □

Below we will deal only with the Minimum Spanning Directed Tree Problem with Root r. For convenience, we will assume that $\delta^{\text{in}}(r) = 0$, without loss of generality. Since edges entering r will never be used in a directed tree rooted at r, we can always just remove those.

Greedy doesn't work

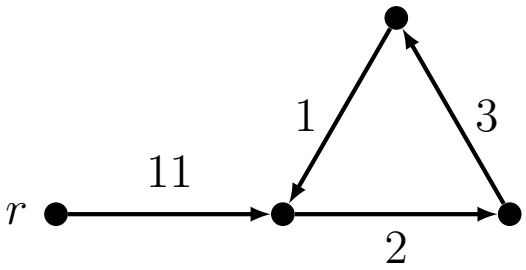
It turns out that for directed trees, the greedy approach doesn't work (it doesn't give a directed tree, or not a spanning one, or it doesn't give a minimum one).

For instance, try to grow a directed tree, by repeatedly adding a minimum edge that preserves connectedness, acyclicity (in the undirected sense), and the single-entry condition. That will fail, for instance for:



You would take the edge with weight 10, then the one with weight 11. That does give an SDT, but there is a smaller one, consisting of the edges with weights 11 and 1.

One could take a different greedy approach that would work for the example above, by growing a directed forest, so preserving only acyclicity and the single-entry condition. But that would fail for other graphs, like



You would end up with the edges of weight 1 and 2, which do not make up a directed tree. Yet there is a directed tree, with the edges of weight 11, 2, and 3.

Of course, it's hard to prove that no alternative greedy approach couldn't work. But below we will see a "almost-greedy" algorithm: it first takes edges greedily, preserving only the single-entry condition, then fixes that to get a directed tree.

In an earlier version there was an algorithm here for finding minimum directed trees that had a serious flaw. Something similar does work, but I don't want to fix this now, right before the exam.

So the greedy-like algorithm for finding minimum weight directed trees is not on the exam. But note that the matroid intersection algorithm can find a maximum cardinality directed tree.