18th Lecture: 3/3

Definition 18.1. A graph G is said to be a *tree* if it is connected and has no cycles. Trees are often denoted by the letter T. A graph which has no cycles but is not necessarily connected is called a *forest*. Forests are often denoted by the letter F. Note that every connected component of a forest is a tree.

Definition 18.2. A *rooted tree* is a pair (T, v), where T is a tree and $v \in V(T)$. The vertex v is called the *root*.

The word "tree", as used in ordinary language, is probably closer in meaning to Definition 18.2 than Definition 18.1. As Figure 18.1 shows, the same tree can "look" quite different, depending on which vertex is chosen as the root.

Definition 18.3. Let $T = (V, E)$ be a tree. A vertex $v \in V$ is called a *leaf* if $\deg(v) = 1$. A vertex v is called a *branching point* if $\deg(v) \geq 3$. If every vertex in T has degree at most 2, then T is called a *chain* or *path*. If T has at least one branching point then it is called a *branched tree*.

Let $f(n)$ denote the number of isomorphism classes of trees on n vertices. One may compute the following values:

n	1	2	3	4	5	6	7
$f(n)$	1	1	1	2	3	6	11

The various trees are exhibited in Figure 18.2. The sequence $(f(n))$ is number A000055 at `oeis.org`. There is no closed formula for $f(n)$. When attempting to list all trees on n nodes, up to isomorphism, a good strategy is to start with a vertex of maximum degree, and consider this as the root, as we have done in Figure 18.2.

The following theorem gives three alternative characterisations of trees. I did not go through the detailed proof in class, because all of it is fairly obvious, just a pain to write down all the details rigorously. I include the full proof here for the sake of completeness. However, this proof will *not* be examinable.

Theorem 18.4. *For a graph $G = (V, E)$, the following are equivalent:*
(i) G is a tree.
(ii) For each pair x, y of distinct nodes, there is a unique path in G between x and y.
(iii) Removing any edge from G results in a graph with two connected components, each of which is a tree.
(iv) G is connected and $|E| = |V| - 1$.

PROOF: (i) $\Leftrightarrow$ (ii): If there were two different paths between some pair x and y of vertices, then G would contain a cycle, part of the closed walk got by following one of the paths from x to y and the other from y back to x.

(ii) $\Leftrightarrow$ (iii): Suppose $G = (V, E)$ satisfies (ii), let $e = \{x, y\}$ be an edge and let G' be the graph remaining when e is removed. The edge e represents a path between x and y in G so, if we remove it, there can no longer be any path between these two vertices. Thus G' is disconnected. Let V_x (resp. V_y) be the subset of V consisting of those vertices which are reachable in G from x (resp. from y) by a path which does not use the edge e and let E_x (resp. E_y) be the set of edges in G between vertices of V_x (resp. V_y). The sets V_x and V_y must be disjoint, as otherwise G would contain a cycle formed by paths from this common vertex to x and y together with the edge e. The subgraphs (V_x, E_x) and (V_y, E_y) are, by definition, connected and, since they are subgraphs of G, they contain no cycles, hence are trees. We claim that these are the two connected components of G'. All that's left in order to prove this is to show that $V_x \cup V_y = V$. So let $v \in V$. Since G is connected, v is reachable by a simple path from each of x and y. Any such path which uses the edge e can do so only once. If it does so, then either it crosses from x to y or vice versa. In the first case, the remainder of the path is a simple path from y to v which doesn't use e at all, hence $v \in V_y$. In the second case, similarly, $v \in V_x$.

Conversely, if for some pair x and y, of vertices, there was more than one path between them, then any two such paths would form a closed walk which in turn must contain a cycle. Removal of any single edge along this cycle would not disconnect the graph.

(i) $\Rightarrow$ (iv): This is a special case of Theorem 16.x. Since it contains no cycles, a tree can always be drawn in the plane without any two edges crossing.

(iv) $\Rightarrow$ (i): We once again proceed by induction on $|V|$. If $|V| = 1$ and $|E| = |V| - 1 = 0$, then G is just a single vertex and is obviously a tree.

Suppose (iv) implies (i) for all graphs on $n \geq 1$ vertices, and let $G = (V, E)$ be a connected graph with $|V| = n + 1$ and $|E| = n$. The degree equation (15.7) implies that

$$\sum_{v \in V} \deg(v) = 2|E| = 2|V| - 2. \tag{18.1}$$

Now since G is connected, every vertex has degree at least one. But since the sum of the degrees is strictly less than $2|V|$, it follows that some vertex must have degree one. Let v be any such vertex, and let e be the unique edge incident to v. Let $G' = (V', E')$ be the n-vertex graph got by removing v and e. Since we've removed one vertex and one edge, this graph also satisfies $|E'| = |V'| - 1$. I claim it is also connected. For let v_1, v_2 be distinct vertices in V'. Since G is connected, there is a path in G between them. But this path cannot use the edge e, as it is the only edge going out to v and a path cannot use any edge more than once. Hence the path lies entirely in G', as required. Therefore, we can apply the induction hypothesis and conclude that G' is a tree, i.e.: that it has no cycles. But then neither does G, since the hanging edge e cannot be part of any cycle.

Corollary 18.5. *Let $T = (V, E)$ be a tree. Then T contains at least two leaves. If it contains exactly two leaves then it is a chain.*

PROOF: This follows from how the degree equation was applied in (18.1).

Definition 18.6. Let G be a connected graph. A subgraph H of G is called a *spanning tree for* G if H is a tree and includes every vertex of G.

Let $g(n)$ denote the number of spanning trees of the complete graph K_n. Another way of saying it is that $g(n)$ is the number of *labelled* trees on n vertices, that is, the number of ways to pick an n-vertex tree and label its vertices from 1 to n such that two labellings are considered the same if and only if the two trees are isomorphic and exactly the same pairs of labels are edges. Hence, the difference from the function $f(n)$ considered earlier is that now two trees on the same vertex set must have exactly the same edges in order to be considered "the same", it is not sufficient that they be isomorphic. In particular, $g(n)$ should grow much faster than $f(n)$. What is perhaps surprising at first sight is that, in sharp contrast to $f(n)$, there is a very simple formula[1] for $g(n)$:

Theorem 18.7. (Cayley's theorem)

$$g(n) = n^{n-2}. \tag{18.2}$$

PROOF: Omitted due to time constraints. The $4^{4-2} = 16$ different spanning trees of K_4 are exhibited in Figure 18.3.

The minimum spanning tree (MST) problem. As already noted in a footnote in Lecture 16, a *weighted graph* is a pair $(G,\, w)$, where G is a graph and $w : E(G) \to \mathbb{R}_+$ is a function from the edges of G to non-negative real numbers. For $e \in E(G)$, the quantity $w(e)$ is called the *weight* or *cost* of the edge e.

The *Minimum Spanning Tree (MST) Problem* asks for an algorithm to find, in a connected, weighted graph G, a spanning tree of minimum weight, where the weight of a subgraph is defined as the sum of the weights of its edges. The problem has many applications involving the cheapest way to connect up a bunch of hubs/nodes in a network.

This turns out to be a very simple problem and there are two standard solutions, Prim's algorithm and Kruskal's algorithm. Both involve the idea of a greedy search for minimal weights, but implement the idea differently.

Prim's algorithm. At the first step choose a vertex $v_1 \in V(G)$ arbitrarily, and choose an edge $e = \{v_1,\, v_2\}$ of minimum weight incident to v_1. If there are several edges of the same weight to choose from, choose one of them at random. Add v_2 to the set of *covered* vertices. At each subsequent step, choose an edge $e = \{v,\, w\}$ of minimum weight among all edges which have one endpoint v in the set of hitherto covered vertices and the other endpoint w among the vertices not yet covered. If there are several edges of equal weight to choose from, choose randomly. Add w to the set of covered vertices. Continue until all of $V(G)$ is covered. The set of chosen edges forms a MST.

[1]Since there are $n!$ permutations of an n-set and two trees are isomorphic if they contain the same edges up to a permutation of their labelled vertices, one would intuitively expect that $g(n)/f(n) \approx n!$ and hence, from (18.2) and Stirling's formula, that $f(n)$ grows roughly exponentially with n. I think this is the case, though I have not checked what the most precise estimates are to date for the growth of $f(n)$.

Kruskal's algorithm. At the first step choose an edge of minimum weight among all edges in G. If there are several edges of the same weight to choose from, choose one of them at random. At each subsequent step, choose an edge e of minimum weight among those which satisfy the following two properties:

(i) the edge e has not yet been chosen

(ii) adding e to the set of chosen edges does not create any cycles.

If there are several edges of equal weight to choose from, choose randomly. Continue until $|V(G)| - 1$ edges have been chosen. The set of chosen edges forms a MST.

The proofs that the two algorithms always yield MSTs are very similar. Here we present the proof for Prim's algorithm. That for Kruskal's algorithm is left as an (optional) exercise to the reader. See Demo5, Exercise 8 for a worked example for both algorithms.

Theorem 18.8. *Prim's algorithm always produces a minimum spanning tree.*

PROOF: Let $G = (V, E)$ be a connected, weighted graph on n vertices and let T be a spanning tree for G produced by Prim's algorithm. Let $e_1, e_2, \ldots, e_{n-1}$ be the sequence of edges chosen by the algorithm in order. Let U be any other spanning tree for G and let i be the smallest index such that e_i is not an edge in U. We will show that there is another spanning tree U^* such that $w(U^*) \leq w(U)$ and U^* contains each of the edges $e_1, \ldots, e_i$. Iterating this procedure at most $n - 1$ times will thus show that $w(T) \leq w(U)$ and hence that T is a minimum spanning tree, since U was chosen arbitrarily.

Now U contains each of the edges $e_1, \ldots, e_{i-1}$ by assumption. Let S be the set of vertices spanned by these edges. Let $e_i = \{x, y\}$, where $x \in S$ and $y \in V \backslash S$ - Prim's algorithm always chooses the next edge so that it covers a new vertex. Since U spans G, there must be *some* path in U from x to y. Call this path $\mathcal{P}_{xy}$. This path starts at a vertex in S and ends at a vertex in $V \backslash S$. Hence there must be a first edge $e : z \to w$ along the path such that $z \in S$ and $w \in V \backslash S$. Now the edge e was available to Prim's algorithm at the i:th step but was not chosen ahead of e_i. Since the algorithm always chooses an edge of minimal weight amongst those available, we must have $w(e) \geq w(e_i)$. Let $U^* := (U \cup \{e_i\}) \backslash \{e\}$. Thus $w(U^*) \leq w(U)$ and U^* contains all the edges $e_1, \ldots, e_i$. So all that remains to be shown is that U^* is a spanning tree for G. Since U^* has the same number of edges as the spanning tree U, it suffices to show that it spans G, for then it will follow immediately from Theorem 18.4(iv) that it contains no cycles.

So let $v_1, v_2 \in V$. Since U spans G, there is a unique path in U from v_1 to v_2. Call it $\mathcal{P}_{v_1 v_2}$. If this path doesn't use the edge e then it is still present in U^*. So suppose the path $\mathcal{P}_{v_1 v_2}$ *does* use the edge e. Let $\mathcal{C}_{xy}$ be the cycle formed by the path $\mathcal{P}_{xy}$ above and the edge e_i. Then we obtain a walk in U^* from v_1 to v_2 by

- following the path $\mathcal{P}_{v_1 v_2}$ until we hit the edge e,
- then replacing e by the rest of the cycle $\mathcal{C}_{xy}$, traversed in the appropriate direction,
- finally continuing to y along the path $\mathcal{P}_{v_1 v_2}$.

We've proven that U^* contains a walk, and hence a path, between any pair of vertices of G, hence it spans G, v.s.v.

Shortest path problem. Let $G = (V, E)$ be a weighted graph or digraph. In this setting, the weights are thought of as *lengths*. The *Shortest Path Problem* asks for an algorithm to find a shortest path between two given vertices $s, t \in V$, where the length of a path is the sum of the lengths of the edges along it. Note that we think of s as the "start" of the path and t as the "terminus". The standard solution to this problem is the following procedure:

Dijkstra's algorithm. Set $l(s) := 0$, $\mathcal{V} := \{s\}$ and $\mathcal{T} := \phi$. The function l is called a *labelling*. $\mathcal{V}$ will be a collection of labelled vertices, updated one vertex at a time until we reach t. $\mathcal{T}$ will be a tree, updated one edge at a time.

Choose an (out)edge $e = \{s, v_1\}$ from s of minimal weight and set $l(v_1) := w(e)$. Add v_1 to $\mathcal{V}$ and add e to $\mathcal{T}$.

At a general step, do the following: for each (directed) edge $e = \{v, w\}$ with startpoint v at a labelled vertex and endpoint w at a not-yet-labelled vertex, compute $l'(w) := l(v) + w(e)$. We call l' a *temporary labelling*. Compare all the temporary labels and choose the smallest one - if there are several equal values to choose from, choose randomly. Make the chosen temporary label $l'(w_0)$ permanent and add the vertex w_0 to $\mathcal{V}$. Add the corresponding edge to $\mathcal{T}$.

Continue until t is labelled. At this point, $\mathcal{T}$ will contain a unique path from s to t, which can be found by backtracking from t. This will be a shortest path in G from s to t.

To prove that Dijkstra's algorithm works is essentially trivial. There is a so-called *Breadth First Search* built into the algorithm which ensures that we always find a shortest path. In particular, it ensures that we never get into trouble by being "too greedy", for example by following a path of cheap edges for a while and suddenly getting stuck in a corner where only very expensive options are available. See Demo5, Excercise 8 for a worked example.