Seventeenth Lecture: 12/5

Definition 17.1. 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 17.2. 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 17.2 than Definition 17.1. As Figure 17.1 shows, the same tree can "look" quite different, depending on which vertex is chosen as the root.

Definition 17.3. 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 17.2. 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 17.2.

Theorem 17.4. *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 simple 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 simple paths between some pair x and y of vertices, then G would contain a cycle, got by following one of the paths from x to y and the other path back to x.

Conversely, suppose there were a cycle in G. This cycle may not be simple, i.e.: it may cross itself several times. However, any cycle must *contain* some simple cycle. So if G contains a cycle, then it contains a simple cycle. Now let x and y be any two vertices along a simple cycle. Then there are two distinct simple paths in G from x to y, obtained by following the cycle in opposite directions.

(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 simple path between them, then any two such paths would form a cycle which in turn must contain a simple cycle. Removal of any single edge along this simple cycle would not disconnect the graph.

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

Suppose (i) implies (iv) for all trees on $n \geq 1$ vertices and let G be a tree on $n + 1$ vertices. Since $n + 1 \geq 2$ and G is connected, it has at least one edge. Pick any edge $e = \{x, y\}$. As proven above, removing e will yield a graph with two connected components $G_x = (V_x, E_x)$ and $G_y = (V_y, E_y)$, each of which is a tree. By the induction hypothesis, $|E_x| = |V_x| - 1$ and $|E_y| = |V_y| - 1$. But $E = E_x \sqcup E_y \sqcup \{e\}$ so $|E| = |E_x| + |E_y| + 1 = (|V_x| + |V_y|) - 1 = |V| - 1$, v.s.v.

(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 (13.3) implies that

$$\sum_{v \in V} \deg(v) = 2|E| = 2|V| - 2. \tag{17.1}$$

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 simple 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 simple 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 simple cycle. □

Corollary 17.5. *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 (17.1). □

Definition 17.6. 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. 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. Hence, 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 17.7. (Cayley's theorem)

$$g(n) = n^{n-2}. \tag{17.2}$$

Proof. See Exercise 16.7.17 on the homepage. □

The $4^{4-2} = 16$ different spanning trees of K_4 are exhibited in Exercise 16.3.2.

The minimum spanning tree (MST) problem. As already noted in a footnote in Lecture 15, 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

[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 (17.2) 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 $f(n)$.

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.

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 and will be presented next day.

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. We will illustrate with examples next day.