Topics in Graph Theory — Lecture Notes I (Tuesday)

1. Basics: Graphs and Spanning Trees

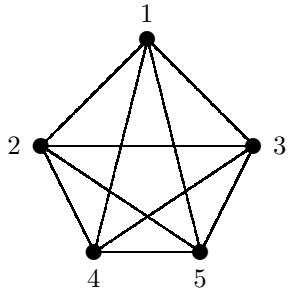
Notation: $G = (V, E)$ means that G is a graph with vertices V and edges E. Each edge e has either one or two vertices as *endpoints*; an edge with only one endpoint (equivalently, two equal endpoints) is called a *loop*. Two edges e, e' are allowed to have the same pairs of endpoints; such edges are called *parallel*. Unless otherwise specified, all graphs are *undirected*—if the endpoints of e are v and w, we will write $e = vw = wv$. If v and w are adjacent (that is, they share at least one edge), then we'll write $v \sim w$.

I'll assume that everyone has seen graphs before, and is either familiar with, or can easily figure out, what such elementary concepts as "vertex", "edge", "connected", etc. mean. However, if you don't know what something means, you should ask immediately—I promise I won't bite, and chances are that you are not the only one!

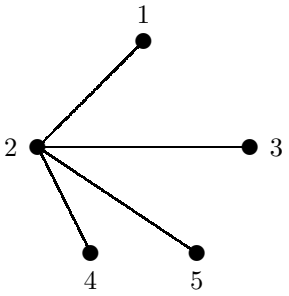
Unless otherwise specified, all graphs are *undirected*—that is, each edge is an *unordered* pair of vertices. Also, I have no problem allowing loops (edges whose two endpoints

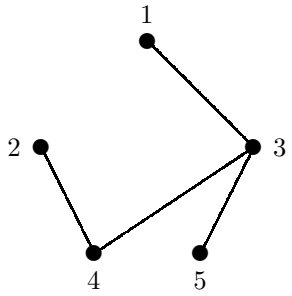
Definition: A **spanning tree** of a connected graph G is a connected, acyclic subgraph of G that spans every vertex.

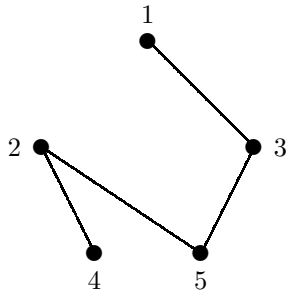
For example, the complete graph K_5 on 5 vertices, which looks like this...



has exactly 125 spanning trees (as we'll see soon), three of which are the following:







Let's introduce the notation

$$\begin{aligned} \mathcal{T}(G) &:= \{\text{spanning trees of } G\}, \\ \tau(G) &:= |\mathcal{T}(G)|. \end{aligned}$$

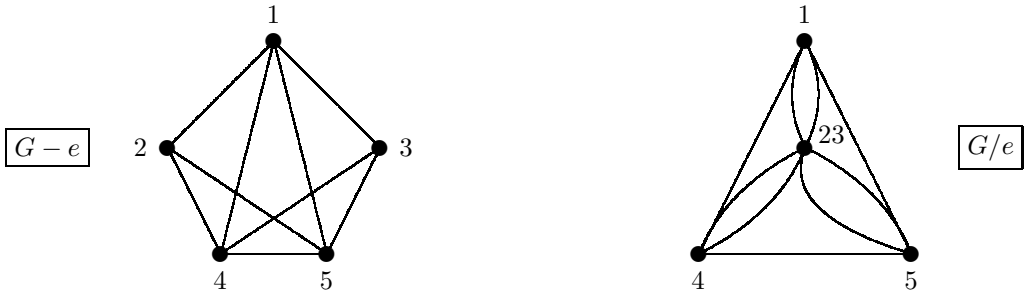
Note that every spanning tree of K_5 has four edges. More generally, every spanning tree of $G = (V, E)$ has precisely $|V| - 1$ edges. This gives us a stupid upper bound for $\tau(G)$, namely

$$\tau(G) \leq \binom{|E|}{|V| - 1}.$$

For example, $\tau(K_5) \leq \binom{10}{4} = 210$. But we can do better. As it turns out, there are two nice ways to calculate $\tau(G)$ exactly.

2. Calculating $\tau(G)$ by Deletion-Contraction

Let $G = (V, E)$ be a connected graph and $e = vw \in E$. The **deletion** $G - e$ is just the graph $(V, E - \{e\})$. The **contraction** G/e is obtained from $G - e$ by identifying v and w (or "fusing" the two vertices together). For contraction to make sense, we usually require that e not be a loop. For example, if $G = K_5$ and $e = 23$, then the deletion and contraction are as follows:



Proposition: Let $e \in E$. There are bijections

$$\{T \in \mathcal{T}(G) \mid e \notin T\} \qquad \longleftrightarrow \qquad \mathcal{T}(G - e)$$

and

$$\{T \in \mathcal{T}(G) \mid e \in T\} \qquad \longleftrightarrow \qquad \mathcal{T}(G/e).$$

That is, the spanning trees of G that **don't** contain e are in bijection with the spanning trees of the deletion $G-e$, and the spanning trees of G that **do** contain e are in bijection with the spanning trees of the contraction G/e.

It is pretty easy to check these bijections; I'll leave it as an exercise. Having done so, we have proven that

$$\tau(G) = \tau(G - e) + \tau(G/e)$$

for any graph G and edge e. This gives an obvious algorithm to compute $\tau(G)$ for any graph; unfortunately, it is computationally inefficient to do so—the runtime of the algorithm is $O(2^{|E(G)|})$. However, this type of deletion-contraction recurrence is theoretically very important; make a mental note of it!

3. The Kirchhoff Matrix-Tree Theorem

A less illuminating, but much more efficient way to calculate $\tau(G)$ uses linear algebra. Assume for the moment that G has no loops—the loops are precisely the edges that belong to no spanning tree of G, so deleting all the loops does not change the value of $\tau(G)$. Also, label the vertices of $V = V(G)$ as $\{v_1, \ldots, v_n\}$.

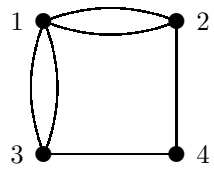
Define the **valence** of a vertex $v \in V(G)$, denoted $\mathrm{val}(v)$ or $\mathrm{val}_G(v)$, to be the number of edges having v as an endpoint.[1] Also, for two vertices $v_i \neq v_j \in V(G)$, let us write ϵ_{ij} for the number of edges joining i and j. So $\epsilon_{ij} \in \mathbb{N}$; we are not requiring that G be a simple graph, so $\epsilon_{ij} \geq 2$ is possible.

[1] This is usually called "degree", but I prefer "valence" because "degree" already has far too many meanings in mathematics. In chemistry, the valence of an atom in a molecule is (essentially) the number of bonds it forms, so it seems an appropriate term to use here.

We now define the **Laplacian matrix** of G to be the $n \times n$ matrix $L(G)$ given by

$$[L(G)]_{ij} \;=\; \begin{cases} \mathrm{val}_G(v_i) & \text{if } i = j, \\ -\epsilon_{ij} & \text{if } i \neq j. \end{cases}$$

Note that $\epsilon_{ij} = \epsilon_{ji}$, so $L(G)$ is a symmetric matrix. For example, if G is the 4-cycle with two adjacent edges doubled, that is,



then the Laplacian matrix is

$$L(G) \;=\; \begin{bmatrix} \mathrm{val}(1) & -\epsilon_{12} & -\epsilon_{13} & -\epsilon_{14} \\ -\epsilon_{21} & \mathrm{val}(2) & -\epsilon_{23} & -\epsilon_{24} \\ -\epsilon_{31} & -\epsilon_{32} & \mathrm{val}(3) & -\epsilon_{34} \\ -\epsilon_{41} & -\epsilon_{42} & -\epsilon_{43} & \mathrm{val}(4) \end{bmatrix} \;=\; \begin{bmatrix} 4 & -2 & -2 & 0 \\ -2 & 3 & 0 & -1 \\ -2 & 0 & 3 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix}.$$

Notice that each row and column of $L(G)$ sums to zero, so it is certainly true that $\det L(G) = 0$. However, something really neat happens with maximal square submatrixes:

Kirchhoff's Matrix-Tree Theorem: Let $v_i \in V(G)$, and let $\hat{L}(G)$ be the matrix obtained from $L(G)$ by deleting the row and column corresponding to v_i. Then

$$\det \hat{L}(G) \;=\; \tau(G).$$

Impressive, isn't it? There are several different ways to prove it; the way that I like the best is by deletion-contraction (because I can understand it). Of course, you need some linear algebra machinery—See the exercises.

There's actually a fancy version of the Matrix-Tree Theorem (at least for simple graphs; I haven't thought about how to extend it to arbitrary graphs but it ought to be possible) which we'll need later.

Souped-Up Matrix-Tree Theorem: Let G be a simple graph with vertices $V = \{v_1, \ldots, v_n\}$. Introduce an indeterminate ϵ_{ij} for each edge $e = v_i v_j$, and define the $n \times n$ **weighted Laplacian matrix** of G, denoted $\mathbf{L}(G)$, by

$$[\mathbf{L}(G)]_{ij} \;=\; \begin{cases} \sum_{v_j \sim v_i} \epsilon_{ij} & \text{if } i = j, \\ -\epsilon_{ij} & \text{if } i \sim j, \\ 0 & \text{otherwise.} \end{cases}$$

Choose a vertex v_i, and let $\hat{\mathbf{L}}(G)$ be the submatrix obtained by deleting the row and column corresponding to v_i. Then

$$\det \hat{\mathbf{L}}(G) \;=\; \sum_{T \in \mathcal{T}(G)} \left(\prod_{e = v_i v_j \in T} \epsilon_{ij} \right).$$

That is, the determinant, which is clearly a polynomial in the variables ϵ_{ij}, is in fact a sum of monomials corresponding to the spanning trees of G. Note that setting $\epsilon_{ij} = 1$ recovers the original Matrix-Tree Theorem.

The Souped-Up Matrix-Tree Theorem is particularly useful when we replace the indeterminates ϵ_{ij} with "natural" weights on the edges—for instance, we might set $\epsilon_{ij} = x_i x_j$ to keep track of vertex valences.

4. Spanning Trees of K_n

Using the Matrix-Tree Theorem, one can prove the following remarkable formula (typically attributed to Cayley):

$$\tau(K_n) \;=\; n^{n-2}. \tag{1}$$

Using the Souped-Up Matrix-Tree Theorem, one can prove the following more general result. Define the **weight** of a spanning tree $T \subset E(K_n)$ to be the monomial

$$\mathrm{wt}(T) \;=\; \prod_{i=1}^{n} x_i^{\mathrm{val}_T(i)}.$$

Then

$$\sum_{T \in \mathcal{T}(K_n)} \mathrm{wt}(T) \;=\; x_1 x_2 \cdots x_n (x_1 + x_2 + \cdots + x_n)^{n-2}. \tag{2}$$

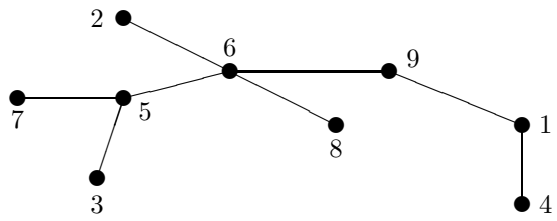
Note that setting all of the x_i's to 1 recovers (1), which is why this result is more general.

There is a beautiful combinatorial proof of (2) called **Prüfer coding**. (There are other proof known, but this is the most popular!) The idea is to define a bijection

$$P : \mathcal{T}(K_n) \to [n]^{n-2}$$

by iteratively "pruning leaves" from a spanning tree T. Here $[n]$ means $\{1, 2, \ldots, n\}$ (this is very standard notation in combinatorics), so $[n]^{n-2}$ denotes the set of $(n-2)$-tuples of members of $[n]$.

In general, a **leaf** of a graph is a vertex of valence 1. A tree with at least two vertices has at least two leaves (prove this!), so it makes sense to speak of the smallest leaf of a tree (with respect to the labeling of the vertices of K_n by the elements of $[n]$). For example, consider the following spanning tree of K_8:

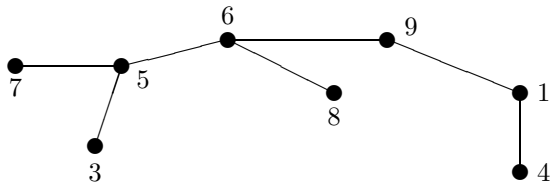


The leaves are 2,3,4,7,8, so the smallest leaf is 2. To obtain the Prüfer code $P(T)$, we run the following algorithm:

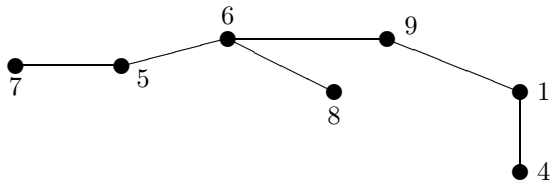
(1) Let $i = 1$.
(2) Let x be the smallest leaf of T.
(3) Let v_i be the "stem" (that is, the unique neighbor) of x.
(4) Replace T with $T - v$ (that is, delete vertex v and edge vw).
(5) If $i = n - 2$, then STOP; otherwise, replace i with $i + 1$ and go to step 2.

The Prüfer code $P(T)$ is then the sequence $(v_1, \ldots, v_{n-2}) \in [n]^{n-2}$.

For the tree T shown above, we begin by deleting 2, writing down the stem $v_1 = 6$, and replacing T with the tree $T - 2$:



Now the leaves are 3,4,7,9, so the smallest leaf is 3. We write down the stem $v_2 = 5$ and delete vertex 3, obtaining the tree



And so on. Ultimately, we obtain the Prüfer code

$$P(T) \;=\; (6,5,1,9,5,6,6) \in [9]^7.$$

That the function P is a bijection is left an exercise to the reader (mwahahaha). It's not that hard; you essentially have to figure out how to run the algorithm in reverse.

Now here's something interesting. For each vertex v_i, Compare the valence $\mathrm{val}_T(v_i)$ with the number of times that v_i appears in the Prüfer code $P(T)$:

Vertex	1	2	3	4	5	6	7	8	9
Valence in T	2	1	1	1	3	4	1	1	2
# occurrences in $P(T)$	1	0	0	0	2	3	0	0	1

It certainly looks like

the number of occurrences of v_i in $P(T)$ is $\mathrm{val}_T(v_i) - 1$.

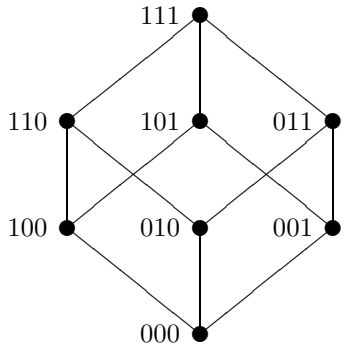
This makes perfect sense—if v_i starts life as a leaf, then v_i can never be the stem of any other vertex and hence cannot appear at all in $P(T)$. On the other hand, if v_i has more than one neighbor, then it does not become a leaf until exactly $\mathrm{val}_T(v_i) - 1$ of the neighbors of v_i have been killed off, and each such off-killing accounts for one instance of v_i in $P(T)$.

This observation leads to a purely combinatorial (as opposed to linear-algebraic) proof of the weighted version (2) of Cayley's formula:

$$\begin{aligned}
\sum_{T \in \mathcal{T}(K_n)} \mathrm{wt}(T) &= \sum_{P=(v_1,\ldots,v_n)\in[n]^{n-2}} \prod_{i=1}^{n} x_i^{1+(\text{number of occurrences of } i \text{ in } P)} \\
&= x_1 \cdots x_n \sum_{P=(v_1,\ldots,v_n)\in[n]^{n-2}} \prod_{i=1}^{n} x_i^{\text{number of occurrences of } i \text{ in } P)} \\
&= x_1 \cdots x_n (x_1 + \cdots + x_n)^{n-2}.
\end{aligned}$$

5. Spanning Trees of Q_n

The **n-cube** Q_n is the graph whose vertices are the bit strings of length n, with two vertices joined by an edge if and only if they differ in exactly one bit. Thus $|V(Q_n)| = 2^n$ and $|E(Q_n)| = n \cdot 2^{n-1}$ (check this!) For instance, the graph Q_3 is as follows:



A side note: A really cool way to draw the graph Q_4 is as follows. First, draw a 7×7 chessboard. Put a dot in the middle square of the top and bottom rows. Now, draw in all the ways to connect the two squares by a sequence of four knight's moves. Amazing.

Using the Matrix-Tree Theorem, one can prove the following surprising formula:

$$\tau(Q_n) \;=\; \prod_{\substack{S \subset [n] \\ |S| \geq 2}} 2|S| \;=\; \prod_{k=2}^{n} (2k)^{\binom{n}{k}}. \tag{3}$$

Again, there is a weighted version of this fact, which appears in a paper I coauthored [1]. Define the **direction** of an edge $e \in E(Q_n)$ to be the unique bit in which its endpoints differ; this is a number $\mathrm{dir}(e) \in [n]$.

Theorem: Introduce indeterminates $q_1, \ldots, q_n$. For a spanning tree $T \in \mathcal{T}(Q_n)$, define

$$\mathrm{wt}(T) \;=\; \prod_{e \in T} q_{\mathrm{dir}(e)} \;=\; \prod_{i=1}^{n} q_i^{\text{number of edges of direction } i \text{ in } T}.$$

Then

$$\sum_{T \in \mathcal{T}(Q_n)} \mathrm{wt}(T) \;=\; q_1 \ldots q_n \prod_{\substack{S \subset [n] \\ |S| \geq 2}} \left(2 \sum_{i \in S} q_i \right) \;=\; \sum_{T \in \mathcal{T}(Q_n)} \mathrm{wt}(T) \;=\; 2^{2^n - n - 1} q_1 \ldots q_n \prod_{\substack{S \subset [n] \\ |S| \geq 2}} \left(\sum_{i \in S} q_i \right). \tag{4}$$

The factor $q_1 \ldots q_n$ can be explained pretty easily: as pointed out in class, every $T \in \mathcal{T}(Q_n)$ must use at least one edge from each of the n possible directions. (For instance, if $T \in \mathcal{T}(Q_3)$ contains no edge in direction 2, then there is no way to get from 001 to 011 using only edges of T.)

Setting all of the q_i's to 1 specializes (4) to (3). As we shall soon see, even (4) can be strengthened. However, according to the "Bible of Combinatorics" [2], no bijective proof is known. Your job is to find a proof!

References

[1] Jeremy L. Martin and Victor Reiner, Factorizations of some weighted spanning tree enumerators. *J. Comb. Theory Ser. A* **104**, no. 2 (2003), pp. 287–300.

[2] R.P. Stanley, *Enumerative Combinatorics, Volume II.* Cambridge Studies in Advanced Mathematics **62**. Cambridge University, 1999.