Trees: Math 454 Lecture 12 (7/18/2017)

Cole Franks

July 19, 2017

Some of these notes follow sections 5.1 and 5.6 of Keller and Trotter, which you can find here:

http://www.rellek.net/book/ch_graphs.html.

More is in Brualdi 11.5.

Contents

1 Spanning Trees **1**
1.1 Finding a spanning tree 1

2 Leaves **4**

3 Prüfer codes **4**

4 Counting spanning trees: Cayley **8**

5 Counting ordered trees: **9**
5.1 Counting Dyck words via a recurrence 12
5.2 Counting Dyck words awesomely 13

1 Spanning Trees

Definition 1.1 (Spanning tree)**.** A spanning tree is a tree $T = (V', E')$ contained in $G = (V, E)$ with $V' = V$. In other words, the tree uses all the vertices of G.

1.1 Finding a spanning tree

A graph is connected if and only if it has a spanning tree.

Theorem 1.1 (baby Kruskal's algorithm)**.** *Let $G = (V, E)$ be a connected graph. Let $S_0 = \emptyset$.*
Set $i = 1$. While it is possible:

1. *Find an edge e of G such that $S_{i-1} \cup e$ has no cycles.*
2. *Let $S_i = S_{i-1} \cup \{e\}$ (that is, add e to S_{i-1}).*
3. *Increase i by 1.*

Do this for as long as possible. No matter how this is performed, you will be able to proceed until exactly $i = n - 1$, and $T = (V, S_{n-1})$ will be a spanning tree of G.

The proof is slightly different from what we did in class.

Proof. Suppose we can't perform the i^{th} step for $i \geq 0$. If not, all edges e of $E \setminus S_{i-1}$ make a cycle when added to the edges of S_{i-1}. That is, $S_{i-1} \cup \{e\}$ contains a cycle. This means that the endpoints of each remaining edge of G are in the same connected component of (V, S_{i-1}). This means adding the remaining edges of G doesn't increase the number of connected components. However, G is connected, so there must be only one component of (V, S_{i-1}). In other words, we can't perform step i exactly when (V, S_{i-1}) is connected. This means (V, S_{i-1}) is a spanning tree. We know $i = n$ because a tree on n vertices has $n - 1$ edges. □

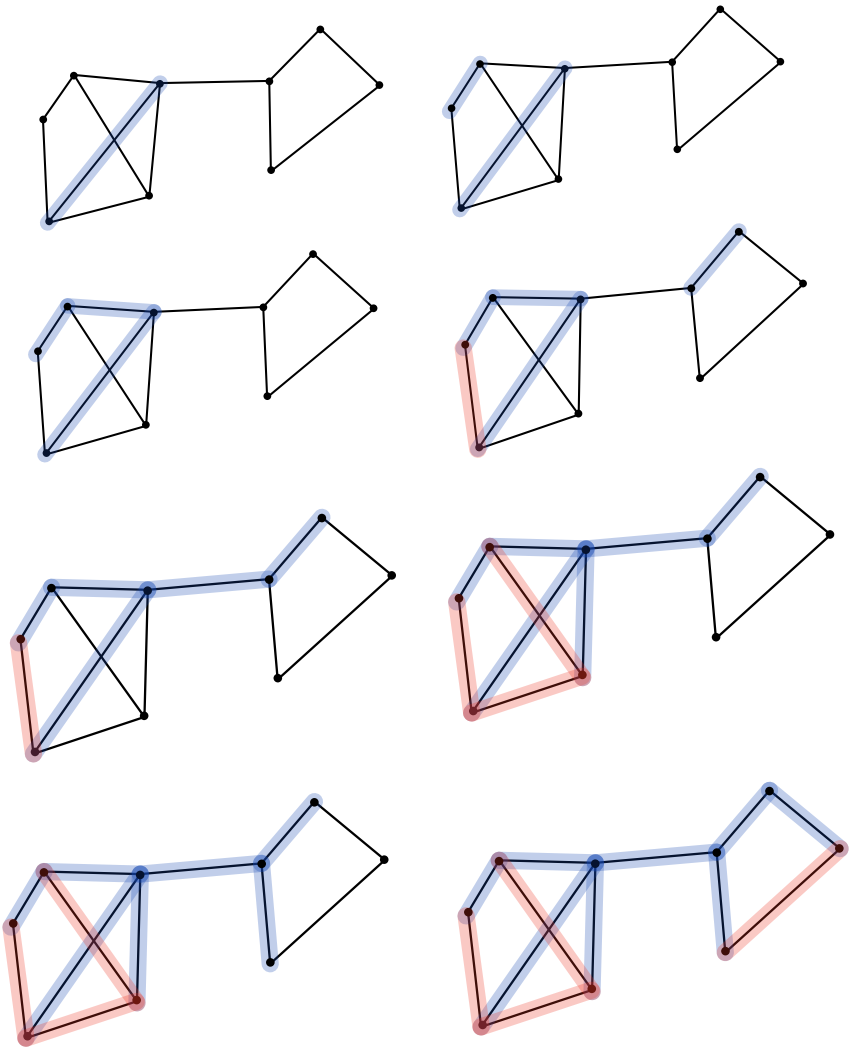
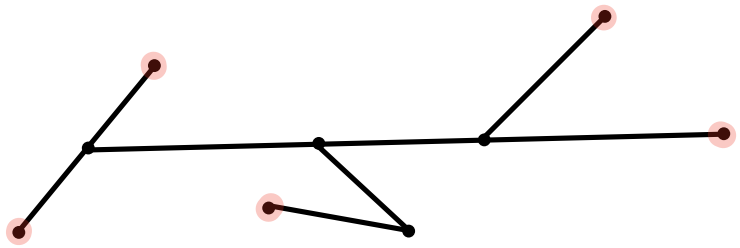


Figure 1: The edges in blue are a spanning tree made by Kruskal's algorithm. In red are edges that cannot be added in the next step.

2 Leaves

Definition 2.1 (Leaf)**.** If T is a tree, a *leaf* of T is a degree 1 vertex of T.

Example 1. The leaves of the following tree are labeled in red.



There's a stronger version of the following theorem on the homework.

Theorem 2.1. *Every tree has at least one leaf.*

Proof. If not, every vertex has degree at least 2, so by the First Theorem of Graph Theory there are at least n edges. We already know a tree has $n-1$ edges, so this is a contradiction. □

3 Prüfer codes

Example 2. Count labeled trees of size 4.

There is a one-to-one correspondence between trees on vertex set V and strings of length $n-2$ with elements of V as entries. The sequence corresponding to T is called a *Prüfer code*, and will be denoted prüfer (T).

We'll need to assume V is in order, so we just assume V is a set of n positive integers. We define prüfer (T) recursively:

Definition 3.1. Let ε denote the empty string, and if a and b are strings let $a \cdot b$ denote their concatenation. Suppose $T = (V, E)$. Define

$$\text{prüfer}\,(T) := \begin{cases} \varepsilon : & |V| = 2 \\ u \cdot \text{prüfer}\,(T - v) : & \text{v is the smallest leaf of } V \text{ and } uv \in E \end{cases}$$

$T - v$ denotes the graph $T - v = (V \setminus \{v\}, E \setminus \{e : v \in e\})$, or the tree obtained by removing v.

Remark 3.1. prüfer (T) is a well-defined function because v is attached to exactly one vertex u, so there is only way to do this procedure. See Figure 2 for an example.

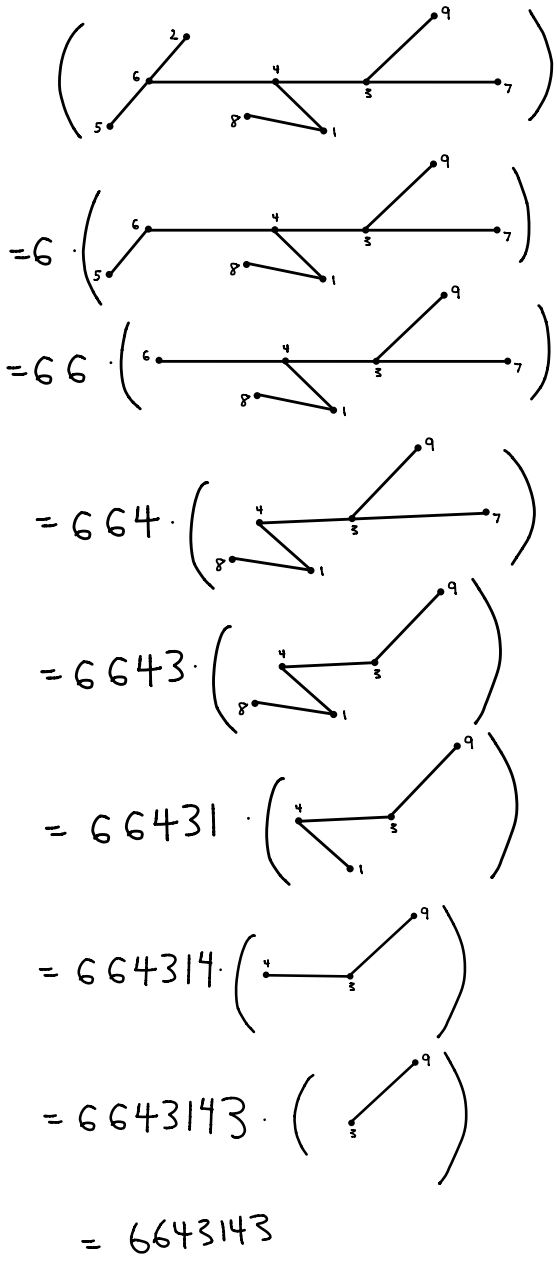


Figure 2: The labeled tree at the top has Prüfer code 6643143.

Theorem 3.1. *Let S be a set of n positive integers.*

$$f : \{\textit{labeled trees with vertex set } S\} \to \{ \textit{ strings of } n-2 \textit{ numbers from } S\}$$

given by $f(T) = \text{prüfer}\,(T)$ is one-to-one and onto.

Proof. We already remarked that $T \to \text{prüfer}\,(T)$ is a function. It remains to show that each string of $n-2$ numbers from S is $\text{prüfer}\,(T)$ for some tree T with vertex set $[n]$.

First, notice that the numbers that don't show up among $\text{prüfer}\,(T)$ are exactly the labels of the leaves of T. This is because no leaf has another leaf connected to it, and eventually every non-leaf vertex will have its neighbor removed which means it will show up in $\text{prüfer}\,(T)$.

With this in hand, we do the rest by induction. Our base case is $n = 2$. In this case, the only such string is the empty string, and it comes only from the tree $T = (\{1, 2\}, \{\{1, 2\}\})$.

If $n \geq 3$, suppose $s_1 s_2 \dots s_n$ is a string of $n-2$ numbers from the set S of n positive integers. Let k be the least number in S not among $\{s_1, s_2, \dots, s_n\}$. By induction, there is exactly one tree T' with vertex set $S \setminus \{k\}$ with $\text{prüfer}\,(T') = s_2, \dots, s_n$. Form T by adding k to T' as a leaf attached to s_1. Then $\text{prüfer}\,(T) = s_1 s_2 \dots s_n$.

T is the only tree we can make with Prüfer code $s_1 s_2 \dots s_n$, because in any tree T'' with Prüfer code $s_1 s_2 \dots s_n$, there had to be a leaf attached to s_1 of lowest label. Since the labels of leaves of T'' are exactly the numbers not among $s_1 s_2 \dots s_n$, said leaf must have label k (the least number among those numbers not among $s_1 s_2 \dots s_n$, and so T'' must be formed by adding k to T' as a leaf attached to s_1. □

We can use the idea in this proof to make a tree from a Prüfer code.

Example 3 (Making tree from Prüfer code)**.** Let's turn the Prüfer code 75531 into a tree on vertex set [7].

Prüfer code	vertex set	edge added
75531	$\{1, 2, 3, 4, 5, 6, 7\}$	$\{2, 7\}$
5531	$\{1, 3, 4, 5, 6, 7\}$	$\{4, 5\}$
531	$\{1, 3, 5, 6, 7\}$	$\{6, 5\}$
31	$\{1, 3, 5, 7\}$	$\{5, 3\}$
1	$\{1, 3, 7\}$	$\{3, 1\}$
ε	$\{1, 7\}$	$\{1, 7\}$

You can also see this pictorially in Figure 3.

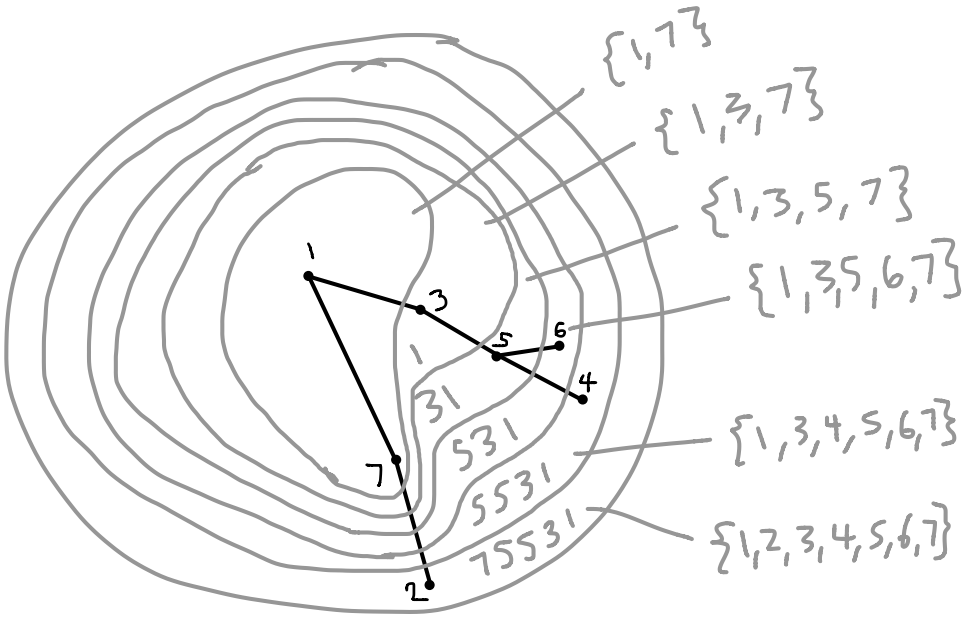


Figure 3: The Prüfer code inside each ring is the code of the tree inside the ring. Each ring is attached to the vertex set of the tree inside by a line. To construct the tree, we work from the outside in.

.

4 Counting spanning trees: Cayley

We immediately get the following theorem:

Theorem 4.1 (Cayley's Formula)**.** *The number of labeled trees on vertex set* $[n]$ *is*

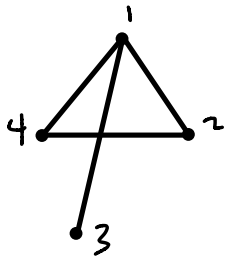
$$n^{n-2}.$$

This is an example of a more general method for computing the number of spanning trees, which we will state but not prove. First we need a definition:

Definition 4.1. Let G be a graph with vertex set $[n]$. If $A(G)$ is the adjacency matrix of G, and $\mathrm{diag}(d(1), \ldots d(v))$ is a diagonal matrix with the degree of vertex i in the i, i entry, then the *Laplacian* of G the matrix

$$L(G) = \mathrm{diag}(d(1), \ldots d(v)) - A(G).$$

Example 4. The adjacency matrix of the graph G given by



is

$$A(G) = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix},$$

and the diagonal matrix with degrees on the diagonal is

$$\begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix},$$

so the Laplacian is

$$L(G) = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 2 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ -1 & -1 & 0 & 2 \end{bmatrix}.$$

Example 5 (Laplacian of K_n)**.** The complete graph has adjacency matrix all ones except zeroes on the diagonal. The degrees are all $n-1$, so the Laplacian $L(K_n)$ will be the matrix with $(n-1)$'s on the diagonal and -1's everywhere else.

Theorem 4.2. *(Kirchoff's Matrix Tree Theorem) Let G be a connected graph. Form the $n \times n$ matrix L' by deleting any row and column from the Laplacian $L(G)$ of G. Then the number of labeled spanning trees of G is* $\det L'$.

Example 6. Let's find the number of spanning trees of the graph from the previous example. The Laplacian is

$$L(G) = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 2 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ -1 & -1 & 0 & 2 \end{bmatrix}.$$

I choose to calculate the spanning trees by deleting the first row and column, so the number of spanning trees is

$$\det \begin{bmatrix} 2 & 0 & -1 \\ 0 & 1 & 0 \\ -1 & 0 & 2 \end{bmatrix} = \det \begin{bmatrix} 2 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 3/2 \end{bmatrix} = 2 \cdot 1 \cdot (3/2) = 3.$$

Remark 4.1. Cayley's formula is a special case of the matrix tree theorem because the number of labeled spanning trees of K_n is exactly the number of labeled trees with vertex set n. If we take delete a row and column from $L(K_n)$, we get an $(n-1) \times (n-1)$ matrix L' with $n-1$'s on the diagonal and -1's everywhere else. You can check that the eigenspace of L' corresponding to the eigenvalue n has dimension $(n-2)$ and the eigenspace of L' corresponding to 1 has dimension 1. Thus the determinant of L' is n^{n-2}.

5 Counting ordered trees:

Definition 5.1. A *rooted tree* is one where one special vertex r is designated to be the root. Pictorially this is usually represented by putting the root at the top (for some reason). Once we do this, we can think of edges going towards or away from the root. We say a vertex w is below v if there is path from r to v through w. An *child* w of a vertex v is a vertex directly below v (that is, adjacent to v and below w), and v is $w's$ unique *parent*. See Figure 4.

Definition 5.2. A *ordered tree* is a rooted tree with an ordering in which the children of any parent are ordered from left to right, and all vertices below a child are to the left of the vertices below a child that is further to the right. See Figure 5 for examples.

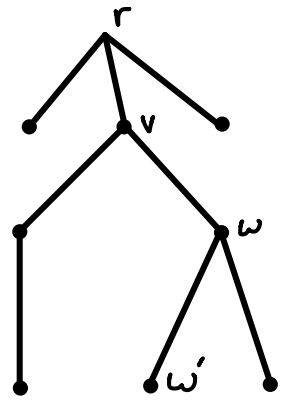


Figure 4: A rooted tree, in which w' is below w and v, w' is a child of w and w is a child of v.

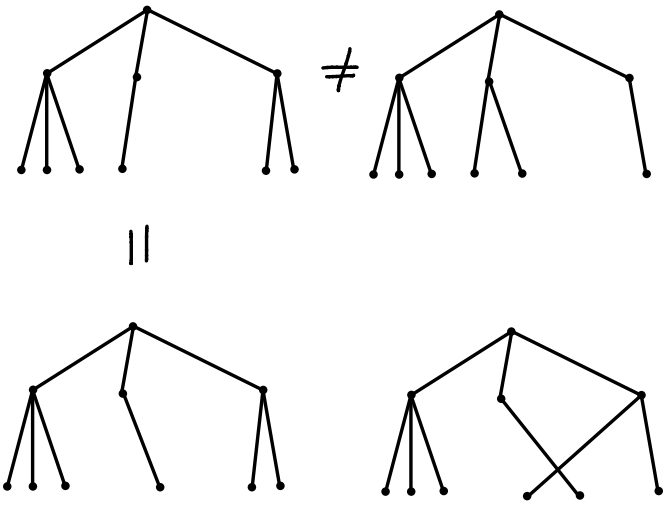


Figure 5: The top two trees are not equal *as ordered trees*. Top left and bottom left are equal as ordered trees. Bottom right is no ordered tree; it's a space station.[2]

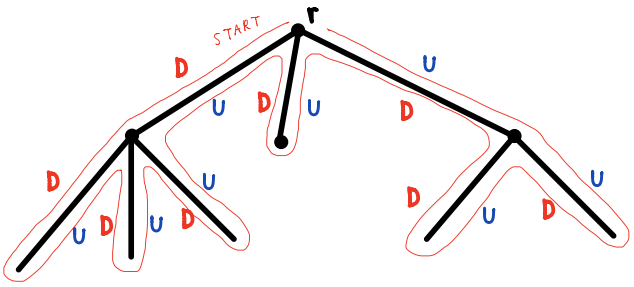


Figure 6: An ordered tree whose traversal is the Dyck word $DDUDUDUUDUDDUDUU$.

Definition 5.3. For this class, a *traversal* of a rooted, ordered tree is a walk along all the edges of the tree that visits vertices in order and uses each edge exactly twice. You can think about edges as having a left and a right side, and traversals as walks that start at the root and use the left and right side of each edge exactly once. A traversal can be written down by noting when you went up and when you went down, say by $U's$ and $D's$. A tree has exactly one transversal.

Remark 5.1. In computer science literature, traversals usually only care about the order you visit the vertices rather than which edges you use, but I don't now what else to call it.

Given a valid up-down sequence of length $2n$, we can make exactly one ordered tree with n edges with that sequence for its traversal! Here's how it's done:

Algorithm 1. Let $w = x_1x_2x_3 \dots x_{2n}$ be an up-down sequence (a sequence of U's and D's). Form a tree as follows:

1. Draw the root r. Sit on r.
2. Read w from left-to-right. If you read a D, add a child to the vertex you are sitting on to the right of all its other children and go sit on that vertex. If you read a U, go to the parent of the current vertex and sit on that vertex.

Remark 5.2. What conditions do we need for this algorithm to work? We need that whenever we read a U, there was a parent to go to. This means that for ANY prefix of the sequence, there at least as many D's as there are U's. We also need that we made it back to the root, so in the entire sequence there should be the same number of U's and D''s.

Definition 5.4. A sequence of n U's and n D's (really doesn't have to be U and D, could be anything) such that every prefix has at least as many D's as it has U's is called a *Dyck word* (pronounced *deek*).

[2]This link does not contain course material. https://youtu.be/JGp_5gOww0E

The following theorem summarizes what we have seen:

Theorem 5.1. *There is a one-to-one correspondence between Dyck words of length $2n$ and ordered trees with n edges.*

From here on out I am actually going to consider the roles of U and D switched, just because I think it makes the pictures nicer. From now on every prefix of a Dyck word has more U's than D's.

Now let's count the number of Dyck words of length $2n$. There are two ways:

5.1 Counting Dyck words via a recurrence

Let C_n be the number of Dyck words of length $2n$. Think of $C_0 = 1$ as usual. Let w be a Dyck word. Think of the graph of w as in Figure 7. Let $2 \leq 2k \leq 2n$ be the *first* point at which w's graph hits the x axis (we are justified in calling it $2k$ because we know it is even). The number of words that first hit the horizon at k is $C_{k-1}C_{n-k}$, because the possible words between 1 and $2k-1$ are the Dyck words of length $2k-2$ and the possible words between $2k$ and $2n$ are the Dyck words of length $2n-2k$. Further, for different k, these sets of words are disjoint. This means that

$$C_n = \sum_{k=1}^{n} C_{k-1}C_{n-k}, \text{ for } n \geq 1$$

or

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-k-1}, \text{ for } n \geq 1$$

or

$$C_{n+1} = \sum_{k=0}^{n} C_k C_{n-k}, \text{ for } n \geq 0.$$

5.2 Counting Dyck words awesomely

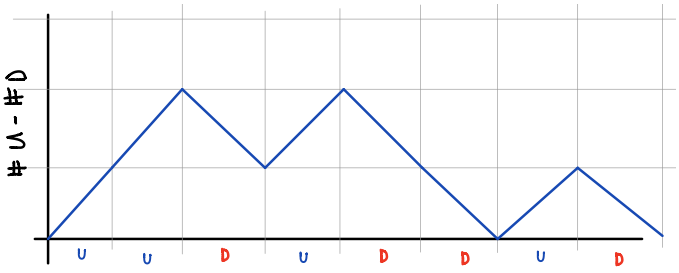


Figure 7: The graph of a Dyck word. One can view Dyck words as mountain ranges that never dip below the horizon.