Lecture 3: Common graph families, trees, and Cayley's theorem · 1MA020

Vilhelm Agdur[1]

[1] vilhelm.agdur@math.uu.se

24 October 2023

We start by introducing a few named families of graphs. Then we introduce the class of *trees*, and prove some results about them. The main result is Cayley's theorem, which counts the number of labelled trees on n vertices.

Common graph families

We warm up today by giving names to some common families of simple graphs that we will see reappearing throughout the course. They are illustrated in Figure 1.

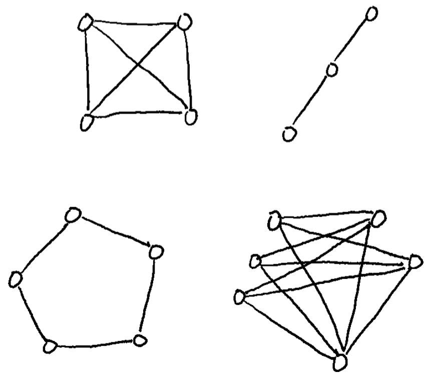
Figure 1: Four graphs: K_4, P_2, C_5, and $K_{3,2,1}$.

1. The complete graphs on n vertices, denoted K_n. These contain all the $\binom{n}{2}$ potential edges. These are also called *cliques* when we see them as subgraphs of a bigger graph.
2. The paths of length ℓ, denoted P_ℓ. If we take the set $\{0, 1, \ldots, \ell\}$ to be our vertex set, the edges are precisely of the form $\{i-1, i\}$ for $i \in [\ell]$.[2]
3. The cycle graphs on n vertices, denoted C_n. We can think of these as a path of length $n-1$ with an extra edge joining the first and last vertex.
4. The complete bipartite graphs on $a+b$ vertices, denoted $K_{a,b}$. These have as vertex set the disjoint union of two sets L and R,[3] with $|L| = a$ and $|R| = b$, and there is an edge between v and w whenever $v \in L$ and $w \in R$. When we see these graphs as subgraphs of a bigger graph, we sometimes also call them *bicliques*.

[2] This is another notation you might not have seen before: For an integer n, we write $[n]$ for the set $\{1, 2, \ldots, n\}$

[3] Think of these as the "left" and "right" vertices.

5. Generalizing the complete bipartite graphs, the complete multipartite graph on r parts with sizes $a_1, a_2, \dots a_r$, denoted $K_{a_1,a_2,\dots,a_r}$, has as its vertex set the disjoint union of r sets $V_1, V_2, \dots, V_r$, where $|V_i| = a_i$, and there is an edge between two vertices whenever they are not in the same part. We can notice that when $r = 2$ this is a complete bipartite graph, and when all the parts are of size 1 this is a complete graph.

For most of these, it is obvious how many edges they will have. Let us state a lemma that shows how many the complete multipartite graphs have.

Lemma 1. *The complete multipartite graph* $K_{a_1,a_2,\dots,a_r}$ *has* $\frac{1}{2}\left(n^2 - a_1^2 - \dots - a_r^2\right)$ *edges.*

Proof. We use the handshake lemma from the previous lecture. Since a vertex in V_i has one edge to every vertex not in V_i, it has degree $n - a_i$, and there are a_i such vertices. Thus we can compute that

$$\begin{aligned} 2|E| &= \sum_{v \in V} d_v = \sum_{i=1}^{r} \sum_{v \in V_i} d_v \\ &= \sum_{i=1}^{r} \sum_{v \in V_i} (n - a_i) = \sum_{i=1}^{r} a_i(n - a_i) \\ &= n \left(\sum_{i=1}^{r} a_i \right) - \sum_{i=1}^{r} a_i^2 = n^2 - \sum_{i=1}^{r} a_i^2 \end{aligned}$$

proving the claim. □

Corollary 2. *The complete bipartite graph* $K_{a,b}$ *has* $\frac{1}{2}\left(n^2 - a^2 - b^2\right) = ab$ *edges.*

Trees

The main topic of this lecture is the so-called *trees*. Informally, they are just graphs that look like trees – though with this intuition we are drawing them upside-down.[4]

[4] Unless you are a computer scientist, in which case you draw them the right way up, I believe.

Definition 3. A *tree* is a graph $T = (V, E)$ that is both connected and contains no cycles.

Just like for graphs in general, there are countless variants of the notion of a tree. We can give the tree a root,[5] we can consider orderings of the vertices in various ways, and so on. None of this will actually be necessary in the course, however, so we skip defining these notions.

[5] Which is quite necessary for a biological tree.

Example 4. Trees appear in many places in various areas – one common place for them to appear is in the study of algorithms. Let's study the quicksort algorithm, and see how it can be represented as a tree. The algorithm sorts a list of numbers in ascending order, and it works as follows:

1. Fix an arbitrary pivot p element from the list.
2. Compare all non-pivot elements with the pivot, and put the ones that are smaller in a list we call L, and the ones that are larger in a list we call R.[6]
3. Apply the quicksort algorithm to L and R if they contain more than one element.[7]
4. Return the list LpR.

[6] This is of course not the most efficient way to *implement* this algorithm – the correct thing to do is to move elements around in a single list, since this can be done "in place" and so will be faster. But this is mathematically equivalent, and easier to phrase.

[7] Since we did not include the pivot element in either set, they are both strictly shorter lists than the one we started with, so this recursion will terminate.

We can represent this algorithm with a rooted ordered binary[8] tree. This means the tree has one designated vertex we call the root, and every vertex has potentially a left child and a right child, where a "child" is a neighbour who is further from the root than themselves.

[8] Binary in the sense of a computer scientist, since this is after all an algorithm. A mathematician would have a slightly different definition of binary tree.

We create a tree from this algorithm by letting the pivot element be the root of the tree, and then recursively letting its left child be the root of of the tree quicksort gives us for the list L, and the right child be the root of the tree we get for the list R. This hopefully becomes clearer if we work through an example:

Consider the list $3, 7, 1, 4, 9, 8, 6, 2, 5$, and choose 7 as our pivot element. Then we get $L = 3, 1, 4, 6, 2, 5$ and $R = 9, 8$.

So, recursing, we now need to sort the list $3, 1, 4, 6, 2, 5$, and we pick 3 as our pivot element, getting a new $L = 1, 2$ and $R = 4, 6, 5$. If we pick 1 as our pivot in L, we get $L = \emptyset$ and $R = 2$. Picking 5 as out pivot in the previous R, we get $L = 4$ and $R = 6$.

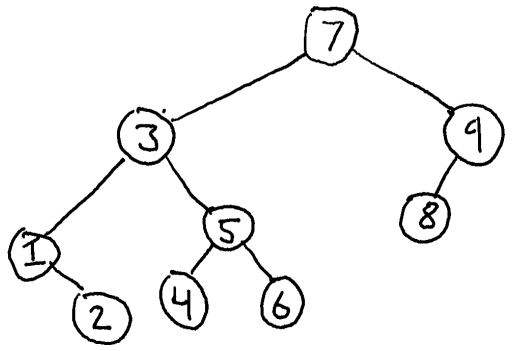

Figure 2: A tree gotten from quicksorting a list of integers in our example.

Finally, we need to quicksort the list $9, 8$, where we can pick 9 as our pivot and get $L = 8$, $R = \emptyset$. The resulting tree of what we just did is drawn in Figure 2.

Exercises