Feb 8, 2018

Lecture 11

Theorem: A tree with n vertices has $n-1$ edges.

Proof:

Without loss of generality we may assume that the tree is a rooted tree. There are $n-1$ vertices which are not the root and each has a unique incoming edge. This accounts for all the edges. q.e.d.

Definition: Let T be a rooted tree. Vertices of T with no children are called leaves of T. Vertices with children are called internal vertices of T. If every internal vertex of T has m children, T is called an m-ary tree. If $m=2$, T is a binary tree.

Theorem: Let T be an m-ary tree with n vertices of which i are internal. Then $n = mi+1$.

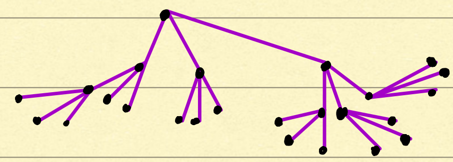
Proof: The internal vertices are exactly the vertices which are the parents of some vertex. The i internal edges have m children each, and since a vertex can have only one parent, this accounts mi children, and mi is the number of vertices which are children of some vertex. The root is the only vertex which is not the child of any vertex so adding that in we see that the total number of vertices is

$$n = mi+1.$$

q.e.d.

Suppose T is an m-ary tree with i internal vertices and l leaves. If n is the total number of vertices then

$$l+i=n.$$



3-ary tree.

On the other hand, from the previous theorem we have $n=mi+1$.

We thus have two equations

$$\left.\begin{aligned} mi+1&=n \\ l+i&=n. \end{aligned}\right\} \text{———} \quad (*)$$

The system of equations $(*)$ can be re-arranged in a number of ways.

Corollary on bottom of p.95 of the textbook.

$$\left.\begin{aligned} l&=(m-1)i+1 \\ n&=mi+1 \end{aligned}\right\} \text{———} \quad (a)$$

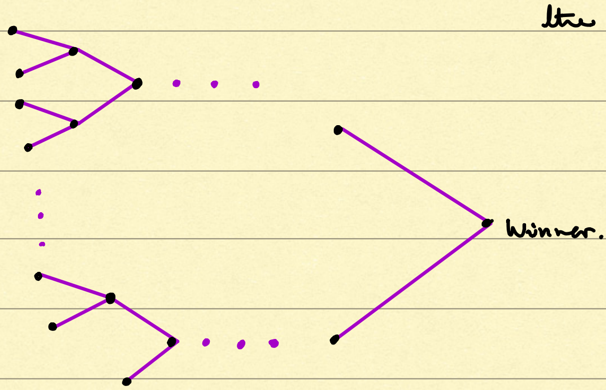
$$\left.\begin{aligned} i&=\frac{l-1}{m-1} \\ n&=\frac{ml-1}{m-1} \end{aligned}\right\} \text{———} \quad (b)$$

$$\left.\begin{aligned} i&=\frac{n-1}{m} \\ l&=\frac{(m-1)n+1}{m} \end{aligned}\right\} \text{———} \quad (c)$$

(a), (b), and (c) are systems of equations, each equivalent to the system $(*)$ given above.

One can use (a) to find l and n if i and m are given; (b) to find i and n if l and m are given; and (c) to find i and l if n and m are given.

Example: A tennis tournament can be modelled as a binary tree with the 1st round entrants as the leaves. The matches are the internal vertices.



Q: If 56 people sign up for a tennis tournament, how many matches will be played?

Solution: Here $l = 56$ and $m = 2$. Use the system of eqns in (b). We have to find i.

$$i = \frac{l-1}{m-1} = \frac{56-1}{2-1} = 55.$$

So 55 matches will be played.

Definitions: The height of a rooted tree is the length of the longest path from the root. The level number of a vertex x in a rooted tree is the length of the unique path from the root to x. A rooted tree of height h is called balanced if all leaves are at levels h and $h-1$.

§ 3.2 Search trees and spanning trees

Definition: A spanning tree of a graph G is a subgraph of G that is a tree containing all vertices of G.

There are two ways of constructing spanning trees.

- Depth first search
- Breadth first search

We will explain each way via an example. The graph we use for both examples is described an adjacency matrix.

Definition: An adjacency matrix of an (undirected) graph is a matrix with a 1 in entry (i,j) if vertices x_i and x_j are adjacent; entry (i,j) is 0 otherwise.

Suppose the graph G is given by the following adjacency matrix

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
x_1	0	1	1	1	0	1	0	0
x_2	1	0	1	0	1	0	1	0
x_3	1	1	0	0	0	0	0	0
x_4	1	0	0	0	0	0	1	0
x_5	0	1	0	0	0	0	1	0
x_6	1	0	0	0	0	0	0	0
x_7	0	1	0	1	1	0	0	1
x_8	0	0	0	0	0	0	1	0

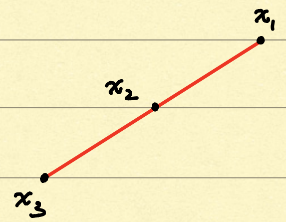
$(*)$

Depth first search: If we have a graph G and we wish to search for a spanning tree using the depth first approach, here is what we do. Start from some vertex. This vertex will be the root of the spanning tree. From this vertex keep following edges. Each of this edges will be part of your spanning tree. STOP when you cannot go further without repeating a vertex already in the tree you are building. The path you have followed is a path in the spanning tree you are building and the vertex you were forced to stop at, a leaf of this tree. Now backtrack to the parent of this leaf and start building a path from this parent in a different direction. When all possible paths from this parent y and its children have been built, we backtrack to the parent of y, and so on.

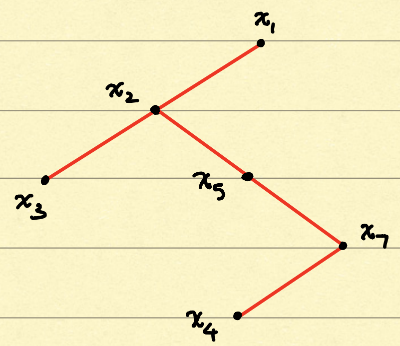
Let us see how this works for the graph given by the adjacency matrix (*) above.

Let us start with x_1 (we have a choice, but say we decide to start with x_1). Now (x_1, x_2) is an edge. It is the first edge in our build up of the spaning tree. So from x_1 we go to x_2. From x_2 let us go to x_3 (Note (x_2, x_3) is also an edge). Now x_3 is adjacent only to x_1 and x_2 and so there is no further place to go. This means x_3 will be a leaf in the spanning tree we are attempting to build.

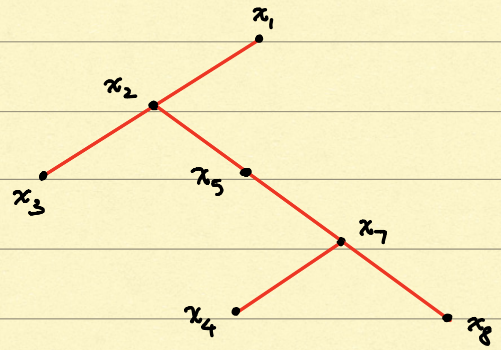
The situation so far:



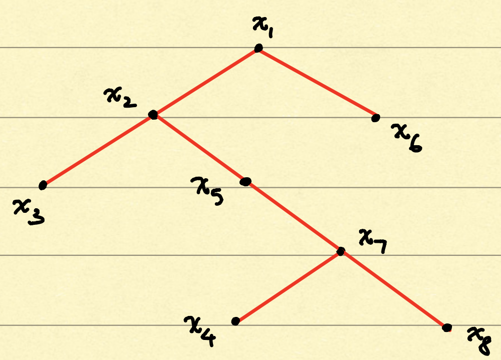
Now backtrack. The parent of the leaf x_3 is x_2. Build a path in a different direction starting from x_2. The vertex x_2 is adjacent to x_1, x_3, x_5, and x_7. Of these x_1 and x_3 have already been used up. So let us move from x_2 to x_5. Then from x_5 move to x_7 (there is no other choice). From x_7 let us move to x_4 (the vertices adjacent to x_7 are x_2, x_4, x_5, and x_8; and x_2 and x_5 have been used). We are now in the following state in our attempt to build a spanning tree.



Note that we have to stop building this path at x_4 because x_4 is adjacent to x_1 and x_7 both of which have been used up. Thus x_4 is a leaf. We now backtrack to its parent x_7 and follow a different direction. The only possibility is to move to x_8 and we have to stop this path there because x_8 is only adjacent to x_7. Here is where we are now:

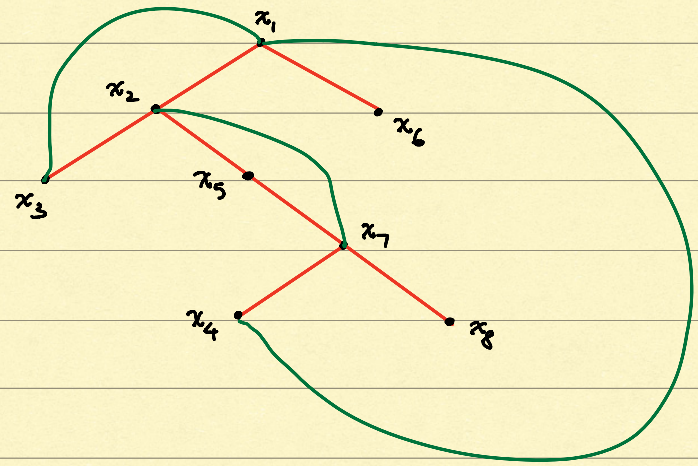


Since x_8 is a leaf, we backtrack to its parent x_7. But we have already exhausted all paths from x_7. Similarly, the parent of x_7, namely x_5 does not give us new paths. We have to go all the way back to x_1. The only vertex adjacent to x_1 that has not been used so far is x_6. So move from x_1 to x_6. And this exhausts all vertices. So the spanning tree obtained by this process and these choices is:



A spanning tree for the graph whose adjacency matrix is (*).

For the record, the graph given by (*) is not a tree. The following picture represents the graph given by (*).

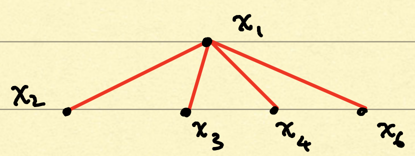


The graph given by (*). It is not a tree.

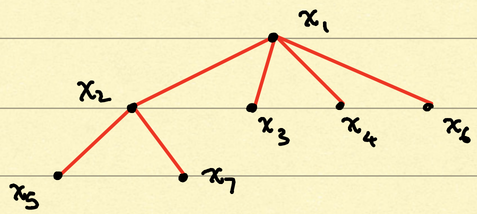
Breadth first search: Again pick a vertex (your choice) to be the root of the spanning tree. Now build a spanning tree as follows. Let this root be called x. Now put all edges leaving x, along with the vertices at the end of these edges, into the spanning tree we are building. Now go to all the vertices adjacent to x, and to each of them add all the edges incident on it, unless it ends in a vertex already in the spanning tree we are building. We keep doing this in a level-by-level fashion.

For the graph given by (*), here is how this algorithm works out:

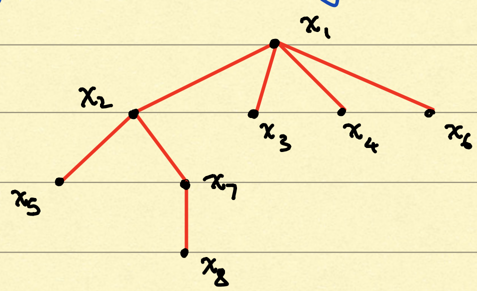
Start with x_1 as the root. It is adjacent to x_2, x_3, x_4, and x_6. At the first stage we build this:



Now x_2 is adjacent to x_1, x_3, x_5, and x_7. Of these, x_1 and x_3 have already been used. So we add x_5 and x_7 to the mix. We now get:



The only remaining vertex is x_8 and it is adjacent to x_7. And this gives the following spanning tree:



Testing for connectedness: Use a depth-first or breadth-first algorithm to search for a spanning tree. If all vertices in the graph are reached in the search, a spanning tree is obtained and the graph is connected. If the search does not reach all the vertices, the graph is not connected.

* **Important:** A breadth-first spanning tree consists of shortest paths from the root to every other vertex in the graph. (Exercise 8 in Section 3.2.)

Example (Pitcher-Pouring puzzle).

We are given three pitchers of sizes 10 quarts, 7 quarts, and 4 quarts. Initially the 10 quart pitcher is full of water and the other two pitchers are empty. One is allowed to pour water from one pitcher to another provided we do it until either the receiving pitcher is full or the pouring pitcher is empty. The aim is to pour in such a way that we end up with either the 7 quart or the 4 quart pitcher having exactly 2 quarts in it. Is this possible? If so what is the minimal sequence of pourings needed for this?

At any stage we have three numbers $a, b,$ and c, giving the volume of water in the 10 quart, 7 quart, and 4 quart pitcher respectively. If we know b and c, then $a = 10 - b - c$. So the data (b,c) captures the volume of water in all three pitchers.

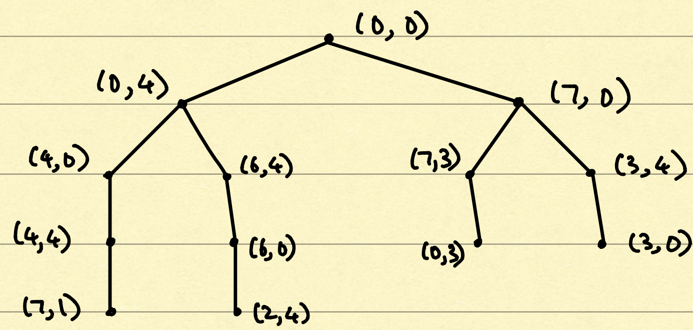
We model this graph theoretically as follows. The vertices are the data (b,c). There is an edge from (b,c) to (β,γ) if by a single pouring we move from (b,c) to (β,γ). For example if the volume of water in the pitches 3, 7, and 0 and we pour water from the first to the third, we have moved from $(7,0)$ to $(7,3)$.

In the beginning all the water is in the 10 quart pitcher and so $b=0$ and $c=0$.

Let us start with vertex $(0,0)$. We want a minimal sequence. So we will use breadth-first. The vertices adjacent to $(0,0)$ are $(7,0)$ and $(0,4)$.

From (7,0) we get to new position (7,3) and (3,4).
From (0,4) the only new positions we can get to are (6,4) and (4,0).

Here is how it all unfolds:



Stop searching once you get here, since now the 7-quart pitcher has 2 quarts of water.

Since we used breadth-first, this gives us the minimal sequence. The minimal sequence is

$$(0,0) \longrightarrow (0,4) \longrightarrow (6,4) \longrightarrow (6,0) \longrightarrow (2,4)$$

//