Summary on Lecture 10, August 4, 2015

More on Rooted Trees

Let $m \geq 1$. Recall that a rooted tree (T, r) is a *complete m-ary tree* if every vertex of T has either m children or no children. Mostly we are interested in the case $m = 2$.

Lemma 1. *Let (T, r) be a complete binary tree. Then $|V(T)|$ is odd.*

Exercise. Prove Lemma 1 by induction.

We would like to count how many complete binary trees are there with $2n + 1$ vertices.

Let (T, r) be a complete binary tree with $2n + 1$ vertices. We use preorder listing to give a a list of all vertices (starting with the root): $rv_1v_2 \dots v_{2n}$. We notice that every move from v_i to v_{i+1} has a direction: its either left (L) or right (R). Hence the list $rv_1v_2 \dots v_{2n}$ gives a sequence of $2n$ L's and R's. Then we notice:

- We visit first the "left" child, then the "right" one. Thus if we count how many L's and R's from the beginning to a given spot, we'll get that the number of L's is greater or equal to the number of R's.
- There are n L's and n R's.

We have seen this problem before, and conclude that the number of such listings (and, consequently, the number of complete binary graphs with $2n + 1$ vertices) is nothing but the *Catalan number*, namely, $\frac{1}{n+1}\binom{2n}{n}$.

Now let $G = (V, T)$ be a connected graph without loops and multiple edges. We assume that the vertices of G are ordered, i.e., $V = \{v_1, \dots, v_n\}$. We would like to find a spanning tree (T, r) (which is *depth-first ordered rooted tree*).

Here is a pseudocode for a recursive version of the Depth-First-Search algorithm:

Depth-First-Search (G, v)
```
    Let v := v_1.  Put v to the list T
    For all edges from v to w in E(G) do
          if w is not in T then call T(G,w) := Depth-First-Search(G,w),
          T := T ∪ T(G,w)
    Return T
```

Exercise. Use **Depth-First-Search** (G, v) algorithm for several large graphs. Find non-trivial examples.

Exercise. Study the **Breadth-First-Search** (G, v) algorithm from the textbook and write a pseudocode for its recursive version.