1. Reading: K. Rosen *Discrete Mathematics and Its Applications*, 8.1, 8.2

2. The main message of this lecture:

Probably the most heavily used type of graphs are trees, i.e. the ones without cycles.

Definition 34.1. A **tree** is a connected undirected acyclic graph. Here "acyclic" means that the graph does not have simple circuits, or cycles. In particular, loops and parallel edges are ruled out by this definition. If the connectivity condition is dropped, T is called a **forest**.

Examples 34.2. (See slides.) Everyday life examples: single-elimination tournament graphs, administrative organizational charts, computer file systems, family trees, formation trees of propositions, etc.

Theorem 34.3. *An undirected graph is a tree if and only if for each pair of vertices there is a unique path between them.*
Proof. Assume T is a tree, i.e. T is connected and does not have (simple) circuits. Pick a, b a pair of vertices in T. If $a = b$, a simple circuit between does not contain edges at all, and thus is unique. Suppose $a \neq b$. Since T is connected there exists a path between a and b. Moreover, this path is unique. Indeed, suppose there are two different paths P_1 and P_2 between a and b. We may assume that neither P_1 nor P_2 has cycles (otherwise, a contradiction is already here). Let a_1 be the first vertex on P_1 after which P_2 deviates from P_1 for the first time. Let also b_1 be the first vertex on P_1 after a_1 belonging to P_2 as well. Then the parts of P_1 and P_2 between a_1 and b_1 constitute a simple circuit, which is impossible.

Assume now that each pair of vertices in T has a unique path between them. We have to show that T is a tree, i.e. that T is connected and acyclic. The connectivity is immediate, since every two distinct vertices are connected by a path. Suppose there is a simple circuit C in T. Case A: C is a loop around one vertex v. Then there are two different paths around v: C and a path with the empty set of edges. Case B: there are at least two distinct vertices u, v on C. Then two different parts of C connecting u and v constitute two different paths in T between u and v.

Definition 34.4. A **rooted** three is a tree with one designated vertex called **root**. Given a tree T one could easily make it a rooted tree: just pick any vertex r as a root, and let the rest if T "hang" from r (see examples on slides). Note that the same tree can produce different rooted trees (see slides). A rooted tree already has a well defined notion of **direction** and **succession**. Since for each vertex v there is a unique path going from the root r to v, all the vertices between r and v may be regarded as **ancestors** of v. An immediate successor of v is called a **child** of v (there may be many of them for a given v). Likewise, we can define **siblings**, **descendants**, etc. Vertices with children are called **internal vertices**. A vertex without children is called a **leaf**. (See slides for more examples).

Definition 34.5. A rooted three in an **m-ary tree** if any vertex has no more than m children. An m-ary tree with $m = 2$ is called a **binary tree**. The tree is called a **full m-ary tree** if every internal vertex has exactly m children.

Definition 34.6. An **ordered rooted tree** is a rooted tree where the children of each internal vertex are ordered. In an ordered binary tree we distinguish between the **left child** and the **right child**, which specify the **left subtree** and the **right subtree** of a given vertex correspondingly.

Example 34.7. Above examples: single-elimination tournament graphs, administrative organizational charts, computer file systems, family trees are all rooted trees. The formation trees of propositions are ordered rooted trees, since the order of propositions by a connective matters ($p \to q$ is different from $q \to p$).

Theorem 34.8. *A tree with n vertices has $n-1$ edges. There are at most $1+m+m^2+\ldots+m^h$ vertices in a full m-ary tree of height h.*
Proof. (slides!).

Example 34.9. A **binary search tree** is an ordered binary tree T with vertices labeled by data (e.g. words). The data are arranged so that, for each vertex v in T, each data item in the left subtree of v is less than the data item in v, and each data item in the right subtree of v is greater than the data item in v. Locating an item in a good binary search trees (mathematically this corresponds to so-called balanced trees) takes log steps of the total number of items in the database. Cf. slides for different examples of arranging the words *OLD PROGRAMMERS NEVER DIE THEY JUST LOSE THEIR MEMORIES* as binary search diagrams.

Example 34.10. Prefix codes has the property that an element of a code cannot be the beginning string of another element of a code. For example, let the strings 111, 101, 01, 11001, 0010, 1101 represent the letters *c, d, e, i, s, u* respectively. First of all, note that this code is a prefix code. As a result, code sequences admit unique decoding. Indeed, try to read 101011111100110101101. First, we match 101 with *"d"* and parse in out : *d**011111100110101101. Then

01 is the code for *"e"*: *de**1111100110101101,
111 stands for *"c"*: *dec**1100110101101,
11001 for *"i"*: *deci**10101101,
101 for *"d"*: *decid**01101,
01 is again *"e"*: *decide**101,
finally, 101 is *"d"*: *decided*.

In a binary tree label every *left edge* by 0 and every *right edge* by 1. Then every leaf (in fact, every node) in such a tree defines a unique string of bits corresponding to the unique path from the root to a given vertex. We shall call the strings corresponding to a given leaf the **leaf path code**. There is a neat coincidence between binary trees and bit prefix codes.

Theorem 34.11. *In any binary tree, the leaf path codes for the leaves in the tree are a prefix code.*
Proof. (By example, see slides).

Homework assignments. (The second installment due Friday 04/27)

34A:Rosen8.1-2; 34B8.1-10:Rosen; 34C:Rosen8.2-12b 34D:Rosen8.2-14d.