Hamilton paths & circuits

Def. A path in a multigraph is a Hamilton path if it visits each vertex exactly once.

Def. A circuit that is a Hamilton path is called a Hamilton circuit.

Hamilton circuits

Constructing a path to visit each vertex exactly once is a natural problem, but no good solution has been found.

Clearly the more edges we have in the graph the more likely we are to be able to solve the problem.

Note: we have to visit each vertex, not each edge.

Hamilton Circuits

Dirac's Theorem (1952)

Suppose G is a simple graph with n vertices (n > 2) and the degree of every vertex is at least n/2. Then G has a Hamilton circuit.

See also Ore's Theorem (1960)
Both are on p. 701 (6th ed p. 641).

I won't expect you to memorize these.

Gray codes

An interesting application of Hamilton circuits is the Gray code, designed to minimize the effect of errors in the transmission of digital signals.

See p. 702 (6th ed. p. 642) for the details, if you are interested.

The Traveling Salesman Problem

The **traveling salesperson problem,** which we looked at earlier, is very like the Hamilton circuit problem, except that in addition to finding a Hamilton circuit we want to minimize the weight of the circuit.

The problem here is to find the **circuit of minimum total weight that visits each vertex exactly once**.

Planar Graphs

Def. A graph is called planar if it can be drawn in the plane without any two edges crossing.

These graphs are discussed in section 10.7

Homeomorphic graphs

If a graph is planar and we divide an edge by adding a new vertex somewhere in the middle, we get a new planar graph, essentially equivalent to the first.

If two graphs can be subdivided in this way and are then isomorphic we say they are homeomorphic.

A theorem of Kuratowski

Theorem (p. 724) (6th ed. p. 664).

A graph is nonplanar iff it contains a subgraph homeomorphic to $K_{3,3}$ or to K_5

$K_{3,3}$ is the complete bipartite graph with partitions of size 3 and 3.

K_5 is the complete graph on 5 vertices.

More on Kuratowski

For more on $K_{n,m}$ see http://en.wikipedia.org/wiki/Complete_bipartite_graph

For more on K_n see http://en.wikipedia.org/wiki/Complete_graph

There are more interesting results in section 10.7, but they are left for investigation by the curious reader.

The Four Color Theorem

In 1850 the question was raised as to the minimum number of colors required to color a map so that no two adjacent regions (e.g. countries) have the same color.

Two regions are adjacent if they have a common boundary of length greater than 0.

The Four Color Theorem

This question was settled in 1976, when it was proved that four colors would suffice. The proof analyzed thousands of cases, using a computer, and, apparently, the kids of the authors, to work out some of the cases.

Four Color Theorem

This question can be posed in graph theory, where we construct a planar graph with each vertex representing a region on the map, with an edge joining any two adjacent regions.

The color condition here would be that no two adjacent vertices have the same color.

Those interested can read about this in 10.8

Let's talk about…

Trees

Chapter 11

Trees

Definition: A **tree** is a connected undirected graph with no simple circuits (Circuit: a path beginning and ending at the same vertex. Simple: no edge is in there more than once)

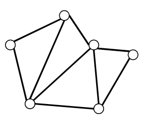
Since a tree cannot have a simple circuit, a tree cannot contain multiple edges or loops.

Therefore, any tree must be a **simple graph**.

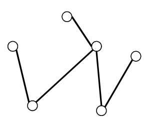
Theorem: (page 746) An undirected graph is a tree if and only if there is a **unique simple path** between any of its vertices.

Trees

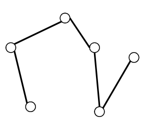
Example: Are the following graphs trees?



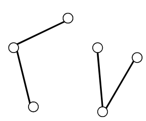
No.



Yes.



Yes.



No.

Trees

Definition: An undirected graph that does not contain simple circuits and is not necessarily connected is called a **forest**.

In general, we use trees to represent **hierarchical structures**.

We often designate a particular vertex of a tree as the **root**. Since there is a unique path from the root to each vertex of the graph, we direct each edge away from the root.

Thus, a tree together with its root produces a **directed graph** called a **rooted tree**.

Tree Terminology

If v is a vertex in a rooted tree other than the root, the **parent** of v is the unique vertex u such that there is a directed edge from u to v.

When u is the parent of v, v is called the **child** of u.

Vertices with the same parent are called **siblings**.

The **ancestors** of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root.

Tree Terminology

The **descendants** of a vertex v are those vertices that have v as an ancestor.

A vertex of a tree is called a **leaf** if it has no children.

Vertices that have children are called **internal vertices**.

If a is a vertex in a tree, then the **subtree** with a as its root is the subgraph of the tree consisting of a and its descendants and all edges incident to these descendants.

Tree Terminology

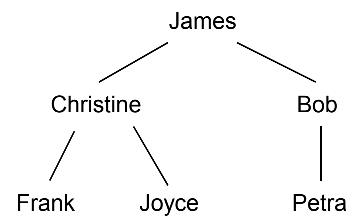
The **level** of a vertex v in a rooted tree is the length of the unique path from the root to this vertex.

The level of the root is defined to be zero.

The **height** of a rooted tree is the maximum of the levels of vertices.

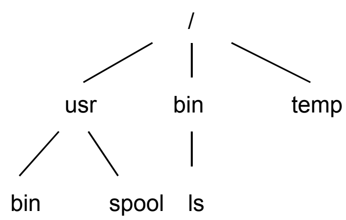
Trees

Example I: Family tree



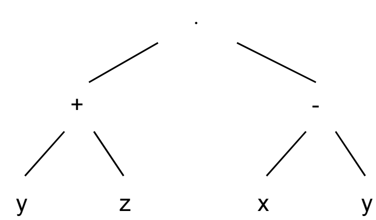
Trees

Example II: File system



Trees

Example III: Arithmetic expressions



This tree represents the expression (y + z)·(x - y).

Trees

Definition: A rooted tree is called an **m-ary tree** if every internal vertex has no more than m children.

A tree is called a **full m-ary tree** if every internal vertex has exactly m children.

A tree is called a **complete m-ary tree** if it is full and every leaf is on the same level.

An m-ary tree with m = 2 is called a **binary tree**.

Theorem: A tree with n vertices has (n – 1) edges.

Theorem: A full m-ary tree with i internal vertices contains n = mi + 1 vertices.

Theorem: A tree with n vertices has (n – 1) edges.

Proof.

Induction on n, no. of vertices.

Base case: n = 1. 1 vertex, 0 edges

Induction step.

A tree with n vertices can be created by adding a leaf to a tree with n-1 vertices. This adds one vertex and one edge.

Theorem: A full m-ary tree with i internal vertices contains n = mi + 1 vertices.

Proof, again by induction

Base case. i = 0. 1 vertex

Induction step. Suppose true for i-1.

Select a vertex of a tree with i internal vertices whose m children are all leaves.

Remove these leaves. Then we have a tree with i-1 internal vertices, so m(i-1) +1 vertices, by induction.

Our original tree thus had

m(i-1) + 1 + m = mi + 1 vertices.

Binary Search Trees

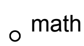
If we want to perform a large number of searches in a particular list of items, it can be worthwhile to arrange these items in a **binary search tree** to facilitate the subsequent searches.

A binary search tree is a binary tree in which each child of a vertex is designated as a **right or left child**, and each vertex is labeled with a **key**, which is one of the items.

When we construct the tree, vertices are assigned keys so that the key of a vertex is both larger than the keys of all vertices in its left subtree and smaller than the keys of all vertices in its right subtree.

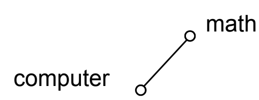
Binary Search Trees

Example: Construct a binary search tree for the strings **math, computer, power, north, zoo, dentist, book**.



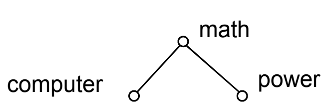
Binary Search Trees

Example: Construct a binary search tree for the strings **math, computer, power, north, zoo, dentist, book**.



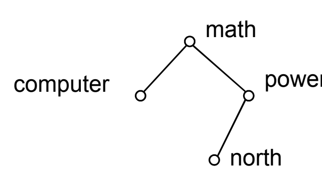
Binary Search Trees

Example: Construct a binary search tree for the strings **math, computer, power, north, zoo, dentist, book**.



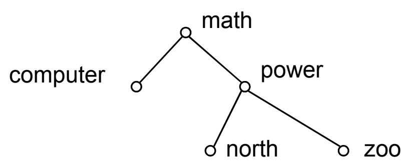
Binary Search Trees

Example: Construct a binary search tree for the strings **math, computer, power, north, zoo, dentist, book**.



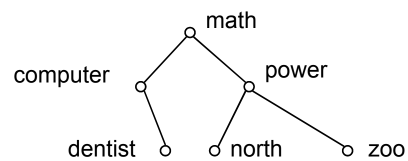
Binary Search Trees

Example: Construct a binary search tree for the strings **math, computer, power, north, zoo, dentist, book**.



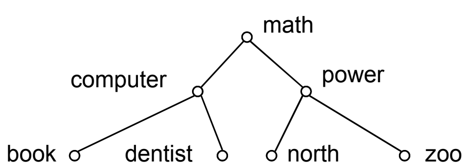
Binary Search Trees

Example: Construct a binary search tree for the strings **math, computer, power, north, zoo, dentist, book**.



Binary Search Trees

Example: Construct a binary search tree for the strings **math, computer, power, north, zoo, dentist, book**.



Binary Search Trees

To perform a search in such a tree for an item x, we can start at the root and compare its key to x. If x is **less** than the key, we proceed to the **left** child of the current vertex, and if x is **greater** than the key, we proceed to the **right** one.

This procedure is repeated until we either found the item we were looking for, or we cannot proceed any further.

In a balanced tree representing a list of n items, search can be performed with a maximum of $\lceil \log(n + 1) \rceil$ steps (compare with binary search).