Slide 1

Administrivia

- Reminder: Quiz 6 Wednesday.
- Homework 8 on Web. Due Friday.

Slide 2

Graphs — Review/Recap

- In context, "graph" is mathematical notion meant to represent relationships among a set of things — focus is on relationships, with details we don't care about abstracted out.
- Formal definition in terms of set of nodes (vertices), arcs (edges), and function mapping arcs to pairs of nodes.
- Much terminology. Key terms:
  - Directed versus undirected.
  - Adjacent nodes, paths, connected graphs.
  - Cycles, acyclic graphs.

Slide 3

Isomorphic Graphs

- What we care about is the relationship between nodes and arcs, not exact visual representation.
- Can formalize this as "isomorphism" — two graphs are isomorphic if one is just a "relabeling" of the other.
- Formal definition is in terms of one-to-one functions, one from nodes of graph G_1 to nodes of graph G_2 and one from arcs of graph G_1 to arcs of graph G_2. Idea is that if an arc connects two nodes in G_1, the corresponding arc in G_2 connects the corresponding nodes.

Slide 4

Computer-Friendly Representation of Graphs

- For humans, representing graphs pictorially usually works well. For computers, other representations work better.
- Key idea is to come up with a way to represent the essential information — set of nodes and which ones are connected.

Slide 5

Adjacency Matrices

- Idea is to put the n nodes in some (arbitrary) order and define an n-by-n matrix A such that A_{ij} is the number of arcs connecting node i and node j.
- For an undirected graph, what property does this matrix have? that it might or might not have for a directed graph?
- Variation: For a weighted graph with no parallel arcs, we could let A_{ij} be the weight of the arc from node i to node j.

Slide 6

Adjacency Lists

- Idea is to again put n nodes in some arbitrary order, but rather than a matrix define an array of n lists, one for each node, with the list for node i containing all nodes j that are adjacent to node i. Parallel arcs mean "duplicate" entries.

Slide 7

Adjacency Matrix Versus Adjacency List

- Which uses less space?
- Which makes it faster to answer the question "is node i adjacent to node j?"

Slide 8

Trees — Overview

- You probably know trees from PAD 2. In math terms, we can say a tree is a kind of graph — acyclic, connected, one node designated "root".
- Can be used to represent any kind of hierarchy, e.g.:
  - Table of contents of a book.
  - Hierarchical help system.
  - Arithmetic expression.

Slide 9

Trees — Terminology

- Some terminology should be familiar: root, subtree, parent, children, root of subtree.
- Other terms:
  - Depth of node (distance from root), height of tree (maximum depth).
  - Binary tree — at most two children per node.
  - Full binary tree.
  - Complete binary tree.

Slide 10

Trees — Recursive Definition

- Tree is either
  - A single node, or
  - The tree formed by combining a root r with (disjoint) subtrees t_1 through t_n.

Slide 11

Computer-Friendly Representation of Trees

- Can of course use any representation that works for general graphs.
- Can also use array representation for binary trees: Number the nodes from 1 to N, and make a 2-by-N array for left/right children.
- Can also use pointer-based representation — simpler for binary trees, but possible for general trees as well.

Slide 12

Tree Traversals

- For linear data structures (lists, arrays, etc.), basically only one reasonable way to "walk through" the structure to visit each element.
- For trees, there are three "reasonable" ways:
  - Preorder traversal (root first, then subtrees).
  - In-order traversal (leftmost subtree first, then root, then rest).
  - Postorder traversal (subtrees first, then root)
- Functions to perform any of these (e.g., and print each node as it is visited) are almost trivial to write write recursively, much more difficult without recursion.

Slide 13

Trees — Special Types

- Special types (familiar from PAD 2?):
  - Sorted binary tree.
  - Heap.

Slide 14

Trees and Recursion/Induction

- Easy to write other recursive algorithms to operate on trees — e.g., function to find height of tree.
- If we use the recursive definition of a tree, we can prove things about trees using induction. Example from textbook — prove that number of arcs is one less than number of nodes.

Slide 15

Minute Essay

- Write a recursive function to count the number of nodes in a tree. Use the datatype and functions from today's class, or just describe in words.
- Reminder: Homework 7 due by 5pm.

Slide 16

Minute Essay Answer

- One solution:

```
int countNodes(Tree t) {
    if (empty(t))
        return 0;
    else
        return 1 + countNodes(left(t)) + countNodes(right(t));
}
```