IE 495 Lecture 10

September 28, 2000

Reading for This Lecture

- Primary
  - Horowitz and Sahni, Chapter 2, Section 3

More Data Structures

Trees

- A (directed) tree is a directed acylic graph satisfying the following:
  - There is exactly one vertex called the root with in-degree 0.
  - Every other vertex has in-degree 1.
  - There is a path from the root node to every other node.
- Trees also have a natural recursive definition.
- Tree terminology
  - If $(u, v) \in E$, then u is called the *father / mother / parent* of v and v is called the *son / daughter* of u.
  - If there is a path from u to v, then v is a *descendant* of u and u is an *ancestor* of v.

More Tree Terminology

- A tree in which each node has out-degree 0, 1, or 2 is called a *binary tree*.
- A tree in which the sons are ordered is called an ordered tree.
- In an ordered binary tree, the two sons are usually called the *left son* and the *right son*.
- The *depth* or *level* of a vertex v is the length of the (unique) path from the root to v.
- The depth of a tree is the maximum depth of any node.

Trees and data structures

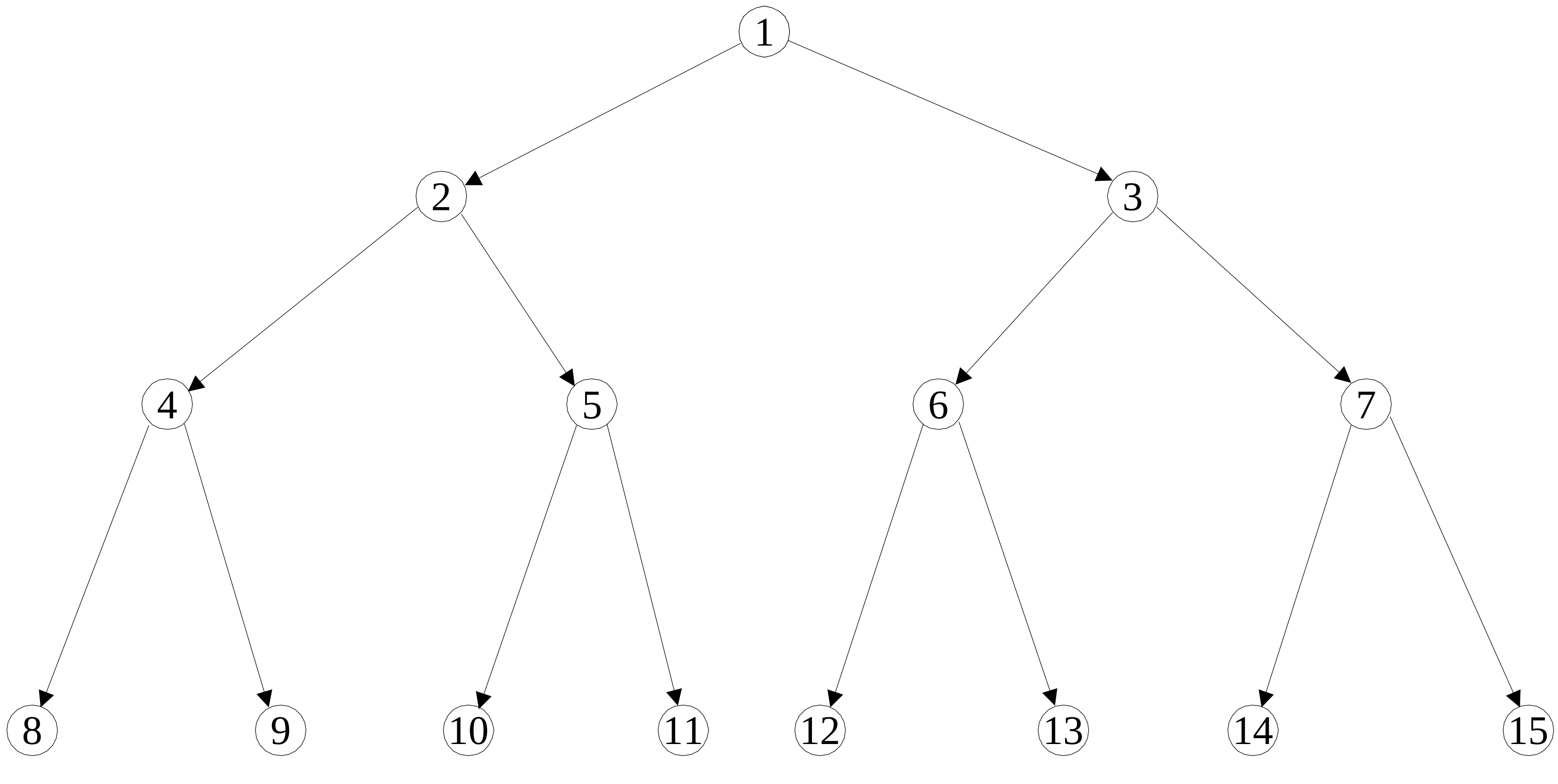
- Trees are an element of many different data structures.
- Trees are naturally associated with recursive and divide and conquer type algorithms.
- We have already seen how trees can help us partition the elements of a set.
- Sample tree operations
  - *link()*
  - *delete_node()*
  - *add_node()*

Storing a binary tree

- Arrays
  - Parent of node i is stored in location $\lfloor i/2 \rfloor$.
  - Easy to go to a specific node.
  - Can use up lots of memory if unbalanced (2^l elements).
  - Not efficient for some tree operations.
- Pointers
  - Can be more memory efficient if unbalanced.
  - Easier tree operations in some cases.

Traversing a Tree

- Many common algorithms involve traversing or searching a tree.
- Traversal schemes
  - preorder
  - postorder
  - depth-first
  - breadth-first


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Binary Search Tree

- A binary search tree is a binary tree satisfying
  - The value stored at *X* is greater than the values in the left child and all its descendants.
  - The value stored in *X* is less than the values in the right child and all its descendants.
- A binary tree can be used to easily perform binary search.

Heaps

- A *heap* is a binary tree in which the value at each node is at least as large as the values in each of its children.
- Hence, a largest element is always at the root.
- Heaps support the following operations
  - *`insert()`*
  - *`delete_max()`*
  - *`make_heap()`*
  - *`adjust_heap()`*
- Heaps implement a priority queue.

Inserting into a heap

- Insert the value into node $n+1$ and "bubble up".
- Compare the value to its parent and swap if necessary.
- Continue swapping until heap property is restored.
- One way to make a heap from n elements is to simply insert them one at a time.
- Analysis
  - *`insert()`*
  - *`make_heap()`*

Adjusting a heap

- If only the root of a heap is out of order, we can restore order by "bubbling down" (*`adjust()`*).
  - Swap the root with the larger child.
  - Continue swapping process until heap property is restored.
- Heapify (create a heap by iterative adjusting)

  For each node i = $\lfloor n/2 \rfloor \rightarrow 1$

  adjust node *i* w.r.t. the subtrees rooted at its children

- Analysis

Deleting from a heap

- To delete the root node,
  - exchange node 1 with node *n*.
  - adjust the heap.
- Heapsort
  - First heapify.
  - Iteratively delete the root node.
- Analysis