CSC115 Lecture 15

LillAnne Jackson

But First: A little quiz on Queues and Stacks!

Trees

1. Definitions & Terminology
2. *Binary Tree ADT*
3. *Tree Traversal*

A Little Quiz #1

```
public interface Stack<E>
{       void    push (E element);
        E       pop() throws StackEmptyException;
        E       peek() throws StackEmptyException;
        boolean isEmpty();
}
```

```
public interface Queue<T> {
        public void enqueue(T element);
        public T dequeue() throws QueueEmptyException;
        public boolean isEmpty();
}
```

Assume an implementation of the Stack and Queue interfaces. Draw the queue and stack the result from:

```
Queue<String>  he = new linkQueue<String>;
Stack<String> she = new arrayStack<String>;
he.enqueue("This");
he.enqueue("is");
she.push(he.dequeue());
she.push("it");
he.enqueue("Wow!");
she.push(he.dequeue());
```

A Little Quiz #2

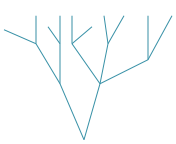
```
public interface Stack<E>
{       void    push (E element);
        E       pop() throws StackEmptyException;
        E       peek() throws StackEmptyException;
        boolean isEmpty();
}
```

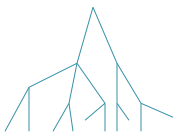
```
public interface Queue<T> {
        public void enqueue(T element);
        public T dequeue() throws QueueEmptyException;
        public boolean isEmpty();
}
```

Assume an implementation of the Stack and Queue interfaces. Draw the queue and stack the result from:

```
Queue<double>  act = new linkQueue<double>;
Stack<double> nit = new arrayStack<double>;
nit.push(3.14159);
act.enqueue(2.718286);
act.enqueue(43.23);
nit.push(12.43);
act.enqueue(nit.pop());
act.enqueue(nit.pop());
```

Trees



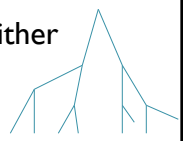


General tree

- A general tree T is a set of one or more nodes such that T is partitioned into disjoint subsets:
  - A single node r, the root
  - Sets that are general trees, called subtrees of r

Binary tree

- A binary tree is a set T of nodes such that either
  - T is empty, or
  - T is partitioned into three disjoint subsets:
    - A single node r, the root
    - Two possibly empty sets that are binary trees, called left and right subtrees of r



Terminology

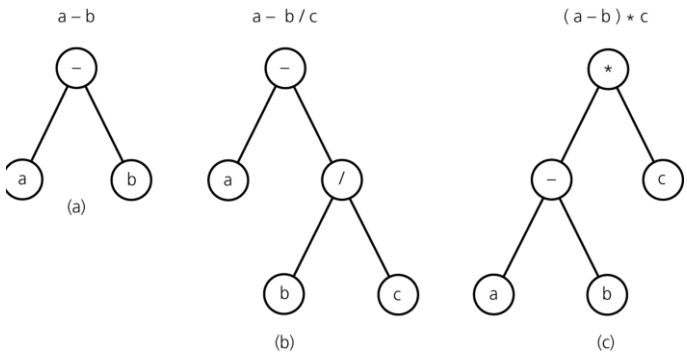


Figure 11-4

Binary trees that represent algebraic expressions

Terminology

A binary search tree

- A binary tree that has the following properties for each node n
  - n's value is greater than all values in its left subtree T_L
  - n's value is less than all values in its right subtree T_R
  - Both T_L and T_R are binary search trees

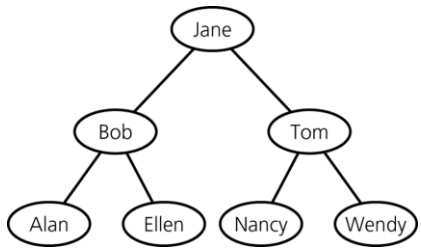


Figure 11-5

A binary search tree of names

Tree Terminology

Level of a node, n, in a Tree, T

- If n is the root of T, it is at level 1
- If n is not the root of T, its level is 1 greater than the level of its parent

Height of a tree, T

- If T is empty, its height is 0
- If T is not empty, its height is equal to the maximum level of its nodes

Terminology

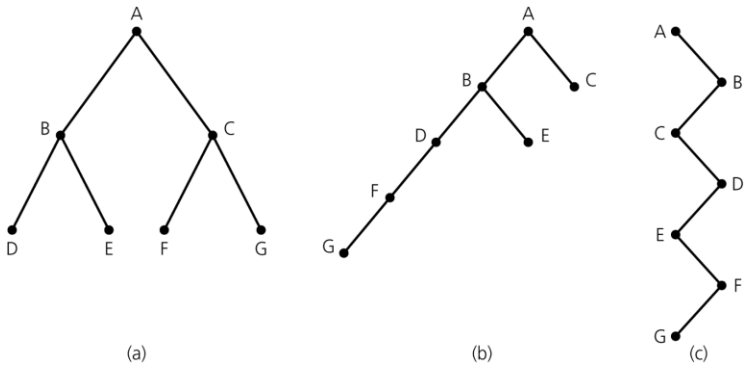


Figure 11-6

Binary trees with the same nodes but different heights

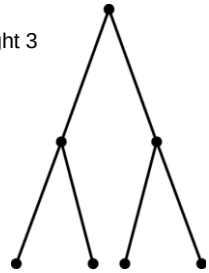
Definitions: Full, Complete, Balanced

Full Binary Trees

- If T is empty, T is a *full* binary tree of height 0
- If T is not empty and has height h > 0, T is a full binary tree if its root's subtrees are both full binary trees of height h – 1

Observe: This is a *Recursive Definition!!*

Example:
A full binary tree, height 3



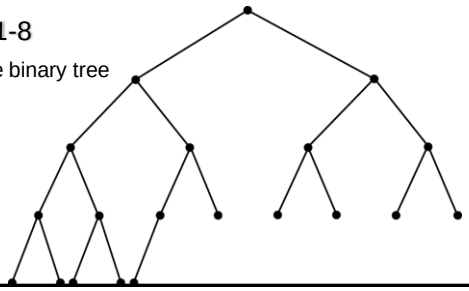
Definitions: Full, Complete, Balanced

Complete binary trees

- A binary tree T of height h is *complete* if
  - All nodes at level h – 2 and above have two children each, and
  - When a node at level h – 1 has children, all nodes to its left at the same level have two children each, and
  - When a node at level h – 1 has one child, it is a left child

Another *Recursive Definition!!*

Figure 11-8
A complete binary tree



Definitions: Full, Complete, Balanced

Balanced binary trees

- A binary tree is *balanced* if the height of any node's right subtree differs from the height of the node's left subtree by no more than 1

Full binary trees are complete

Complete binary trees are balanced

 11 A-11

Tree Terminology

General Tree
- A set of one or more nodes, partitioned into a root node and subsets that are general subtrees of the root

Parent of node n
- The node directly above node n in the tree

Child of node n
- A node directly below node n in the tree

Root
- The only node in the tree with no parent

Leaf
- A node with no children

Siblings
- Nodes with a common parent

Ancestor of node n
- A node on the path from the root to n

Descendant of node n
- A node on a path from n to a leaf

Subtree of node n
- A tree that consists of a child (if any) of n and the child's descendants

Tree Terminology: Summary

Height
- Number of nodes on the longest path from the root to a leaf

Binary tree
- A set of nodes that is either empty or partitioned into a root node and one or two subsets that are binary subtrees of the root
- Each node has at most two children: left child & right child

Left (right) child of node n
- A node directly below and left (right) of node n in a binary tree

Left (right) subtree of node n
- In a binary tree, left (right) child (if any) of n plus its descendants

Binary search tree
- The value in any node n is greater than every node in n's left subtree, but less than every node in n's right subtree

Empty binary tree
- A binary tree with no nodes

Tree Terminology: Summary

Full binary tree
- A binary tree of height h with no missing nodes
- All leaves are at level h and all other nodes each have two children

Complete binary tree
- A binary tree of height h that is full to level h – 1 and has level h filled in from left to right

Balanced binary tree
- A binary tree in which the left and right subtrees of any node have heights that differ by at most 1

Observe: These definitions are not given recursively!!

The ADT Binary Tree: Basic Operations of the ADT Binary Tree

- The operations available for a particular ADT binary tree depend on the type of binary tree being implemented
- Basic operations of the ADT binary tree
  - `createBinaryTree()`
  - `createBinaryTree(rootItem)`
  - `makeEmpty()`
  - `isEmpty()`
  - `getRootItem() throws TreeException`

General Operations of the ADT Binary Tree

- General operations of the ADT binary tree
  - `createBinaryTree (rootItem, leftTree, rightTree)`
  - `setRootItem(newItem)`
  - `attachLeft(newItem) throws TreeException`
  - `attachRight(newItem) throws TreeException`
  - `attachLeftSubtree(leftTree) throws TreeException`
  - `attachRightSubtree(rightTree) throws TreeException`
  - `detachLeftSubtree() throws TreeException`
  - `detachRightSubtree() throws TreeException`

Traversals of a Binary Tree

- A traversal algorithm for a binary tree visits each node in the tree
- Recursive traversal algorithms
  - Preorder traversal
  - Inorder traversal
  - Postorder traversal
- Traversal is O(n)

Traversal of a Binary Tree

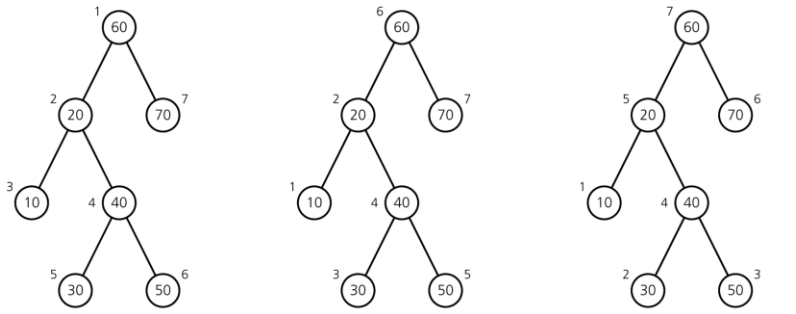


Figure 11-10

Traversals of a binary tree: a) preorder; b) inorder; c) postorder

