INFSCI 2500
Lecture 7
Binary Trees

Today's Plan

- HW 3 review
- Midterm review
- Understand what binary trees are
- Binary Tree Theorem
- External Path Length Theorem
- HW 4 Assign

HW3 Review

Midterm Review

Binary Tree Definition

A binary tree t is either empty or consists of an element, called the root, and two distinct binary trees called left subtree and right subtree.

leftTree(t)

rightTree(t)

Binary Tree Definition

- Root is shown at the top
- Elements are connected by lines
- Subtrees are also binary trees

- Examples 1-4

Binary Tree Properties

- tree – whole structure
- root – topmost element
- branch – line from root to subtree
- leaf – element with empty subtrees

Binary Tree Properties

- Number of leaves in a tree t (pseudocode)

```
if t is empty
        leaves(t) = 0
else if t consists of root only
        leaves(t) = 1
else
        leaves(t)=leaves(leftTree(t))+
                                leaves(rightTree(t))
```

Binary Tree Properties

- Each element is uniquely identified by location (ex 5)

element value = "-" at root(t)

element value = "-" at root(rightTree(leftTree(t)))

Binary Tree Properties

- Parent – X is parent of Y and Z (ex 6)
- Left child – Y is left child of X
- Right child – Z is right child of X

- Each element has 0,1, or 2 children
- Each element has 0 or 1 parent

Binary Tree Properties

- Ancestor – A is an ancestor of B if B is in the subtree with root A
- Descendant – B is a descendant of A if A is the root of the subtree B is in.
  - If A is ancestor of B, B is descendant of A

- Path – If A is ancestor of B, path from A to B is the sequence where each element is the parent of the next in sequence (ex 4)

Binary Tree Properties

- Height – Number of branches between root and farthest leaf. (ex 7)
  - AKA 1 plus height of tallest subtree
  - Root only tree has height=0
  - Therefore empty tree has height = -1

Binary Tree Properties

Recursive definition of height pseudocode

```
if t is empty
    height(t)= -1
else
  height(t) =
    1+max[height(leftTree(t)),height(rightTree(t))]
```

Binary Tree Properties

- Height describes the whole tree

- Level partially describes an element's position
  - level(e) = # of branches between root and e (ex 7)

Binary Tree Properties

- Recursive definition of level pseudocode

```
if x is the root element,
    level(x) = 0
else
    level(x)=1+level(parent(x))
```

Binary Tree Properties

- Two-tree
  - binary tree that is empty or each nonleaf has two branches (ex 8)

Binary Tree t is a two-tree if:
t is empty
OR
both subtrees of t are empty or
both subtrees of t are nonempty two-trees

Binary Tree Properties

- Full tree – binary tree t is full if it is a two-tree with all leaves on same level (ex 9)

Binary tree t is full if t is empty
OR
t's left and right subtrees have the same height and are both full

Binary Tree Properties

- Number of elements n(t) in a full binary tree is proportional to height(t)

- $n(t)=2^{k+1}-1$, $k>=1$
  - k=height(t), n = # elements

```
if t is empty,
    n(t)=0
else
    n(t) = 1+n(leftTree(t))+n(rightTree(t))
```

Binary Tree Properties

- Complete – If tree t is full through to the next-to-lowest level and leaves are on the left.
- All full binary trees are complete
  - reverse not true (ex 10)

Binary Tree Properties

- Position – we can assign position numbers to elements in a complete binary tree

Root = 0
If element at position i has children,
    left child position = 2i+1
    right child position = 2i+2 (ex11)
    parent position = (i-1)/2   //int division

- We can use position to implement binary trees with arrays (element at position i stored at index i)

Binary Tree Theorem

For a binary tree t,

leaves(t)<= n(t) and leaves(t)=n(t)

IFF

t is empty or t only has one element

Binary Tree Theorem

For a nonempty binary tree t

1. leaves(t) <= [n(t)+1]/2.0 //float division (ex 12)
2. [n(t)+1]/2.0 <= $2^{height(t)}$
3. Equality holds in 1 IFF t is a two-tree (ex 13)
4. Equality holds in 2 IFF t is full (ex 13)

Binary Tree Theorem

Logic lesson – IFF implies a bi-conditional relationship.

Part 1 - leaves(t) <= [n(t)+1]/2.0

Part 3 - Equality holds in 1 IFF t is a two-tree

If t is a nonempty two-tree

THEN leaves(t) = [n(t)+1]/2.0

AND if leaves(t) = [n(t)+1]/2.0

THEN t must be a nonempty two-tree

Binary Tree Theorem

From part 4:
[n(t)+1]/2.0 = $2^{height(t)}$

height(t)=$\log_2$([n(t)+1]/2.0)
    =$\log_2$(n(t)+1)-1

Therefore height(t) for a full tree grows logarithmically
    Lists grow linearly!

Binary Tree Theorem

- Chain – binary tree where each nonleaf has one child (ex 14)
- Height(t) grows linearly
- If we maintain nonchain trees, then inserting and removing will be O(log(n))
- Arraylist/linkedlist inserting/removing at index is O(n)

External Path Length

For a nonempty binary tree t,

external path length of t, E(t), is the sum of the depths(levels) of the leaves in t (ex 15)

External Path Length Theorem

For a binary tree t with k>0 leaves,

E(t) >= (k/2) floor($\log_2$k)

Lower bound on external path length

Binary Tree Traversal

Traversal – algorithm which processes each element in binary tree t exactly once.

inOrder
postOrder
preOrder
breadthFirst

inOrder Traversal

- Left-Root-Right
  - First process left subtree, then root, then right subtree

```
inOrder(t){
    if(t is not empty){
        inOrder(leftTree(t));
        process root of t;
        inOrder(rightTree(t));
    }
} (ex 16)
```

inOrder Traversal

- For a binary search tree, inOrder processes elements in order (ex 17)
- Binary Search Tree (BST) – all elements in left subtree are less than the root, which is less than all elements in the right subtree. Also both subtrees are BST's

postOrder Traversal

- Left-Right-Root
  - First process left subtree, then right subtree, then root

```
postOrder(t){
    if(t is not empty){
        postOrder(leftTree(t));
        postOrder(rightTree(t));
        process root of t;
    }
} (ex 18)
```

postOrder Traversal

- For an expression tree, postOrder produces postFix notation (reverse polish notation)
- Expression tree – Each nonleaf is a binary operator with operands in left and right subtrees.

preOrder Traversal

- Root-Left-Right
  - First process root, then left subtree, then right subtree

```
preOrder(t){
    if(t is not empty){
        process root of t;
        preOrder(leftTree(t));
        preOrder(rightTree(t));
    }
} (ex 19)
```

preOrder Traversal

- For an expression tree, preOrder produces preFix notation (polish notation)
- preOrder is AKA depth-first search
  - goes all the way left (down) first, then right

breadthFirst Traversal (Level by Level)

- Root, then children of root left to right, then grandchildren of the root left to right, etc.
  (ex 20)
- Generate level by level a list of nonempty subtrees.
- Retrieve subtrees in the same order they were generated.
- What data structure to use?

breadthFirst Traversal (Level by Level)

```
//queue is a queue of binary trees, tree is a binary tree
breadthFirst(t){
    if (t is not empty){
        queue.enqueue(t);
        while(queue not empty){
            tree = queue.dequeue;
            process tree's root;
            if(leftTree(tree) is not empty){
                queue.enqueue(leftTree(tree));     }
            if(rightTree(tree) is not empty){
                queue.enqueue(rightTree(tree)); }
        }//end while
    }//end if
}//end breadthFirst          (ex 21)
```

Binary Tree Exercises (ex 22)

- What is the root element?
- What is n(t)?
- What is leaves(t)?
- What is height(t)?
- What is height(leftTree(t))?
- What is height(rightTree(t))?
- What is level of F?
- What is depth of C?

Binary Tree Exercises

- How many children does C have?
- What is the parent of F?
- What are the descendants of B?
- What are the ancestors of F?
- Output of inOrder transversal?
- Output of postOrder transversal?
- Output of preOrder transversal?
- Output of breadthFirst transversal?

Binary Tree Exercises

- Construct a 2-tree that is not complete
- Construct a complete tree that is not a 2-tree
- Construct a complete 2-tree that is not full
- How many leaves in a 2-tree with 17 elements
- How many leaves in a 2-tree with 731 elements
- Why must a 2-tree always have an odd number of elements?

Build SimpleBinaryTree.java

HW4 Assign