# Tara Walton

# CSC 232: Data Structures

**Lab8: Trees**

**Due: 11/19/15 @ 2359**

**Objectives**

* Learn the properties of the tree data structure
* Develop critical thinking skills to improve problem solving

1. Answer the questions by referring to the following figure (also 7.3 on p.270):
   1. Which node is the root?

**/user/rt/courses/**

* 1. What are all the internal nodes?

**Cs016/, cs252/, (cs016/)grades, homeworks/, programs/, projects/, (cs252/)grades, papers/, demos/**

* 1. How many descendants does node cs016/ have?

**9 (3 children, 6 grandchildren)**

* 1. How many ancestors does node cs016/ have?

**1 (root)**

* 1. What are the siblings of node homeworks/?

**2**

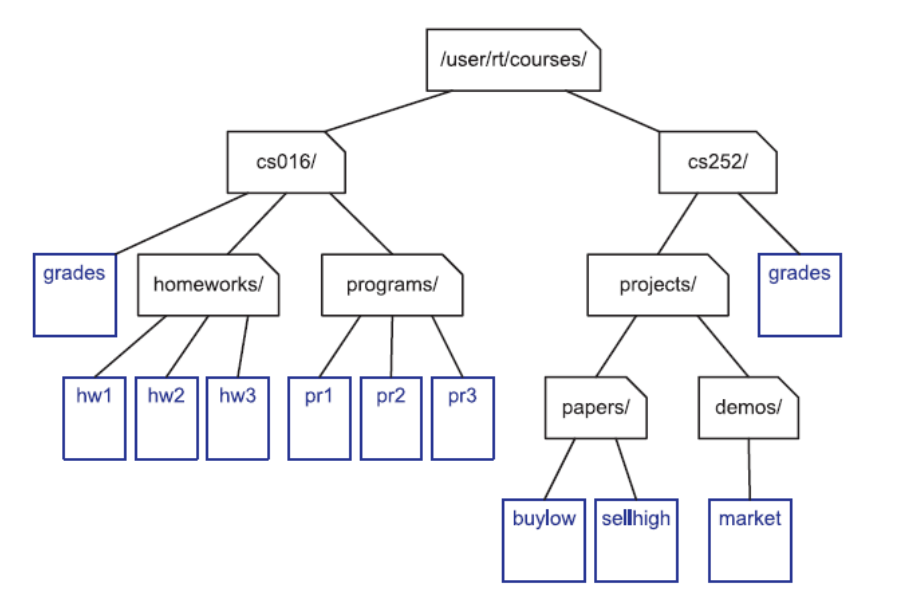
* 1. Which nodes are in the subtree rooted at node projects/?

**Papers/, buylow, sellhigh, demos/, market**

* 1. What is the depth of node papers/?

**2**

* 1. What is the height of the tree?

**4**

1. Review algorithm height2(*T*, *p*) (below or see p.277). What is the running time when:
   1. Called on the root? **O(cR), where cR is the number of children of the root**
   2. Called on a subtree rooted at node A? **O(cA), where cA is the number of children to node A**

**Algorithm** height2(*T*, *p*):

**if** *p*.isExternal() **then**

**return** 0

**else**

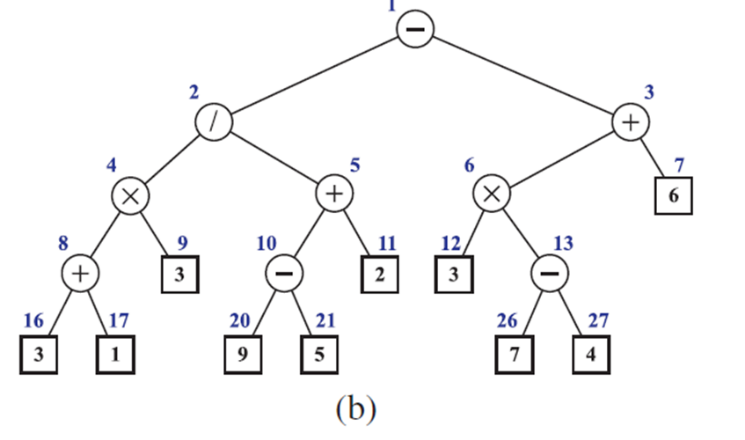
*h* = 0

**for** each *q* ∈ *p*.children() **do**

*h* = max(*h*, height2(*T*,*q*))

**return** 1+*h*

1. Refer to the following tree (also Figure 7.16(b) p.296). List the equation generated and solve (if possible) by traversing the tree in:
   1. Preorder

**(-(/(\*(+31)3)(+((-95)2))(+(\*3(-74))6)))**

**Sol’n: -13**

* 1. Postorder

**((((31+)3\*)((95-)2+)/)((3(74-)\*)6+)-)**

**Sol’n: -13**

* 1. Inorder

**(((3+1)\*3)/((9-5)+2))-((3\*(7-4))+6)**

**Sol’n: -13**

Hints: polish notation and reverse polish notation.

1. Given that tree *T* is a general ordered tree with nodes > 1.
   1. Can a preorder traversal of *T* visit the nodes in exactly the same order as a postorder traversal? If yes, give an example; if no, justify.

**No, preorder can only be the same as postorder if and only if there is only one node in the tree.**

* 1. Can a preorder traversal of *T* visit the nodes exactly in reverse order of a postorder traversal? If yes, give an example; if no, justify.

**Yes, but only if a tree consists of a root and two children.**

1. What is the running time of parenPrint(*T*, *T*.root())?  **O(n), where n is the number of nodes**

void parenPrint(const Tree& T, const Position& p) {

cout << \*p; // print node’s element

if (!p.isExternal()) {

PositionList ch = p.children(); // list of children

cout << "( "; // open

for (Iterator q = ch.begin(); q != ch.end(); ++q) {

if (q != ch.begin())

cout << " "; // print separator

parenPrint(T, \*q); } // visit the next child

cout << " )"; // close

}}

1. Regarding Proposition 7.10 (also on p.287), justify the following:
   1. What is the maximum number of external nodes for a binary tree with height h?
   2. *J* is a proper binary tree with *h* = height and nodes = *n.* What values of *n* and *h* make the following equation an equality? .

**where c is a scaling multiplier.**

Proposition 7.10: Let *T* be a nonempty binary tree, and let and *h* denote the number of nodes, number of external nodes, number of internal nodes, and height of *T* , respectively. Then *T* has the following properties:

Also, if *T* is proper, then it has the following properties:

1. Refer back to the tree in problem #1. List the output of executing the algorithm postorderPrint(*T*,*T*.root()) on it.

void postorderPrint(const Tree& T, const Position& p) {

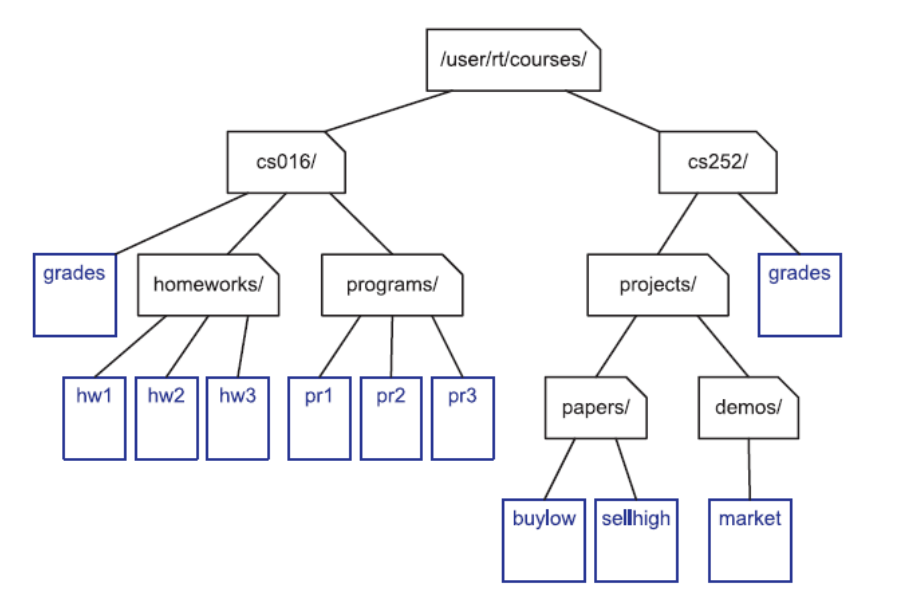
PositionList ch = p.children(); // list of children

for (Iterator q = ch.begin(); q != ch.end(); ++q) {

postorderPrint(T, \*q);

cout << " "; }

cout << \*p; } // print element



**OUTPUT::**

**grades hw1 hw2 hw3 pr1 pr2 pr3 homeworks/ programs/ cs016/ buylow sellhigh market papers/ demos/ projects/ grades cs252/ /user/rt/courses/**

**Extra Credit**

Justify Proposition 7.4 on p.276, that the height of a tree is equal to the maximum depth of its external nodes.

**To justify we must go back to the definition of the height of a tree. The height of a given node *p* of a tree *T* is defined as being the number of edges on the longest path from *p* to a leaf, ie, the external node. This means that a node *p* will have a depth of 0 and a height h while its furthest descendant node with trace the path in reverse and will have a height of 0 and a depth of h.**