TREES

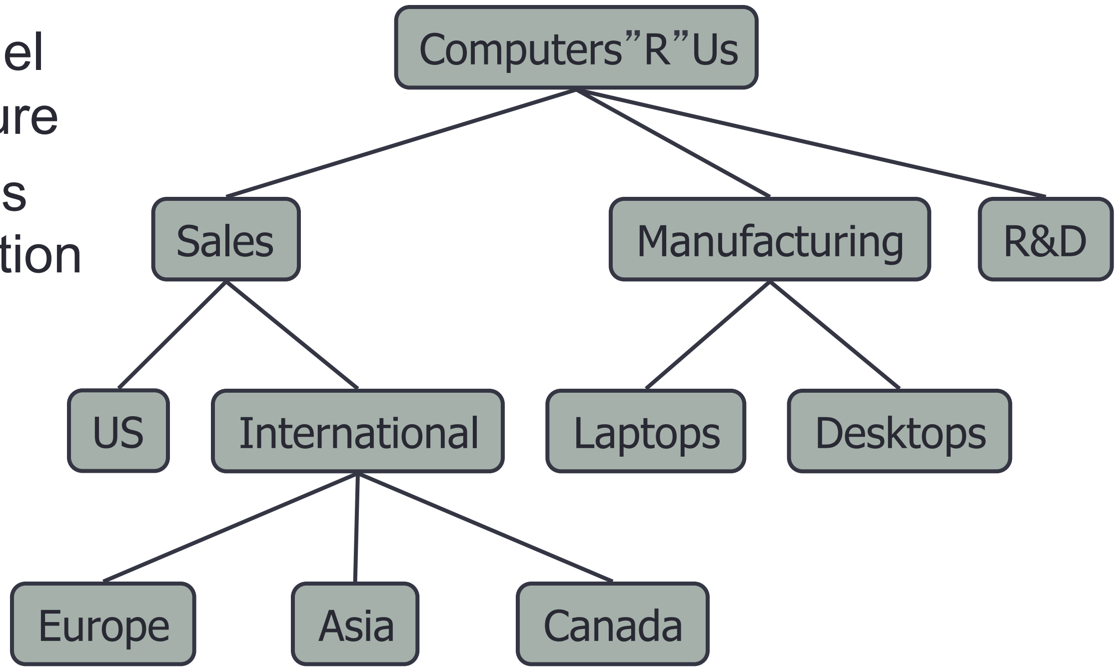
EECS 2011

Trees

- Linear access time of linked lists is prohibitive
  - Does there exist any simple data structure for which the running time of most operations (search, insert, delete) is O(log N)?
- Trees
  - Basic concepts
  - Tree traversal
  - Binary trees
  - Binary search trees
  - AVL trees

General Trees (8.1)

- In computer science, a tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Applications:
  - Organization charts
  - File systems
  - Programming environments



Example: File Systems

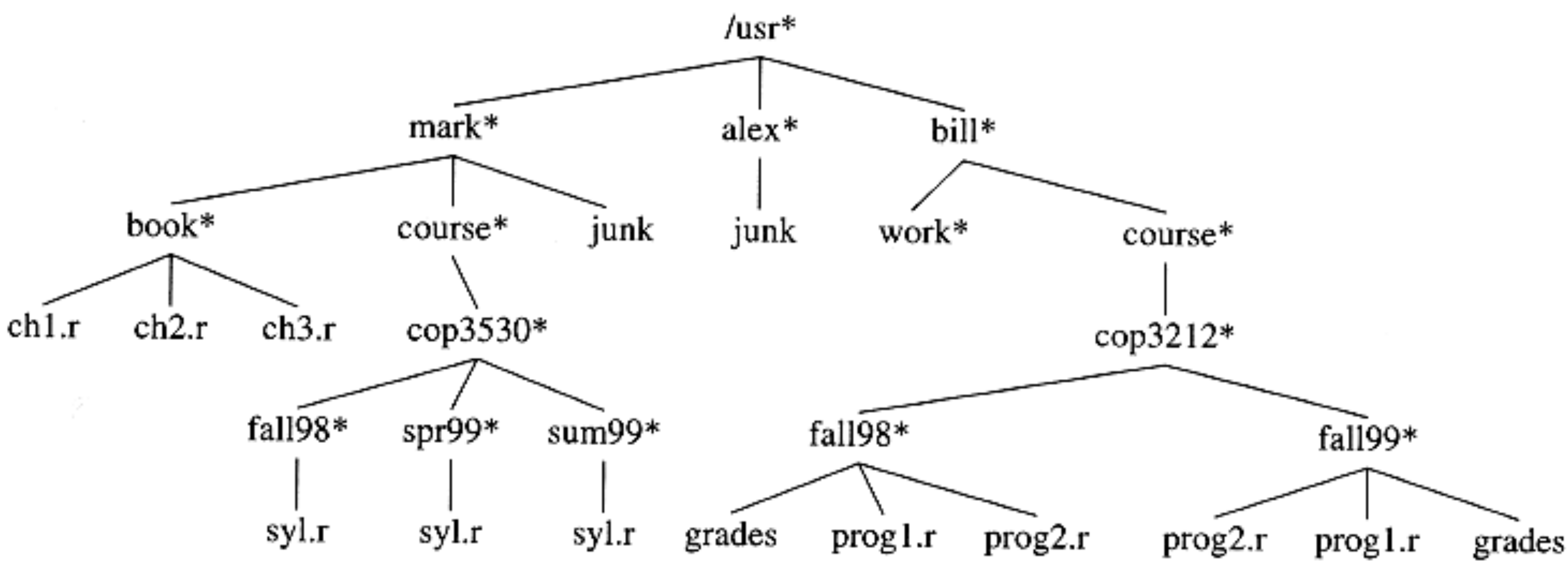


Figure 4.5 UNIX directory

Example: Expression Trees

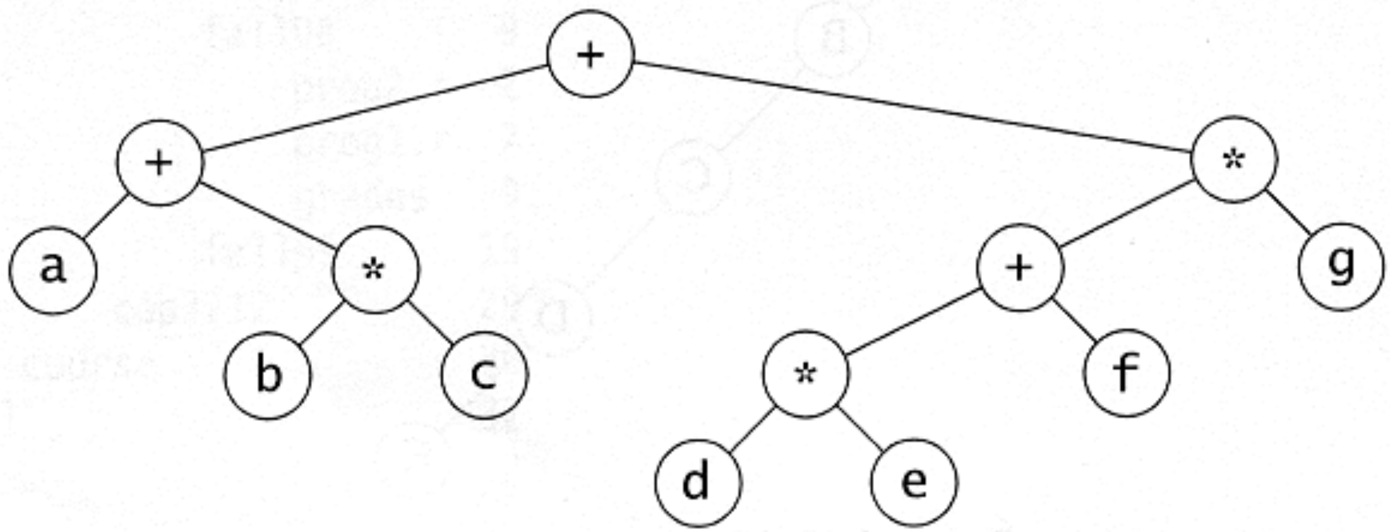


Figure 4.14 Expression tree for (a + b * c) + ((d * e + f) * g)

- Leaves are operands (constants or variables)
- The internal nodes contain operators

Recursive Definition

- A tree is a collection of nodes.
  - The collection can be empty.
  - Otherwise, a tree consists of a distinguished node *r* (the *root*), and zero or more nonempty *subtrees* T_1, T_2, . . . , T_k, each of whose roots is connected by a directed *edge* from *r.*

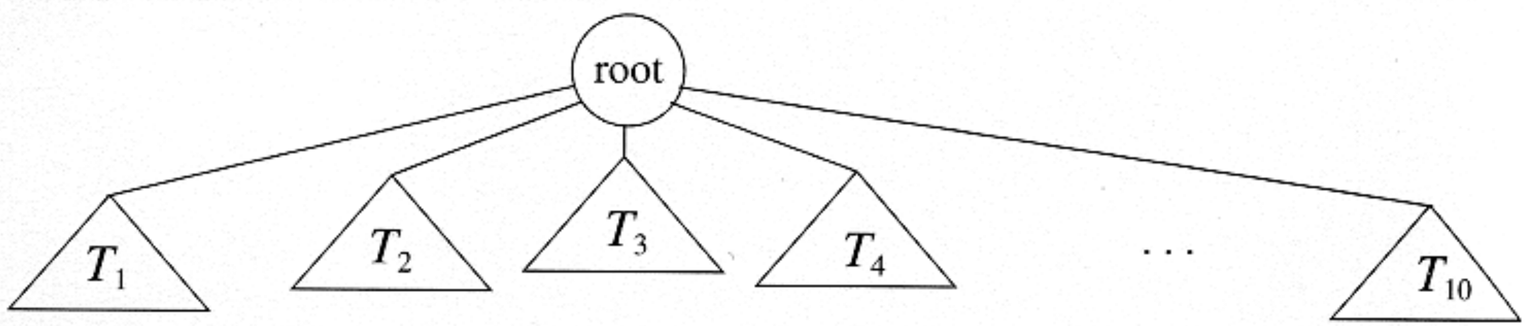


Figure 4.1 Generic tree

Applying the Recursive Definition

```
void operation ( T ) {
  if ( T is not empty )
    for every subtree Ti of T
      operation( Ti )
}
```

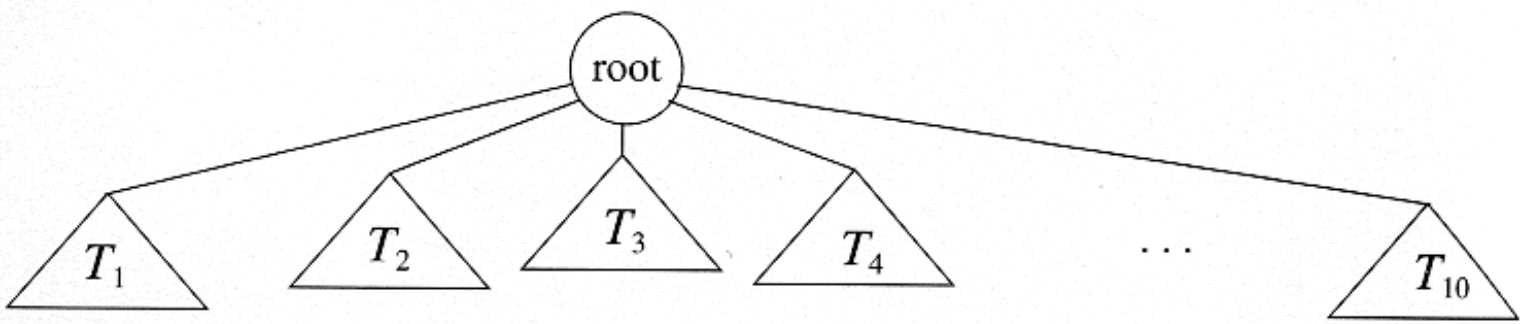
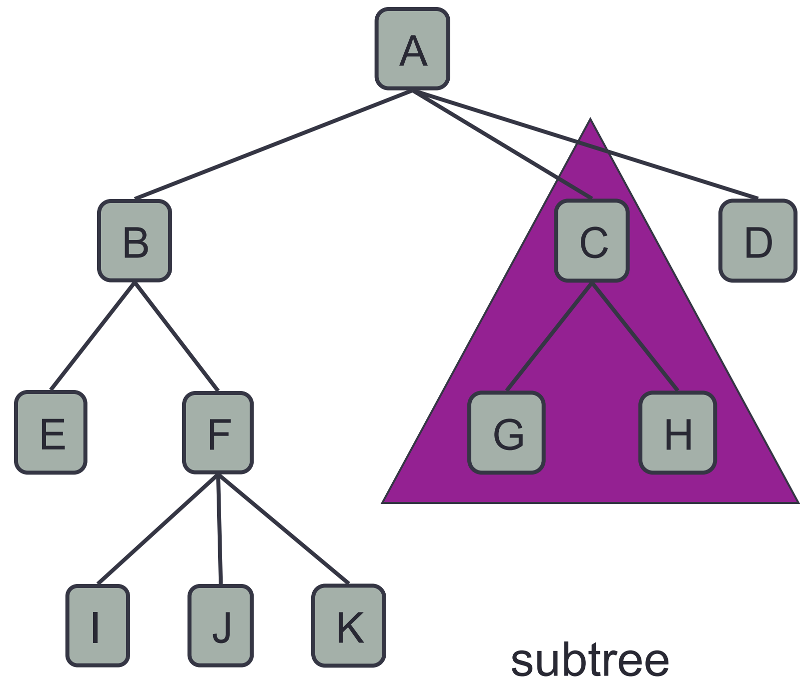


Figure 4.1 Generic tree

Tree Terminology

- Root: node without parent (A)
- Internal node: node with at least one child (A, B, C, F)
- External node (a.k.a. leaf): node without children (E, I, J, K, G, H, D)
- Ancestors of a node: parent, grandparent, grand-grandparent, etc.
- Depth of a node: number of ancestors
- Height of a tree: maximum depth of any node (3 in the example)
- Descendant of a node: child, grandchild, grand-grandchild, etc.

- Subtree: tree consisting of a node and its descendants



Tree Terminology (2)

- *Siblings*: nodes having the same parent
- *Path*: a sequence of edges
- *Length of path*: number of edges on the path

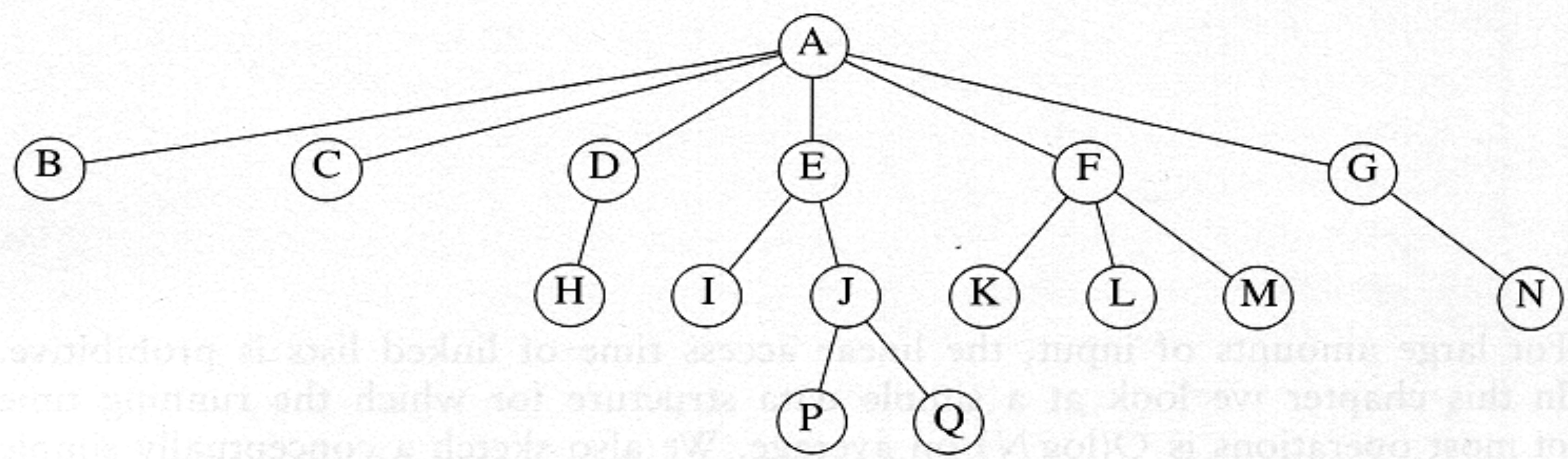


Figure 4.2 A tree

Tree Terminology (3)

- *Height* of a node
  - length of the longest path from that node to a leaf
  - all leaves are at height 0
- The height of a tree = the height of the root
  = maximum depth of any node

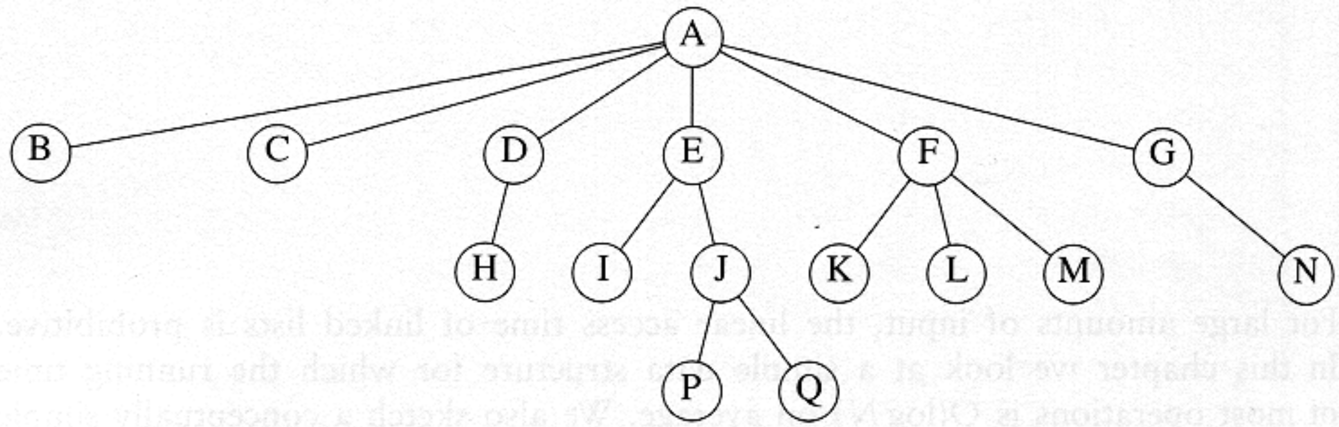


Figure 4.2 A tree

Tree ADT

- We use positions to abstract nodes (position ≡ node)
- Accessor methods:
  - position root()
  - position parent(p)
  - Iterable children(p)
  - Integer numChildren(p)
- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Generic methods:
  - integer size()
  - boolean isEmpty()
  - Iterator iterator(): returns an iterator of all elements in the tree.
  - Iterable positions(): returns an iterable collection of all positions of the tree.
- Additional update methods may be defined by data structures implementing the Tree ADT

Java Interface

Methods for a Tree interface:

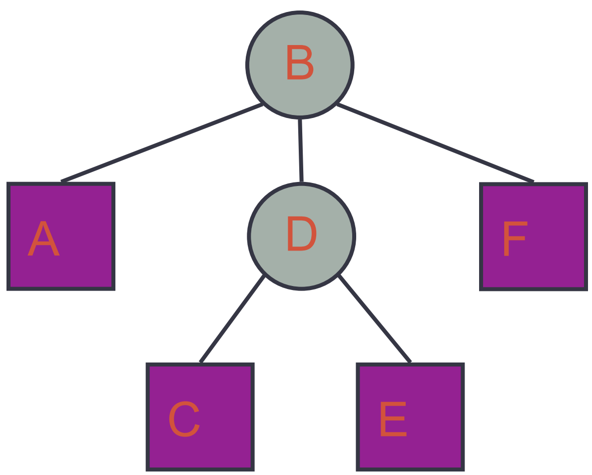
```java
/** An interface for a tree where nodes can have an arbitrary number of children. */
public interface Tree<E> extends Iterable<E> {
  Position<E> root();
  Position<E> parent(Position<E> p) throws IllegalArgumentException;
  Iterable<Position<E>> children(Position<E> p)
                                    throws IllegalArgumentException;
  int numChildren(Position<E> p) throws IllegalArgumentException;
  boolean isInternal(Position<E> p) throws IllegalArgumentException;
  boolean isExternal(Position<E> p) throws IllegalArgumentException;
  boolean isRoot(Position<E> p) throws IllegalArgumentException;
  int size();
  boolean isEmpty();
  Iterator<E> iterator();
  Iterable<Position<E>> positions();
}
```

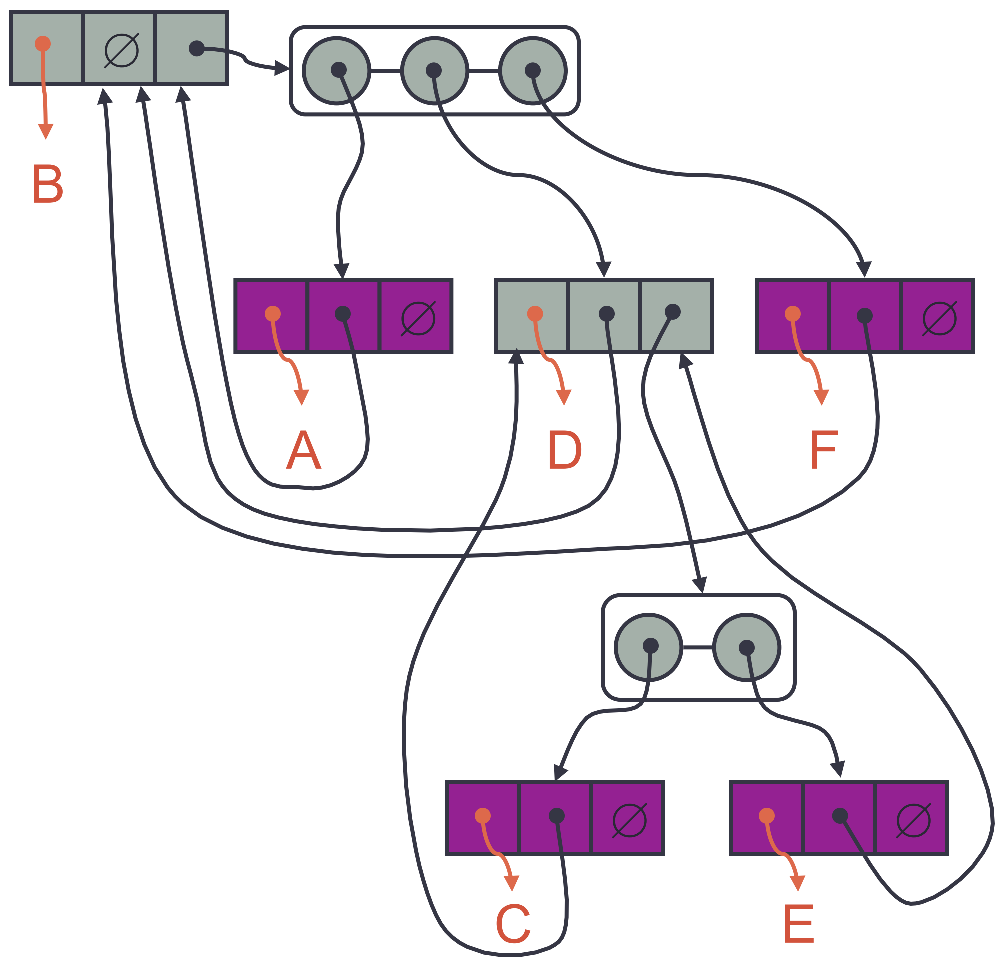
Implementing Trees

- Arrays ?
- Linked structures (pointers) ?

Linked Structure for Trees (8.3.3)

- A node is represented by an object storing
  - Element
  - Parent node
  - Sequence of children nodes





Tree Traversal Algorithms (8.4)

- Preorder
  - Visit *v* first
  - Then visit the descendants of *v*

- Postorder
  - Visit the descendants of *v* first
  - Then visit *v* last

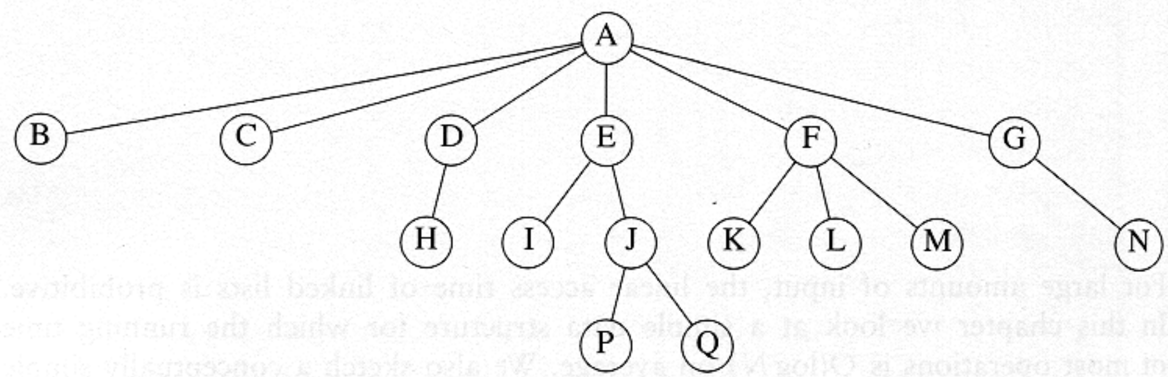
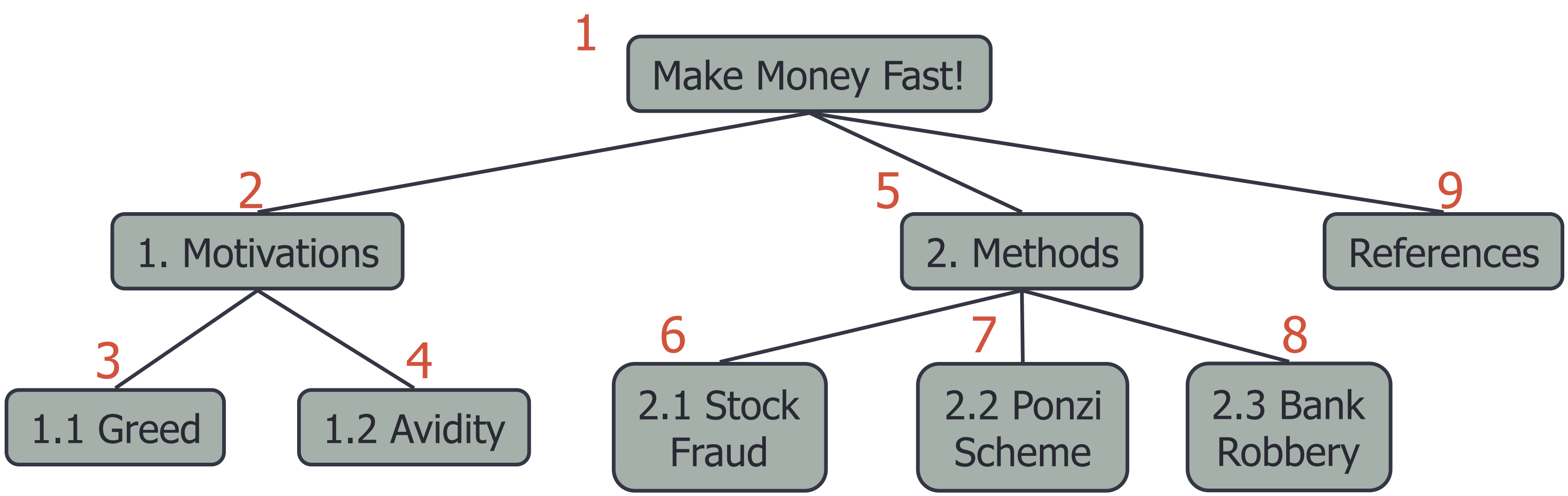


Figure 4.2 A tree

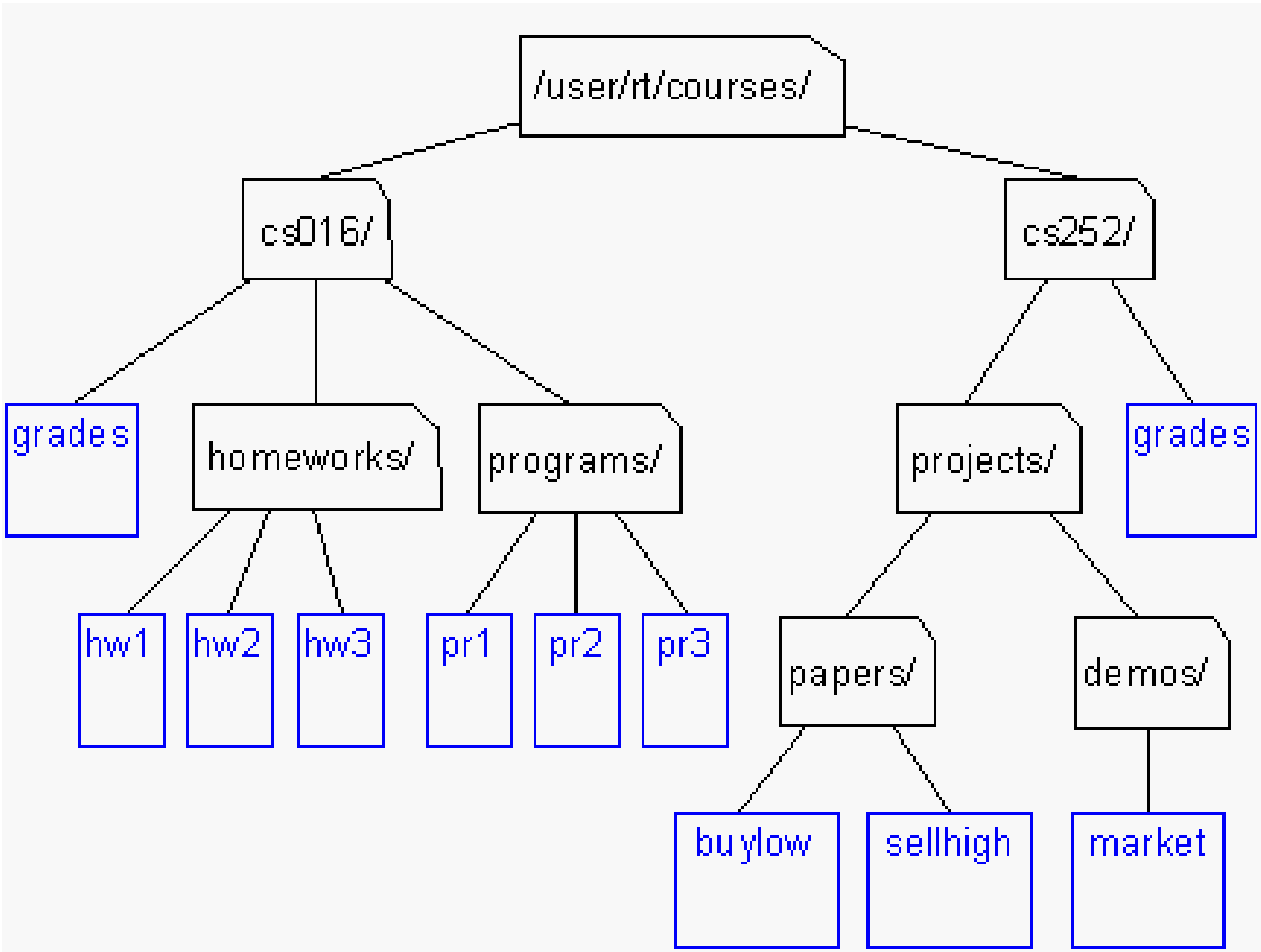
Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

Algorithm *preOrder*(*v*)
    visit(*v*)
    for each child ***w*** of ***v***
        preOrder (*w*)



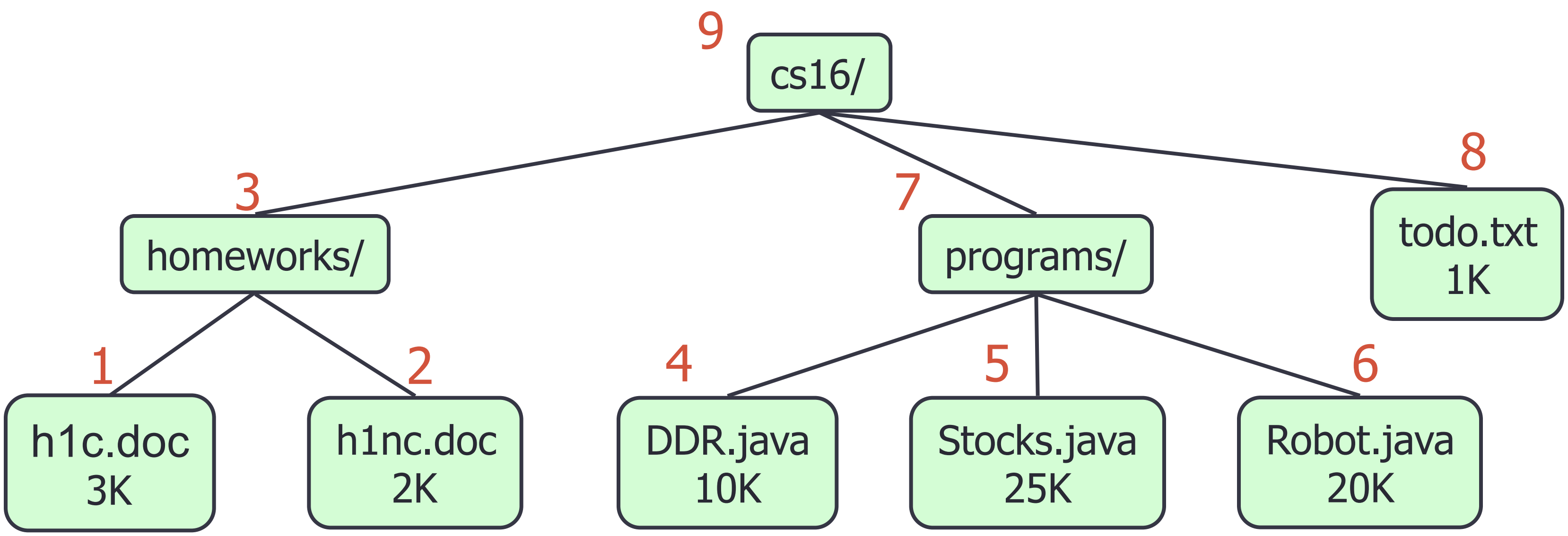
An Example


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

Postorder Traversal

- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

Algorithm *postOrder*(*v*)
    for each child ***w*** of ***v***
        postOrder (*w*)
    visit(*v*)



Applications

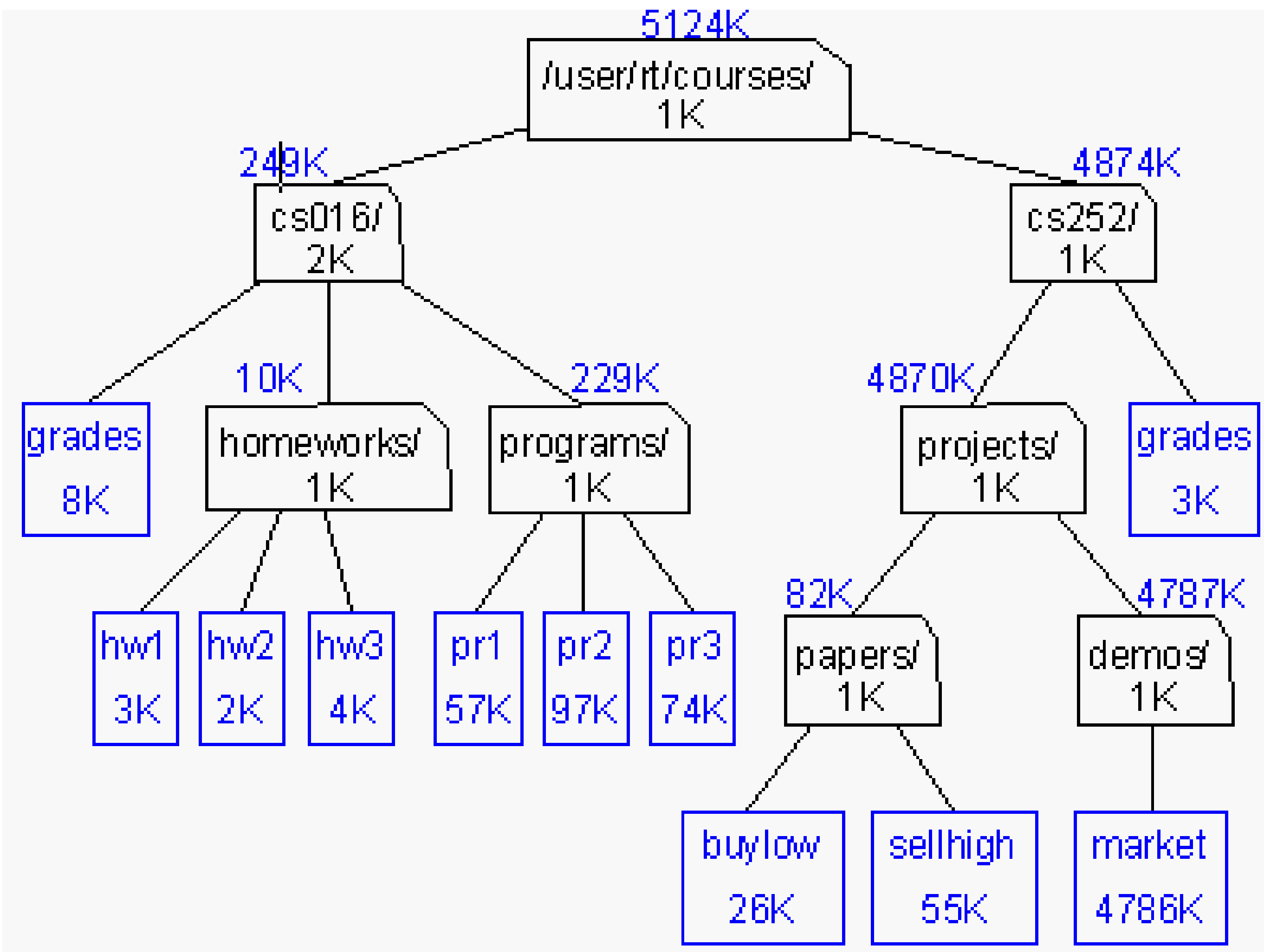
- Either preorder traversal or postorder traversal can be used when the order of computation is not important.

  Example: printing the contents of a tree (in any order)
- *Preorder* traversal is required when we must perform a computation for each node ***before*** performing any computations for its descendents.

  Example: Printing the headings of chapters, sections, sub-sections of a book.
- *Postorder* traversal is needed when the computation for a node *v* requires the computations for *v*' s children to be done first.

  Example: Given a file system, compute the disk space used by a directory.

Example: Computing Disk Space



Example: UNIX Directory Traversal

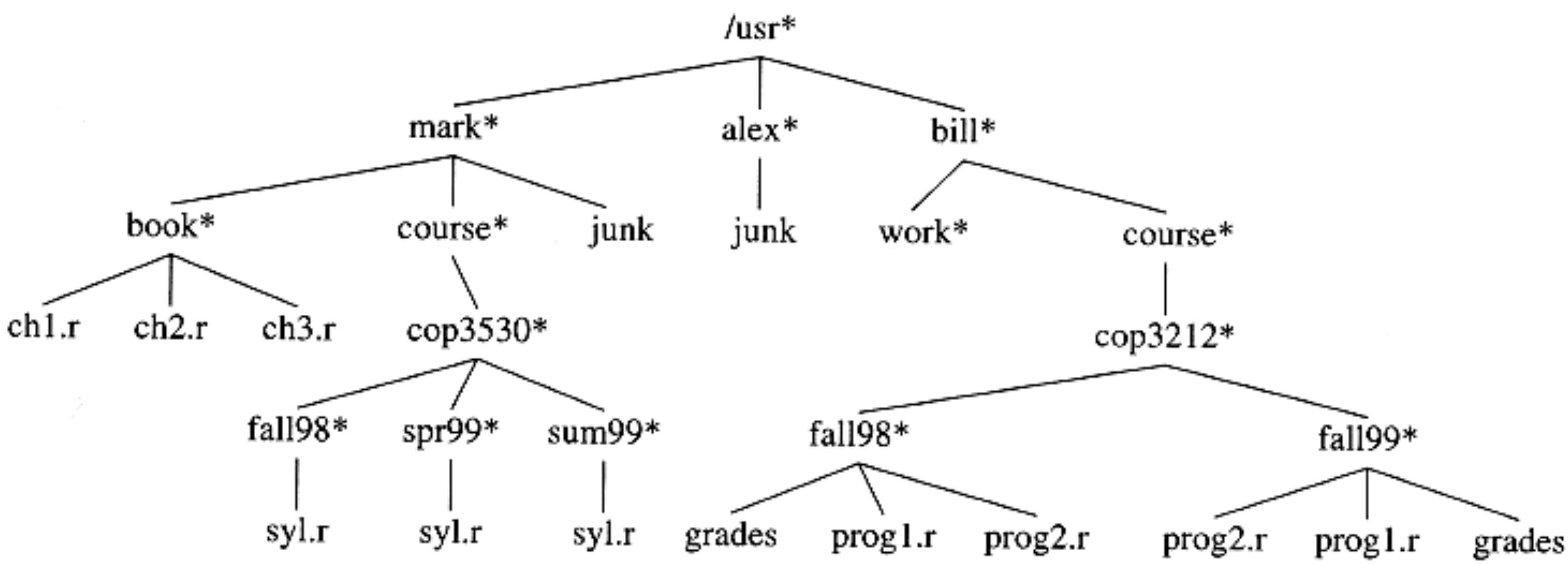


Figure 4.5 UNIX directory

Example: Unix Directory Traversal

Preorder

```
/usr
    mark
        book
            ch1.r
            ch2.r
            ch3.r
        course
            cop3530
                fall98
                    syl.r
                spr99
                    syl.r
                sum99
                    syl.r
        junk
    alex
        junk
    bill
        work
        course
            cop3212
                fall98
                    grades
                    prog1.r
                    prog2.r
                fall99
                    prog2.r
                    prog1.r
                    grades
```

Postorder

```
            ch1.r          3
            ch2.r          2
            ch3.r          4
        book              10
                syl.r      1
            fall98         2
                syl.r      5
            spr99          6
                syl.r      2
            sum99          3
        cop3530           12
    course                13
    junk                   6
mark                      30
    junk                   8
alex                       9
    work                   1
                grades     3
                prog1.r    4
                prog2.r    1
            fall98         9
                prog2.r    2
                prog1.r    7
                grades     9
            fall99        19
        cop3212           29
    course                30
bill                      32
/usr                      72
```

Computing Depth and Height

Depth

- *Depth* of node *v*: number of ancestors of *v*.
- Recursive definition:
  - If *v* is the root, then depth of *v* is 0.
  - Otherwise, depth of *v* is 1 plus the depth of *v*'s parent.

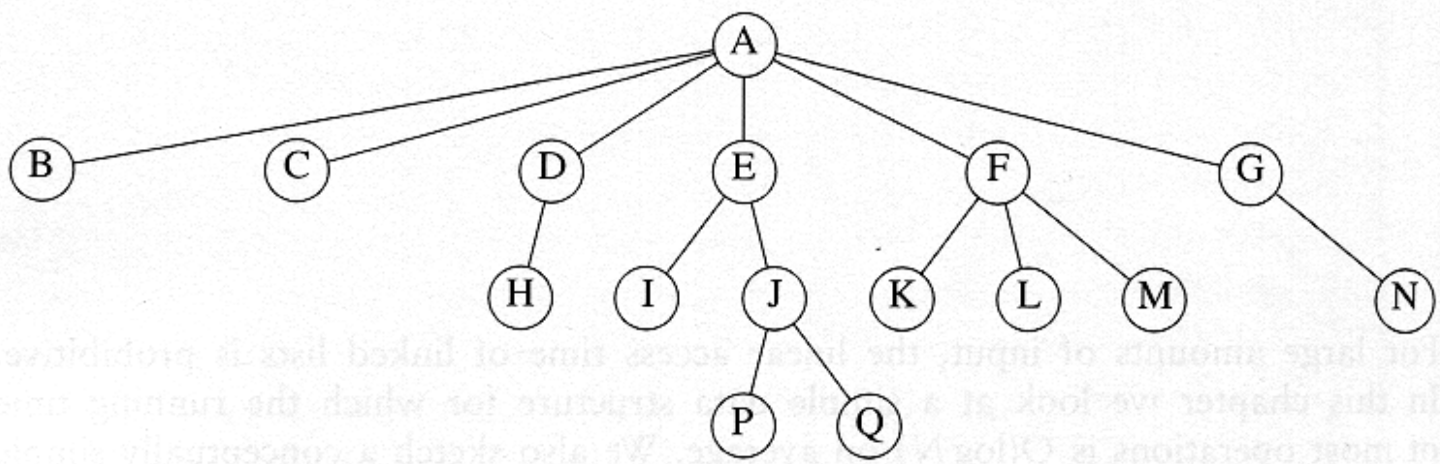


Figure 4.2 A tree

Algorithm *depth*

```
Algorithm depth( T, v ) {
   if ( isRoot( v ) )
      return 0;
   return ( 1 + depth( T, parent( v ) ) );
}
```

Running time = $d_v + 1$

Worst case: $O(n)$

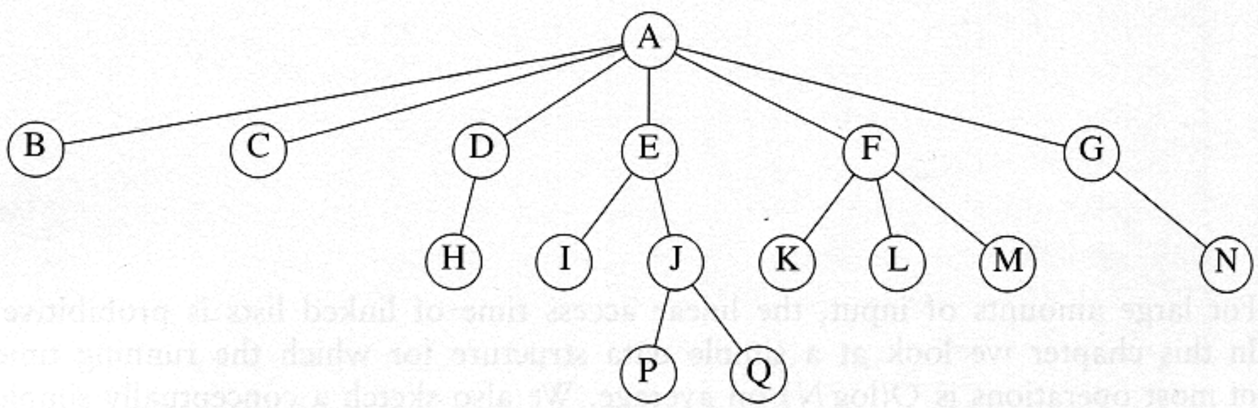


Figure 4.2 A tree

Computing Height of a Tree – Method 1

- The height of a tree = maximum depth of any node

```
Algorithm Tree_Height( T ) {
  h = 0;
  for every node v in T
     if( isExternal( v ) )
        h = max( h, depth( T, v ) );
}
```

Running time: $O(n) + \Sigma_v(d_v + 1)$ for all external nodes v

$\Sigma_v d_v = O(n^2)$ in the worst case (C-8.31) $\Rightarrow$ not efficient

Height

- *Height* of a node
  - length of the longest path from that node to a leaf
  - all leaves are at height 0
- The height of a tree = the height of the root
  = maximum depth of any node

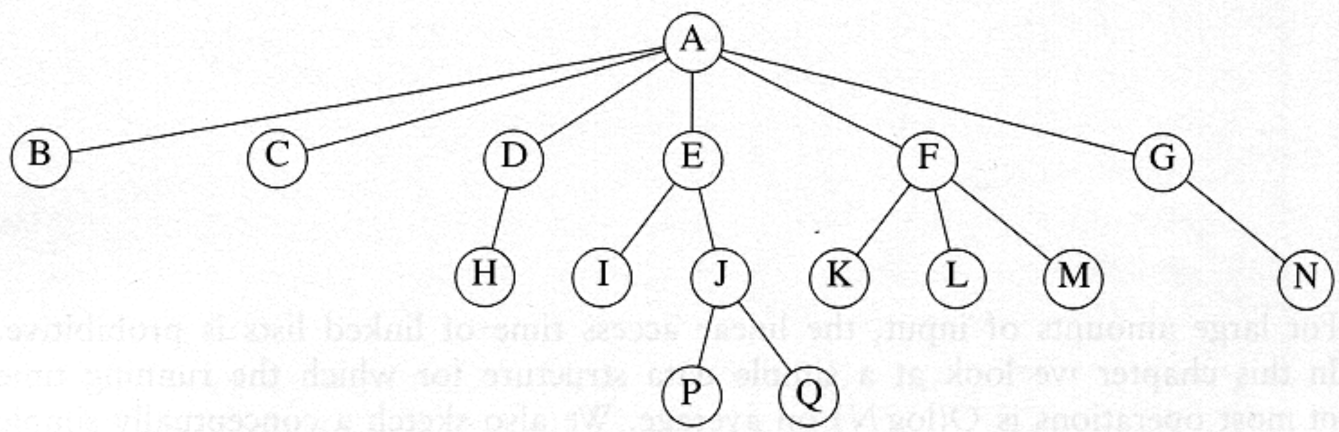


Figure 4.2 A tree

Recursive Definition of Height of a Node

- The height of a node v in a tree T is defined as follows:
  - If v is a leaf node, then height of v is 0.
  - Otherwise, height of v is 1 plus the maximum height of a child of v.

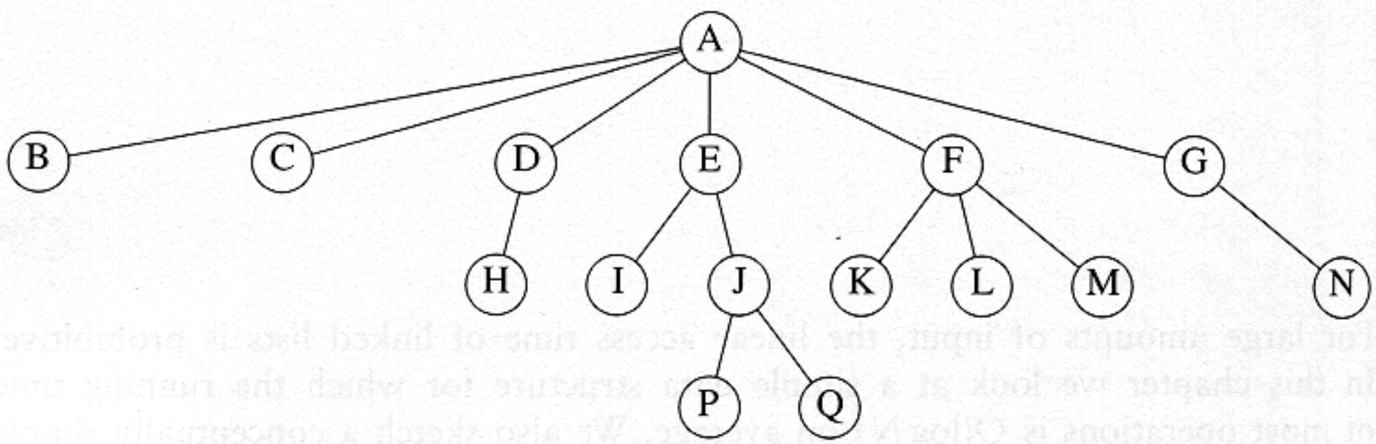


Figure 4.2 A tree

Algorithm *height*

```
Algorithm height( T, v ) {
   if ( isExternal( v ) )
      return 0;
   h = 0;
   for every child w of v
      h = max( h, height( T, w ) );
   return( 1 + h );
}
```

- Running time: $\Sigma_u(c_u + 1)$ for every node u in sub-tree rooted at v
- We visit each node exactly once.

Computing Height of a Tree – Method 2

- Height of the tree:

```
H = height( T, root );
```

- Running time: $\Sigma_u(c_u + 1)$ for every node *u* in the tree
- We visit each node exactly once.
- $O(n)$

Next lecture …

- Binary Trees (8.2)