Positions, Iterators, Tree Structures and Tree Traversals

Readings - Chapter 7.3, 7.4 and 8

Positional Lists

- A position acts as a marker or token within the broader positional list.
- A position p is unaffected by changes elsewhere in a list; the only way in which a position becomes invalid is if an explicit command is issued to delete it.
- A position instance is a simple object, supporting only the following method:
  - P.getElement( ): Return the element stored at position p.
- To provide for a general abstraction of a sequence of elements with the ability to identify the location of an element, we define a **positional list** ADT.

Positional List ADT

- Accessor methods:

first( ): Returns the position of the first element of L (or null if empty).

last( ): Returns the position of the last element of L (or null if empty).

before(p): Returns the position of L immediately before position p (or null if p is the first position).

after(p): Returns the position of L immediately after position p (or null if p is the last position).

isEmpty( ): Returns true if list L does not contain any elements.

size( ): Returns the number of elements in list L.

Positional List ADT (2)

- Update methods:

addFirst(e): Inserts a new element e at the front of the list, returning the position of the new element.

addLast(e): Inserts a new element e at the back of the list, returning the position of the new element.

addBefore(p, e): Inserts a new element e in the list, just before position p, returning the position of the new element.

addAfter(p, e): Inserts a new element e in the list, just after position p, returning the position of the new element.

set(p, e): Replaces the element at position p with element e, returning the element formerly at position p.

remove(p): Removes and returns the element at position p in the list, invalidating the position.

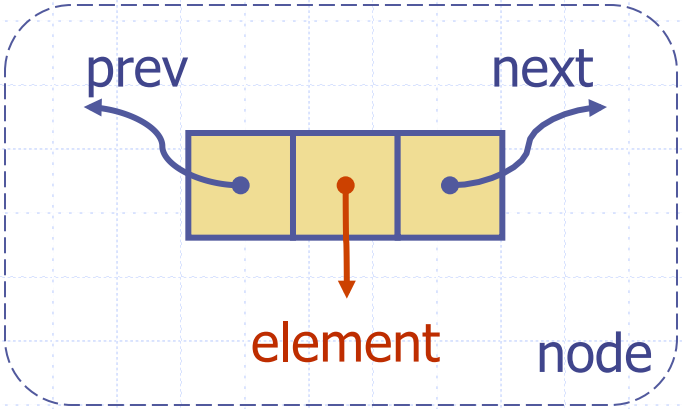
Example

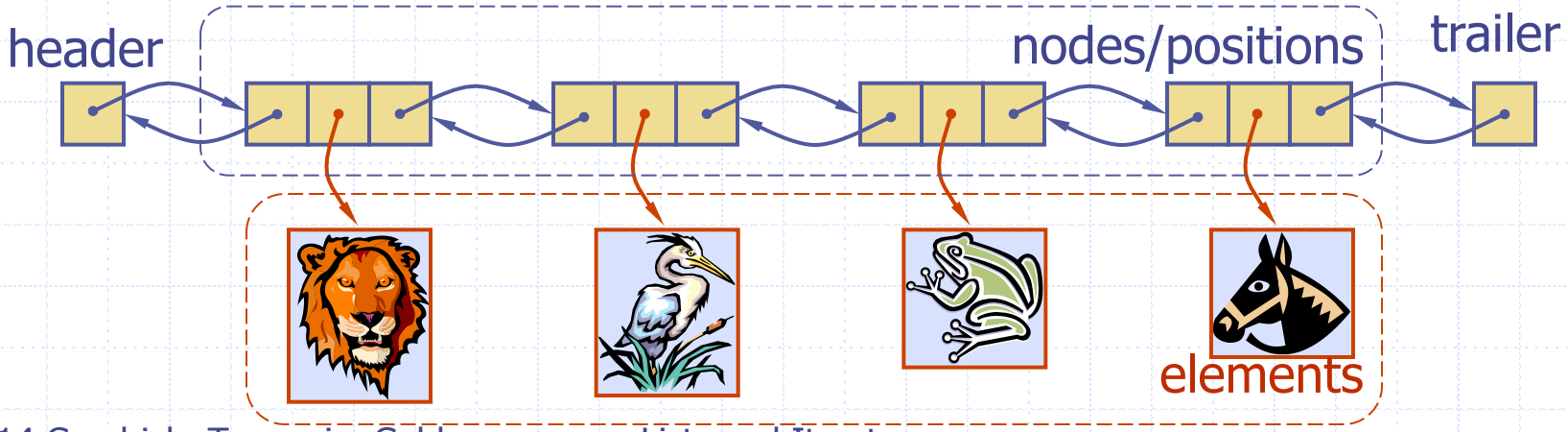
- A sequence of Positional List operations:

Method	Return Value	List Contents
addLast(8)	p	(8_p)
first( )	p	(8_p)
addAfter(p, 5)	q	$(8_p, 5_q)$
before(q)	p	$(8_p, 5_q)$
addBefore(q, 3)	r	$(8_p, 3_r, 5_q)$
r.getElement( )	3	$(8_p, 3_r, 5_q)$
after(p)	r	$(8_p, 3_r, 5_q)$
before(p)	null	$(8_p, 3_r, 5_q)$
addFirst(9)	s	$(9_s, 8_p, 3_r, 5_q)$
remove(last( ))	5	$(9_s, 8_p, 3_r)$
set(p, 7)	8	$(9_s, 7_p, 3_r)$
remove(q)	"error"	$(9_s, 7_p, 3_r)$

Positional List Implementation

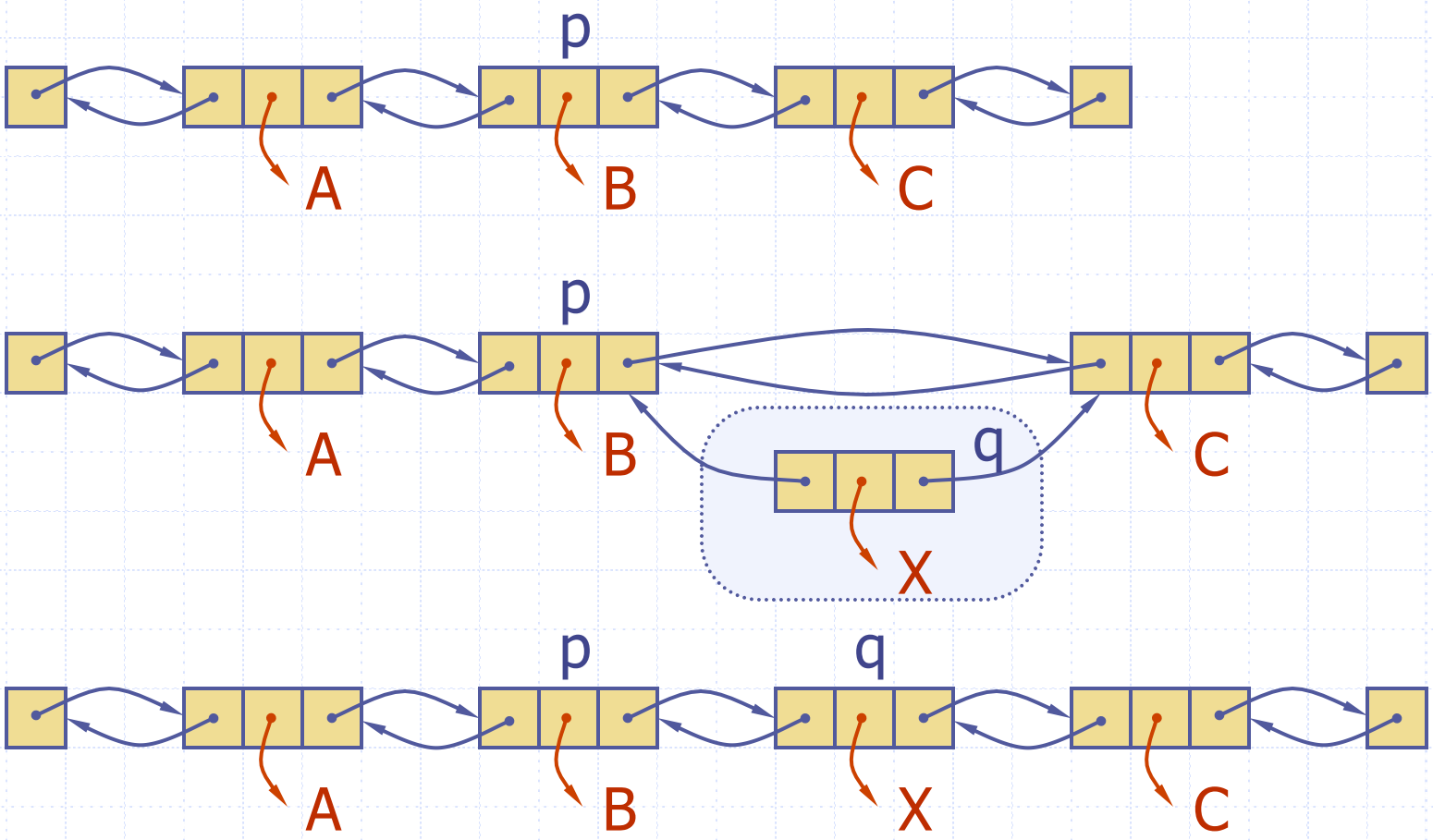
- The most natural way to implement a positional list is with a doubly-linked list.





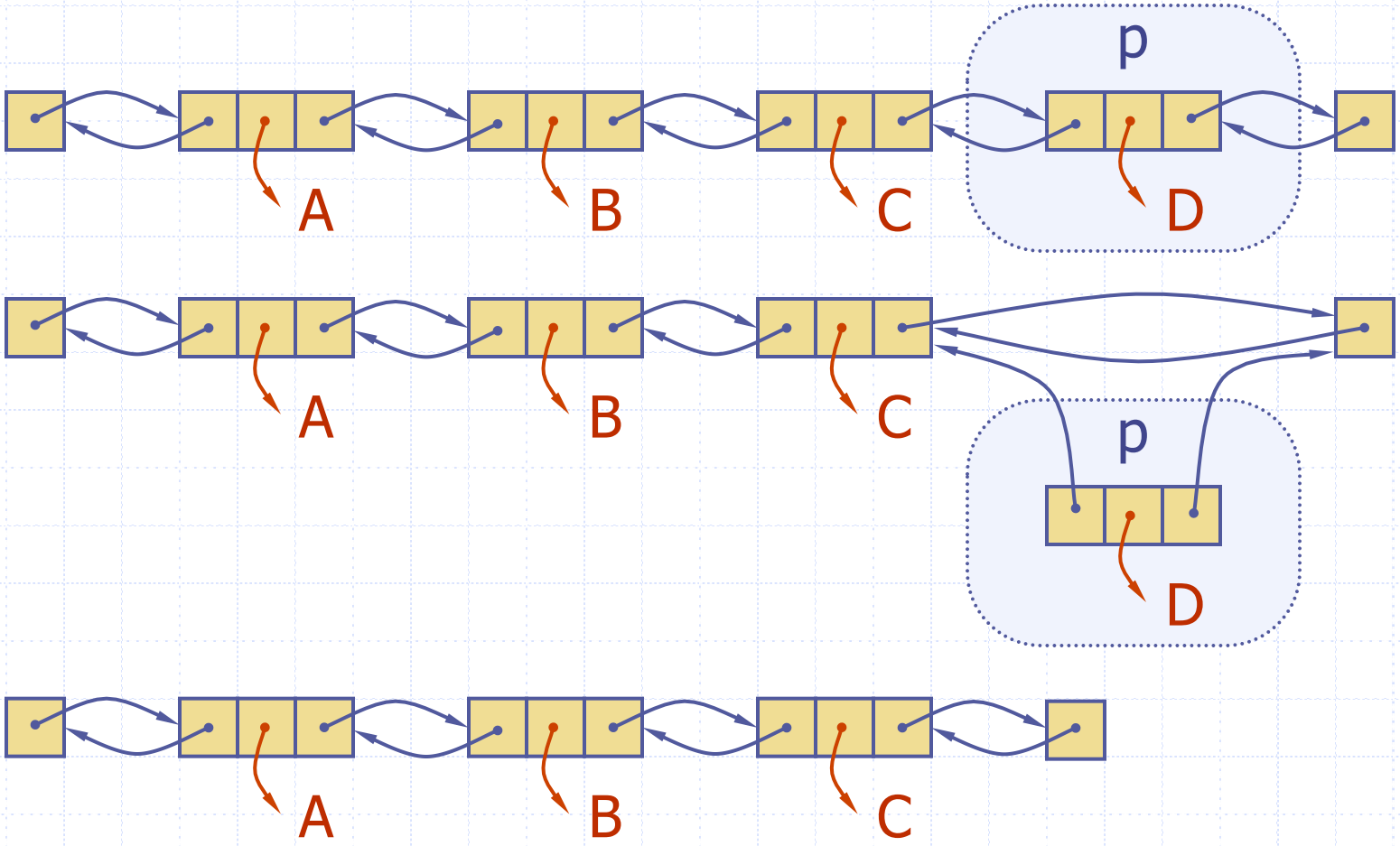
Insertion

- Insert a new node, q, between p and its successor.



Deletion

- Remove a node, p, from a doubly-linked list.



Iterators

- An iterator is a software design pattern that abstracts the process of scanning through a sequence of elements, one element at a time.

hasNext( ):	Returns true if there is at least one additional element in the sequence, and false otherwise.
next( ):	Returns the next element in the sequence.

The Iterable Interface

- Java defines a parameterized interface, named Iterable, that includes the following single method:
  - iterator( ): Returns an iterator of the elements in the collection.
- An instance of a typical collection class in Java, such as an ArrayList, is iterable (but not itself an iterator); it produces an iterator for its collection as the return value of the iterator( ) method.
- Each call to iterator( ) returns a new iterator instance, thereby allowing multiple (even simultaneous) traversals of a collection.

The for-each Loop

- Java's Iterable class also plays a fundamental role in support of the "for-each" loop syntax:

```
for (ElementType variable : collection) {
  loopBody                                   // may refer to "variable"
}
```

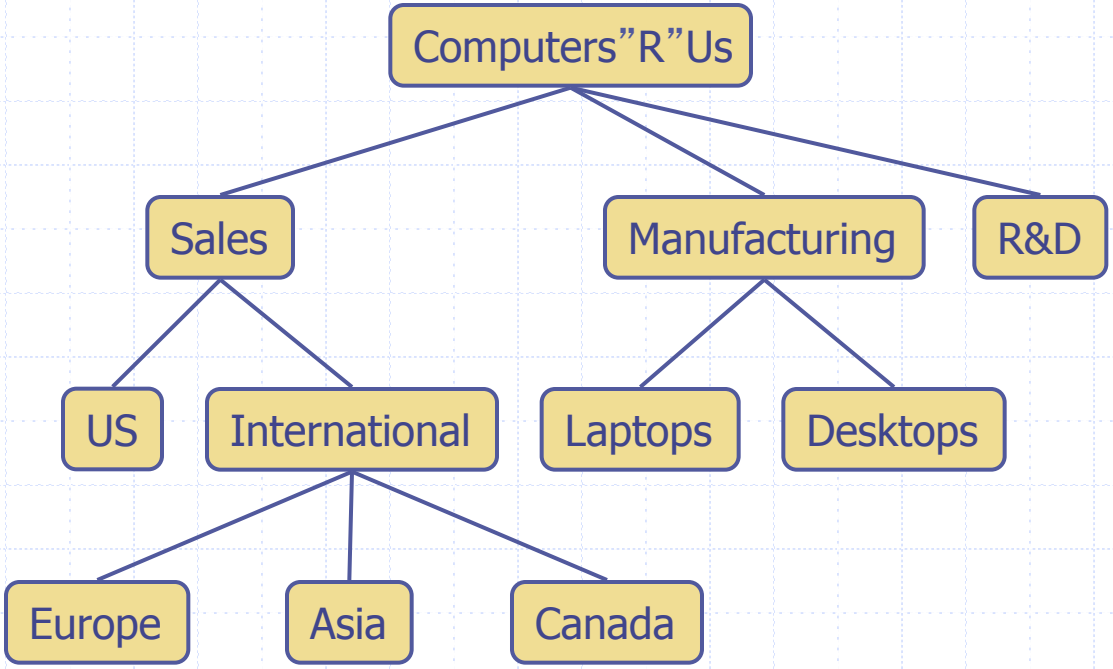
```
Iterator<ElementType> iter = collection.iterator( );
while (iter.hasNext( )) {
  ElementType variable = iter.next( );
  loopBody                                   // may refer to "variable"
}
```

Trees

- Abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation

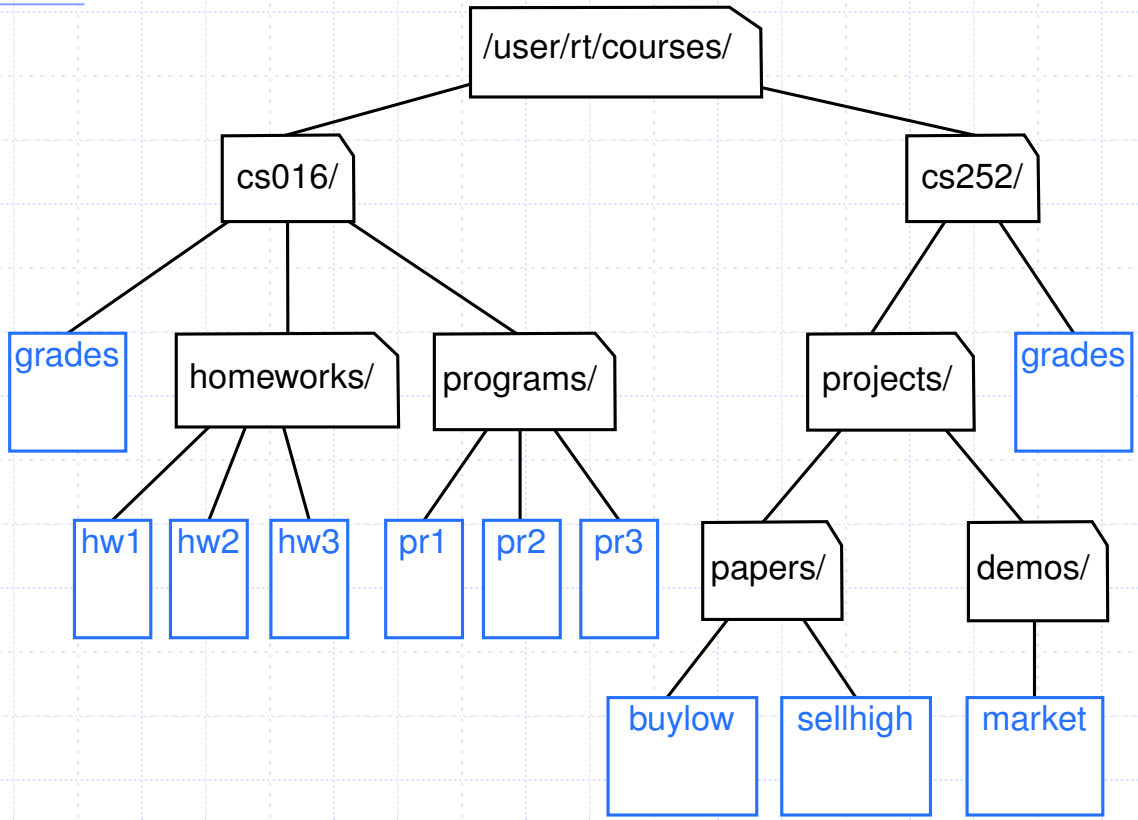
google images

Trees - Examples



organization structure of a corporation

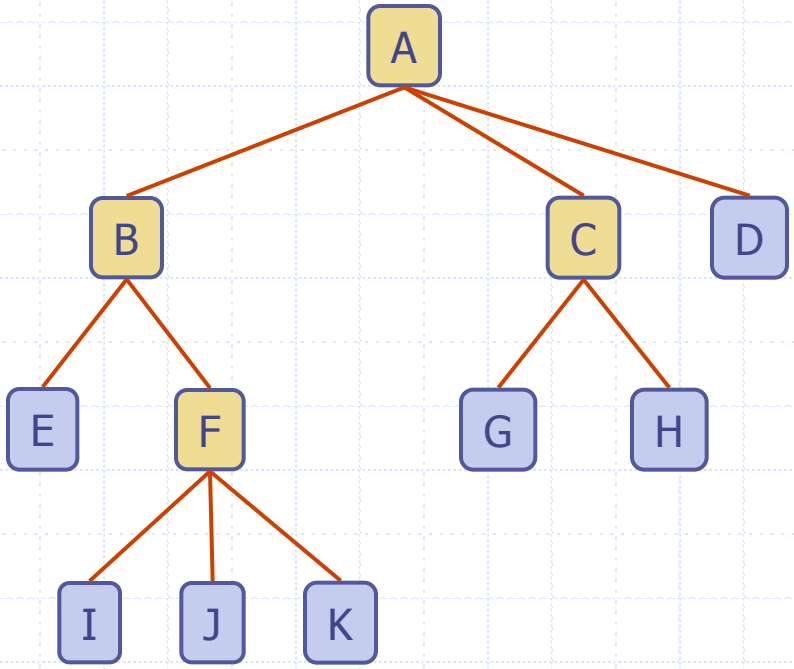
Trees - Examples (2)



Portion of a file system

Trees - Terminology

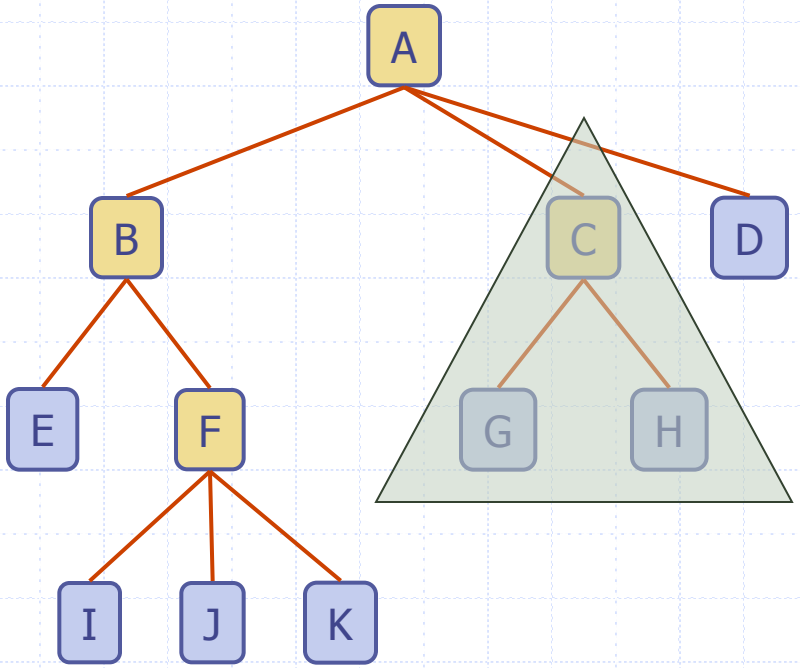
- *A* is the *root* node
- *B* is *parent* of *E* and *F*
- *A* is *ancestor* of *E* and *F*
- *E* and *F* are *descendants* of *A*
- *C* is the *sibling* of *B*
- *E* and *F* are *children* of *B*
- *E*, *I*, *J*, *K*, *G*, *H*, and *D* are *leaves*
- *A*, *B*, *C*, and *F* are *internal nodes*



Trees - Terminology (2)

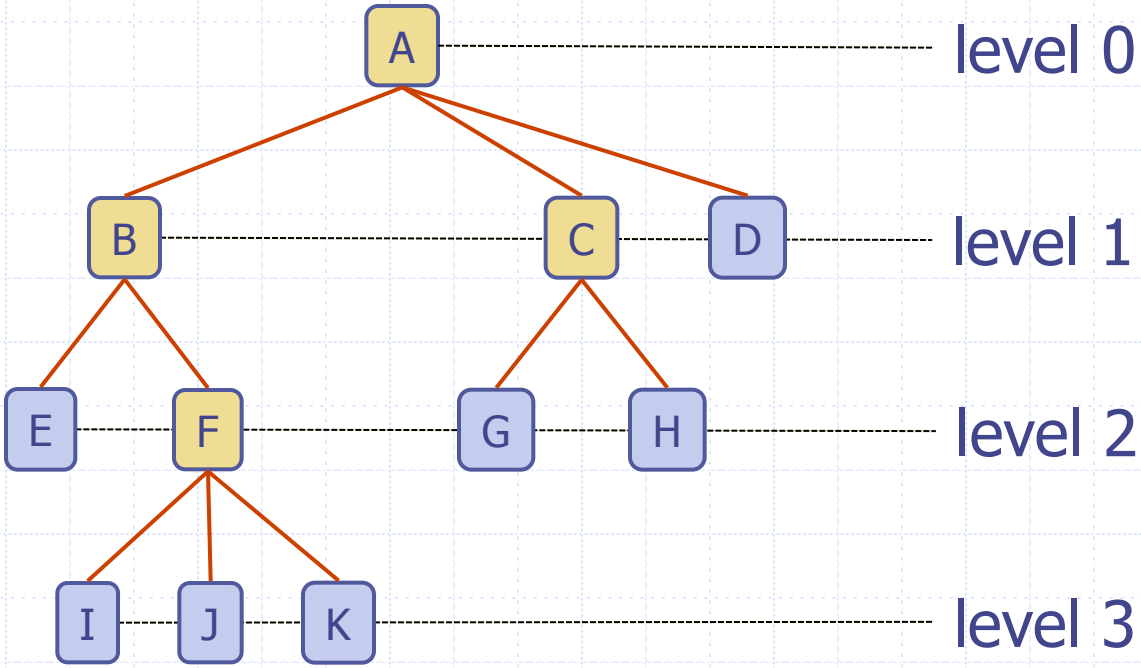
- *A* is the *root* node
- *B* is *parent* of *E* and *F*
- *A* is *ancestor* of *E* and *F*
- *E* and *F* are *descendants* of *A*
- *C* is the *sibling* of *B*
- *E* and *F* are *children* of *B*
- *E*, *I*, *J*, *K*, *G*, *H*, and *D* are *leaves*
- *A*, *B*, *C*, and *F* are *internal nodes*

- *Subtree:* tree consisting of node and its descendants



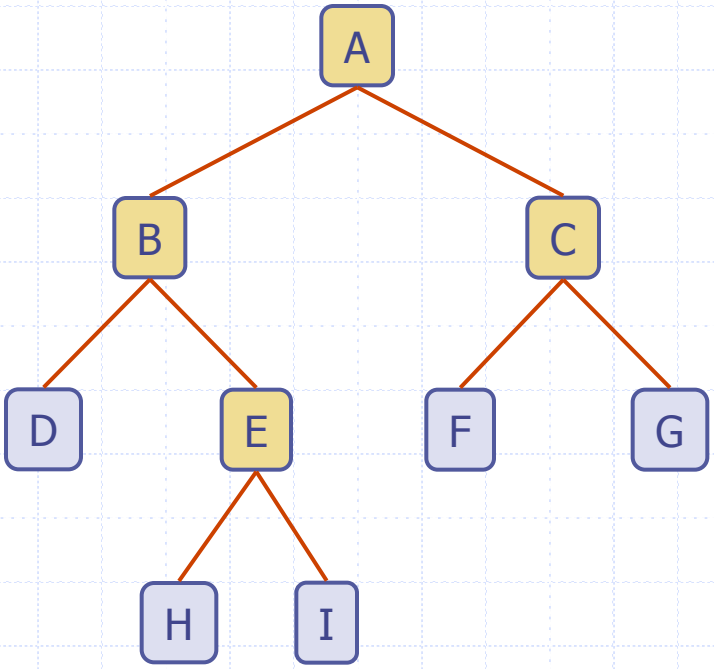
Trees - Terminology (3)

- The *depth* (*level*) of *E* is 2
- The *height* of the tree is 3
- The *degree* of node F is 3



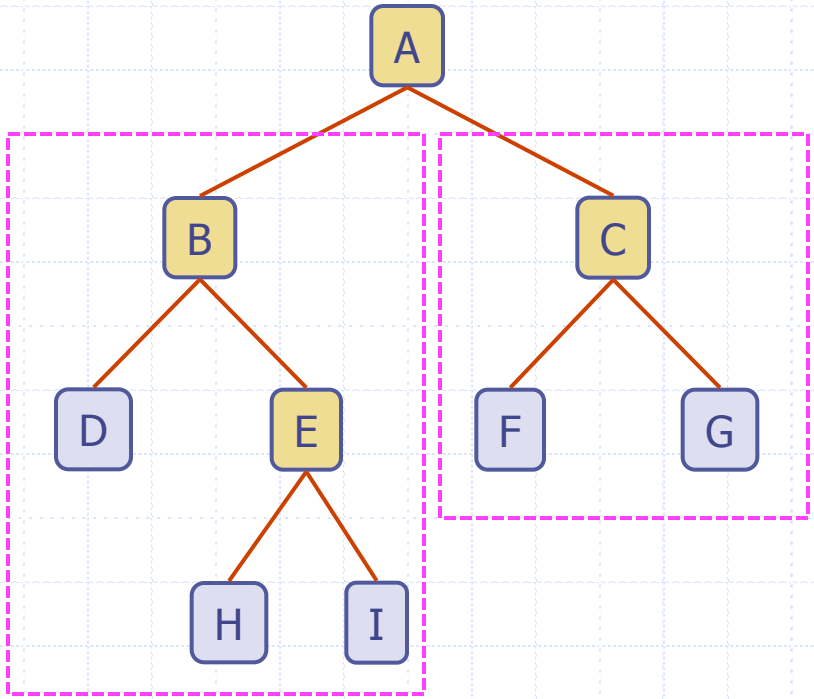
Binary Trees

- An **ordered tree** is one in which the children of each node are ordered
- **Binary tree**: ordered tree with all nodes having at most 2 children
  - left child and right child



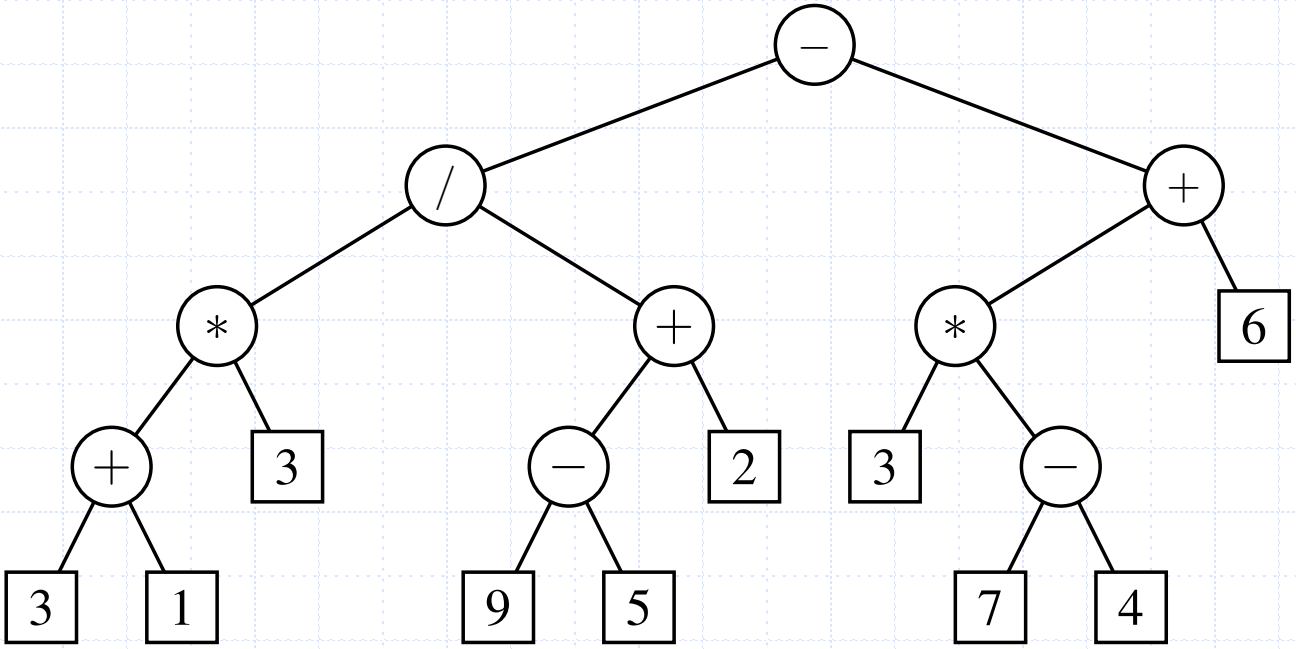
Binary Trees

- Recursive definition of binary tree
  - either a leaf or
  - an internal node (the root) and one/two binary trees (left subtree and/or right subtree)



Example of Binary Trees - Arithmetic Expression Tree

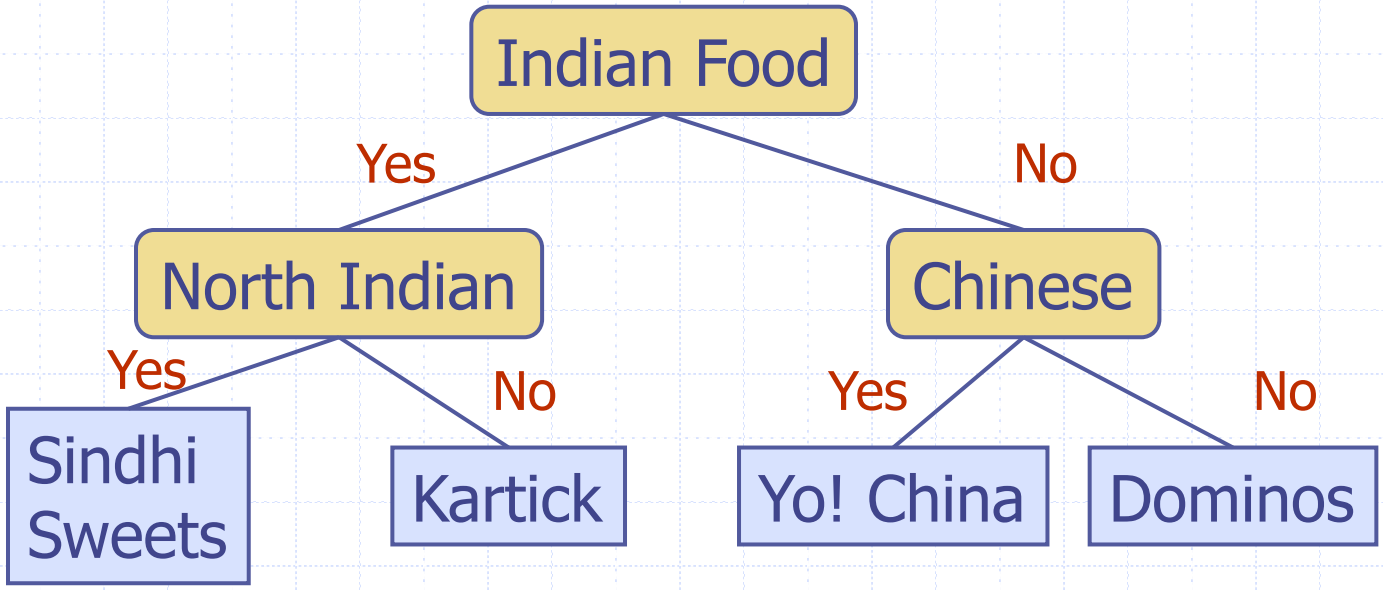
- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands



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

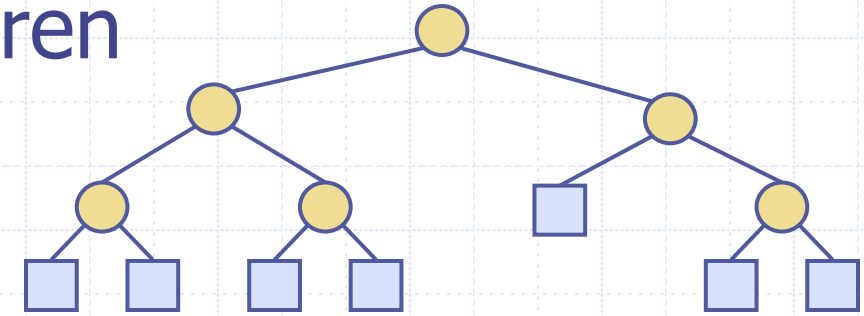
Example of Binary Trees - Decision Tree

- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision

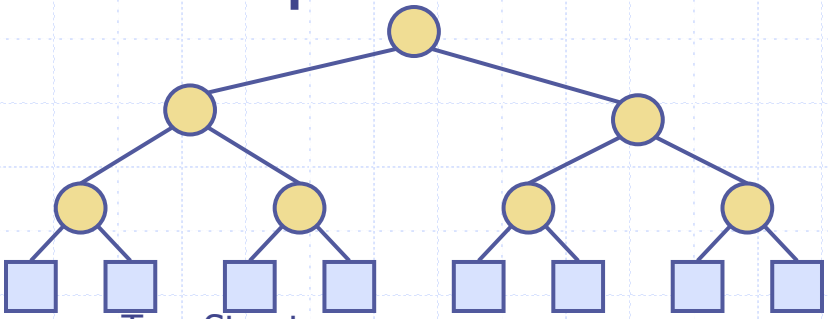


Proper, Full, Complete Binary Trees

- Proper/Full - Every node has either zero or two children

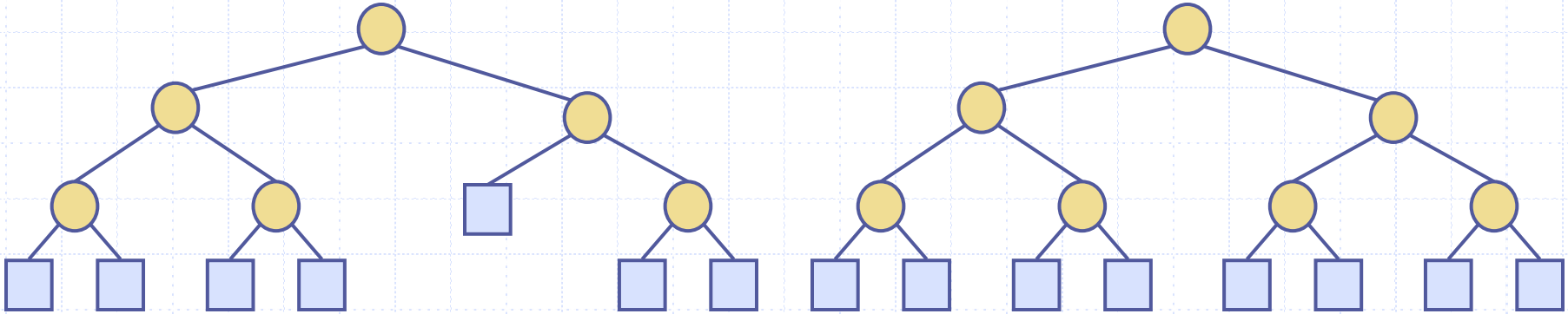


- Complete - every level except possibly the last is completely filled and all leaf nodes are as left as possible.



Binary tree from a complete binary tree

- A binary tree can be obtained from appropriate complete binary tree by pruning.



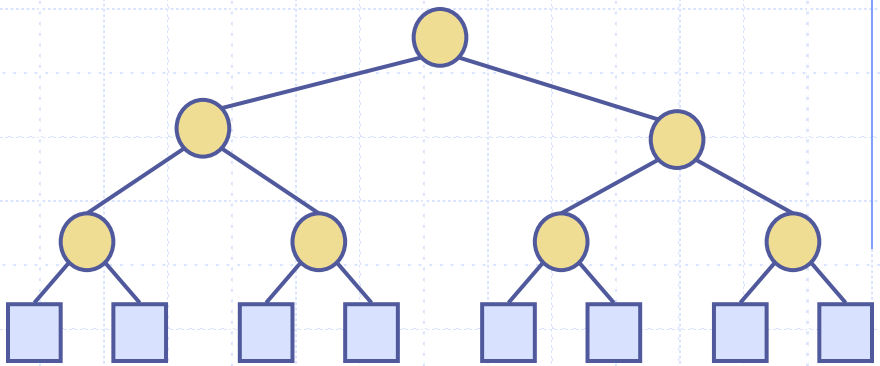
Properties of a Binary Tree

- Notations
  - n - number of nodes
  - n_E - number of leaves (external nodes)
  - n_I - number of internal nodes
  - h - height of the tree
- $h+1 \leq n \leq 2^{h+1} - 1$
- $1 \leq n_E \leq 2^h$
- $h \leq n_I \leq 2^h - 1$
- $\log(n+1) - 1 \leq h \leq n-1$

Properties of Binary Trees (2)

- $n_E \leq n_I + 1$
- proof by induction on n_I
  - Tree with 1 node has a leaf but no internal node
  - Assume $n_E \leq n_I + 1$ for tree with k-1 internal nodes
  - A tree with k internal nodes has k_1 internal nodes in the left subtree and k-k_1-1 internal nodes in the right subtree
  - By induction $n_E \leq (k_1+1) + (k-k_1-1+1) = k+1$

Complete Binary Tree

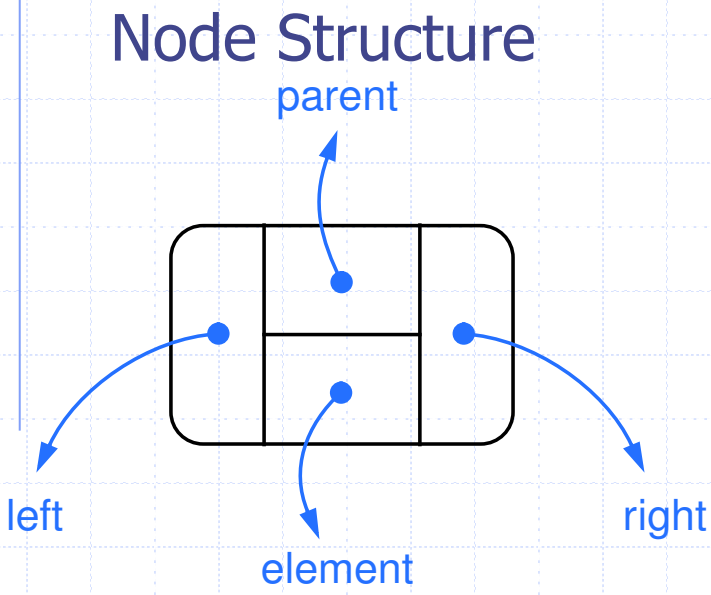


- level i has 2^i nodes
- In a tree of height h
  - leaves are at level h
  - $n_E = 2^h$
  - $n_I = 1 + 2 + 2^2 + \ldots + 2^{h-1} = 2^h - 1$
  - $n_I = n_E - 1$
  - $n = 2^{h+1} - 1$
- In a tree of n nodes
  - n_E is $(n+1)/2$
  - $h = \log_2 (n_E)$

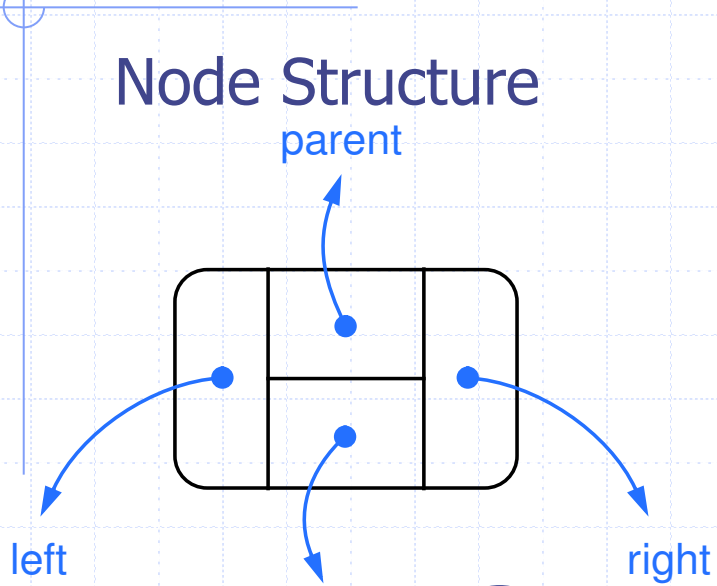
Tree ADT

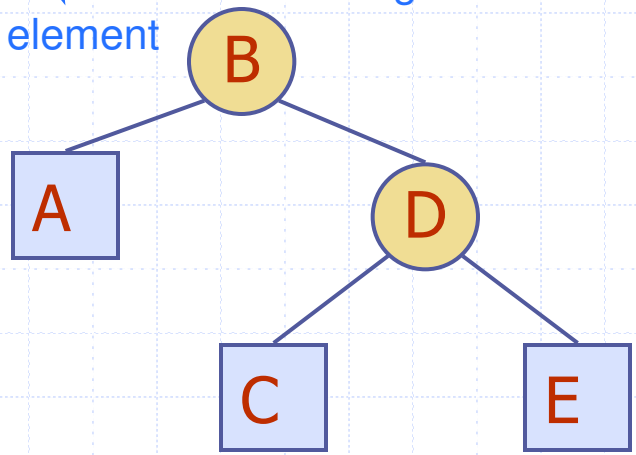
- We use positions to abstract nodes
- Generic methods:
  - integer size()
  - boolean isEmpty()
  - Iterator iterator()
  - Iterable positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - Iterable children(p)
  - Integer numChildren(p)
- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Additional update methods may be defined by data structures implementing the Tree ADT

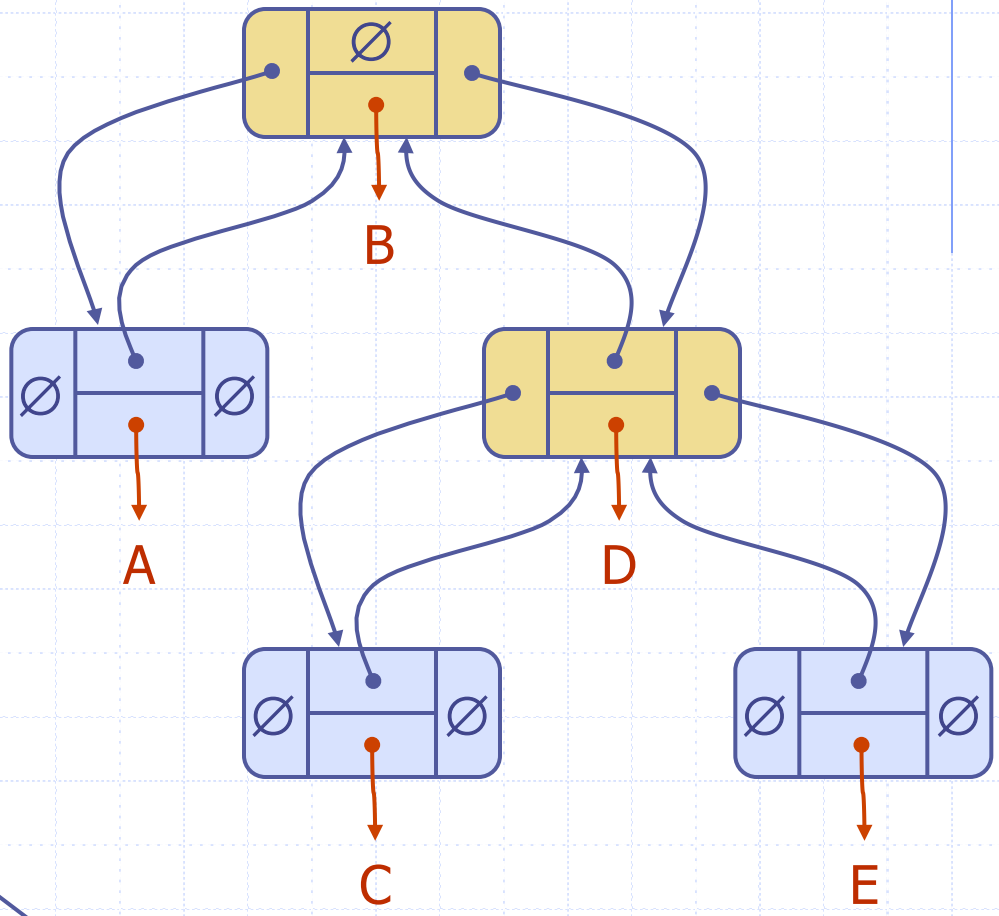
Linked Structure for Binary Trees



Linked Structure for Binary Trees



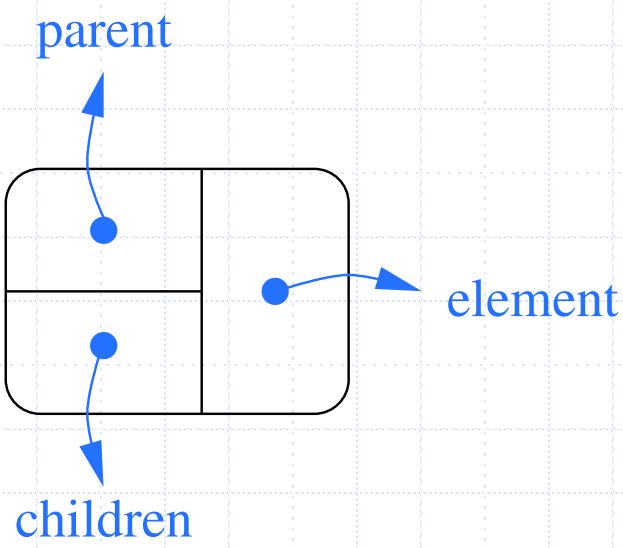




Code Fragments
8.8, 8.9, 8.10, 8.11

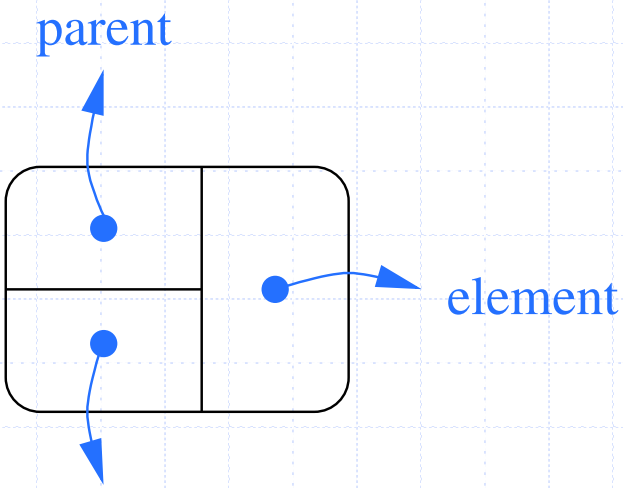
Linked Structure for General Trees

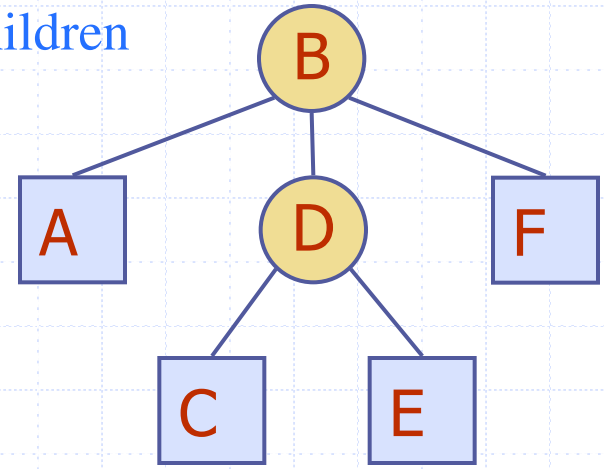
Node Structure

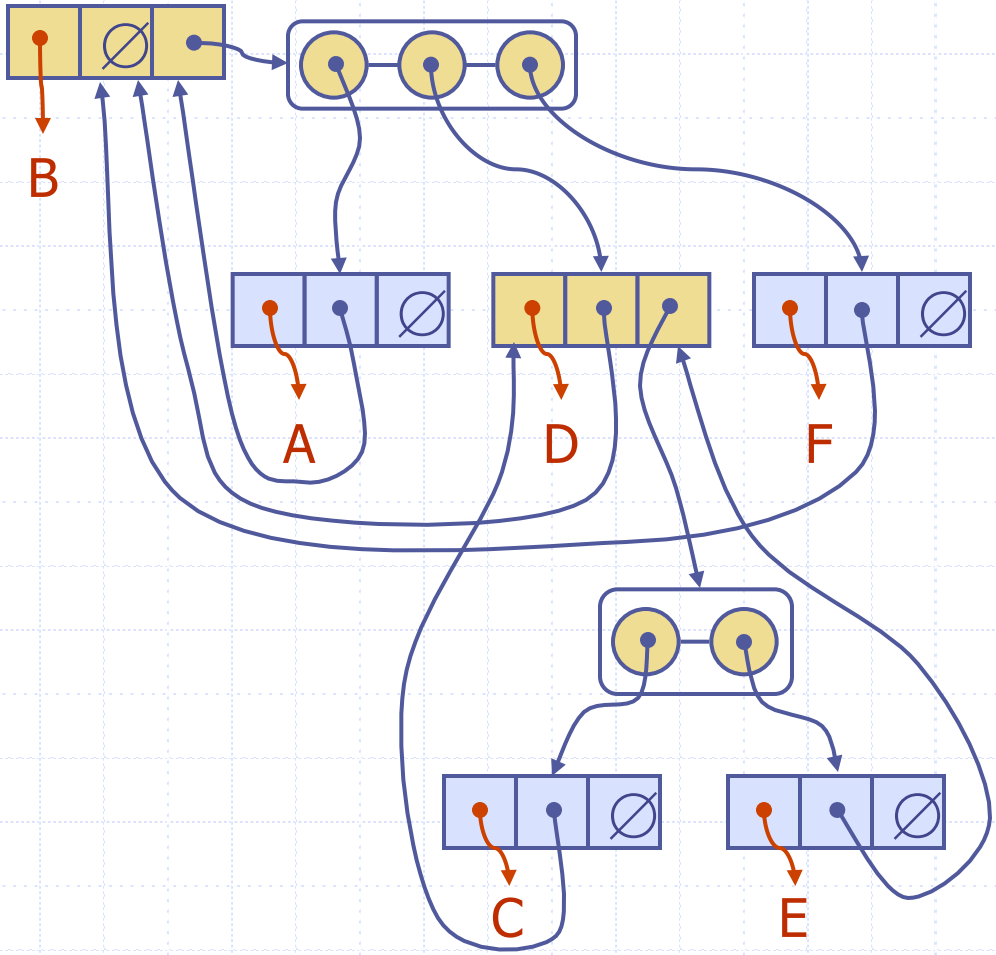


Linked Structure for General Trees

Node Structure







Computing Depth

- p be a position within the tree T
- calculate depth(p)

```
  public int depth(Position<E> p) throws
IllegalArgumentException {
      if (isRoot(p))
        return 0;
      else
        return 1 + depth(parent(p));
   }
```

Computing Height

```
 private int heightBad() { // works, but quadratic
worst-case time
    int h = 0;
    for (Position<E> p : positions())
      if (isExternal(p))// only consider leaf positions
        h = Math.max(h, depth(p));
    return h;
  }
```

Analysis:

$O(n + \Sigma_{p \varepsilon L}(d_p+1))$ is $O(n^2)$

Computing Height

```
 public int height(Position<E> p) throws
IllegalArgumentException {
    int h = 0;              // base case if p is external
    for (Position<E> c : children(p))
      h = Math.max(h, 1 + height(c));
    return h;
  }
```

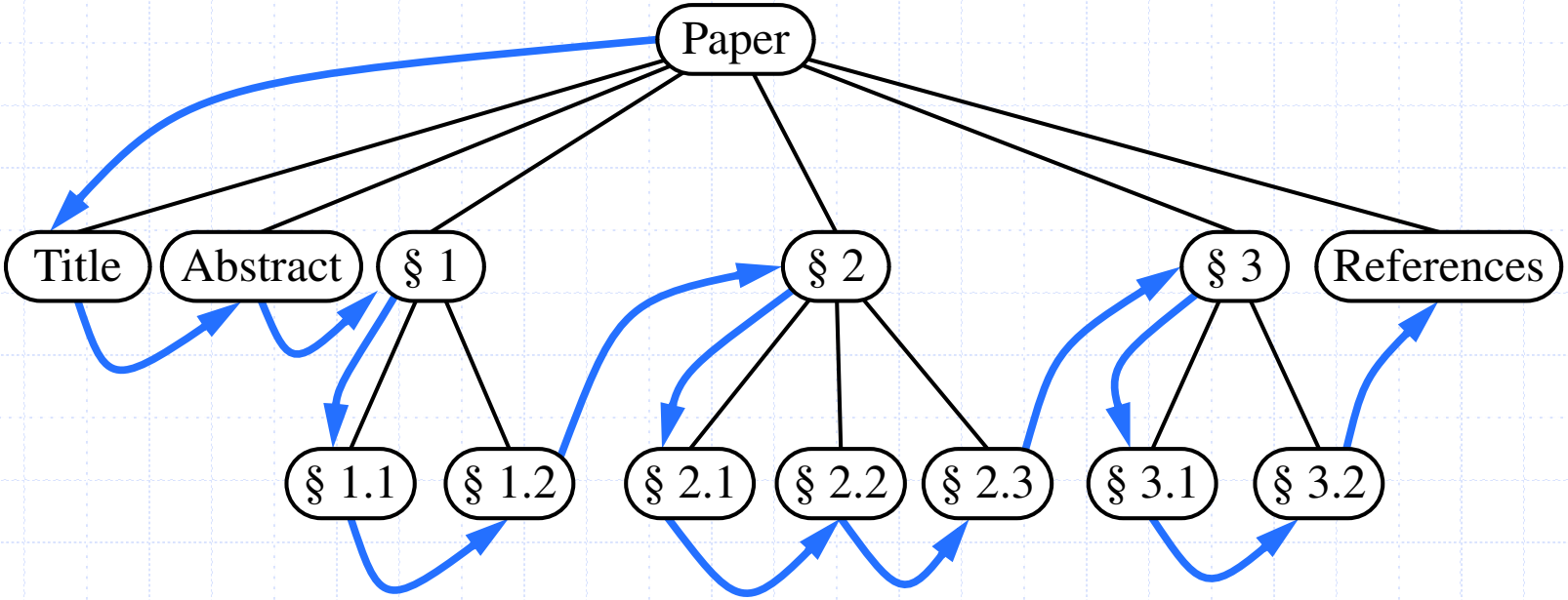
Analysis

$O(\Sigma_p(c_p+1))$ is $O(n)$

Tree Traversals

- Systematic way of visiting all nodes in a tree in a specified order
  - preorder - processes each node before processing its children
  - postorder - processes each node after processing its children

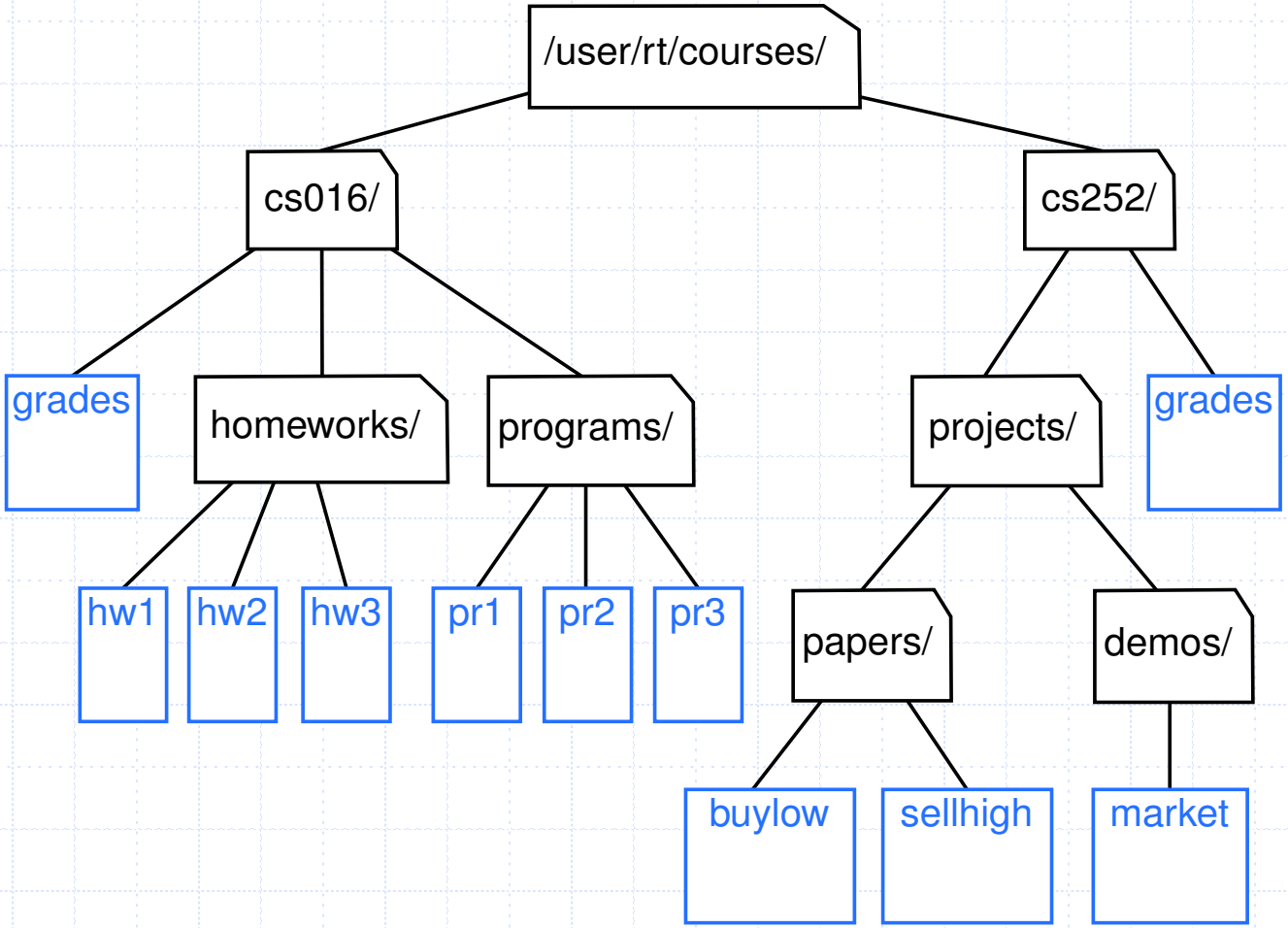
Preorder Traversal



Preorder Traversal - Algorithm

- Algorithm preorder(p)
  - perform the "visit" action for position p
  - for each child c in children(p) do
    - preorder(c)
- Example:
  - reading a document from beginning to end

Postorder Traversal



Postorder Traversal - Algorithm

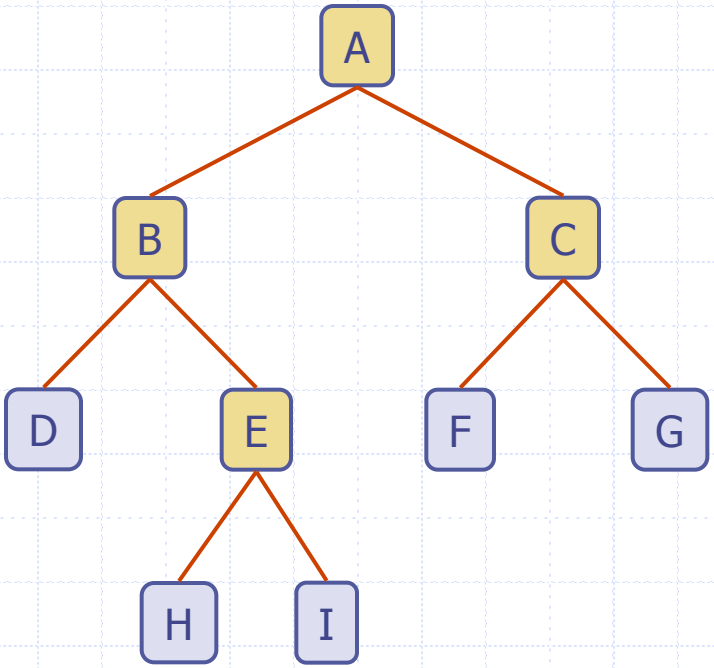
- Algorithm postorder(p)
  - for each child c in children(p) do
    - postorder(c)
  - perform the "visit" action for position p
- Example
  - du - disk usage command in Unix

Traversals of Binary Trees

- preorder(v)
  - visit(v)
  - preorder(v.leftchild())
  - preorder(v.rightchild())
- postorder(v)
  - postorder(v.leftchild())
  - postorder(v.rightchild())
  - visit(v)

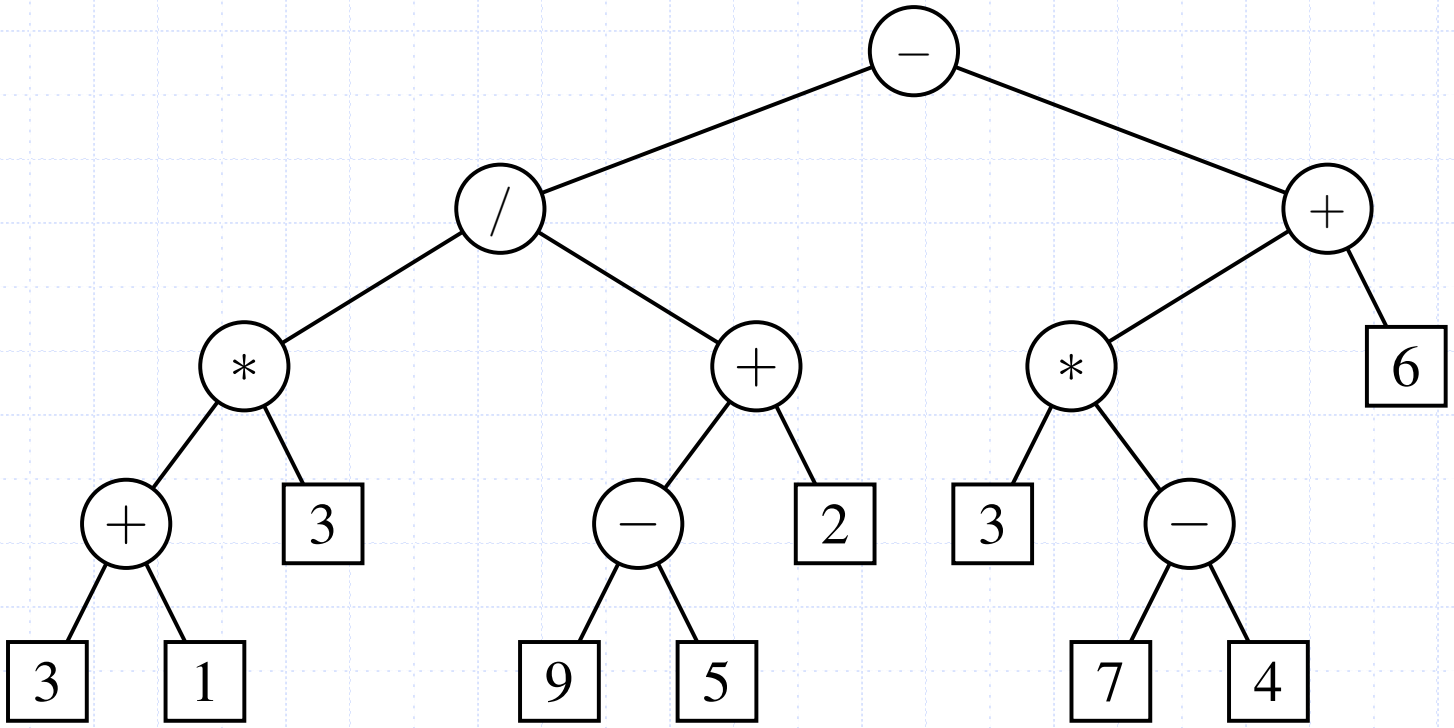
More Example of Traversals

- Visit - printing the data in the node
- Preorder traversal
  - a b d e h i c f g
- Postorder traversal
  - d h i e b f g c a



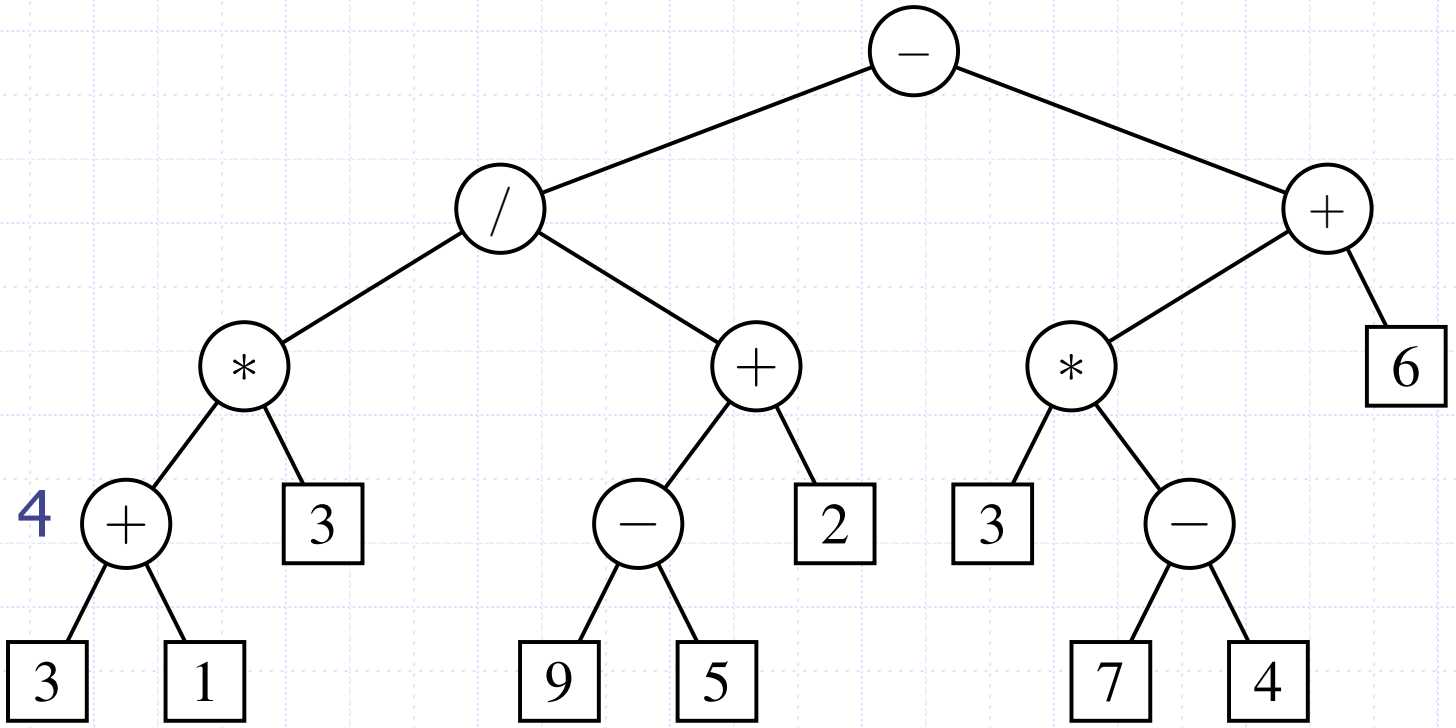
Application of Postorder Traversal

- Evaluating Arithmetic Expressions



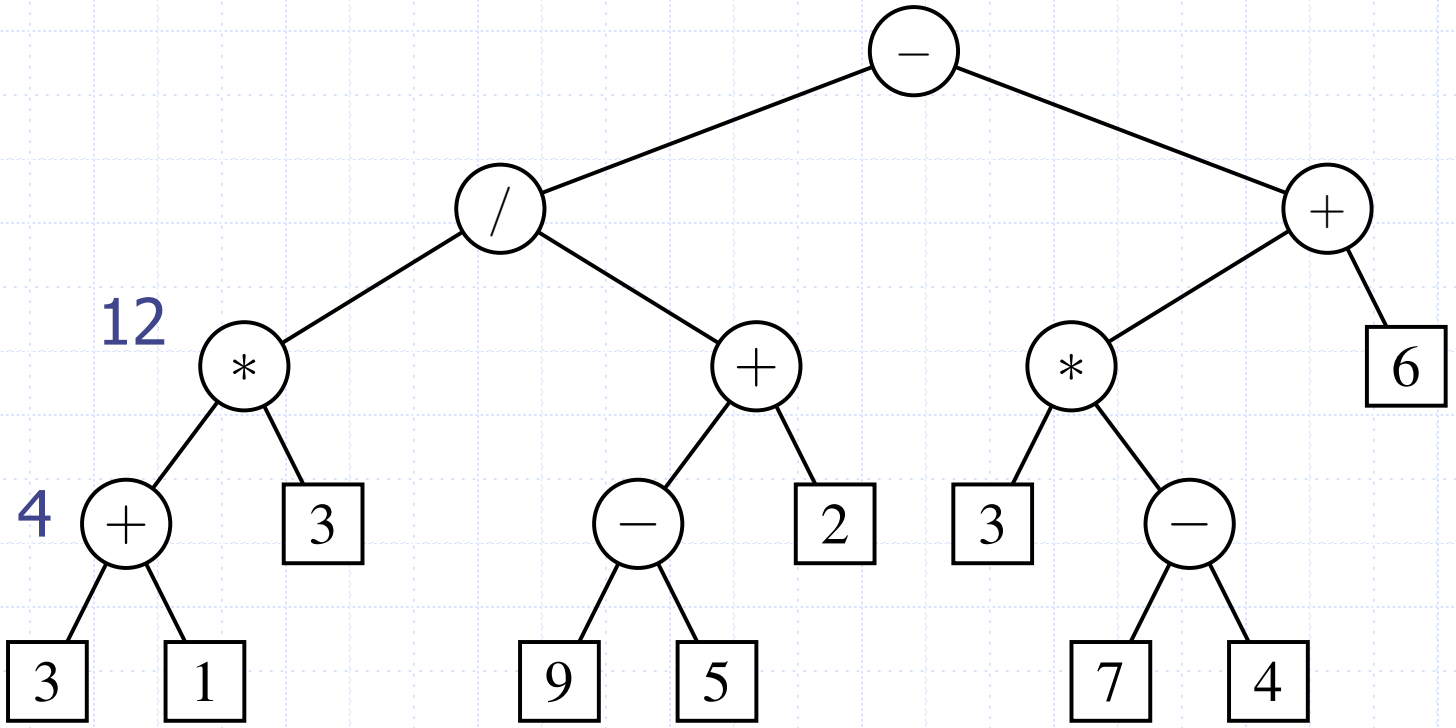
Application of Postorder Traversal

- Evaluating Arithmetic Expressions



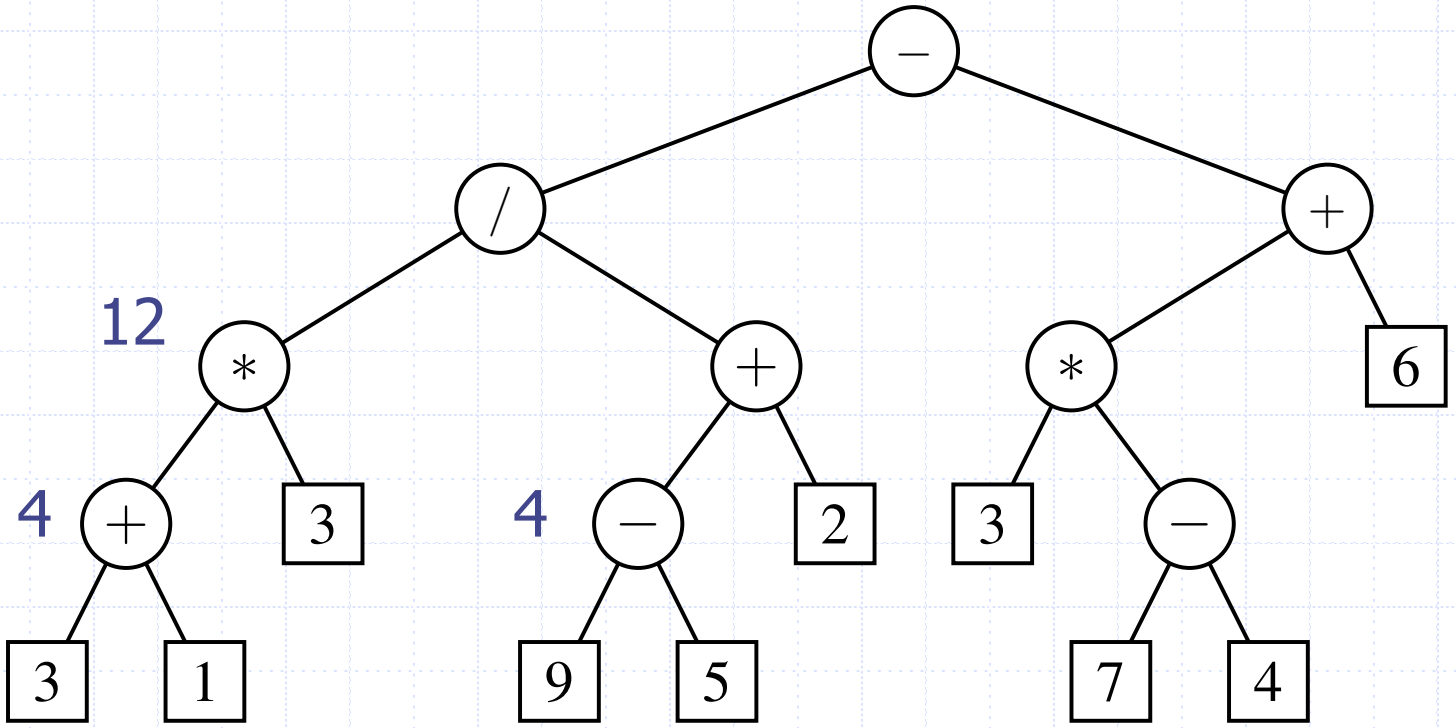
Application of Postorder Traversal

- Evaluating Arithmetic Expressions



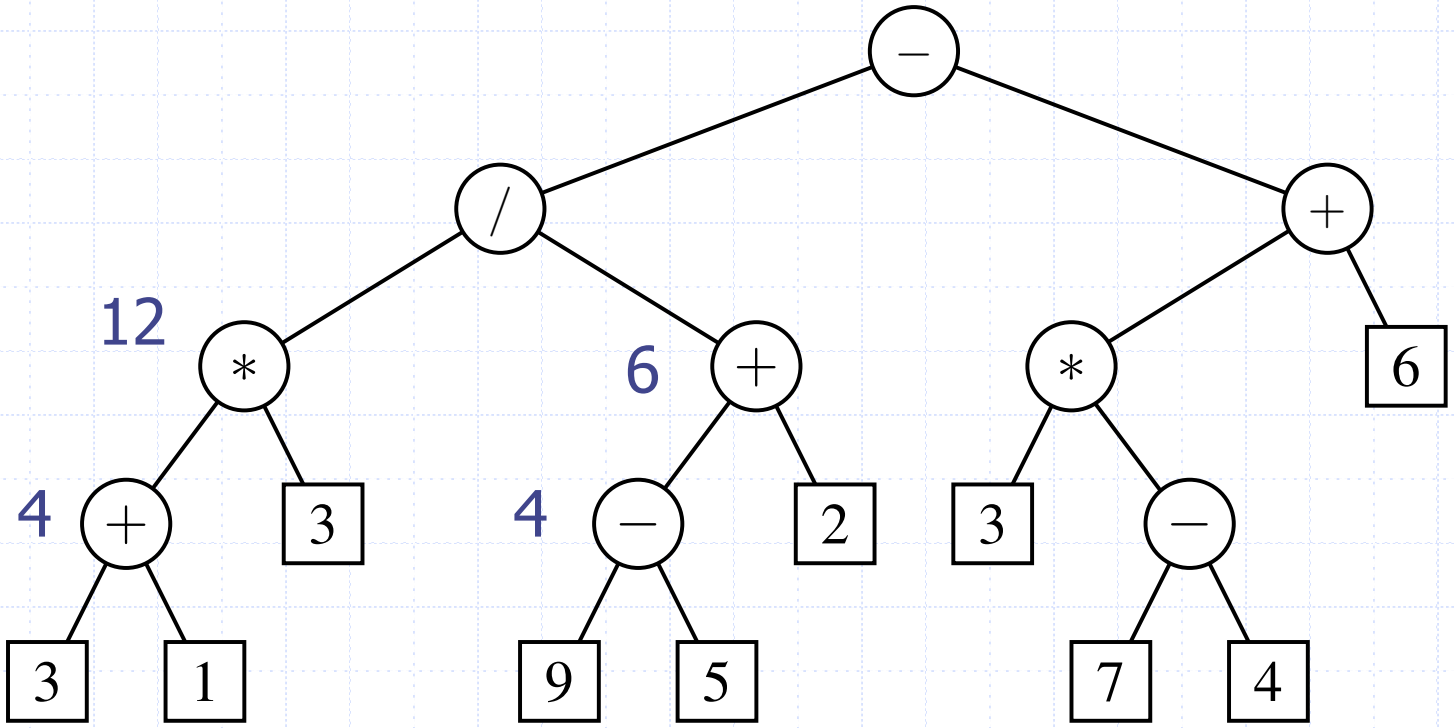
Application of Postorder Traversal

- Evaluating Arithmetic Expressions



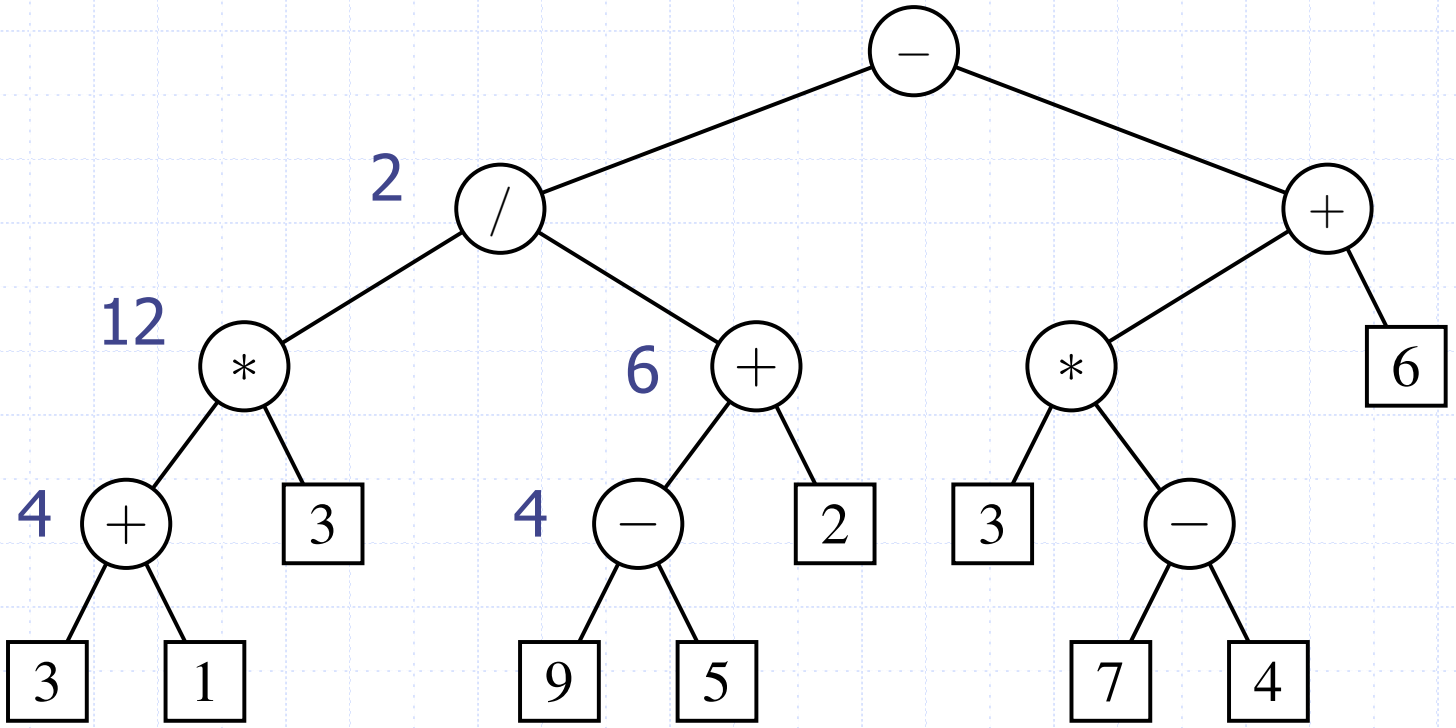
Application of Postorder Traversal

- Evaluating Arithmetic Expressions



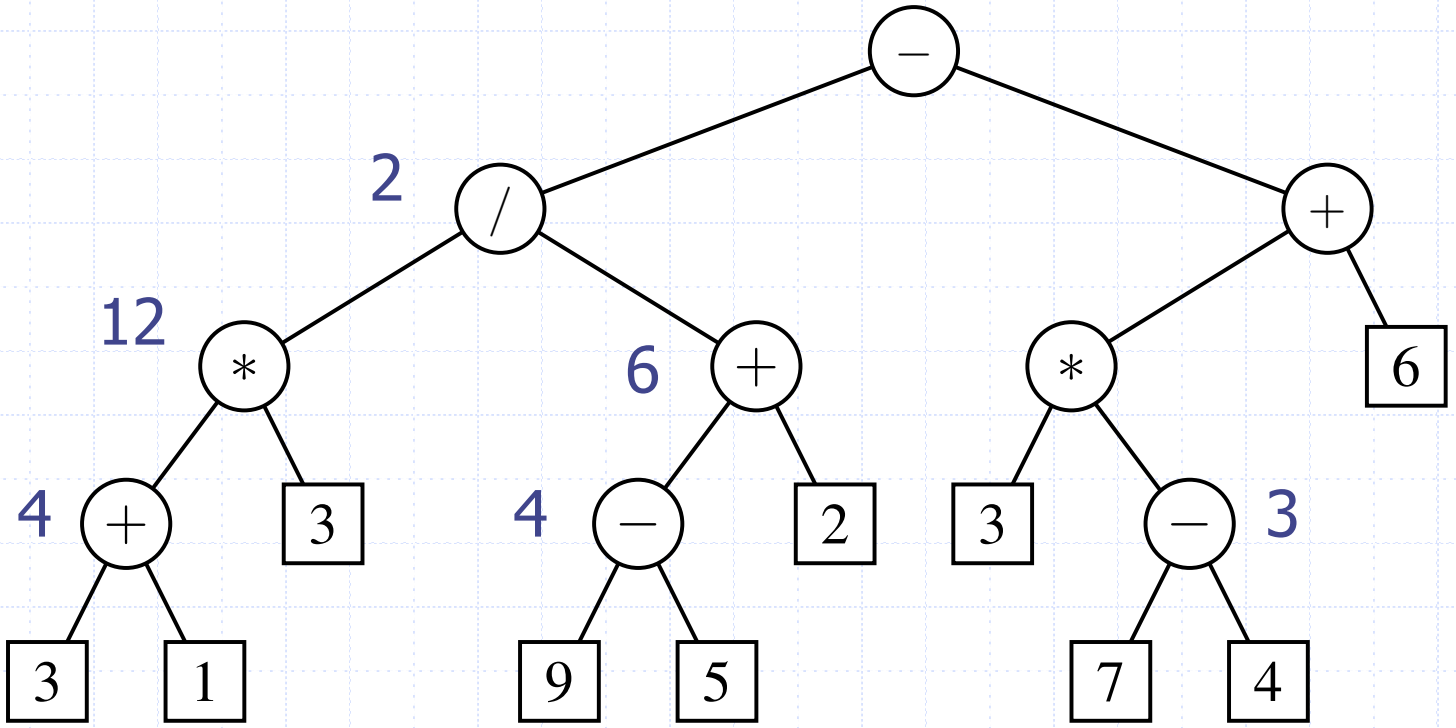
Application of Postorder Traversal

- Evaluating Arithmetic Expressions



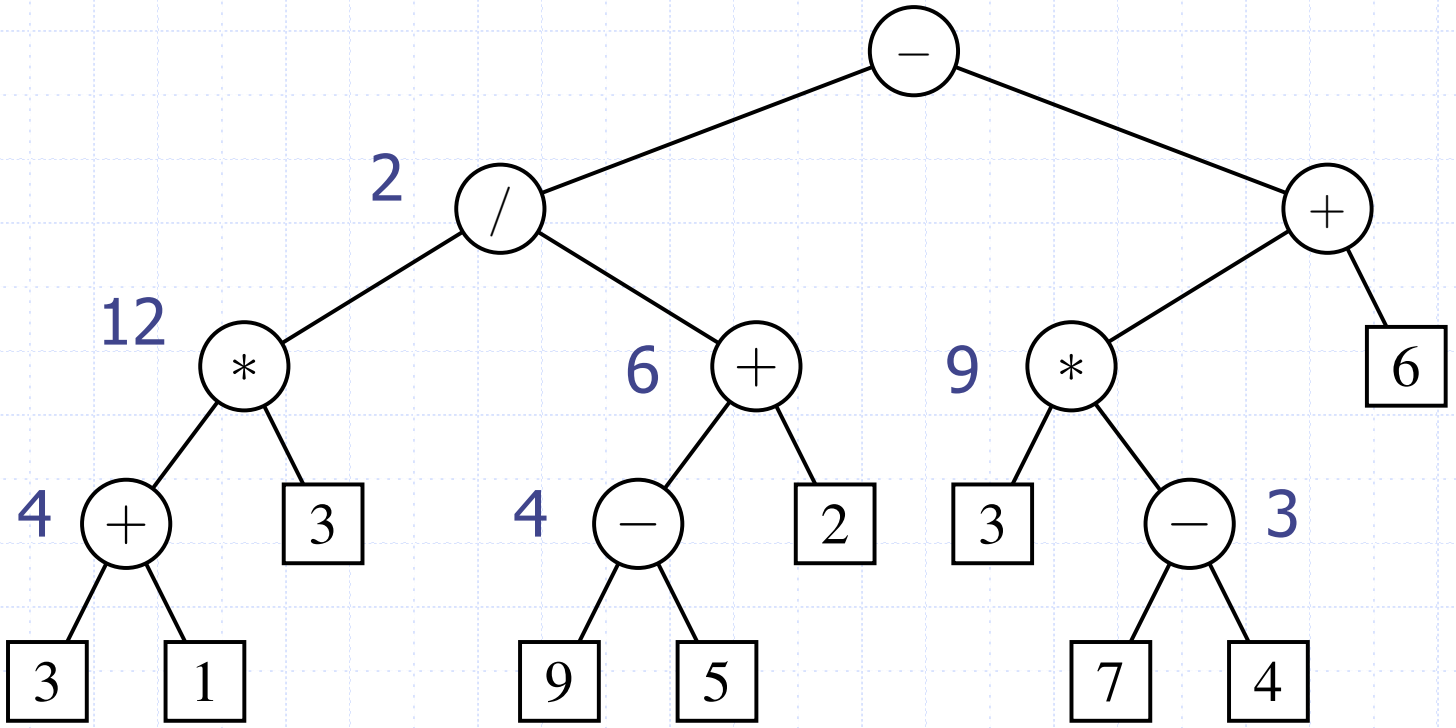
Application of Postorder Traversal

- Evaluating Arithmetic Expressions



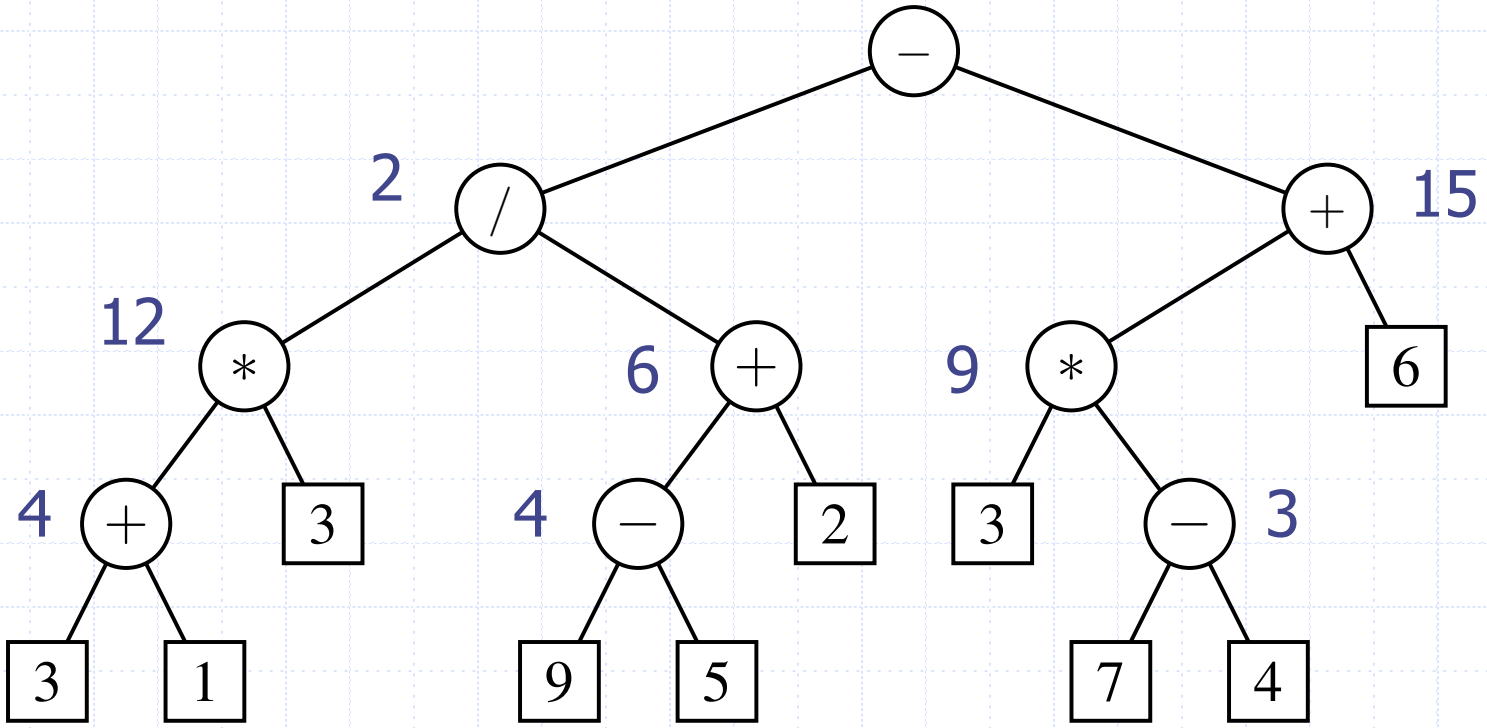
Application of Postorder Traversal

- Evaluating Arithmetic Expressions



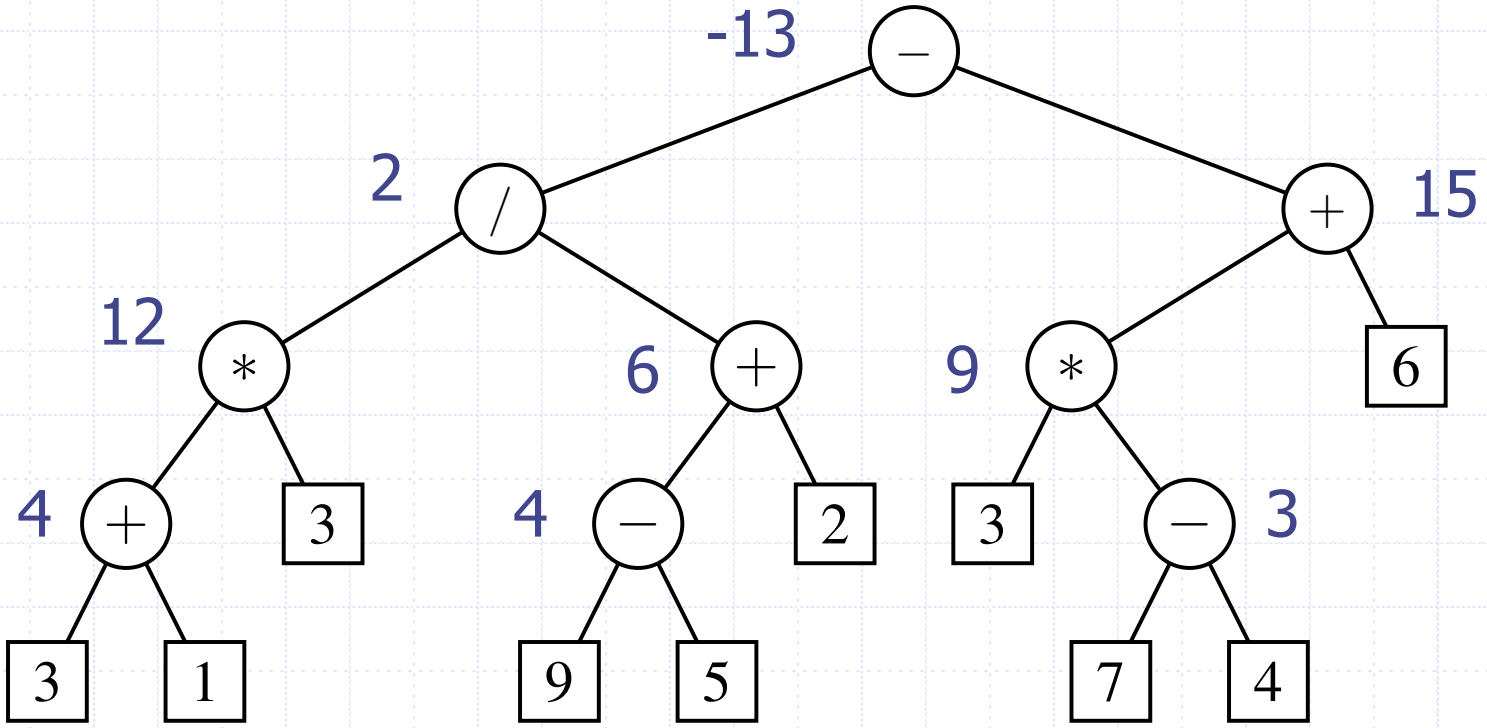
Application of Postorder Traversal

- Evaluating Arithmetic Expressions



Application of Postorder Traversal

- Evaluating Arithmetic Expressions

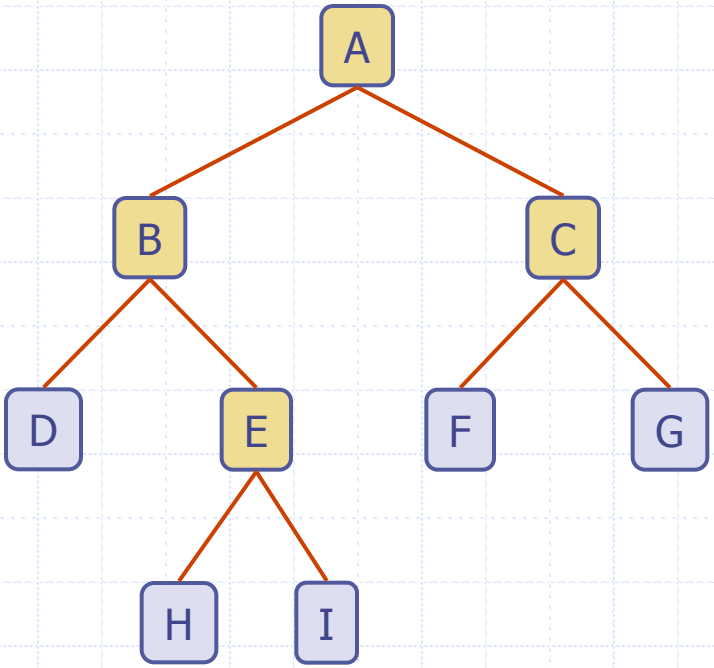


Inorder traversals

- Visit the node between the visit to the left and right subtree
- Algorithm inorder(p)
  - If p has a left child lc then
    - inorder(lc)
  - perform "visit" action for position p
  - If p has a right child rc then
    - inorder(rc)

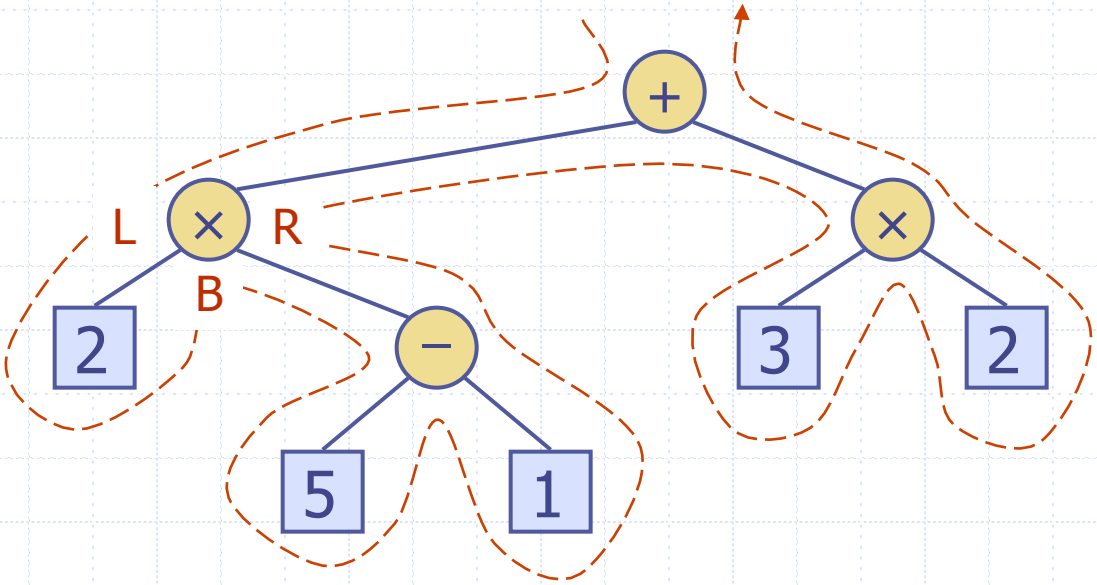
Example - Inorder Traversal

- Inorder
  - d b h e i a f c g



Euler Tour Traversal

- Generic traversal of a binary tree
- Includes as special cases the preorder, postorder and inorder traversals
- Walk around the tree and visit each node three times:
  - on the left (preorder)
  - from below (inorder)
  - on the right (postorder)



Building Tree from Preorder Traversal

- Given the preorder traversal, can we uniquely determine the binary tree?

Preorder

a b d e h i c f g

Building Tree from Postorder Traversal

- Given the postorder traversal, can we uniquely determine the binary tree?

Postorder

d h i e b f g c a

Building Tree from Pre- and In-Order Traversals

- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree

Preorder	Inorder
a b d e h i c f g	d b h e i a f c g

Building Tree from Pre- and In-Order Traversals

- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree

Preorder

a b d e h i c f g

Inorder

d b h e i a f c g

A

Building Tree from Pre- and In-Order Traversals

- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree

Preorder

a	b	d	e	h	i	c	f	g

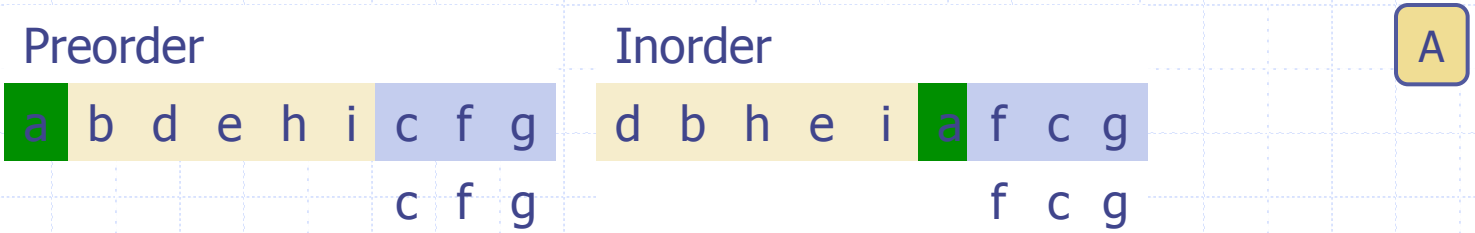
Inorder

d	b	h	e	i	a	f	c	g

A

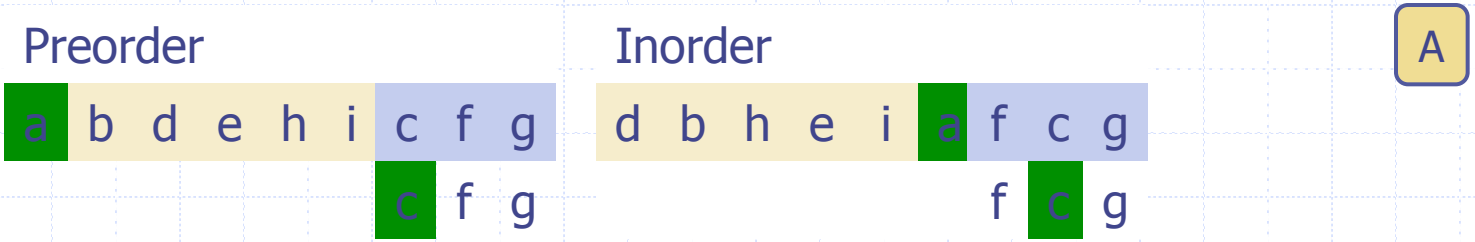
Building Tree from Pre- and In-Order Traversals

- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree



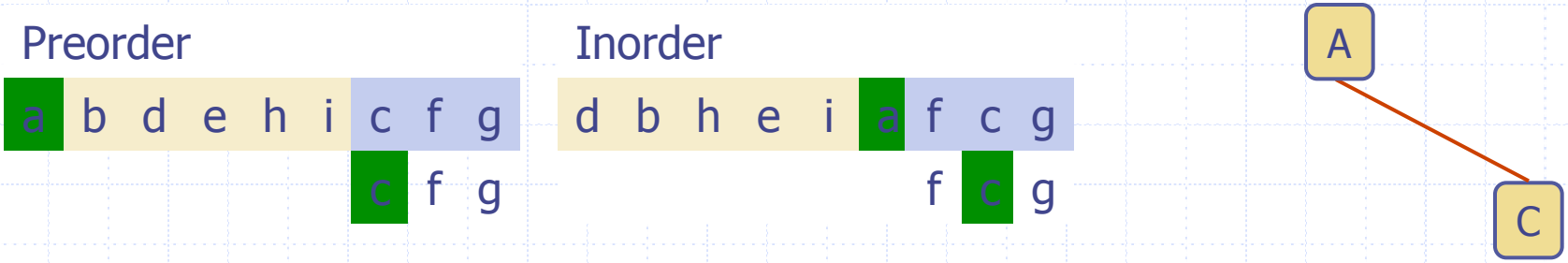
Building Tree from Pre- and In-Order Traversals

- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree



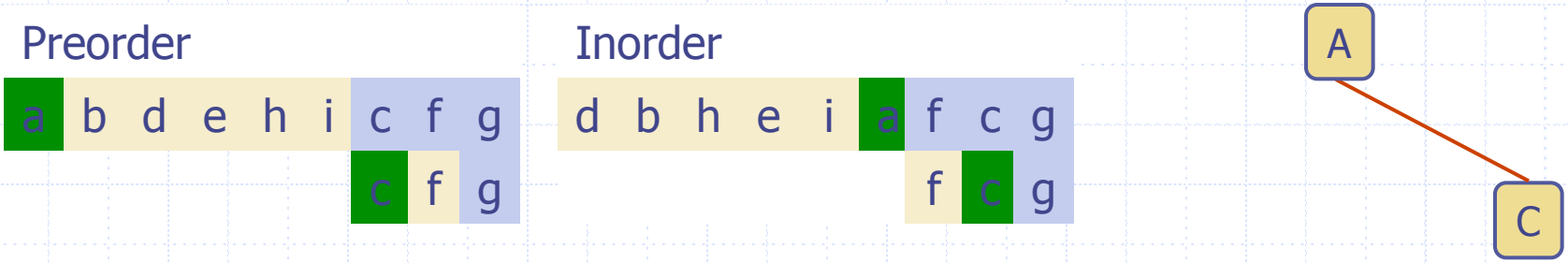
Building Tree from Pre- and In-Order Traversals

- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree



Building Tree from Pre- and In-Order Traversals

- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree



Building Tree from Pre- and In-Order Traversals

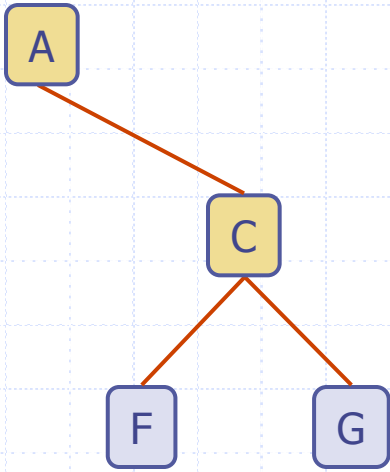
- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree

Preorder

a	b	d	e	h	i	c	f	g
						c	f	g

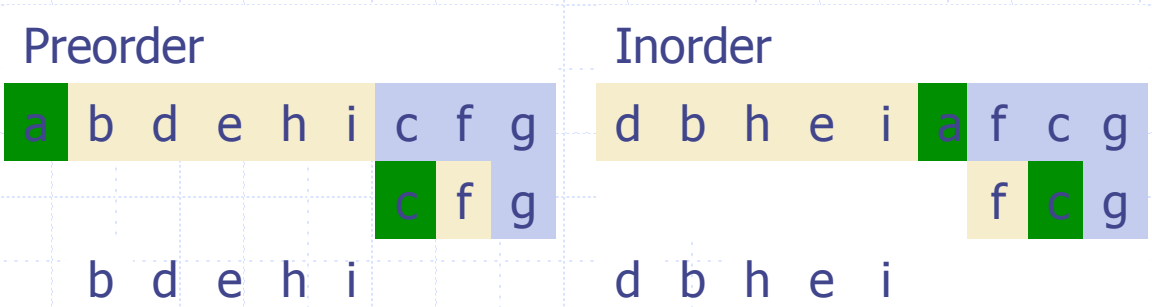
Inorder

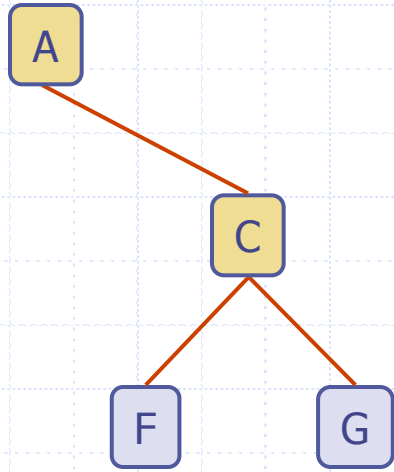
d	b	h	e	i	a	f	c	g
						f	c	g



Building Tree from Pre- and In-Order Traversals

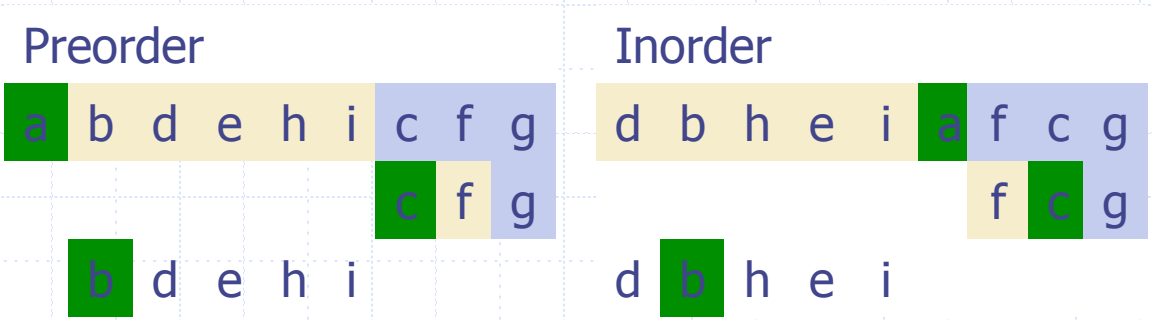
- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree

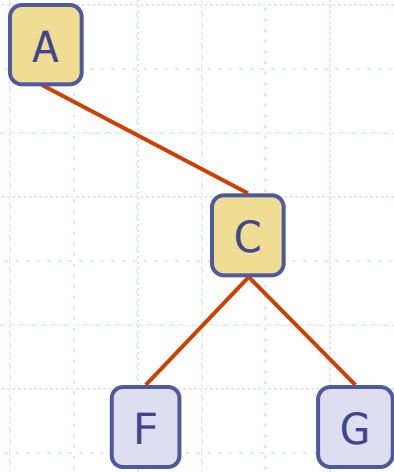




Building Tree from Pre- and In-Order Traversals

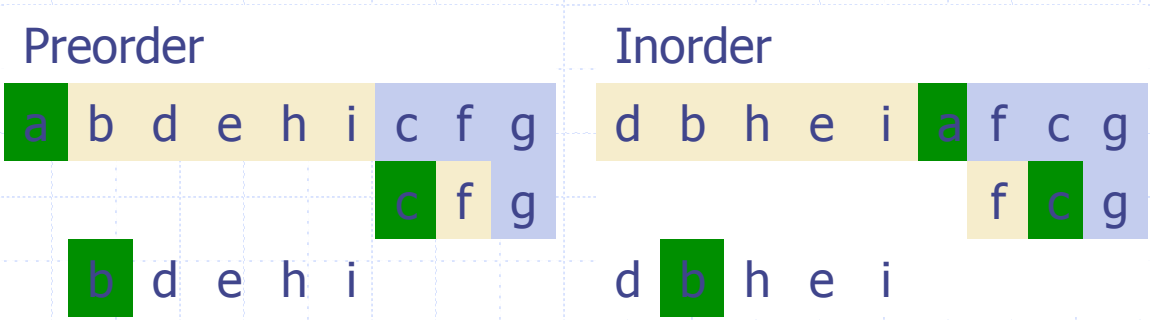
- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree

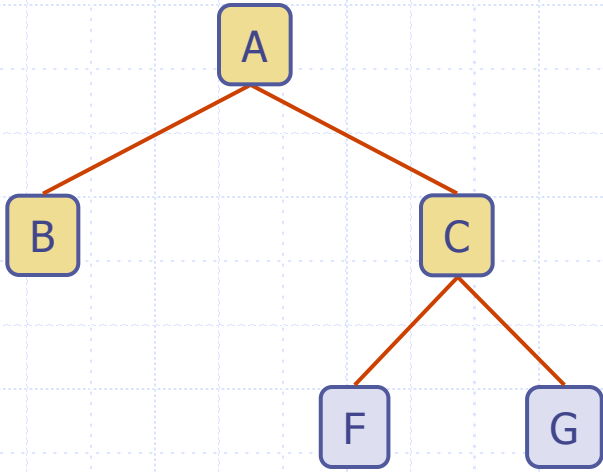




Building Tree from Pre- and In-Order Traversals

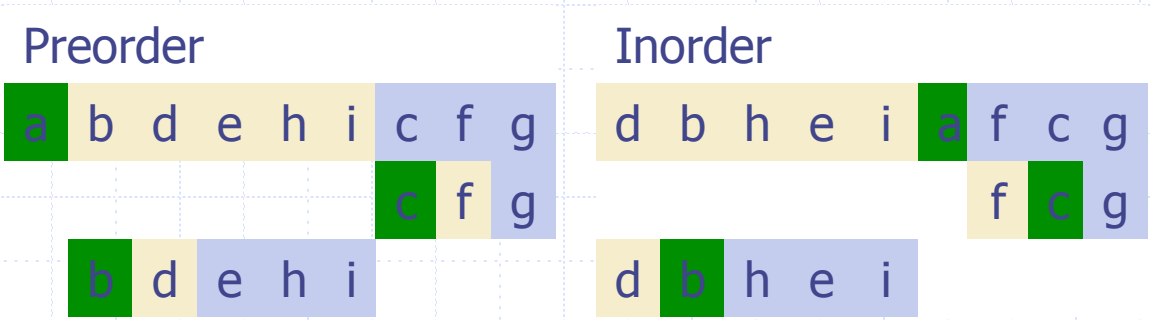
- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree

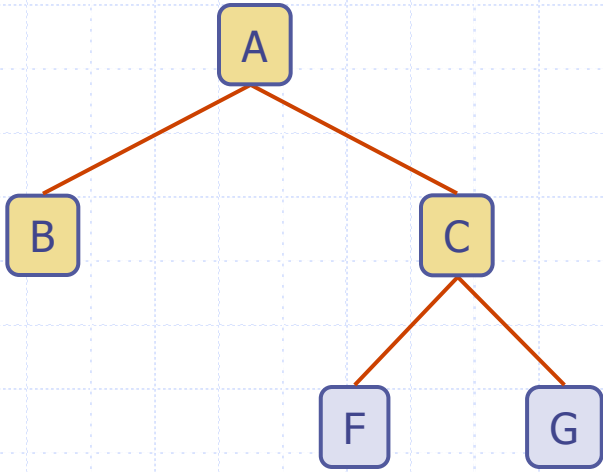




Building Tree from Pre- and In-Order Traversals

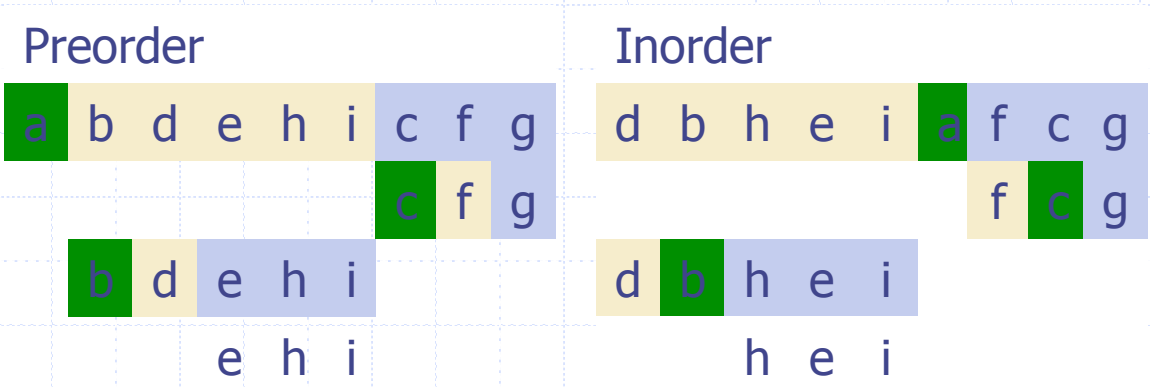
- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree

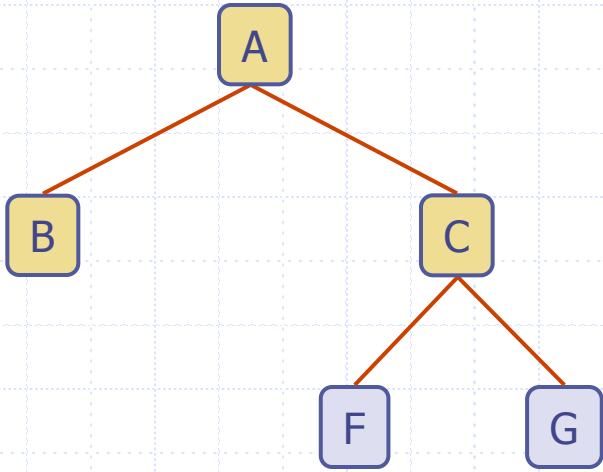




Building Tree from Pre- and In-Order Traversals

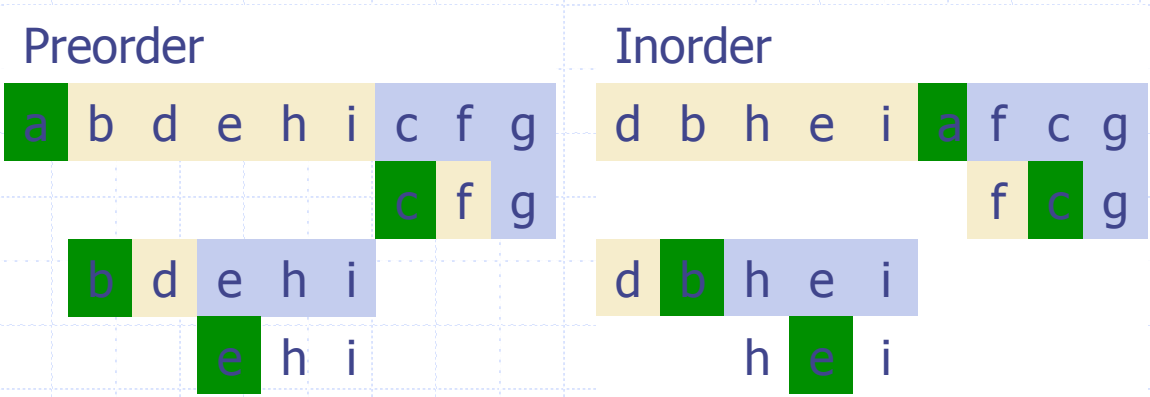
- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree

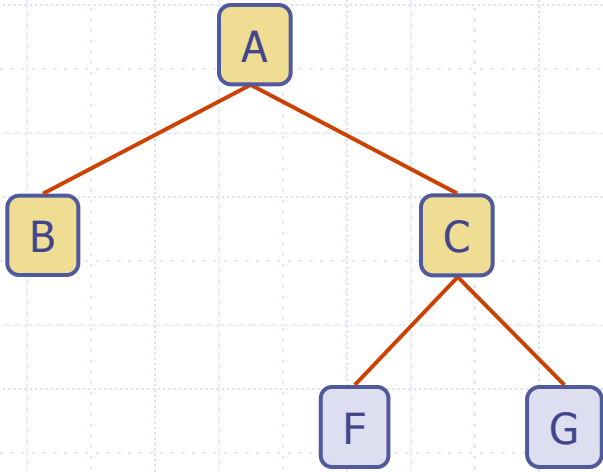




Building Tree from Pre- and In-Order Traversals

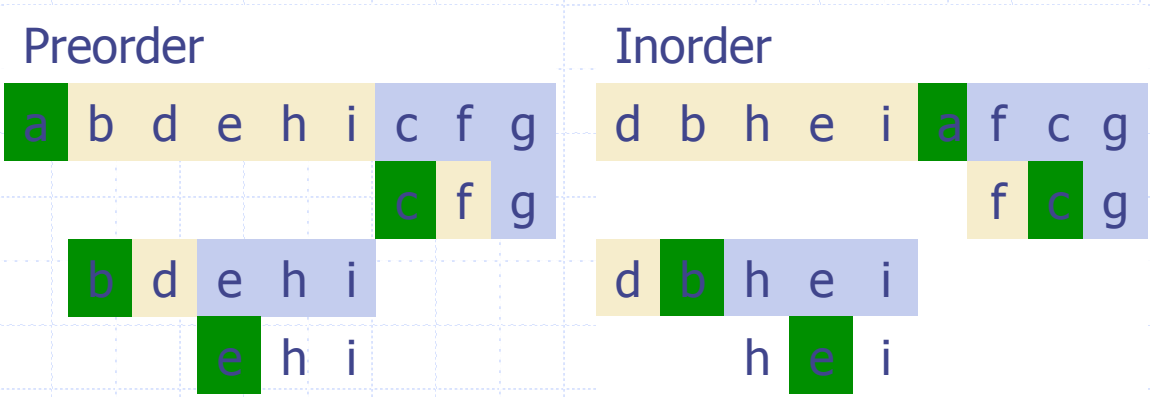
- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree

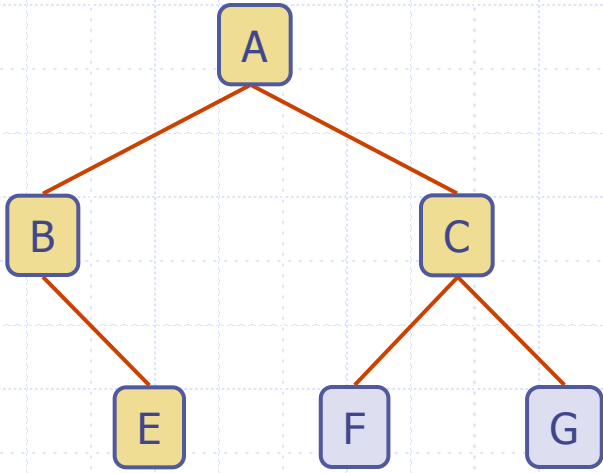




Building Tree from Pre- and In-Order Traversals

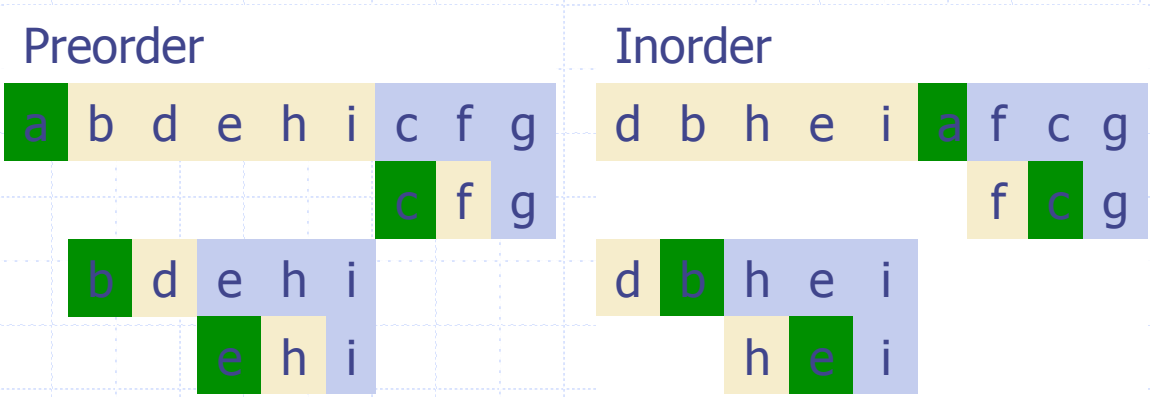
- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree

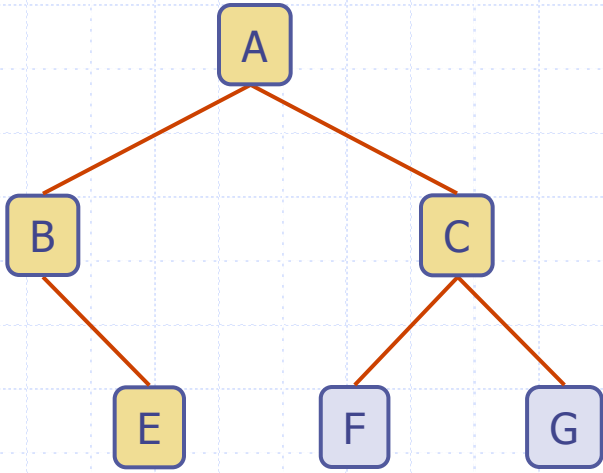




Building Tree from Pre- and In-Order Traversals

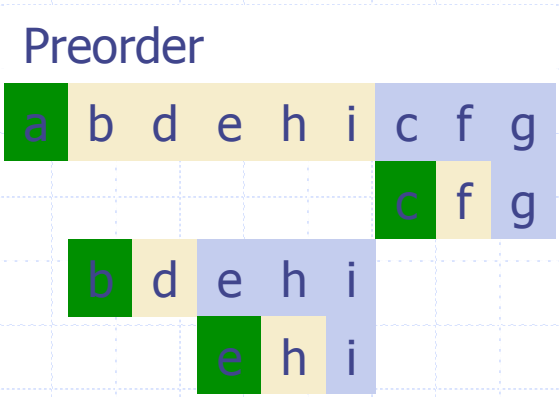
- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree

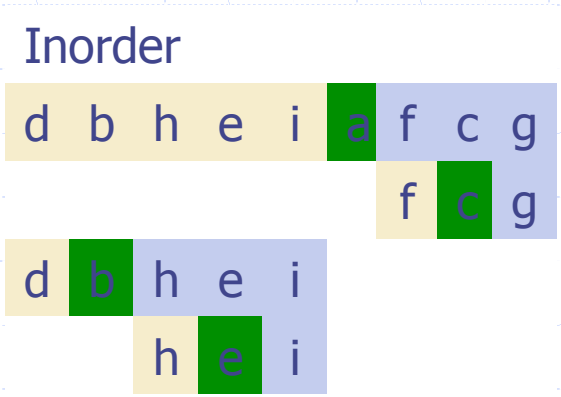


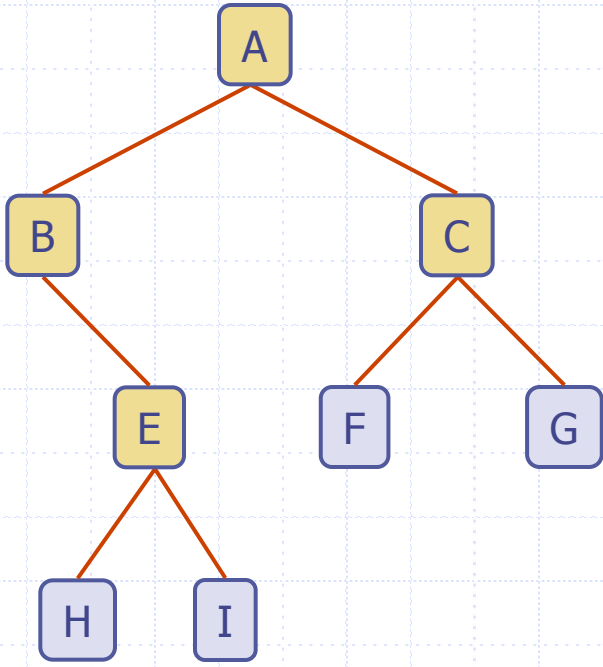


Building Tree from Pre- and In-Order Traversals

- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree

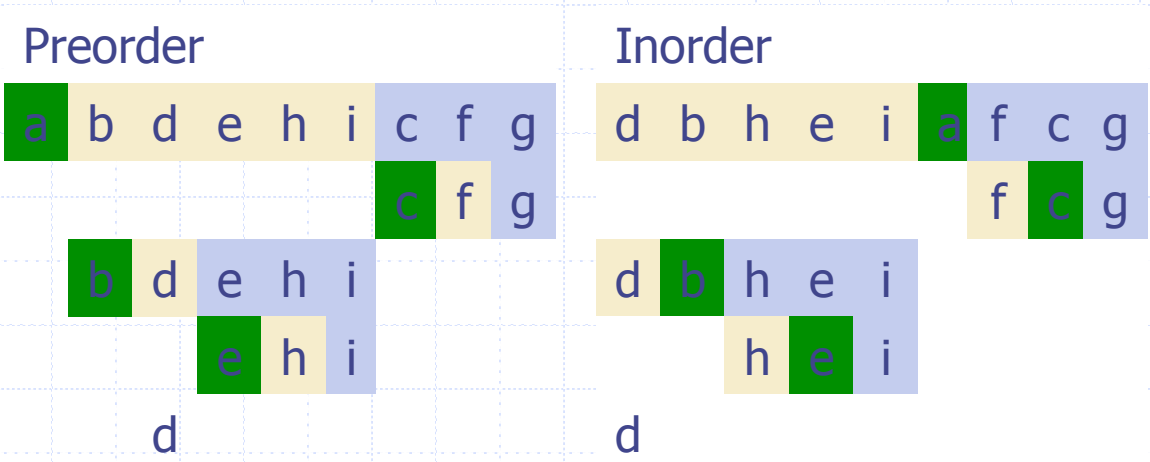


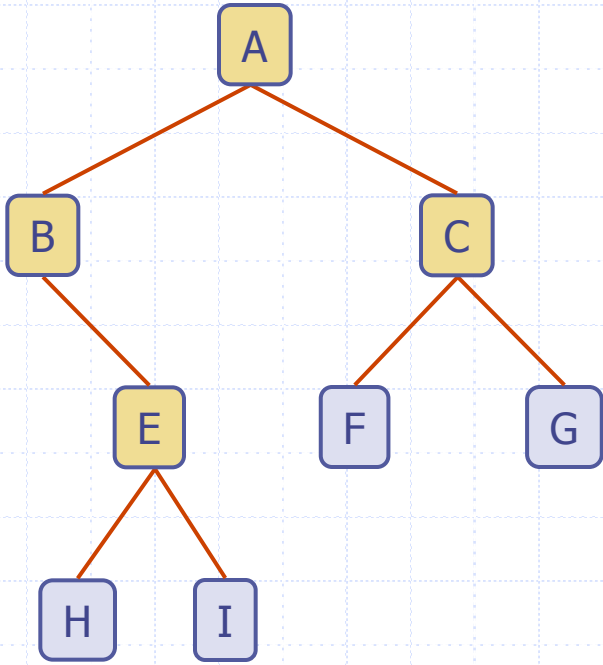




Building Tree from Pre- and In-Order Traversals

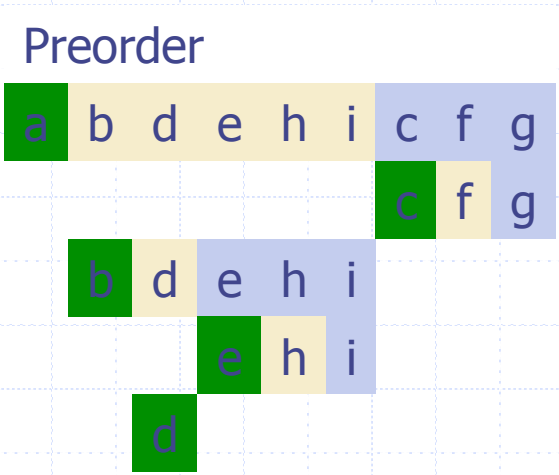
- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree

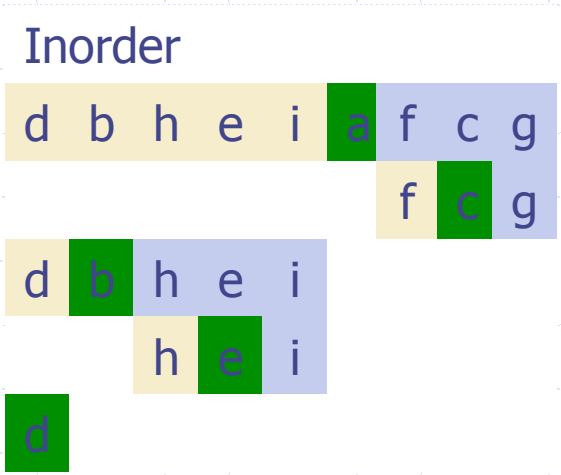


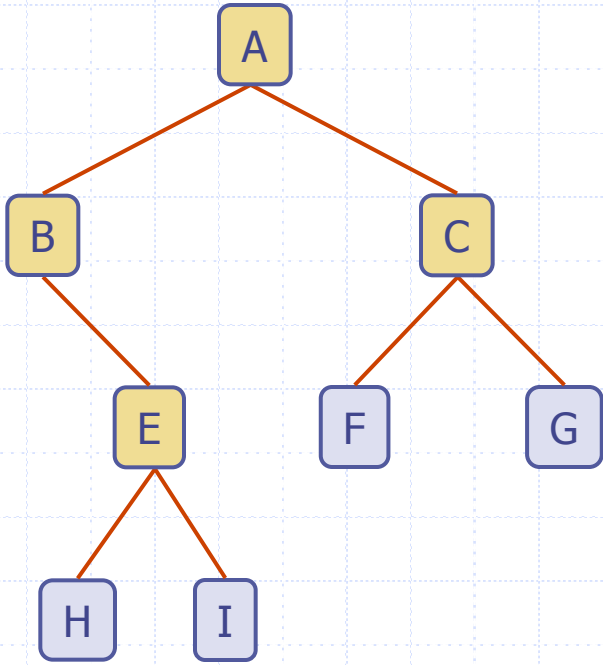


Building Tree from Pre- and In-Order Traversals

- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree

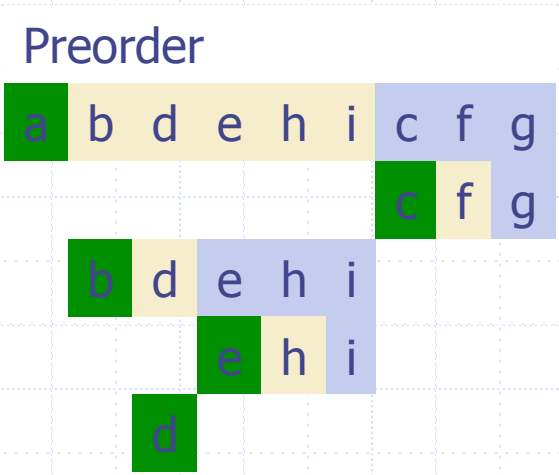


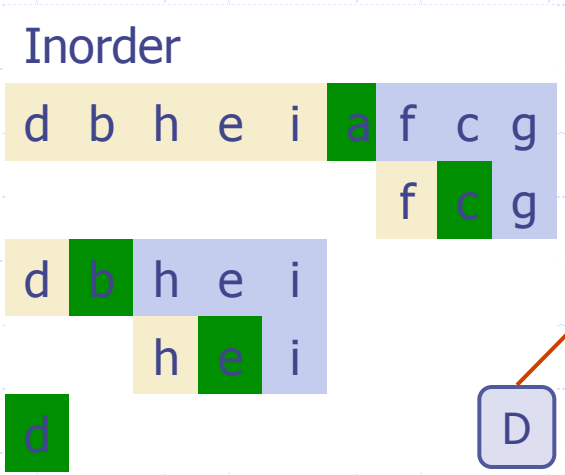


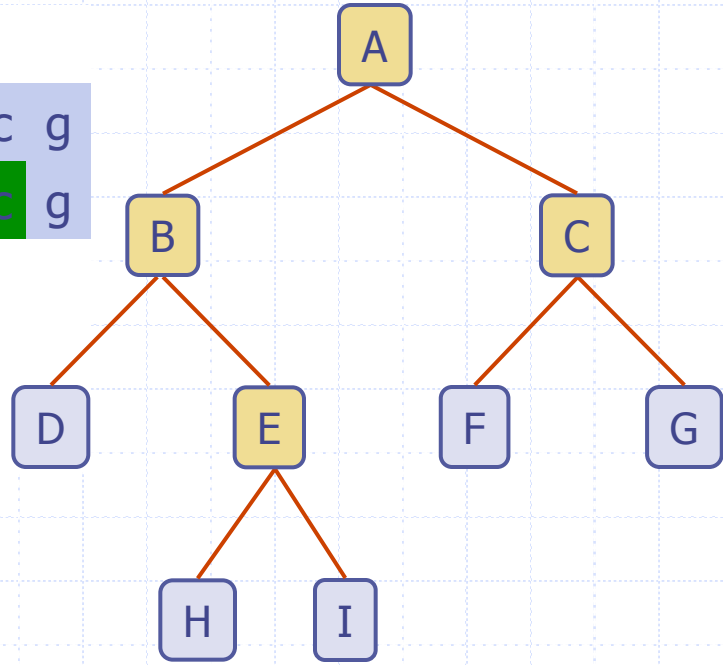


Building Tree from Pre- and In-Order Traversals

- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree







Building Tree from Post- and In-Order Traversals

- Given the postorder and inorder traversals of a binary tree we can uniquely determine the tree
- The last node visited in the postorder traversal is the root of the binary tree

Postorder	Inorder
d h i e b f g c a	d b h e i a f c g

Building Tree from Post- and In-Order Traversals

- Given the postorder and inorder traversals of a binary tree we can uniquely determine the tree
- The last node visited in the postorder traversal is the root of the binary tree

Postorder

d h i e b f g c a

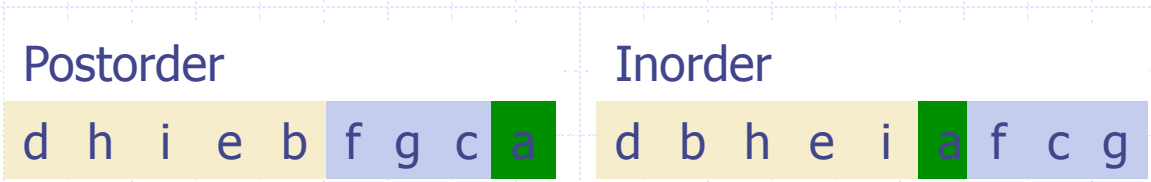
Inorder

d b h e i a f c g

A

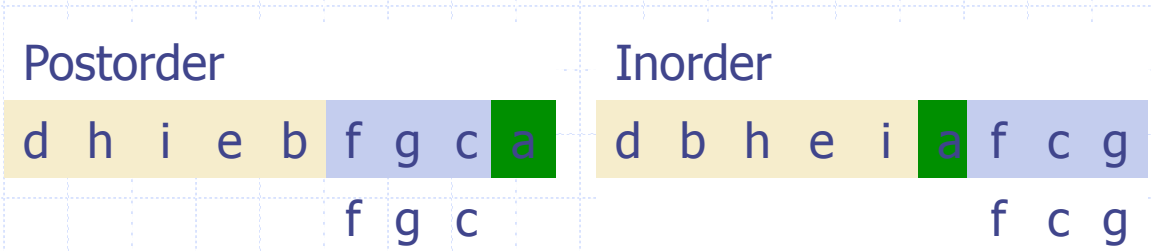
Building Tree from Post- and In-Order Traversals

- Given the postorder and inorder traversals of a binary tree we can uniquely determine the tree
- The last node visited in the postorder traversal is the root of the binary tree



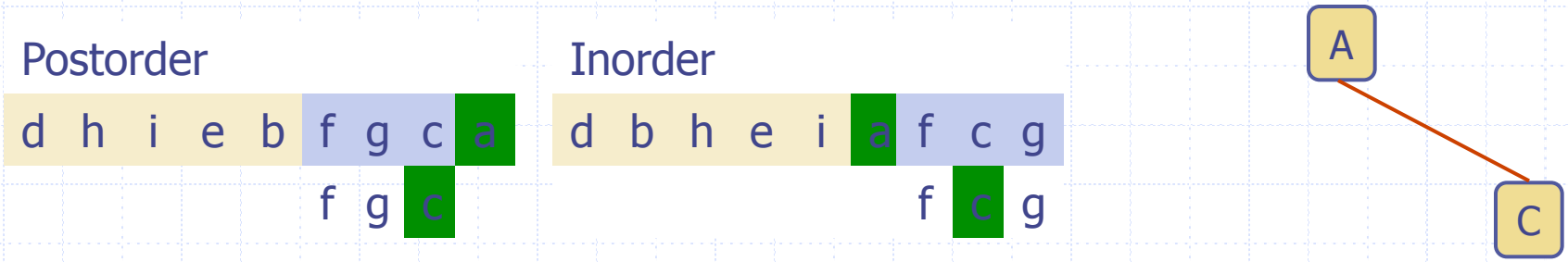
Building Tree from Post- and In-Order Traversals

- Given the postorder and inorder traversals of a binary tree we can uniquely determine the tree
- The last node visited in the postorder traversal is the root of the binary tree



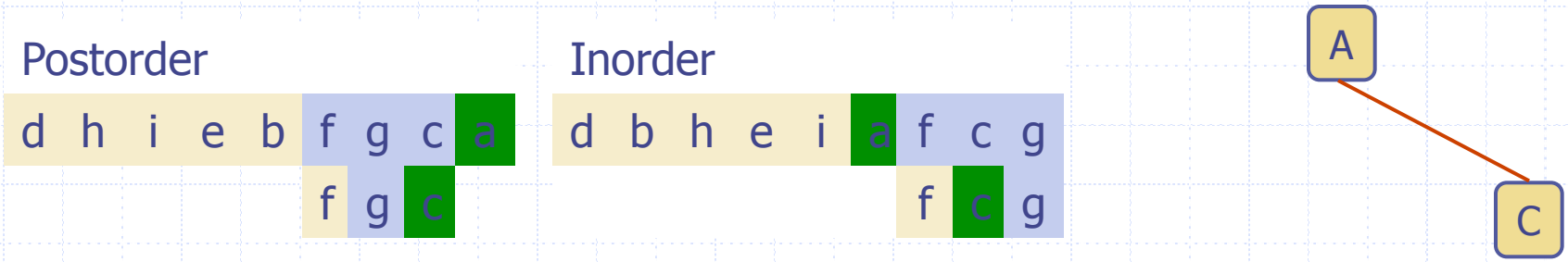
Building Tree from Post- and In-Order Traversals

- Given the postorder and inorder traversals of a binary tree we can uniquely determine the tree
- The last node visited in the postorder traversal is the root of the binary tree



Building Tree from Post- and In-Order Traversals

- Given the postorder and inorder traversals of a binary tree we can uniquely determine the tree
- The last node visited in the postorder traversal is the root of the binary tree



Building Tree from Post- and In-Order Traversals

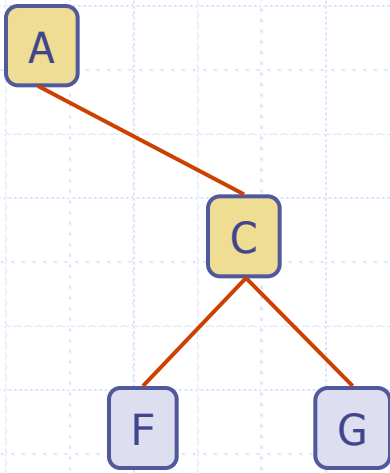
- Given the postorder and inorder traversals of a binary tree we can uniquely determine the tree
- The last node visited in the postorder traversal is the root of the binary tree

Postorder

d	h	i	e	b	f	g	c	a
					f	g	c	

Inorder

d	b	h	e	i	a	f	c	g
						f	c	g



Building Tree from Post- and In-Order Traversals

- Given the postorder and inorder traversals of a binary tree we can uniquely determine the tree
- The last node visited in the postorder traversal is the root of the binary tree

Postorder

d h i e b f g c a

f g c

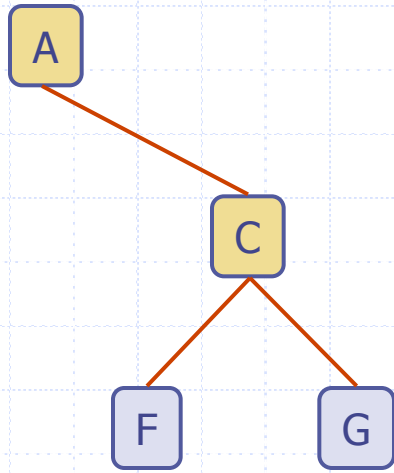
d h i e b

Inorder

d b h e i a f c g

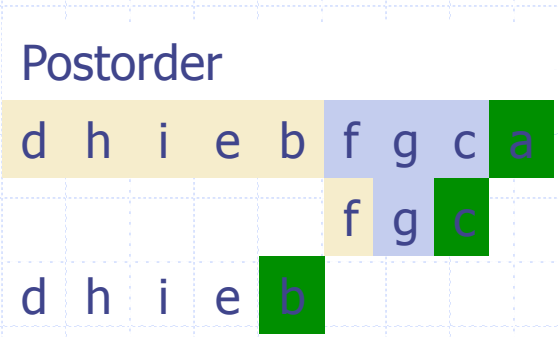
f c g

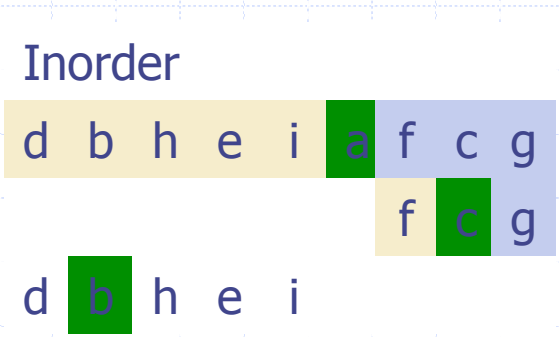
d b h e i

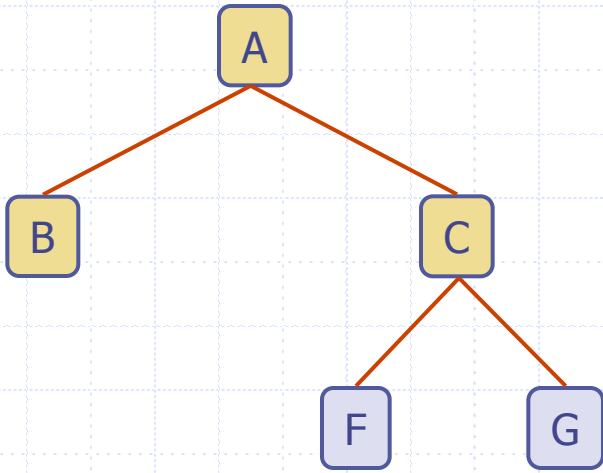


Building Tree from Post- and In-Order Traversals

- Given the postorder and inorder traversals of a binary tree we can uniquely determine the tree
- The last node visited in the postorder traversal is the root of the binary tree







Building Tree from Post- and In-Order Traversals

- Given the postorder and inorder traversals of a binary tree we can uniquely determine the tree
- The last node visited in the postorder traversal is the root of the binary tree

Postorder

d h i e b f g c a

f g c

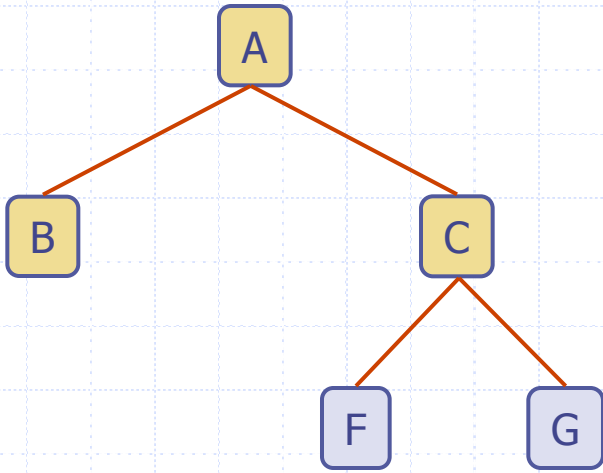
d h i e b

Inorder

d b h e i a f c g

f c g

d b h e i



- and so on..

Building Tree from Pre- and Post-Order Traversals

- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

Preorder	Postorder
a b d e h i c f g	d h i e b f g c a

Building Tree from Pre- and Post-Order Traversals

- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

Preorder

a b d e h i c f g

Postorder

d h i e b f g c a

A

Building Tree from Pre- and Post-Order Traversals

- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

Preorder

a b d e h i c f g

Postorder

d h i e b f g c a

A

Building Tree from Pre- and Post-Order Traversals

- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

Preorder

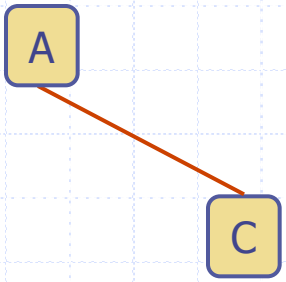
a b d e h i c f g

c f g

Postorder

d h i e b f g c a

f g c



Building Tree from Pre- and Post-Order Traversals

- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

Preorder

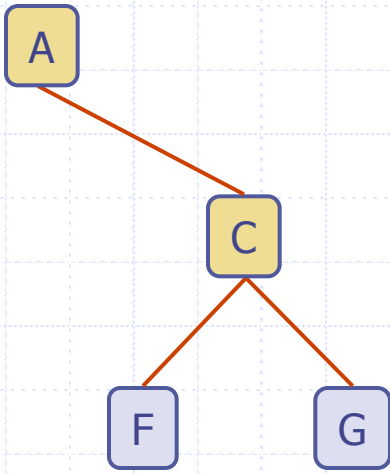
a b d e h i c f g

c f g

Postorder

d h i e b f g c a

f g c



Building Tree from Pre- and Post-Order Traversals

- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

Preorder

a b d e h i c f g

c f g

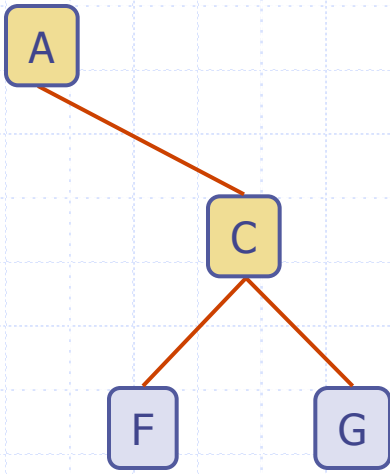
b d e h i

Postorder

d h i e b f g c a

f g c

d h i e b



Building Tree from Pre- and Post-Order Traversals

- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

Preorder

a b d e h i c f g

c f g

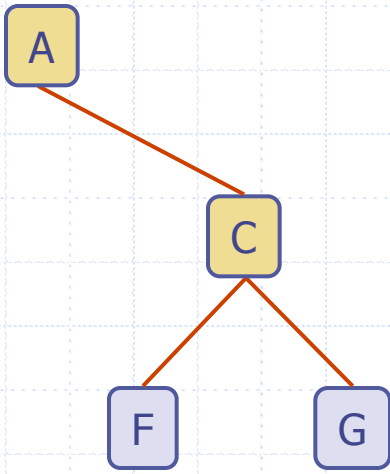
b d e h i

Postorder

d h i e b f g c a

f g c

d h i e b



Building Tree from Pre- and Post-Order Traversals

- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

Preorder

a b d e h i c f g

c f g

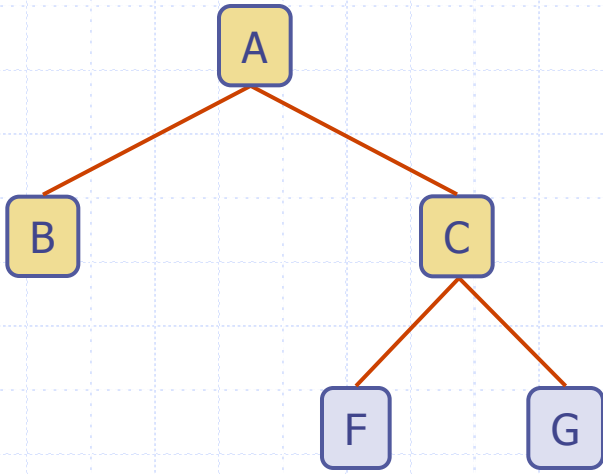
b d e h i

Postorder

d h i e b f g c a

f g c

d h i e b



Building Tree from Pre- and Post-Order Traversals

- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

Preorder

a b d e h i c f g

c f g

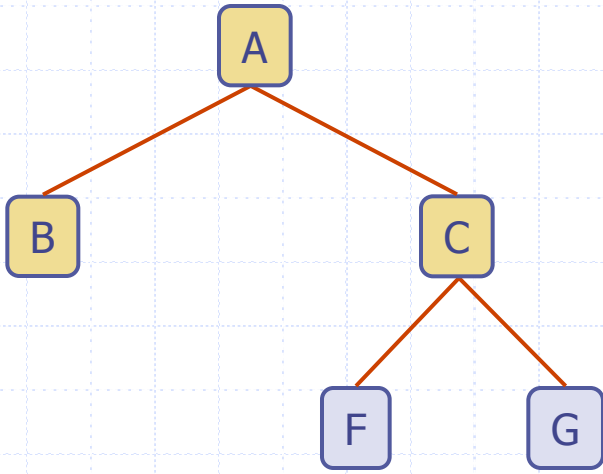
b d e h i

Postorder

d h i e b f g c a

f g c

d h i e b



Building Tree from Pre- and Post-Order Traversals

- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

Preorder

a b d e h i c f g

c f g

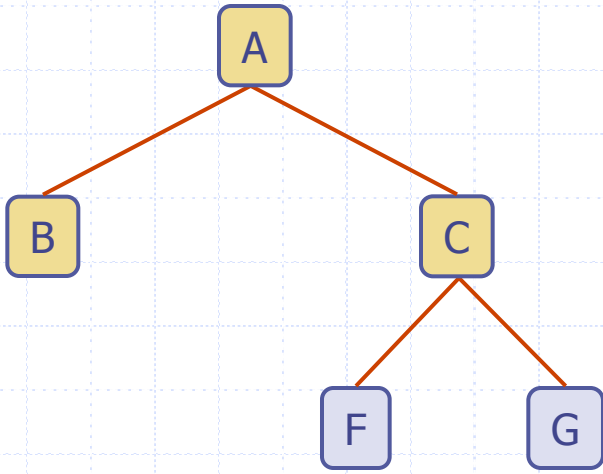
b d e h i

Postorder

d h i e b f g c a

f g c

d h i e b



Building Tree from Pre- and Post-Order Traversals

- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

Preorder

a b d e h i c f g

c f g

b d e h i

e h i

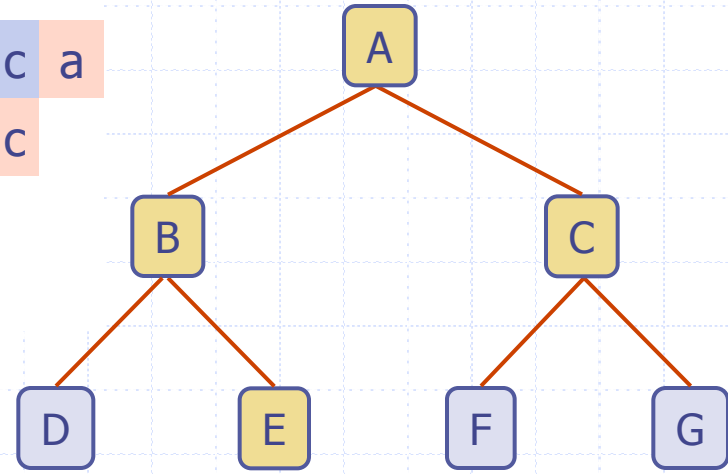
Postorder

d h i e b f g c a

f g c

d h i e b

h i e



Building Tree from Pre- and Post-Order Traversals

- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

Preorder

a b d e h i c f g

c f g

b d e h i

e h i

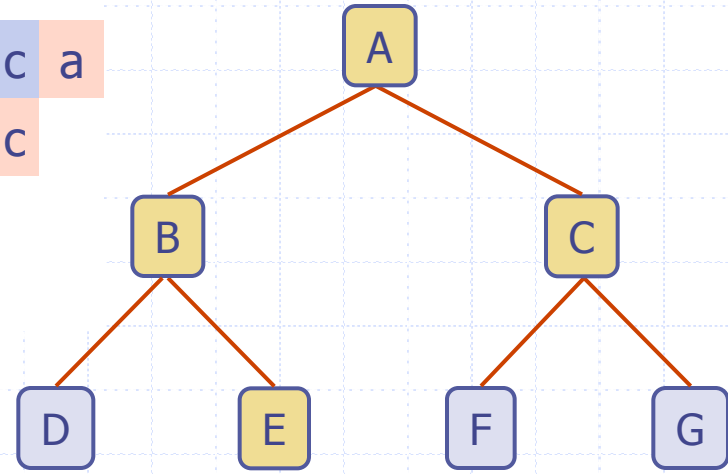
Postorder

d h i e b f g c a

f g c

d h i e b

h i e



Building Tree from Pre- and Post-Order Traversals

- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

Preorder

a b d e h i c f g

c f g

b d e h i

e h i

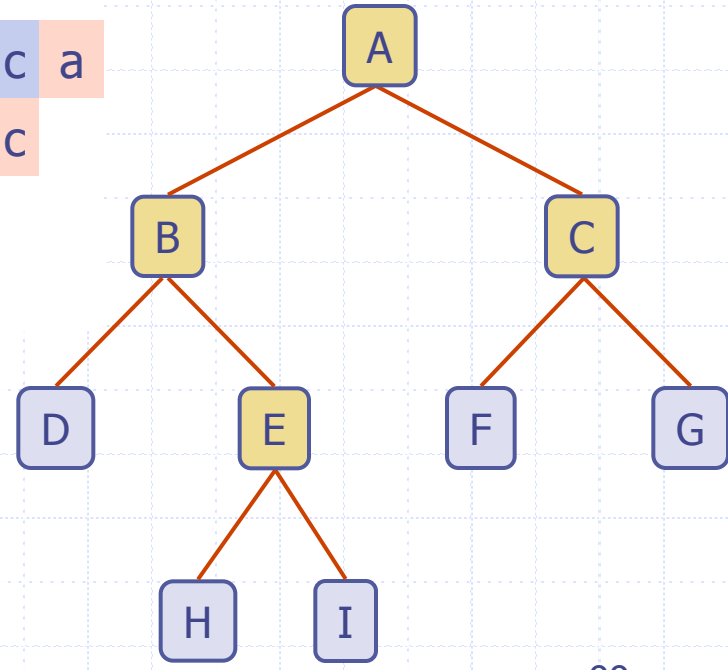
Postorder

d h i e b f g c a

f g c

d h i e b

h i e



Building Tree from Pre- and Post-Order Traversals

- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

Preorder	Postorder
a b e i c	i e b c a

Building Tree from Pre- and Post-Order Traversals

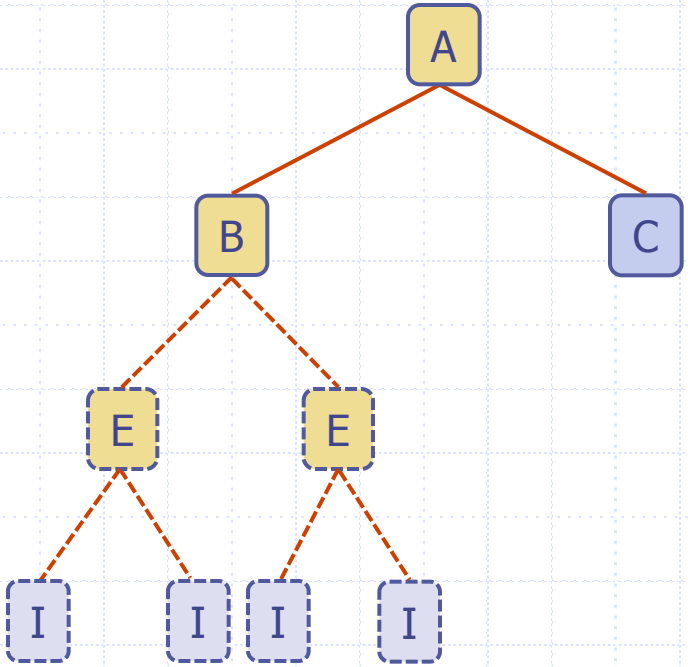
- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

Preorder

a b e i c

Postorder

i e b c a

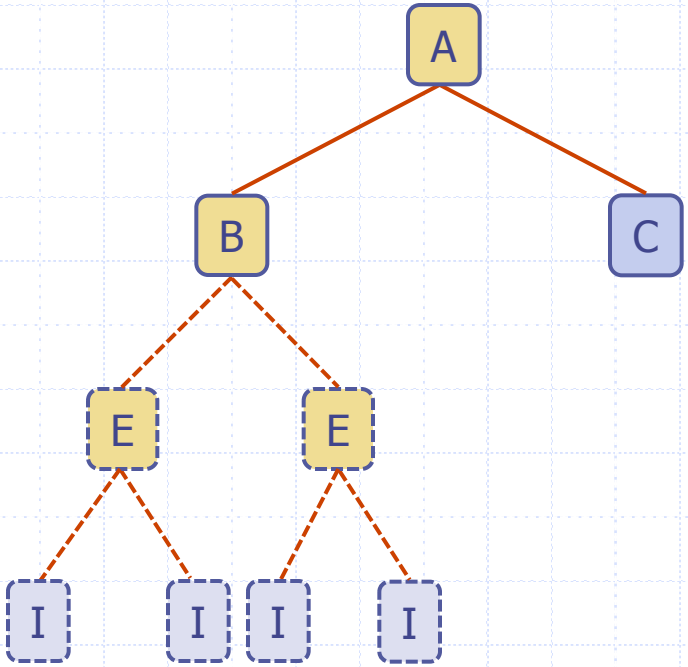


Building Tree from Pre- and Post-Order Traversals

- Given the pre and postorder traversal of a binary tree, can we uniquely reconstruct the tree?

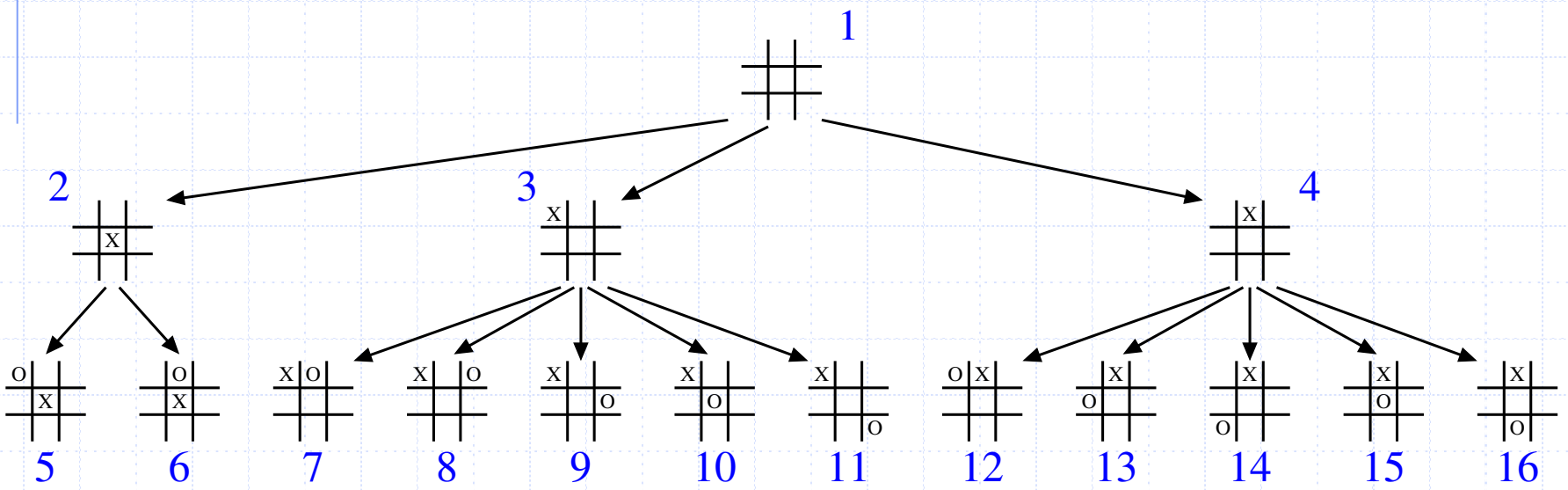
Preorder	Postorder
a b e i c	i e b c a

Only if the internal nodes in a binary tree have exactly two children



Breadth First Traversal

- Visit all positions at depth d, before visiting the positions at depth d+1



Let T be a binary tree with n nodes and let D be the sum of the depths of all the external nodes of T. Show that if T has the minimum number of external nodes possible, then D is O(n) and if T has the maximum number of external nodes possible, then D is O(nlogn)

Let T be an ordered tree with n >1. Is it possible that the preorder traversal of T visits the nodes in the same order as the postorder traversal of T? If so given an example, otherwise explain why. Is it possible that the preorder traversal of T visits the nodes in the reverse order of postorder traversal?

Give the pseudocode for a nonrecursive method for performing an inorder traversal of a binary tree in linear time.

Let T be a tree with n positions. Define the lowest common ancestor(LCA) between two positions p and q as the lowest point in T that has both p and q as descendants. Given two positions p and q, describe an efficient algorithm for finding the LCA of p and q. What is the running time of your algorithm?