ساختمان داده ها و الگوریتم ها(CE203)

جلسات چهاردهم و پانزدهم: درخت

سجاد شیرعلی شهرضا

پاییز 1400

شنبه 22 و دوشنبه 24 آبان 1400

Based on Washington University CSE 373 Slides from Autumn 2020 by Hunter Schafer

جلسات چهاردهم و پانزدهم: درخت

شنبه، 18 و دوشنبه 20 اردیبهشت 1400

اطلاع رسانی

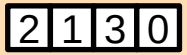
- بخش مرتبط کتاب برای این جلسه: 10.4
- یادآوری نظرسنجی سوم: شنبه، 29 آبان، ساعت 8 صبح
- امتحان میان ترم: دوشنبه هفته آینده، 1 آذر 1400، در ساعت کلاس

درخت

Data Structures

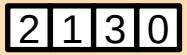
- **Data structure**
  - Organization or format for storing or managing data
  - Concrete realization of an abstract data type
- Operations
  - Always a tradeoff: some operations more efficient, some less, for any data structure
  - Choose efficient data structure for operations of concern

Example Data Structures

Data Structure	add(val v)	get(int i)
Array 		
Linked List 		

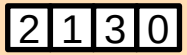
```
add(v): append v
get(i): return element at position i
```

Example Data Structures

Data Structure	add(val v)	get(int i)
Array 	$O(n)$	
Linked List 		

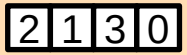
```
add(v): append v
get(i): return element at position i
```

Example Data Structures

Data Structure	add(val v)	get(int i)
Array 	$O(n)$	$O(1)$
Linked List 		

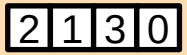
```
add(v): append v
get(i): return element at position i
```

Example Data Structures

Data Structure	add(val v)	get(int i)
Array 	$O(n)$	$O(1)$
Linked List 	$O(1)$	

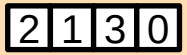
```
add(v): append v
get(i): return element at position i
```

Example Data Structures

Data Structure	add(val v)	get(int i)
Array 	$O(n)$	$O(1)$
Linked List 	$O(1)$	$O(n)$

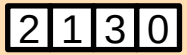
```
add(v): append v
get(i): return element at position i
```

Example Data Structures

Data Structure	add(val v)	get(int i)	contains(val v)
Array 	$O(n)$	$O(1)$	
Linked List 	$O(1)$	$O(n)$	

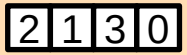
```
add(v): append v
get(i): return element at position i
contains(v): return true if contains v
```

Example Data Structures

Data Structure	add(val v)	get(int i)	contains(val v)
Array 	$O(n)$	$O(1)$	$O(n)$
Linked List 	$O(1)$	$O(n)$	

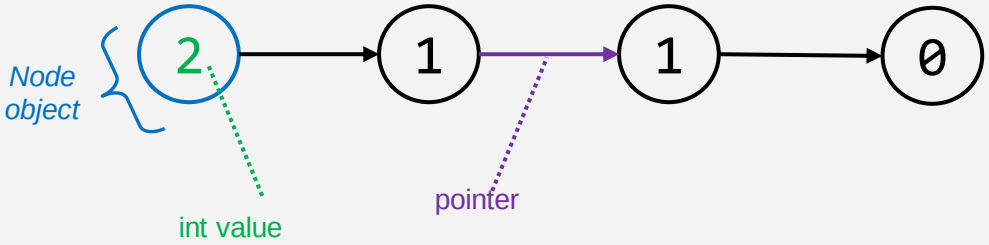
```
add(v): append v
get(i): return element at position i
contains(v): return true if contains v
```

Example Data Structures

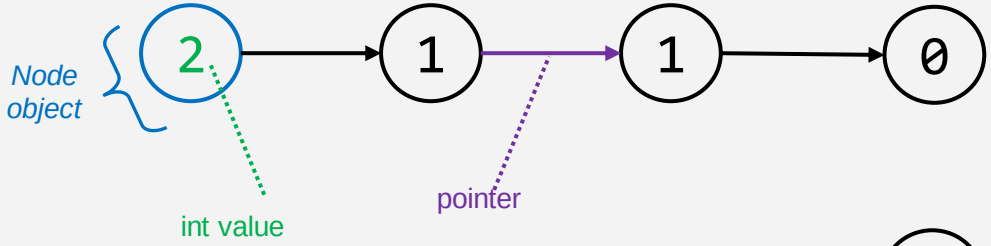
Data Structure	add(val v)	get(int i)	contains(val v)
Array 	$O(n)$	$O(1)$	$O(n)$
Linked List 	$O(1)$	$O(n)$	$O(n)$

```
add(v): append v
get(i): return element at position i
contains(v): return true if contains v
```

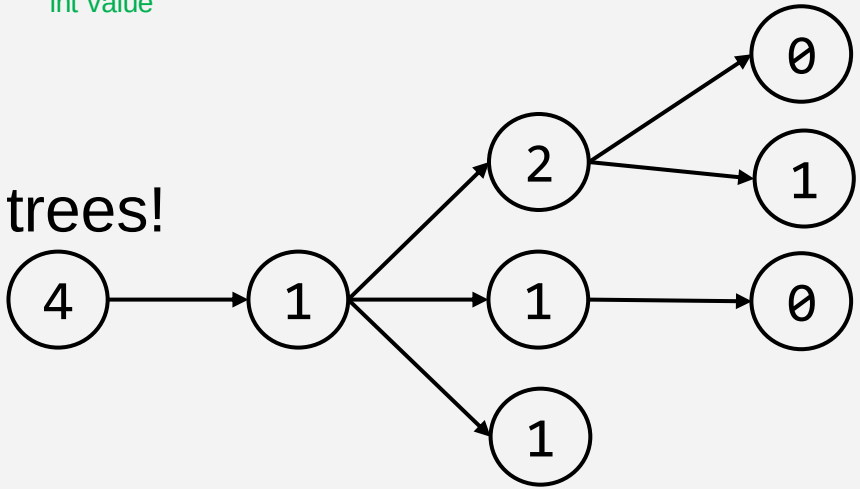
Singly linked list:



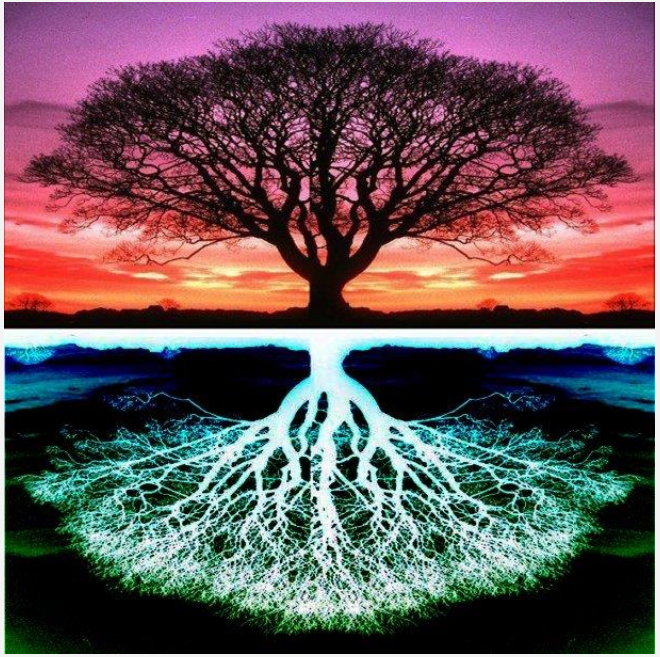
Singly linked list:



Today: trees!



Trees



In CS, we draw trees “upside down”

Tree Overview

Tree: data structure with nodes, similar to linked list

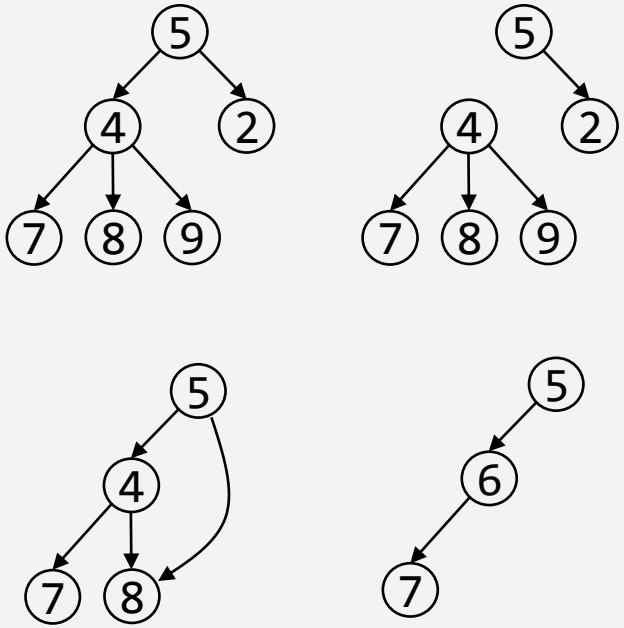
- Each node may have **zero or more *successors*** (**children**)
- Each node has **exactly one *predecessor*** (**parent**) except the *root*, which has none
- All nodes are reachable from *root*

Tree Overview

Tree: data structure with nodes, similar to linked list

- Each node may have **zero or more *successors*** (**children**)
- Each node has **exactly one *predecessor*** (**parent**) except the *root*, which has none
- All nodes are reachable from *root*

A tree or not a tree?

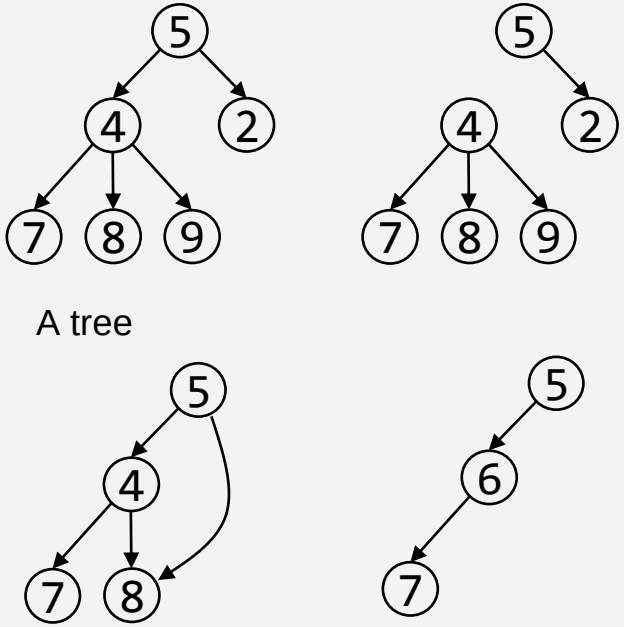


Tree Overview

Tree: data structure with nodes, similar to linked list

- Each node may have **zero or more *successors*** (**children**)
- Each node has **exactly one *predecessor*** (**parent**) except the *root*, which has none
- All nodes are reachable from *root*

A tree or not a tree?

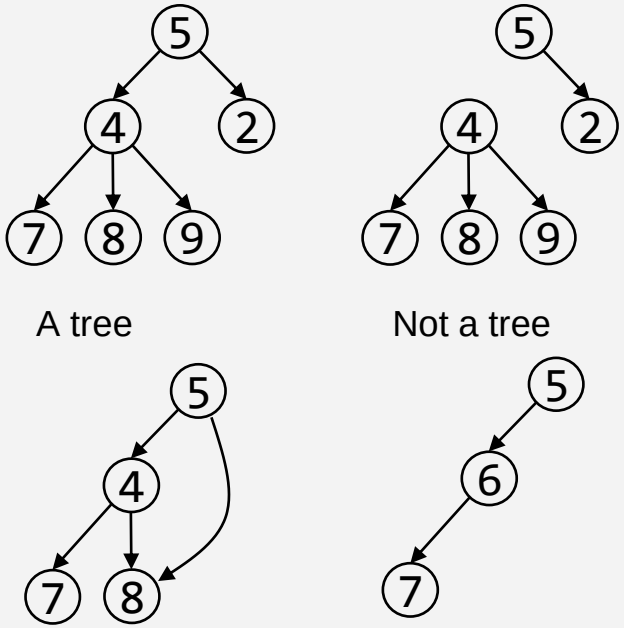


Tree Overview

Tree: data structure with nodes, similar to linked list

- Each node may have **zero or more *successors*** (**children**)
- Each node has **exactly one *predecessor*** (**parent**) except the *root*, which has none
- All nodes are reachable from *root*

A tree or not a tree?

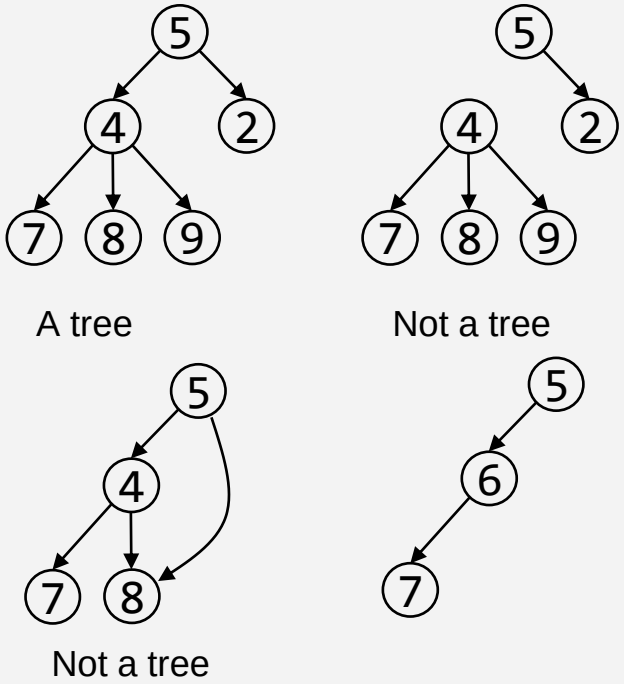


Tree Overview

Tree: data structure with nodes, similar to linked list

- Each node may have **zero or more *successors*** (**children**)
- Each node has **exactly one *predecessor*** (**parent**) except the *root*, which has none
- All nodes are reachable from *root*

A tree or not a tree?

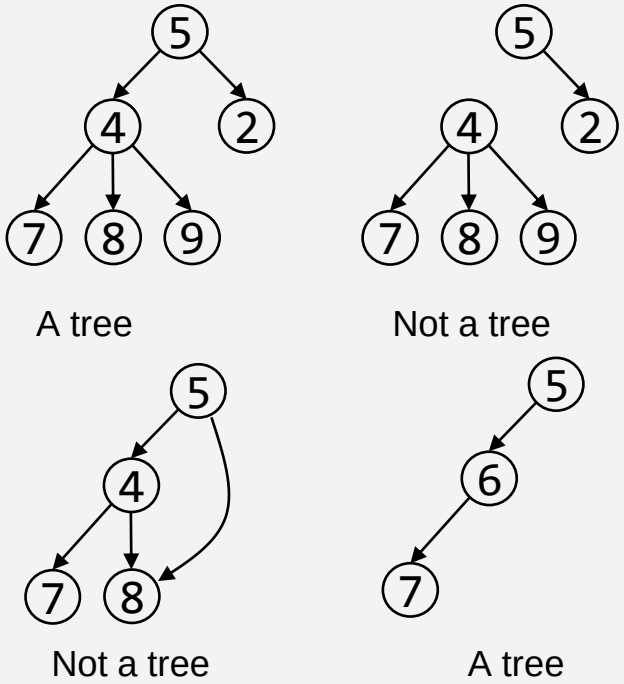


Tree Overview

Tree: data structure with nodes, similar to linked list

- Each node may have **zero or more *successors*** (**children**)
- Each node has **exactly one *predecessor*** (**parent**) except the *root*, which has none
- All nodes are reachable from *root*

A tree or not a tree?

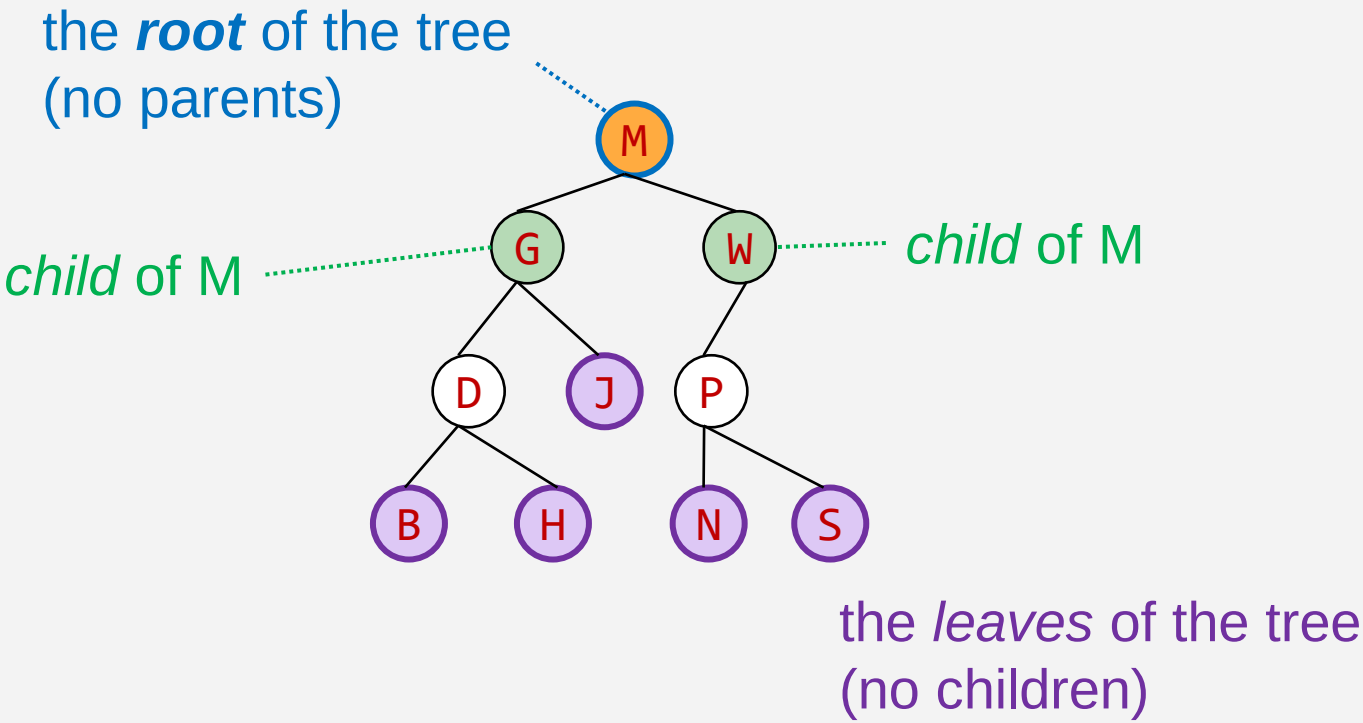


سوال؟

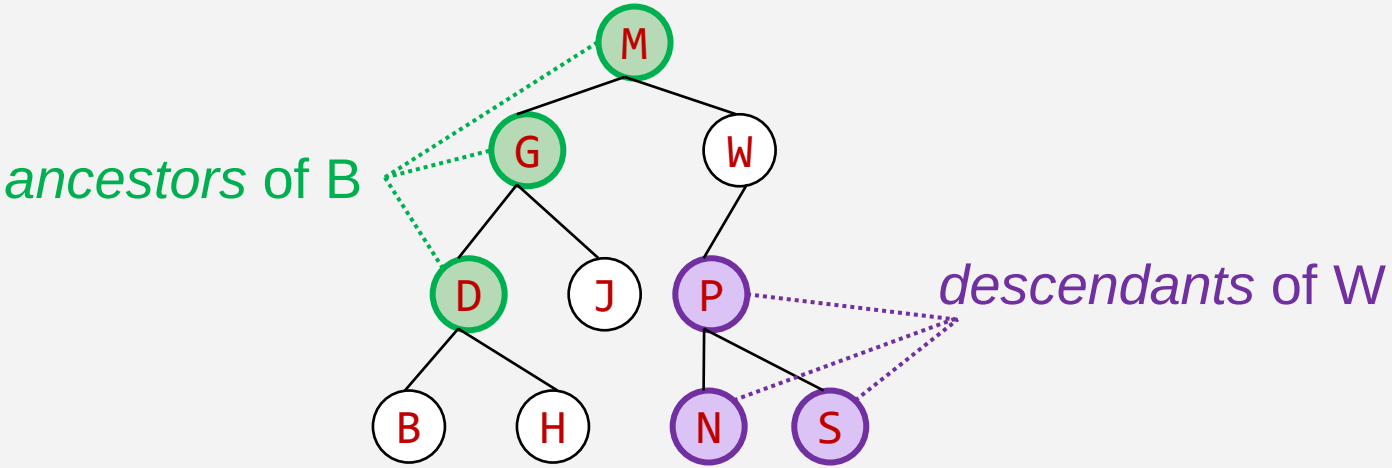
تعریف درخت

تعریف برخی مفاهیم مورد استفاده در مورد درخت

Tree Terminology: Parent, Child, Leaves, Root

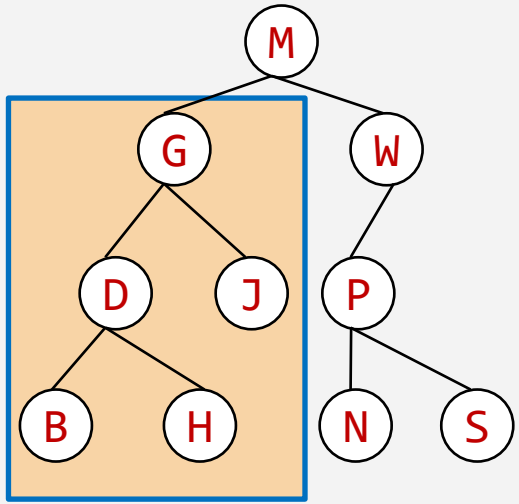


Tree Terminology: Ancestors and Descendants



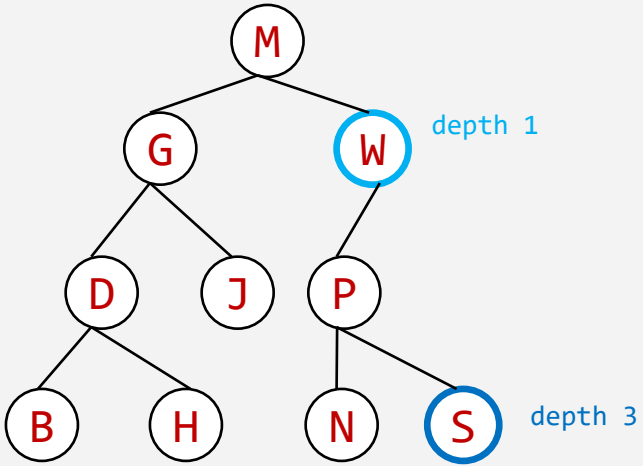
Tree Terminology: Subtree

subtree of M



Tree Terminology: Depth & Height

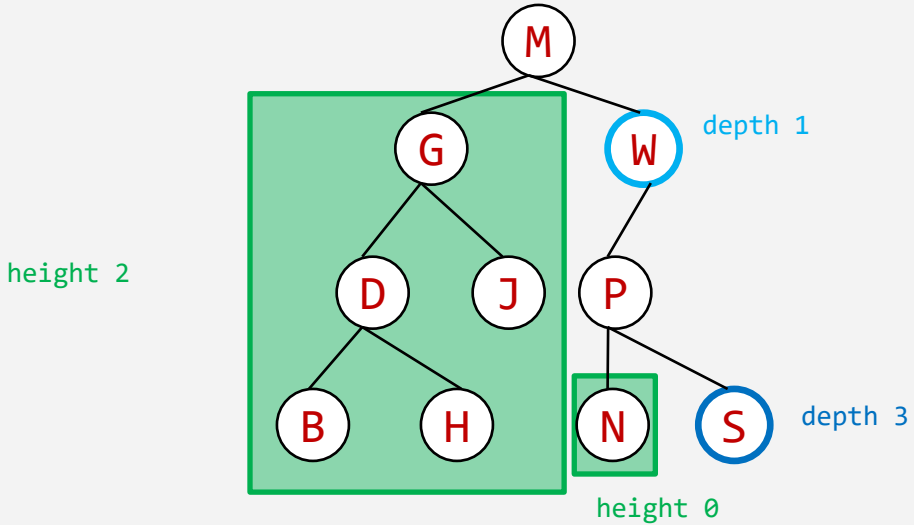
A node's ***depth*** is the length of the path to the root.



Tree Terminology: Depth & Height

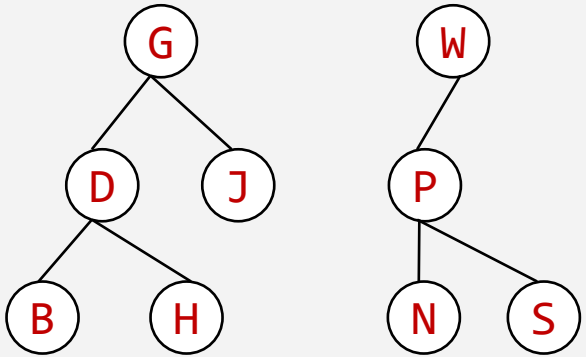
A node's ***depth*** is the length of the path to the root.

A tree's (or subtree's) ***height*** is the length of the longest path from the root to a leaf.



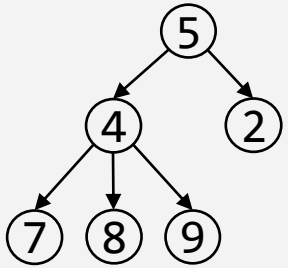
Tree Terminology: Forest

Multiple trees: a ***forest***



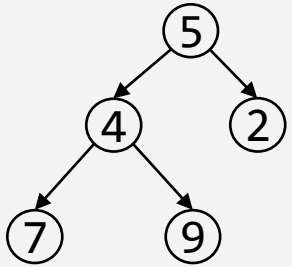
General vs. Binary Trees

General tree: every node can have an arbitrary number of children



General tree

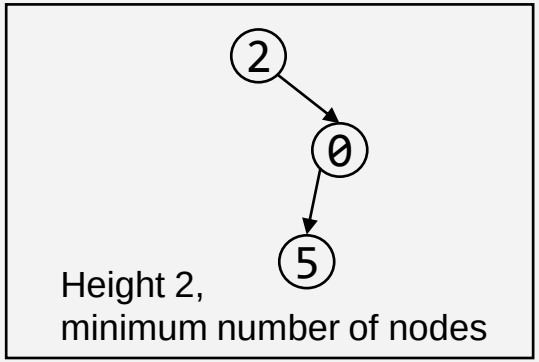
Binary tree: at most two children, called *left* and *right*



Binary tree

…often "tree" means binary tree

Special kinds of binary trees



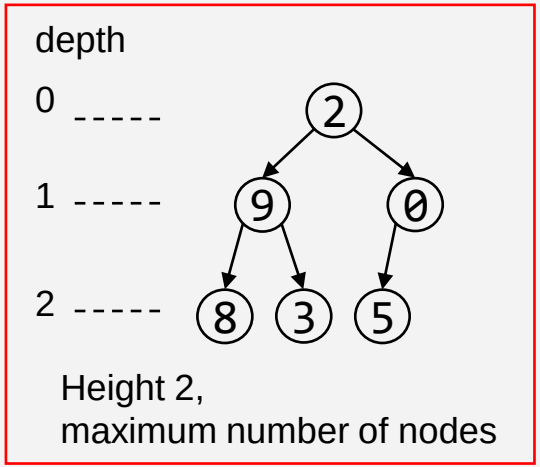
Max # of nodes at depth d: 2^d

If height of tree is h:

min # of nodes: h + 1

max #of nodes: (Perfect tree)

$2^0 + \ldots + 2^h = 2^{h+1} - 1$



Complete binary tree

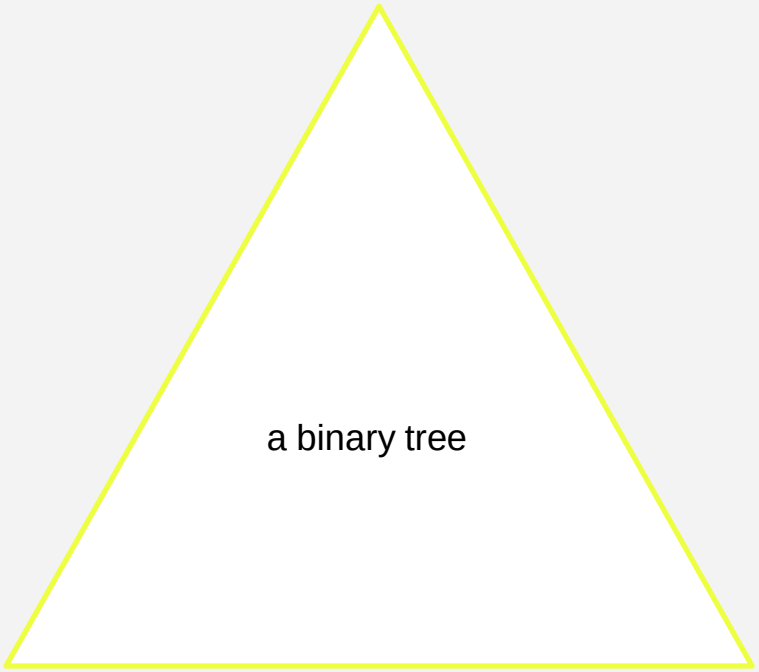
Every level, except last, is completely filled, nodes on bottom level as far left as possible. No holes.

سوال؟

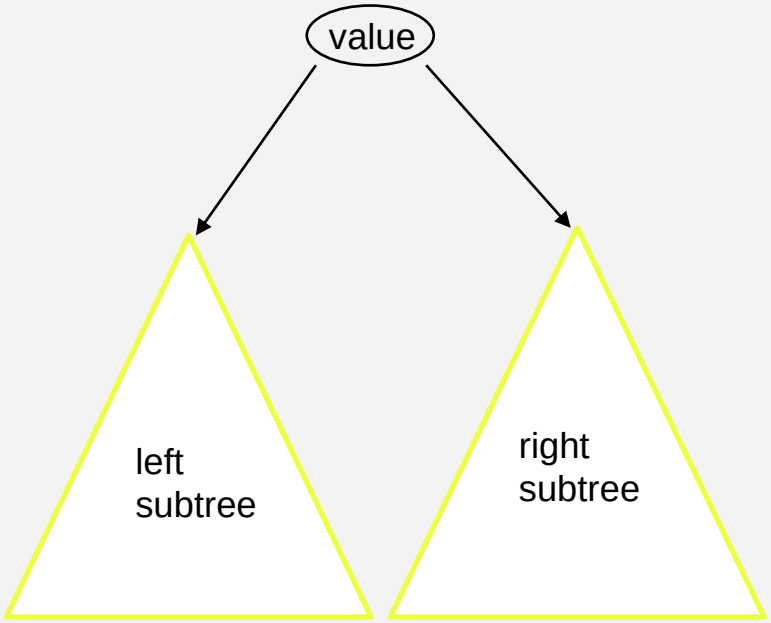
پردازش درخت

انجام عملیات بر روی درخت

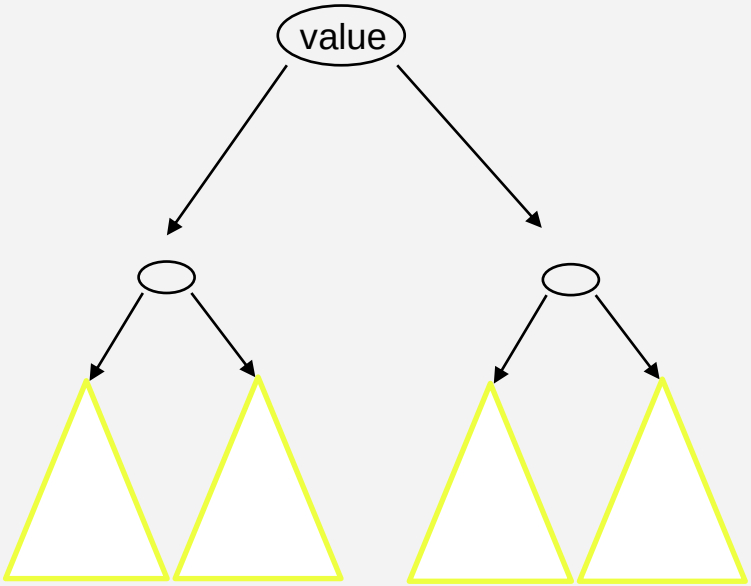
Trees are recursive



Trees are recursive

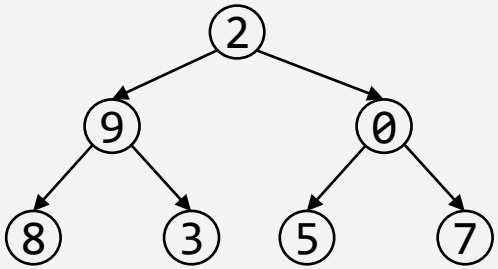


Trees are recursive



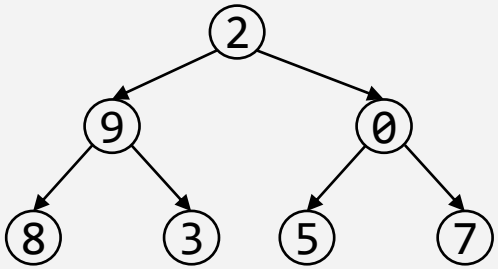
Trees are recursive

Binary Tree



Trees are recursive

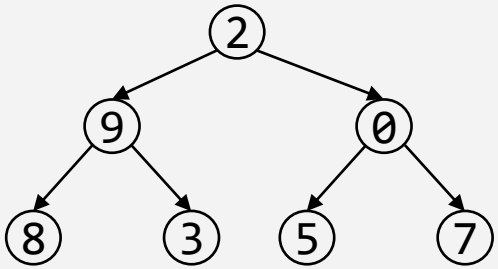
Binary Tree



Left subtree (a binary tree)

Trees are recursive

Binary Tree



Left subtree (a binary tree)

Right subtree (also a binary tree)

Trees are recursive

A binary tree is either **null**

or an object consisting of a value, a left binary tree, and a right binary tree.

A Recipe for Recursive Functions

Base case:

If the input is “easy,” just solve the problem directly.

Recursive case:

Get a smaller part of the input (or several parts).
Call the function on the smaller value(s).
Use the recursive result to build a solution for the full input.

A Recipe for Recursive Functions on Binary Trees

Base case:

If the input is “easy,” just solve the problem directly.

Recursive case:

Get a smaller part of the input (or several parts).
Call the function on the smaller value(s).
Use the recursive result to build a solution for the full input.

A Recipe for Recursive Functions on Binary Trees

an empty tree (null), or possibly a leaf

Base case:

If the input is "easy," just solve the problem directly.

Recursive case:

Get a smaller part of the input (or several parts).
Call the function on the smaller value(s).
Use the recursive result to build a solution for the full input.

A Recipe for Recursive Functions on Binary Trees

Base case:

an empty tree (null), or possibly a leaf

If the input is "easy," just solve the problem directly.

Recursive case:

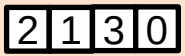
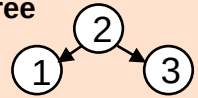
~~Get a smaller part of the input (or several parts).~~

Call the function on ~~the smaller value(s).~~ each subtree

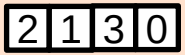
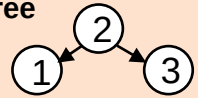
Use the recursive result to build a solution for the full input.

سوال؟

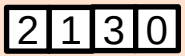
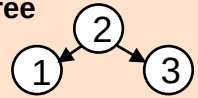
Comparing Searches

Data Structure	add(val v)	get(int i)	contains(val v)
Array 	$O(n)$	$O(1)$	$O(n)$
Linked List 	$O(1)$	$O(n)$	$O(n)$
Binary Tree 			

Comparing Searches

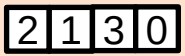
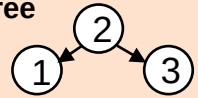
Data Structure	add(val v)	get(int i)	contains(val v)
Array 	$O(n)$	$O(1)$	$O(n)$
Linked List 	$O(1)$	$O(n)$	$O(n)$
Binary Tree 			$O(n)$

Comparing Searches

Data Structure	add(val v)	get(int i)	contains(val v)
Array 	$O(n)$	$O(1)$	$O(n)$
Linked List 	$O(1)$	$O(n)$	$O(n)$
Binary Tree 			$O(n)$

Node could be *anywhere* in tree

Comparing Searches

Data Structure	add(val v)	get(int i)	contains(val v)
Array 	$O(n)$	$O(1)$	$O(n)$
Linked List 	$O(1)$	$O(n)$	$O(n)$
Binary Tree 			$O(n)$

Node could be *anywhere* in tree

Binary search on arrays: O(log n)
Requires invariant: array sorted
…analogue for trees?
TO BE CONTINUED!
(in a future lecture)

سوال؟

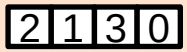
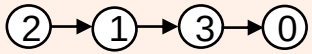
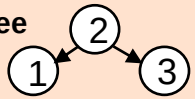
پیمایش درخت

پیمایش و ذخیره یک درخت

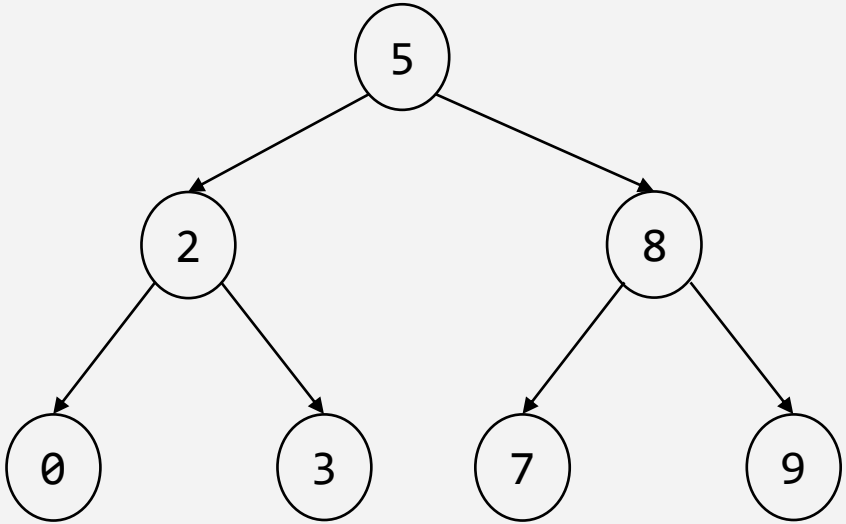
Iterate through data structure

Iterate: process elements of data structure

- Sum all elements
- Print each element
- …

Data Structure	Order to iterate
Array 	Forwards: 2, 1, 3, 0 Backwards: 0, 3, 1, 2
Linked List 	Forwards: 2, 1, 3, 0
Binary Tree 	**???**

Iterate through a tree



Discuss: What would a reasonable order be?

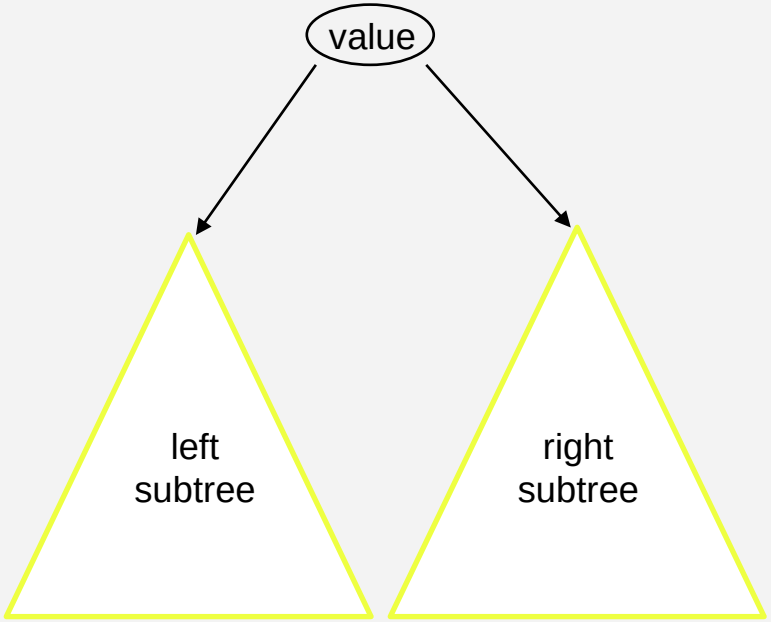
Tree traversals

- Iterating through tree is aka tree traversal
- Well-known recursive tree traversal algorithms:
  - Preorder
  - Inorder
  - Postorder
- Another, non-recursive: level order

پیمایش پیش ترتیب

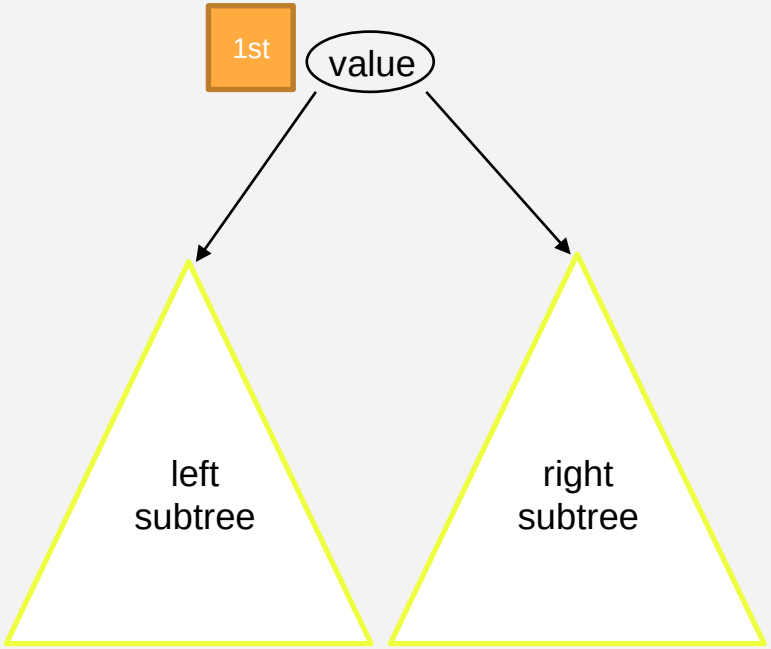
Preorder

"Pre:" process root before subtrees



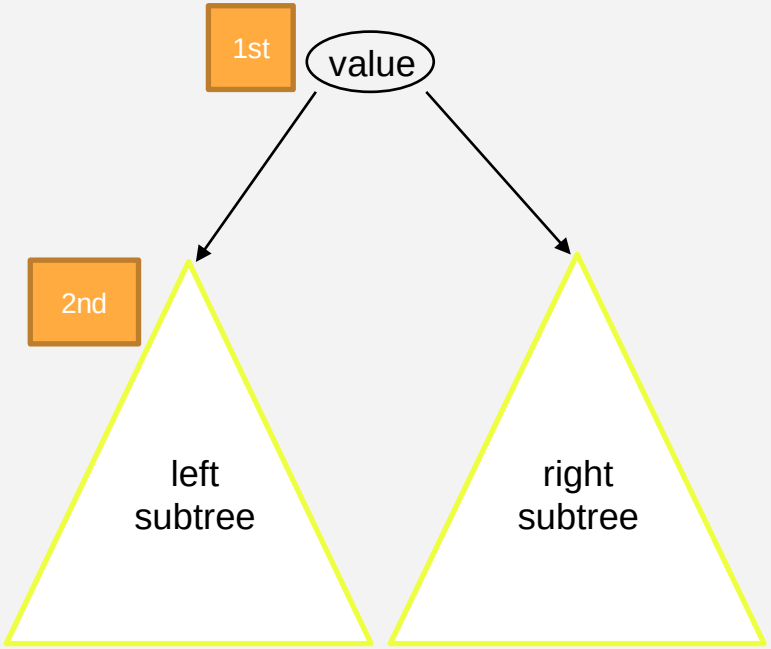
Preorder

"Pre:" process root before subtrees



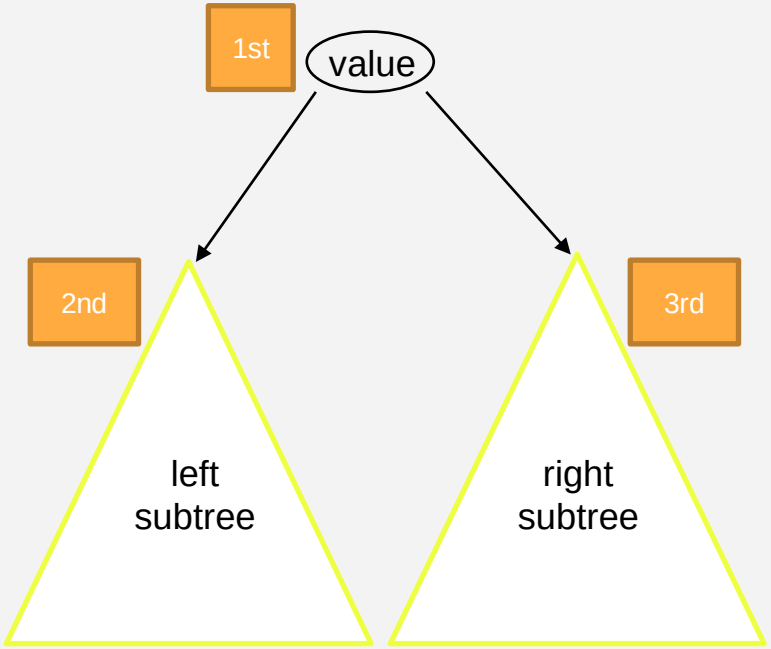
Preorder

"Pre:" process root before subtrees



Preorder

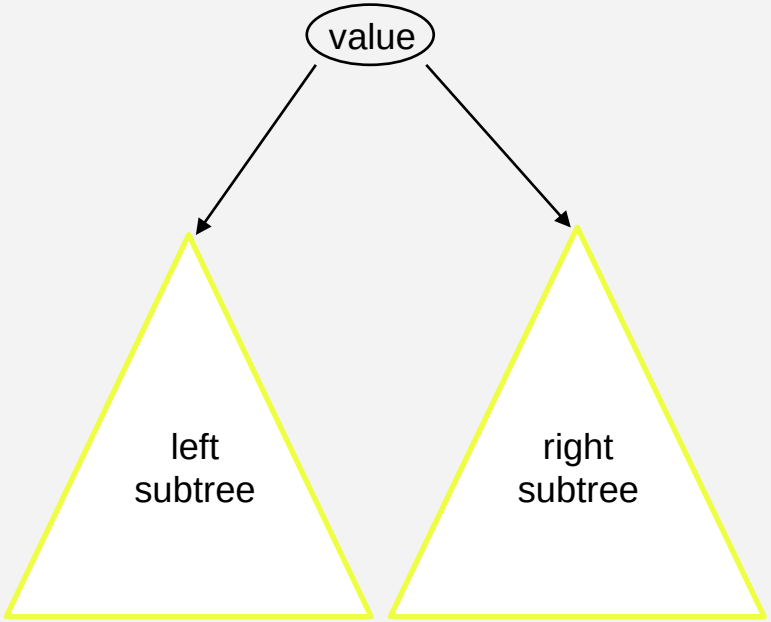
"Pre:" process root before subtrees



پیمایش میان ترتیب

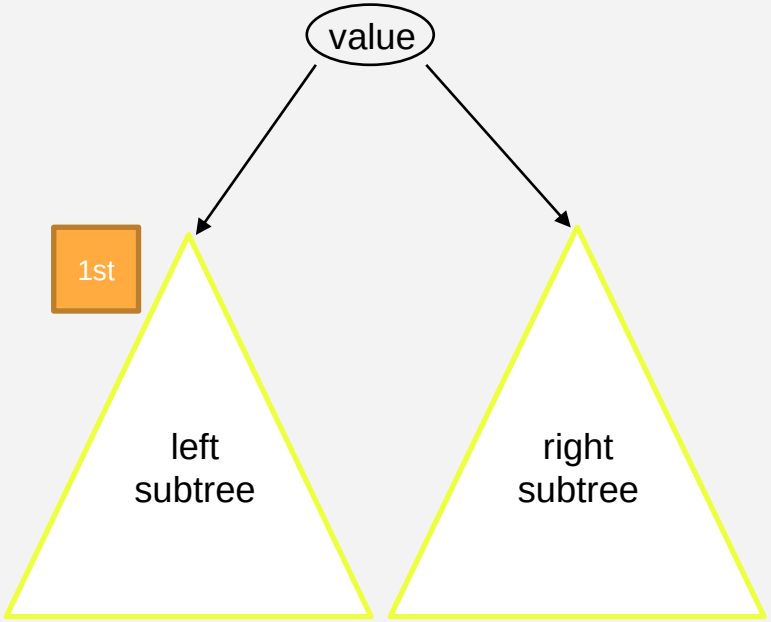
Inorder

"In:" process root in-between subtrees



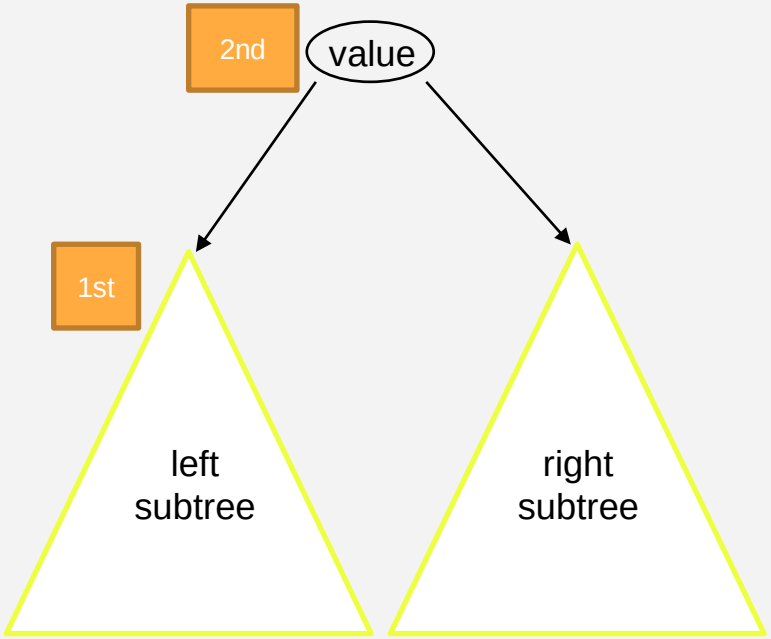
Inorder

"In:" process root in-between subtrees



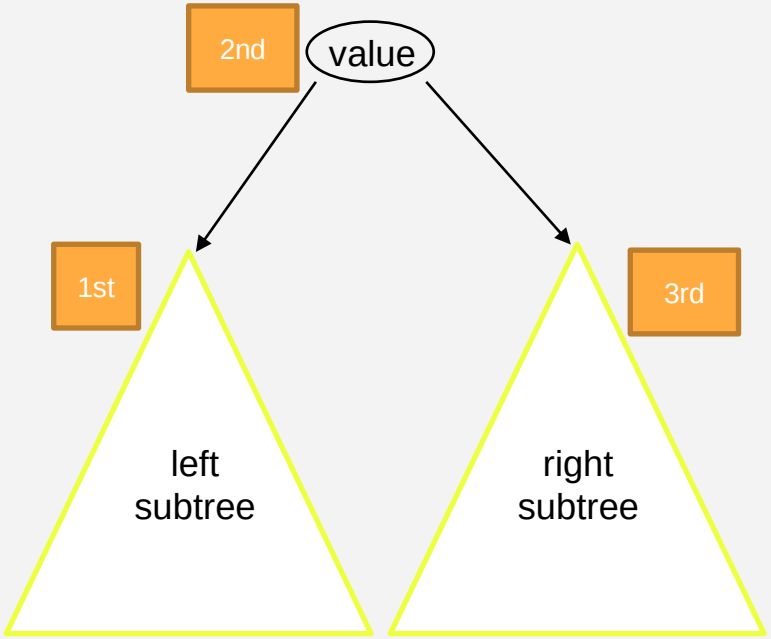
Inorder

"In:" process root in-between subtrees



Inorder

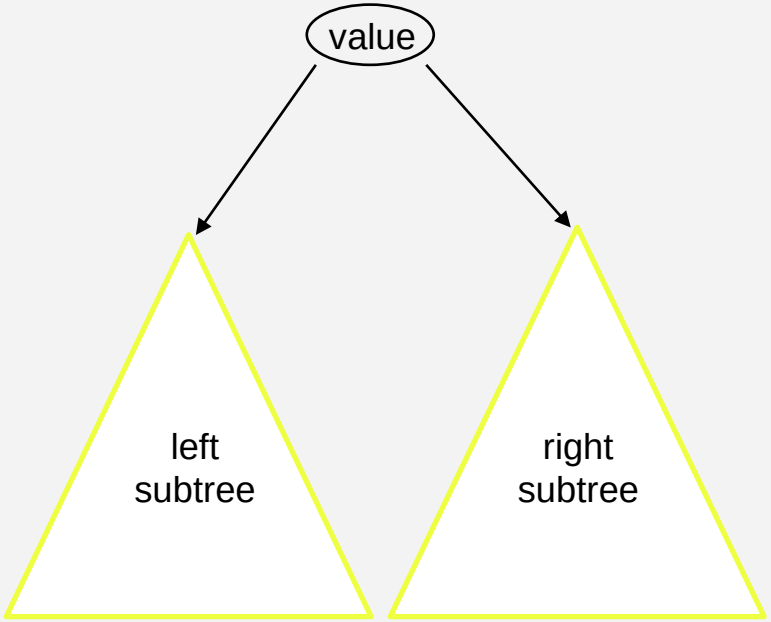
"In:" process root in-between subtrees



پیمایش پس ترتیب

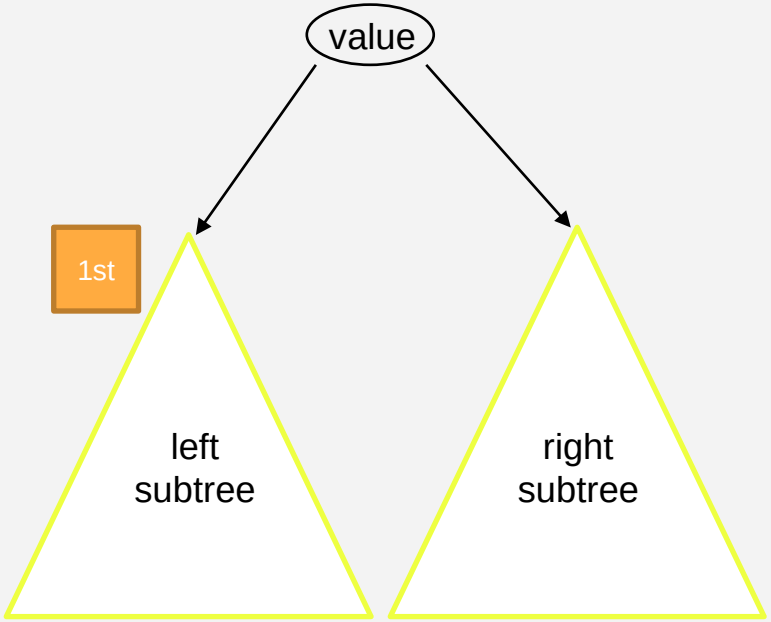
Postorder

"Post:" process root after subtrees



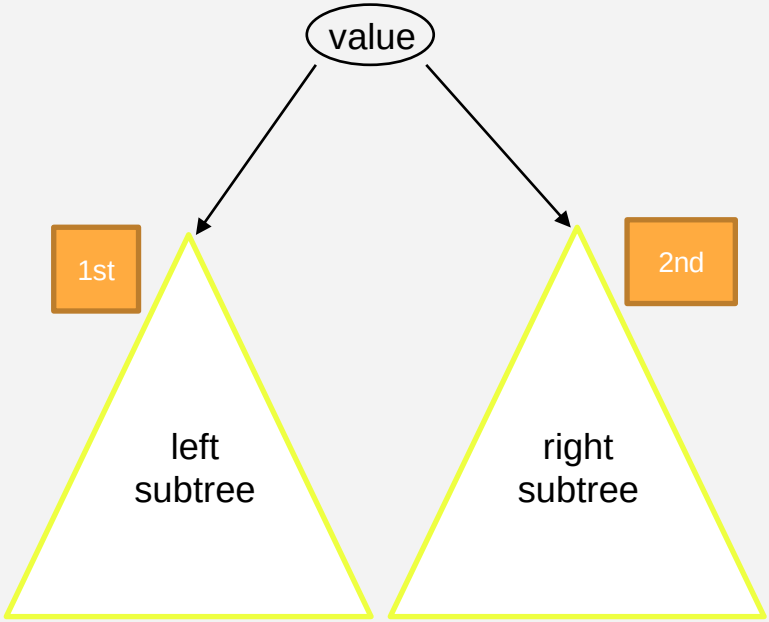
Postorder

"Post:" process root after subtrees



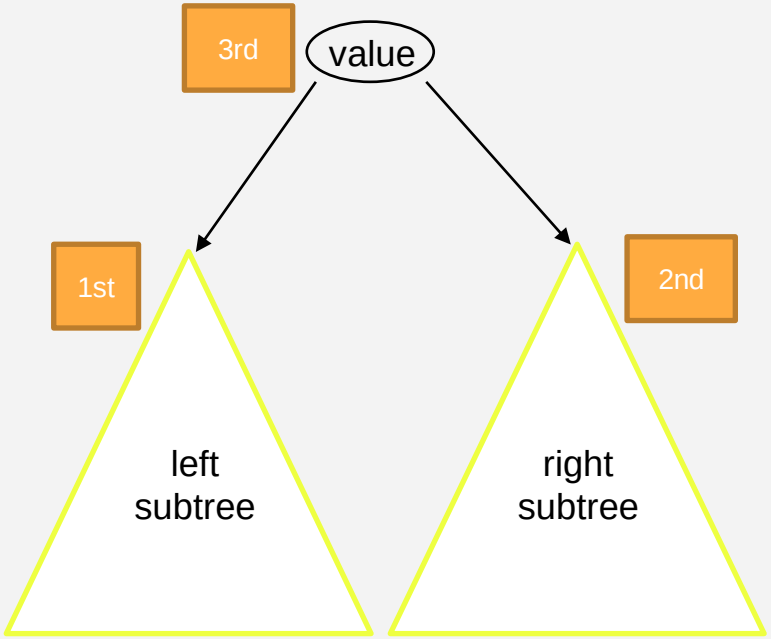
Postorder

"Post:" process root after subtrees



Postorder

"Post:" process root after subtrees



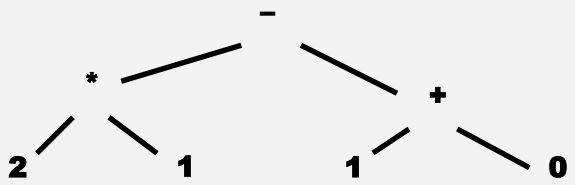
سوال؟

درخت عبارت

نمونه ای از کاربرد درخت و پیمایش آن

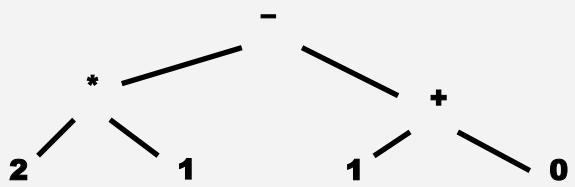
Syntax Trees

- Trees can represent (Java) expressions
- Expression: **2 * 1 – (1 + 0)**
- Tree:



پیمایش پیش ترتیب عبارت

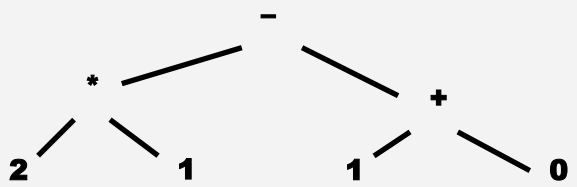
Traversals of expression tree



Preorder traversal

1. Visit the root
2. Visit the left subtree
3. Visit the right subtree

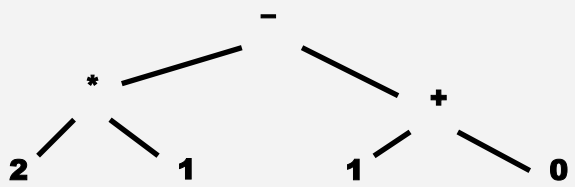
Traversals of expression tree



Preorder traversal -

1. Visit the root
2. Visit the left subtree
3. Visit the right subtree

Traversals of expression tree

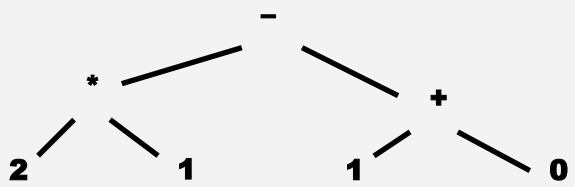


Preorder traversal

1. Visit the root
2. Visit the left subtree
3. Visit the right subtree

- *

Traversals of expression tree

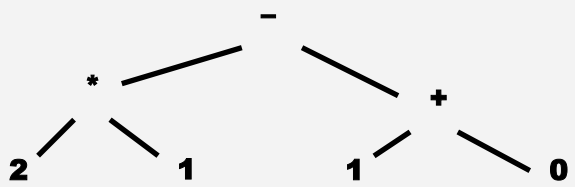


Preorder traversal

1. Visit the root
2. Visit the left subtree
3. Visit the right subtree

- * 2

Traversals of expression tree

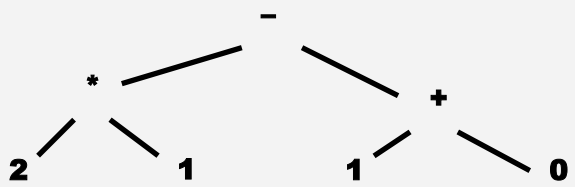


Preorder traversal

1. Visit the root
2. Visit the left subtree
3. Visit the right subtree

- * 2 1

Traversals of expression tree

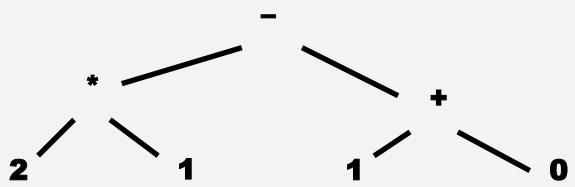


Preorder traversal

1. Visit the root
2. Visit the left subtree
3. Visit the right subtree

- * 2 1 +

Traversals of expression tree

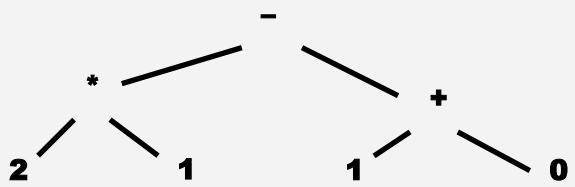


Preorder traversal

1. Visit the root
2. Visit the left subtree
3. Visit the right subtree

- * 2 1 + 1

Traversals of expression tree



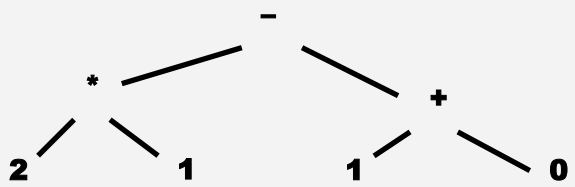
Preorder traversal

1. Visit the root
2. Visit the left subtree
3. Visit the right subtree

- * 2 1 + 1 0

پیمایش پس ترتیب عبارت

Traversals of expression tree



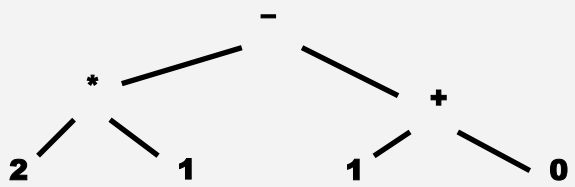
Preorder traversal

- * 2 1 + 1 0

Postorder traversal

1. Visit the left subtree
2. Visit the right subtree
3. Visit the root

Traversals of expression tree



Preorder traversal

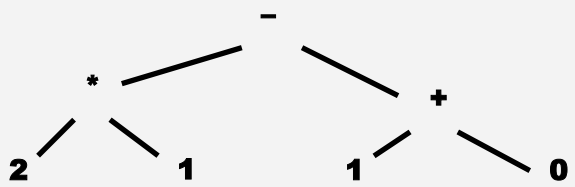
- * 2 1 + 1 0

Postorder traversal

1. Visit the left subtree
2. Visit the right subtree
3. Visit the root

2

Traversals of expression tree



Preorder traversal

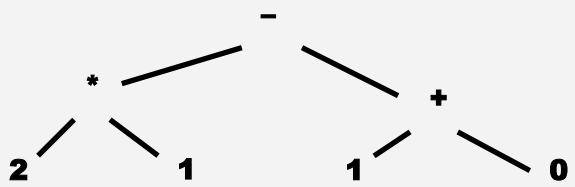
- * 2 1 + 1 0

Postorder traversal

1. Visit the left subtree
2. Visit the right subtree
3. Visit the root

2 1

Traversals of expression tree



Preorder traversal

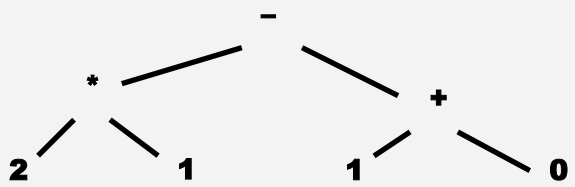
- * 2 1 + 1 0

Postorder traversal

1. Visit the left subtree
2. Visit the right subtree
3. Visit the root

2 1 *

Traversals of expression tree



Preorder traversal

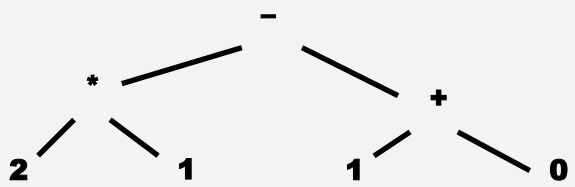
- * 2 1 + 1 0

Postorder traversal

1. Visit the left subtree
2. Visit the right subtree
3. Visit the root

2 1 * 1

Traversals of expression tree



Preorder traversal

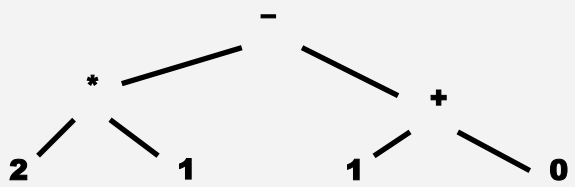
- * 2 1 + 1 0

Postorder traversal

1. Visit the left subtree
2. Visit the right subtree
3. Visit the root

2 1 * 1 0

Traversals of expression tree



Preorder traversal

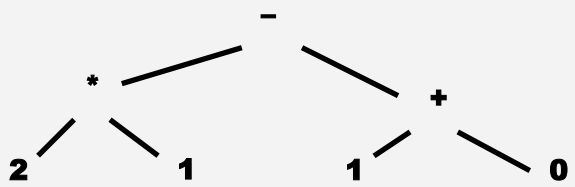
- * 2 1 + 1 0

Postorder traversal

1. Visit the left subtree
2. Visit the right subtree
3. Visit the root

2 1 * 1 0 +

Traversals of expression tree



Preorder traversal

- * 2 1 + 1 0

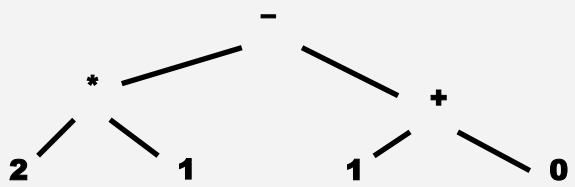
Postorder traversal

1. Visit the left subtree
2. Visit the right subtree
3. Visit the root

2 1 * 1 0 + -

پیمایش میان ترتیب عبارت

Traversals of expression tree



Preorder traversal

- * 2 1 + 1 0

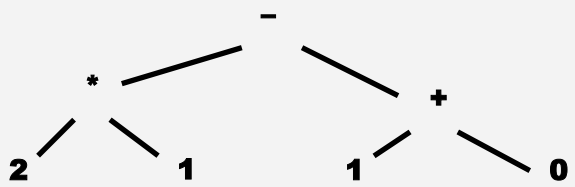
Postorder traversal

2 1 * 1 0 + -

Inorder traversal

1. Visit the left subtree
2. Visit the root
3. Visit the right subtree

Traversals of expression tree



Preorder traversal

- * 2 1 + 1 0

Postorder traversal

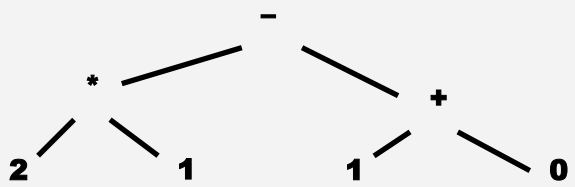
2 1 * 1 0 + -

Inorder traversal

1. Visit the left subtree
2. Visit the root
3. Visit the right subtree

2

Traversals of expression tree



Preorder traversal

- * 2 1 + 1 0

Postorder traversal

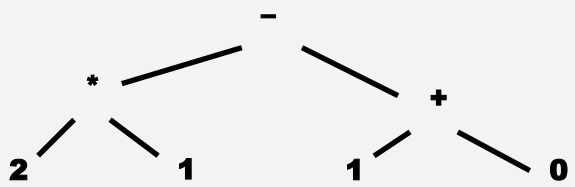
2 1 * 1 0 + -

Inorder traversal

1. Visit the left subtree
2. Visit the root
3. Visit the right subtree

2 *

Traversals of expression tree



Preorder traversal

- * 2 1 + 1 0

Postorder traversal

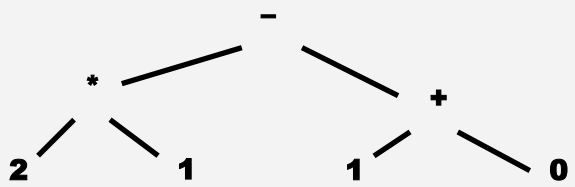
2 1 * 1 0 + -

Inorder traversal

1. Visit the left subtree
2. Visit the root
3. Visit the right subtree

2 * 1

Traversals of expression tree



Preorder traversal

- * 2 1 + 1 0

Postorder traversal

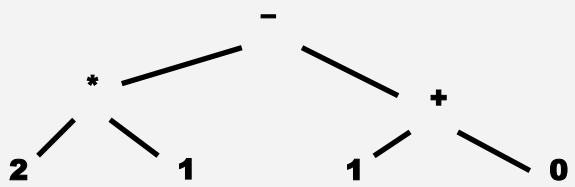
2 1 * 1 0 + -

Inorder traversal

1. Visit the left subtree
2. Visit the root
3. Visit the right subtree

2 * 1 -

Traversals of expression tree



Preorder traversal

- * 2 1 + 1 0

Postorder traversal

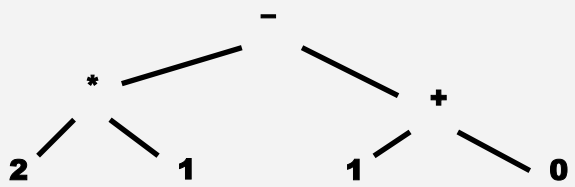
2 1 * 1 0 + -

Inorder traversal

1. Visit the left subtree
2. Visit the root
3. Visit the right subtree

2 * 1 - 1

Traversals of expression tree



Preorder traversal

- * 2 1 + 1 0

Postorder traversal

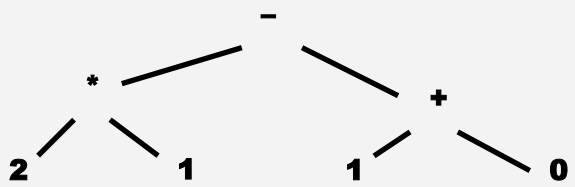
2 1 * 1 0 + -

Inorder traversal

1. Visit the left subtree
2. Visit the root
3. Visit the right subtree

2 * 1 - 1 +

Traversals of expression tree



Preorder traversal

- * 2 1 + 1 0

Postorder traversal

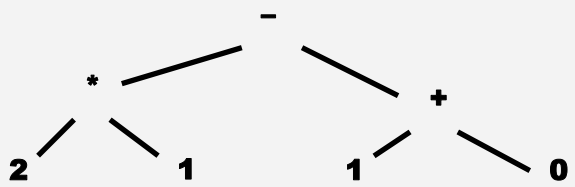
2 1 * 1 0 + -

Inorder traversal

1. Visit the left subtree
2. Visit the root
3. Visit the right subtree

2 * 1 - 1 + 0

Traversals of expression tree



Preorder traversal

Postorder traversal

Inorder traversal

- * 2 1 + 1 0

2 1 * 1 0 + -

2 * 1 - 1 + 0

Original expression,
except for parenthesis

Prefix notation

- Function calls in most programming languages use prefix notation: e.g., **add(37, 5)**.
- Aka Polish notation (PN) in honor of inventor, Polish logician Jan Łukasiewicz
- Some languages (Lisp, Scheme, Racket) use prefix notation for *everything* to make the syntax uniform.

```
(- (* 2 1) (+ 1 0))

(define (fib n)
  (if (<= n 2)
      1
      (+ (fib (- n 1)) (fib (- n 2)))))
```

Postfix notation

- Some languages (Forth, PostScript, HP calculators) use postfix notation
- Aka reverse Polish notation (RPN)

```
2 1 mul 1 0 add sub

/fib { dup
       3 lt
         { pop 1 }
         { dup 1 sub fib exch 2 sub fib add }
       ifelse
     } def
```

Syntax trees: in code

```
public interface Expr {
  int eval();
  String inorder();
}

public class Int implements Expr {
  private int v;
  public int eval() { return v; }
  public String inorder() { return " " + v + " "; }
}

public class Add implements Expr {
  private Expr left, right;
  public int eval() { return left.eval() + right.eval(); }
  public String inorder() {
    return "(" + left.infix() + "+" + right.infix() + ")";
  }
}
```

سوال؟

بازسازی درخت

ساخت درخت از روی یک پیمایش آن

Recover tree from traversal

Suppose inorder is B C A E D.

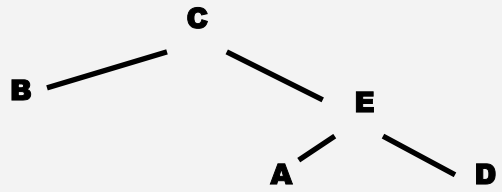
Can we recover the tree uniquely?

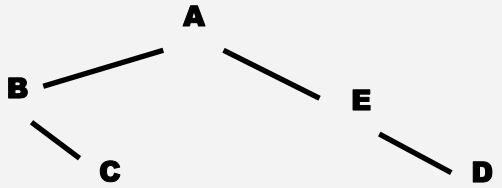
Discuss.

Recover tree from traversal

Suppose inorder is B C A E D.

Can we recover the tree uniquely? No!





Recover tree from traversals

Suppose inorder is B C A E D

preorder is A B C D E

Can we determine the tree uniquely?

Recover tree from traversals

Suppose inorder is B C A E D

preorder is A B C D E

Can we determine the tree uniquely? Yes!

- What is root?

Recover tree from traversals

Suppose inorder is B C A E D

preorder is A B C D E

Can we determine the tree uniquely? Yes!

- What is root? Preorder tells us: A

Recover tree from traversals

Suppose inorder is B C A E D

preorder is A B C D E

Can we determine the tree uniquely? Yes!

- What is root? Preorder tells us: A
- What comes before/after root A?

Recover tree from traversals

Suppose inorder is B C A E D

preorder is A B C D E

Can we determine the tree uniquely? Yes!

- What is root? Preorder tells us: A
- What comes before/after root A?
  - Inorder tells us:
    - Before: B C
    - After: E D

Recover tree from traversals

Suppose inorder is B C A E D

preorder is A B C D E

Can we determine the tree uniquely? Yes!

- What is root? Preorder tells us: A
- What comes before/after root A?
  - Inorder tells us:
    - Before: B C
    - After: E D
- Now recurse! Figure out left/right subtrees using same technique.

Recover tree from traversals

Suppose inorder is B C A E D

preorder is A B C D E

Root is A; left subtree contains B C; right contains E D

Recover tree from traversals

Suppose inorder is B C A E D

preorder is A B C D E

Root is A; left subtree contains B C; right contains E D

Left:

Inorder is B C

Preorder is B C

- What is root? Preorder: B
- What is before/after B?
  Inorder:
  - Before: nothing
  - After: C

Recover tree from traversals

Suppose inorder is B C A E D

preorder is A B C D E

Root is A; left subtree contains B C; right contains E D

Left:

Inorder is B C

Preorder is B C

- What is root? Preorder: B
- What is before/after B?
  Inorder:
  - Before: nothing
  - After: C

Right:

Inorder is E D

Preorder is D E

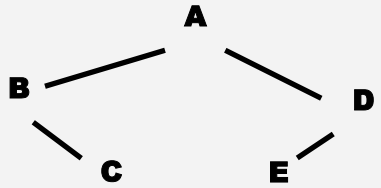
- What is root? Preorder: D
- What is before/after D?
  Inorder:
  - Before: E
  - After: nothing

Recover tree from traversals

Suppose inorder is B C A E D

preorder is A B C D E

Tree is



سوال؟