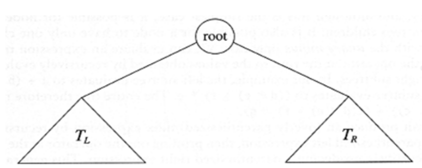
Binary Trees (7.3)

CSE 2011

Winter 2011

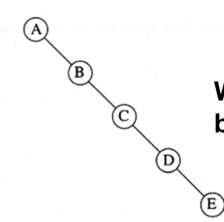
Binary Trees

- A tree in which each node can have at most two children.



Generic binary tree

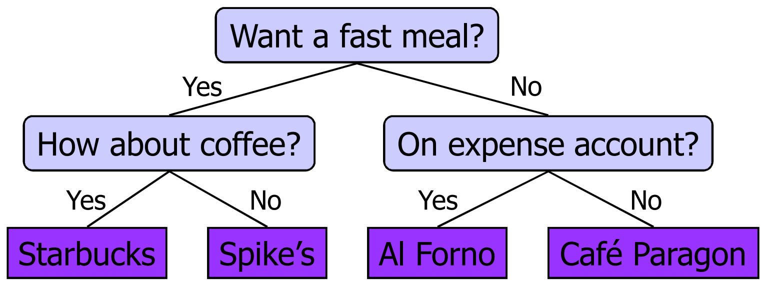
- The depth of an "average" binary tree is considerably smaller than N. In the worst case, the depth can be as large as N – 1.



Worst-case binary tree

Decision Tree

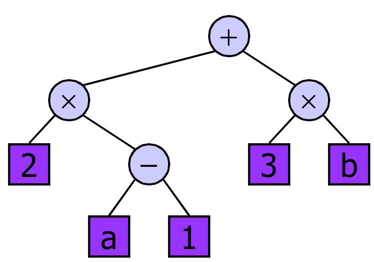
- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision



3

Arithmetic Expression Tree

- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$



4

BinaryTree ADT

- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods:
  - position left(p)
  - position right(p)
  - boolean hasLeft(p)
  - boolean hasRight(p)
- Update methods may be defined by data structures implementing the BinaryTree ADT

Implementing Binary Trees

- Arrays?
  - Discussed later
- Linked structure?

Linked Structure of Binary Trees

```
class BinaryNode {
    Object          element
    BinaryNode      left;
    BinaryNode      right;
    BinaryNode      parent;
}
```

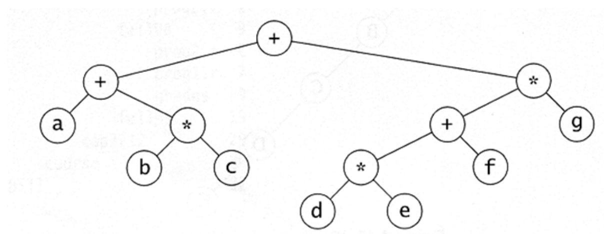
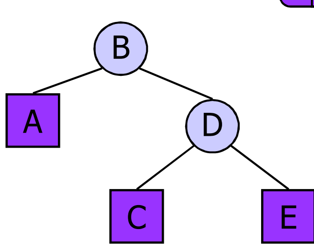


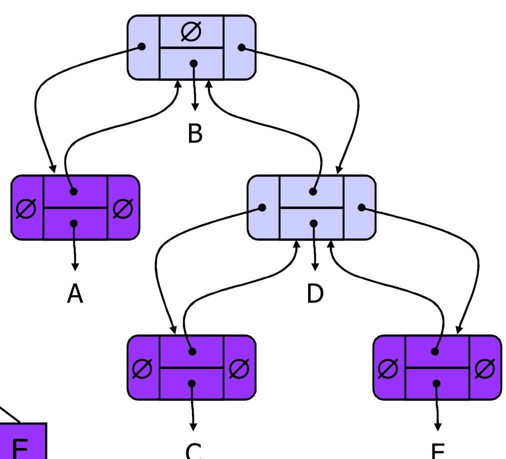
Figure 4.14 Expression tree for (a + b * c) + ((d * e + f) * g)

7

Linked Structure of Binary Trees (2)

- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node





8

Binary Tree Traversal

- Preorder (node, left, right)
- Postorder (left, right, node)
- Inorder (left, node, right)

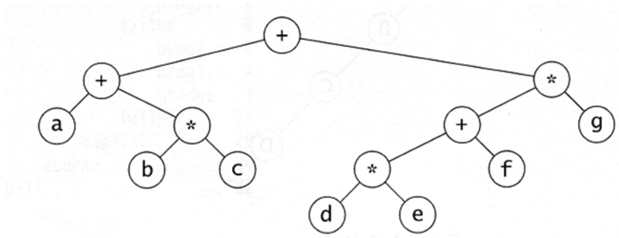


Figure 4.14 Expression tree for (a + b * c) + ((d * e + f) * g)

9

Preorder Traversal: Example

- Preorder traversal
  - node, left, right
  - prefix expression
    - + + a * b c * + * d e f g

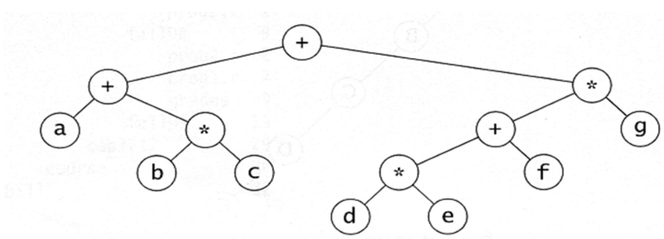


Figure 4.14 Expression tree for (a + b * c) + ((d * e + f) * g)

10

Postorder Traversal: Example

- Postorder traversal
  - left, right, node
  - postfix expression
    - a b c * + d e * f + g * +

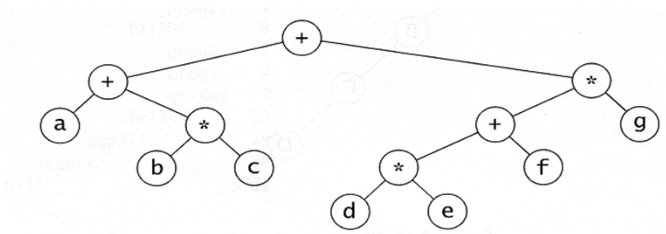


Figure 4.14 Expression tree for (a + b * c) + ((d * e + f) * g)

11

Inorder Traversal: Example

- Inorder traversal
  - left, node, right
  - infix expression
    - a + b * c + d * e + f * g

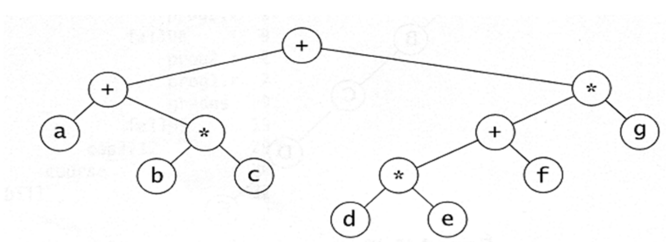


Figure 4.14 Expression tree for (a + b * c) + ((d * e + f) * g)

12

Pseudo-code for Binary Tree Traversal

```
Algorithm Preorder(x)
Input: x is the root of a subtree.
if x ≠ NULL
    then output key(x);
         Preorder(left(x));
         Preorder(right(x));
```

```
Algorithm Postorder(x)
Input: x is the root of a subtree.
if x ≠ NULL
    then Postorder(left(x));
         Postorder(right(x));
         output key(x);
```

```
Algorithm Inorder(x)
Input: x is the root of a subtree.
if x ≠ NULL
    then Inorder(left(x));
         output key(x);
         Inorder(right(x));
```


Properties of Proper Binary Trees

- A binary trees is ***proper*** if each node has either zero or two children.
- Level: depth
  The root is at level 0
  Level d has at most 2^d nodes
- Notation:
  - n number of nodes
  - e number of external (leaf) nodes
  - i number of internal nodes
  - h height

$n = e + i$

$e = i + 1$

$h+1 \le e \le 2^h$

$n = 2e - 1$

$h \le i \le 2^h - 1$

$2h+1 \le n \le 2^{h+1} - 1$

$\log_2 e \le h \le e - 1$

$\log_2 (i + 1) \le h \le i$

$\log_2 (n + 1) - 1 \le h \le (n - 1)/2$

Properties of (General) Binary Trees

- Level: depth
  The root is at level 0
  Level d has at most 2^d nodes
- Notation:
  - n number of nodes
  - e number of external (leaf) nodes
  - i number of internal nodes
  - h height

$$h+1 \le n \le 2^{h+1}-1$$

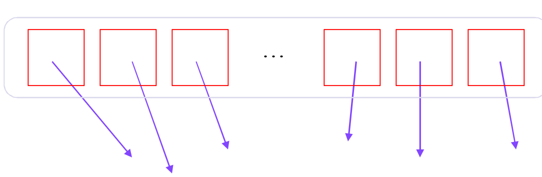
$$1 \le e \le 2^h$$

$$h \le i \le 2^h-1$$

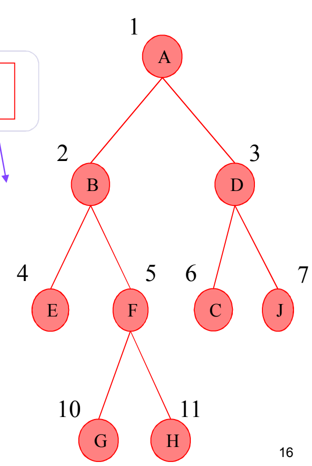
$$\log_2(n+1)-1 \le h \le n-1$$

Array-Based Implementation

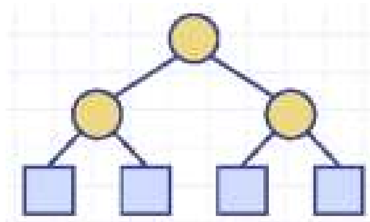
- Nodes are stored in an array.

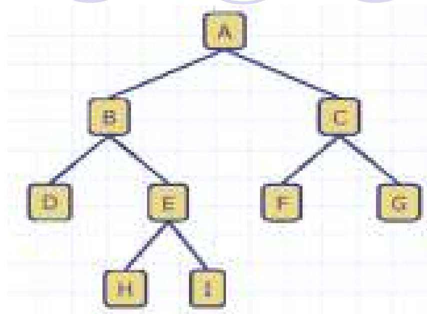


- Let rank(v) be defined as follows:
  - rank(root) = 1
  - if v is the left child of parent(v),
    rank(v) = 2 * rank(parent(v))
  - if v is the right child of parent(v),
    rank(v) = 2 * rank(parent(v)) + 1



Array Implementation of Binary Trees





Each node *v* is stored at index *i* defined as follows:

- If *v* is the root, $i = 1$
- The left child of *v* is in position $2i$
- The right child of *v* is in position $2i + 1$
- The parent of *v* is in position ???

Space Analysis of Array Implementation

- *n*: number of nodes of binary tree *T*
- p_M: index of the rightmost leaf of the corresponding ***full*** binary tree (or size of the full tree)
- *N*: size of the array needed for storing *T*; $N = p_M + 1$

Best-case scenario: balanced, full binary tree $p_M = n$

Worst case scenario: unbalanced tree

- Height $h = n - 1$
- Size of the corresponding full tree:

  $p_M = 2^{h+1} - 1 = 2^n - 1$
- $N = 2^n$

Space usage: $O(2^n)$

Arrays versus Linked Structure

Linked lists

- Slower operations due to pointer manipulations
- Use less space if the tree is unbalanced
- AVL trees: rotation (restructuring) code is simple

Arrays

- Faster operations
- Use less space if the tree is balanced (no pointers)
- AVL trees: rotation (restructuring) code is complex

19

Next time …

- Binary Search Trees (10.1)

20