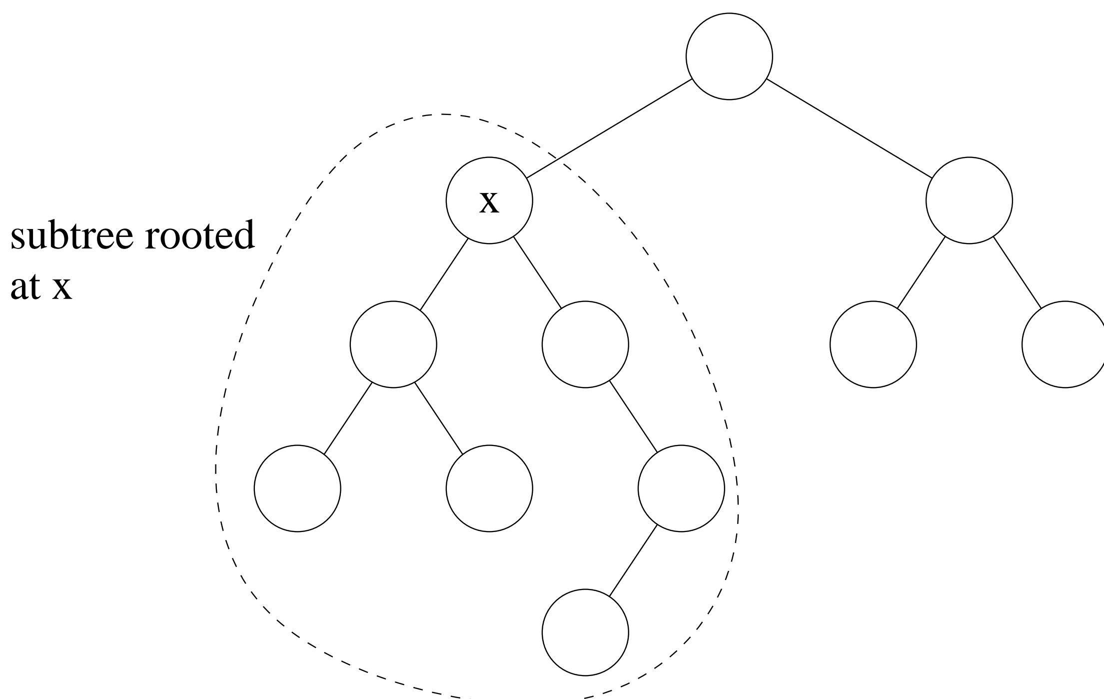
H.O.#12 supplement
Fall 2015

Binary Tree Terminology

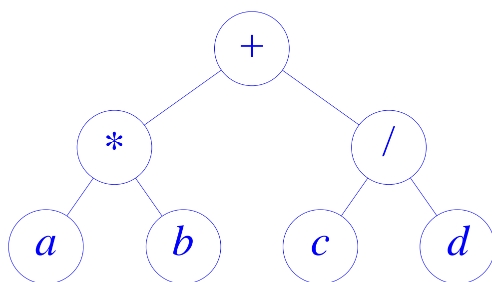
- A node that has one or two children is called an *internal node*.
- The two children of a node are *siblings* of each other. For example, left(x) is the left sibling of right(x) and right(x) is the right sibling of left(x).
- A node usually contains a data field as well to store some value.
- Subtree:

  pick any node x, then x and its *descendants* form a subtree, which we call the subtree rooted at x.

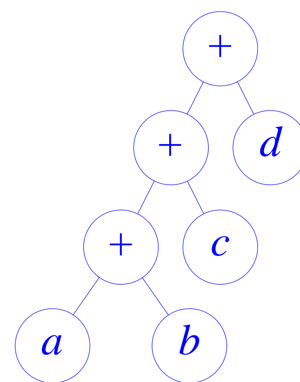


Expression Trees

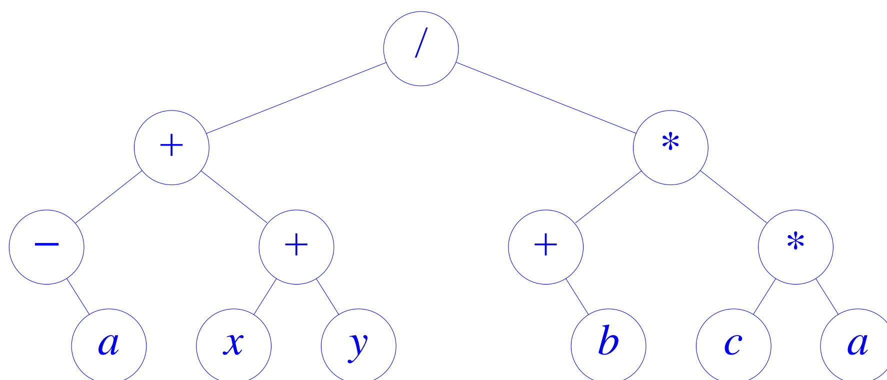
- Binary trees that represent arithmetic expressions
- One application of expression trees is in the generation of optimal computer code to evaluate an expression



(a) $(a * b) + (c / d)$



(b) $((a + b) + c) + d)$



(c) $((-a) + (x + y)) / ((+b) * (c * a))$

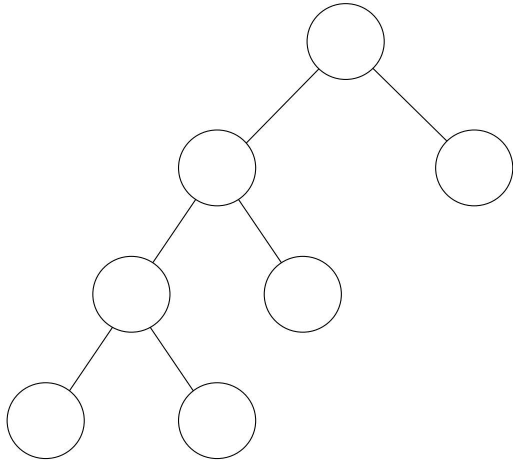
Figure 8.5 Expression trees

Formation of Expression Tree

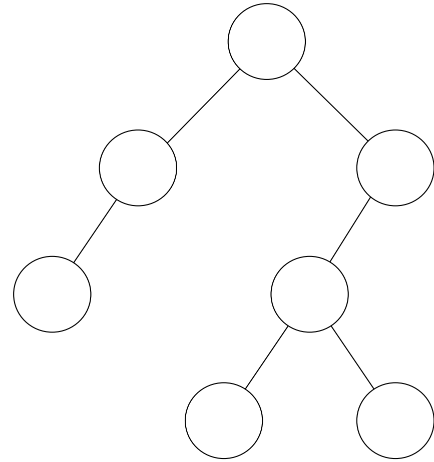
1. Start from a postfix expression
   - If the input is in infix, convert it to postfix first using stack
2. Initialize a stack whose data is pointer to character
3. Push operand into the stack (i.e., make the pointer pointing to the operand)
4. When an operator is encountered:
   (a) Create a binary node with the operator
   (b) pop once to become the right child of the operator
   (c) pop another time to become the left child of the operator
   (d) push the operator into the stack
5. Repeat Step 3 until the postfix expression is completely scanned.

Full Binary Tree

Every internal node has exactly two children.



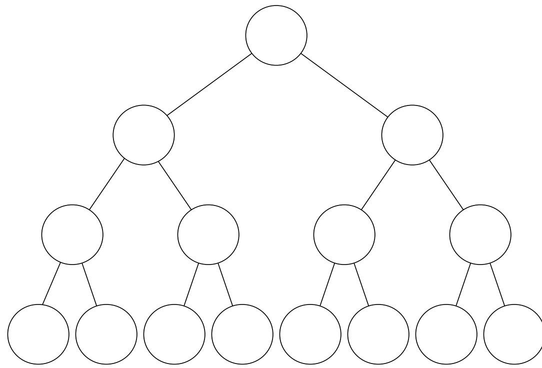
full binary tree



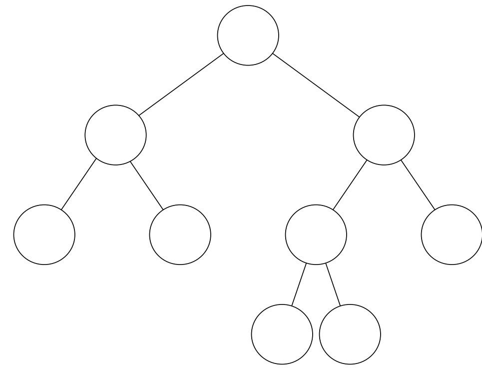
not a full binary tree

Perfect Binary Tree

Every level is full.



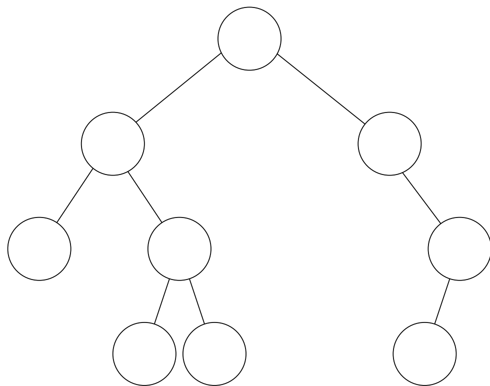
perfect binary tree



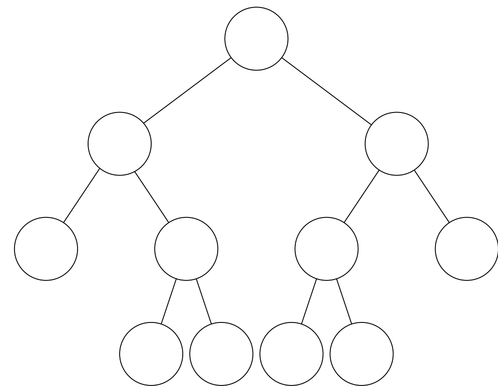
not a perfect binary tree

Complete Binary Tree

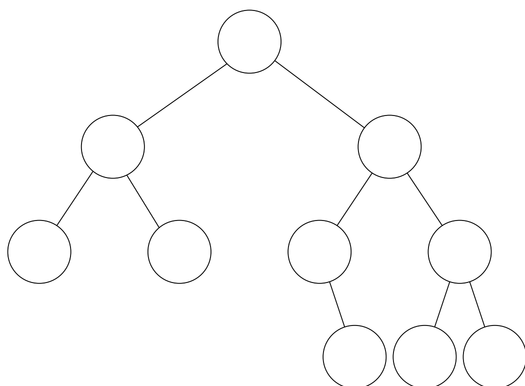
Each level is full except possibly the bottommost level. If the bottommost level is not full, then the nodes must be packed to the left.



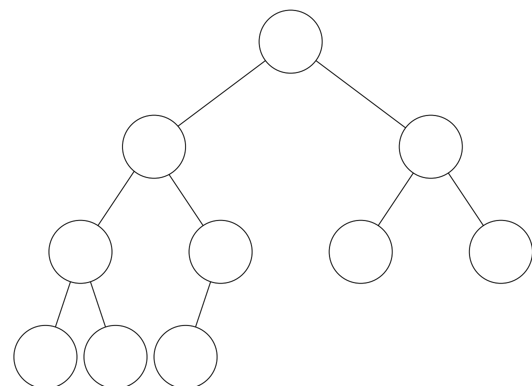
not a complete binary tree



not a complete binary tree



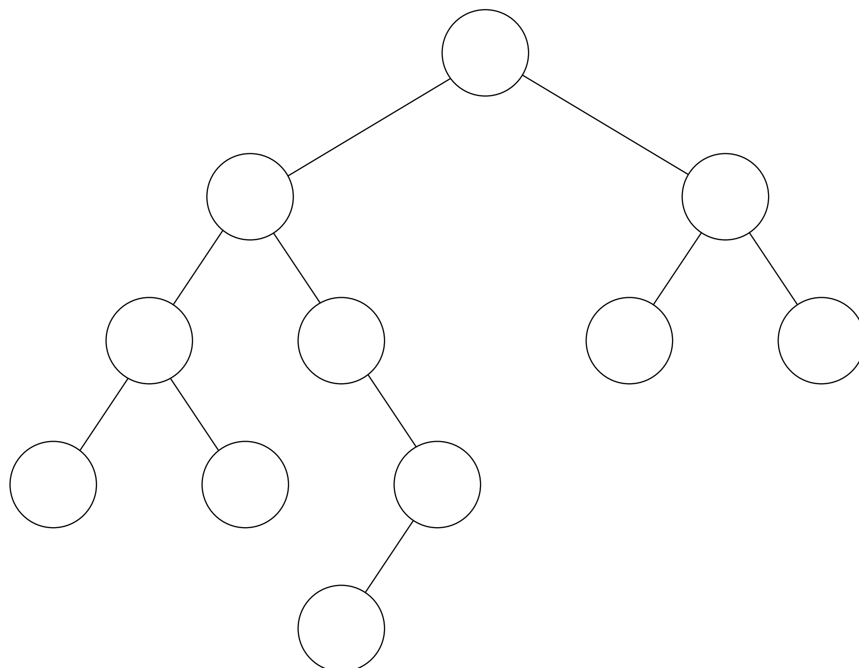
not a complete binary tree



complete binary tree

Height of a Binary Tree

- The number of edges on the longest path from the root to a leaf.
- A binary tree of height k has
  - at least $k+1$ elements (linear chain), and
  - at most $2^{k+1} - 1$ elements (perfect tree):
    * 1st level: 1 node (root)
    * 2nd level: 2 nodes
    * 3rd level: $2 \cdot 2 = 2^2$ nodes
    * 4th level: $2 \cdot 2^2 = 2^3$ nodes
    * kth level: 2^{k-1} nodes
    * $(k+1)$th level (bottom-most level): 2^k nodes
    * $N = 1 + 2 + \ldots 2^k = 2^{k+1} - 1$.



height = 4

Height of a Binary Tree (Cont.)

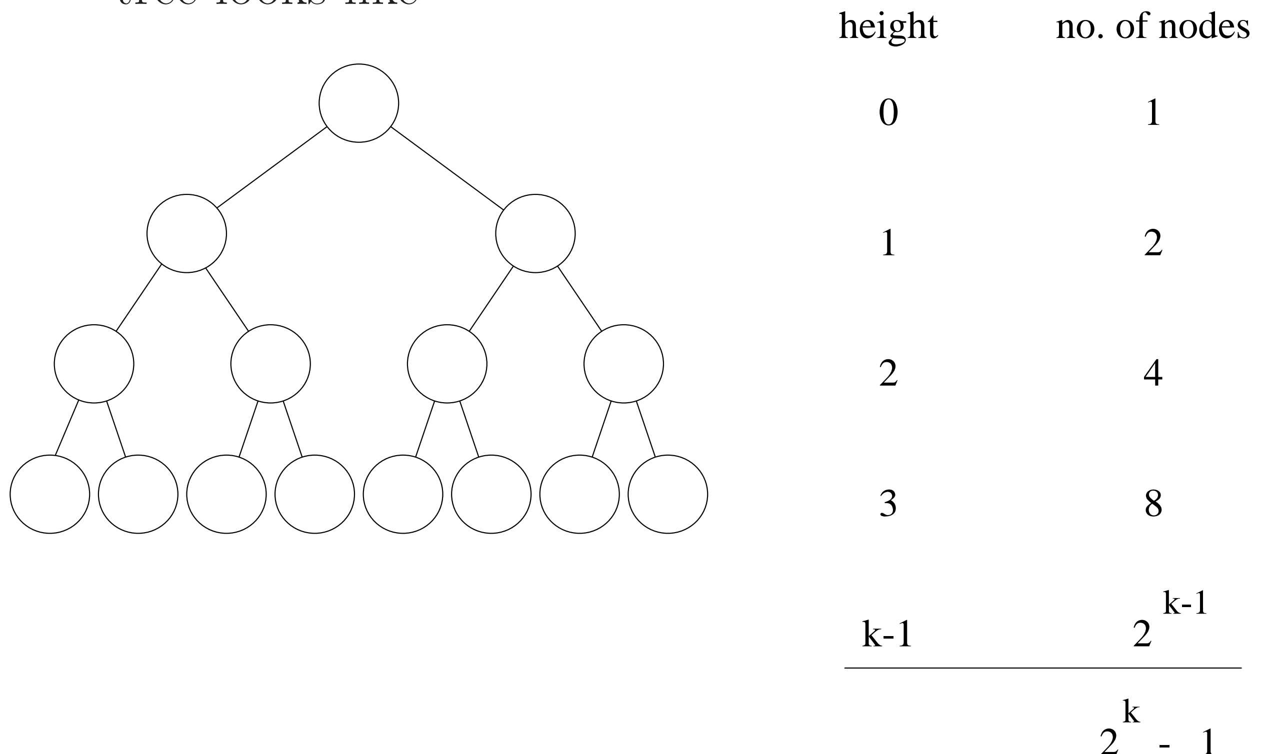
Claim: *A complete binary tree of m nodes has $h_m = \lfloor \log_2 m \rfloor$ height.*

Proof. By induction on m.

Basis step: $m = 1$, $h_1 = 0 = \lfloor \log_2 m \rfloor$. Therefore the statement is true for $m = 1$.

Induction step: Assume the statement is true for $m = n$, i.e., its height is $h_n = \lfloor \log_2 n \rfloor$. We need to consider two cases: $n = 2^k - 1$ and otherwise.

Case (1): If $n = 2^k - 1$, for some k, then the tree looks like



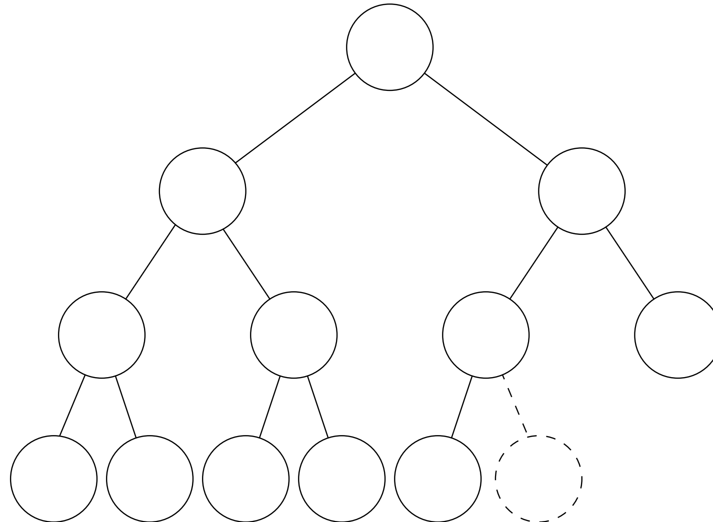
which obviously has height $h_n = k - 1$.

So a tree of $n + 1$ nodes must increase its height by 1, i.e., equal to

$$h_{n+1} = h_n + 1 = (k-1) + 1 = k = \lfloor \log_2(n+1) \rfloor.$$

Height of a Binary Tree (Cont.)

Case (2): We have $n \neq 2^k - 1$. Suppose $2^k - 1 < n < 2^{k+1} - 1$, then the tree has height $h_n = k$ and it looks like



A complete binary tree with $n + 1$ nodes is obtained by packing one more node at the bottommost level. So the height is still $h_{n+1} = k = \lfloor \log_2(n + 1) \rfloor$.

Therefore, by MI, the statement is true for all m.

Programming Height of a Binary Tree

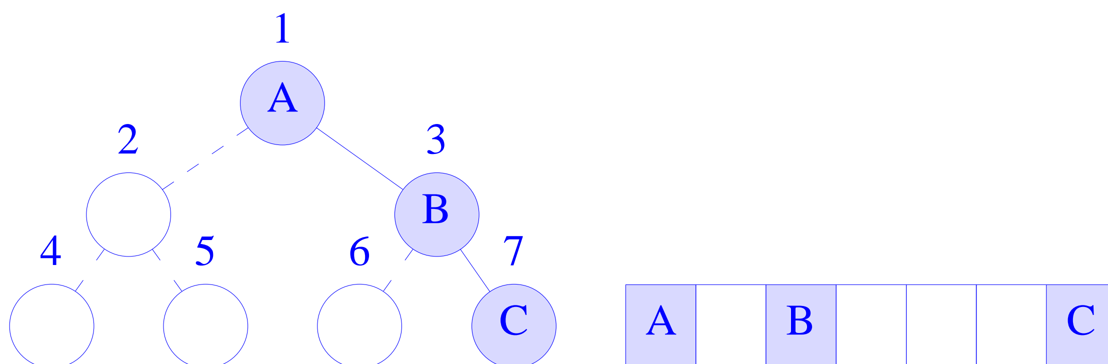
Height = `max{hl, hr} + 1.`

This includes the Null node (i.e., a null node is at height 0), and a single node is at height 1. This is one more than our previous definition. If you want to revert back to the previous definition, simply get the value and minus it by 1.

```
template <class T>
int BinaryTree<T>::Height(BinaryTreeNode<T> *t) const
{// Return height of tree *t.
   if (!t) return 0;               // empty tree
   int hl = Height(t->LeftChild);  // height of left
   int hr = Height(t->RightChild); // height of right
   if (hl > hr) return ++hl;
   else return ++hr;
}
```

Formula-Based Tree Representation

- Missing elements are represented by white circles and boxes.
- The number i on top of a circle is the *position* of the element, not its array index
  - The parent of node i is given by $\lfloor i/2 \rfloor$.
- Application: efficiently represent a complete binary tree as an array for sorting (heapsort)



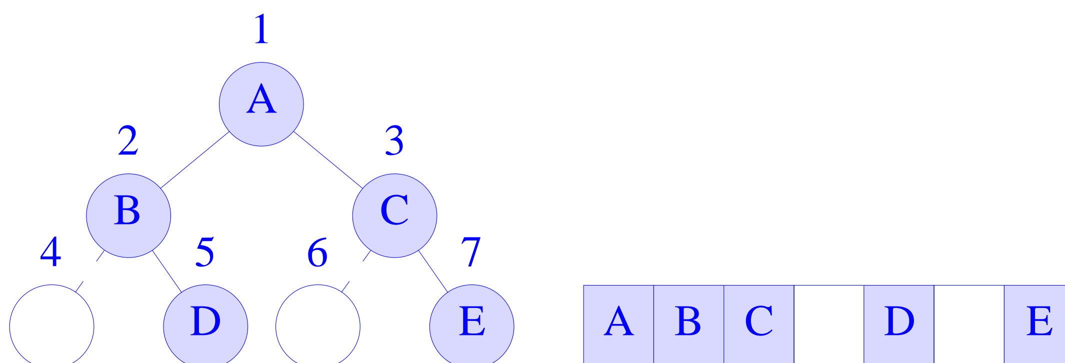
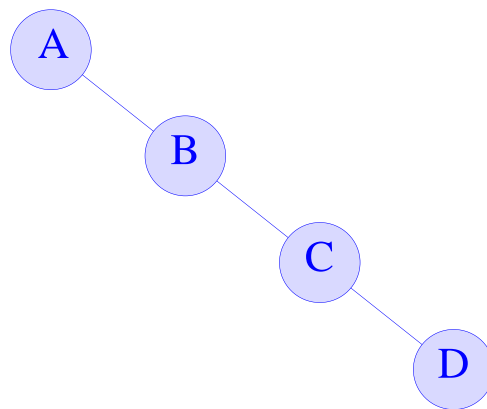


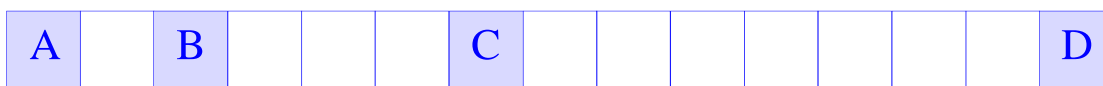
Figure 8.8 Incomplete binary trees

Large Storage Requirement for Arbitrary Binary Tree

- An arbitrary binary tree that has n elements may require an array of size up to $2^n - 1$ for its representation
- The case when each element is the right child of its parent, i.e., the right-skewed binary tree



(a) Right-skewed tree



(b) Array representation

Figure 8.9 Right-skewed binary tree

Linked Representation

- More efficient to represent an arbitrary binary tree
- Use links or pointers (`LeftChild` and `RightChild`)
- A variable `t` is used to keep track of the root of the tree

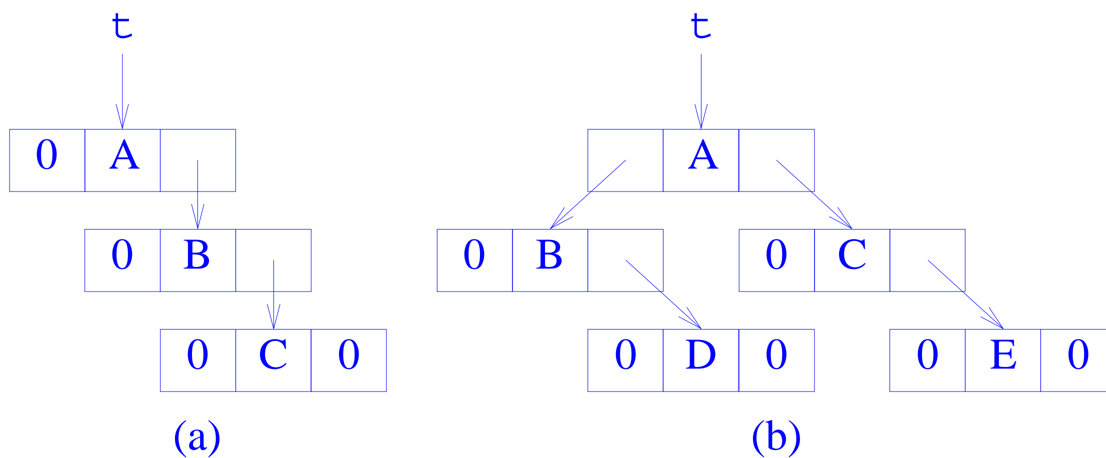


Figure 8.10 Linked representations

Tree Traversals and Search Trees

Please go back to the slides