Algorithms and Data Structures

7th Lecture: Tree

Yutaka Watanobe, Jie Huang, Yan Pei, Wenxi Chen,
S. Semba, Deepika Saxena, Yinghu Zhou, Akila Siriweera

University of Aizu

Last Updated: 2025/01/05

Outline

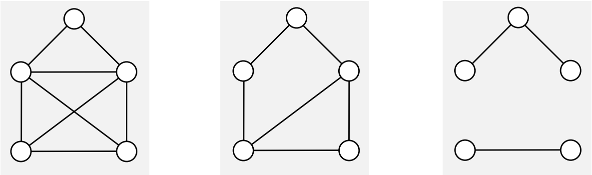
- Trees
- Rooted Trees
- Ordered Trees
- Binary Trees
- Tree Walk

Trees (1)

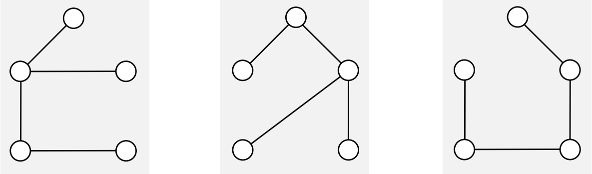
- A tree is a connected, acyclic, undirected graph.
- Let $G = (V, E)$ be an undirected graph. The following statements are equivalent.
  1. G is a free tree.
  2. Any two vertices in G are connected by a unique simple path.
  3. G is connected, but if any edge is removed from E, the resulting graph is disconnected.
  4. G is connected, and $|E| = |V| - 1$.
  5. G is acyclic, and $|E| = |V| - 1$.
  6. G is acyclic, but if any edge is added to E, the resulting graph contains a cycle.

Trees (2)

Graphs



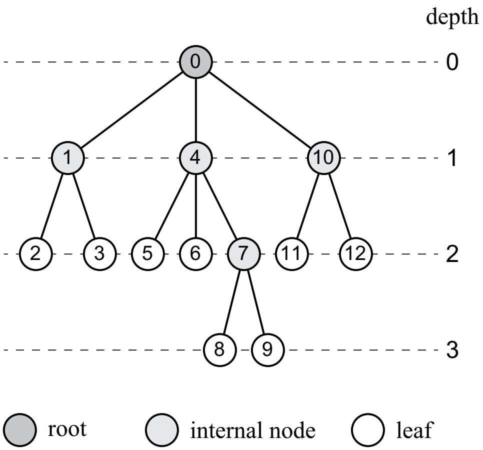
Trees



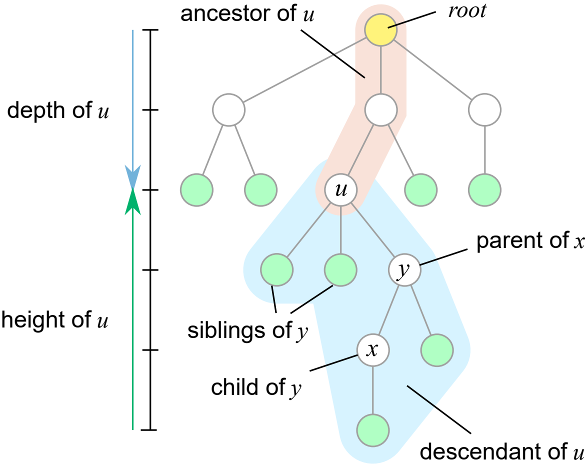
Rooted Trees (1)

- A rooted tree is a tree in which one of the vertices is distinguished from the others.
- The distinguished vertex is called the **root** of the tree.
- We often refer to a vertex of a rooted tree as a **node** of the tree.

Rooted Trees (2)



Rooted Trees (3)



Rooted Trees (4)

Consider a node x in a rooted tree T with root r.

- Any node y on the unique path from r to x is called an **ancestor** of x.
- If y is an ancestor of x, then x is a **descendant** of y.
- Every node is both an ancestor and a descendant of itself.
- If y is an ancestor of x and x is not equal to y, the y is a **proper ancestor** of x and x is a **proper descendant** of y.
- The **subtree** rooted at x is the tree induced by descendant of x, rooted at x.

Rooted Trees (5)

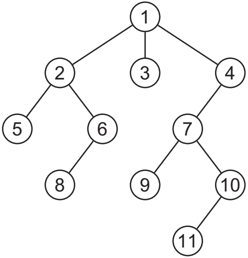
- If the last edge on the path from the root r of a tree T to a node x is (y, x), then y is the **parent** of x, and x is a **child** of y.
- The **root** is the only element in T that has no parent.
- If two nodes have the same parent, they are **siblings**.
- A node with no children is an **external node** or **leaf**.
- A non-leaf node is an **internal node**.

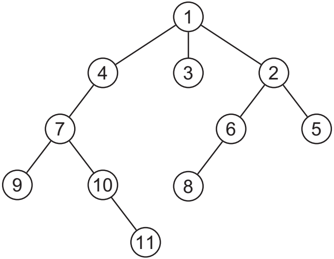
Rooted Trees (6)

- The number of children of a node x in a rooted tree T is called the **degree** of x.
- The length of the path from the root r to a node x is the **depth** of x in T.
- The **height** of a node in a tree is the number of edges on the longest simple downward path from the node to a leaf, and the **height of a tree** is the height of its root.
- The height of a tree is also equal to the largest depth of any node in the tree.

Ordered Trees

- An ordered tree is a rooted tree in which the children of each node are ordered. That is, if a node has k children, then there is a first child, a second child, ..., and a k-th child.
- The following trees are different when considered to be ordered trees, but the same when considered to be just rooted trees.

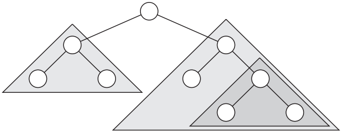




Binary Trees (1)

Binary trees are defined recursively. A binary tree T is a structure defined on a finite set of nodes that either

- contains no nodes, or
- is composed of three disjoint sets of nodes: a **root** node, a binary tree called its **left subtree**, and a binary tree called its **right subtree**.



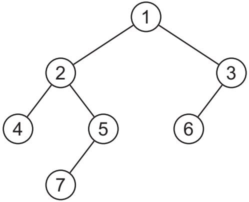
Binary Trees (2)

- The binary tree that contains no nodes is called the **empty tree** or **null tree**, sometimes denoted **NIL**.
- If the left subtree is nonempty, its root is called the **left child** of the root of the entire tree.
- Likewise, the root of a nonnull right subtree is the **right child** of the root of the entire tree.
- If a subtree is the null tree **NIL**, we say that the child is **absent** or **missing**.

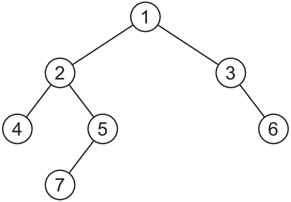
Binary Trees (3)

- A binary tree is not simply an ordered tree in which each node has degree at most 2.
- For example, in a binary tree, if a node has just one child, the position of the child - whether it is the left child or the right child - matters.
- In an ordered tree, there is no distinguishing a sole child as being either left or right.

Binary Trees (4)



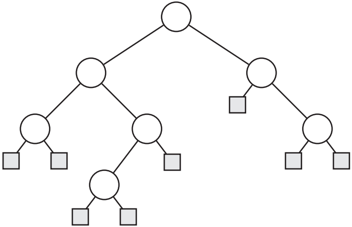
(a)



(b)

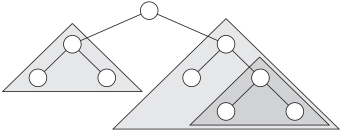
(b) is a binary tree different from the one in (a). In (a), the left child of node 3 is 6 and the right child is absent. In (b), the left child of node 3 is absent and the right child is 6. As ordered trees, these trees are the same, but as binary trees, they are distinct.

Binary Trees (5)



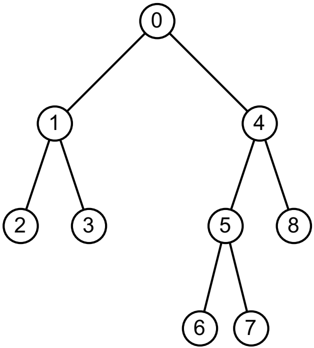
This is the binary tree in (b) represented by the internal nodes of a full binary tree: an ordered tree in which each internal node has degree 2. The leaves in the tree are shown as squares.

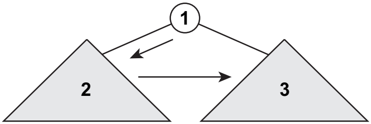
Tree Walk



- A **preorder tree walk** prints the root before the values in either subtree.
  root $\rightarrow$ left subtree $\rightarrow$ right subtree
- An **inorder tree walk** prints the root of a subtree between its left subtree and right subtree.
  left subtree $\rightarrow$ root $\rightarrow$ right subtree
- A **postorder tree walk** prints the root after the values in its subtrees.
  left subtree $\rightarrow$ right subtree $\rightarrow$ root

Preorder Tree Walk

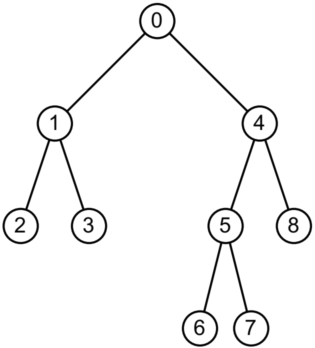


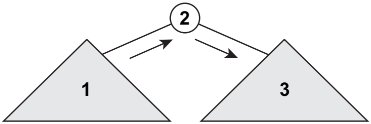



```
preParse(u)
  if u == NIL
    return
  print u
  preParse(T[u].left)
  preParse(T[u].right)
```

$0 \to 1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7 \to 8$

Inorder Tree Walk

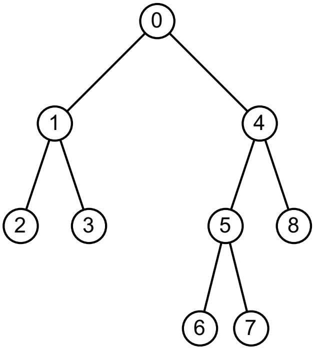


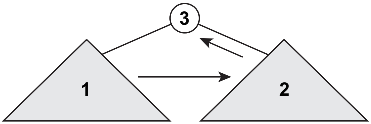



```
inParse(u)
  if u == NIL
    return
  inParse(T[u].left)
  print u
  inParse(T[u].right)
```

$2 \rightarrow 1 \rightarrow 3 \rightarrow 0 \rightarrow 6 \rightarrow 5 \rightarrow 7 \rightarrow 4 \rightarrow 8$

Postorder Tree Walk





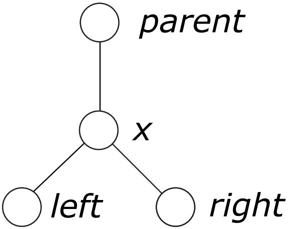

```
postParse(u)
  if u == NIL
    return
  postParse(T[u].left)
  postParse(T[u].right)
  print u
```

$2 \rightarrow 3 \rightarrow 1 \rightarrow 6 \rightarrow 7 \rightarrow 5 \rightarrow 8 \rightarrow 4 \rightarrow 0$

Representing Binary Trees (1)

- We use the fields *p*, *left*, and *right* to store pointers to the parent, left child, and right child of each node in a binary tree T.
- If $p[x]$ = NIL, then x is the root.
- If node x has no left child, then $left[x]$ = NIL, and similarly for the right child.
- The root of the entire tree T is pointed to by the attribute *root* of T. If *root* = NIL, then the tree is empty.

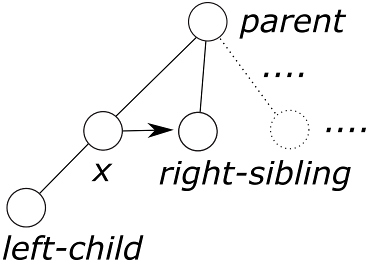
Representing Binary Trees (2)



Representing Rooted Trees (1)

- There is a clever scheme for using binary trees to represent trees with arbitrary numbers of children.
- It has the advantage of using only $O(n)$ space for any n-node rooted tree.
- In **left-child, right-sibling representation**, each node contains a parent pointer p, and *root* points to the root of tree T.
- Instead of having a pointer to each of its children, each node x has only two pointers:
    1. *left-child*[x] points to the leftmost child of node x, and
    2. *right-sibling*[x] points to the sibling of x immediately to the right.
- If node x has no children, then *left-child*[x] = NIL, and if node x is the rightmost child of its parent, then *right-sibling*[x] = NIL.

Representing Rooted Trees (2)



Reference

1. Introduction to Algorithms (third edition), Thomas H.Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. The MIT Press, 2012.