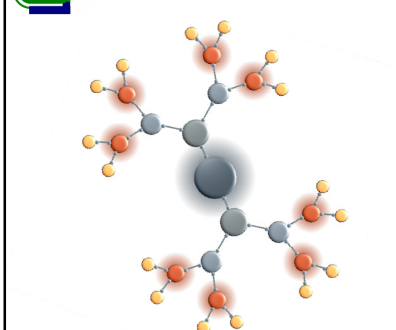
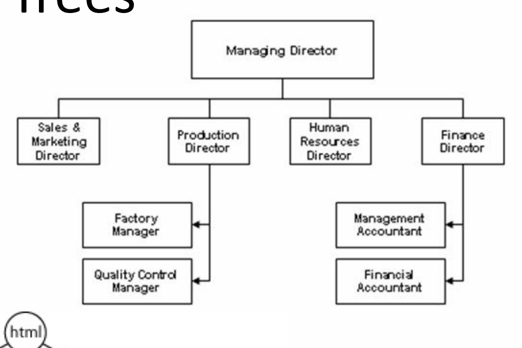
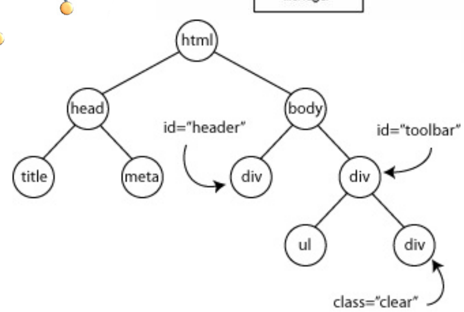
Binary Trees

Trees





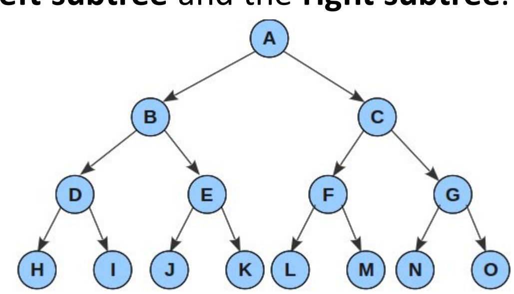


Advantages of Trees

- Trees reflect structural relationships in the data
- Trees are used to represent hierarchies
- Trees provide an efficient insertion and searching
- Trees are very flexible data structures, allowing to move subtrees around with minimum effort

A binary tree

- **Definition:** A binary tree is either **empty** or consists of a node called the **root** together with two binary trees called the **left subtree** and the **right subtree**.



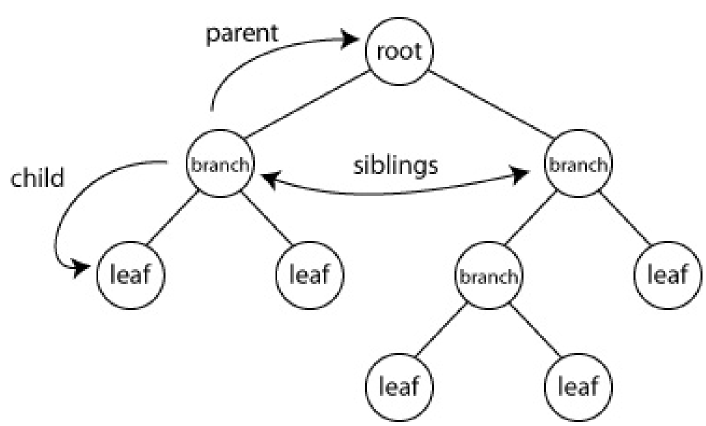
Number of Leaves (n) = 8
Number of Nodes (2n-1) = 2x8-1 = 16-1 = 15

Parts of a binary tree

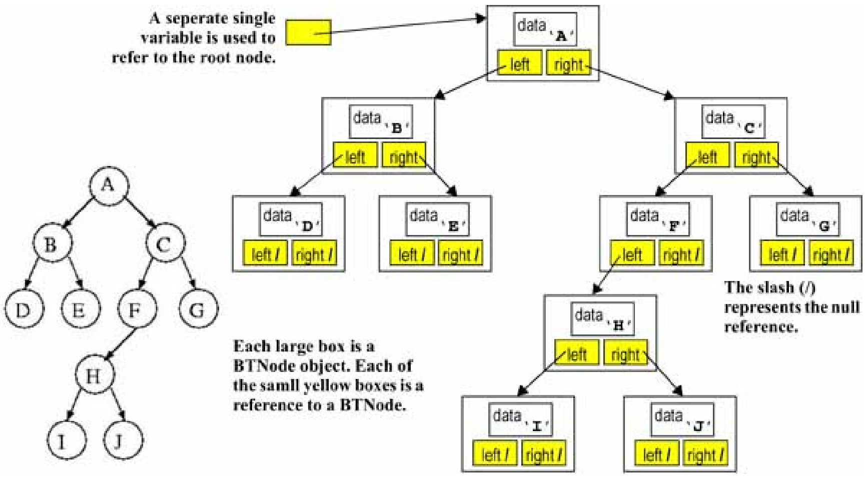
- A binary tree is composed of zero or more nodes
- Each node contains:
  - A value (some sort of data item)
  - A reference or pointer to a left child (may be null), and
  - A reference or pointer to a right child (may be null)
- A binary tree may be *empty* (contain no nodes)
- If not empty, a binary tree has a root node
  - Every node in the binary tree is reachable from the root node by a *unique* path
- A node with neither a left child nor a right child is called a leaf

5

Parts of a binary tree



Picture of a binary tree

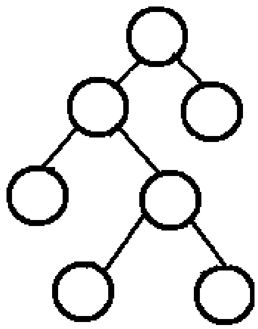


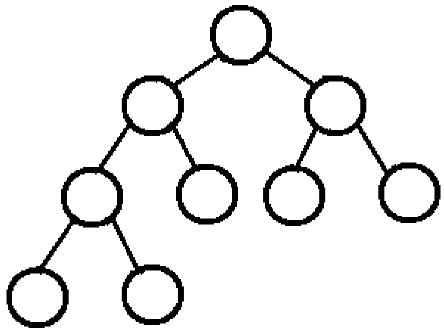
7

Types of Binary Trees

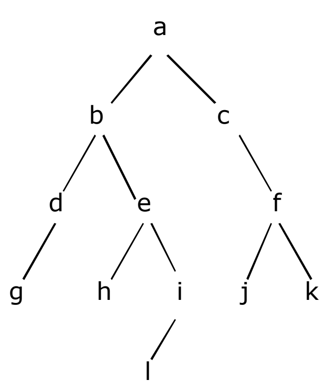
- A **full binary tree** (sometimes **proper binary tree** or **2-tree** or **strictly binary tree**) is a tree in which every node other than the leaves has two children.
- A **perfect binary tree** is a *full binary tree* in which all *leaves* are at the same *depth* or same *level*, and in which every parent has two children
- A **complete binary tree** is a binary tree in which every level, *except possibly the last*, is completely filled, and all nodes are as far left as possible
- A **balanced binary tree** is commonly defined as a binary tree in which the *depth* of the two subtrees of every node never differ by more than 1

?





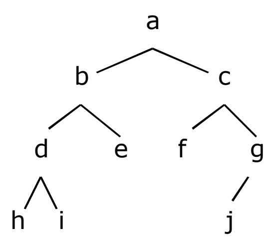
Size and depth (or height)



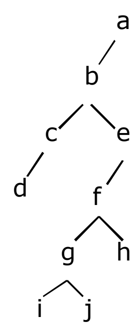
- The size of a binary tree is the number of nodes in it
  – This tree has size 12
- The depth of a node is its distance from the root
  – a is at depth zero
  – e is at depth 2
- The depth of a binary tree is the depth of its deepest node
  – This tree has depth 4

10

Balance Binary Trees



A balanced binary tree



An unbalanced binary tree

- A binary tree is balanced if every level above the lowest is “full” (contains 2^n nodes)
- In most applications, a reasonably balanced binary tree is desirable

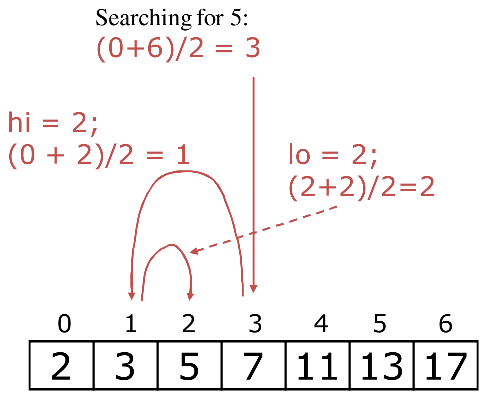
11

Height of a Binary Tree

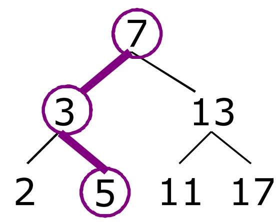
- If h = height of a binary tree,
  max # of leaves = 2^h
  max # of nodes = $2^{h+1} - 1$

Binary search in an array

- Look at array location (lo + hi)/2



Using a binary search tree



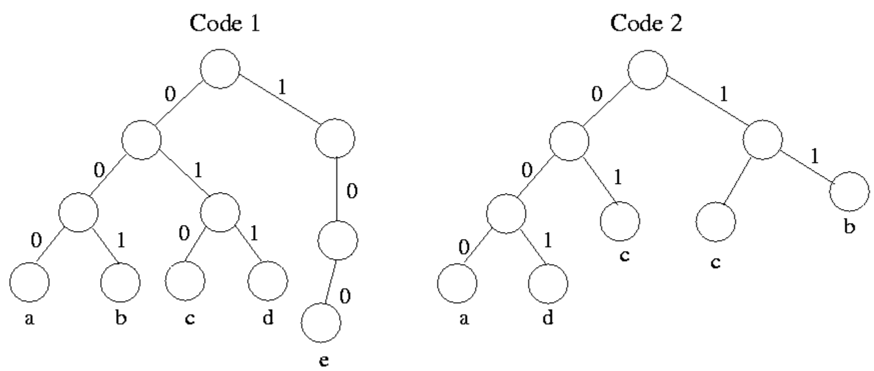
13

Huffman Coding – A Binary Tree Application [David A Huffman' 52]

- Suppose we have a message made from the five characters a, b, c, d, e, with probabilities 0.12, 0.40, 0.15, 0.08, 0.25, respectively

Symbol	Prob	code 1	code 2
a	0.12	000	000
b	0.40	001	11
c	0.15	010	01
d	0.08	011	001
e	0.25	100	10

Huffman's Coding cont…



Average length per character
Code 1: (0.12)(3) + (0.4)(3) + (0.15)(3) + (0.08)(3) + (0.25)(3) = 3
Code 2: (0.12)(3) + (0.4)(2) + (0.15)(2) + (0.08)(3) + (0.25)(2) = 2.2

Huffman's Coding cont…

- So the resulting "average length per character" = ?
- And the resulting Huffman tree for the example ?

Binary Tree traversals

- A binary tree is defined recursively: it consists of a root, a left subtree, and a right subtree
- To traverse (or walk) the binary tree is to visit each node in the binary tree exactly once
- Tree traversals are naturally recursive
- Since a binary tree has three "parts," there are six possible ways to traverse the binary tree:
  - root, left, right
  - left, root, right
  - left, right, root
  - root, right, left
  - right, root, left
  - right, left, root

17

Binary Tree Traversals

- Depth First Traversals
  - Preorder
  - Inorder
  - Postorder
- Breadth First Traversal

Preorder Traversal

- In preorder, the root is visited *first*
- Here's a preorder traversal to print out all the elements in the binary tree:

```
public void preorder(BinaryTree bt) {
    if (bt == null) return;
    printf(bt.value);
    preorder (bt.leftChild);
    preorder (bt.rightChild);
}
```

19

Inorder Traversal

- In inorder, the root is visited *in the middle*
- Here's an inorder traversal to print out all the elements in the binary tree:

```
public void inorder (BinaryTree bt) {
    if (bt == null) return;
    inorder(bt.leftChild);
    printf(bt.value);
    inorder(bt.rightChild);
}
```

20

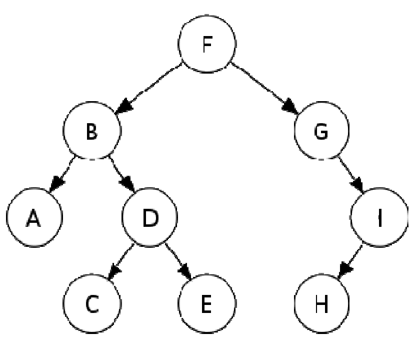
Postorder Traversal

- In postorder, the root is visited *last*
- Here's a postorder traversal to print out all the elements in the binary tree:

```
public void postorder(BinaryTree bt) {
    if (bt == null) return;
    postorder(bt.leftChild);
    postorder(bt.rightChild);
    printf(bt.value);
}
```

21

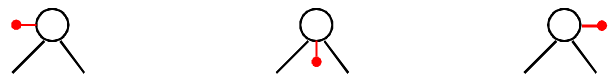
Tree Traversals – An Example



Depth-first
Preorder traversal sequence: F, B, A, D, C, E, G, I, H (root, left, right)
Inorder traversal sequence: A, B, C, D, E, F, G, H, I (left, root, right)
Postorder traversal sequence: A, C, E, D, B, H, I, G, F (left, right, root)

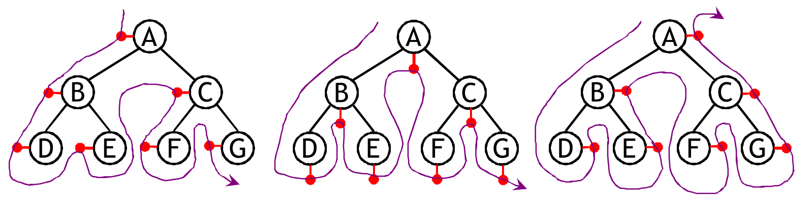
Tree traversals using “flags”

- The order in which the nodes are visited during a tree traversal can be easily determined by imagining there is a “flag” attached to each node, as follows:



preorder   inorder   postorder

- To traverse the tree, collect the flags:



A B D E C F G   D B E A F C G   D E B F G C A

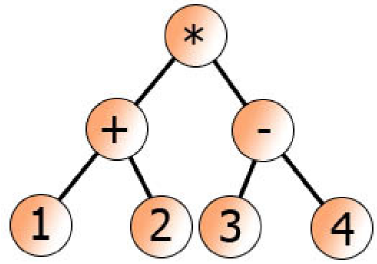
23

Other traversals

- The other traversals are the reverse of these three standard ones
  - That is, the right subtree is traversed before the left subtree is traversed
- Reverse preorder: root, right subtree, left subtree
- Reverse inorder: right subtree, root, left subtree
- Reverse postorder: right subtree, left subtree, root

24

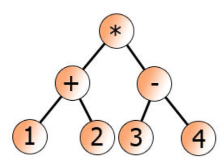
Binary Tree as Expression Tree



((1+2)*(3-4))

Binary Tree as Expression Tree

- Traversing the expression tree using
  - Preorder: results in to prefix code
  - In-order: results in to infix code (same expression)
  - Post-order: results in to postfix (reverse polish) code
- Find these codes for this example



((1+2)*(3-4))

Summary (Trees)

- A non-linear, hierarchical, and recursive data structures
- Form the basis of many useful and efficient data structures
- Traversals
  - Depth first
    - Pre-order
    - In-order
    - Post-order
  - Breadth First Traversal
- Applications of Binary Trees
  - Huffman coding
  - Expression Trees

27