Contents

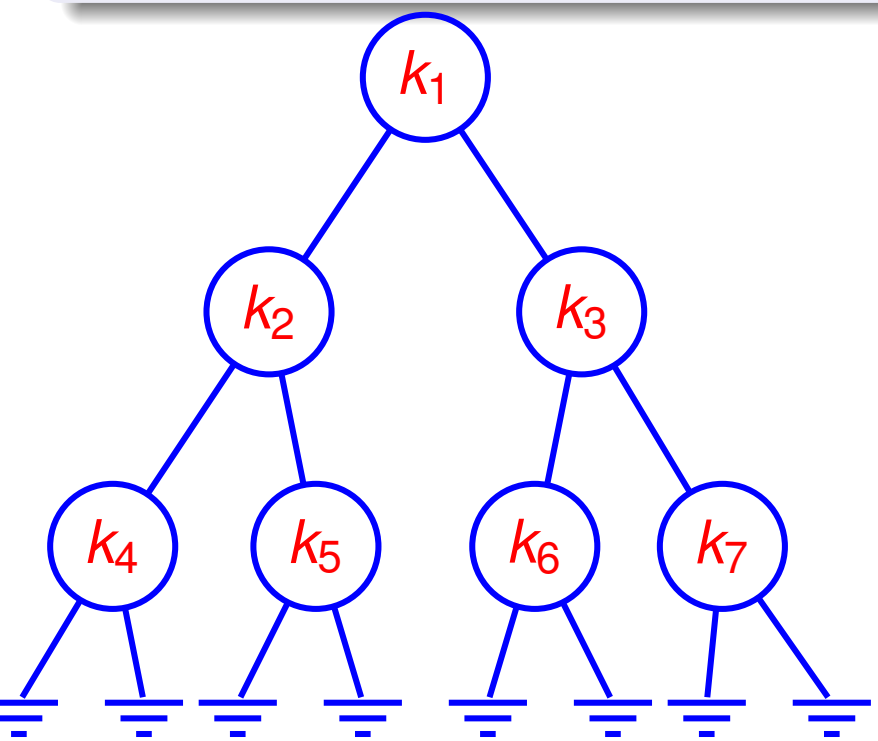
Section outline

1. **Binary trees**
   - Definition of a binary tree
   - Terminology for binary trees
   - Types of binary trees
   - Few binary tree properties
   - An important binary tree property
   - Binary tree traversals
   - Illustration of binary tree traversals
   - Binary tree reconstruction from traversals

Definition of a binary tree

Definition (Rooted binary tree)

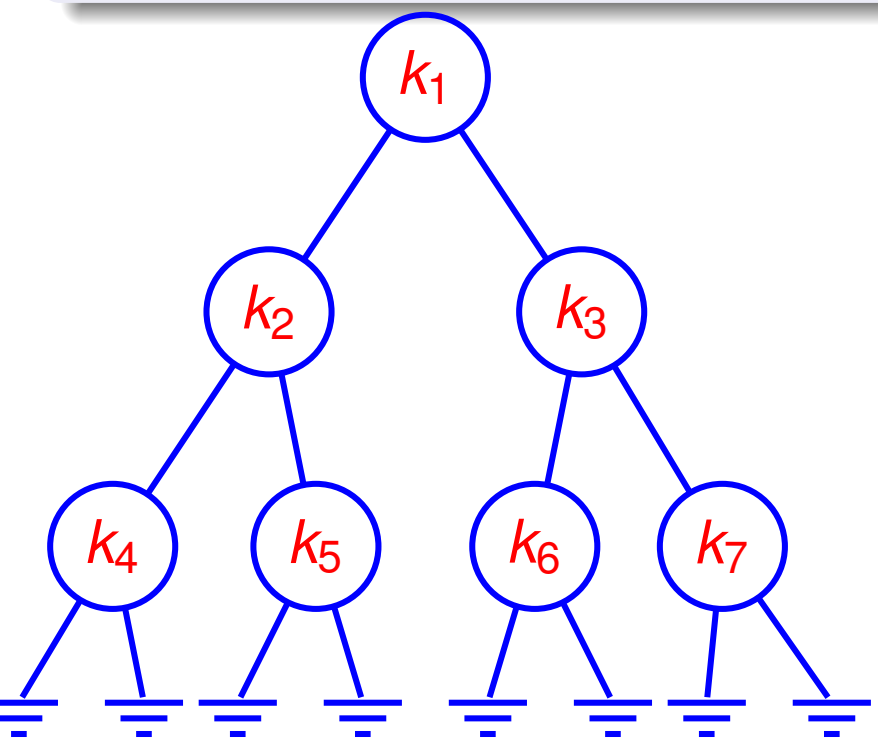
A binary tree is either *empty* or consists of a node called the *root* together with two binary trees called the *left subtree* and the *right subtree*



Definition of a binary tree

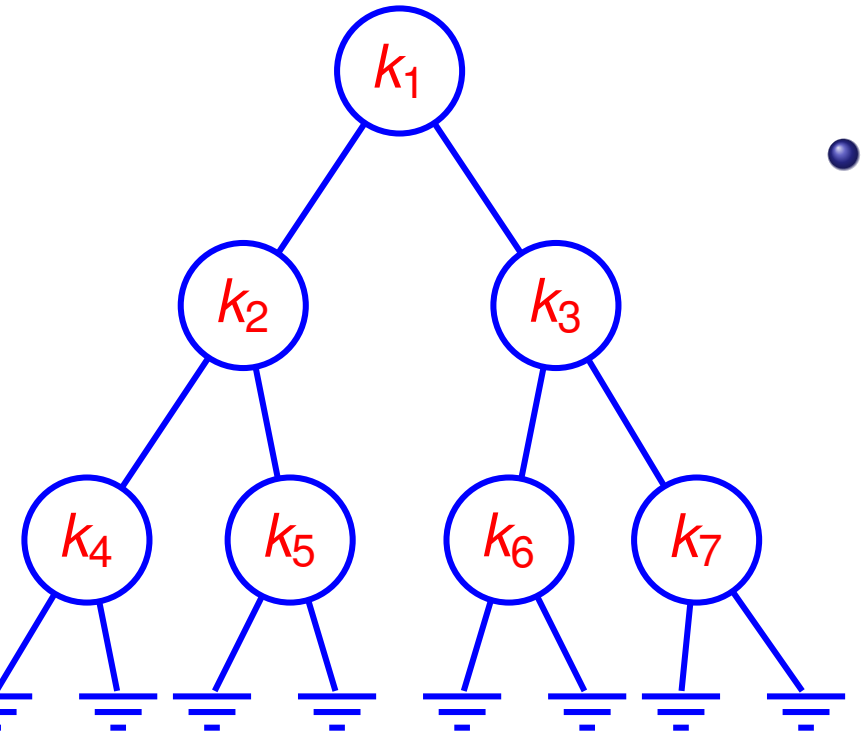
Definition (Rooted binary tree)

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*



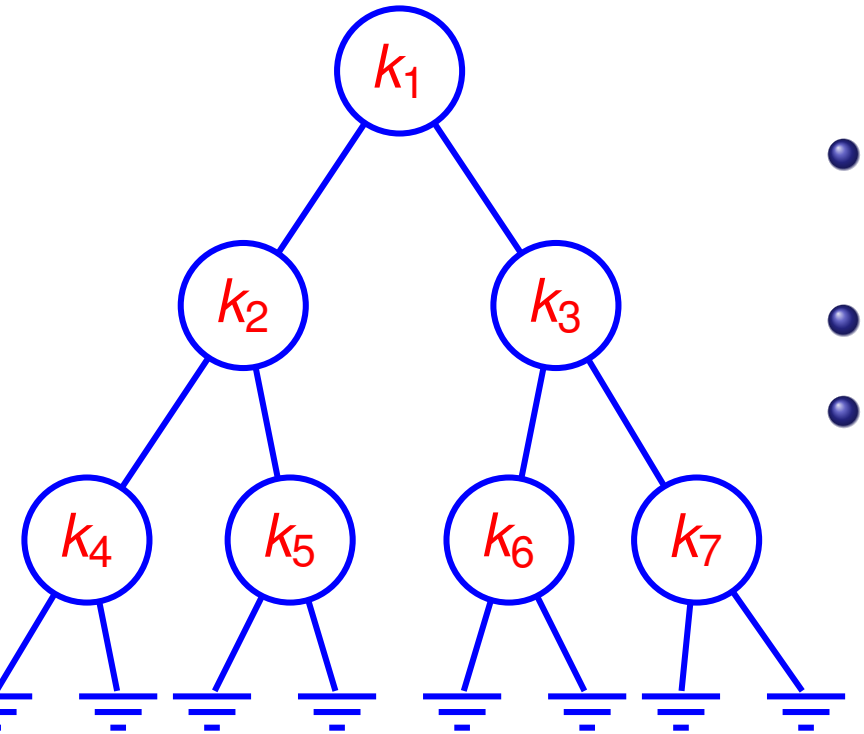

```
typedef int keyTyp;

typedef struct binTTag {
  keyTyp key;
  struct binTTag
    *lChild, *rChild;
} binTreeTyp, *binTreePtr;
```

Terminology for binary trees



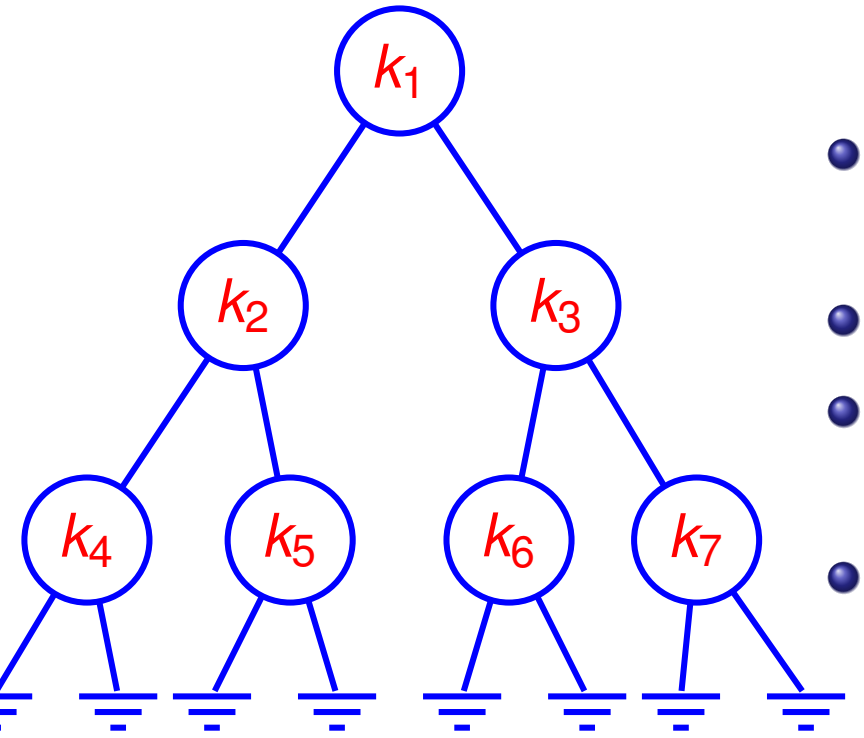
- If c is a left or right child of a node p in the binary tree, then p is said to be the parent of c
- The parent-child relationship may be indicated by a directed edge from p to c

Terminology for binary trees



- If c is a left or right child of a node p in the binary tree, then p is said to be the parent of c
- The parent-child relationship may be indicated by a directed edge from p to c
- The root has no parent (in-degree is 0)
- All other (non-root) nodes have exactly one parent (in-degree is 1)

Terminology for binary trees



- If c is a left or right child of a node p in the binary tree, then p is said to be the parent of c
- The parent-child relationship may be indicated by a directed edge from p to c
- The root has no parent (in-degree is 0)
- All other (non-root) nodes have exactly one parent (in-degree is 1)
- A node which has no children (or only empty binary trees as both children) is termed a *leaf node*
- Any node that is not a leaf node is said to be an *internal node*

More binary tree terminology

- The *depth* of a node *n* is the length of the path (counting the edges) from the root to the node
  The root node is at depth zero

More binary tree terminology

- The *depth* of a node *n* is the length of the path (counting the edges) from the root to the node

  The root node is at depth zero
- The set of all nodes at a given depth is sometimes called a *level* of the tree

More binary tree terminology

- The *depth* of a node *n* is the length of the path (counting the edges) from the root to the node

  The root node is at depth zero
- The set of all nodes at a given depth is sometimes called a *level* of the tree
- The *height* of a tree is the length of the path from the root to the deepest node in the tree

  A (rooted) tree with only one node (the root) has a height of zero

  The null tree is defined to have a height of -1

More binary tree terminology

- The *depth* of a node *n* is the length of the path (counting the edges) from the root to the node

  The root node is at depth zero
- The set of all nodes at a given depth is sometimes called a *level* of the tree
- The *height* of a tree is the length of the path from the root to the deepest node in the tree

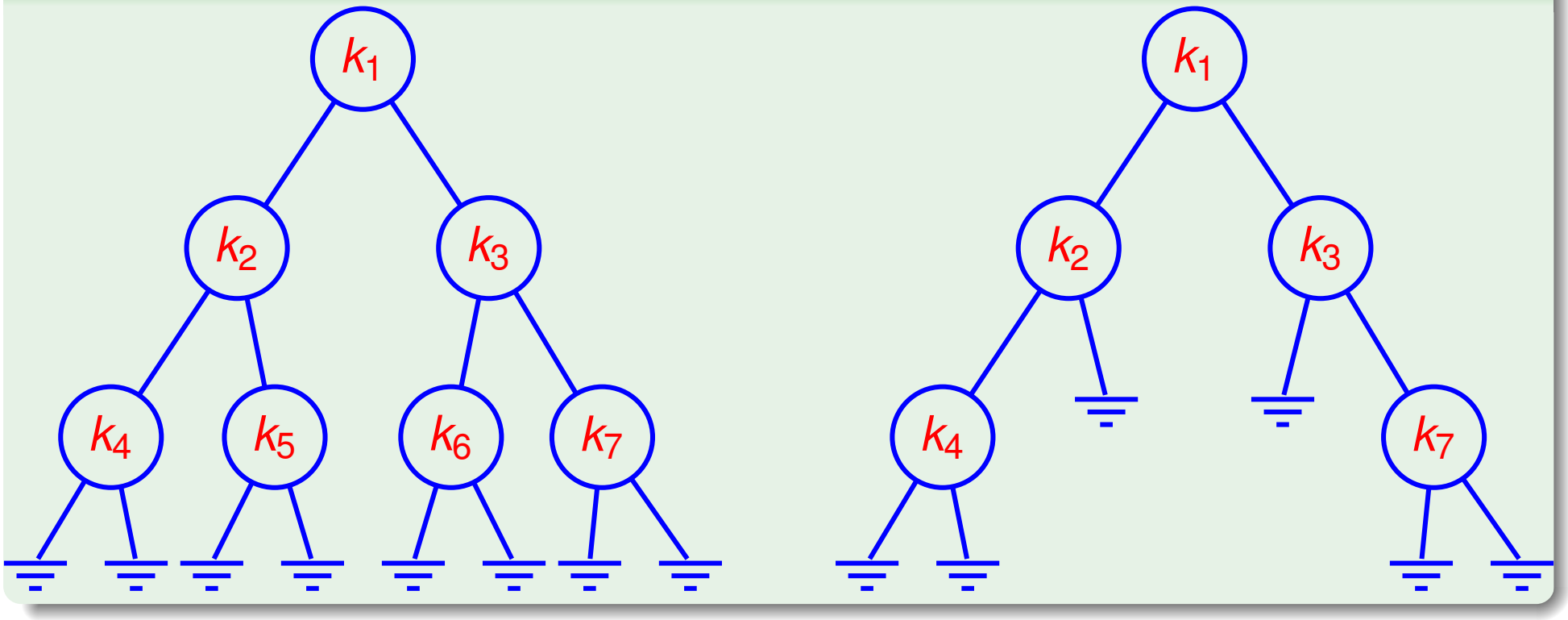
  A (rooted) tree with only one node (the root) has a height of zero

  The null tree is defined to have a height of -1
- Siblings are nodes that share the same parent node
- A node *p* is an ancestor of a node *q* if it exists on the path from *q* to the root.

  The node *q* is then termed a descendant of *p*
- The *size* of a node is the number of descendants it has including itself

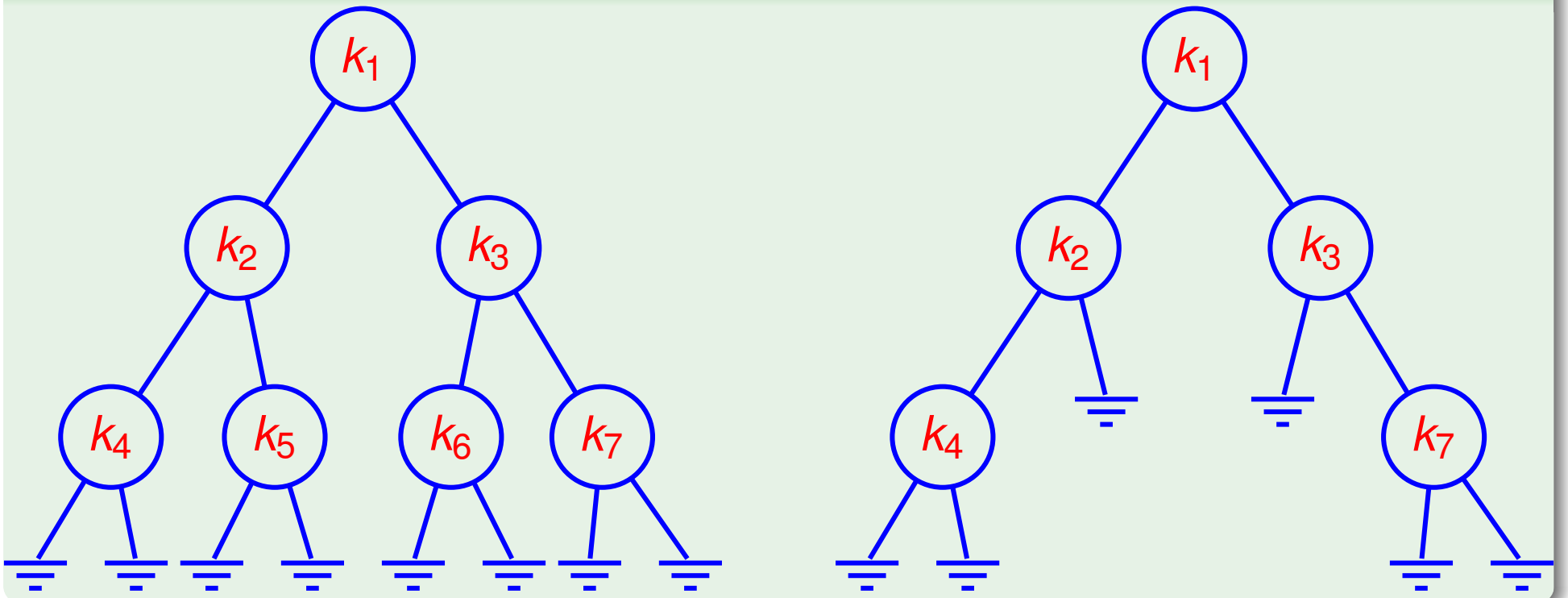
Another binary tree illustrated

Example (Binary trees with height of 2)



Another binary tree illustrated

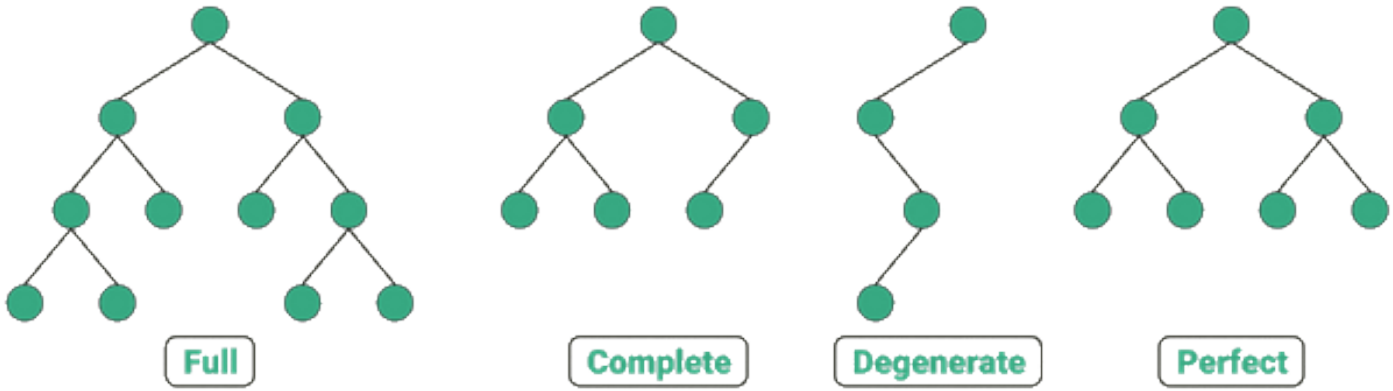
Example (Binary trees with height of 2)



- Empty binary trees (as grounding) is usually not shown explicitly
- A binary tree with all possible nodes up to its height is said to be a *perfect binary tree*
- Maximum number of nodes in a binary tree of height h is $2^{h+1} - 1$

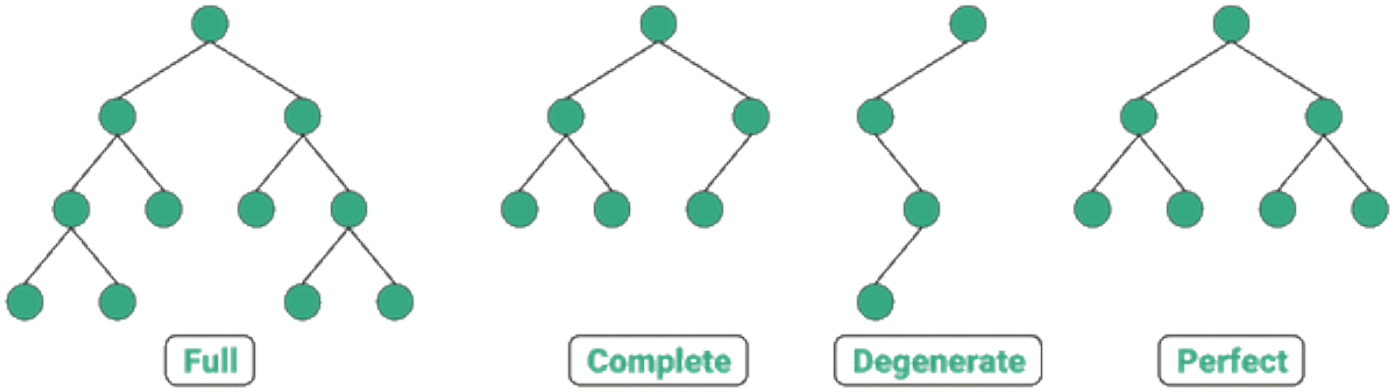
Types of binary trees

- A *full binary tree* (sometimes *proper 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
- 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 *degenerate tree* is a tree where for each parent node, there is only one associated child node



Types of binary trees

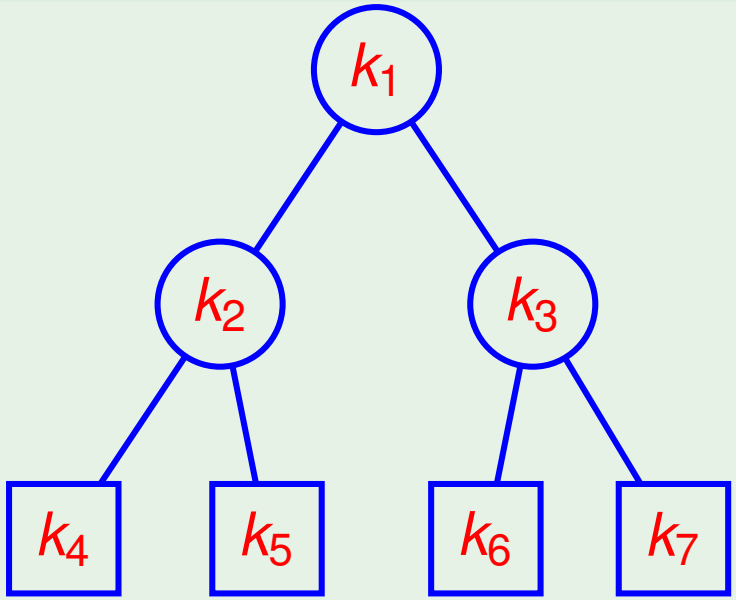
- A *full binary tree* (sometimes *proper binary tree*) is a tree in which every node other than the leaves has two children
- A *perfect binary tree* is a full binary tree in which all leaves are at the same depth or same level
- A *complete binary tree* is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible
- A *degenerate tree* is a tree where for each parent node, there is only one associated child node

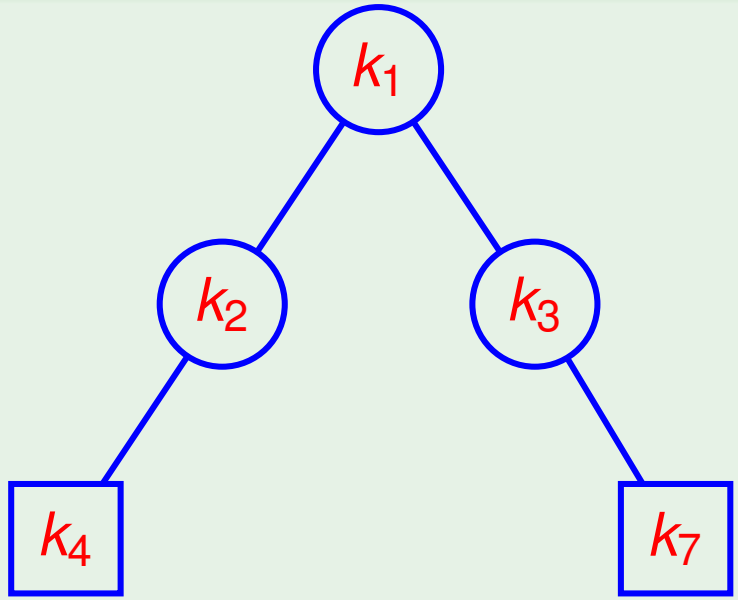


- A *heap* (to be studied later) is a complete binary tree

Binary trees showing leaf nodes

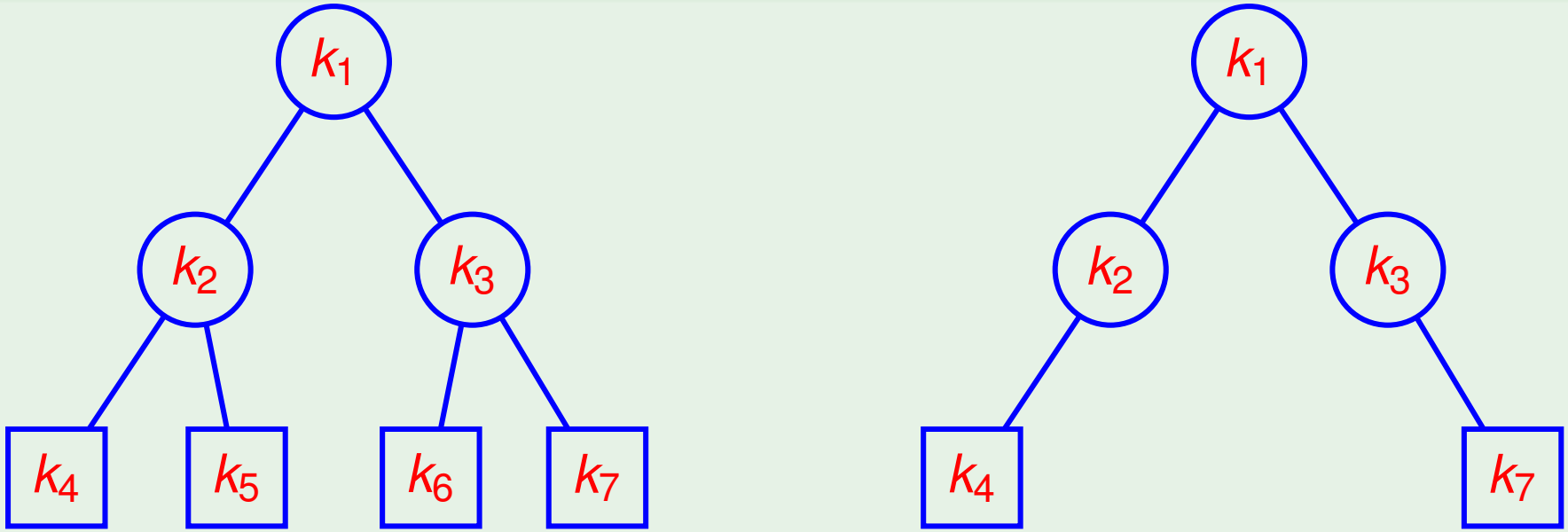
Example (Binary trees with height of 2)





Binary trees showing leaf nodes

Example (Binary trees with height of 2)



- Maximum number of leaf nodes in a binary tree of height h is 2^h

Few binary tree properties

- Let the height of the tree be h
- The number of nodes n in a perfect binary tree is $2^{h+1} - 1$
- The number of nodes in a complete binary tree is at least 2^h and at most $2^{h+1} - 1$
- The number of leaf nodes in a perfect binary tree is 2^h

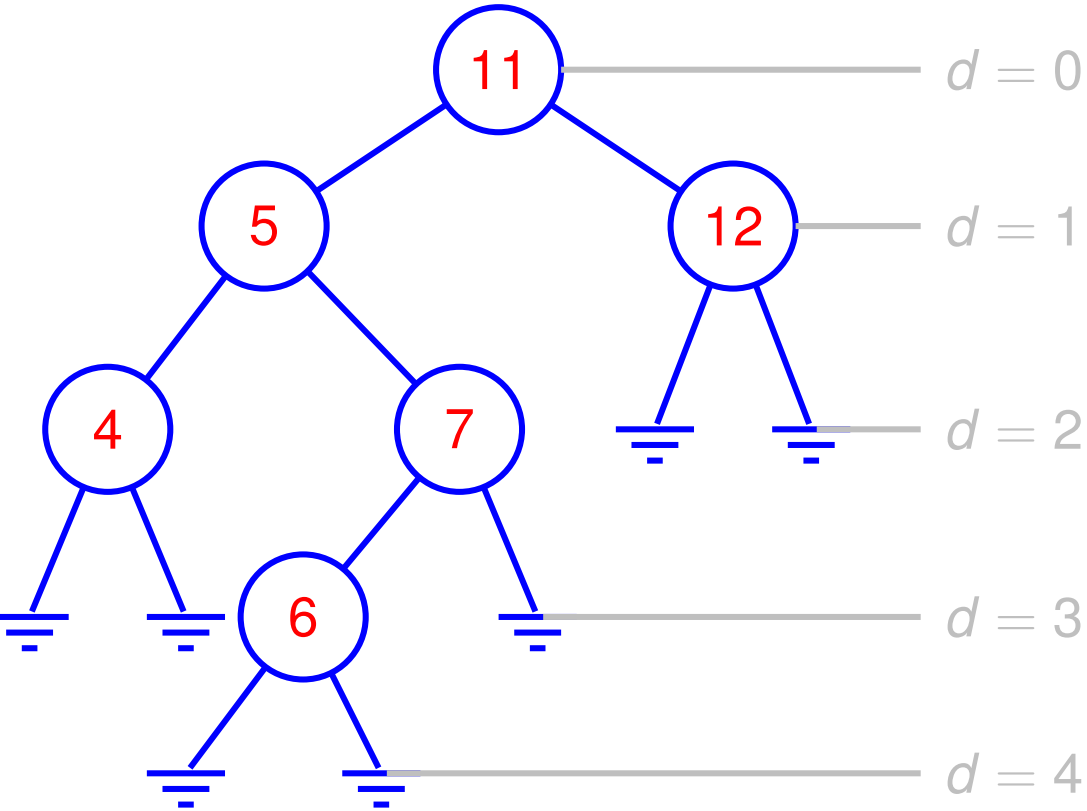
Few binary tree properties

- Let the height of the tree be h
- The number of nodes n in a perfect binary tree is $2^{h+1} - 1$
- The number of nodes in a complete binary tree is at least 2^h and at most $2^{h+1} - 1$
- The number of leaf nodes in a perfect binary tree is 2^h
- If L is the number of leaf nodes in a perfect binary tree, then the tree has $2L - 1$ nodes
- The number of null links (absent children of nodes) in any binary tree of n nodes is $n + 1$

Few binary tree properties

- Let the height of the tree be h
- The number of nodes n in a perfect binary tree is $2^{h+1} - 1$
- The number of nodes in a complete binary tree is at least 2^h and at most $2^{h+1} - 1$
- The number of leaf nodes in a perfect binary tree is 2^h
- If L is the number of leaf nodes in a perfect binary tree, then the tree has $2L - 1$ nodes
- The number of null links (absent children of nodes) in any binary tree of n nodes is $n + 1$
- The number internal nodes (non-leaf nodes) in a complete binary tree of n nodes is $\lfloor \frac{n}{2} \rfloor$
- For any non-empty binary tree with n_0 leaf nodes and n_2 nodes of degree 2, $n_0 = n_2 + 1$

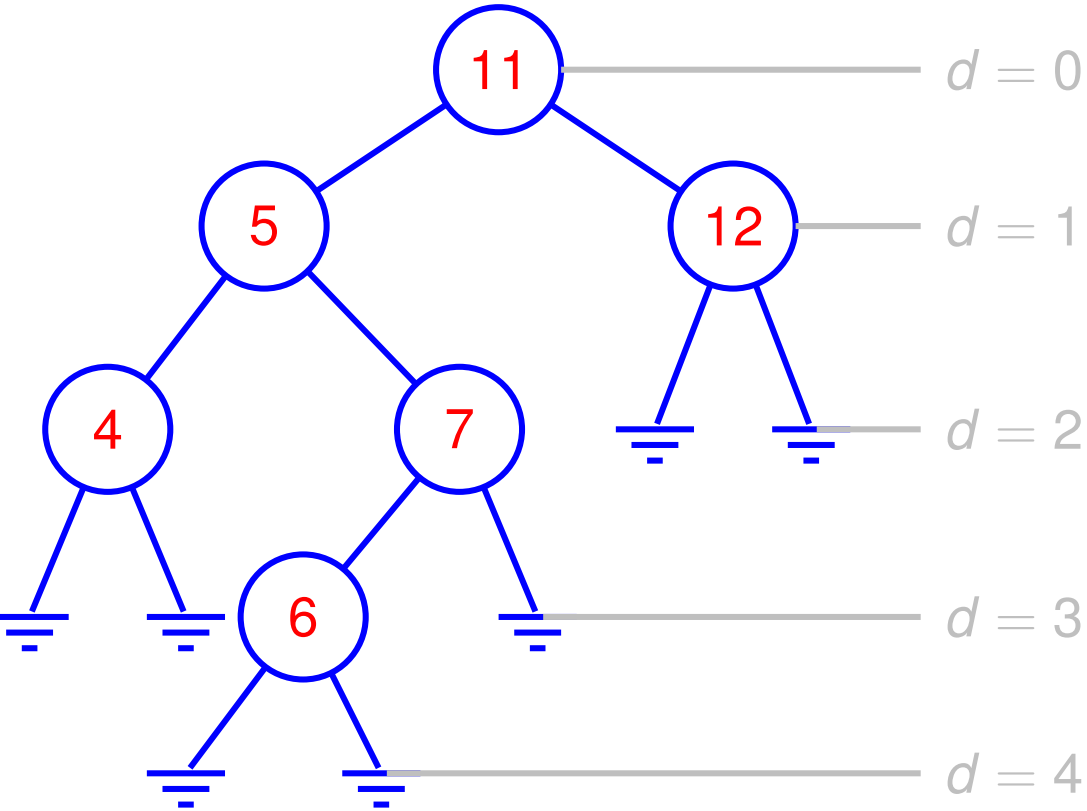
An important binary tree property



- Let T be *any* binary with n nodes
- $|\text{NULLs}| = 1 + |\text{BinTNodes}|$
- $I_n \equiv$ sum of levels of all BinTNodes
- $E_n \equiv$ sum of levels of all NULLs

- $n = 6$, $|\text{NULLs}| = n + 1 = 7$, $I = 0+1\times2+2\times2+3\times1 = 9$, $E = 2\times2+3\times3+4\times2 = 21 = I + 2n$
- Claim: $E_n = I_n + 2n$

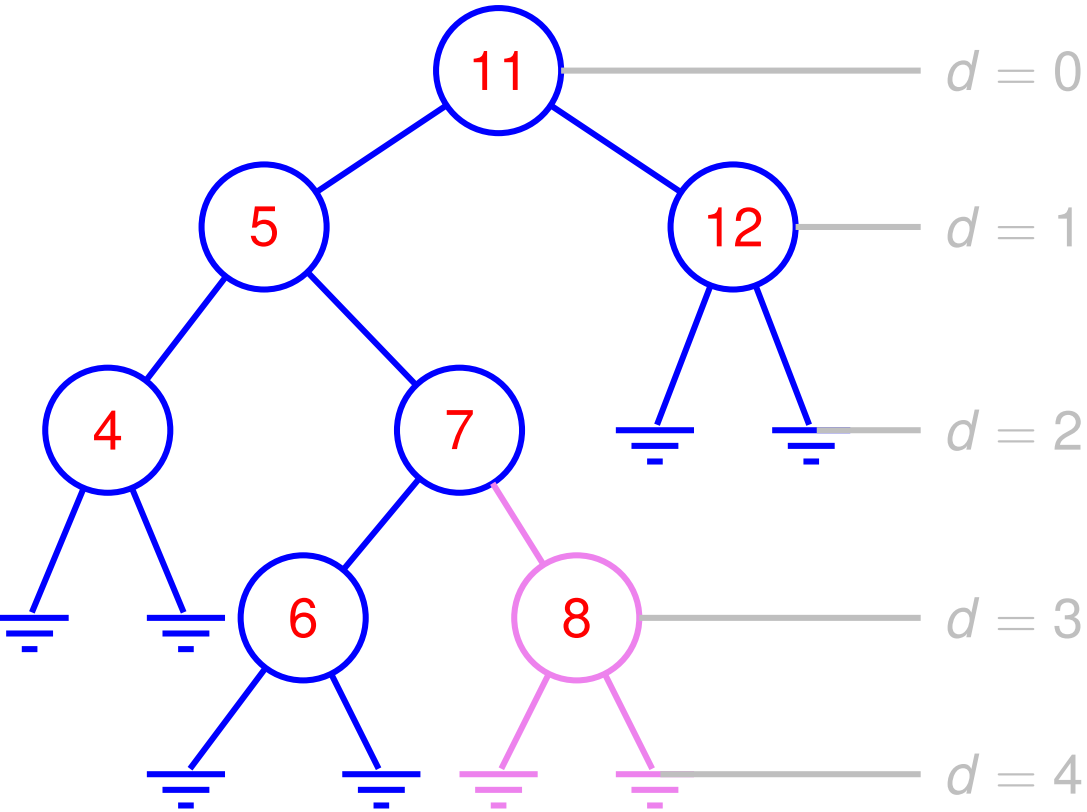
An important binary tree property



- Let T be *any* binary with n nodes
- $|\text{NULLs}| = 1 + |\text{BinTNodes}|$
- $I_n \equiv$ sum of levels of all BinTNodes
- $E_n \equiv$ sum of levels of all NULLs

- $n = 6$, $|\text{NULLs}| = n + 1 = 7$, $I = 0+1\times2+2\times2+3\times1 = 9$, $E = 2\times2+3\times3+4\times2 = 21 = I + 2n$
- Claim: $E_n = I_n + 2n$
- Base (binary tree of one node): $I_1 = 0, E_1 = 1 + 1 = 2$, so $E_1 = I_1 + 2\times1$ ✓
- Assume it holds for any binary tree of n nodes
- Inductive hypothesis: $E_n = I_n + 2n$ (5.1)

An important binary tree property (contd.)



- Replace any NULL with a binary tree node
- Let d be the level of that NULL, so that $E_n = E'_n + d$

- Following hold in the modified tree with $n+1$ nodes
- $I_{n+1} = I_n + d$
- $E_{n+1} = \underbrace{(E_n + 1)}_{\text{l-NULL}} + \underbrace{(d+1)}_{\text{r-NULL}}$
- $E_{n+1} = E_n + d + 2$
- $E_{n+1} = \underbrace{(I_n + 2n)}_{E_n} + d + 2$ [by inductive hypothesis]
- $E_{n+1} = (I_n + d) + 2n + 2$
- $E_{n+1} = I_{n+1} + 2(n+1)$ ■

Binary tree traversals

Depth first traversal Nodes of one subtree are visited before visiting nodes of the other subtree

Breadth first traversal Nodes of one level are visited before visiting nodes of the next level

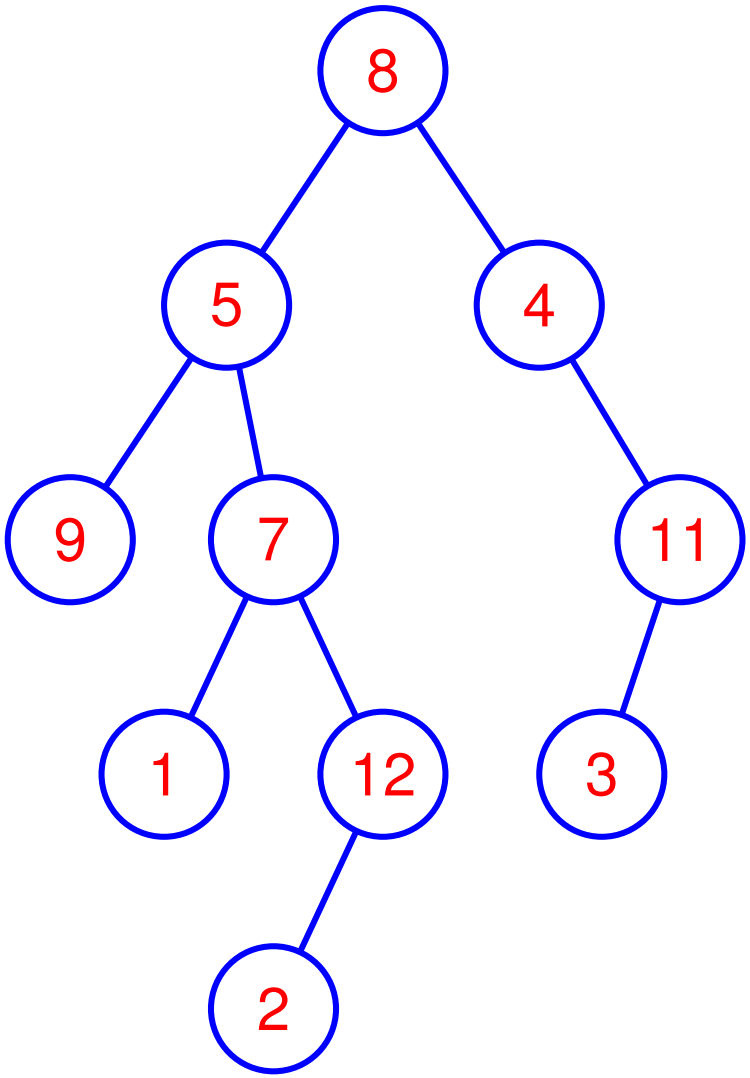
There are three types of depth first traversals:

Preorder Visit root, visit left subtree in preorder, visit right subtree in preorder

Postorder Visit left subtree in postorder, right subtree in postorder, then the root

Inorder Visit left subtree in inorder, then the root, then the right subtree in inorder

Illustration of binary tree traversals



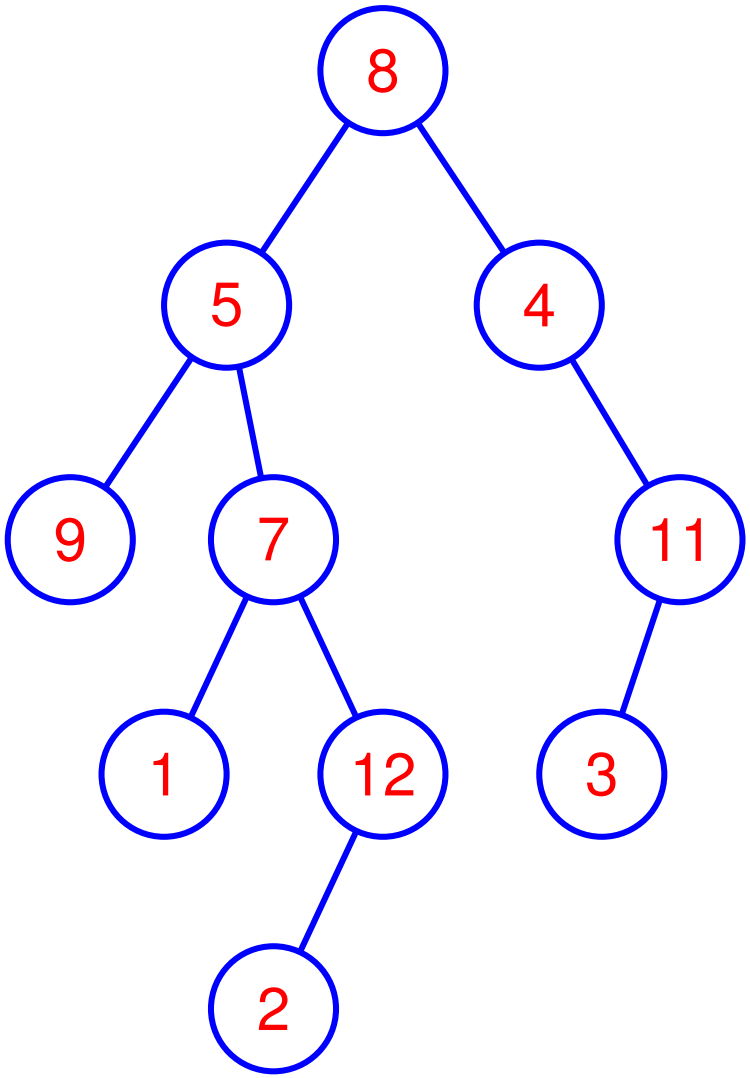
Breadth first: 8, 5, 4, 9, 7, 11, 1, 12, 3, 2

PreOrder: 8, 5, 9, 7, 1, 12, 2, 4, 11, 3

InOrder: 9, 5, 1, 7, 2, 12, 8, 4, 3, 11

PostOrder: 9, 1, 2, 12, 7, 5, 3, 11, 4, 8

Illustration of binary tree traversals



Breadth first: 8, 5, 4, 9, 7, 11, 1, 12, 3, 2

PreOrder: 8, 5, 9, 7, 1, 12, 2, 4, 11, 3

InOrder: 9, 5, 1, 7, 2, 12, 8, 4, 3, 11

PostOrder: 9, 1, 2, 12, 7, 5, 3, 11, 4, 8

It is possible to reconstruct the binary tree from the inorder and any one of the preorder or postorder traversals – by a recursive divide-and-conquer scheme.

Recursive definitions for binary tree traversal

binTree_preOrder(binTreePtr tP)

Base cases

CBa tP==NULL // tree is empty

ABa return; // nothing to do

Inductive/recursive case

CIa tP != NULL

AIa1 visit(tP); // such a print the key

AIa2 binTree_preOrder(tP->lChild);

AIa3 binTree_preOrder(tP->rChild);

Some problems that can be handled using traversals

- Finding the sum/maximum/minimum of keys
- Finding the height of the tree
- Printing the keys

Binary tree reconstruction from traversals

binTreePtr binTree_fromInPre(int n, int inorder[], int preorder[])

Base cases

CBa n==0

ABa return NULL;

CBb n==1

ABb return binTreeLNode(inorder[0]);

Binary tree reconstruction from traversals

binTreePtr binTree_fromInPre(int n, int inorder[], int preorder[])

Base cases

CBa n==0

ABa return NULL;

CBb n==1

ABb return binTreeLNode(inorder[0]);

Example

PreOrder: 8, 5, 9, 7, 1, 12, 2, 4, 11, 3

InOrder: 9, 5, 1, 7, 2, 12, 8, 4, 3, 11

Inductive/recursive case

Cla n>1

Ala1 rIdx=locate(preorder[0], inorder)

Ala2 lSz=irIdx rSz=n-(rIdx+1);

Ala3 lChild=binTree_fromInPre(lSz, inorder, preorder+1);

Ala4 rChild=binTree_fromInPre(rSz, inorder+lSz+1, preorder+lSz+1);

Ala5 return binTreeNode(inorder[0],lChild,rChild);