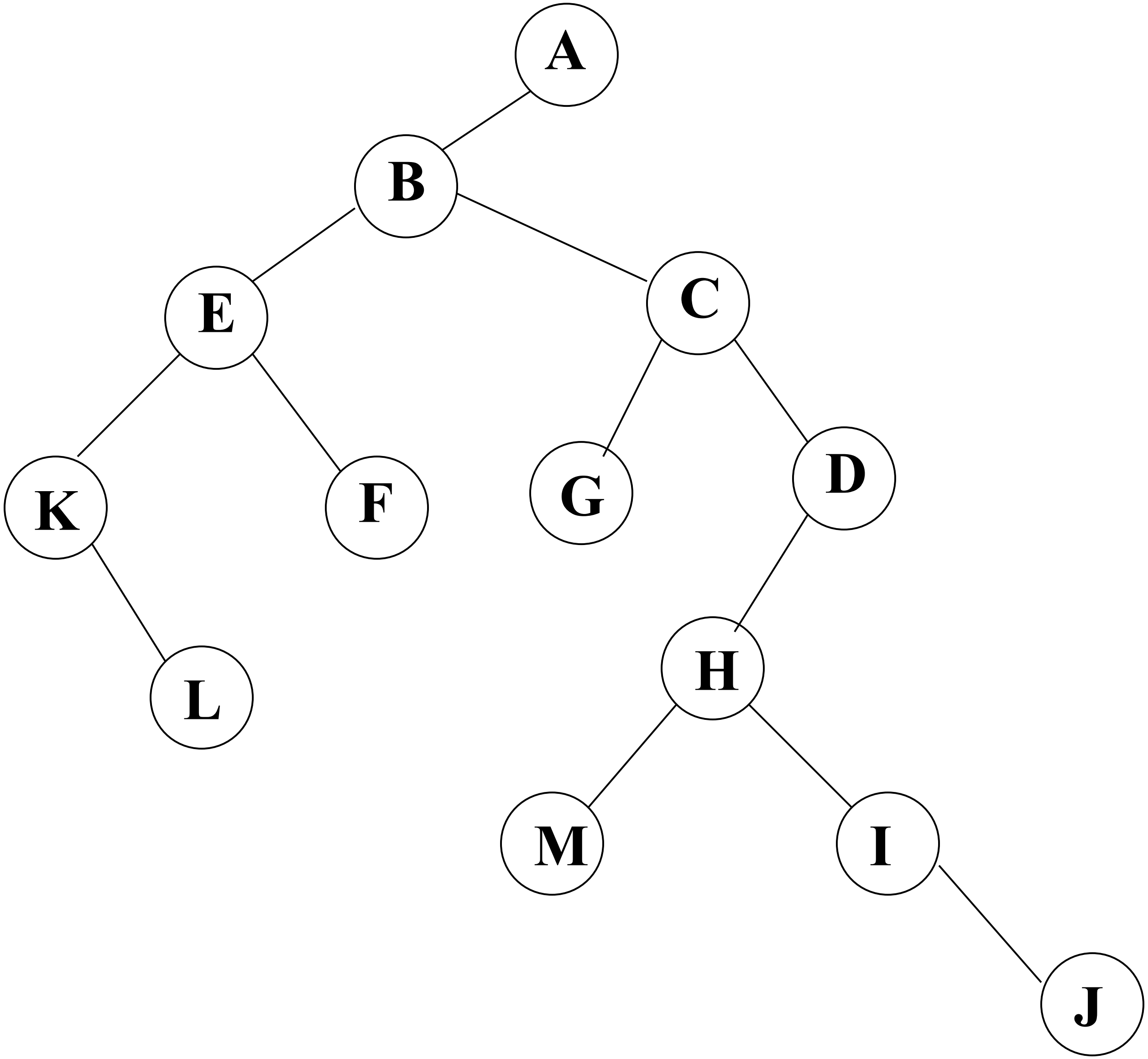
Binary Trees

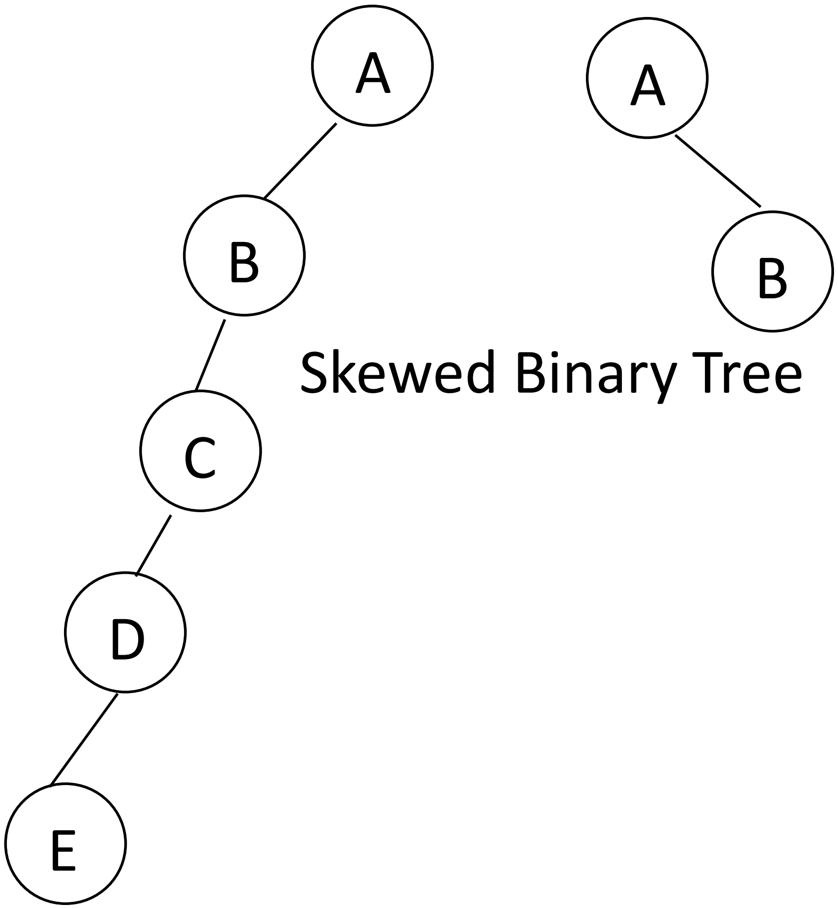
Binary Trees

- A special class of trees: max degree for each node is 2 (children)
- Recursive definition: A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called *the left subtree* and *the right subtree*.
- A binary tree is composed of zero or more nodes
  - 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

Example

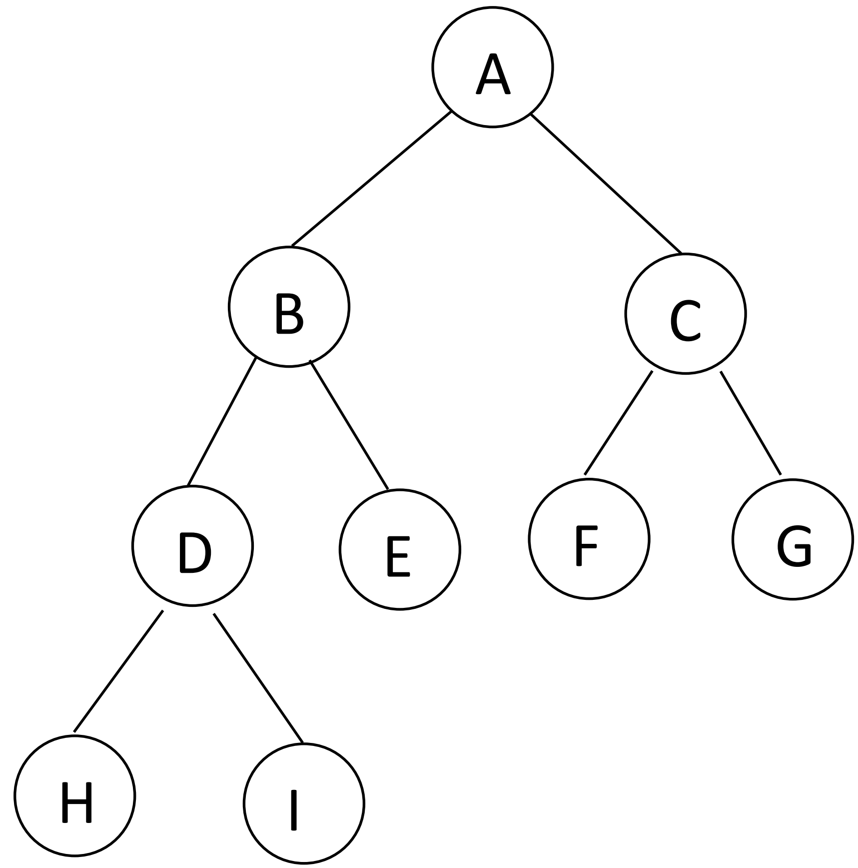


Samples of Binary Trees

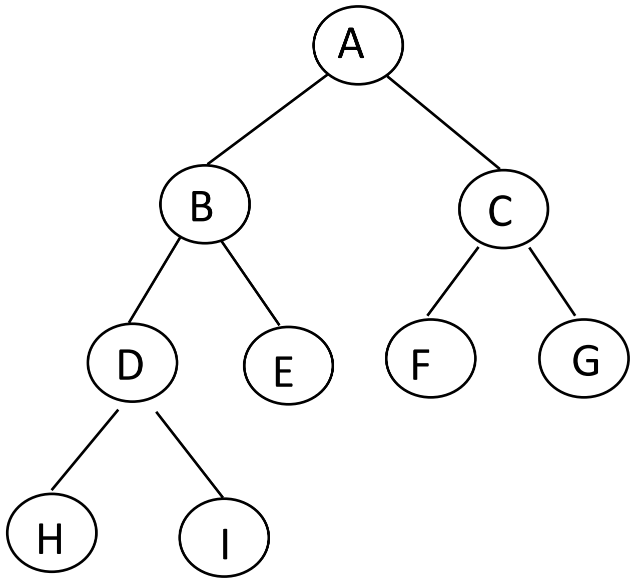


Skewed Binary Tree

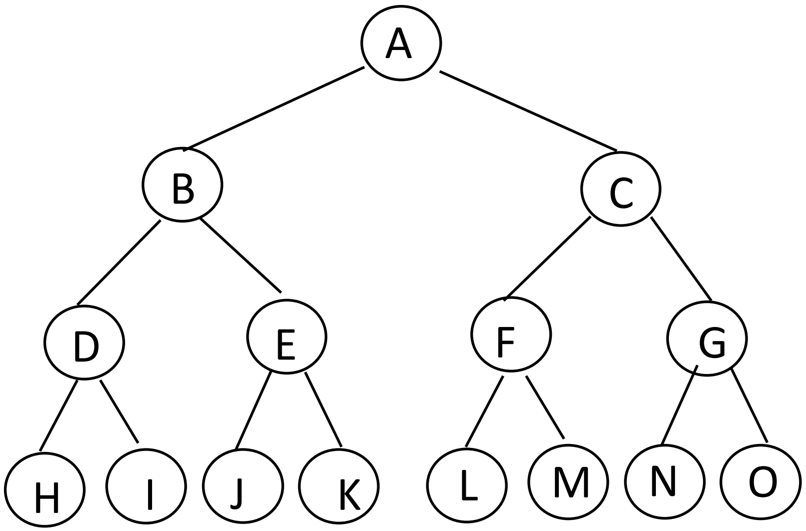
Complete Binary Tree



Full BT vs. Complete BT



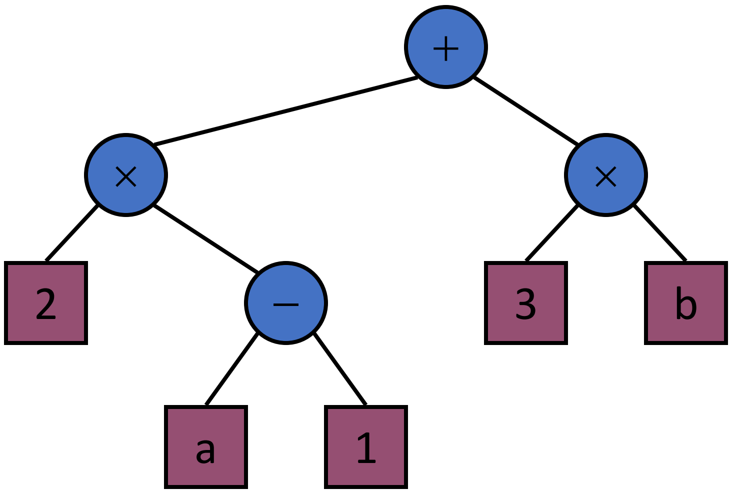
Complete binary tree



Full binary tree of depth 3

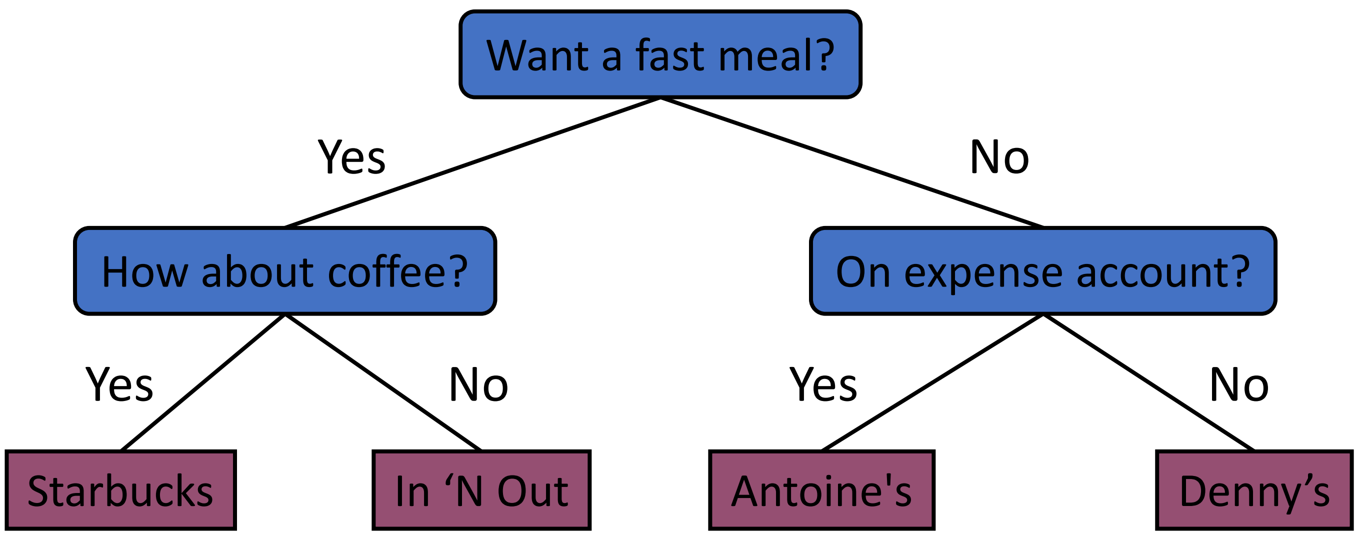
Arithmetic Expression Tree

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



Decision Tree

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

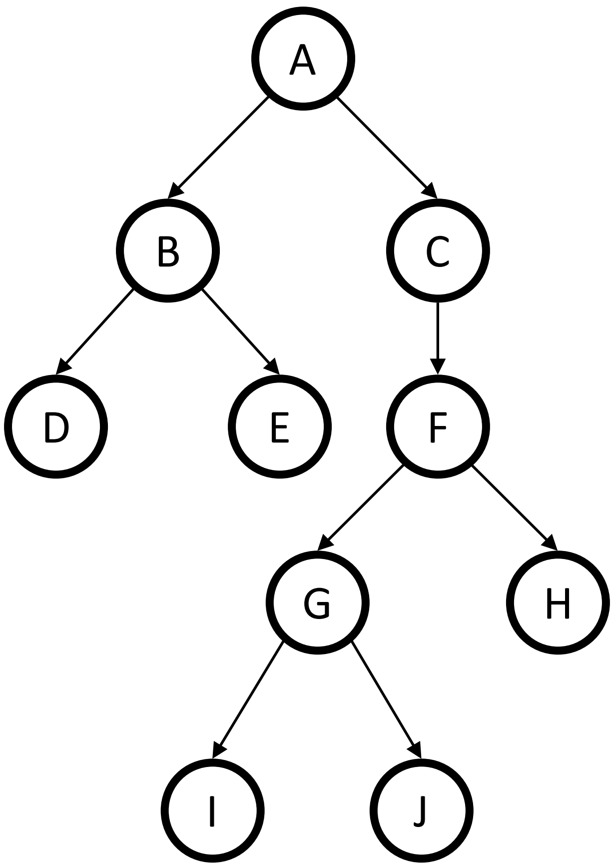


Binary Trees

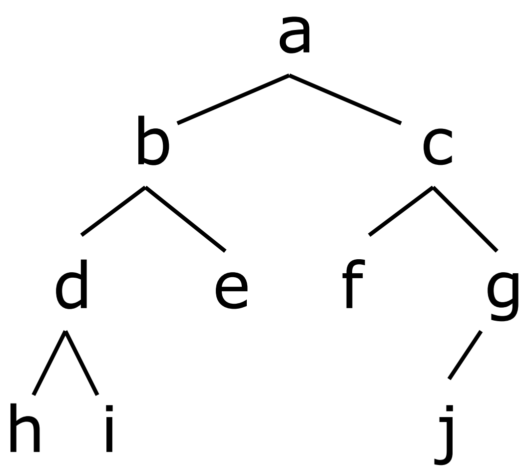
- Properties

  Notation: depth(tree) = MAX {depth(leaf)} = height(root)

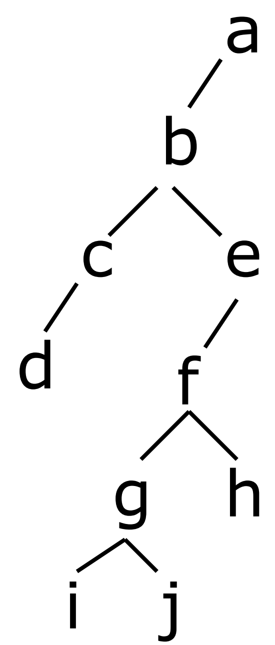
  - max # of leaves = $2^{depth(tree)}$
  - max # of nodes = $2^{depth(tree)+1} - 1$
  - max depth = n-1



Balance



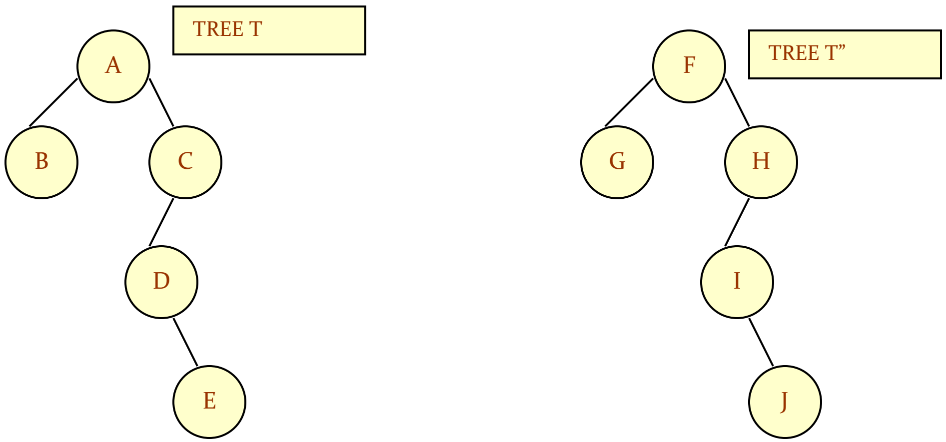
A balanced binary tree



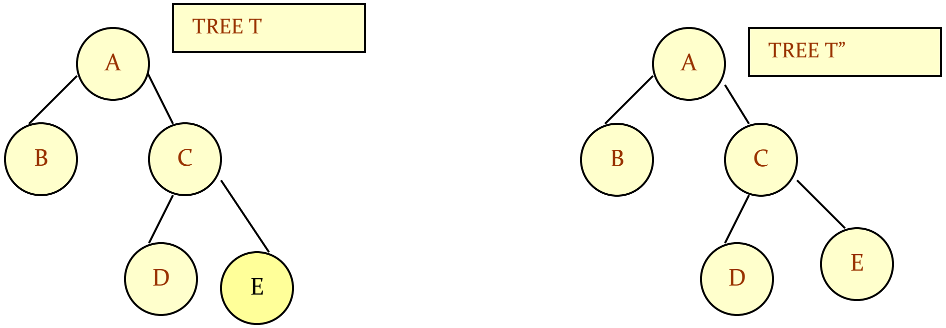
An unbalanced binary tree

In most applications, a reasonably-balanced binary tree is desirable.

- Similar binary trees: Given two binary trees T and T’ are said to be similar if both these trees have the same structure.

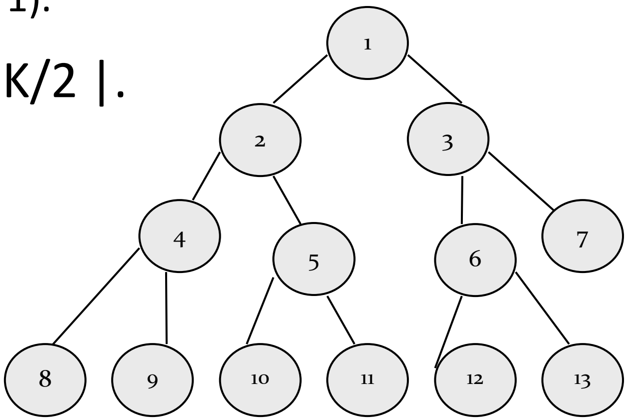


- Copies of binary trees: Two binary trees T and T’ are said to be copies if they have similar structure and same content at the corresponding nodes.



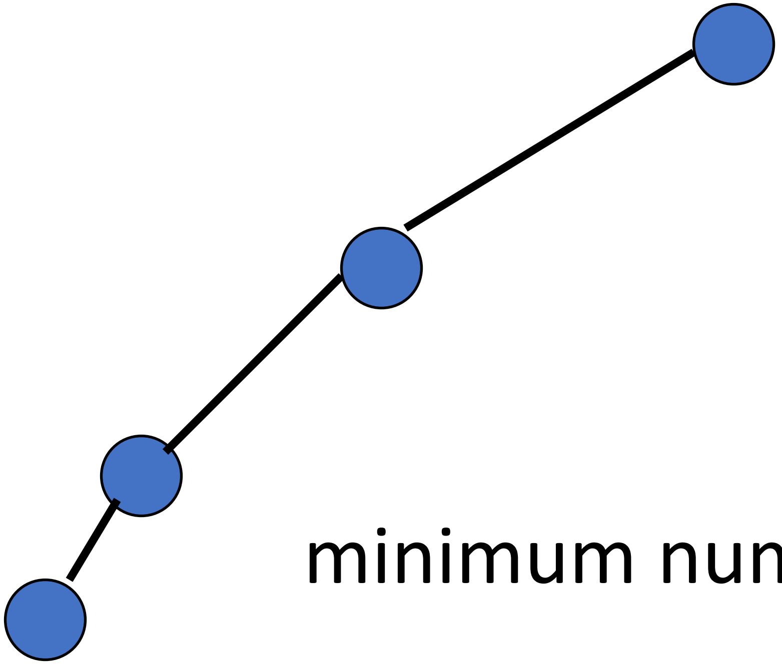
Complete Binary Trees

- A *complete binary tree* is a binary tree which satisfies two properties.
  - First, in a complete binary tree every level, except possibly the last, is completely filled.
  - Second, all nodes appear as far left as possible
- In a complete binary tree T_n, there are exactly n nodes and level r of T can have at most 2^r nodes.
- The formula to find the parent, left child and right child can be given as:
- If K is a parent node, then its left child can be calculated as 2 * K and its right child can be calculated as 2 * K + 1.
  - For example, the children of node 4 are 8 (2*4) and 9 (2* 4 + 1).
- Similarly, the parent of node K can be calculated as | K/2 |.



Minimum Number Of Nodes

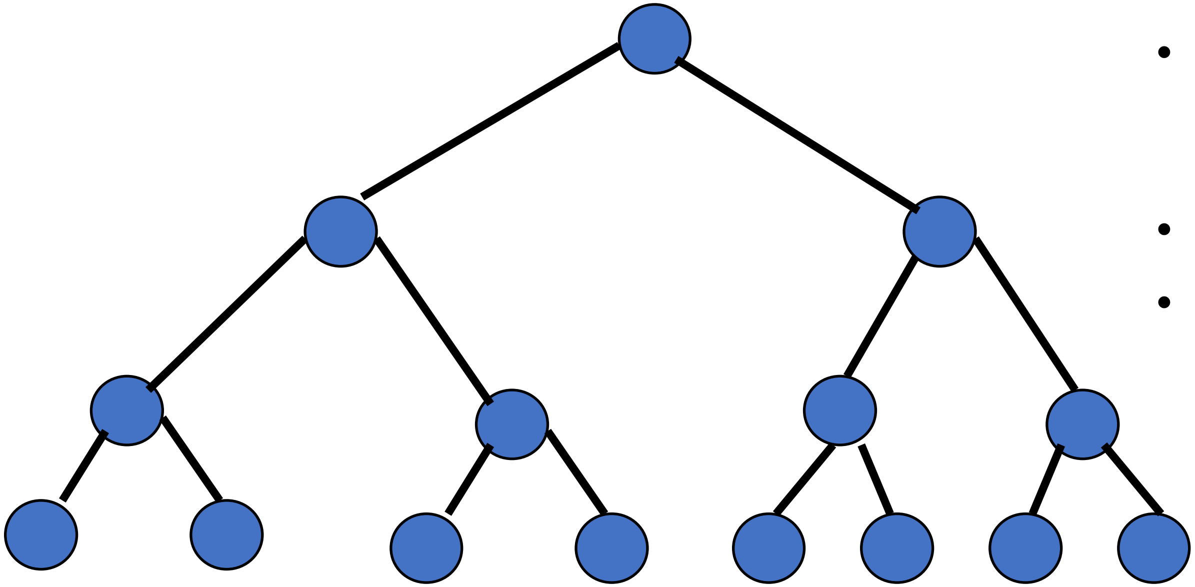
- Minimum number of nodes in a binary tree whose height is h.
- At least one node at each of first h levels.



minimum number of nodes is h+1

Maximum Number Of Nodes

- All possible nodes at first h levels are present.



- Let n be the number of nodes in a binary tree whose height is h.
- $h+1 <= n <= 2^{h+1} - 1$
- $\log_2(n+1) <= h+1 <= n$

Maximum number of nodes

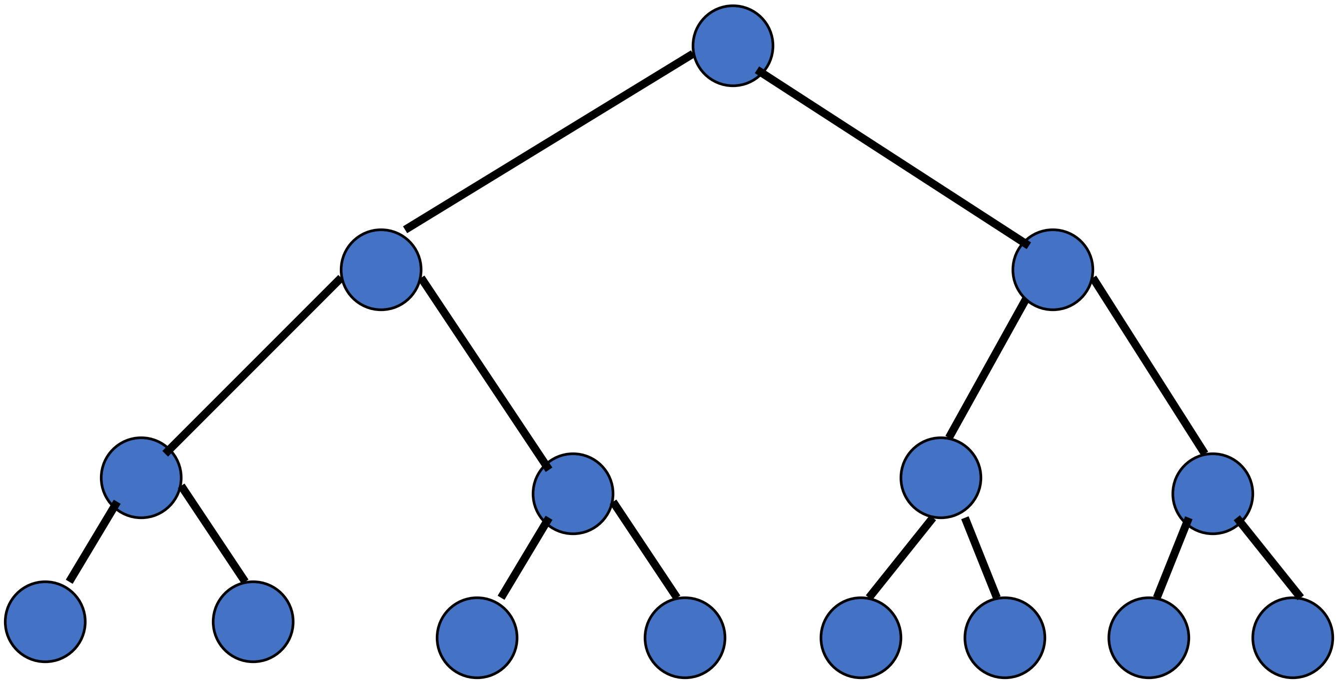
$S = 1 + 2 + 4 + 8 + \ldots + 2^h$

$2S = 2 + 4 + 8 + \ldots + 2^h + 2^{h+1} = S - 1 + 2^{h+1}$

$S = -1 + 2^{h+1}$

Full Binary Tree

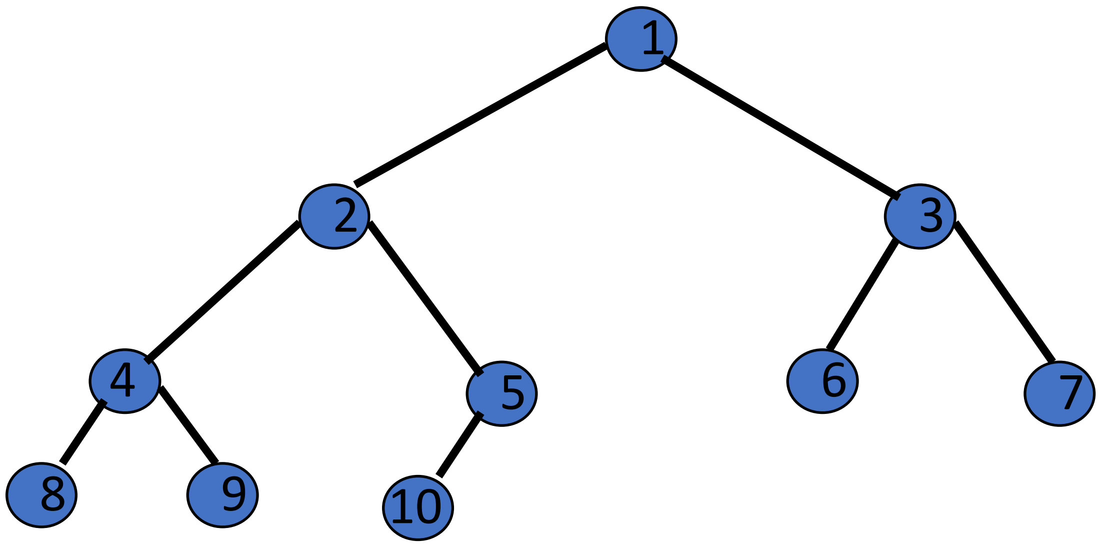
- A full binary tree of a given height h has $2^{h+1} – 1$ nodes.



Height 3 full binary tree.

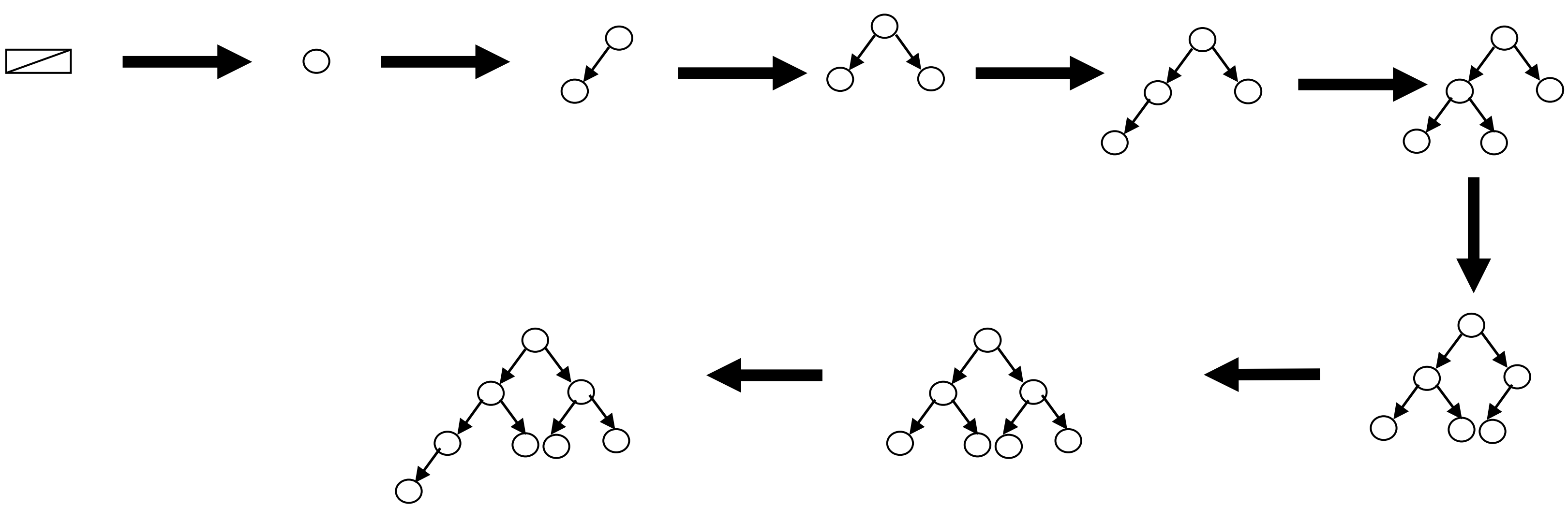
Example

In a complete binary tree every level, except possibly the last, is filled, and all nodes in the last level are as far left as possible. It can have between 1 and 2^h nodes at the last level h.



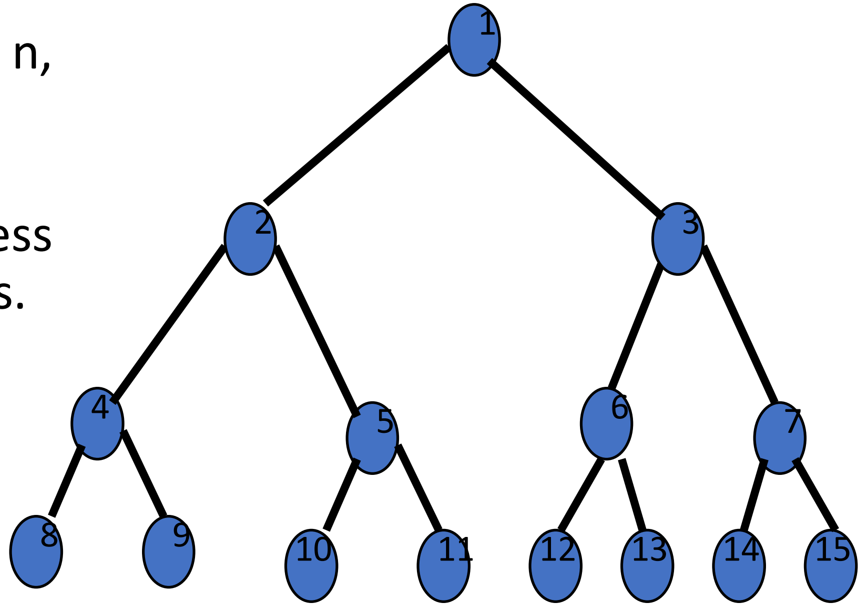
Complete binary tree with 10 nodes.

Example showing the growth of a complete binary tree:



Numbering Nodes in A Full Binary Tree

- Number the nodes 1 through $2^{h+1} – 1$.
- Number by levels from top to bottom.
- Within a level number from left to right.
  - Parent of node i is node i / 2, unless i = 1.
- Node 1 is the root and has no parent.
  - Left child of node i is node 2i, unless 2i > n, where n is the number of nodes.
    - If 2i > n, node i has no left child.
  - Right child of node i is node 2i+1, unless 2i+1 > n, where n is the number of nodes.
    - If 2i+1 > n, node i has no right child.



Representation of Binary tree

- Implicit and explicit representation
  - Implicit representation
    - Sequential / Linear representation, using arrays.
  - Explicit representation
    - Linked list representation, using pointers.

Sequential Representation

- This representation is static.
- Block of memory for an array is allocated, before storing the actual tree.
- Once the memory is allocated, the size of the tree will be fixed.
- Nodes are stored level by level, starting from the zeroth level.
- Root node is stored in the starting memory location, as the first element of the array.

Sequential Representation

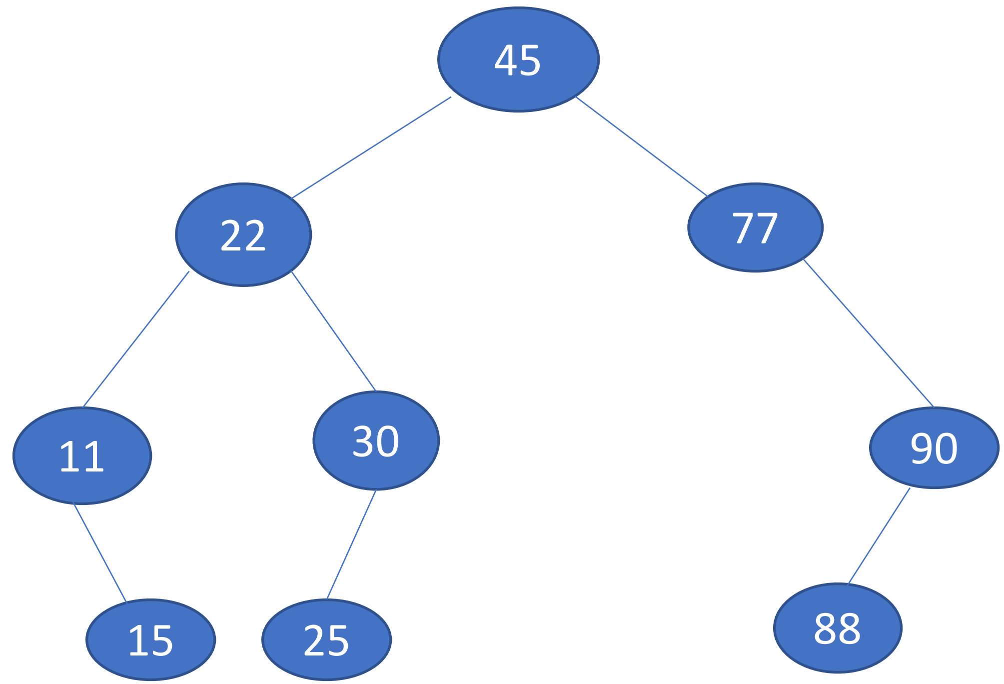
- Consider a linear array TREE

Rules for storing elements in TREE are:

1. The root R of T is stored in location **1 (index 0)**.
2. For any node with index i 1<i<=n:
    1. PARENT(i)= i/2
       For the node when i=1,there is no parent.
    2. LCHILD(i)=2*i
       If 2* i >n, then i has no left child
    3. RCHILD(i)=2*i+1
       If 2*i+1>n, then i has no right child.

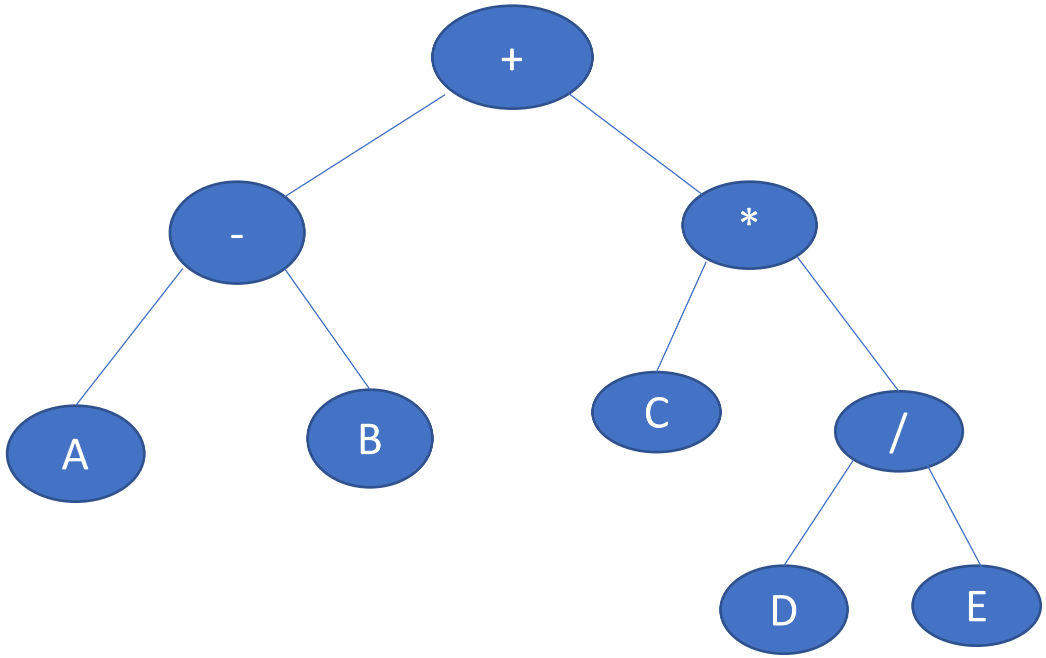
Sequential Representation - Example

- TREE[1] will store the data of the root element.
- The children of a node K will be stored in location (2*K) and (2*K+1).
- The maximum size of the array TREE is given as $(2^{h+1}-1)$, where h is the height of the tree.
- An empty tree or sub-tree is specified using NULL. If TREE[1] = NULL, then the tree is empty.



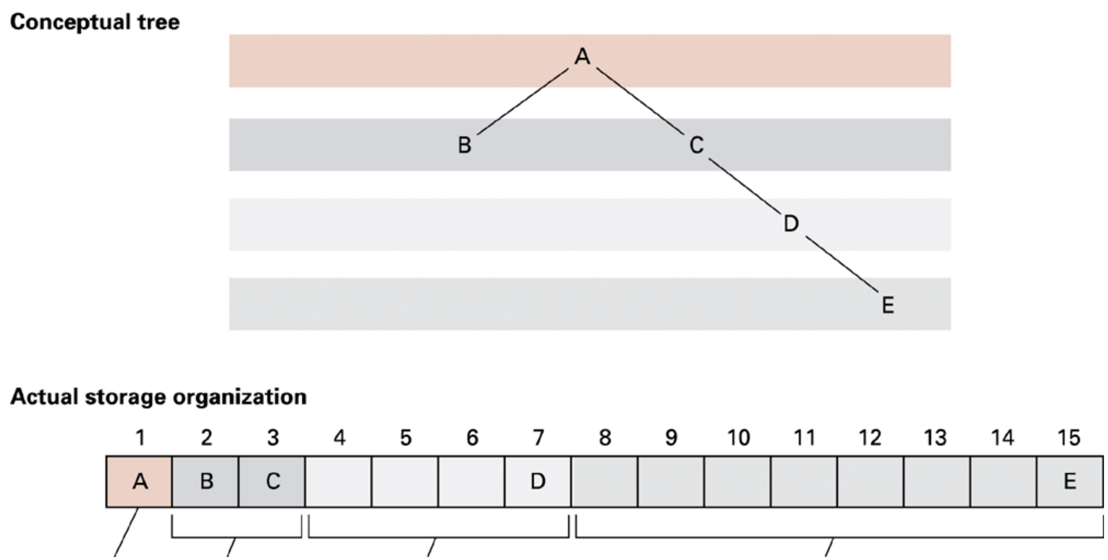
1	2	3	4	5	6	7	8	9	10	11	12	13	14
45	22	77	11	30		90		15	25				88

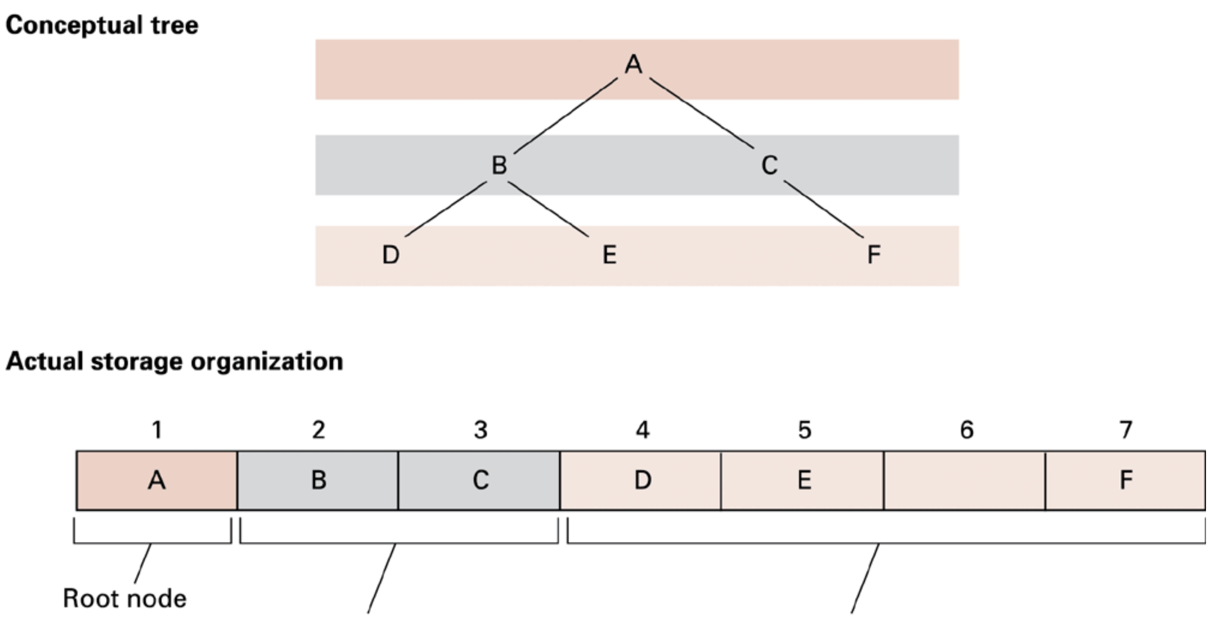
Sequential Representation - Example



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
+	-	*	A	B	C	/							D	E

Example


Conceptual tree
A
B
C
D
E
Actual storage organization
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
A B C D E


Conceptual tree
A
B
C
D
E
F
Actual storage organization
1 2 3 4 5 6 7
A B C D E F
Root node

Sequential Representation - Advantages:

1. Any node can be accessed from any other node by calculating the index.
2. Here, data are stored simply without any pointers to their successor or predecessor.
3. In programming languages, where dynamic memory allocation is not possible (like BASIC, FROTRAN), only array representation is possible.
4. Inserting a new node and deletion of an existing node is easy.

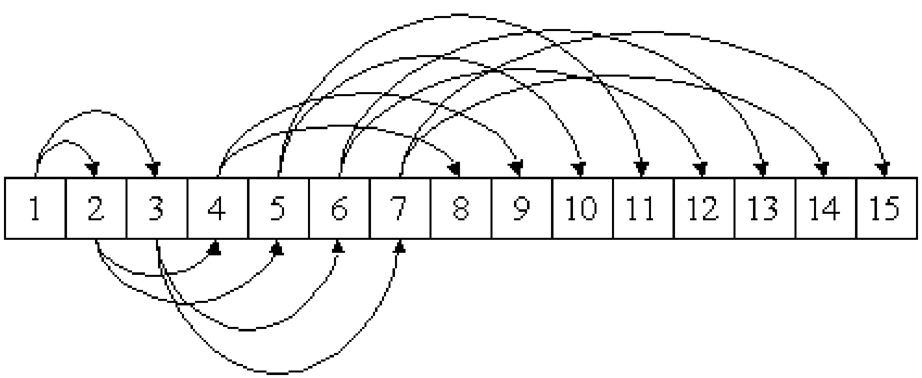
Sequential Representation - Disadvantages:

1. Other than full binary trees, majority of the array entries may be empty.
2. It allows only static representation. It is not possible to enhance the tree structure, if the array structure is limited.

Array Implementation - Summary

- We can embed the nodes of a binary tree into a one-dimensional array by defining a relationship between the position of each parent node and the position of its children.
  1. left_child of node i is 2*i
  2. right_child of node i is 2*i+1
  3. parent of node i is i/2 (integer division)
- How much space is required for the array to represent a tree of depth d?

$2^{d+1} – 1$ (must assume full tree)



Draw the tree structure whose array representation is given:

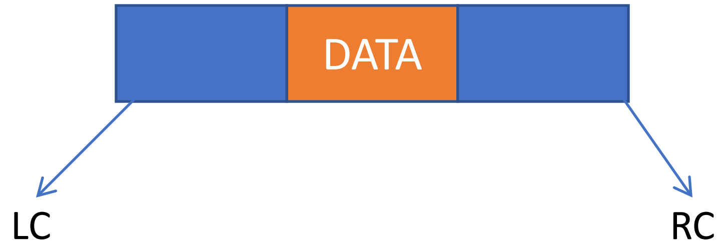
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	B		C				D								E

Linked Representation

- Each binary tree node is represented as an object whose data type is binaryTreeNode.
- The space required by an n node binary tree is n * (space required by one node).

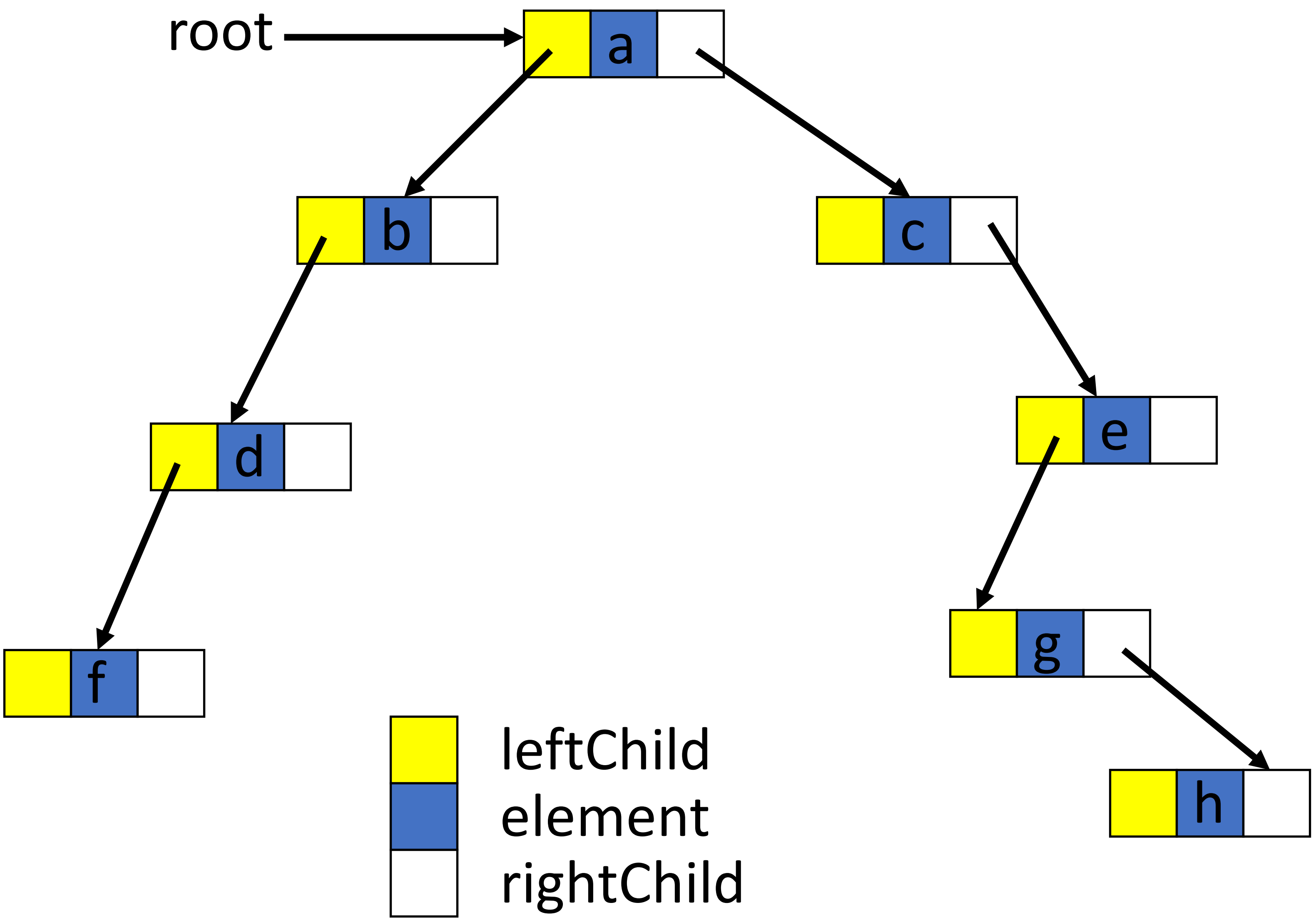
```
template <class T>
struct binaryTreeNode
{
   T element;
   binaryTreeNode<T> *leftChild,
                     *rightChild;
   binaryTreeNode()
      {leftChild = rightChild = NULL;}
   // other constructors come here
};
```

- It consist of three parallel arrays DATA, LC and RC



- Each node N of T will correspond to a location K such that:
  - DATA[K] contains the data at the node N
  - LC[K] contains the location of the left child of node N
  - RC[K] contains the location of the right child of node N

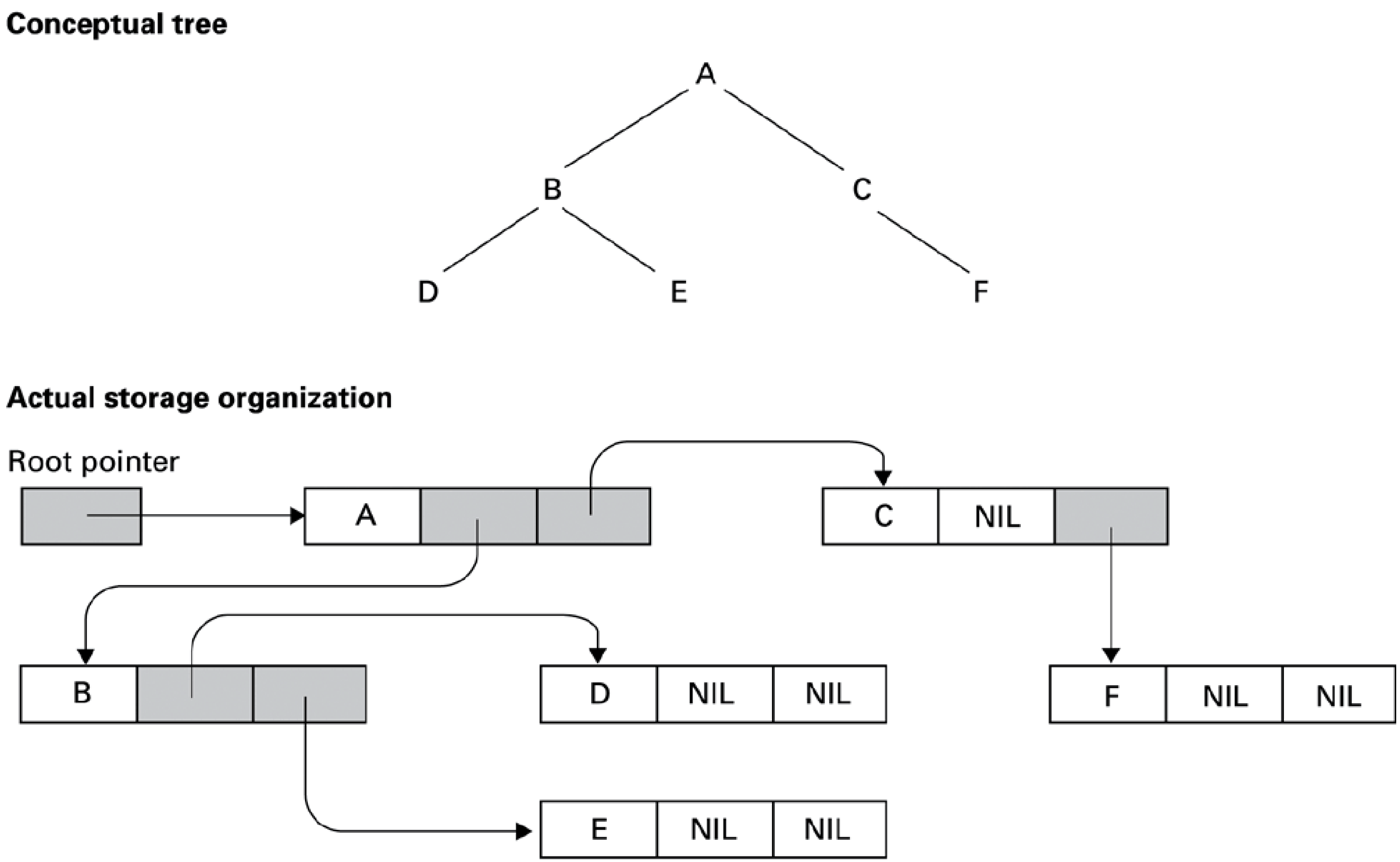
Linked Representation Example



Some Binary Tree Operations

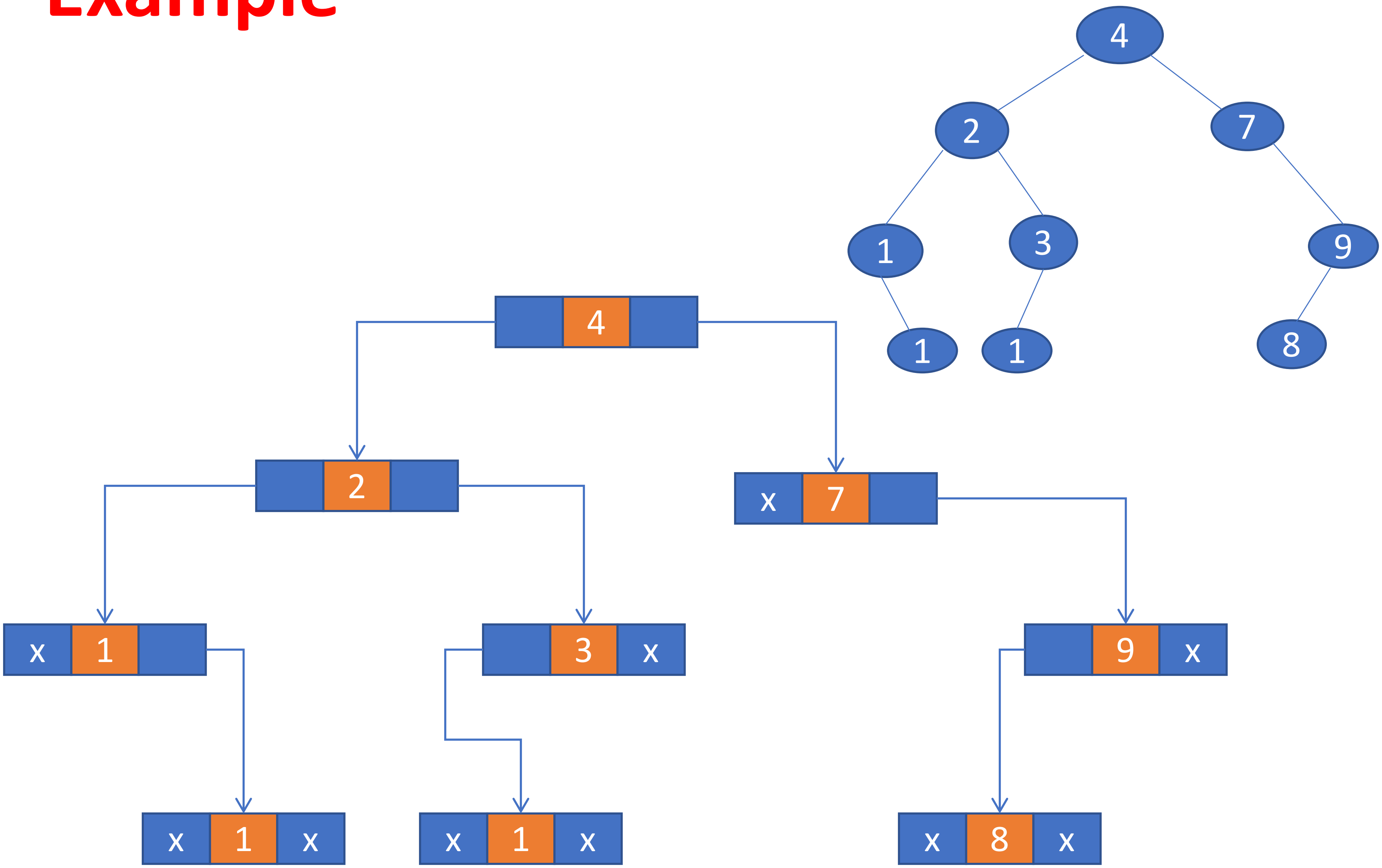
- Determine the height.
- Determine the number of nodes.
- Make a clone.
- Determine if two binary trees are clones.
- Display the binary tree.
- Evaluate the arithmetic expression represented by a binary tree.
- Obtain the infix form of an expression.
- Obtain the prefix form of an expression.
- Obtain the postfix form of an expression.

Example

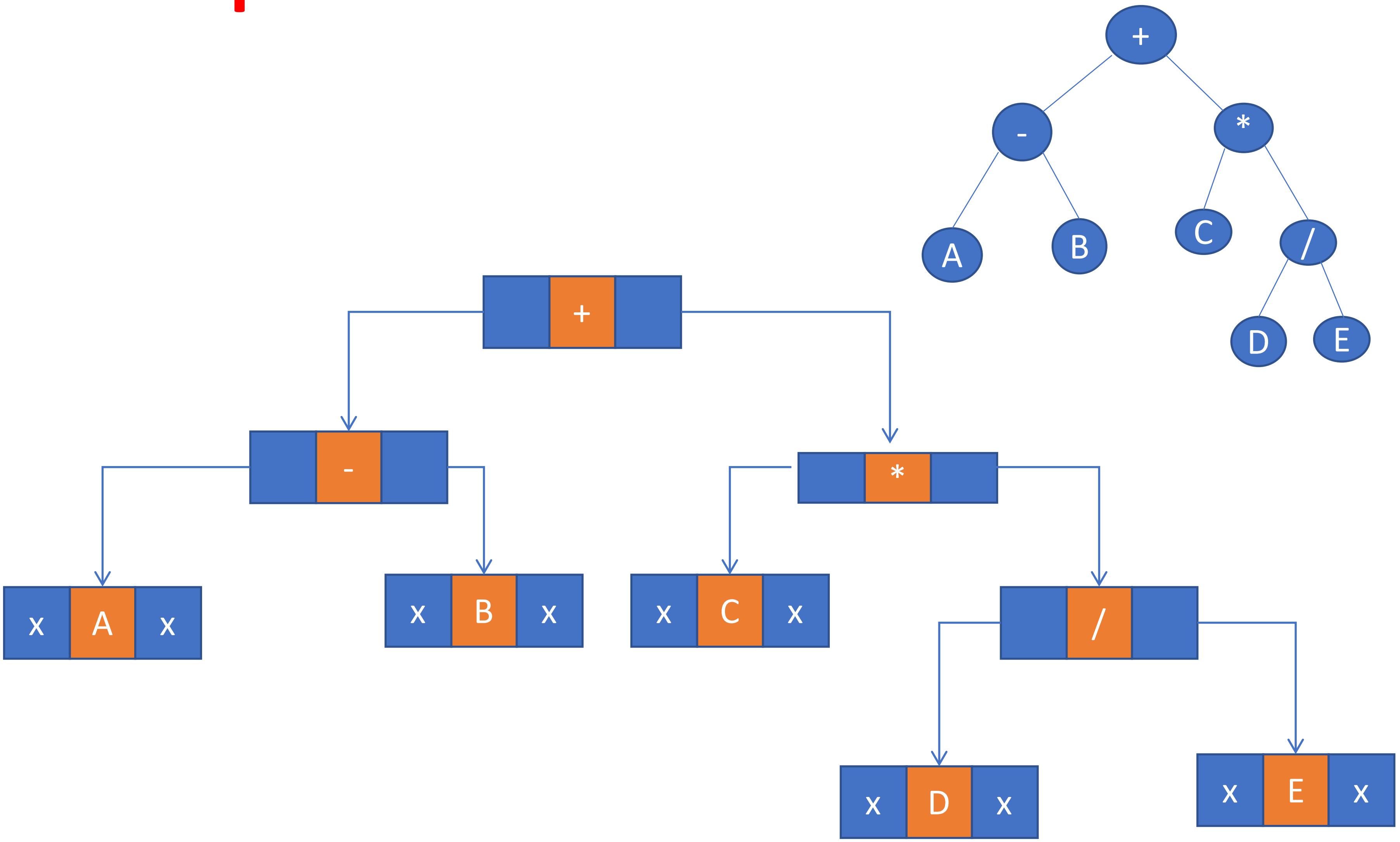


Conceptual tree
A
B
C
D
E
F
Actual storage organization
Root pointer
A
C
NIL
B
D
NIL
NIL
F
NIL
NIL
E
NIL
NIL

Example



Example

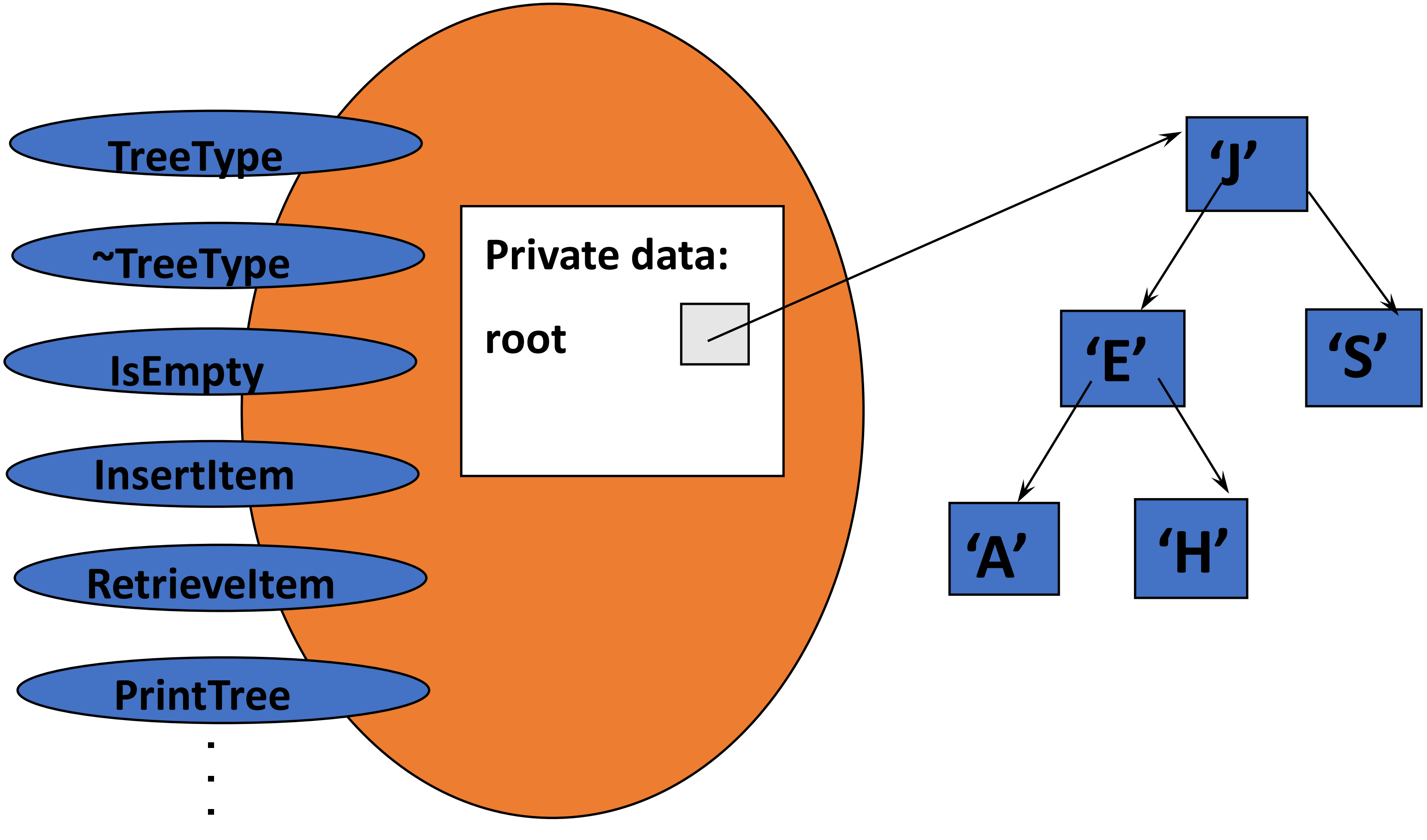


Binary Tree ADT

```
///////////// Accessors ////////////
int size ();
void postOrder(TreeNode* ptr);
void postOrder(TreeNode* ptr);
void preOrder(TreeNode* ptr);

///////////// Mutators ////////////
void AddLeft(TreeNode* p,char c);
void AddRight(TreeNode* p,char c);
```

Tree Class


TreeType
~TreeType
IsEmpty
InsertItem
RetrieveItem
PrintTree
Private data:
root
'J'
'E'
'S'
'A'
'H'

Example

- Represent using array and linked list