Tree 1

Tree

1. Tree Definitions
2. Binary Tree
   - Traversals
   - Binary Search Tree
   - Representations
   - Application : Expression Tree

3. AVL Tree
4. Which Representions ?
5. n-ary Tree
6. Generic Tree
7. Multiway Search Tree
8. B-Trees

Tree Definitions

A **tree** อาจ

1. empty ไม่มี **nodes** เรียก **null tree** / **empty tree**

หรือ

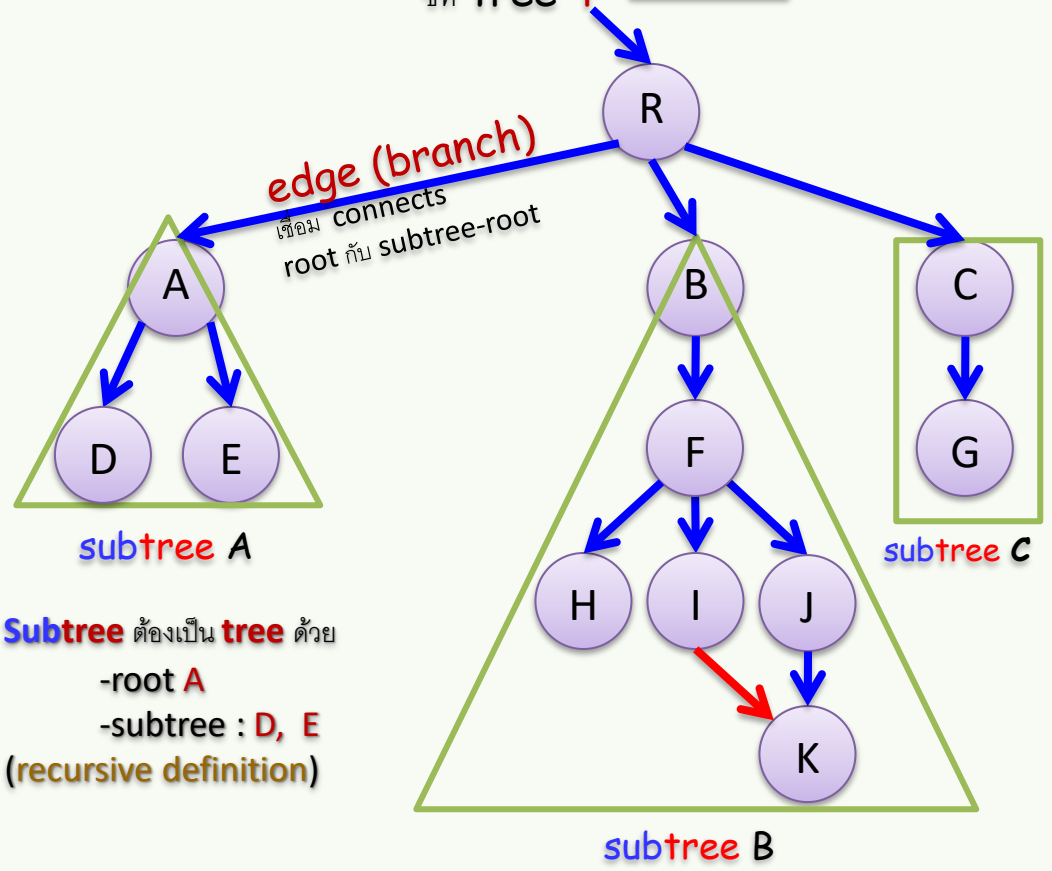
2. ประกอบด้วย
   1. **root node**
   2. **>=0 subtrees**

Node ใน tree ต้อง **disjoint** กัน

ต้องไม่มี node ร่วมกัน ใน root หรือ ใน subtrees

ชื่อของ tree นิยมเรียกตาม **root**

tree R



Subtree ต้องเป็น **tree** ด้วย

-root A

-subtree : D, E

(recursive definition)

K ไม่ **disjoint** (อยู่ในทั้ง subtree I & subtree J) → R ไม่ใช่ **tree**

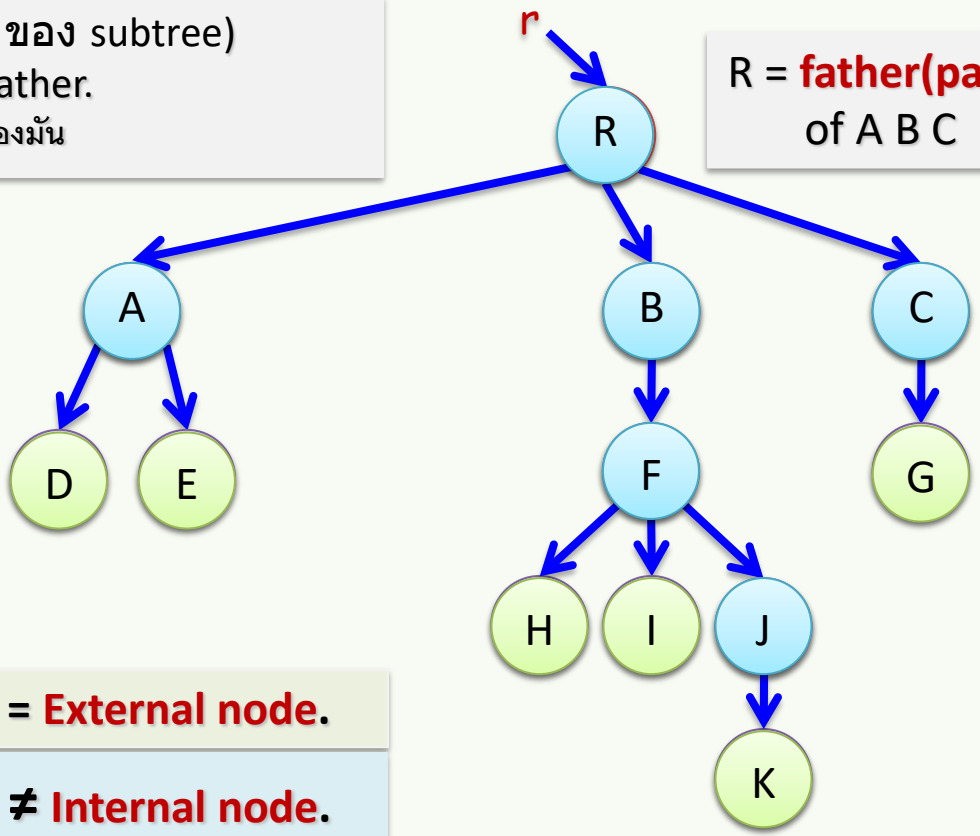
ใน tree : branches ไม่เชื่อมเป็นวง

Tree Definitions

Root = **father (parent)** of subtree's root.
= **father** ของ root ของ subtree

Subtree's root (root ของ subtree)
= **son (child)** of his father.
= **son (child)** ของพ่อของมัน

Root ไม่มีพ่อ



R = **father(parent)** of A B C

A B C = **sons(childs)** of R

Father of **F** ?
Sons of **A** ?
Father of **R** ?
Sons of **H** ?

Leaf (leave) = External node.

Leaf (leave) ≠ Internal node.

Leaf (leave)
= node with no son.
= node ที่ไม่มีลูก

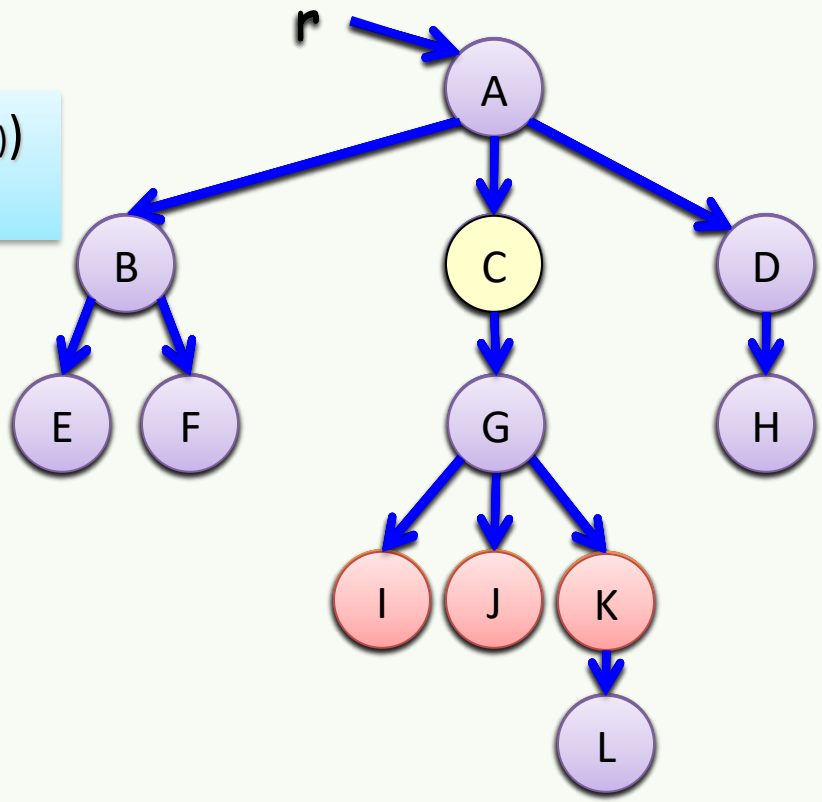
Leaf node (external, outer, terminal)
node with no son

Branch node (internal, inner, inode (for short))
node ≠ leaf

Siblings (brothers)
node with same father

Grand Parent
father of father

Grand Child
son of son



Path, Path Length, Depth, Height

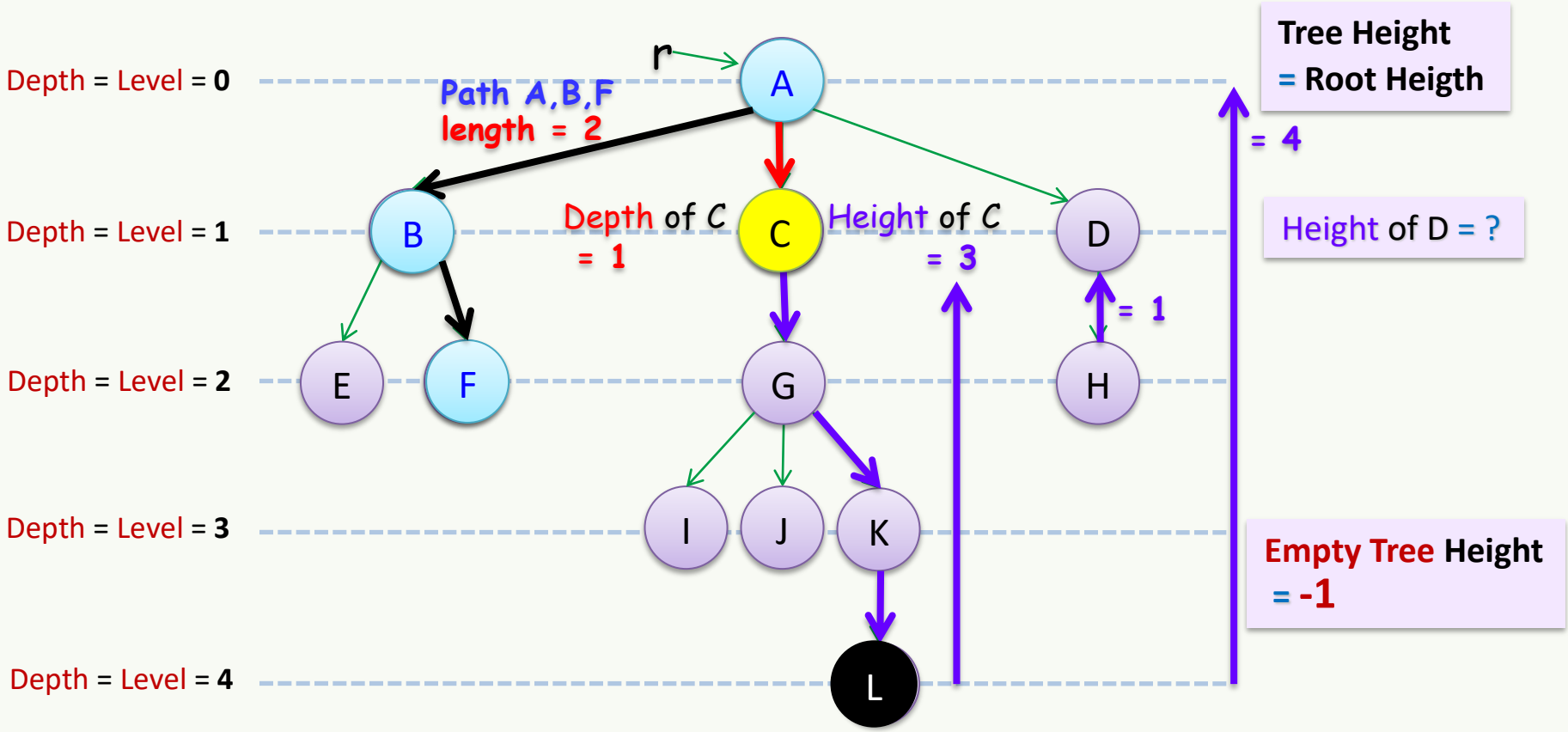
Path (from n to d) sequence of nodes and edges connecting a node n with a descendant d

In tree, only 1 path from node to node

Depth (level) of node = path length from root to node

Height of node = longest path length from node to leaf

Path length = # edge in path



Ancestor & Descendant

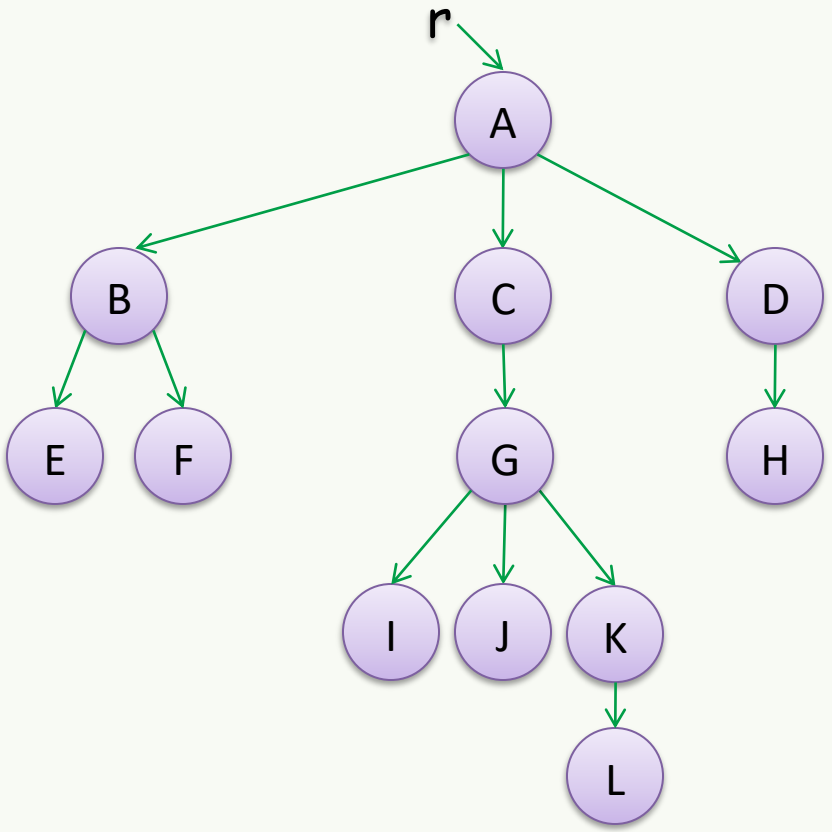
Ancestor บรรพบุรุษ
father of ancestor

A = ancestor of D
if has path from A to D

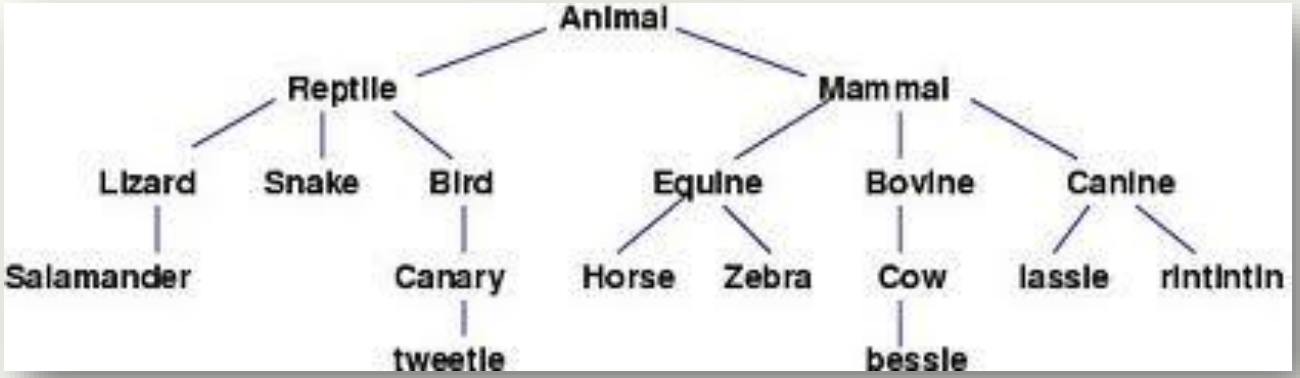
Descendant ลูกหลาน
son of descendant

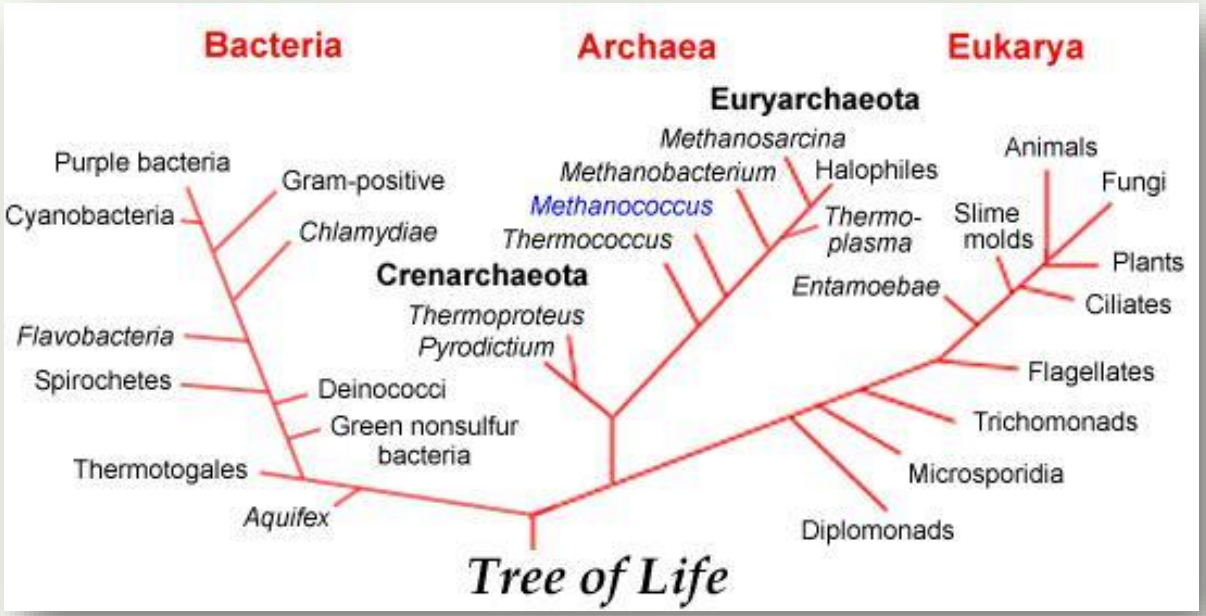
A = Proper Ancestor of D
if A ≠ D

D = Proper Decendent of A
if D ≠ A

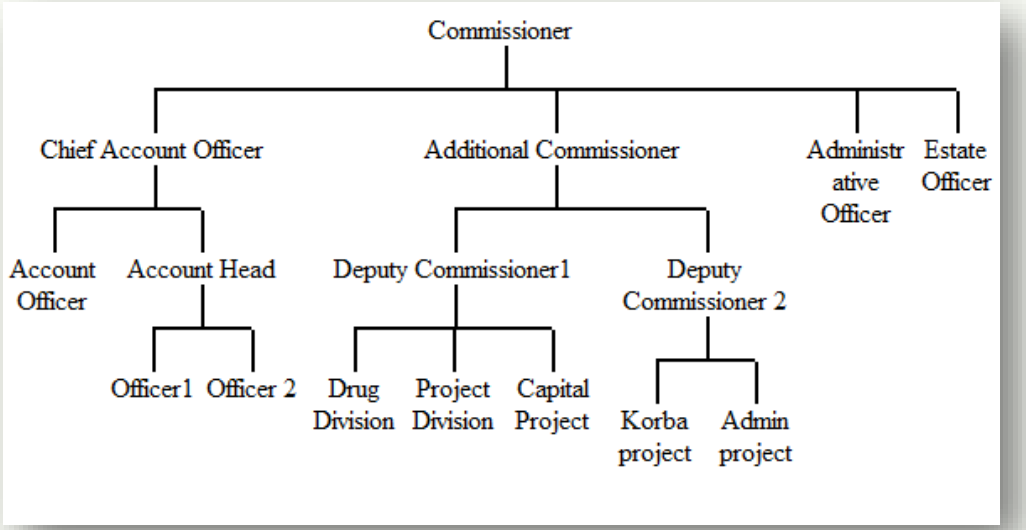


Tree Examples

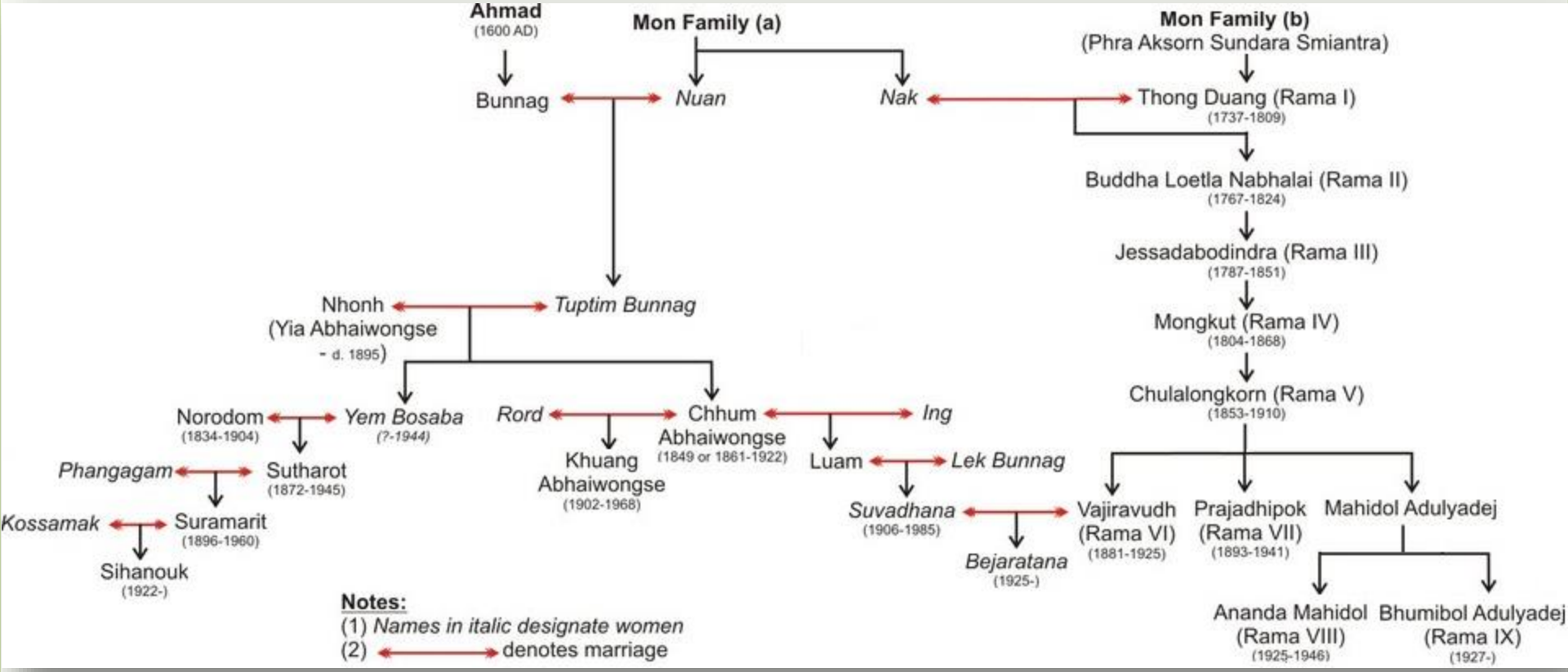




Tree Examples



Thai Royal Family Tree



Tree

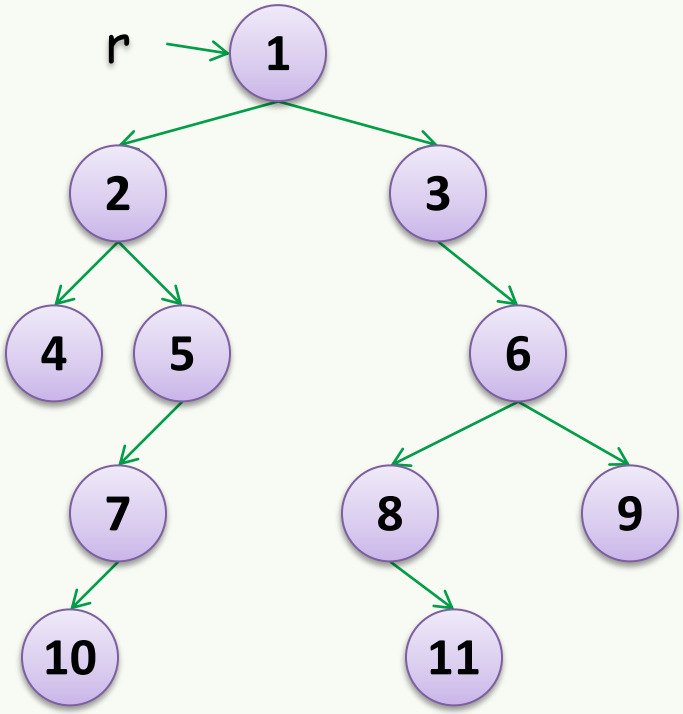
1. Tree Definitions
2. Binary Tree
   - Traversals
   - Binary Search Tree
   - Representations
   - Application : Expression Tree

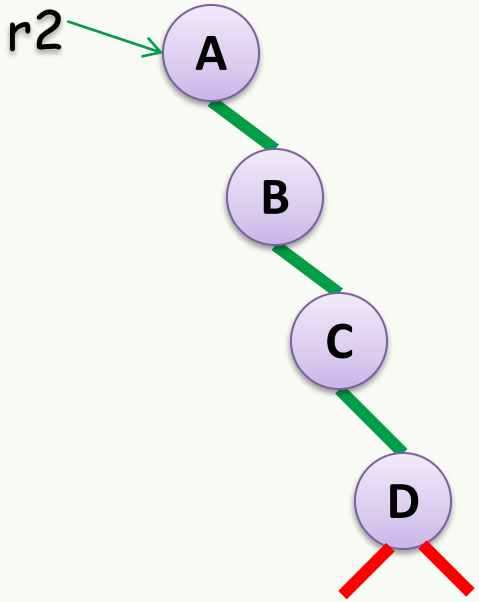
3. AVL Tree
4. Which Representions ?
5. n-ary Tree
6. Generic Tree
7. Multiway Search Tree
8. B-Trees

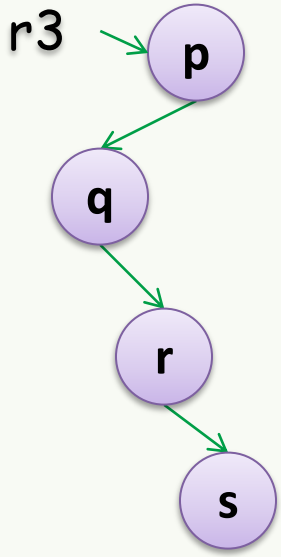
Binary Tree

bi = 2

Binary Tree : มีอย่างมากที่สุด 2 subtrees (0, 1 or 2 subtrees)



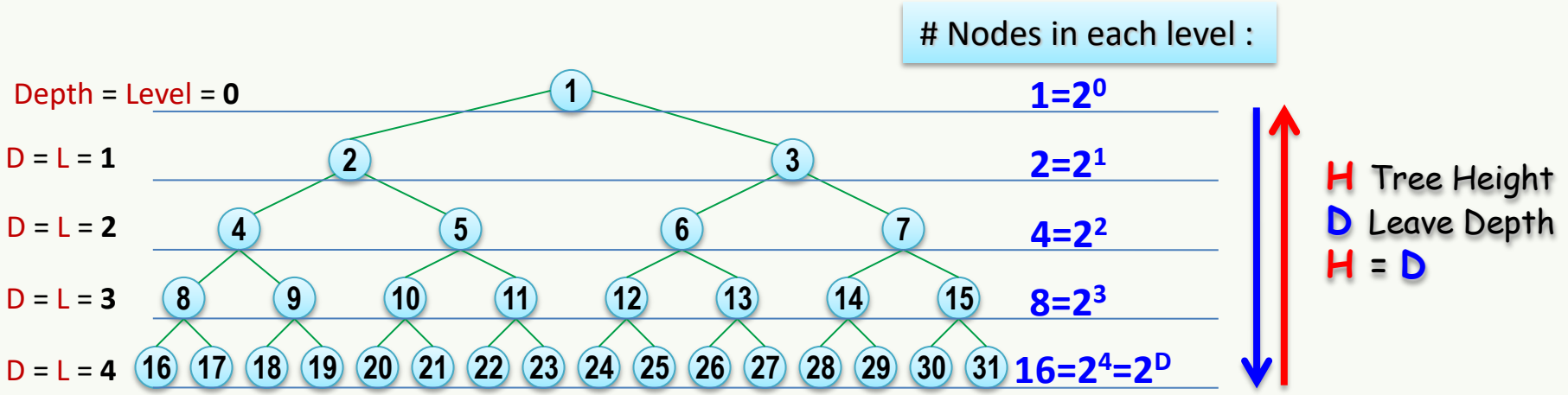




Normally not draw
- branch direction
- null ptr

Perfect Binary Tree

Perfect Binary Tree (Ambiguously also called Complete Binary Tree) Every level is completely filled



N = จำนวน node ใน perfect binary tree

$N = 2^0+2^1+...+2^H$ $\sum_{i=0}^{N} r^i = \frac{1-r^{n+1}}{1-r}, r <> 1$

$N = 2^{H+1} - 1$

$H = D = \log_2 (N+1) - 1$

$O(\log_2(N))$

$N = (2^H \text{external}) + (2^H\text{-}1 \text{ internal})$

Perfect Binary Tree of N nodes height H :

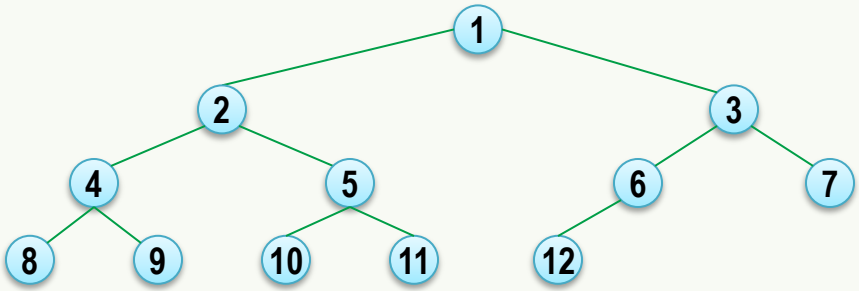
$H = \log_2 (N+1) - 1$

$N = 2^{H+1} - 1$

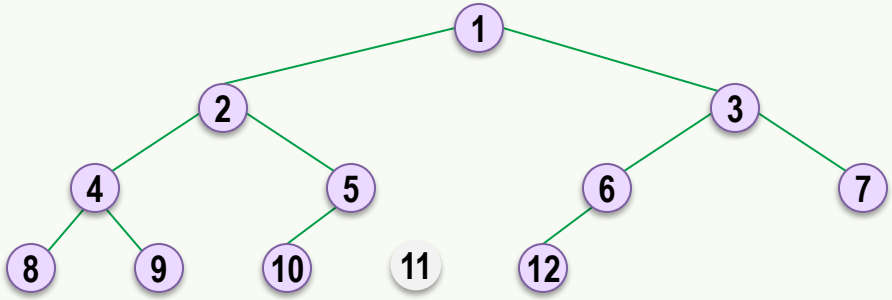
Complete BinaryTree

Complete Binary Tree

Every level, except possibly the last, is completely filled &
All nodes are as far left as possible.



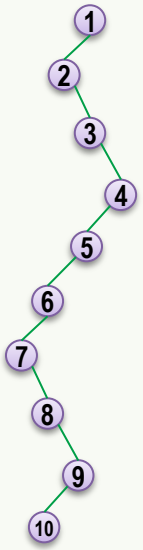
Complete binary tree

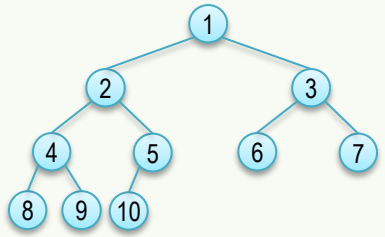


Without node 11 :
Not a complete binary tree

Complete Binary Tree : H and N

How can 10 nodes in a binary tree has
- longest search path ?
- shortest search path ?





Worst Case Height

Like Linked list : **H** = N – 1

O(N)

Best Case Height

Complete Binary Tree of **N** nodes height **H** :

$H = D = \lceil \log_2 (N+1) - 1 \rceil = \lfloor \log_2 N \rfloor$

O(log N)

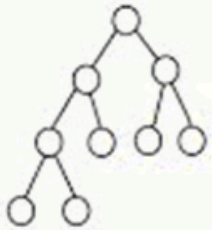
How much different N vs $\log_2 N$?

$\text{Log}_2 1{,}000{,}000 < 20$

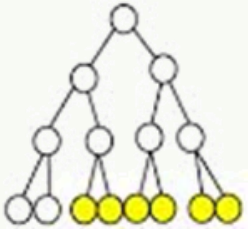
So
Complete Binary Tree
is an ideal tree

Full Complete Perfect

full

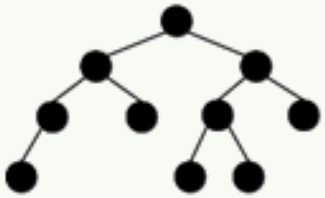


complete

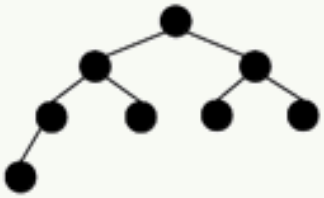


perfect

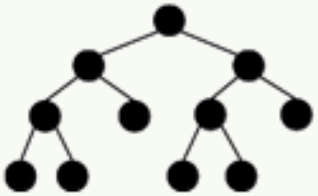
Neither complete nor full



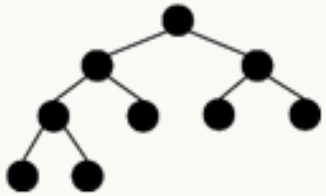
Complete but not full



Full but not complete



Complete and full

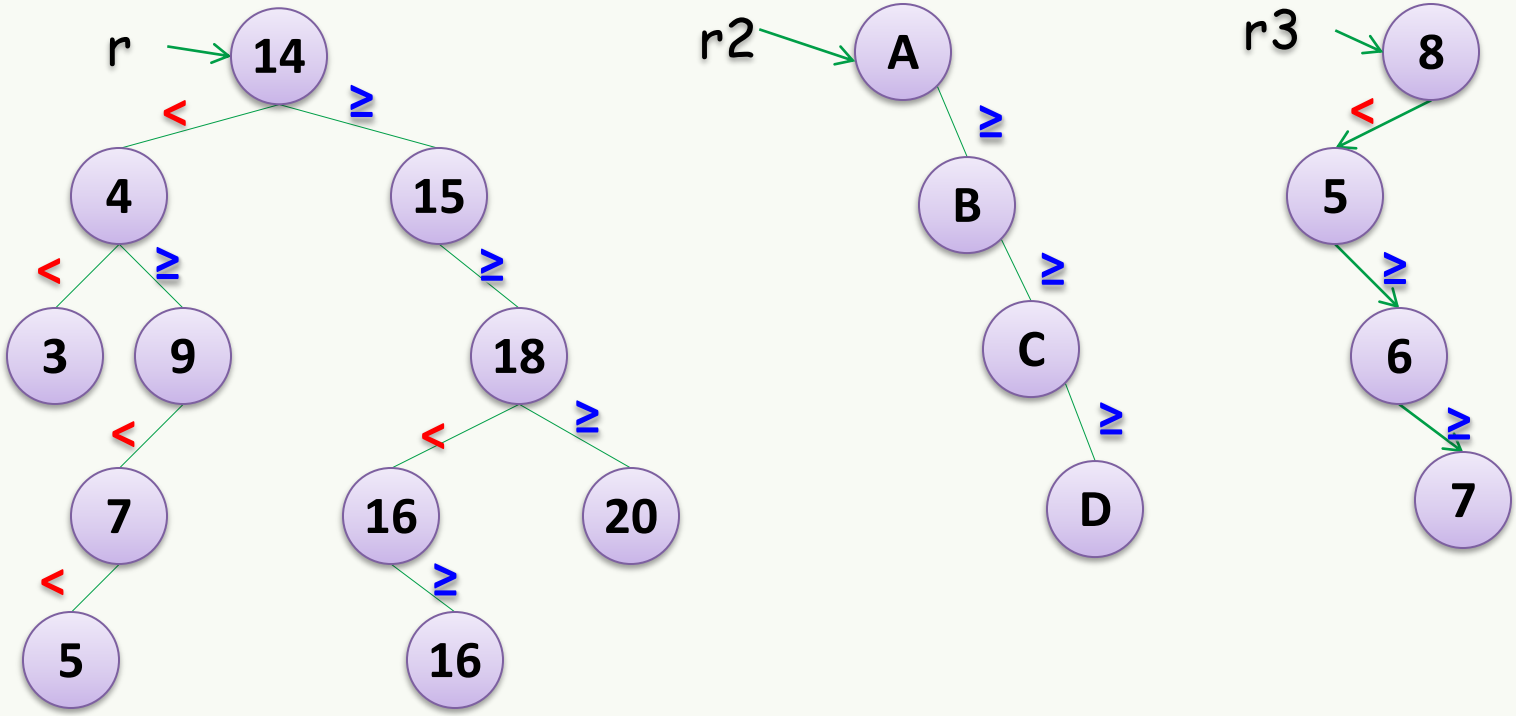


Binary Search Tree

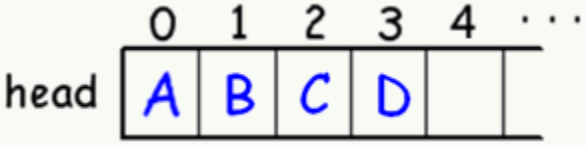
Binary Search Tree : for every node A

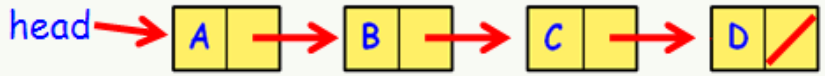
left descendants < A

right descendants ≥ A



Why Tree ?





Implicit (Sequential) Array

Insertion – Deletion Problems

Linear Linked list

Linear Search

O(n)

Tree

Tree Search

$O(\log_2 n)$

Tree

1. Tree Definitions
2. Binary Tree
   - Traversals
   - Binary Search Tree
   - Representations
   - Application : Expression Tree

3. AVL Tree
4. Which Representions ?
5. n-ary Tree
6. Generic Tree
7. Multiway Search Tree
8. B-Trees

Tree Traversal

การไปเยี่ยม (visit เช่นการพิมพ์ การ update ข้อมูล) ทุก node node ละ 1 ครั้ง อย่างมีระบบ

แบ่งตามลำดับของการ visit

Breadth First (Level Order) จาก root ไปด้านข้างก่อน

1. ลูกคนแรก ลูกคนที่ 2 ลูกคนที่ 3 ...จนหมดลูกทุกคน
2. ทำข้อ 1. กับทุกคนทีไปเยี่ยมมาตามลำดับ

Depth-First Order จาก root ไปด้านลึกก่อนไปด้านข้าง

ลูกคนแรก แล้วไปหลานคนแรกก่อนไปที่ลูกคนที่ 2

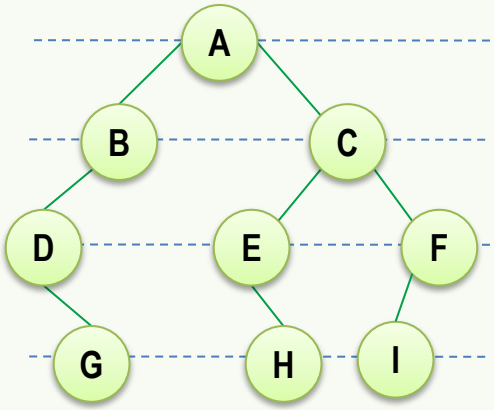
Breadth - First (Level Order)

Breadth First จาก root ไปด้านข้างก่อน ใช้ queue ช่วยในการหา

1. ลูกคนแรก ลูกคนที่ 2 ลูกคนที่ 3 ...จนหมดลูกทุกคน
2. ทำข้อ 1. กับทุกคนที่ไปเยี่ยมมาตามลำดับ

A B C D E F G H I

ไล่ไปทีละ level จึงเรียกอีกอย่างหนึ่งว่า **Level Order**



queue

deQ & visit

```
enQ ( root )
while ( notEmptyQ ) {
    n = deQ ( )
    visit(n)
    for each child c in n.children()
        enQ ( c )
}
```

Depth – First Order

Depth-First Order จาก root ไปด้านลึกก่อนไปด้านข้าง ใช้ stack ช่วยในการหา

ไปลูกคนแรก แล้วไปหลานคนแรกก่อนไปที่ลูกคนที่ 2

แบ่งเป็น 3 แบบ ขึ้นกับการวางตำแหน่งการ **visit root** ไว้ที่ใด

Inorder (Symmetric Order)

1. inOrder(leftSubtree)
2. **visit_root**
3. inOrder(rightSubtree)

Preorder

1. **visit_root**
2. preOrder(leftSubtree)
3. preOrder(rightSubtree)

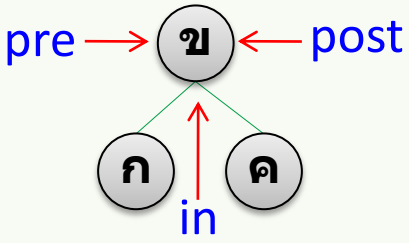
Postorder

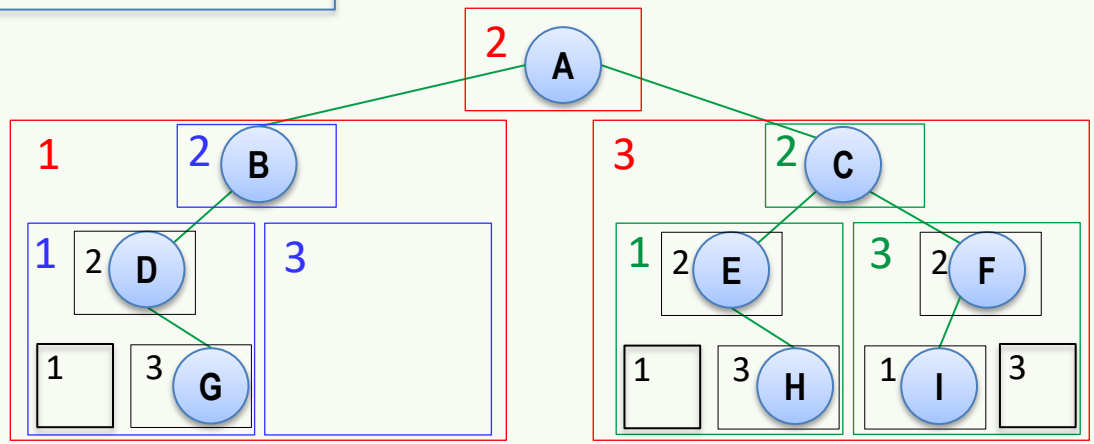
1. postOrder(leftSubtree)
2. postOrder(rightSubtree)
3. **visit_root**

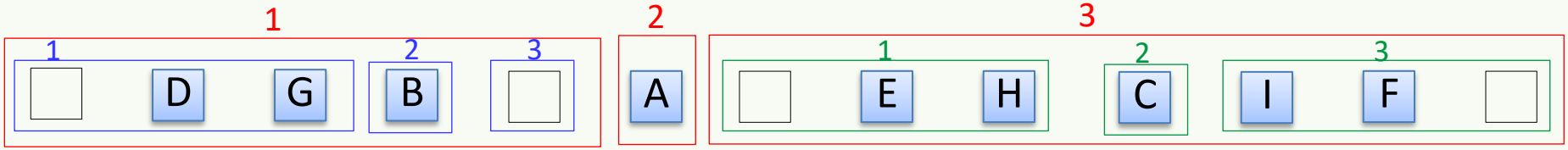
Inorder (Symmetric Order)

Algorithm:

```
inOrder( root ) {
    if (root != null){
        inOrder(left subtree)
        visit(root)
        inOrder(right subtree)
    }
}
```



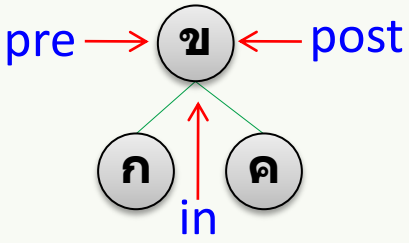


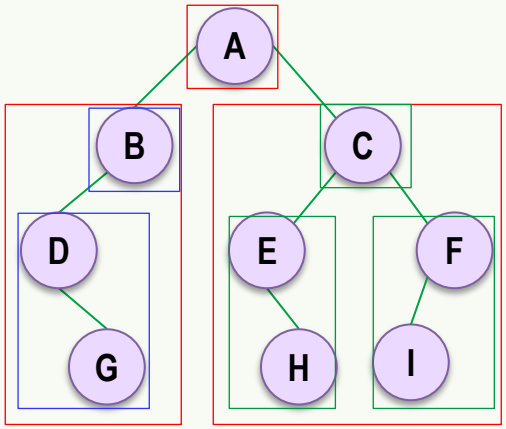


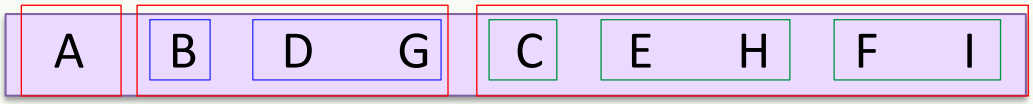
Preorder

Algorithm:

```
preOrder( root ) {
   if (root != null){
       visit(root)
       preOrder(left subtree)
       preOrder(right subtree)
    }
}
```



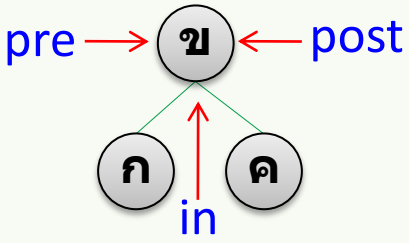


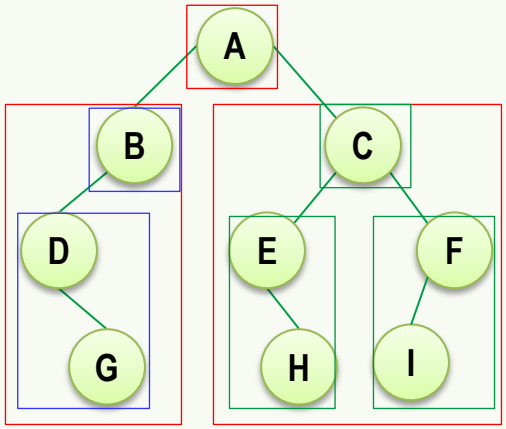


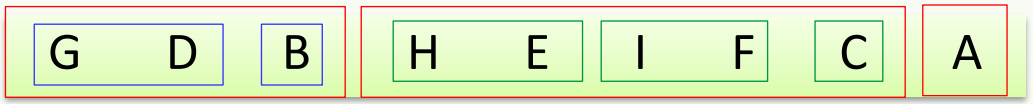
Postorder

Algorithm:

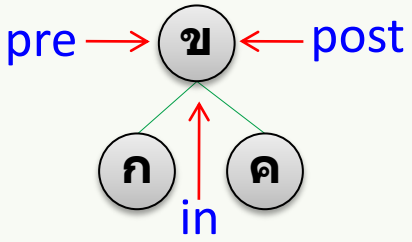
```
postOrder( root ) {
    if (root != null){
        postOrder(left subtree)
        postOrder(right subtree)
        visit(root)
    }
}
```

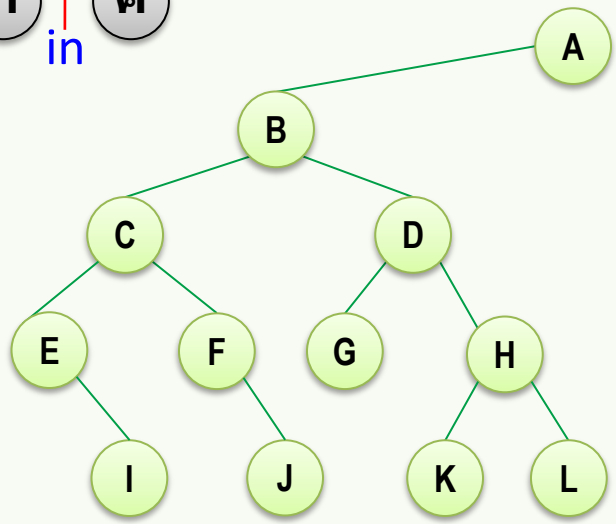




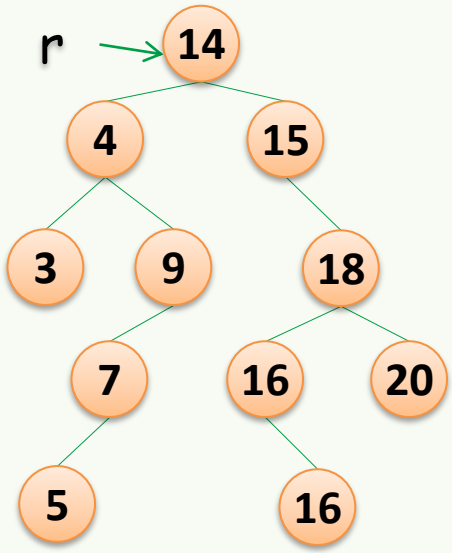


Test Your Self





Binary Search Tree



Inorder : LrootR
1. inOrder(L)
2. visit_root
3. inOrder(R)

Postorder : LRroot

Preorder : rootLR

Inorder	E I C F J B G D K H L A	3 4 5 7 9 14 15 16 16 18 20	Asscending Order !
Preorder	A B C E I F J D G H K L	14 4 3 9 7 5 15 18 16 16 20	
Postorder	I E J F C G K L H D B A	3 5 7 9 4 16 16 20 18 15 14	

Tree

1. Tree Definitions
2. Binary Tree
   - Traversals
   - Binary Search Tree
   - Representations
   - Application : Expression Tree

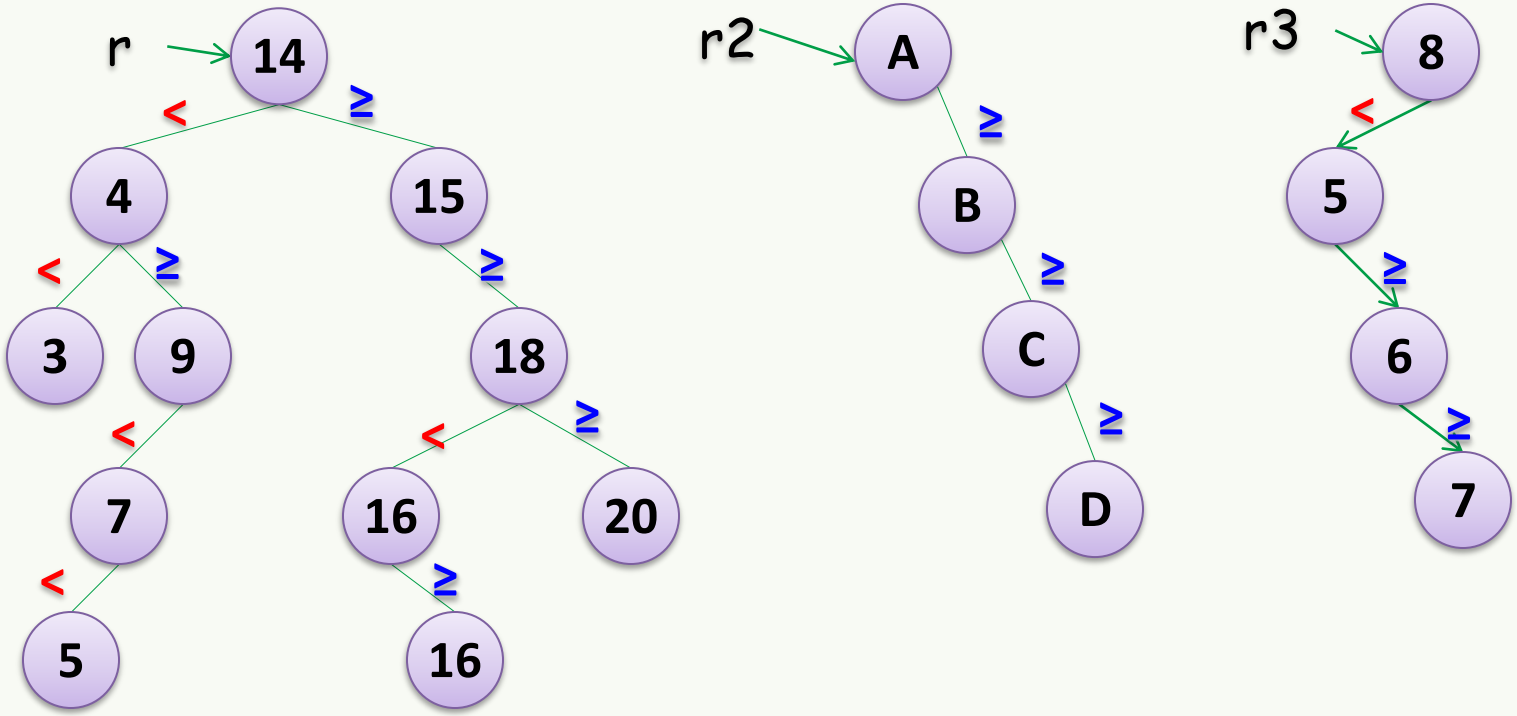
3. AVL Tree
4. Which Representions ?
5. n-ary Tree
6. Generic Tree
7. Multiway Search Tree
8. B-Trees

Binary SearchTree

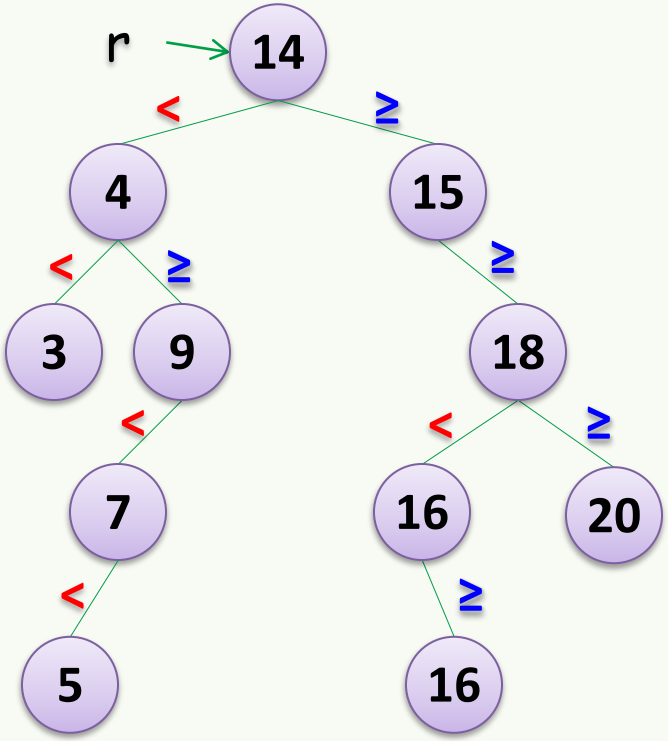
Binary Search Tree : for every node A

left descendants < A

right descendants ≥ A

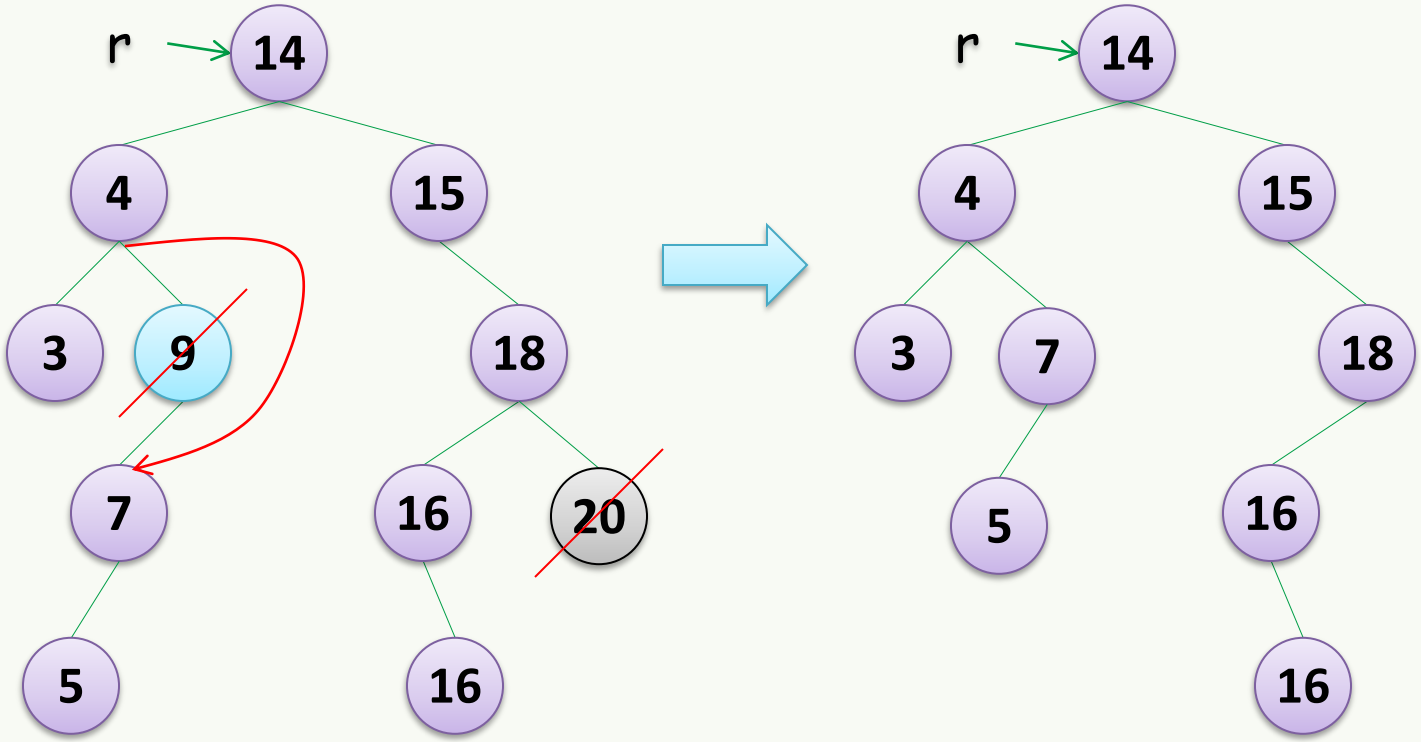


Insertion



Inserts : 14 4 9 7 15 3 18 16 20 5 16

Delete a : Leaf, Node with only 1 child



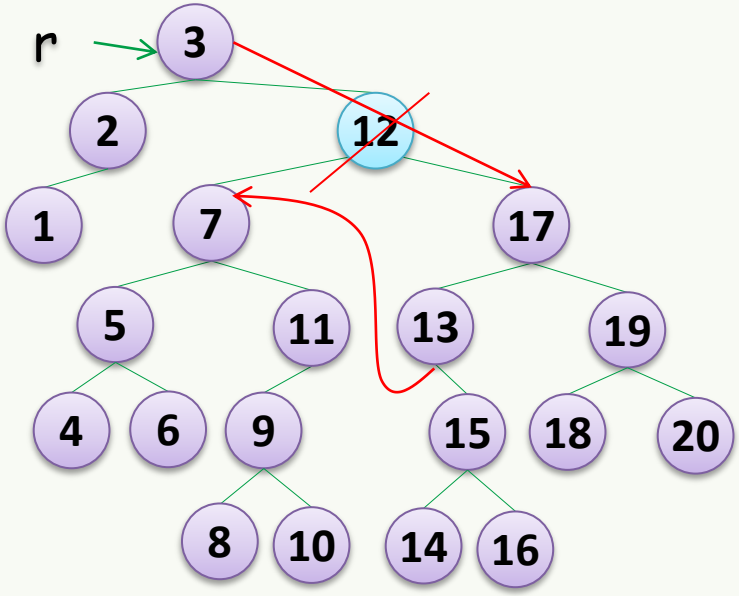
Delete Leaf

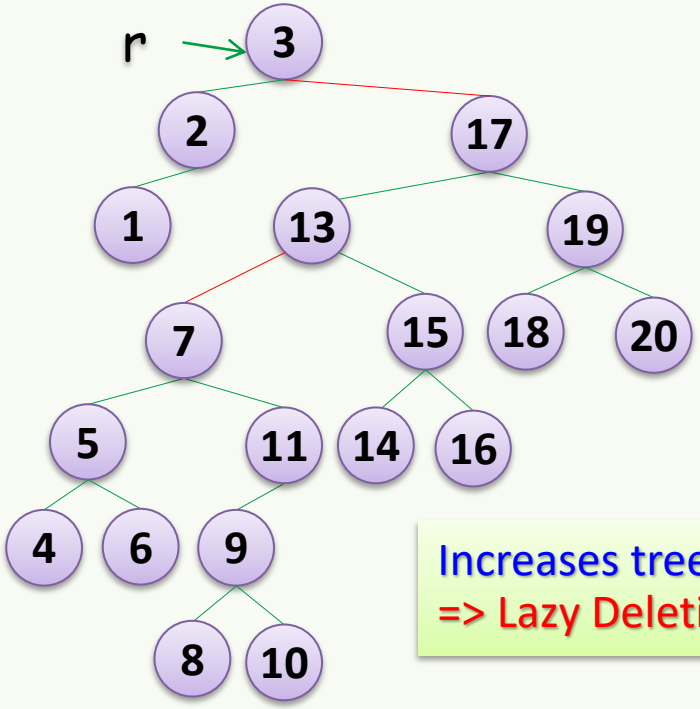
20 can be deleted right away, still be a binary search tree

Delete a node with only one child

Delete 9, replace a subtree at the deleted node

Delete a Node with Both Children : Lazy Deletion 1





Increases tree height.
=> Lazy Deletion

Delete 12

Where can we put tree 7, and tree 17 ?

Can replace only 1 at the deleted node.

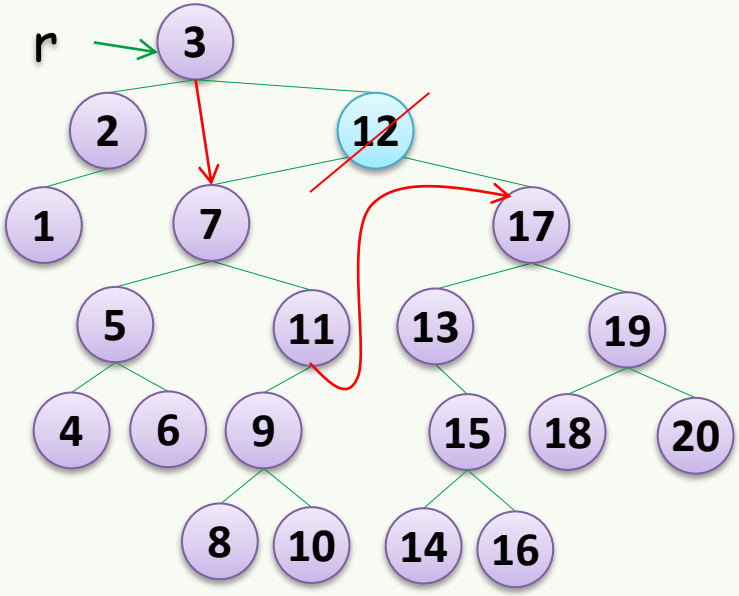
Let choose 17.

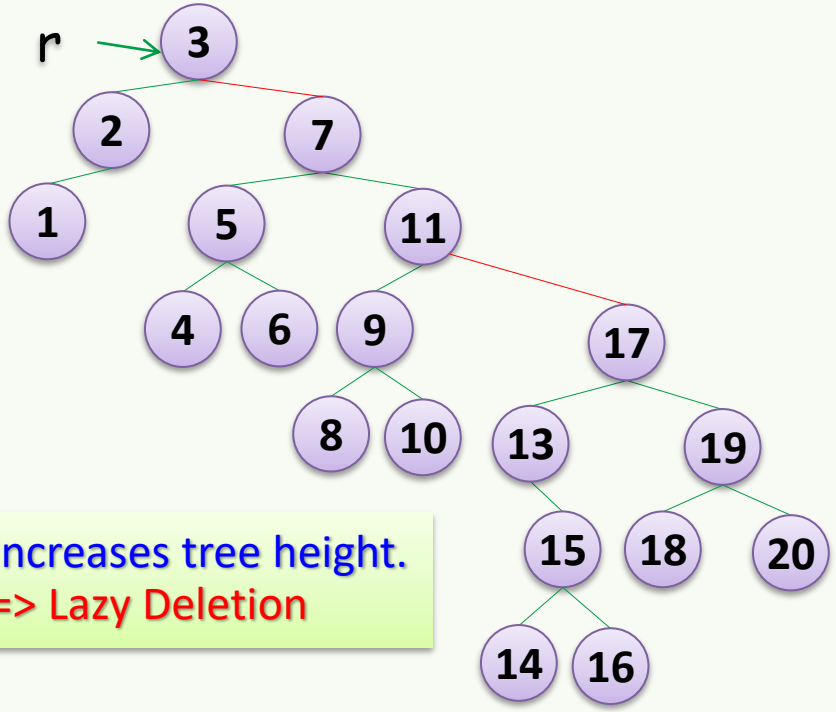
Where to put tree 7 ?

It's a binary search tree!

Where is it's place ?

Delete a Node with Both Children : Lazy Deletion 2





Increases tree height.
=> Lazy Deletion

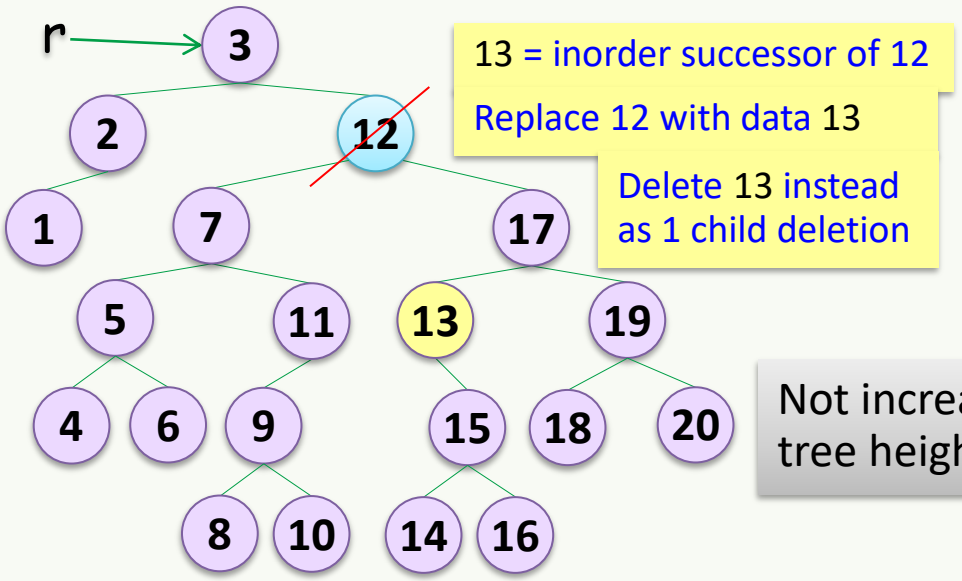
Delete 12

Can replace only 1 at the deleted node.

If we choose 7.

Where is 17's place ?

Deletion : Using Inorder Successor / Predecessor

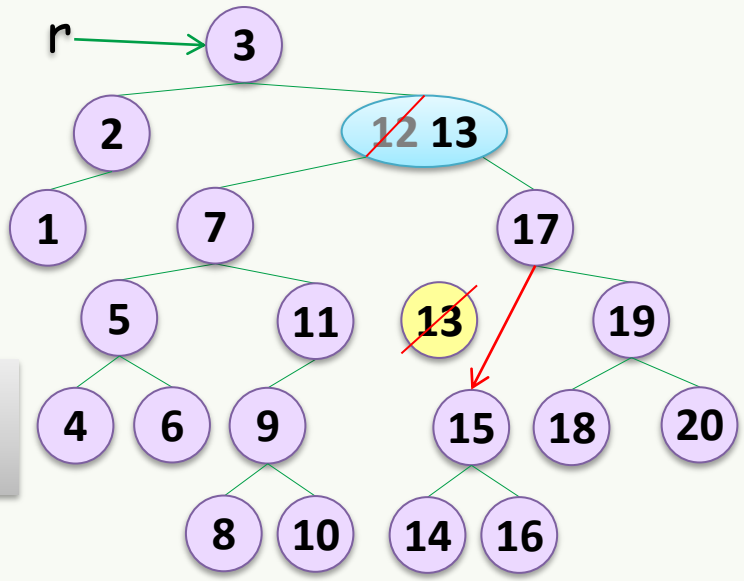


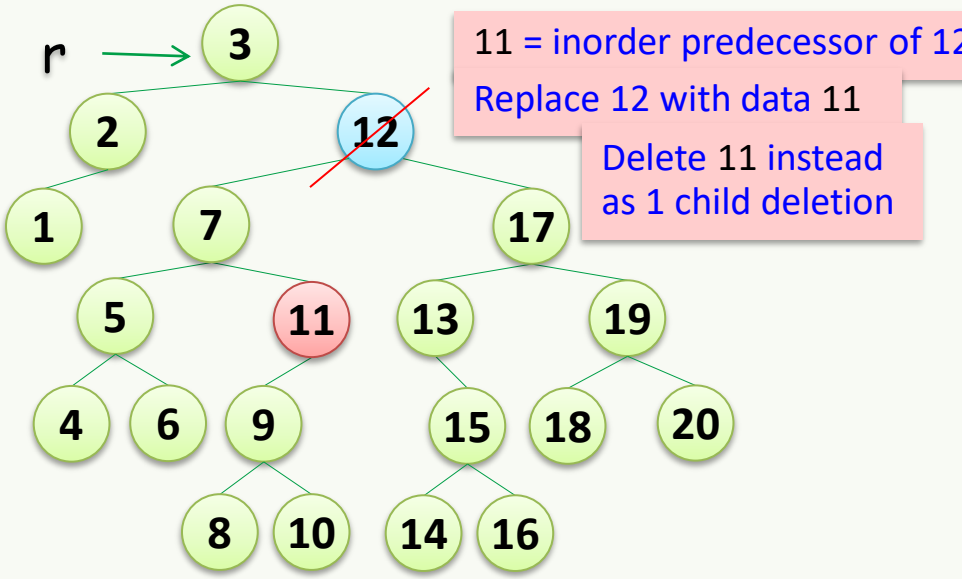
13 = inorder successor of 12

Replace 12 with data 13

Delete 13 instead as 1 child deletion

Not increasing tree height !!!

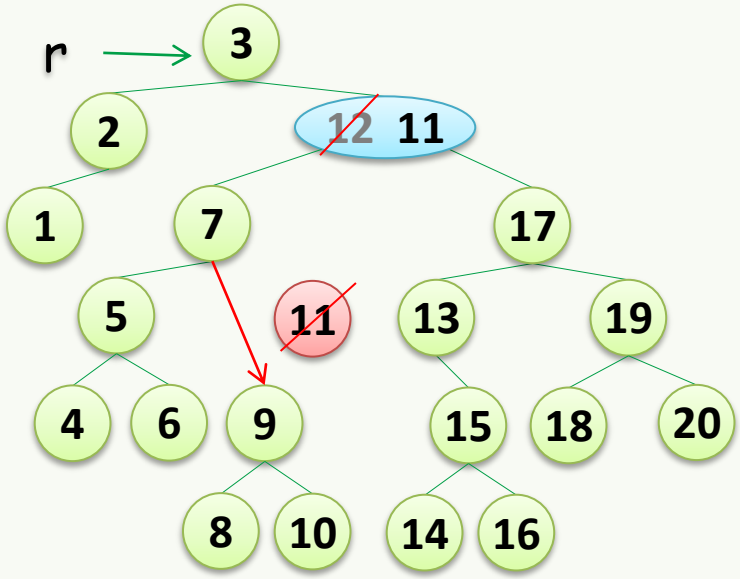




11 = inorder predecessor of 12

Replace 12 with data 11

Delete 11 instead as 1 child deletion



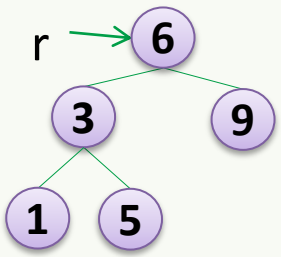
Tree

1. Tree Definitions
2. Binary Tree
   - Traversals
   - Binary Search Tree
   - Representations
   - Application : Expression Tree

3. AVL Tree
4. Which Representions ?
5. n-ary Tree
6. Generic Tree
7. Multiway Search Tree
8. B-Trees

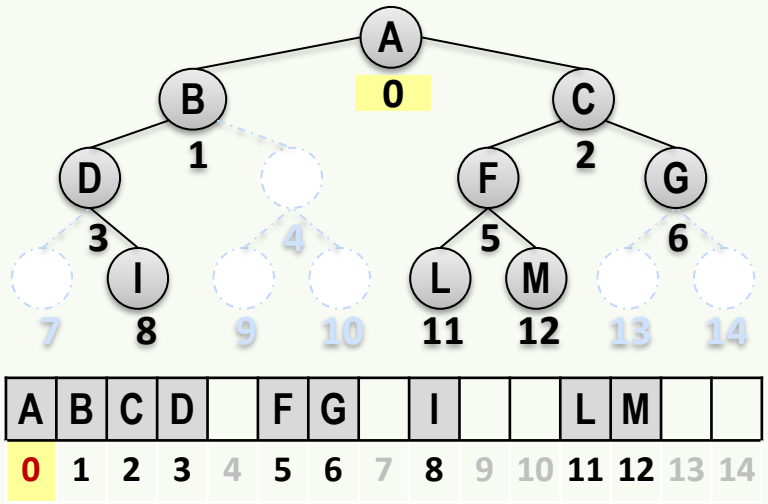
Binary Tree Representations

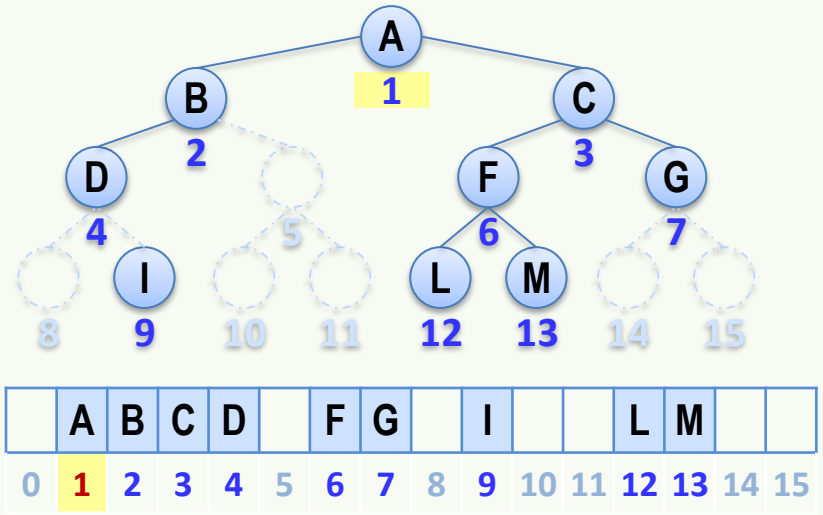
1. Dynamic



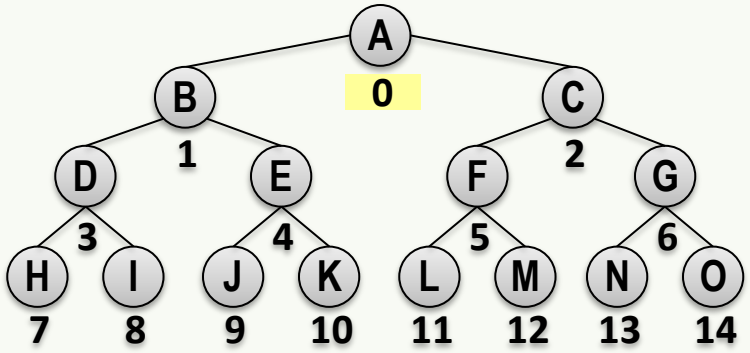

```
class node:

    def __init__(self,  data, left = None, right = None):
        self.data = data
        self.left  = None if left is None else left
        self.right  = None if right is None else right
```

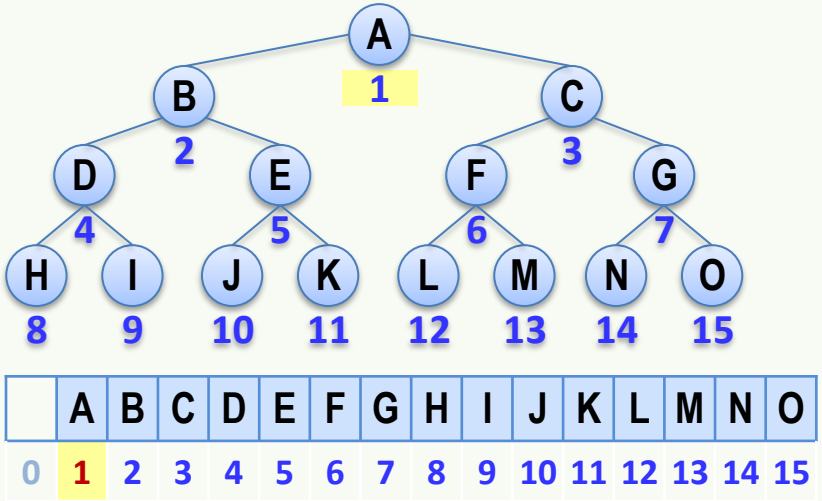
2. Sequential (Implicit) Array

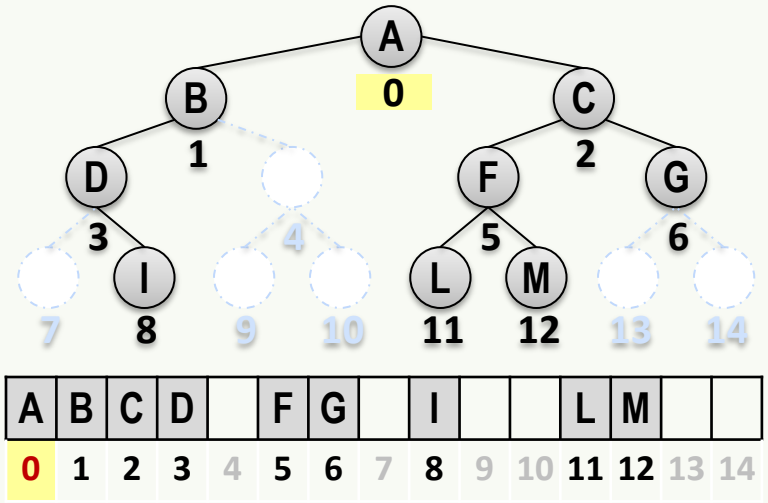


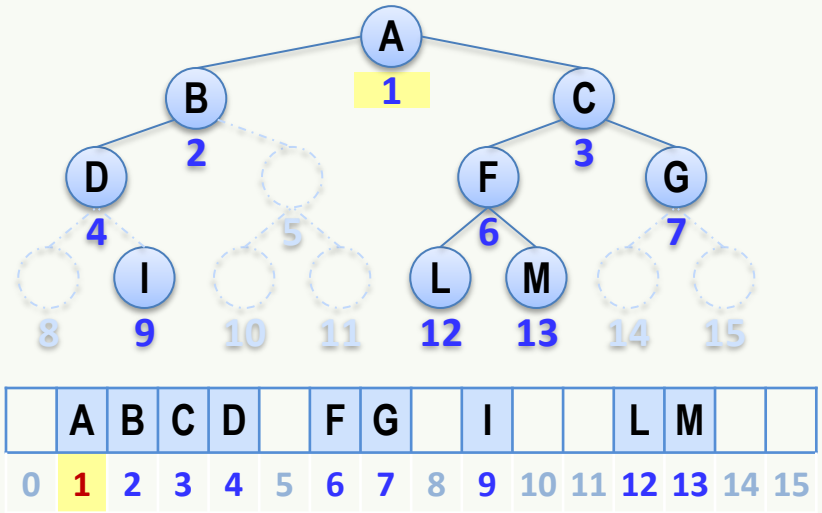


(Sequential) Implicit Array

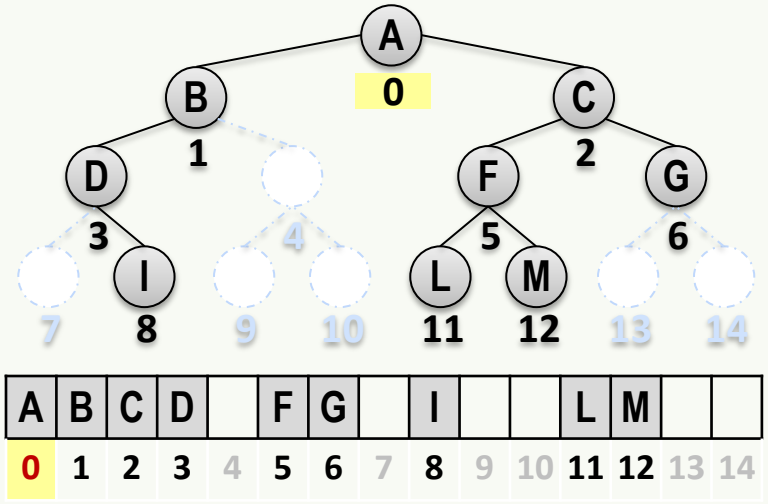


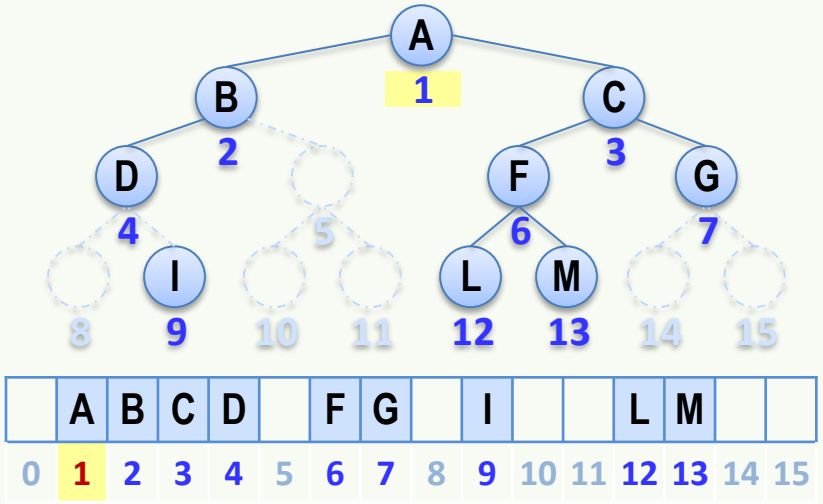






(Sequential) Implicit Array





	Start at index 0	Start at index 1
Where is the root ?	0	1
The node at index i 's left son ?	2 i + 1	2 i
right son ?	2 i + 2	2 i + 1
father ?	(i - 1) div 2	i div 2

So good ! No memory for link ! Easy to calculate !

What happen if we have only node at indices : 0, 2, 6, 14 ? ie. sparse

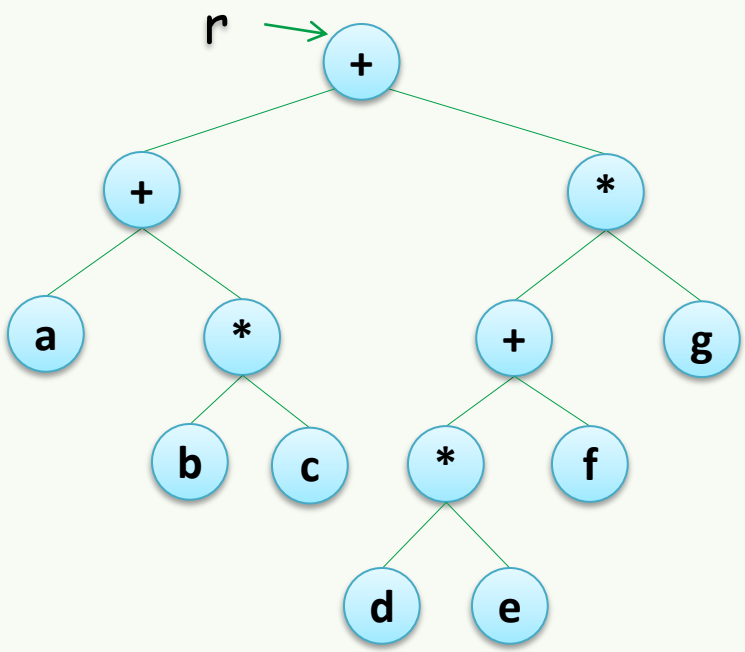
What shape of tree should be best for sequential array?

Tree

1. Tree Definitions
2. Binary Tree
   - Traversals
   - Binary Search Tree
   - Representations
   - Application : Expression Tree

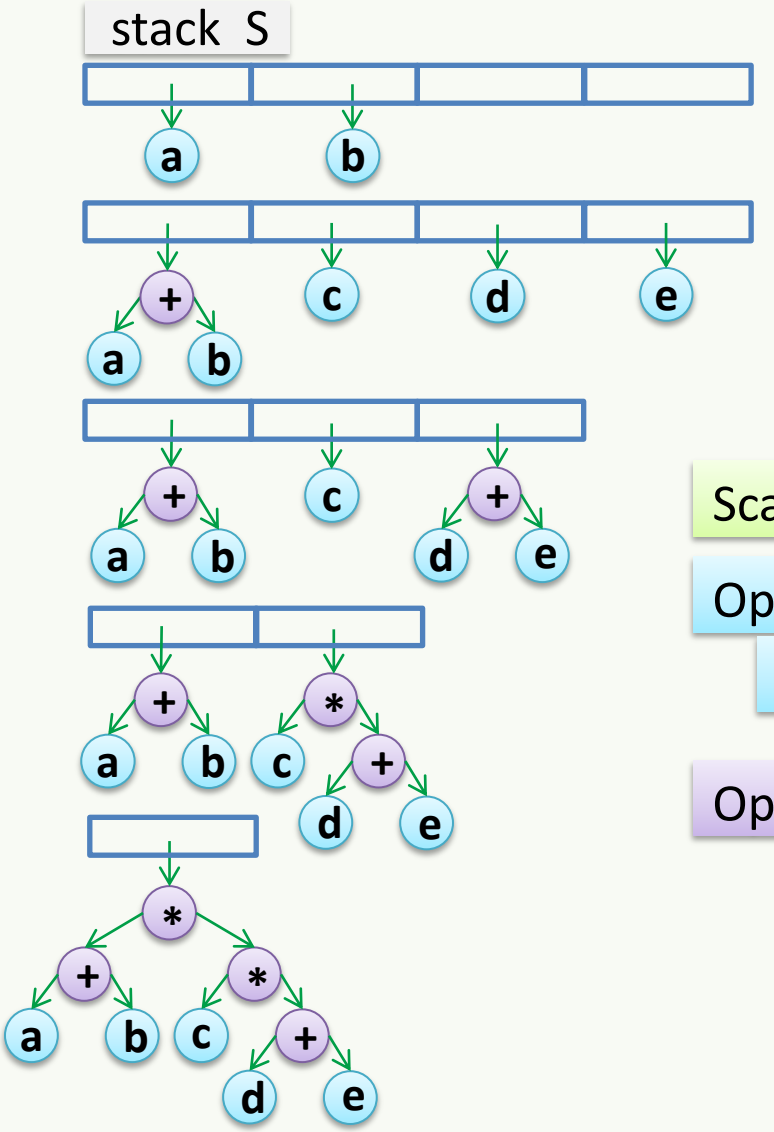
3. AVL Tree
4. Which Representions ?
5. n-ary Tree
6. Generic Tree
7. Multiway Search Tree
8. B-Trees

Example : Expression Tree



(a + b * c) + ((d * e + f) * g)

Constructing an Expression Tree



input postfix form

a b + c d e + * *

Scan input from left to right.

Operand

Create an operand node and push to the stack

Operator

Create an operator node

Pop 2 operands to be its children.

Push to the stack.

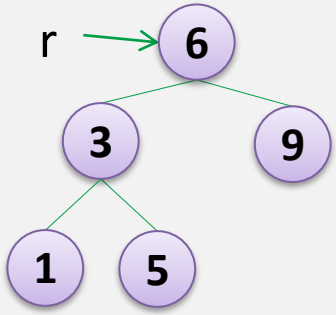
Node

```
class node:
    def __init__(self, data, left = None, right = None):
        self.data = data
        self.left = None if left is None else left
        self.right = None if right is None else right
```

```
    def __str__(self):
        return str(self.data)

    def getData(self):          # accessor
        return self.data

    def getLeft(self):          # accessor
        return self.left

    def getRight(self):         # accessor
        return self.right

    def setData(self, data):    # mutator
        self.data = data

    def setLeft(self, left):    # mutator
        self.left = left

    def setRight(self, right): # mutator
        self.right = right
```

Dynamic Representation 1

```
class BST:
  def __init__(self, root = None):
    self.root = None if root is None else root

  def addI(self, data):
    if self.root is None:
       self.root = node(data)
    else:
       # YOUR CODE

  def add(self, data):
    self.root = BST._add(self.root, data)

  def _add(root, data):
    if root is None:
       return node(data)
    else:
       if data < root.data:
         root.left = BST._add(root.left, data)
       else:
         root.right = BST._add(root.right, data)
    return root
```




```
if data < root.getData():
  root.setLeft(BST._add(root.getLeft(), data))
else:
  root.setRight(BST._add(root.getRight(), data))
```