# CMPT 125

# Introduction to Computing Science and Programming II

# November 8, 2021

# Assignment 4

- Assignment 4 is due to November 19, 23:59

  https://www.cs.sfu.ca/~ishinkar/teaching/fall21/cmpt125/assignments.html

- You need to submit one file to Canvas – *assignment4.c*
- Please make sure it compiles with the provided makefile

>> make

>> ./run_test4

Topics:

- Stacks – using the provided API, without relying on the implementation details
- Binary Trees

# Today

- Introduction to Graphs
- Introduction to Trees
- Binary Trees
- Traversing Trees

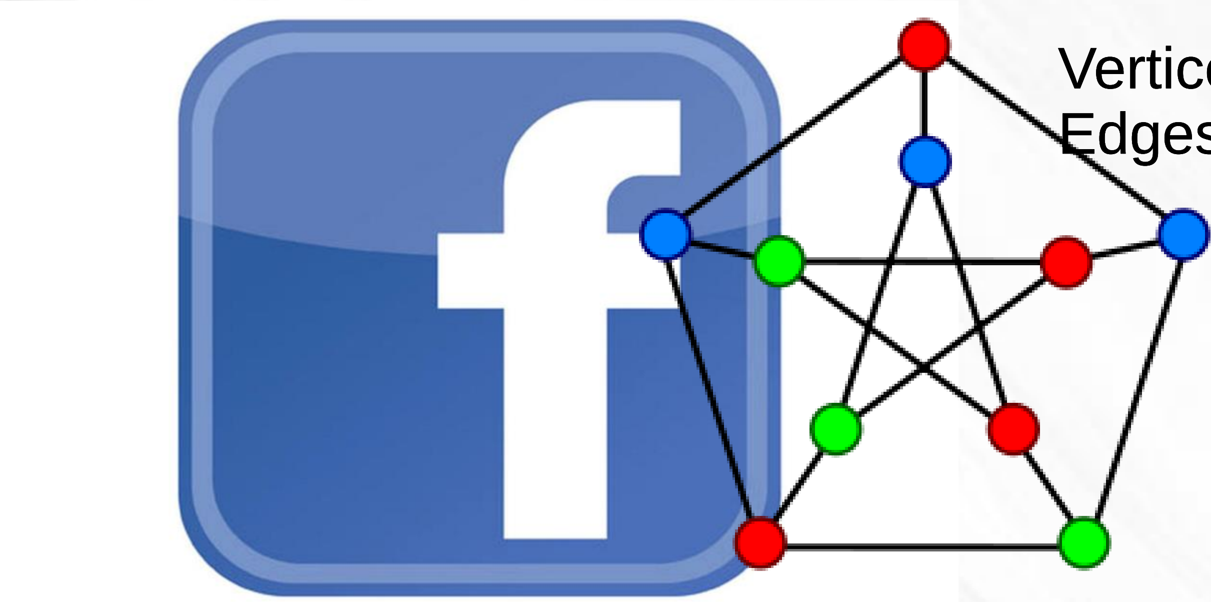
# Graphs

# Graphs

A *graph* describes relationships among items in a collection

- the items are the *vertices* (depicted by dots or junctions)
- the relations are the *edges* (depicted by lines between dots)

What are the vertices and edges in these common graphs?

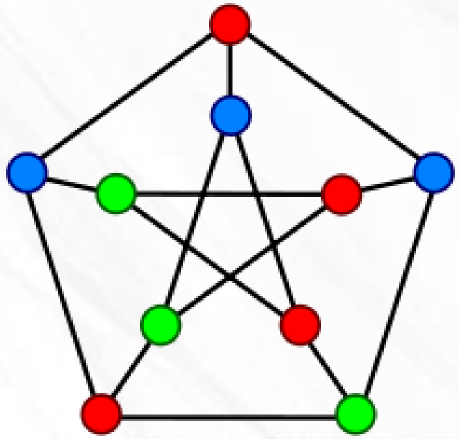


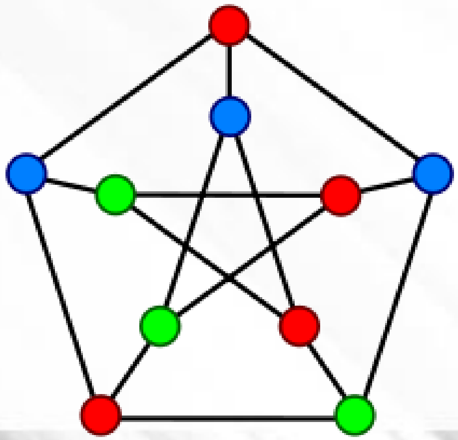
Vertices - people / intersections
Edges - friendships / roads

# Graphs

- A *path* from vertex *u* to vertex *v* is sequence of edges which connect a sequence of vertices between them.
- A graph is *connected* if each pair of vertices has a path.

- How do you visualize a disconnected graph?
- What if the Skytrain graph was disconnected



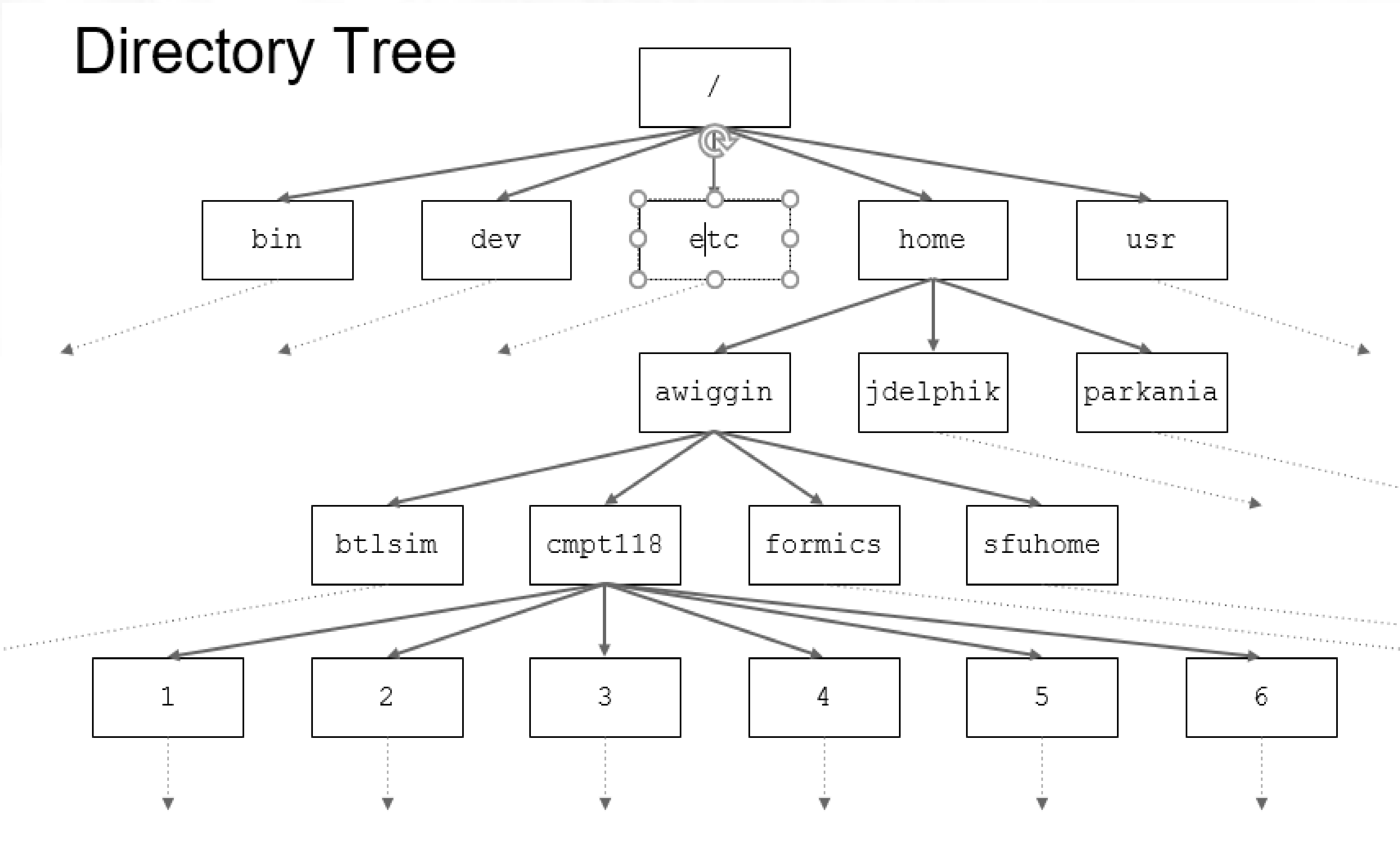


# Trees

# Trees

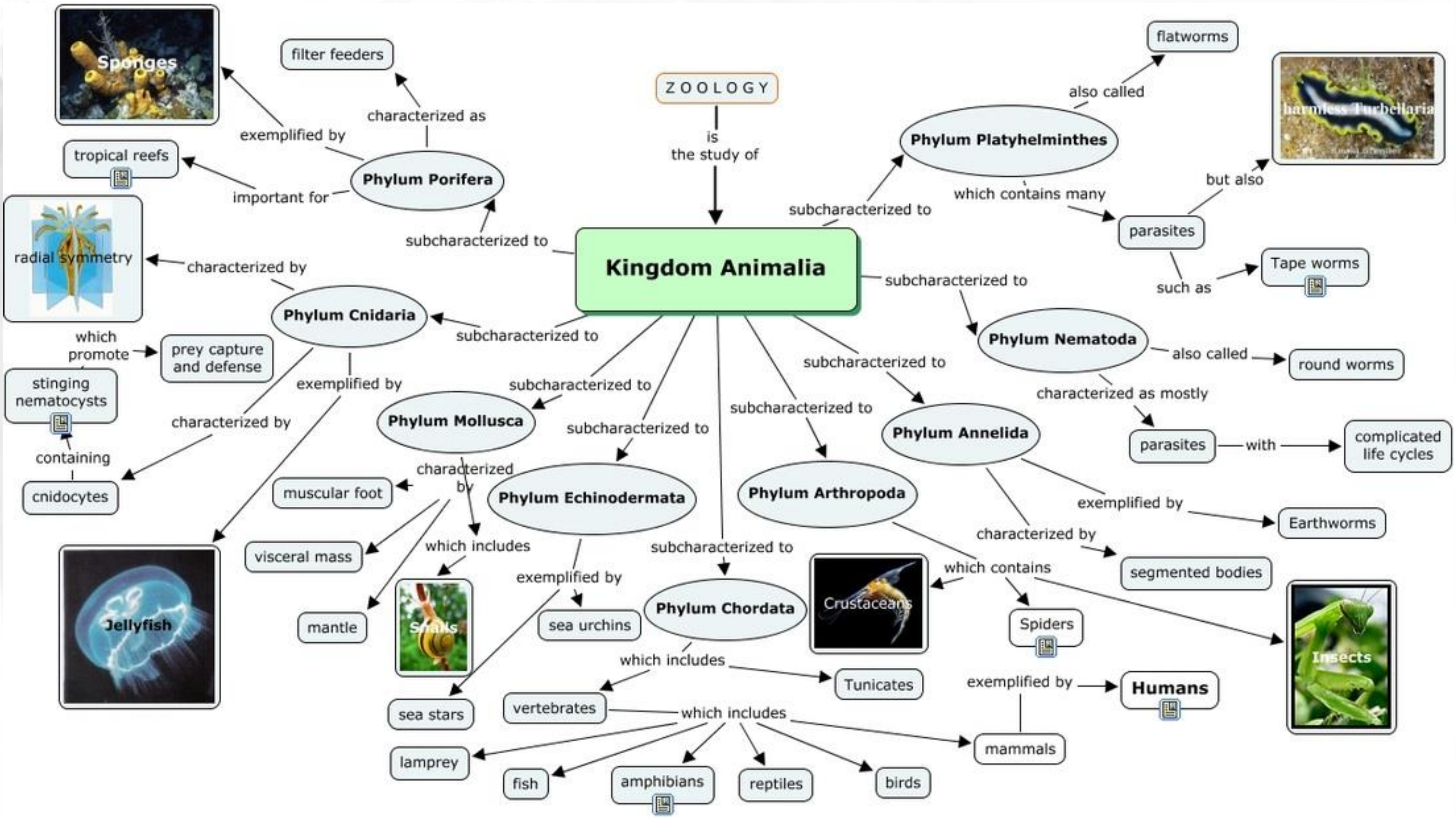
- A *tree* is a minimally connected graph
  - all vertices connected
  - no cycles

# Trees



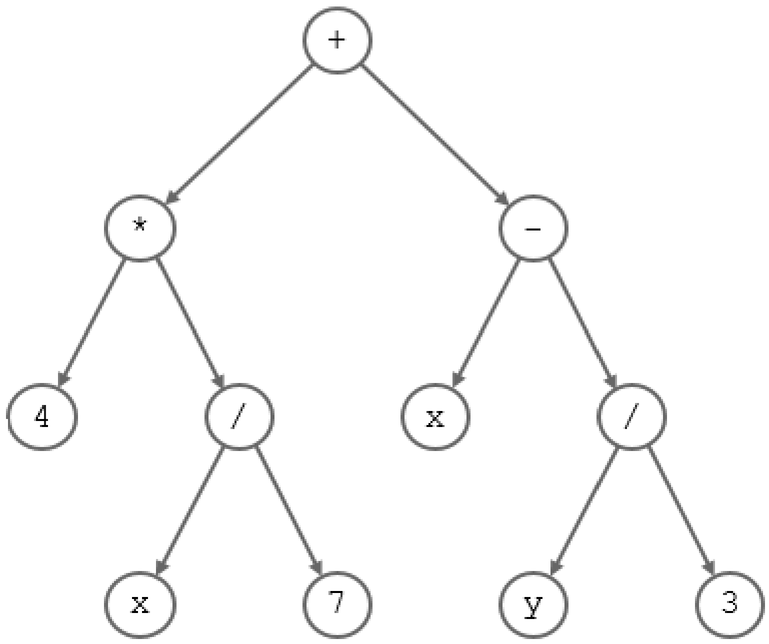
Directory Tree
/
bin
dev
etc
home
usr
awiggin
jdelphik
parkania
btlsim
cmpt118
formics
sfuhome
1
2
3
4
5
6

# Trees

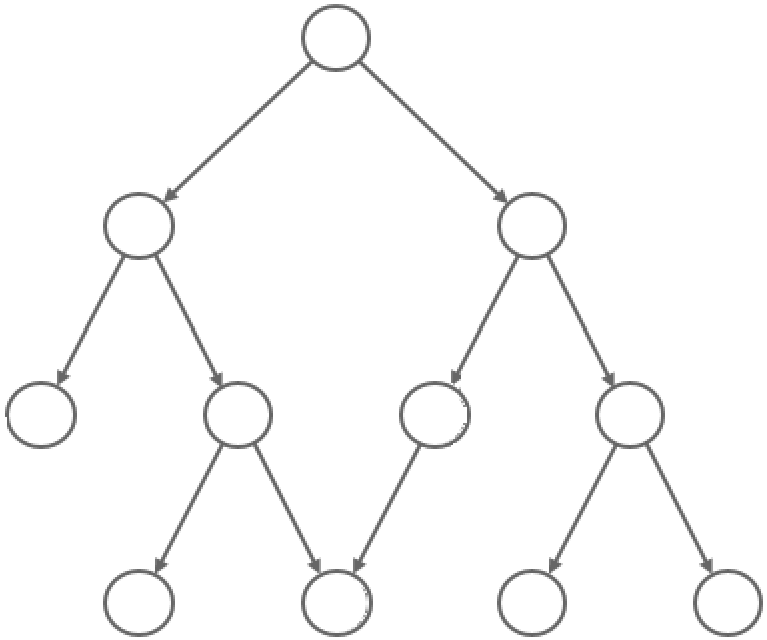


ZOOLOGY
is the study of
Kingdom Animalia
subcharacterized to
Phylum Porifera
characterized as
filter feeders
exemplified by
Sponges
important for
tropical reefs
Phylum Cnidaria
characterized by
radial symmetry
which promote
prey capture and defense
stinging nematocysts
containing
cnidocytes
exemplified by
Jellyfish
Phylum Mollusca
characterized by
muscular foot
visceral mass
mantle
which includes
Snails
Phylum Echinodermata
exemplified by
sea urchins
sea stars
Phylum Chordata
which includes
Tunicates
vertebrates
lamprey
fish
amphibians
reptiles
birds
mammals
exemplified by
Humans
Phylum Arthropoda
which contains
Crustaceans
Spiders
Insects
Phylum Annelida
characterized by
segmented bodies
Earthworms
Phylum Nematoda
also called
round worms
characterized as mostly
parasites
with
complicated life cycles
Phylum Platyhelminthes
flatworms
which contains many
but also
harmless Turbellaria
such as
Tape worms

# A Tree

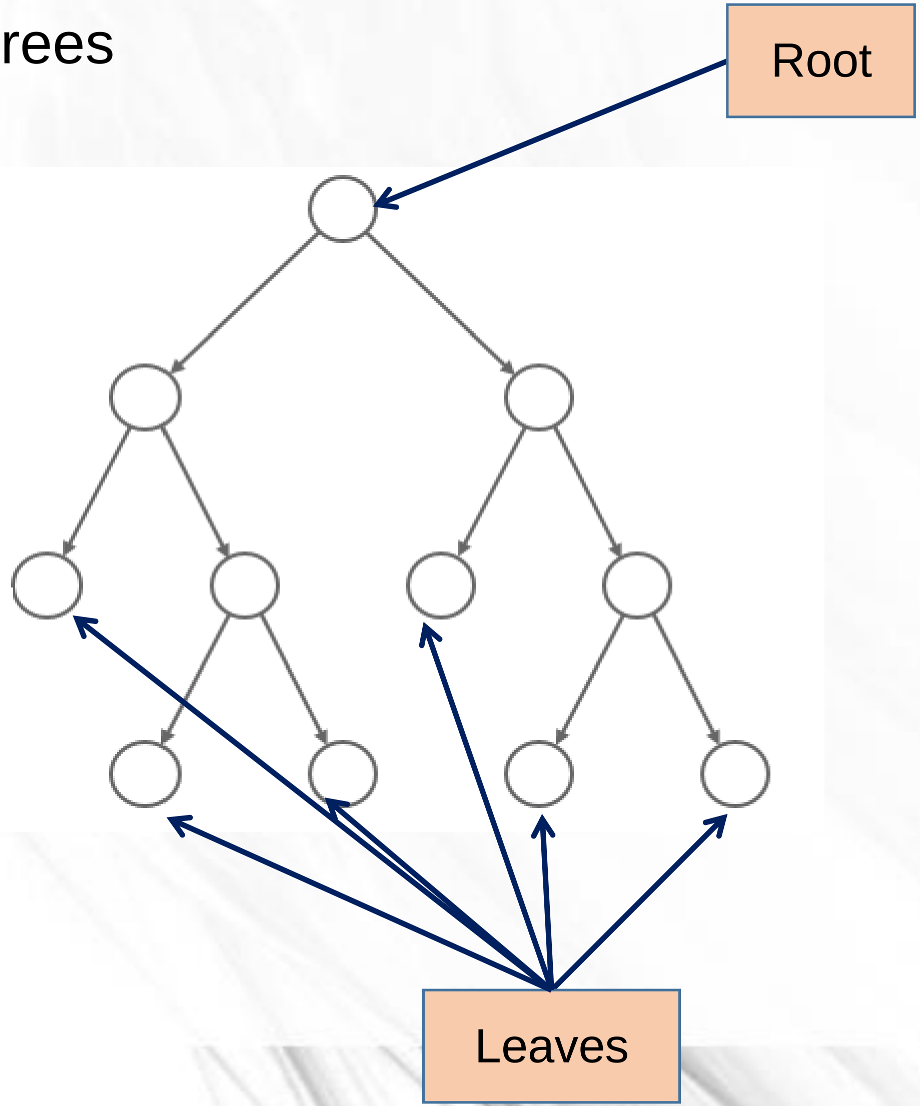


# Not a Tree



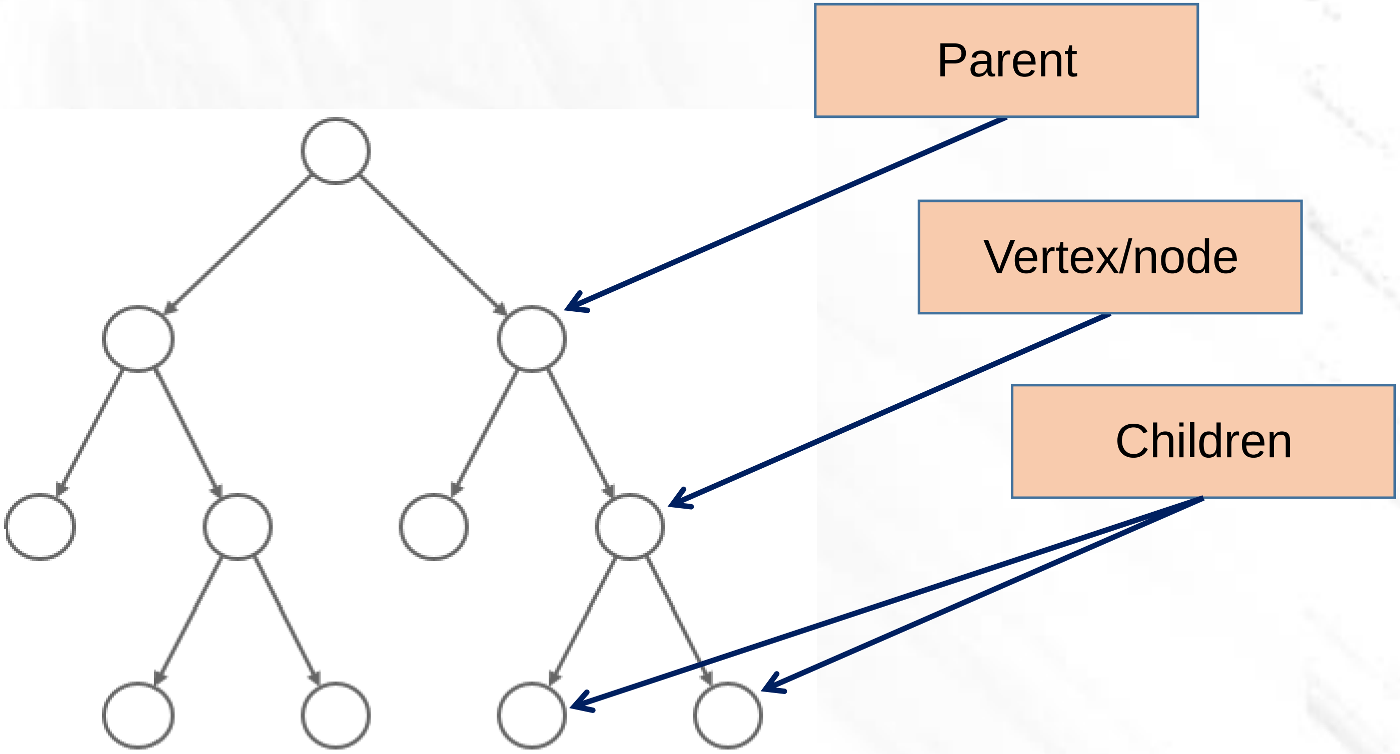
# Rooted Trees

Structure of rooted trees



# Rooted Trees

Structure of rooted trees



# Rooted Trees

For every vertex in the tree there is a unique shortest path from the root to this vertex.

*Depth of a node* is the length from the root to this node.

Example: Depth(root) = 0,

*Depth of a tree* is the maximal depth of a node in the tree.

Claim: if *depth(tree) = d*, then there is a leaf in a tree of depth d.

*Size of a tree* is the total number of nodes in the tree.

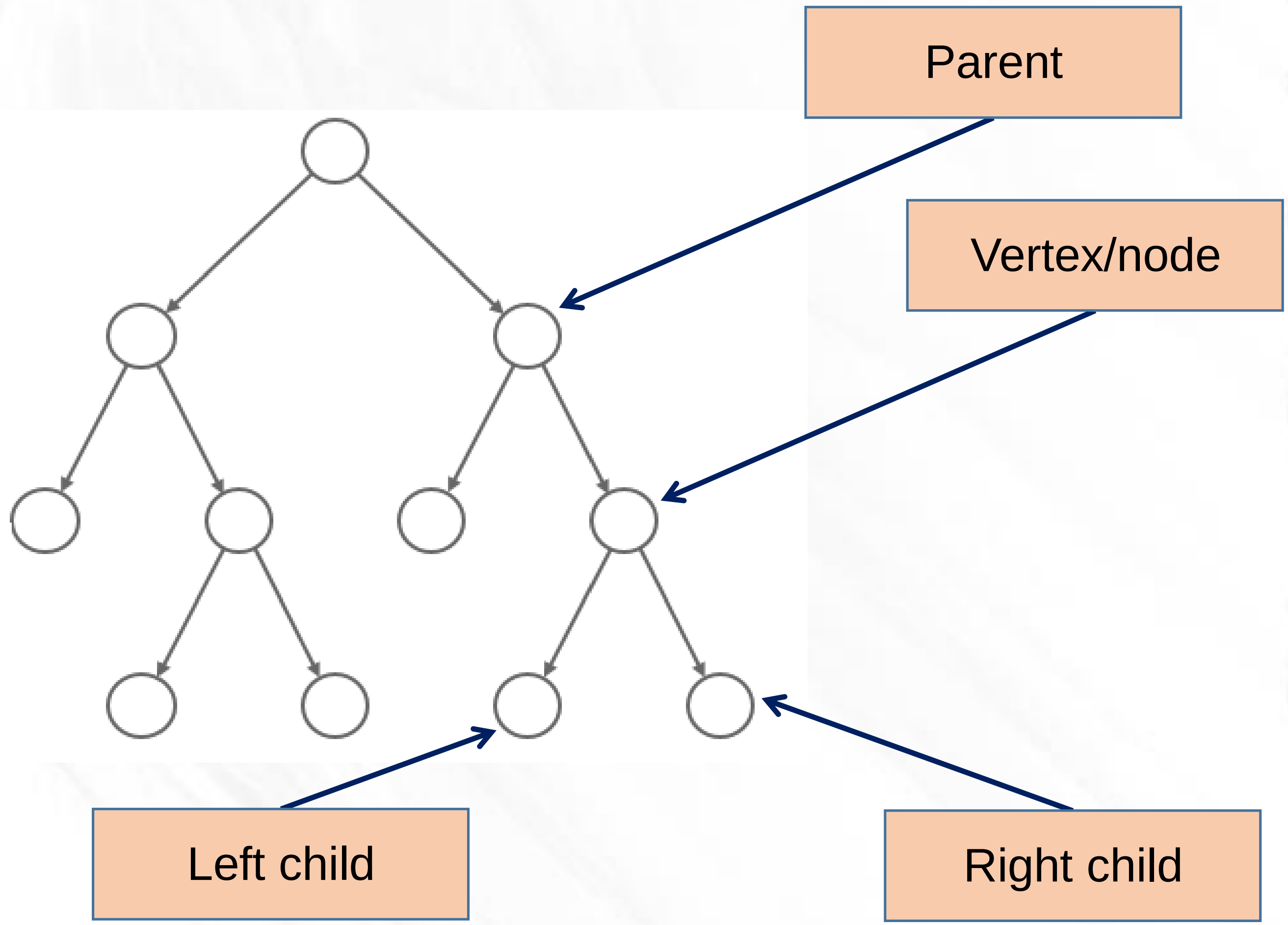
# Binary Trees

# Binary Trees

- An ***m-ary tree*** is a tree in which each vertex has at most *m* children.
- This course: focus on ***binary trees:*** 2-ary trees.

# Binary Trees

Structure of rooted trees

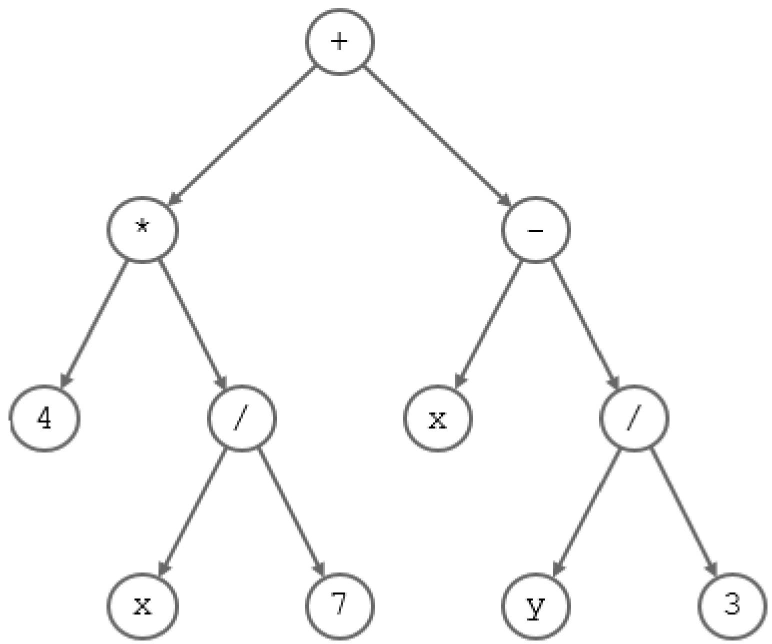


# Binary Trees

- An *m-ary tree* is a tree in which each vertex has at most *m* children.
- The common name for a 2-ary tree is *binary tree*.
- Usage: algebraic expressions.

Represents the expression:

(4*(x/7)) + (x-(y/3))




Q. How do you evaluate an expression tree?

# Binary Trees

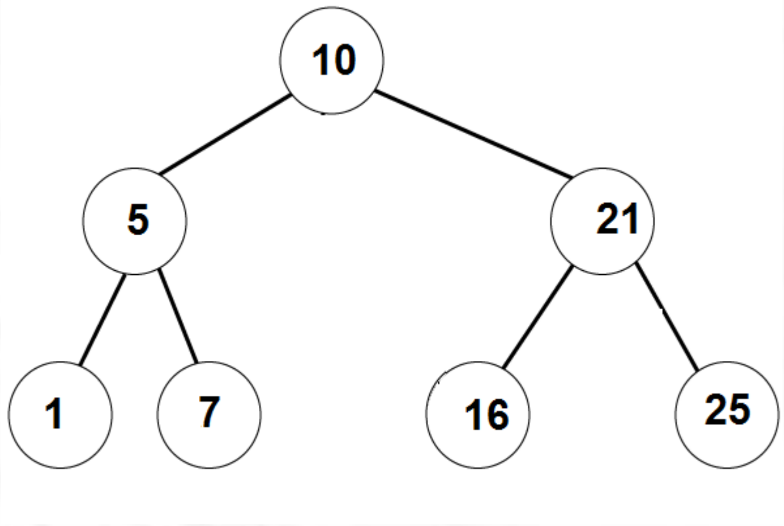
- Claim: In binary tree there are *at most* $2^k$ nodes in level k.
- Proof: Induction on the depth of a node.
  - Base case: For depth = 0.
    There is only the root, So the number of nodes is $1 = 2^0$.
  - Induction:

    Suppose the claim holds for depth = d.

    Let's prove it for depth = d+1:
  - By the induction hypothesis, there are at most $2^d$ nodes at level d
  - Each of these node has at most 2 children.
  - Therefore, the number of nodes at level d+1 is at most $2*2^d=2^{d+1}$.
- Therefore, if a binary tree has depth d, then it has at most $1+2+4+…+2^d = 2^{d+1}-1$ nodes.

# Binary Trees

- A binary tree is ***full*** if for all k<=depth it has $2^k$ nodes in level k.



# Binary Trees

Claim1:If a binary tree has N vertices, then its depth is at least log(N)-1.

Claim2: If a binary tree has N vertices, then its depth is at most N-1.

Proof of Claim 1: Denote the depth by d.

Then $N <= 2^{d+1}-1 < 2^{d+1}$ (by the claim from before)

Therefore, $d+1> \log_2(N)$, and hence $d> \log_2(N)-1$

Proof of Claim 2: It has depth N-1 only if it is a path/straight line.

Otherwise the depth will < N-1.

# Binary Trees

Implementing Binary Tree in C:

```
struct BTnode {
    int data;  //
    struct BTnode* left; // left child
    struct BTnode* right; // right child
    struct  BTnode* parent;
}
typedef struct BTnode  BTnode_t;


struct binary_tree {
    struct BTnode* root;
}
```

# Binary Trees

Write a function that gets a BTnode, and checks if the node is a leaf.

```
is_leaf(BTnode* node)
    return (node->left == NULL && node->right == NULL);
```

Write a function that gets a BTnode, and checks if the node is a root.

```
is_root(BTnode* node)
    return (node->parent==NULL);
```

# Binary Trees

Write a function that gets a binary tree and computes its size.

# Binary Trees

```
get_size (BTnode* root)
{
    if  ( root == NULL )
        return  0;
    else
        return  get_size (root->left ) + get_size (root->right ) + 1;
}
```

# Binary Trees

Write a function that gets a binary tree and computes its depth.

# Binary Trees

```
get_depth (BTnode* root) {
    if  ( root == NULL )
        return  -1;
   // if (is_leaf(root))
   //     return 0;
    else
        return  max(get_depth (root->left ), get_depth (root->right ) ) +1;
}
```

# Traversing Binary Trees

# Traversing Binary Trees

<u>InOrder traversal:</u>

- First visit the left subtree,
- Then the root,
- Then the right subtree.

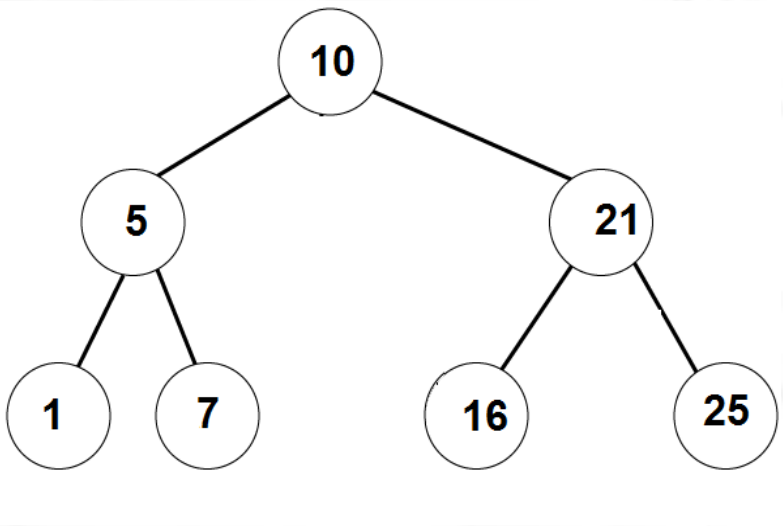
Example:

__left___ 10, __right__

__**,5,**__, 10, ___ **21** __

1, 5, 7, 10, 16, 21, 25

Answer: [1,5,7,10,16,21,25]



# Traversing Binary Trees

PreOrder traversal:

- First visit the root,
- Then the left subtree,
- Then the right subtree.

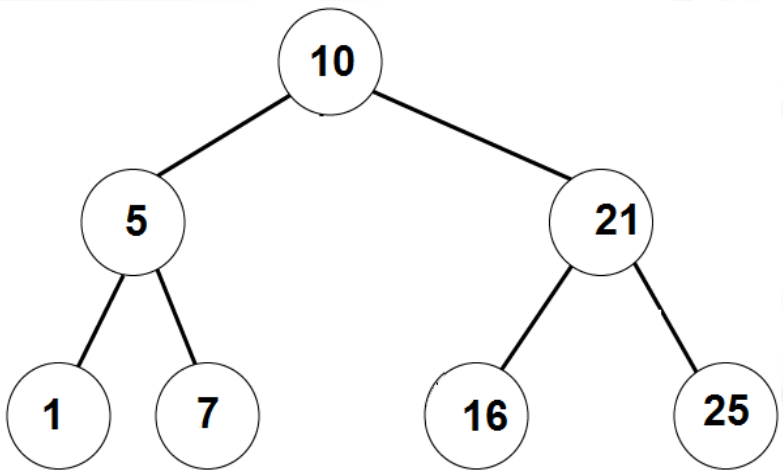
Example:

10, __left__, ___right___

10,  5,_left_, _right_,  21, left_,right_

10,    5, 1, 7,    21, 16, 25

Answer: [10,5,1,7,21,16,25]



# Traversing Binary Trees

PostOrder traversal:

- First visit the left subtree,
- Then the right subtree.
- Then the root

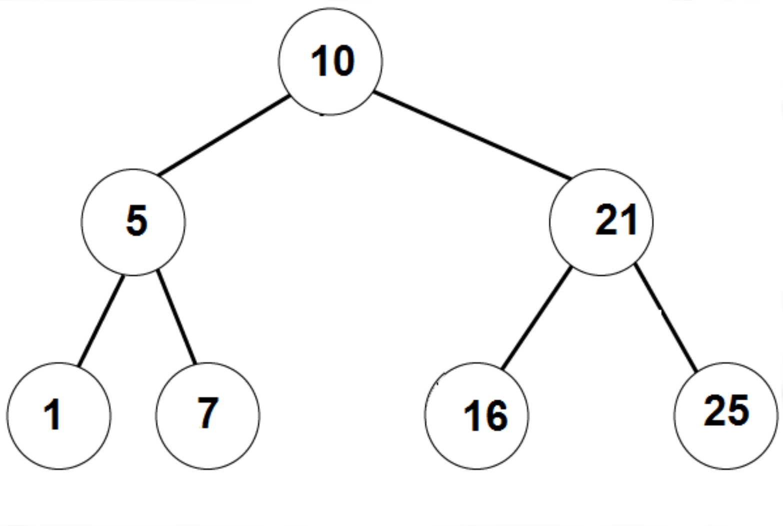
Example:

__left__, ___right_, 10

_left_, right_, 5,   _left_, _right_,21,  10

1, 7, 5,   16, 25, 21,    10

Answer: [1,7,5,16,25,21,10]



# PreOrder traversal algorithm

// prints the tree in PreOrder

PreOrderTraversal (BTnode root):

1. If root==NULL

   1.1 do nothing

2. Else

   2.1 print( root.data )

   2.2 PreOrderTraversal (root.left)

   2.3 PreOrderTraversal(root.right)

*Q:Change the algorithm to get the InOrder/PostOrder traversal*

# Practice Problems on Binary Trees

Is descendant?

# Binary Tree Problems

Write a function that gets two nodes in a binary tree, u and v and checks if u is a descendant of v

bool is_descendant (BTnode_t* u, BTnode_t* v):

**Idea:**

If (u ==v) return true.

If (u->parent == NULL) return false

Set p = u->parent

While (p!=NULL)

    If (p == v) return true

    Set p = p->parent

Return false

*Running time:*

*If u is a descendant of v,*
*Then running time is O(dist(u,v))*

*Otherwise: O(depth(u))*

*Implement this*

# Compute distance

# Binary Tree Problems

Write a function that gets two nodes in a binary tree, u and v and computes the distance between u and v

int distance (BTnode* u, BTnode* v)

distance(0, 1) = 4

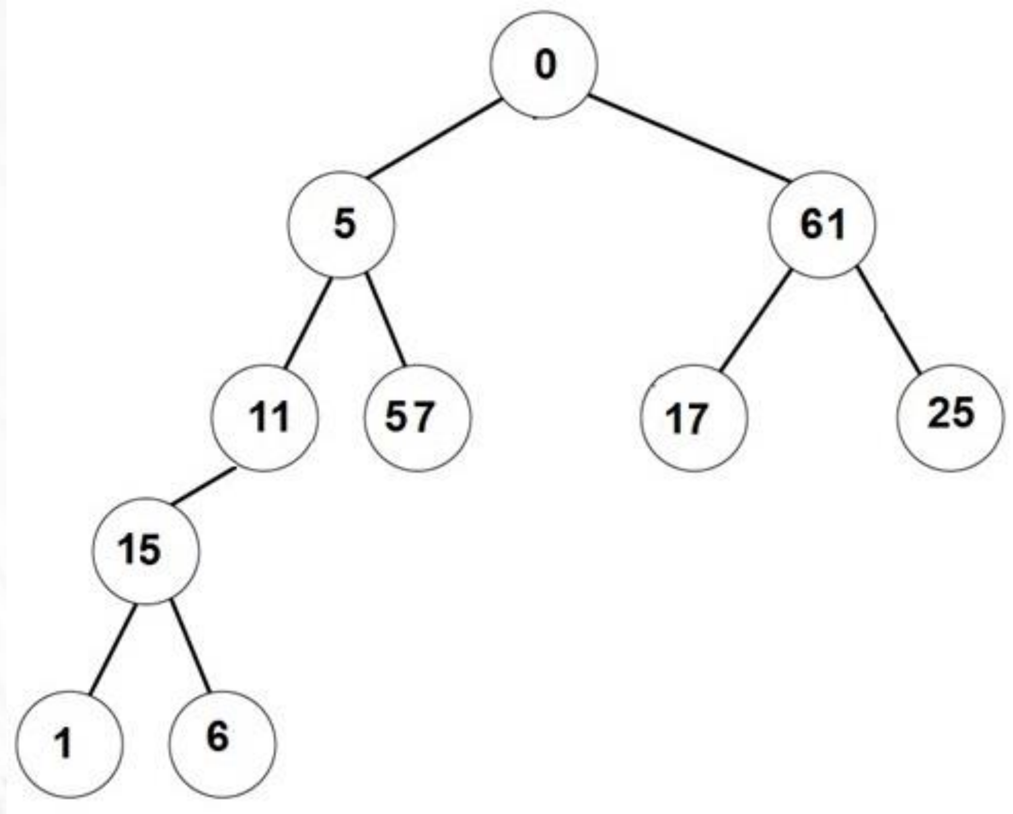
distance(15, 11) = 1

distance(1, 6) = 2

distance(1, 25) = 6

distance(6, 6) = 0

distance(57, 1) = 4



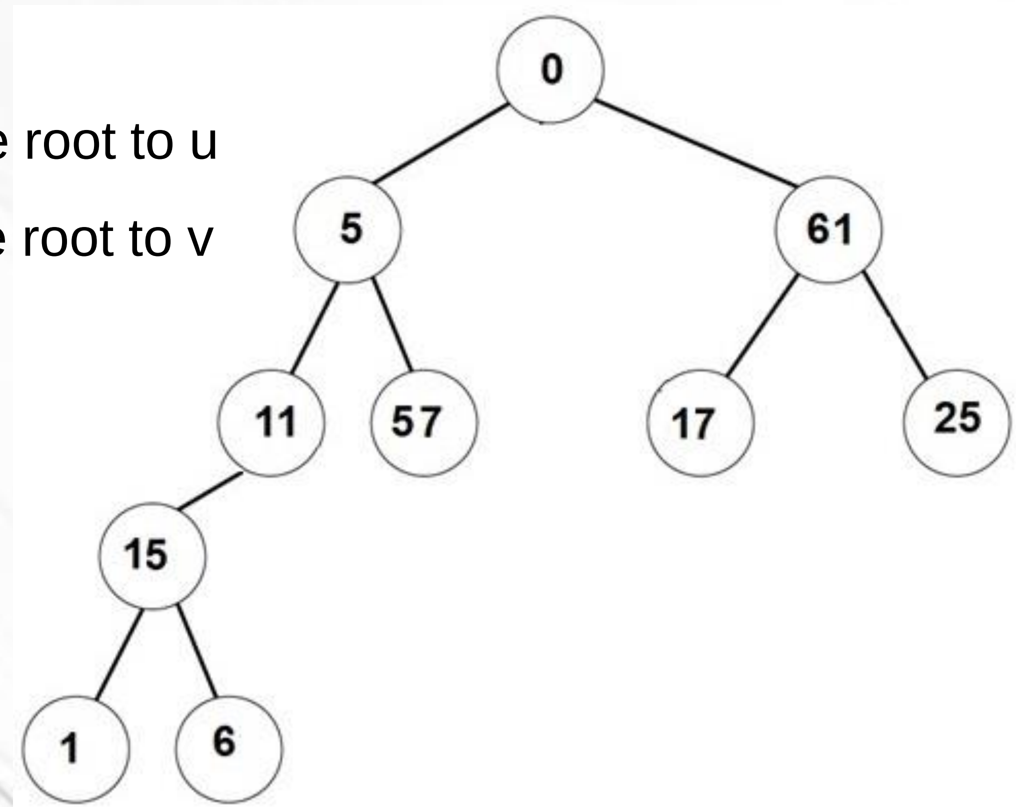
# Binary Tree Problems

Write a function that gets two nodes in a binary tree, u and v and computes the distance between u and v

int distance (BTnode* u, BTnode* v)

***Idea:***

1. Compute $Path_u$ = the path from the root to u
2. Compute $Path_v$ = the path from the root to v
3. Find the common ancestors
4. Return ?



# Binary Tree Problems

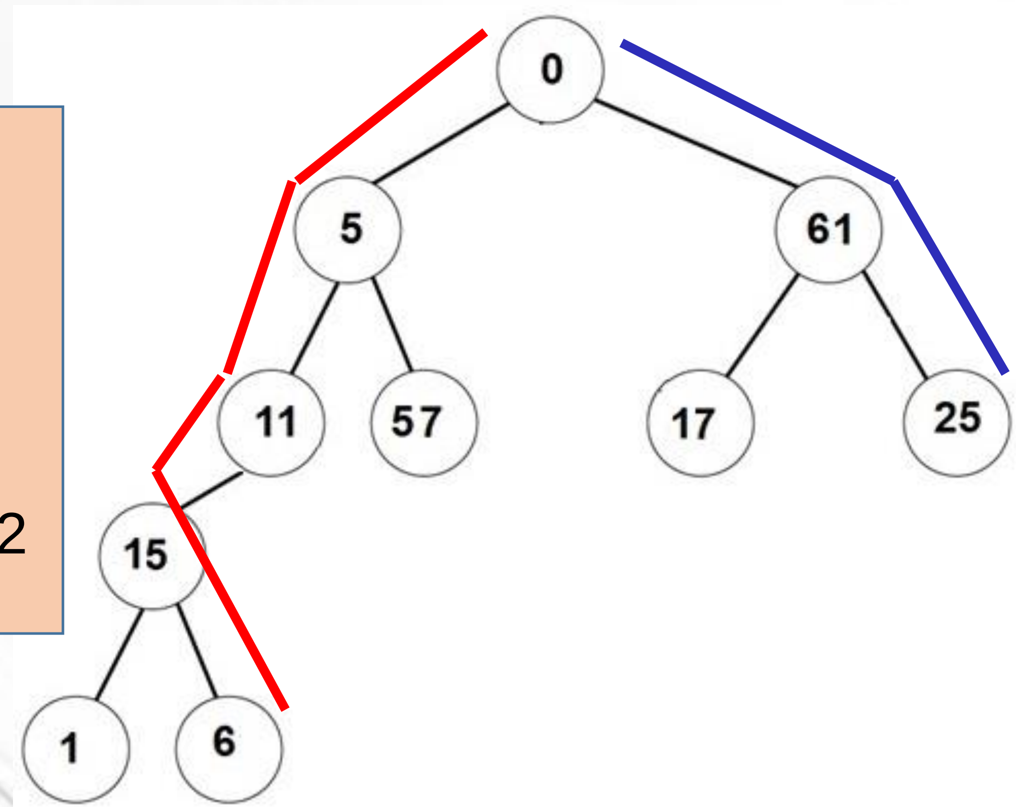
Write a function that gets two nodes in a binary tree, u and v
and computes the distance between u and v

int distance (BTnode* u, BTnode* v)

Example: u = 25, v = 6

$Path_u$ = (0, 61, 25)
$Path_v$ = (0, 5, 11, 15, 6)

Return length($Path_u$)+length($Path_v$)-2



# Binary Tree Problems

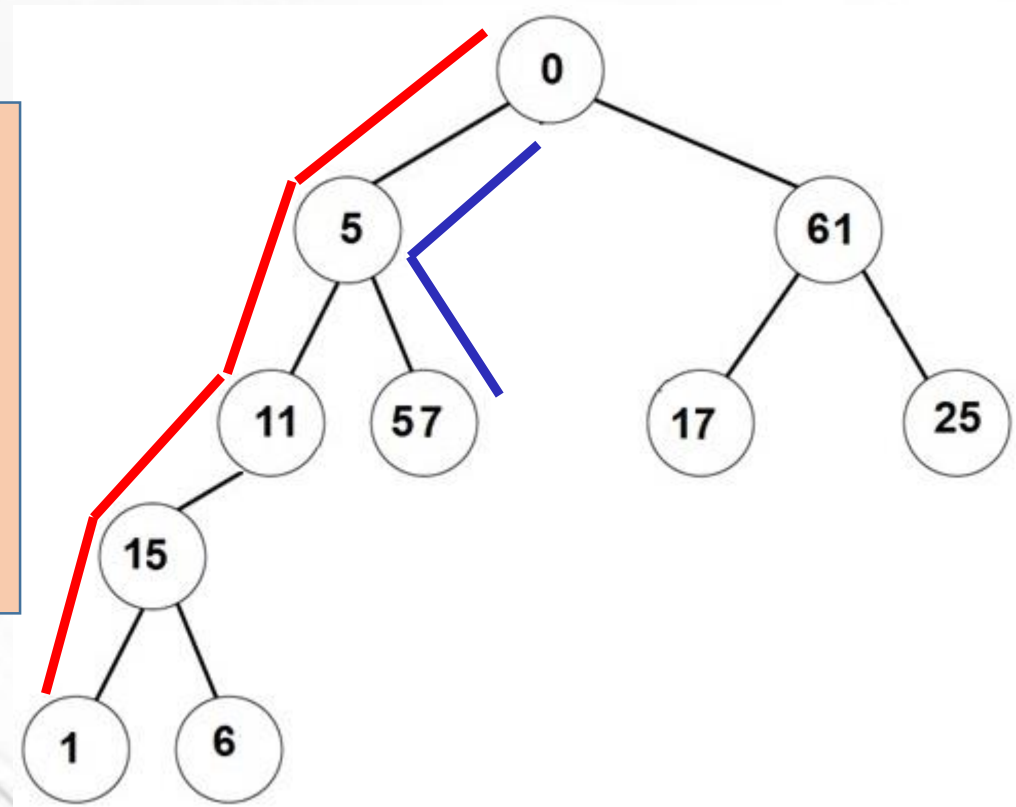
Write a function that gets two nodes in a binary tree, u and v and computes the distance between u and v

int distance (BTnode* u, BTnode* v)

Example: u = 1, v = 57

$Path_u$ = (0, 5, 11, 15, 1)
$Path_v$ = (0, 5, 57)
common ancestors = {0,5}

Return length($Path_u$)+length($Path_v$)-4



# Binary Tree Problems

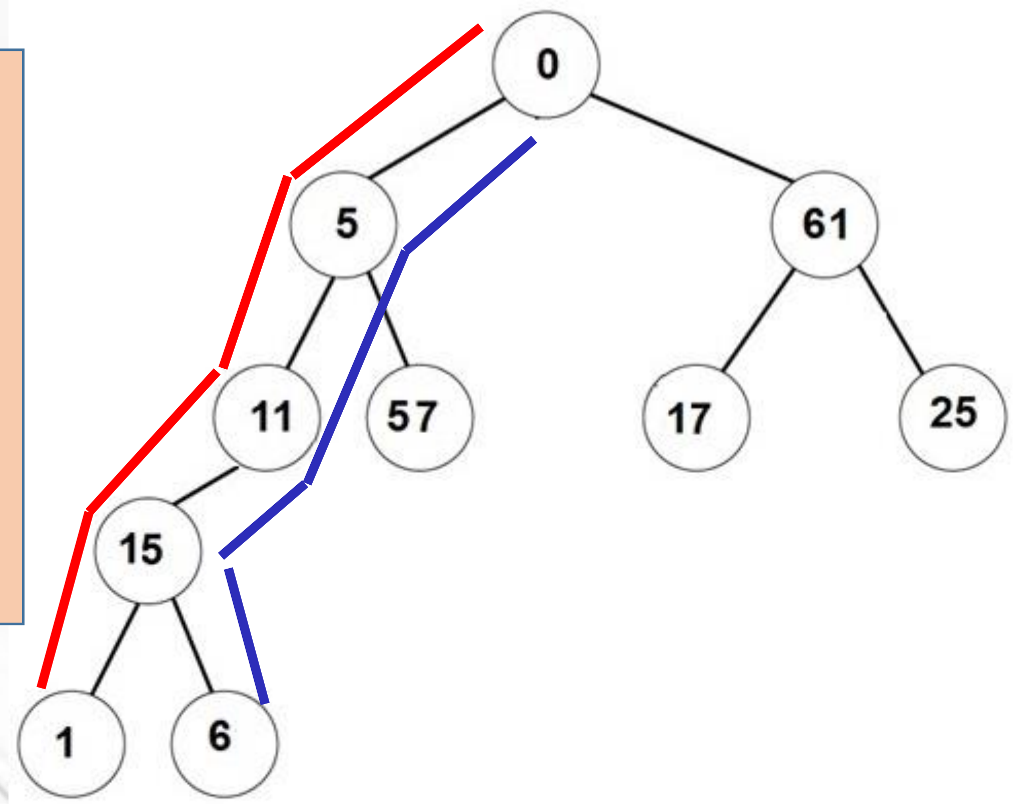
Write a function that gets two nodes in a binary tree, u and v and computes the distance between u and v

int distance (BTnode* u, BTnode* v)

Example: u = 1, v = 6

$Path_u$ = (0, 5, 11, 15, 1)
$Path_v$ = (0, 5, 11, 15, 6)
common ancestors = {0, 5, 11, 15}

Return length($Path_u$)+length($Path_v$)-8

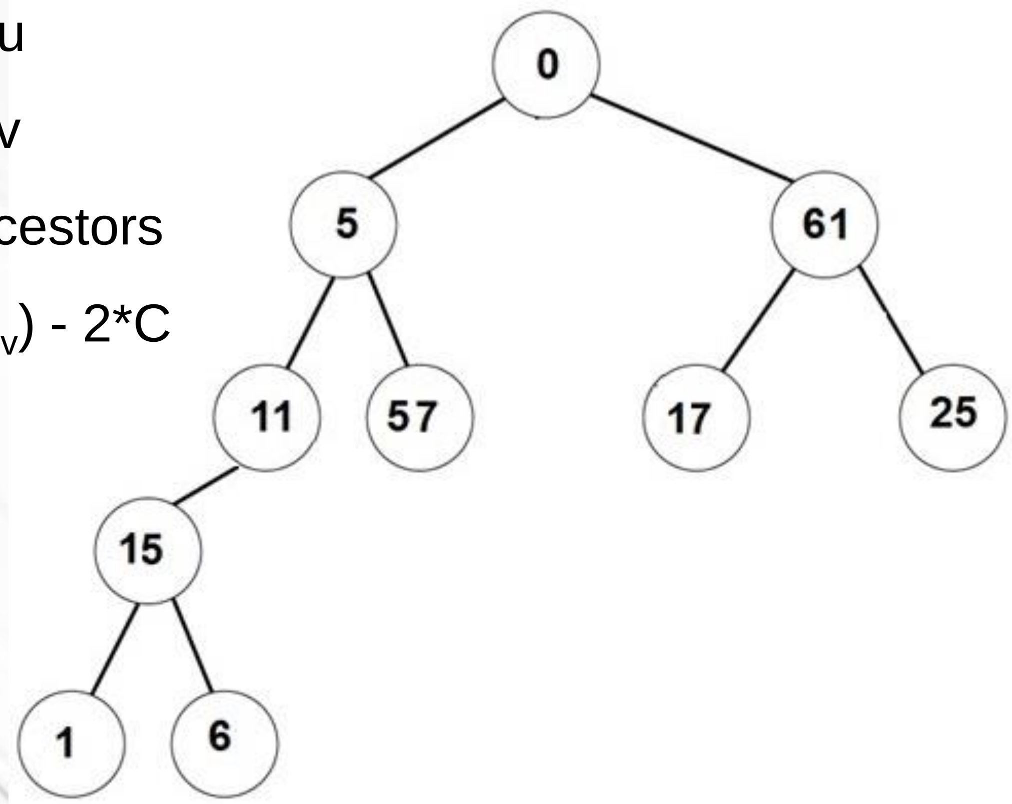


# Binary Tree Problems

Write a function that gets two nodes in a binary tree, u and v and computes the distance between u and v

int distance (BTnode* u, BTnode* v)

1. Compute $Path_u$ = the ancestors of u
2. Compute $Path_v$ = the ancestors of v
3. Let C = the number of common ancestors
4. Return $length(Path_u) + length(Path_v) - 2*C$



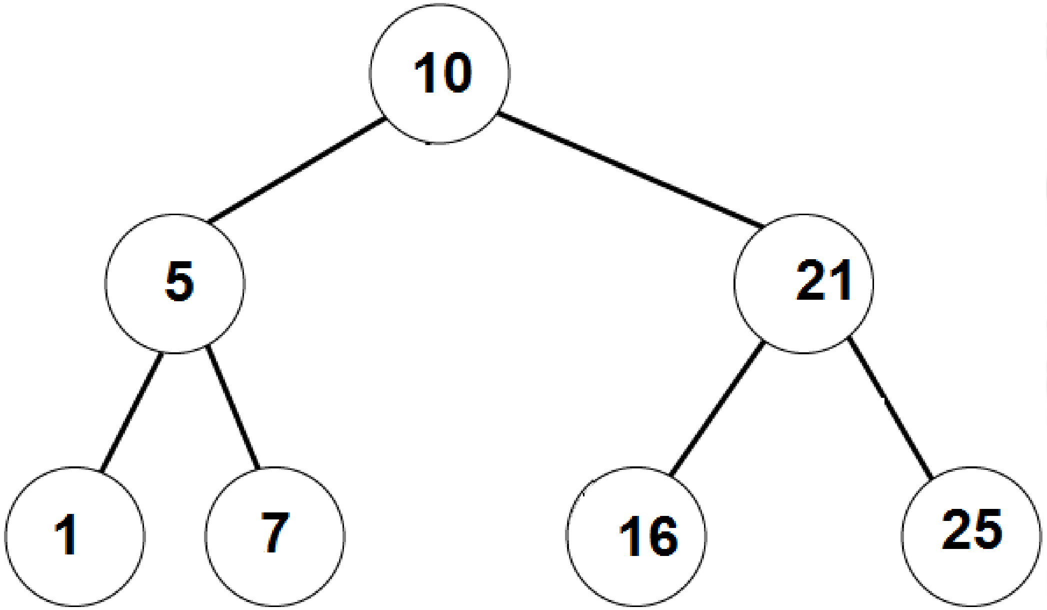

The running time is:
O(depth(u)+depth(v))

# Breadth First Search

# Tree traversal – non-recursive

Write a non-recursive algorithm that prints PreOrder traversal of a binary tree.

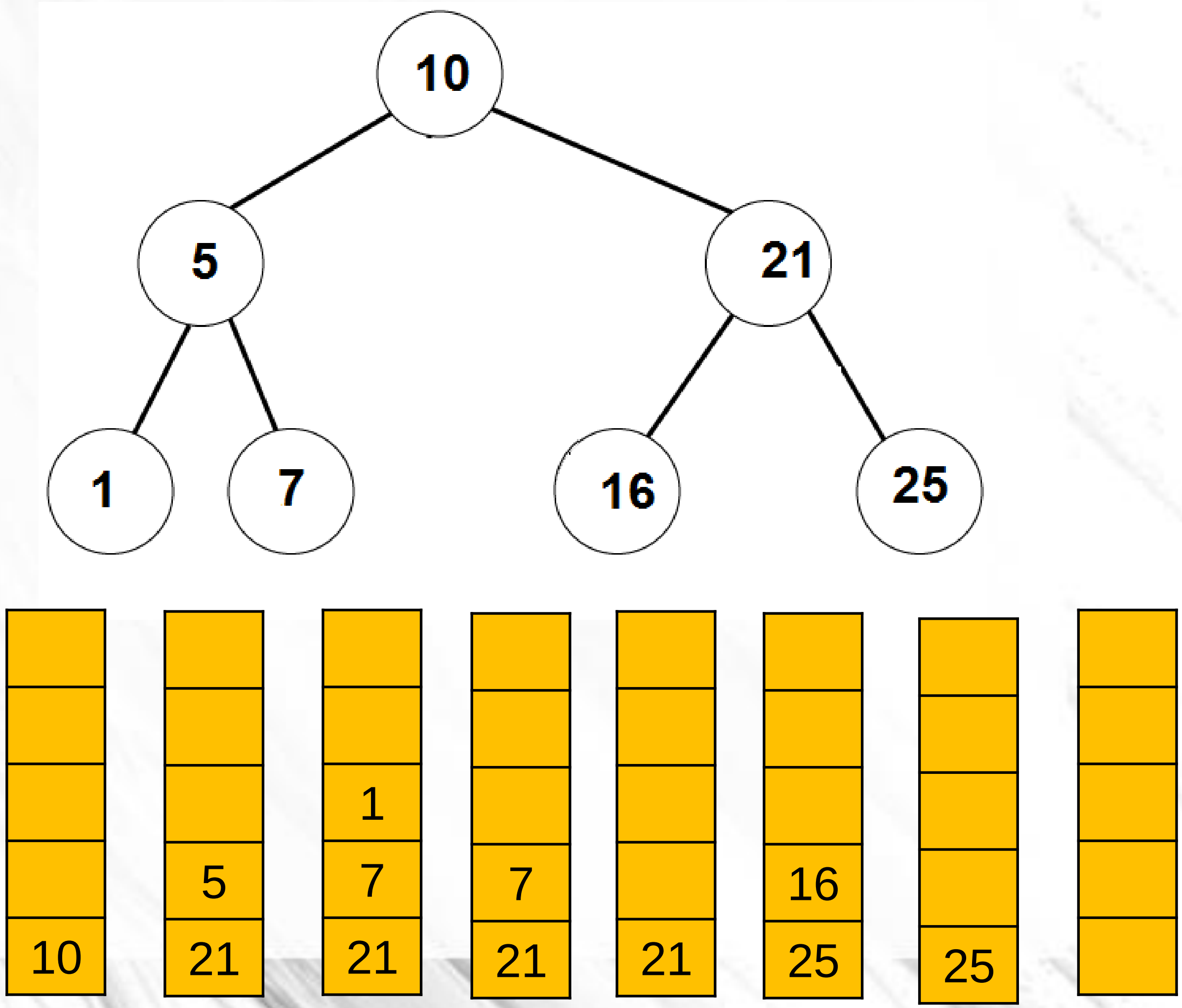
```
PreOrder(root):
    s = create stack of nodes
    s.push(root)
    While s is not empty:
        node = s.pop()
        printf(node->value
        if (node->right != NULL)
            s.push(node->right)
        if (node->left != NULL)
            s.push(node->left)
```



# Tree traversal – non-recursive

Write a non-recursive algorithm that prints PreOrder traversal of a binary tree.

```
PreOrder(root):
    s = create stack of nodes
    s.push(root)
    While s is not empty:
        node = s.pop()
        printf(node->value)
        if (node->right != NULL)
            s.push(node->right)
        if (node->left != NULL)
            s.push(node->left)
```



# Tree traversal – non-recursive

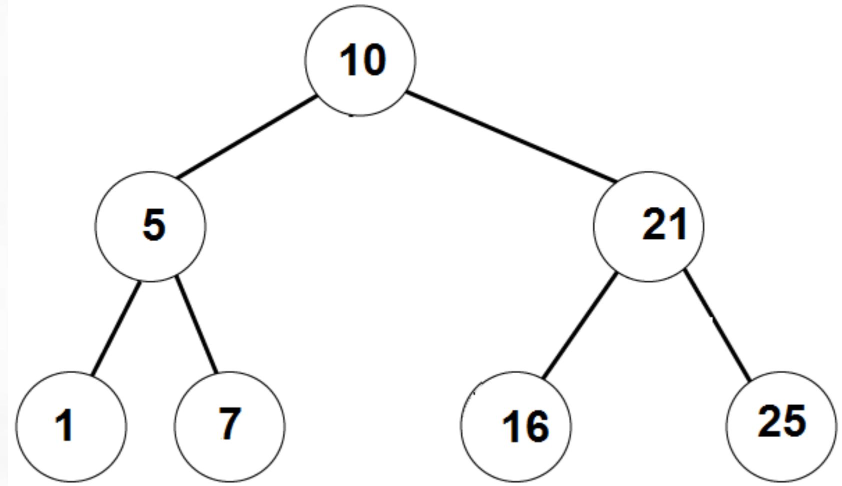
Write a non-recursive algorithm that prints PreOrder traversal of a binary tree.

```
PreOrder(root):
    s = create stack of nodes
    s.push(root)
    While s is not empty:
        node = s.pop()
        printf(node->value)
        if (node->right != NULL)
            s.push(node->right)
        if (node->left != NULL)
            s.push(node->left)
```

Implement this algorithm!

*What if we replace the stack with a queue?*

# Breadth First Search



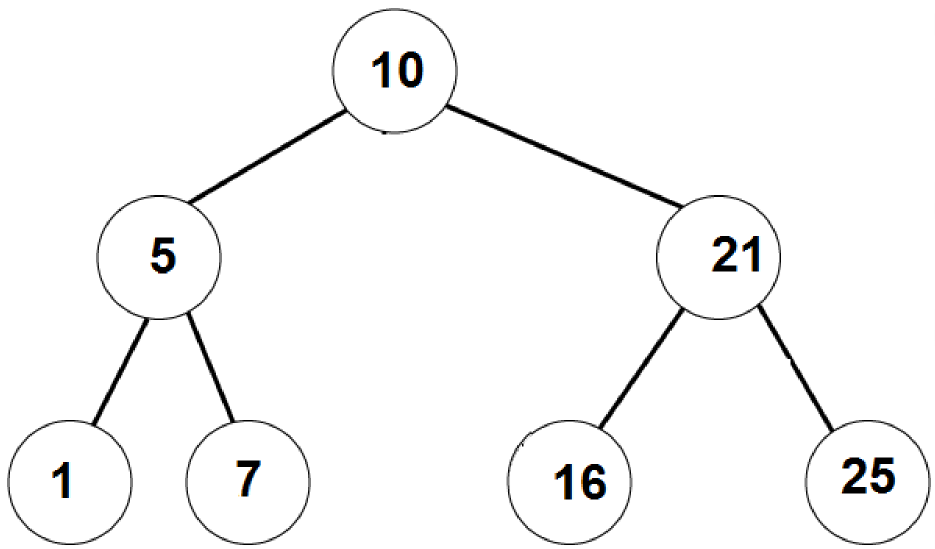

```
BreadthFirstSearch(root):
    q = create queue of nodes
    q.enqueue(root)
    while q is not empty:
        node = q.dequeue()
        printf(node->value)
        if (node->left != NULL)
            q.enqueue(node->left)
        if (node->right != NULL)
            q.enqueue(node->right)
```

| | | | | |
|---|---|---|---|---|
| 10 | | | | *Print 10* |
| 5 | 21 | | | *Print 5* |
| 21 | 1 | 7 | | *Print 21* |
| 1 | 7 | 16 | 25 | *Print 1* |
| 7 | 16 | 25 | | *Print 7* |
| 16 | 25 | | | *Print 16* |
| 25 | | | | *Print 25* |
| | | | | |

Implement this algorithm!

# Breadth First Search

```
BreadthFirstSearch(root):
    q = create queue of nodes
    q.enqueue(root)
    while q is not empty:
        node = q.dequeue()
        printf(node->value)
        if (node->left != NULL)
            q.enqueue(node->left)
        if (node->right != NULL)
            q.enqueue(node->right)
```




*These algorithms have applications to*

- *Exploring unknown territory*
  - *Finding shortest paths*
- *Some AI tasks*
  - *Solving puzzles*

Questions?

Comments?