CSE 1320

Week of 04/08/2019

Instructor : Donna French

```
typedef struct node
{
    int node_number;
    struct node *next_ptr;
}node;
```

Can be almost anything

Declare a linked listhead node

```
node *LinkedListHead = NULL;
```

Declare a stack top

```
node *StackTop = NULL;
```

Declare a queue head and queue tail

```
node *QueueHead, *QueueTail;
```

```
typedef struct node
{
    int node_number;
    struct node *next_ptr;
}node;


node *LinkedListHead = NULL;

AddNode(node_number, &LinkedListHead);
```

We pass the address of the `LinkedListHead` because we want to alter it in the function.

ICQ11

```
void AddNode(int NewNodeNumber, node **LinkedListHead)
{
	node *TempPtr, *NewNode;

	NewNode = malloc(sizeof(node));
	NewNode->node_number = NewNodeNumber;
	NewNode->next_ptr = NULL;

	/* Linked list is empty so point head at new node */
	if (*LinkedListHead == NULL)
	{
		*LinkedListHead = NewNode;
	}
	else
	{
		TempPtr = *LinkedListHead;

		/* Traverse the linked list to find the end node */
		while (TempPtr->next_ptr != NULL)
			TempPtr = TempPtr->next_ptr;

		/* Change end node to point to new node */
		TempPtr->next_ptr = NewNode;
	}
}
```

We passed in the address of the `LinkedListHead` so we need to receive it as a pointer to a pointer since `LinkedListHead` itself is a pointer. (`node *LinkedListHead = NULL;`)

```
void AddNode(int NewNodeNumber, node **LinkedListHead)
{
    node *TempPtr, *NewNode;

    NewNode = malloc(sizeof(node));
    NewNode->node_number = NewNodeNumber;
    NewNode->next_ptr = NULL;
```

This allocates the memory space for the new node and sets the node number and sets the next pointer to `NULL`.

We check if the `LinkedListHead` is pointing to `NULL`.

If it is, then it is not pointing to anything yet/has no links.

We initialized it to `NULL` when we created it.

```
node *LinkedListHead = NULL;
```

```
/* Linked list is empty so point head at new node */
if (*LinkedListHead == NULL)
{
    *LinkedListHead = NewNode;
}
```

Since we don't want to mess with the `LinkedListHead`, we set `TempPtr` equal to the `LinkedListHead` and manipulate it.

```
else
{
     TempPtr = *LinkedListHead;

     /* Traverse the linked list to find the end node */
     while (TempPtr->next_ptr != NULL)
          TempPtr = TempPtr->next_ptr;

     /* Change end node to point to new node */
     TempPtr->next_ptr = NewNode;
}
```

Add the first node to a linked list

```
177             node *LinkedListHead = NULL;
(gdb) p LinkedListHead
$1 = (node *) 0x0
```

Set `node_number` to 11

```
17          node *TempPtr, *NewNode;

19          NewNode = malloc(sizeof(node));
20          NewNode->node_number = NewNodeNumber;
21          NewNode->next_ptr = NULL;

(gdb) p NewNode
$2 = (node *) 0x602010

(gdb) p *NewNode
$3 = {node_number = 11, next_ptr = 0x0}
```

```c
/* Linked list is empty so point head at new node */
if (*LinkedListHead == NULL)
{
    *LinkedListHead = NewNode;
}
```

```
(gdb) p *LinkedListHead
$4 = (node *) 0x0
```

```
(gdb) p NewNode
$2 = (node *) 0x602010
```

```
(gdb) step
```

```
(gdb) p *LinkedListHead
$5 = (node *) 0x602010
```

Now we add another node

Set `node_number` to 22

```
node *TempPtr, *NewNode;

NewNode = malloc(sizeof(node));
NewNode->node_number = NewNodeNumber;
NewNode->next_ptr = NULL;
```

```
(gdb) p NewNode
$2 = (node *) 0x602030

(gdb) p *NewNode
$3 = {node_number = 22, next_ptr = 0x0}

(gdb) p *LinkedListHead
$9 = (node *) 0x602010
```

```
/* Linked list is empty so point head at new node */
if (*LinkedListHead == NULL)
{
     *LinkedListHead = NewNode;
}
else
{
     TempPtr = *LinkedListHead;

     /* Traverse the linked list to find the end node */
     while (TempPtr->next_ptr != NULL)
          TempPtr = TempPtr->next_ptr;

     /* Change end node to point to new node */
     TempPtr->next_ptr = NewNode;
}
```

```
(gdb) p *LinkedListHead
$9 = (node *) 0x602010
```

```
else
{
      TempPtr = *LinkedListHead;

(gdb) p TempPtr
$11 = (node *) 0x602010

      /* Traverse the linked list to find the end node */
      while (TempPtr->next_ptr != NULL)
            TempPtr = TempPtr->next_ptr;

(gdb) p TempPtr->next_ptr
$12 = (struct node *) 0x0

      /* Change end node to point to new node */
      TempPtr->next_ptr = NewNode;

(gdb) p NewNode
$2 = (node *) 0x602030
(gdb) p *TempPtr
$14 = {node_number = 11, next_ptr = 0x602030}
```

```
(gdb) p *LinkedListHead
$9 = (node *) 0x602010
```

```
void AddNode(int NewNodeNumber, node **LinkedListHead)
{
        node *TempPtr, *NewNode;

        NewNode = malloc(sizeof(node));
        NewNode->node_number = NewNodeNumber;
        NewNode->next_ptr = NULL;

        /* Linked list is empty so point head at new node */
        if (*LinkedListHead == NULL)
        {
                *LinkedListHead = NewNode;
        }
        else
        {
                TempPtr = *LinkedListHead;

                /* Traverse the linked list to find the end node */
                while (TempPtr->next_ptr != NULL)
                        TempPtr = TempPtr->next_ptr;

                /* Change end node to point to new node */
                TempPtr->next_ptr = NewNode;
        }
}
```

Add another node

Set `node_number` to 33

```
      NewNode = malloc(sizeof(node));
      NewNode->node_number = NewNodeNumber;
      NewNode->next_ptr = NULL;

(gdb) p *NewNode
$17 = {node_number = 33, next_ptr = 0x0}

      /* Linked list is empty so point head at new node */
      if (*LinkedListHead == NULL)
      {
           *LinkedListHead = NewNode;
      }

(gdb) p *LinkedListHead
$18 = (node *) 0x602010
```

```
      else
      {
            TempPtr = *LinkedListHead;
(gdb) p TempPtr
$19 = (node *) 0x602010

            /* Traverse the linked list to find the end node */
            while (TempPtr->next_ptr != NULL)
                  TempPtr = TempPtr->next_ptr;
(gdb) p TempPtr->next_ptr
$2 = (struct node *) 0x602030
(gdb) p TempPtr
$4 = (node *) 0x602030
(gdb) p TempPtr->next_ptr
$3 = (struct node *) 0x0

            /* Change end node to point to new node */
            TempPtr->next_ptr = NewNode;
(gdb) p NewNode
$5 = (node *) 0x602050
(gdb) p TempPtr->next_ptr
$36 = (struct node *) 0x602050
```

```
(gdb) p *LinkedListHead
$38 = (node *) 0x602010

(gdb) p *LinkedListHead->next_ptr

$39 = {node_number = 22, next_ptr = 0x602050}

(gdb) p *LinkedListHead->next_ptr->next_ptr

$40 = {node_number = 33, next_ptr = 0x0}

Node Number 11  Node Address 0x602010  Node Next Pointer 0x602030

Node Number 22  Node Address 0x602030  Node Next Pointer 0x602050

Node Number 33  Node Address 0x602050  Node Next Pointer (nil)
```

Binary Tree

What is a binary tree?

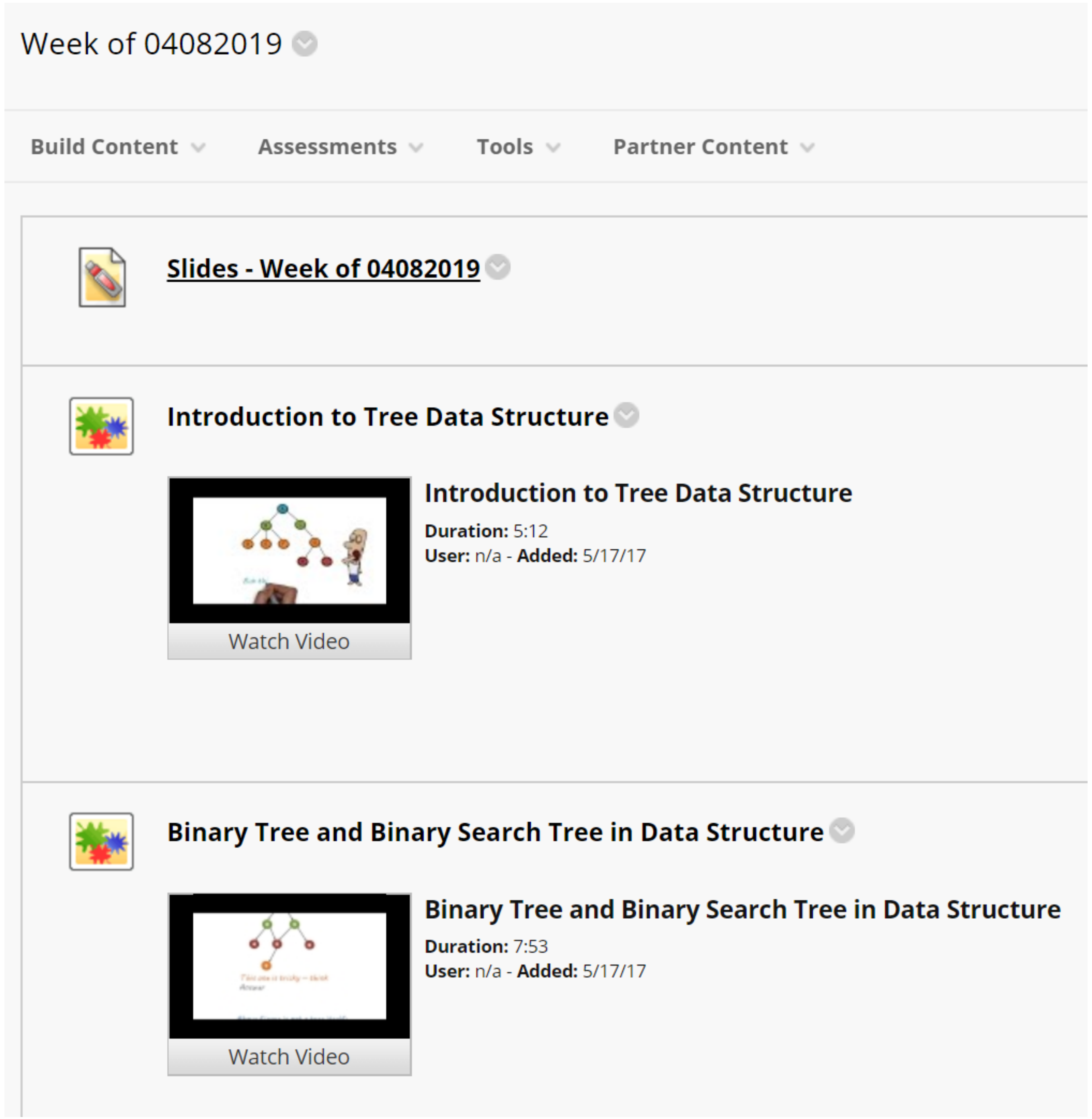
A binary tree is a non-linear tree-like data structure consisting of nodes where each node has up to two child nodes, creating the branches of the tree.

The two children are usually called the left and right nodes.

Parent nodes are nodes with children, while child nodes may include references to their parents.

Binary trees organize data hierarchically instead of linearly.

Binary trees are used to implement binary search trees and binary heaps. They are also often used for sorting data as in a heap sort.



Week of 04082019

Build Content Assessments Tools Partner Content

Slides - Week of 04082019

Introduction to Tree Data Structure

Watch Video

Introduction to Tree Data Structure

Duration: 5:12
User: n/a - **Added:** 5/17/17

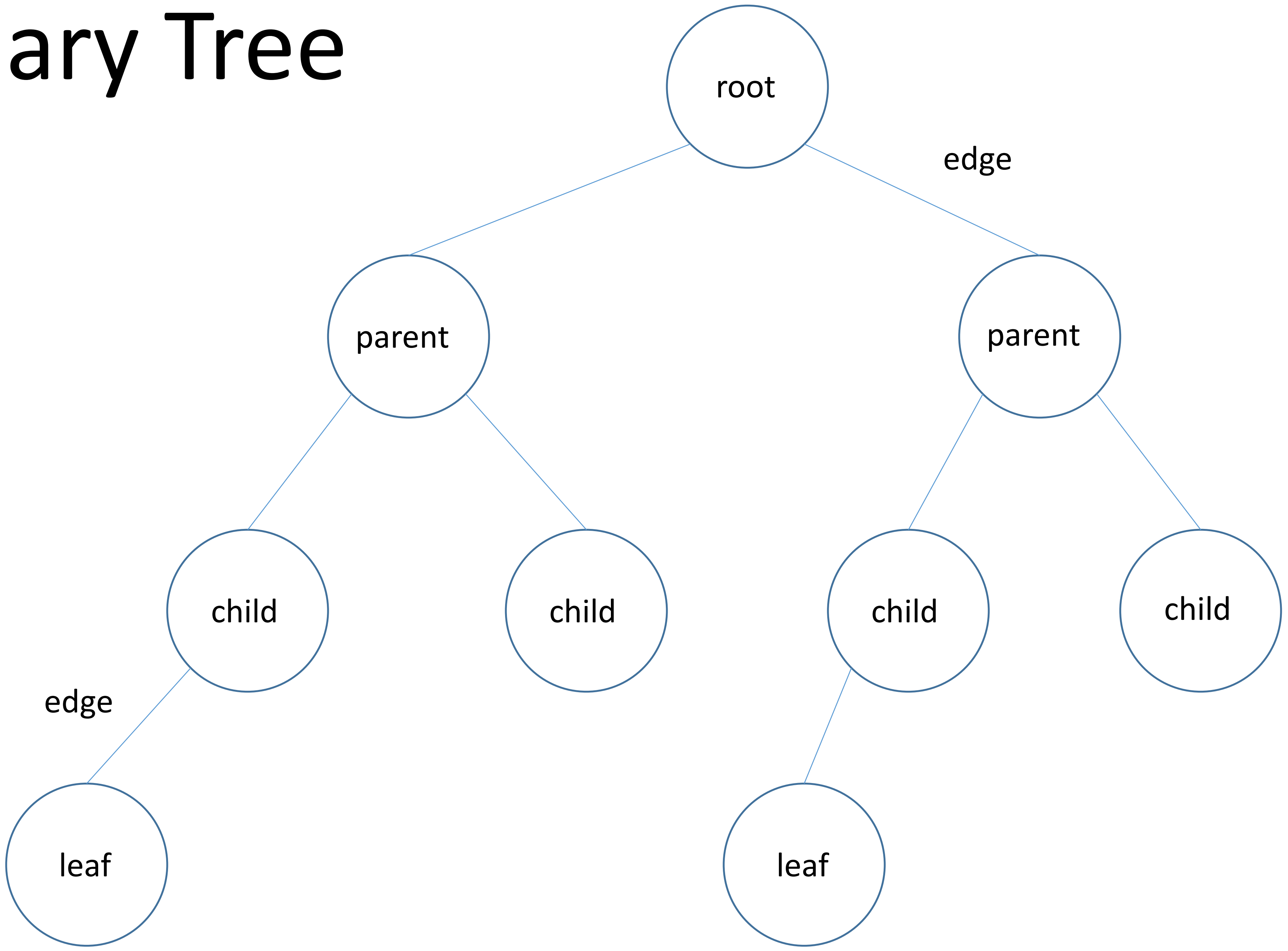
Binary Tree and Binary Search Tree in Data Structure

Watch Video

Binary Tree and Binary Search Tree in Data Structure

Duration: 7:53
User: n/a - **Added:** 5/17/17

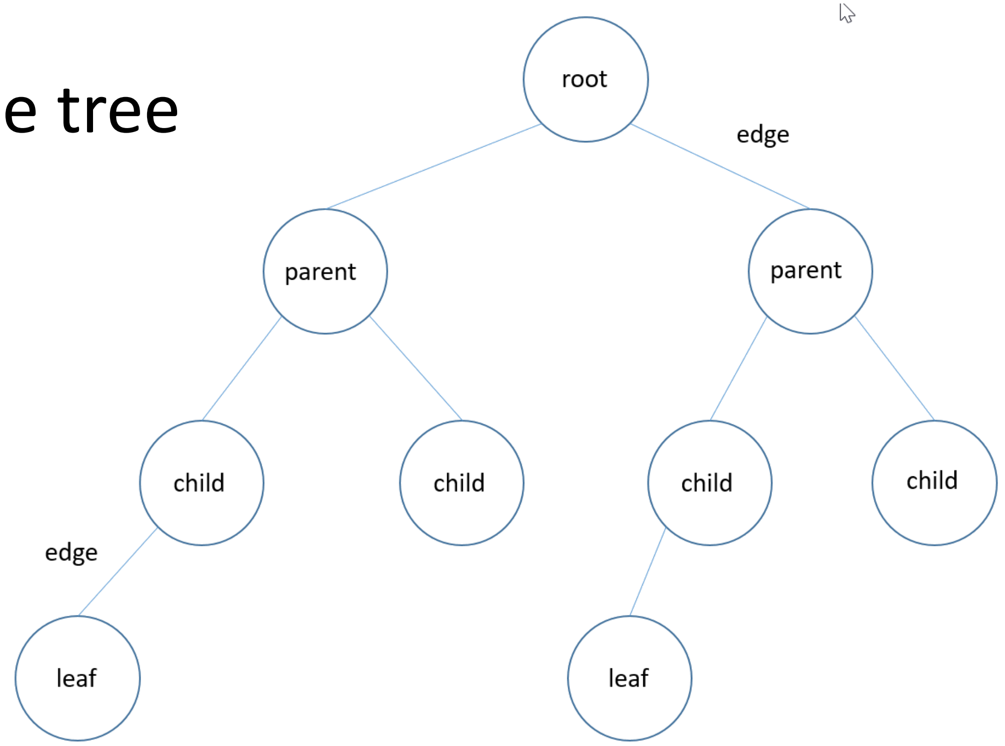
Binary Tree


root
edge
parent
parent
child
child
child
child
edge
leaf
leaf

Binary Tree

Tree Vocabulary

topmost node	root of the tree
node directly under another node	child
node directly above another node	parent
node with no children	leaf
link between two nodes	edge
length of the longest path to a leaf	height
length of the path to its root	depth



Binary Tree vs Linked List

Linked List Node

```
struct node
{
   int node_number;
   struct node *next_ptr;
}


struct node *LinkedListHead;
```

Binary Tree Node

```
struct node
{
   int node_number;
   struct node *left_ptr;
   struct node *right_ptr;
};

struct node *root;
```

Binary Tree vs Linked List

Linked List

```
NewNode = malloc(sizeof(struct node));
NewNode->node_number = NodeNumber;
NewNode->next_ptr = NULL;
```

Binary Tree

```
NewNode = malloc(sizeof(struct node));
NewNode->node_number = NodeNumber;
NewNode->left_ptr = NULL;
NewNode->right_ptr = NULL;
```

Binary Tree vs Linked List

Add a node to the end of a linked list

```
    NewNode = malloc(sizeof(struct node));
    NewNode->node_number = NewNodeNumber;
```

Set the pointer of the last node to the new node

```
    TempPtr->next_ptr = NewNode;
```

Add a node to a binary tree

```
    node = malloc(sizeof(struct node));
    node->node_number = NodeNumber;
    node->left_ptr = NULL;
    node->right_ptr = NULL;
```

Set the parent node’s left or right ptr to the address of the new child

```
struct node
{
        int node_number;
        struct node *left_ptr;
        struct node *right_ptr;
}node;

/* NewNode() allocates a new node with the given data and
   NULL left_ptr and right_ptr pointers */
node *NewNode(int NodeNumber)
{
        // Allocate memory for new node and initialize left_ptr and right_ptr to NULL
        node *node = malloc(sizeof(struct node));
        node->left_ptr = NULL;
        node->right_ptr = NULL;

        // Assign data to this node
        node->node_number = NodeNumber;

        return(node);
}
```

```
/*declare root*/
struct node *root;

/* create root */
root = NewNode(1);

/* following is the tree after above statement

        1
      /   \
     NULL  NULL

*/

printf("\nleft_ptr(1) %p\tright_ptr(1) %p\n",
        root->left_ptr, root->right_ptr);
```

```
Node Number 1 0x1a8a010

left_ptr(1) (nil)       right_ptr(1) (nil)
```

```
root->left_ptr  = NewNode(2);
root->right_ptr = NewNode(3);

printf("\nleft_ptr(2) %p\tright_ptr(3) %p\n",
        root->left_ptr, root->right_ptr);

/* 2 and 3 become left_ptr and right_ptr children of 1

           1
         /   \
        2      3
     /    \    /  \
    NULL NULL NULL NULL
*/
```

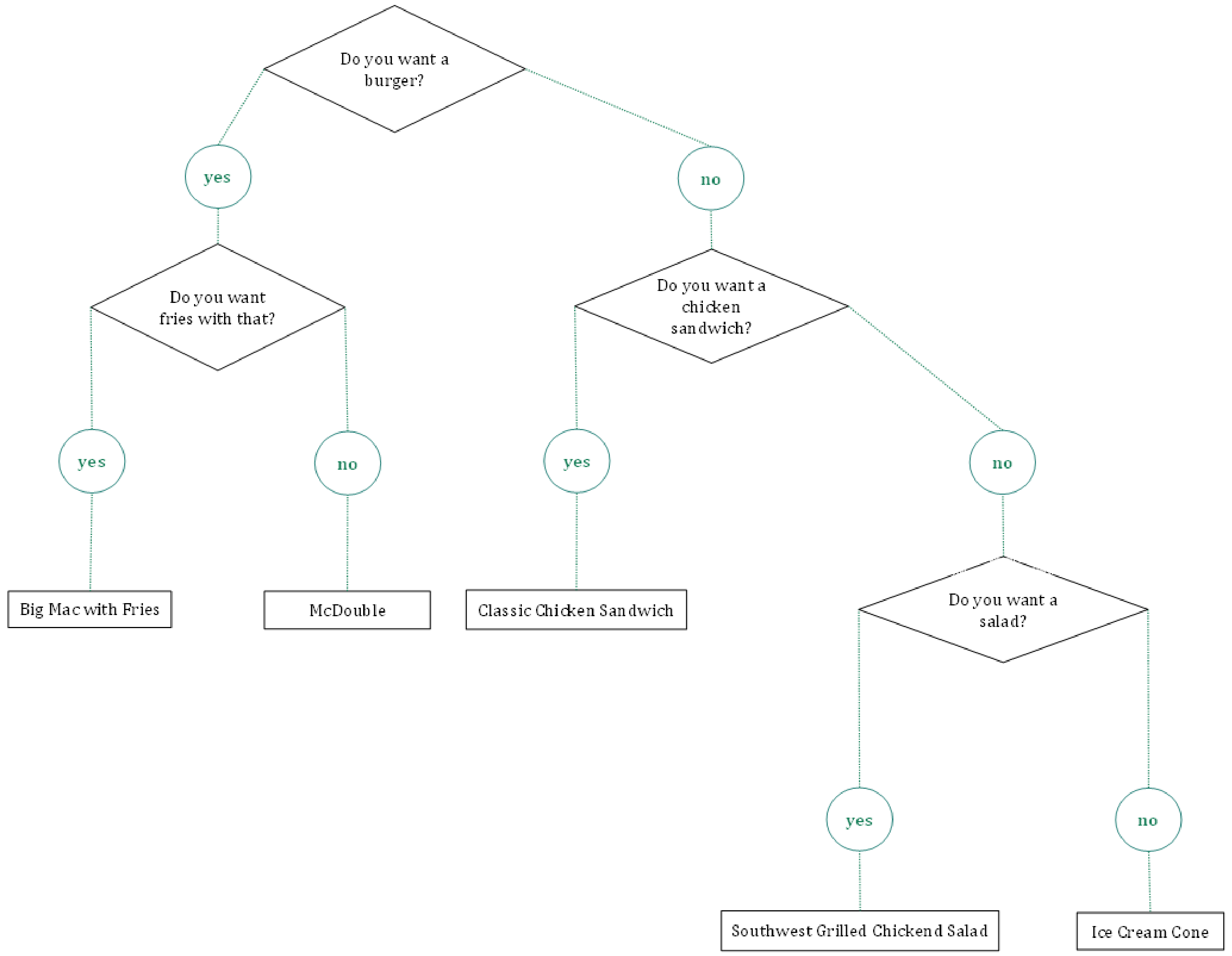
```
Node Number 2 0x1a8a030
Node Number 3 0x1a8a050

left_ptr(2) 0x1a8a030   right_ptr(3) 0x1a8a050
```

BinaryTreeDemo.c

DECISION TREE

Lunch at the McDonald's Drive Thru



Binary Search Tree (BST) Traversals

- Inorder Traversal
  - Gives us the nodes in increasing order
- Preorder Traversal
  - Used to create a copy of the tree
  - File systems use it to track your movement through directories
- Postorder Traversal
  - Used to delete the tree
  - File systems use it to delete folders and the files under them

Binary Search Tree (BST) Traversals

Depth First Tree Traversals

Preorder

Root, Left, Right

4 2 1 3 5

Postorder

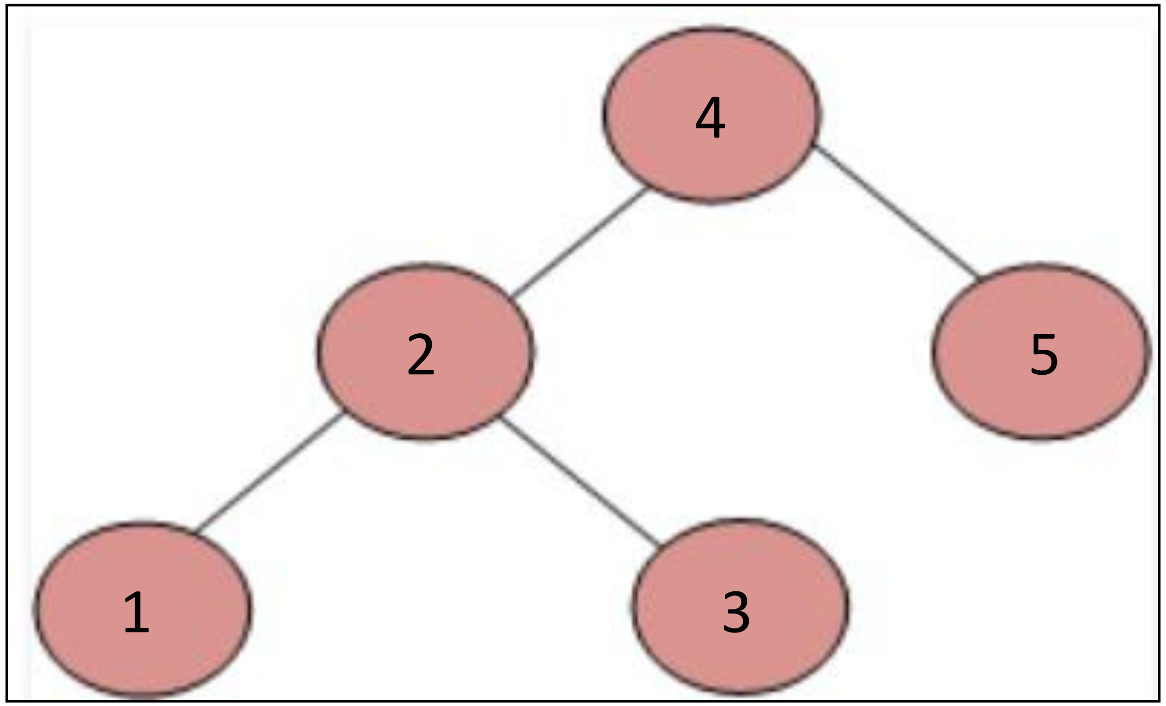
Left, Right, Root

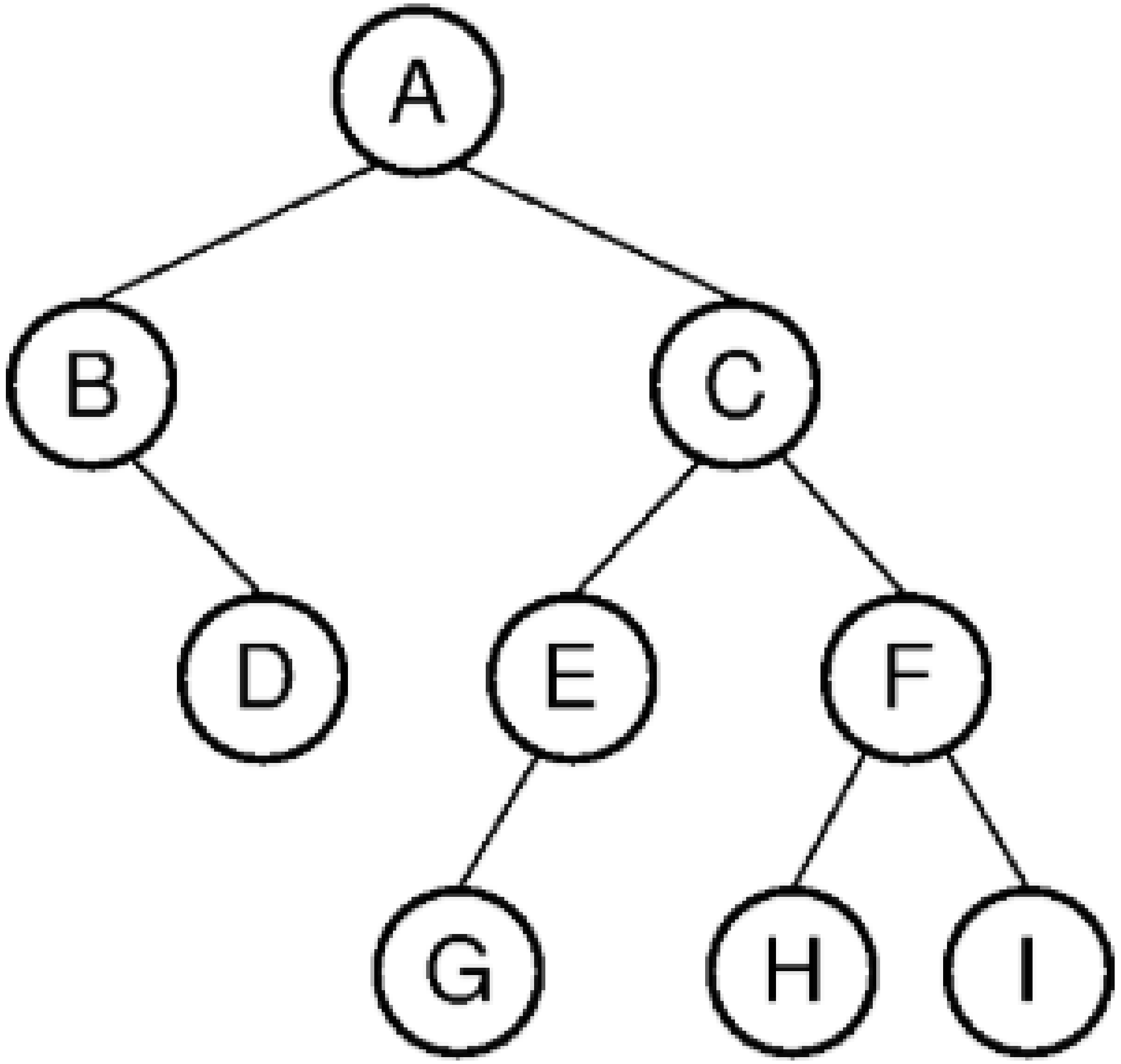
1 3 2 5 4

Inorder

Left, Root, Right

1 2 3 4 5





Depth First Tree Traversals

Preorder

Root, Left, Right

A B D C E G F H I

Postorder

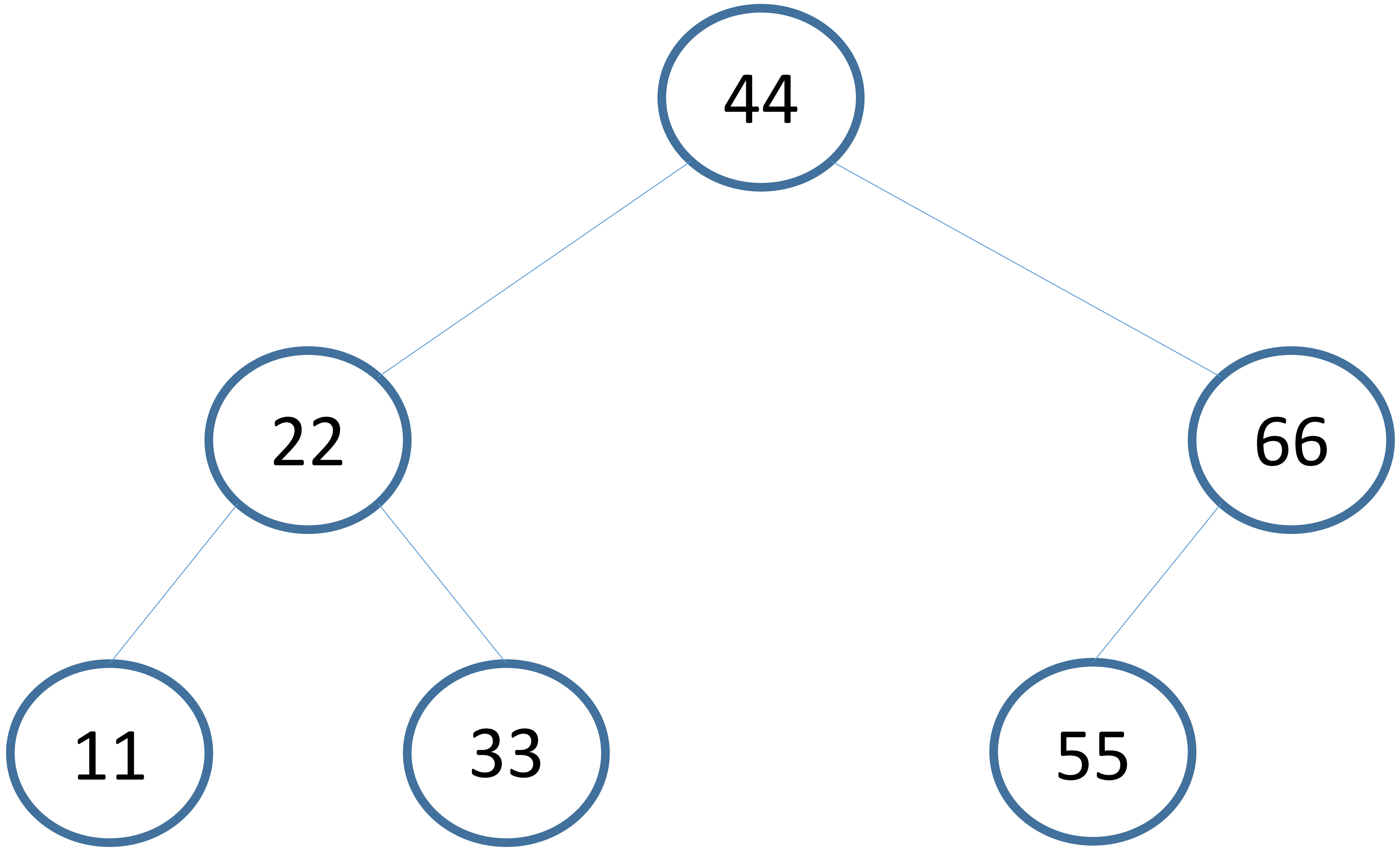
Left, Right, Root

D B G E H I F C A

Inorder

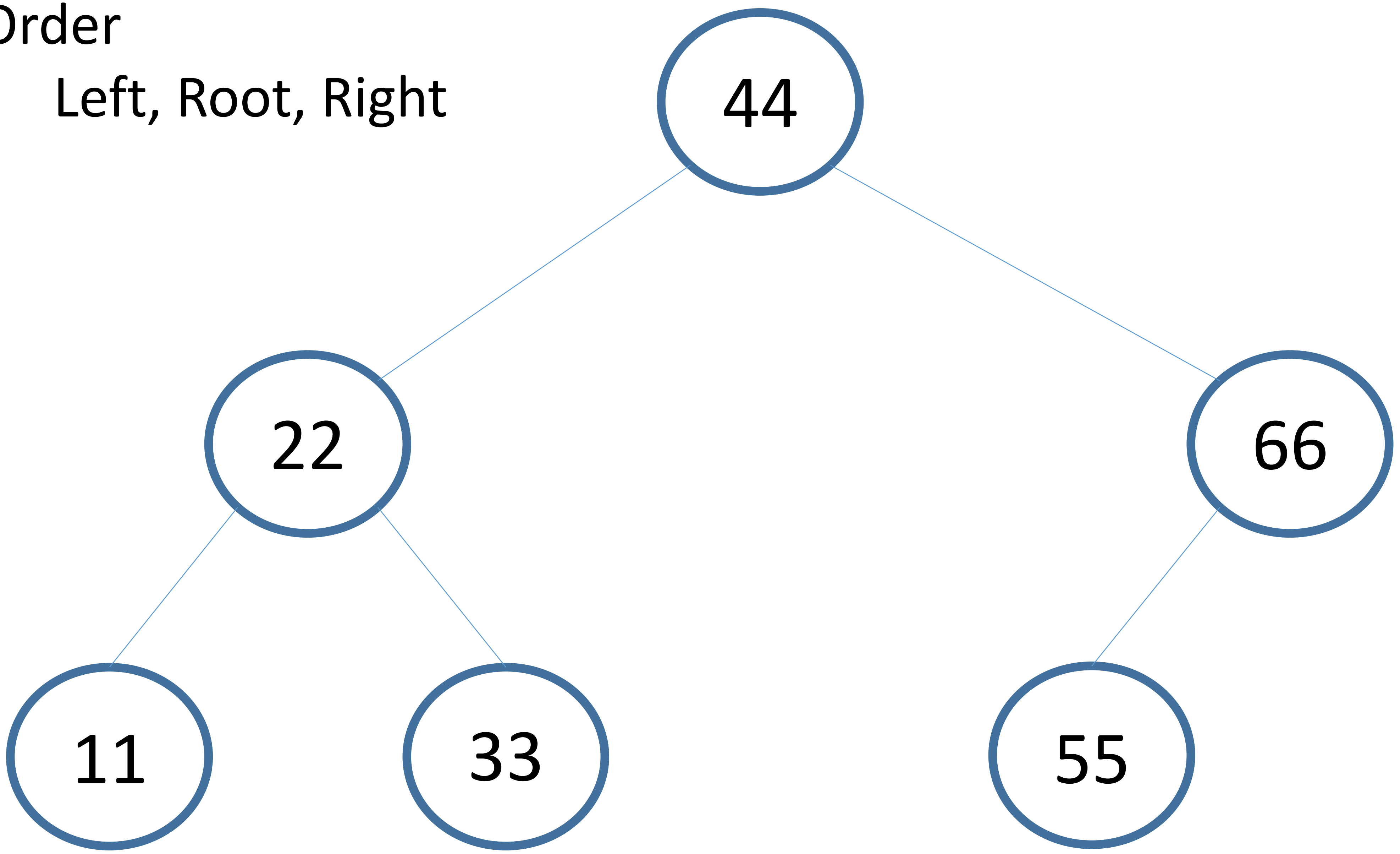
Left, Root, Right

B D A G E C H F I


44
22
66
11
33
55

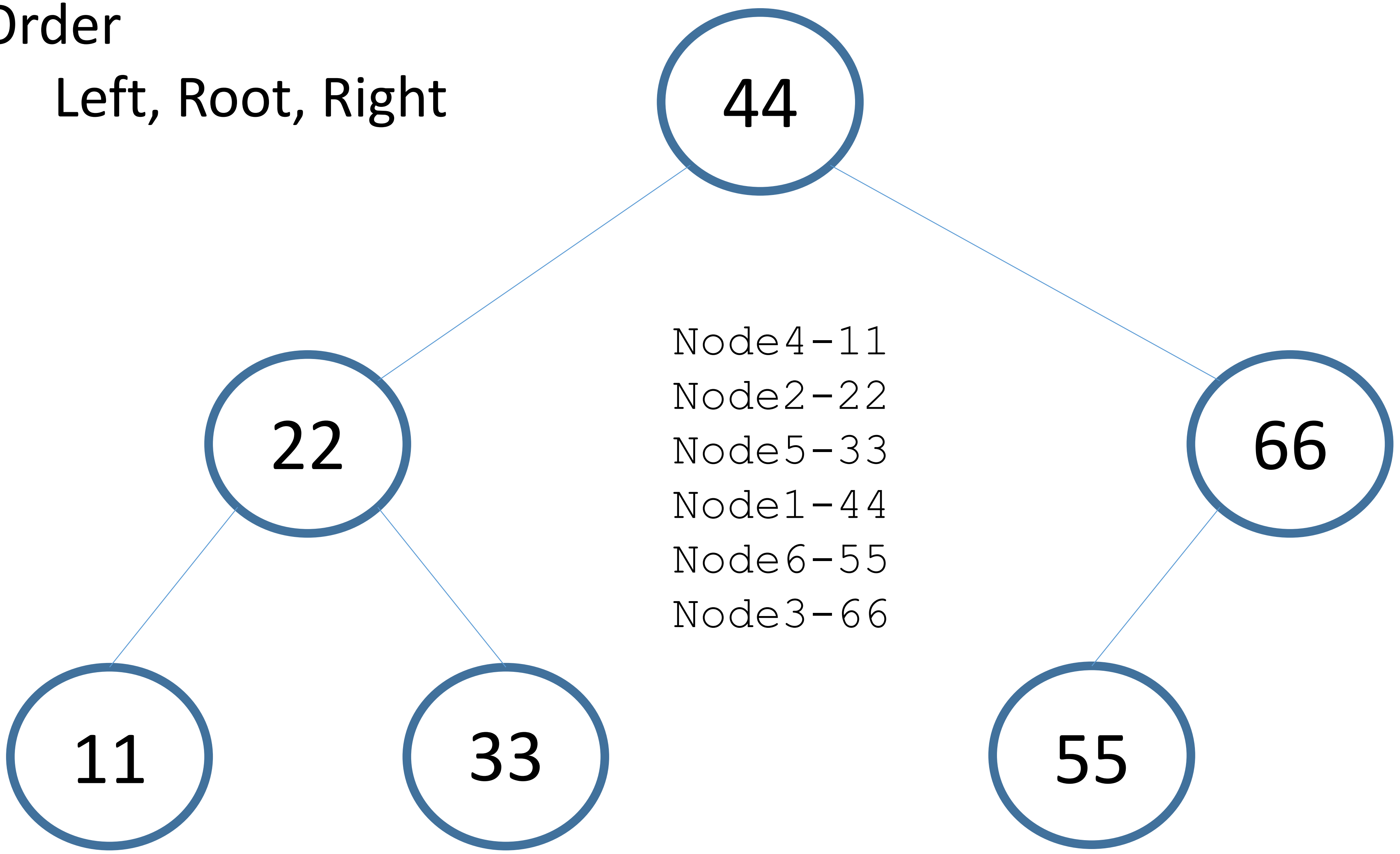
In Order

Left, Root, Right



In Order

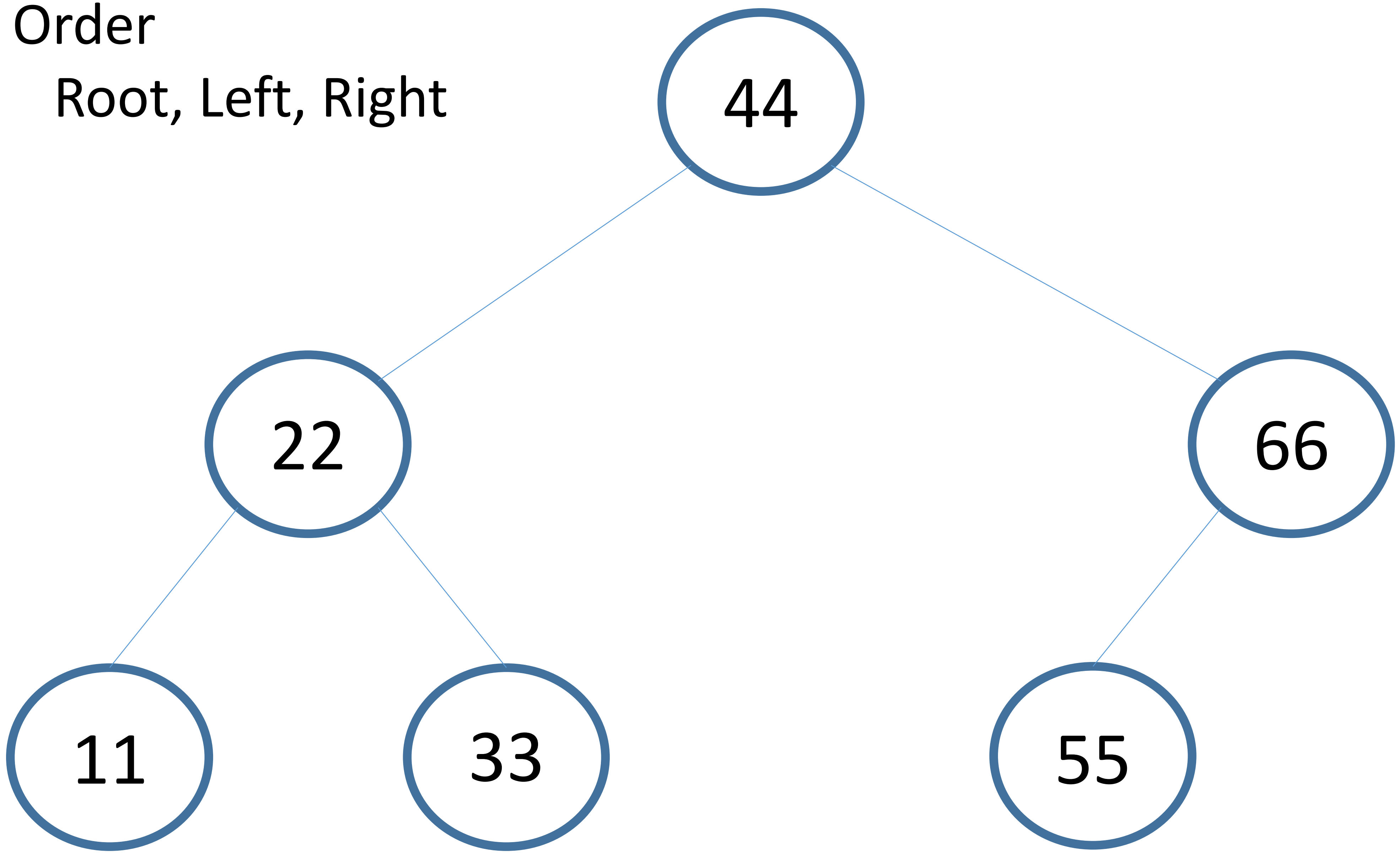
Left, Root, Right



```
Node4-11
Node2-22
Node5-33
Node1-44
Node6-55
Node3-66
```

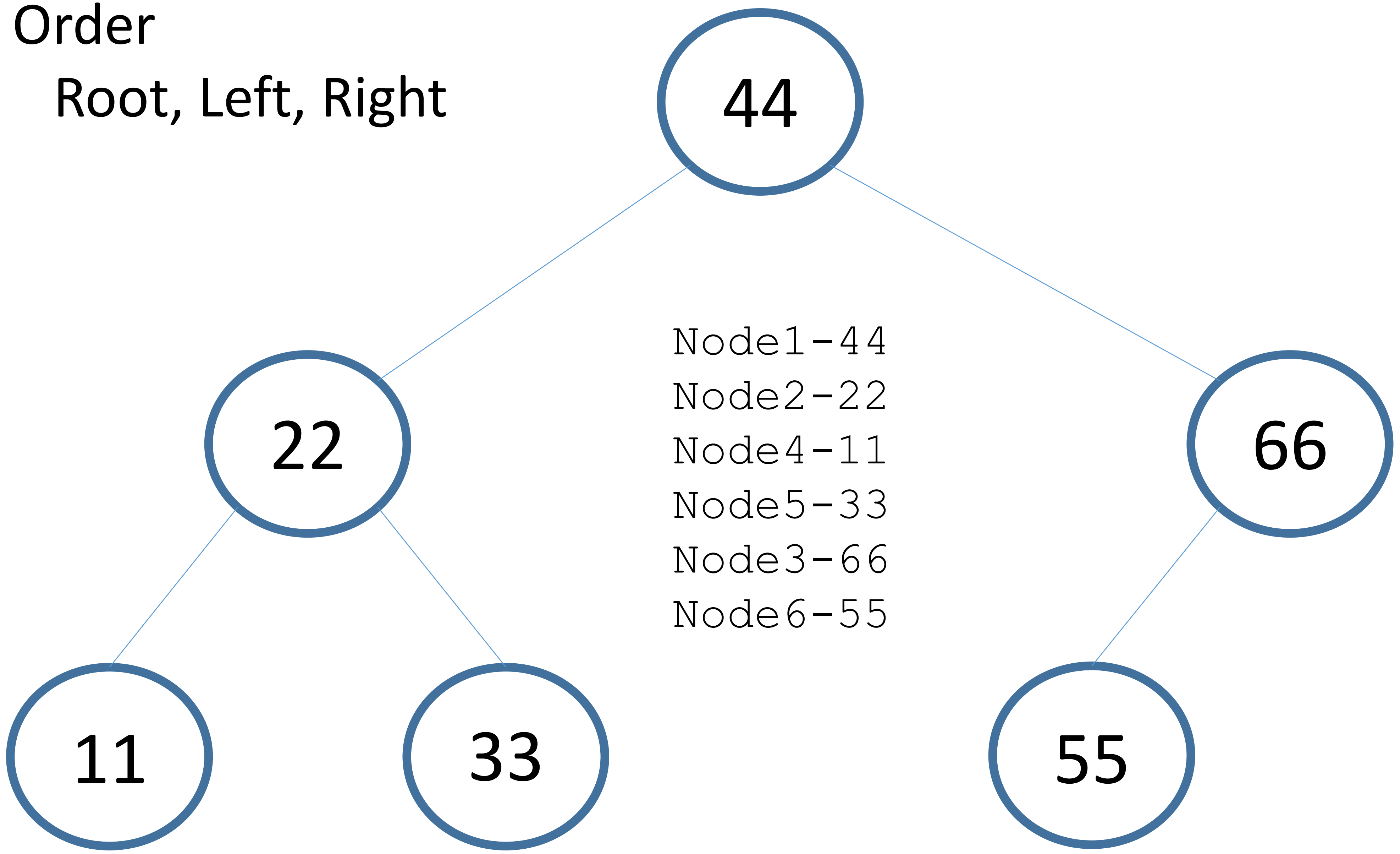
Pre Order

Root, Left, Right



Pre Order

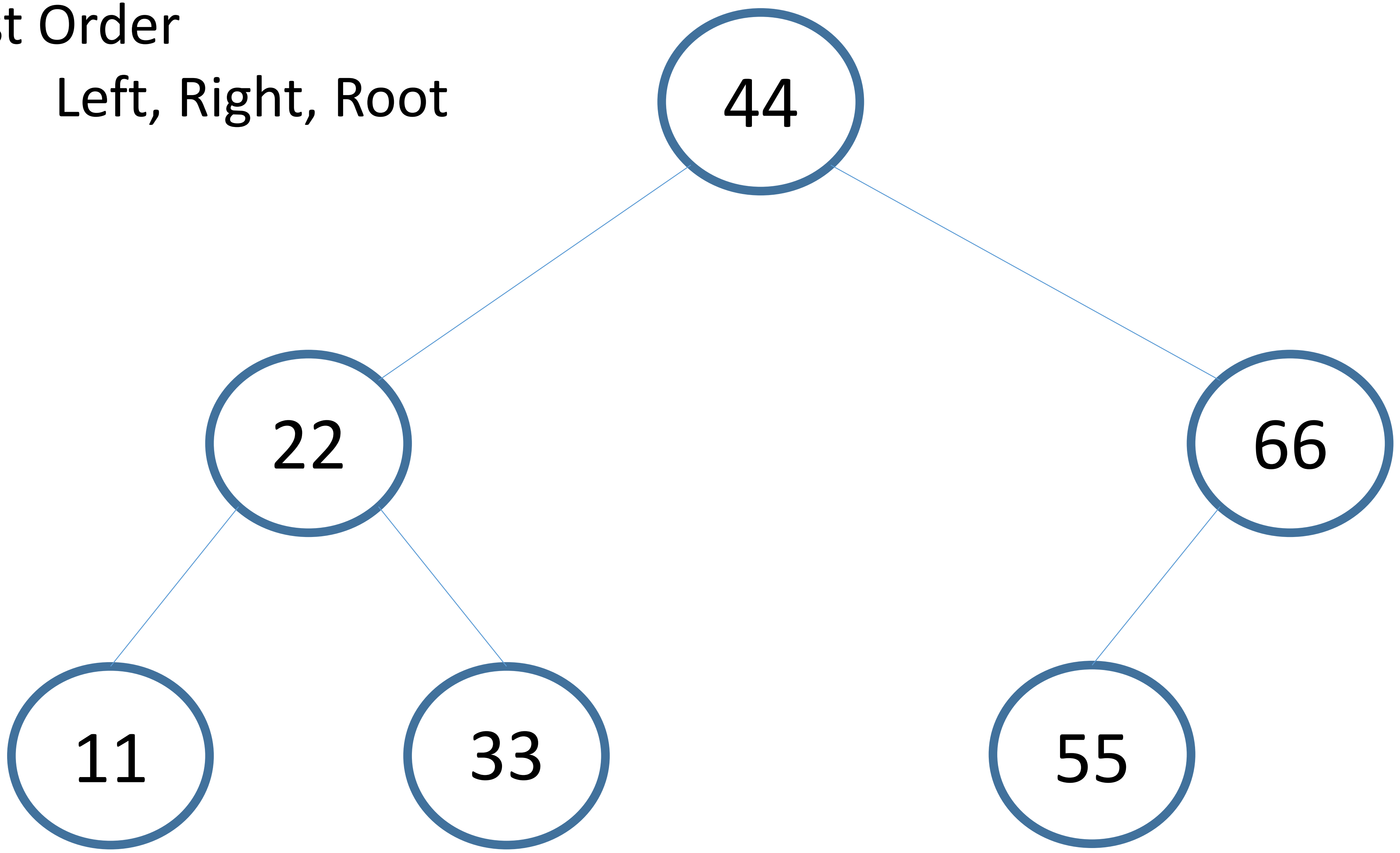
Root, Left, Right



```
Node1-44
Node2-22
Node4-11
Node5-33
Node3-66
Node6-55
```

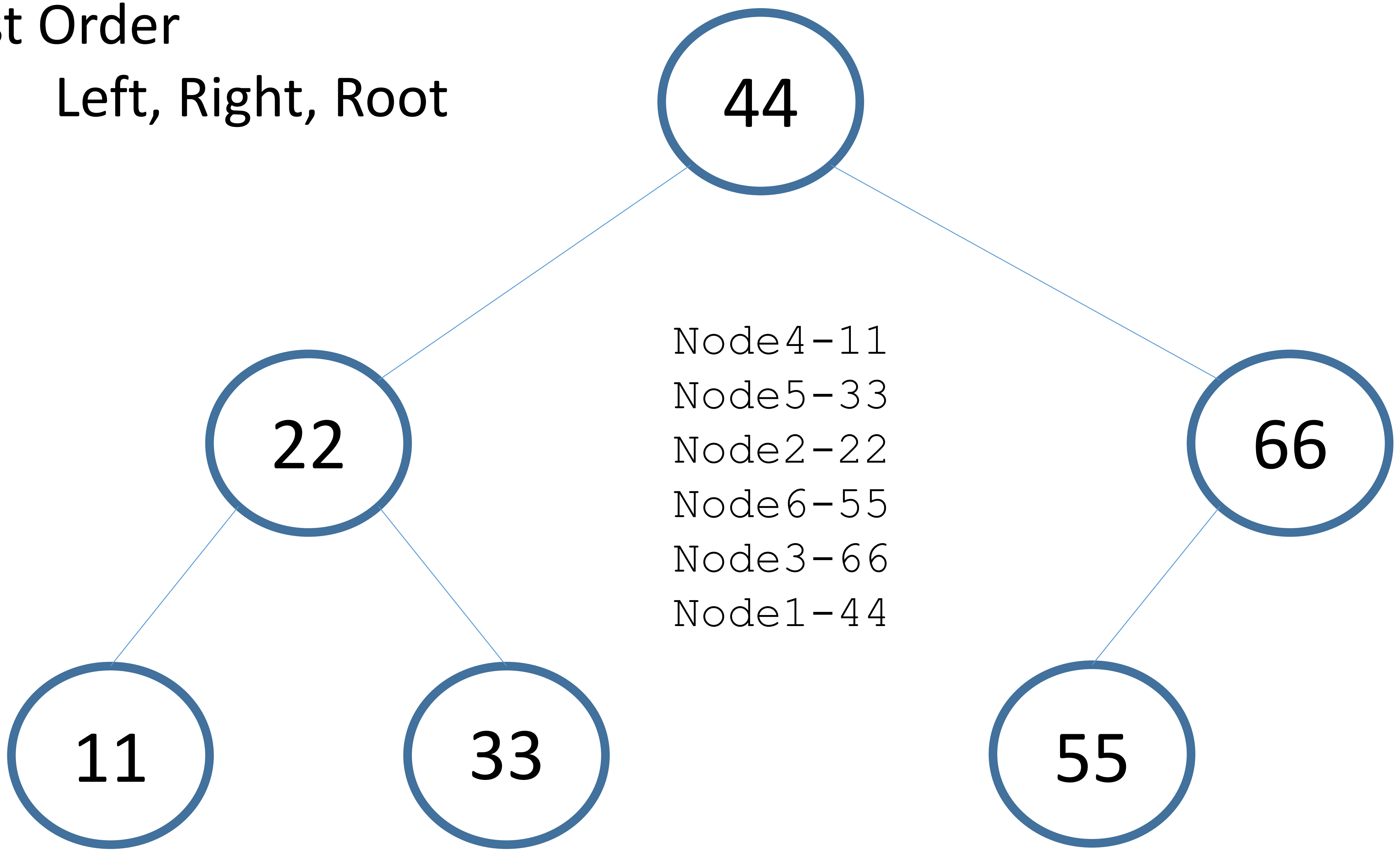
Post Order

Left, Right, Root



Post Order

Left, Right, Root



```
Node4-11
Node5-33
Node2-22
Node6-55
Node3-66
Node1-44
```

ICQ 12

I will give you a binary tree and you will need to write the

Preorder Traversal

Postorder Traversal

Inorder Traversal

Prepare for this ICQ by using the binary tree examples here in the slides. See if you can properly fill in the 3 traversal paths.

You won't see the binary tree for this ICQ until class time. It is important to learn the technique and not just memorize a traversal. The Final Exam will have a tree you have never seen; therefore, you need to learn the technique we discussed in class.