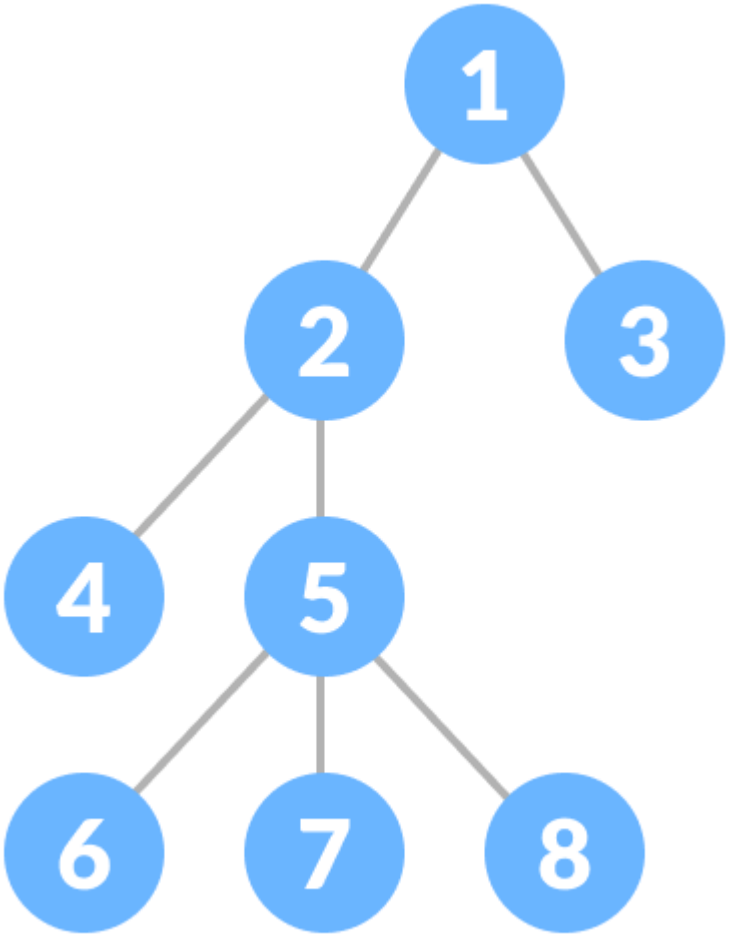
Tree Data Structure

A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges.



Why Tree Data Structure?

Other data structures such as arrays, linked list, stack, and queue are linear data structures that store data sequentially. In order to perform any operation in a linear data structure, the time complexity increases with the

increase in the data size. But, it is not acceptable in today's computational world.

Different tree data structures allow quicker and easier access to the data as it is a non-linear data structure.

Tree Terminologies

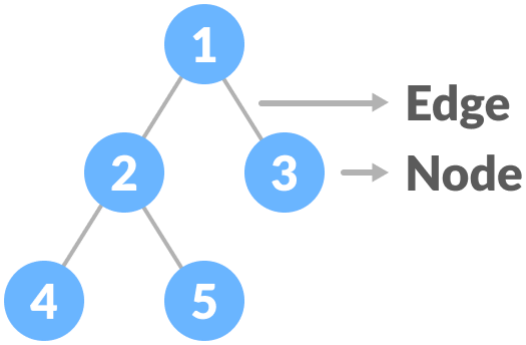
Node

A node is an entity that contains a key or value and pointers to its child nodes.

The last nodes of each path are called **leaf nodes or external nodes** that do not contain a link/pointer to child nodes.
The node having at least a child node is called an **internal node**.

Edge

It is the link between any two nodes.



Nodes and edges of a tree

Root

It is the topmost node of a tree.

Height of a Node

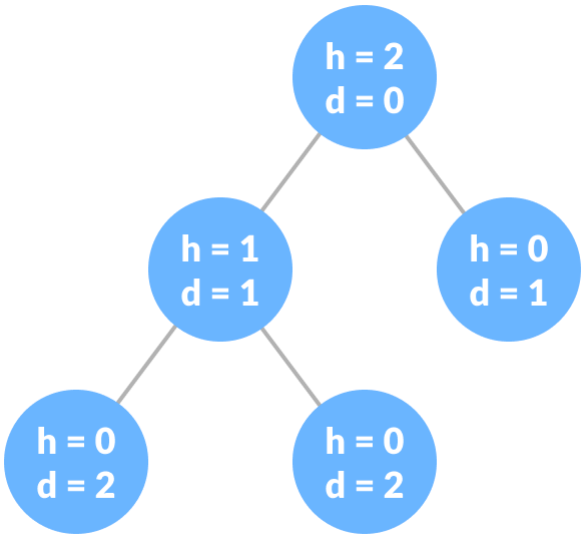
The height of a node is the number of edges from the node to the deepest leaf (ie. the longest path from the node to a leaf node).

Depth of a Node

The depth of a node is the number of edges from the root to the node.

Height of a Tree

The height of a Tree is the height of the root node or the depth of the deepest node.



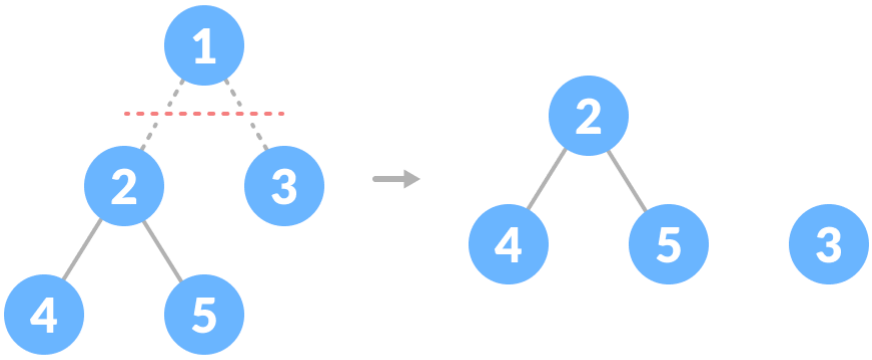
Height and depth of each node in a tree

Degree of a Node

The degree of a node is the total number of branches of that node.

Forest

A collection of disjoint trees is called a forest.



Creating forest from a tree

You can create a forest by cutting the root of a tree.

Types of Tree

1. Binary Tree
2. Binary Search Tree
3. AVL Tree
4. B-Tree

Tree Traversal

In order to perform any operation on a tree, you need to reach to the specific node. The tree traversal algorithm helps in visiting a required node in the tree.

To learn more, please visit tree traversal.

Tree Applications

- Binary Search Trees(BSTs) are used to quickly check whether an element is present in a set or not.
- Heap is a kind of tree that is used for heap sort.
- A modified version of a tree called Tries is used in modern routers to store routing information.
- Most popular databases use B-Trees and T-Trees, which are variants of the tree structure we learned above to store their data
- Compilers use a syntax tree to validate the syntax of every program you write.

Traversal에 대한 내용

https://www.geeksforgeeks.org/postorder-traversal-of-binary-tree/

https://www.geeksforgeeks.org/preorder-traversal-of-binary-tree/?ref=ml_lbp

https://www.geeksforgeeks.org/inorder-traversal-of-binary-tree/?ref=ml_lbp

무조건 필독***

Tree in C

```c
#include <stdio.h>
#include <stdlib.h>

// 이진 트리 노드 구조체 정의
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};

// 새로운 노드 생성 함수
struct TreeNode* createNode(int data) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
    if (!newNode) {
        printf("Memory allocation error\n");
        return NULL;
    }
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
```

Insert Function with Binary Search Tree

```c
struct TreeNode* insertNode(struct TreeNode* root, int data) {
    if (root == NULL) {
        root = createNode(data);
        return root;
    }

    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else if (data > root->data) {
        root->right = insertNode(root->right, data);
    }
```

```
    return root;
}
```

Preorder Traversal

```
void preorderTraversal(struct TreeNode* root) {
    if (root == NULL) return;
    printf("%d -> ", root->data);
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}
```

Inorder Traversal

```
void inorderTraversal(struct TreeNode* root) {
    if (root == NULL) return;
    inorderTraversal(root->left);
    printf("%d -> ", root->data);
    inorderTraversal(root->right);
}
```

Postorder Traversal

```
void postorderTraversal(struct TreeNode* root) {
    if (root == NULL) return;
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    printf("%d -> ", root->data);
}
```

Main function

```c
int main() {
    struct TreeNode* root = NULL;

    // 노드 삽입
    root = insertNode(root, 50);
    root = insertNode(root, 30);
    root = insertNode(root, 70);
    root = insertNode(root, 20);
    root = insertNode(root, 40);
    root = insertNode(root, 60);
    root = insertNode(root, 80);

    // 전위 순회
    printf("Preorder Traversal: ");
    preorderTraversal(root);
    printf("\n");

    // 중위 순회
    printf("Inorder Traversal: ");
    inorderTraversal(root);
    printf("\n");

    // 후위 순회
    printf("Postorder Traversal: ");
    postorderTraversal(root);
    printf("\n");

    return 0;
}
```