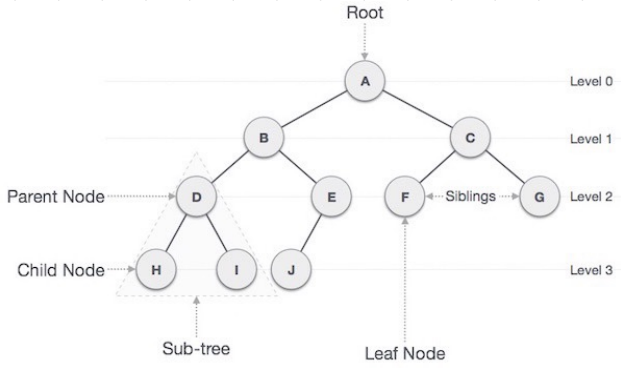
〈트리 1〉

트리란?

트리 구조란 그래프의 일종으로, 여러 노드가 한 노드를 가리킬 수 없는 구조이다. 회로가 없고, 서로 다른 두 노드를 잇는 길이 하나뿐인 그래프를 트리라고 부른다.

트리에서 최상위 노드를 루트 노드(root node)라고 한다. 또한 노드 A가 노드 B를 가리킬 때 A를 B의 부모 노드(Parent node), B를 A의 자식 노드(Child node)라고 한다. 자식 노드가 없는 노드를 잎 노드(leaf node)라고 한다. 잎 노드가 아닌 노드를 내부 노드(internal node)라고 한다.

트리에서 사용하는 용어



Root Node : 트리에서 최상위에 존재하는 노드.(A노드)

Node : 트리의 구성요소에 해당하는 요소들.(A ~ J 노드)

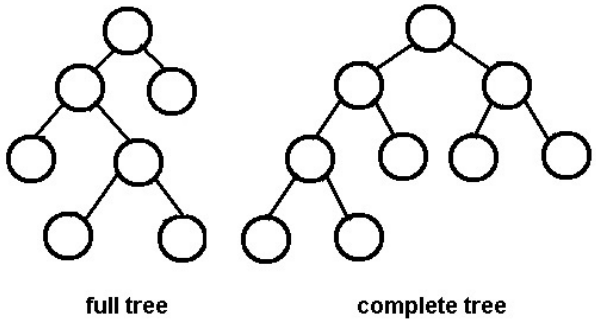
Edge : 노드와 노드를 연결하는 연결선.

Terminal Node(Leaf Node) : 밑으로 또 다른 노드가 연결되어 있지 않은 노드.(H, I, J, F, G 노드)

Sub-Tree : 큰 트리(전체)에 속하는 작은 트리.

Level : 트리에서 각 층별로 숫자를 매긴 것.

Height : 트리의 최고 레벨.(3)



이진 트리 : "이진트리가 되려면, 루트 노드를 중심으로 둘로 나뉘는 두 개의 서브트리도 이진 트리이어야 하고, 그 서브 트리의 모든 서브 트리도 이진 트리이어야 한다."

포화 이진 트리(Full Binary Tree) : 모든 레벨이 꽉 찬 이진 트리.

완전 이진 트리(Complete Binary Tree) : 포화 이진 트리처럼 모든 레벨이 꽉 찬 상태는 아니지만, 차곡차곡 빈 틈 없이 노드가 채워진 이진 트리.

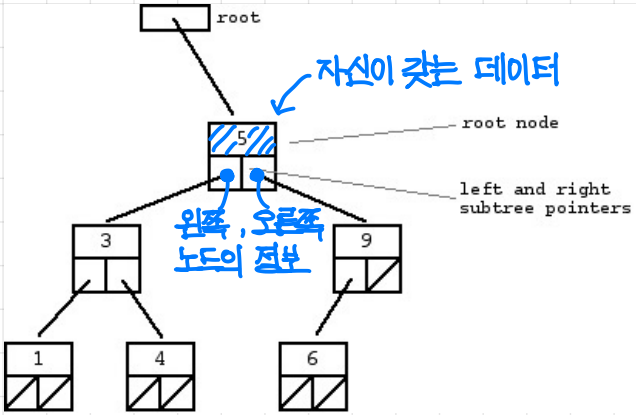
트리의 구현 방법

구현하기에 앞서

트리 또한 다른 자료 구조와 마찬가지로 배열과 연결 리스트 모두 사용하여 구현이 가능하다. 하지만 트리를 구현한다는 것은 너무나도 포괄적인 표현이다. 트리의 모양은 자기가 만들고 싶은 대로 만들 수가 있어, 모든 트리를 표현할 수 있는 코드를 구현하기는 어렵다.

구현

일단은 트리의 대표격인 이진트리를 (자식 노드가 2개 이하)를 구현해보자.



연결 리스트 기반의 트리 구현 방식은 하나의 노드에 왼쪽 자식, 오른쪽 자식의 정보를 담는 변수와 자신이 가지는 데이터만 있으면 된다.

```
typedef struct node
{
        int data;
        struct node* left;
        struct node* right;
}node;

node* root;
```

노드 정의 : 데이터와 왼쪽, 오른쪽 포인터가 있다.

```
node* insert(node* root,int data)
{
        if(root == NULL)
        {
                root = (node*)malloc(sizeof(node));
                root->right = root->left = NULL;
                root->data = data;
                return root;
        }
        else
        {
                if(data < root->data)
                        root->left = insert(root->left,data);
                else
                        root->right = insert(root->right,data);
        }
        return root;
}
```

노드 추가 함수

```
node* delete(node* root,int data)
{
        node *tmproot = NULL;

        if(root==NULL)
                return NULL;
        if(data < root->data)
                root->left = delete(root->left,data);
        else if(data > root->data)
                root->right = delete(root->right,data);
        else
        {
                if(root -> left!=NULL && root->right != NULL)
                {
                        tmproot = fMin(root->right);
                        root->data = tmproot->data;
                        root->right = delete(root->right,tmproot->data);
                }
                else
                {
                        tmproot = (root->left == NULL) ? root->right : root->left;
                        free(root);
                        return tmproot;
                }
        }
        return root;
}
```

노드 제거 함수

순회

```
void preorderPrint(node* root)
{
        if(root==NULL)
                return;
        printf("%d ",root->data);
        print(root->left);
        print(root->right);
}
```

전위 순회 :

뿌리(root) 부터 먼저 방문.

0 → 1 → 3 → 7 → 8 → 4 → 9 → 10 → 2 → 5 → 11 → 6

```
void inorderPrint(node* root)
{
        if(root==NULL)
                return;
        print(root->left);
        printf("%d ",root->data);
        print(root->right);
}
```

중위 순회 :

왼쪽 하위 트리 방문 후 뿌리(root) 방문.

7 → 3 → 8 → 1 → 9 → 4 → 10 → 0 → 11 → 5 → 2 → 6

```
void postorderPrint(node* root)
{
        if(root==NULL)
                return;
        print(root->left);
        print(root->right);
        printf("%d ",root->data);
}
```

후위 순회 :

하위 트리 모두 방문 후 뿌리(root) 방문.

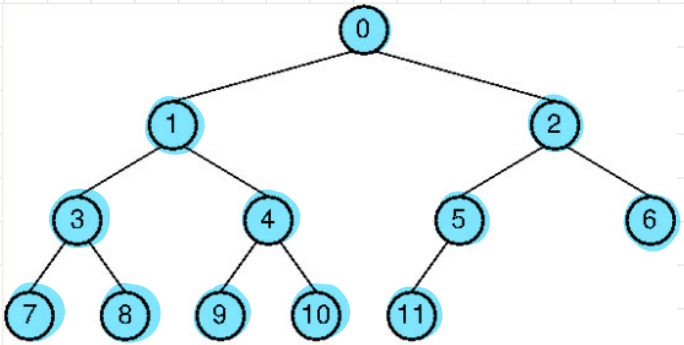
7 → 8 → 3 → 9 → 10 → 4 → 1 → 11 → 5 → 6 → 2 → 0

```
void printLevelOrder(node* root)
{
        int front = 0, rear = 0;
        node** queue = (node**)malloc(sizeof(node*)*Q_size);

    node* temp_node = root;

        while (temp_node)
        {
                printf("%d ", temp_node->data);

                if (temp_node->left)
                        insertQ(queue, &front, &rear, temp_node->left);
                if (temp_node->right)
                        insertQ(queue, &front, &rear, temp_node->right);
                temp_node = deQ(queue, &front);
        }
}
```

레벨 순회 :

위 쪽 노드들 부터 아래 방향으로 차례로 방문.

0 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 11



그래프와 트리의 차이

	그래프	트리
정의 :	노드와 그 노드를 연결하는 간선을 하나로 모아 놓은 자료 구조.	그래프의 한 종류. DAG(Directed Acyclic Graph)의 한 종류.
방향성 :	방향 그래프(Directed), 무방향 그래프(Undirected) 모두 존재.	방향 그래프(Directed Graph)
사이클 :	사이클(Cycle) 가능. 자체 간선(Self-loop) 가능. 순환 그래프(Cyclic), 비순환 그래프(Acyclic) 모두 존재.	사이클(Cycle) 불가능. 자체 간선(Self-loop) 불가능. 비순환 그래프(Acyclic Graph).
루트 노드 :	루트 노드의 개념 없음.	한 개의 루트 노드만이 존재. 모든 자식 노드는 한 개의 부모 노드 만을 가짐.
부모-자식 :	부모-자식의 개념이 없음.	부모-자식 관계. top-bottom 또는 bottom-top 으로 이루어짐.