1. **Terminology**
2. Root : tree 구조에서 제일 첫번째로 위치해 있는 노드(N)
3. Edge : tree에서 두 노드 사이를 연결시켜주는 선(N-1)
4. Parent : 부모 노드
5. Child : 자식 노드, 자식노드라고 해서 바로 밑의 노드 뿐만이 아닌 Parent의 밑에 있는 노드이면 다 Child라고 할 수 있다.
6. Siblings : 같은 부모노드를 지니고 있고, 같은 Level에 있는 노드들을 말한다.
7. Leaf(External Node) : tree에서 child 노드가 없는 노드들을 지칭한다.
8. Internal Nodes : child 노드가 하나라도 존재하는 노드를 의미한다.
9. Degree : 해당 노드와 바로 연결되어 있는 자식노드의 수를 뜻한다.
10. Level : 노드의 계층을 뜻한다. 다시 말해, 라인의 수를 말한다.?
11. Height : Leaf 노드에서 특정 노드까지의 edge의 수를 말함.
12. **Formulas**
13. 주어진 Node들을 이용한 Binary Tree의 수
14. Shape(Tree가 나올 수 있는 모양의 수) – 한번 그려보면 됨.

* 1개의 노드에서 나올 수 있는 모양은 1개이다.
* 2개의 노드에서 나올 수 있는 모양도 2개이다.
* 3개의 노드에서 나올 수 있는 모양은 5개이다.
* 따라서 이를 공식화 하면 이 도출 된다.

1. Permutation (각 노드에 들어가야할 value의 경우의 수)

* 2개의 노드가 있다면 2개의 value를 넣는 경우의 수는 2\*1개이다.
* 3개의 노드가 있다면, 3\*2\*1 개가 된다.
* 따라서 n개의 노드가 있다면, n! 가 될 것이다.

1. Max, Min Height or Node 구하기
2. Height가 주어졌을 경우

* Min Nodes : n=h+1
* Max Nodes : n=2h+1-1

1. Node가 주어졌을 경우

* Min Height : h=
* Max Height : h=n-1

1. Degree 구하기

* Deg(0) = external node
* Deg(0) = Deg(2)+1

1. **Strict Binary Tree**
2. Strict Binary Tree : Degree 가 2 or 0만을 가지고 있는 tree.
3. Height 가 주어졌을 때

* Min Nodes : n=2h+1
* Max Nodes : n=2h+1-1

1. Node 가 주어졌을 때

* Min Height : h=
* Max Height :

1. Internal node와 External node의 관계

* E=I+1

1. **N-ary Trees(N : degree of tree)**
2. 한가지 주의 해야 할 점은 예를 들어, 4-ary Tree의 경우 나올수 있는 자식의 수는 {0,1,2,3,4}가 되는 데, 반드시 자식의 수가 4개가 있어야 4-ary Tree가 아니다. 즉, 4미만의 자식의 수를 가지고 있는 Tree도 4-ary Tree가 될 수 있다.
3. Strict n-ary Tree : n개 자식 또는 0개의 자식만을 가지는 Tree
4. **Analysis of n-Ary Trees**
5. If height is given

* Min Nodes : n=mh+1 (m : degree of tree)
* Max Nodes : (등비 수열)

1. If number of node is given

* Min Height :
* Max Hight :

1. Internal Node and External Node in strict n-ary Tree

* E=(m-1)I+1

1. **Representation of Binary Tree(Array Rep, Linked Rep)**
2. Array Representation



* 위와 같이 위, 왼쪽을 기준으로 순서대로 적힌 Tree가 있다고 가정하자. 배열을 만들어 A~G까지 순서대로 넣게 되면 A,B,C,D,E,F,G 와 같이 넣어 지게 될 것이다 이를 분석해보면,

|  |  |  |  |
| --- | --- | --- | --- |
| Element | index | L.Child | R.Child |
| A | 1 | 2 | 3 |
| B | 2 | 4 | 5 |
| C | 3 | 6 | 7 |
|  | i | 2\*i | 2\*i+1 |

* Parent = i/2

1. Linked Representation of Binary Tree

Node : L.Child – data – R.Child

* Null pointer = node+1
* E=I+1

1. **Full vs Complete Binary Tree**
2. **Full Binary Tree** : A **full** binary tree (sometimes referred to as a **proper** or **plane** binary tree) is a tree in which every node has either 0 or 2 children. Another way of defining a full binary tree is a [recursive definition](https://en.wikipedia.org/wiki/Recursive_definition). A full binary tree is either:

* A single vertex
* A tree whose root node has two subtrees, both of which are full binary trees.

1. **Complete Binary Tree** : In a **complete** binary tree every level, *except possibly the last*, is completely filled, and all nodes in the last level are as far left as possible. It can have between 1 and 2*h* nodes at the last level *h*. An alternative definition is a perfect tree whose rightmost leaves (perhaps all) have been removed.

* Udemy의 설명에는 Complete BT는 배열에 나열했을 때, 꽉 채워져 있는 것을 의미한다. 즉, 왼쪽부터 차례대로 다 채워져 있는 tree를 의미한다.

1. **Strict vs Complete Binary Tree**
2. Strict의 경우 배열의 중간이 비어져 있어도, 0 or 2개의 자식노드만 있다면 Strict Binary Tree가 성립되지만, Complete의 경우 무조건 왼쪽부터 채워져 있어야 배열에 빈 부분이 없이 들어가기 때문에 이 부분에서 다르다고 할 수 있다.
3. **Binary Tree Traversals**
4. Preorder : visit(node), Preorder(left subtree), Preorder(right subtree)
5. Inorder : Inorder(left), visit(node), Inorder(right)
6. Postorder : Postorder(left), Postorder(right), visit(node)
7. Level\_Order : Level by Level
8. **Binary Tree Traversal Easy Method 1**
9. Preorder



* 이 방법으로 하면 preorder의 순을 쉽게 알 수 있다. Inorder, postorder도 마찬가지이다.

1. Inorder



1. Postorder



* Preorder를 오른쪽으로 그리는 것이라고 이해하면 된다.

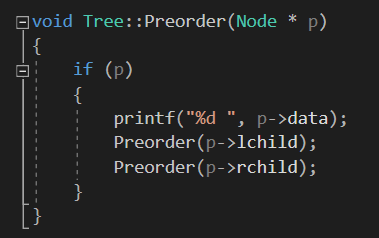
1. **Binary Tree Traversal Easy Method 2.**

* 한 붓그리기로 왼쪽바깥에서부터 Tree의 구조를 한바퀴 도는 방법인데, 여기서 pre의 경우 노드의 왼쪽에 하나의 선을 그려놓고. Inorder의 경우 노드의 밑에 선을 그려놓고, Postorder의 경우 노드의 오른쪽에 선을 그려놓은뒤, 한붓그리는 선에 만나는 순서대로 순서를 배열하는 방법이다.
* 1의 방법이 더 좋은 것 같다.

1. **Binary Tree Treversal Easy Method 3.**

* 2 방법과 유사한데,,, 쓰지 말자. 몰라도 됨.

**287. Preorder Tree Traversal**



289. Iterative Preorder

299. BST intro (Binary Search Tree)

- 찾고자 하는 수가 있을 때 현재 노드를 기준으로 작으면 왼쪽의 노드로 크면 오른쪽의 노드로 이동하면서 원하는 수를 찾는 방법이다.

- Property

\* No Duplicates

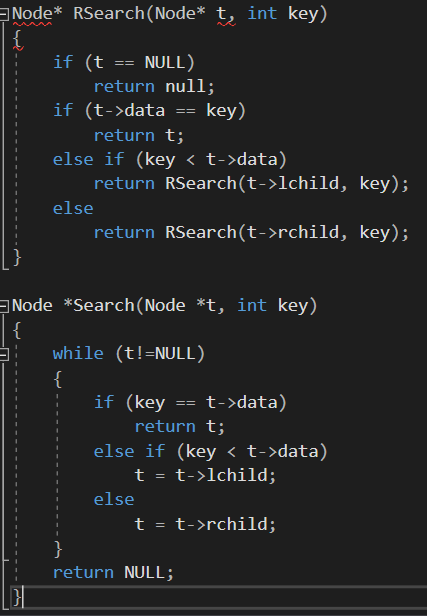
\* Inorder gives sorted order

\* Number of BST for ‘n’ nodes

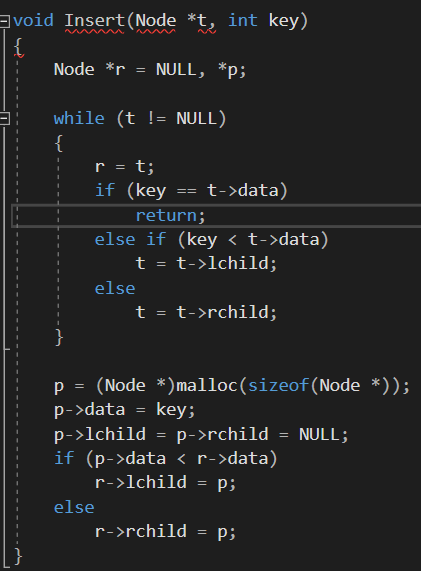
\* Number of Shape :

300. Searching in BST

- O(h), log n <= h <= n



* Insert



307. Generating BST from preorder

1) Preorder + Inorder

2) Postorder + Inorder

310. Introduction to AVL trees

1) AVL(Adelson-Velsky and Landis)

2) Balance Factor = height of left subtree – height of right subtree

- Bf = hl – hr = {-1, 0, 1}

- /bf/ = /hl – hr/ <=1 balanced

- /bf/ = /hl-hr/ >1 imbalanced

311. Inserting in AVL with Rotations

1) LL – Rotation : Single Rotation

2) RR – Rotation : Single Rotation

3) LR – Rotation : Double Rotation

4) RL – Rotation : Double Rotation

* L과 R의 기준 : 값을 inserting 했을 때, root에서 inserting value로 가는길에 있는 2개의 자식노드를 기준으로 LL,RR,LR,RL을 확인한다. 예를 들어, value가 제일 왼쪽(inorder 했을 때 제일 첫번째 배열) 일 때 root노드에서 2개의 자식노드의 위치가 LL에 존재하므로 LL 이라고 함.(만약에 LLLR 에 위치에 있다고 해도, 2개의 자식노드가 기준이므로 LL이라고 한다)
* 이렇게 하는 이유는 Rotation의 기준이 Root~2개의 자식노드를 기준으로 Rotation을 하기 때문이다.