트리 자료구조

박정민

01. 트리의 개념

트리의 정의

트리는 1개 이상의 노드를 갖는 집합으로 노드들은 다음 조건을 만족한다.

1) 트리에는 **루트(root)**라고 부르는 특별한 노드가 있다.

2) 다른 노드들은 원소가 중복되지 않는 n개의 부속 트리 (subtree)

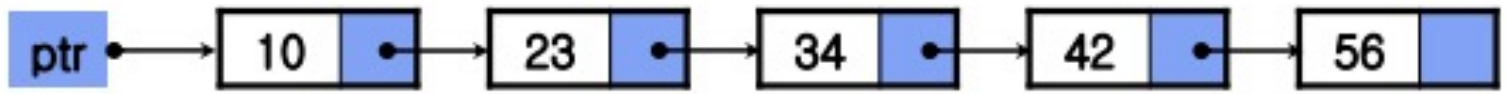
$T_1, T_2, \cdots, T_n$으로 나누어지며 T_i 각각은 루트의 부속 트리라고 부른다.

(트리는 사이클이 없는 그래프 (acyclic graph)이며 계층 구조를 이룬다.)

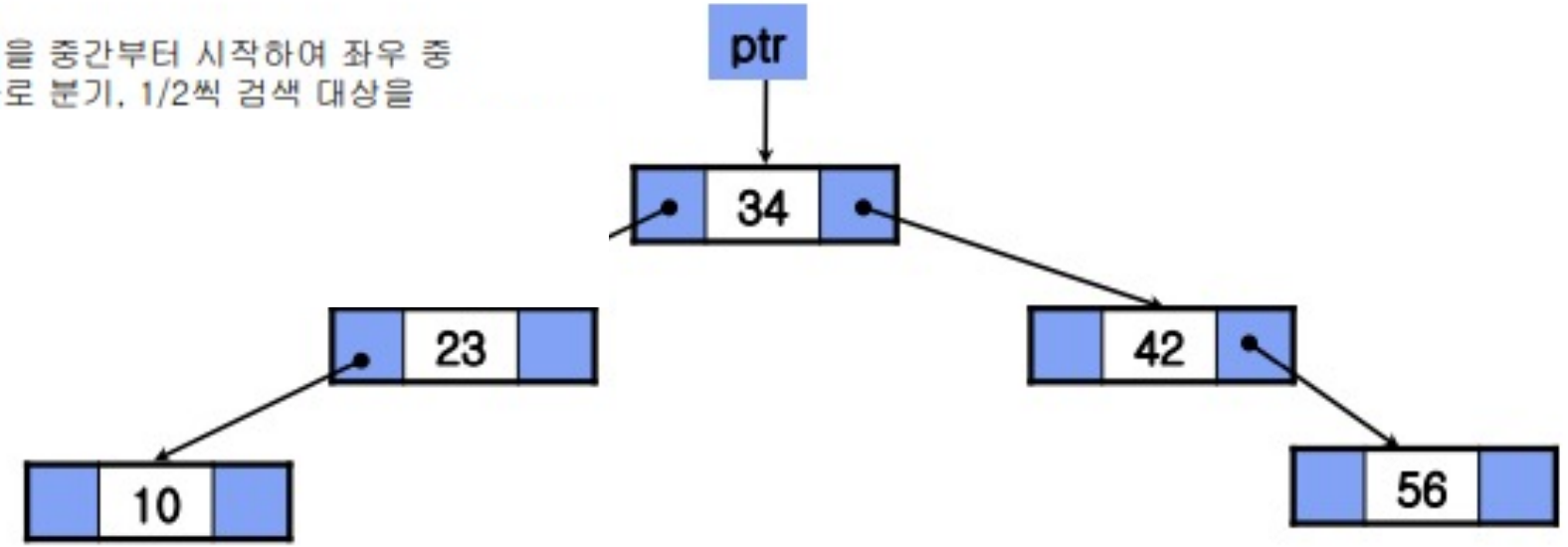
01 **트리 자료구조 필요성**

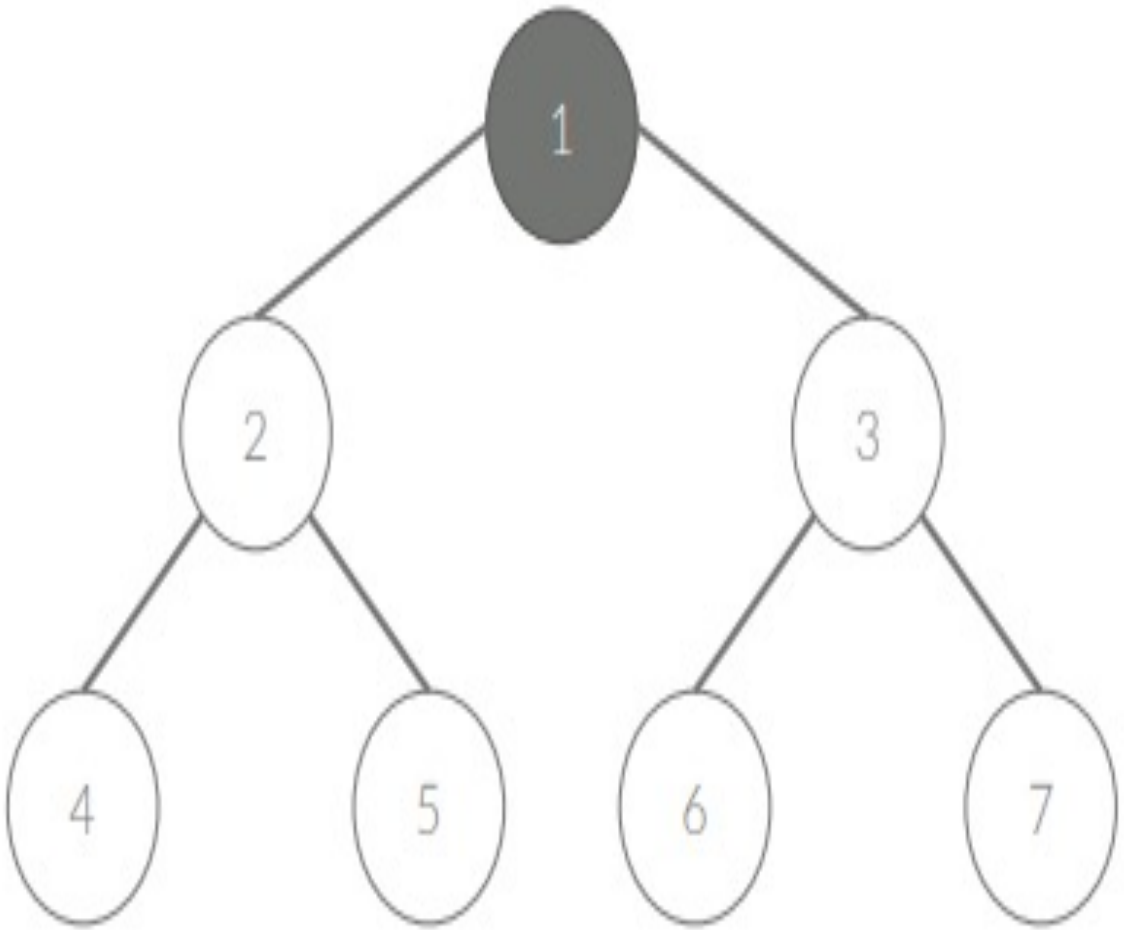
연결리스트의 삽입삭제 시 데이터를 이동하지 않는 장점을 살리자.
연결리스트의 검색시 노드의 처음부터 찾아가야하는 단점을 보완하자.

⇒ 데이터를 중간부터 찾아가는 이진검색의 장점을 이용하자.
⇒ 연결리스트의 포인터를 리스트의 중간에 두는 방법?



› 검색을 중간부터 시작하여 좌우 중 하나로 분기, 1/2씩 검색 대상을





노드: 트리의 구성요소

루트: 부모가 없는 노드

서브트리: 하나의 노드와 그 노드들의 자손으로 이루어진 노드

단말노드: 자식이 없는 노드

레벨: 트리의 각층의 번호

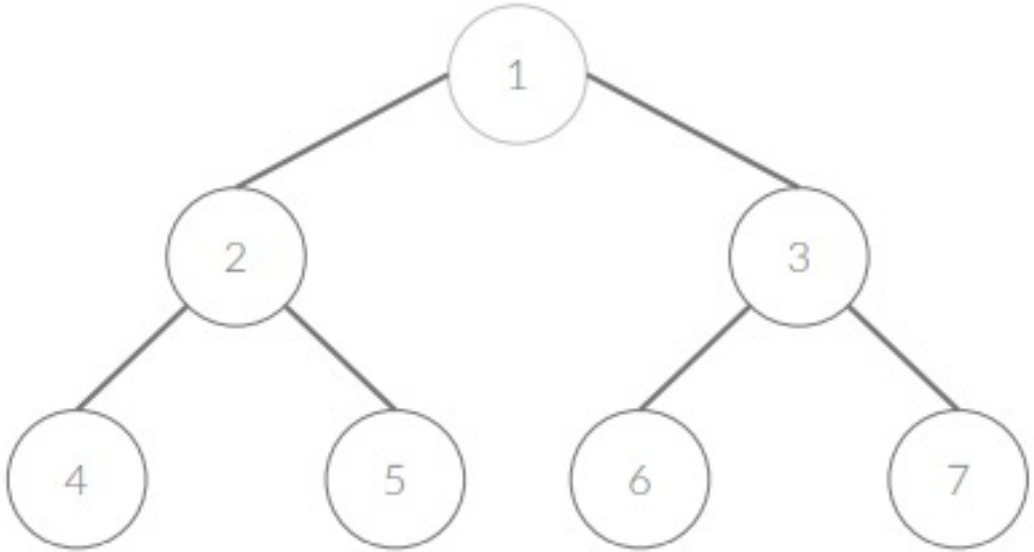
높이: 트리의 최대 레벨

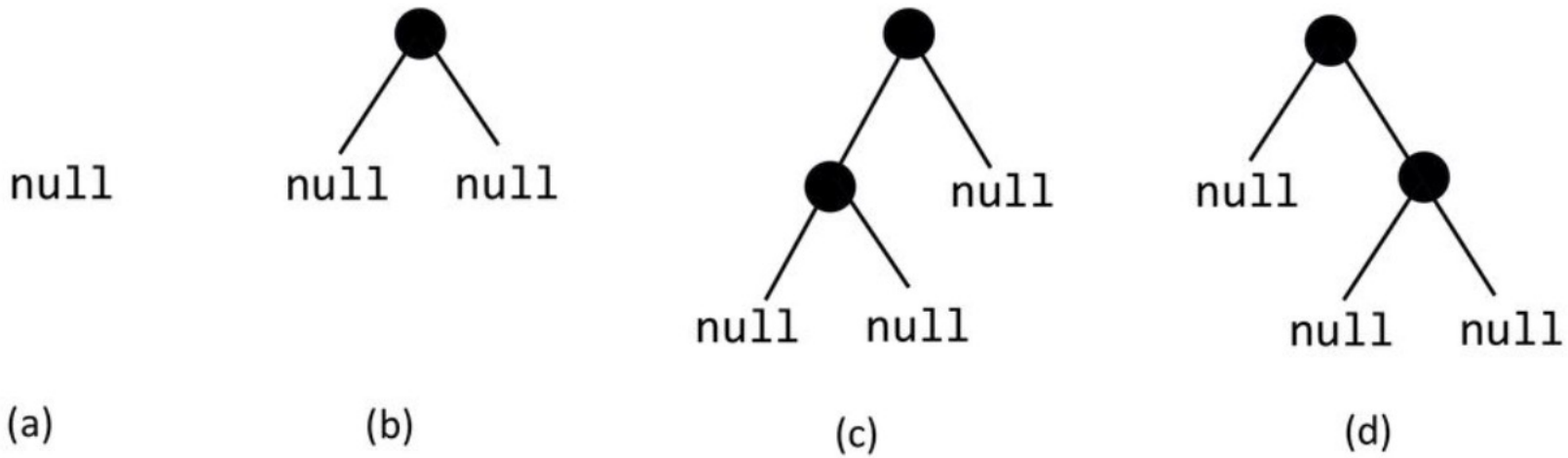
차수: 노드가 가지고있는 자식 노드의 개수

이진 트리

Binary Tree

- 자식을 최대 2개만 가지고 있는 트리



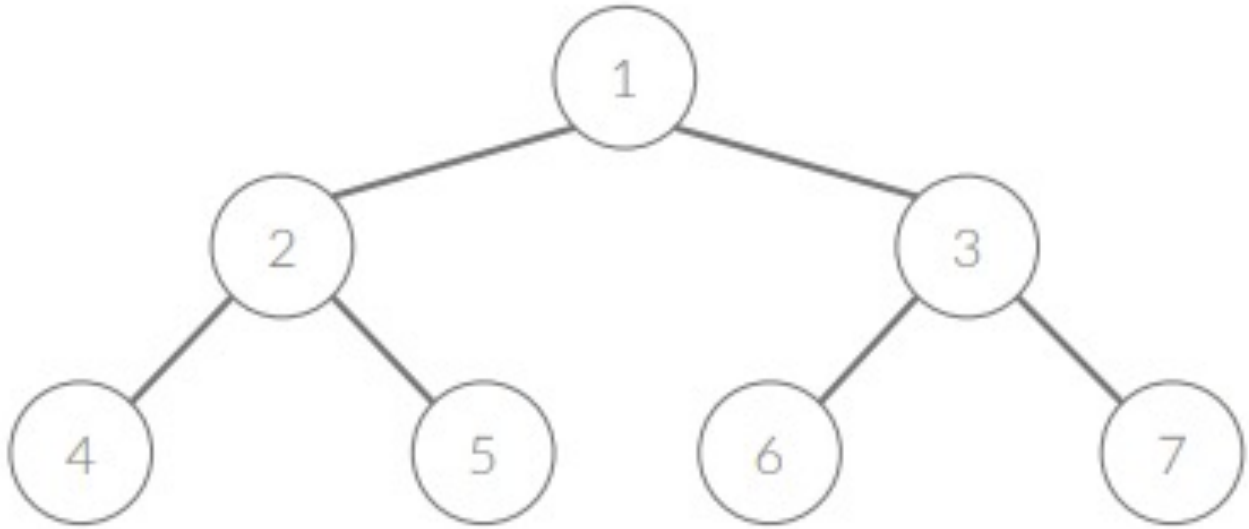


(a) (b) (c) (d)

포화 이진 트리

Perfect Binary Tree

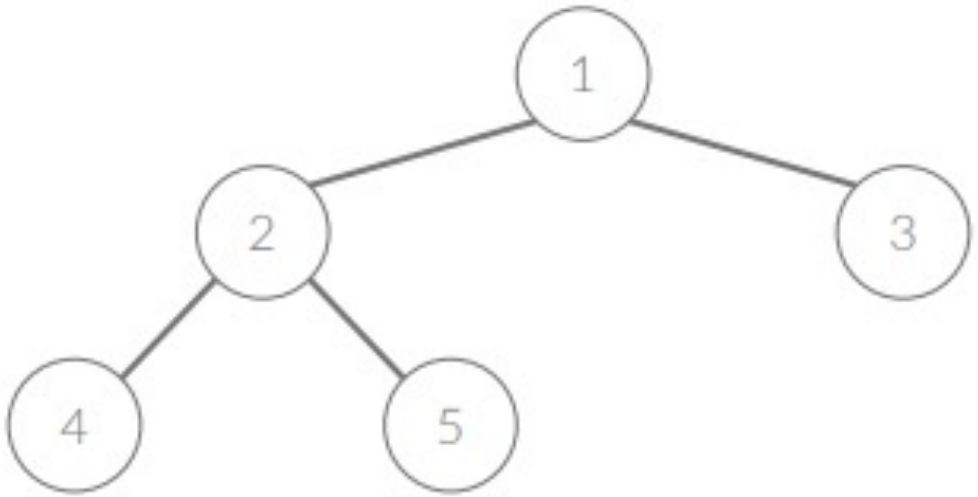
- 리프 노드를 제외한 노드의 자식의 수: 2
- 리프 노드의 자식의 수: 0
- 모든 리프 노드의 깊이가 같아야 함
- 높이가 h인 트리의 노드 개수 = 2^h-1

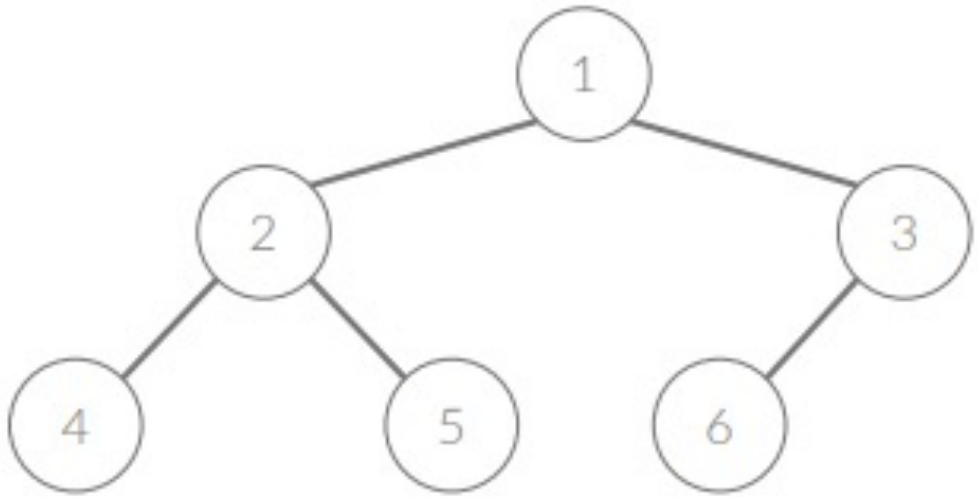


완전 이진 트리

Complete Binary Tree

- 리프 노드를 제외한 노드의 자식의 수: 2
- 리프 노드의 자식의 수: 0
- 마지막 레벨에는 노드가 일부는 없을 수도 있음
- 오른쪽에서부터 몇 개가 사라진 형태



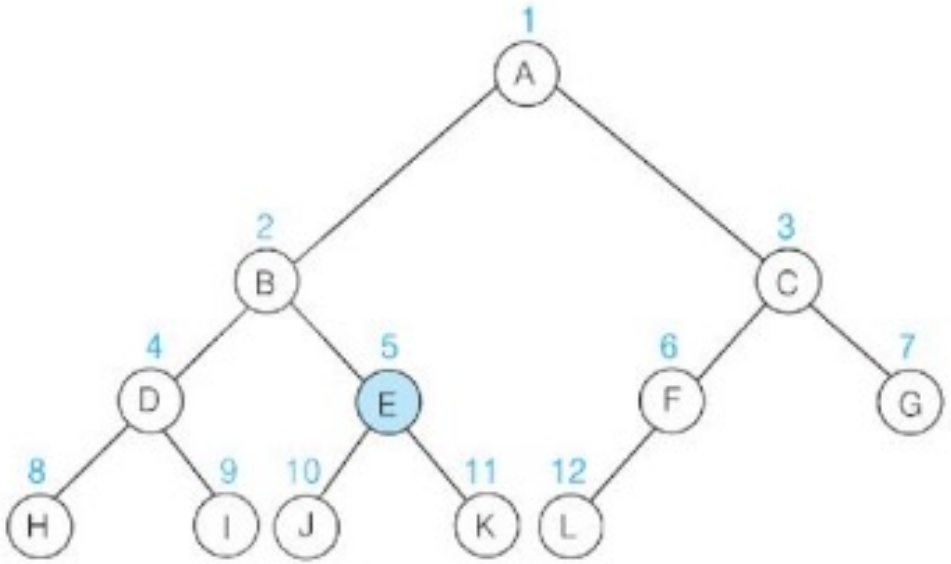


이진트리의 속성

- 레벨 k에 있는 최대 노드 수 = 2^{k-1} , k = 1, 2, 3, ⋯
- 높이가 h인 포화이진트리에 있는 노드 수 = $2^{h} - 1$
- N개의 노드를 가진 완전이진트리의 높이 = $\lceil \log_2(N+1) \rceil$

1차원 배열을 통한 이진트리의 구현

- 예) 완전 이진 트리의 1차원 배열 표현



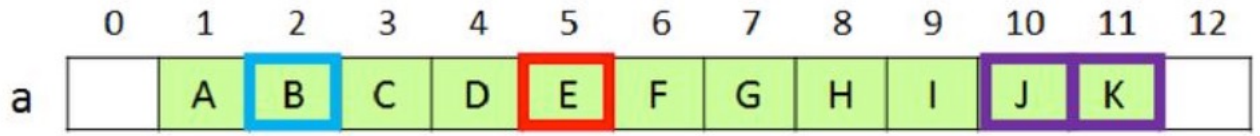
[0]	
[1]	A
[2]	B
[3]	C
[4]	D
[5]	E
[6]	F
[7]	G
[8]	H
[9]	I
[10]	J
[11]	K
[12]	L

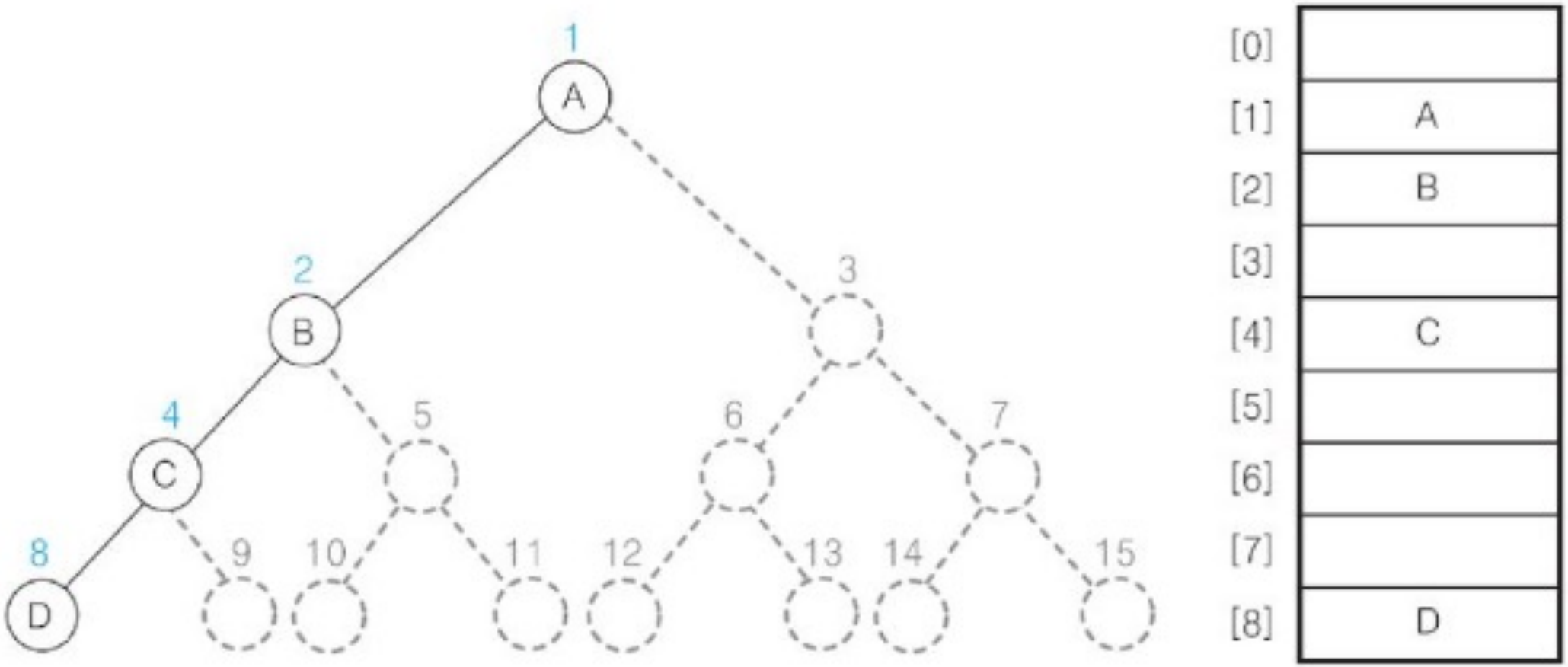
부모노드의 인덱스 = 2

왼쪽 자식노드의 인덱스 = 10

오른쪽 자식노드의 인덱스 = 11

- a[i]의 부모노드는 a[i/2]에 있다. 단, i > 1이다.
- a[i]의 왼쪽 자식노드는 a[2i]에 있다. 단, 2i ≤ N이다.
- a[i]의 오른쪽 자식노드는 a[2i+1]에 있다. 단, 2i + 1 ≤ N이다.



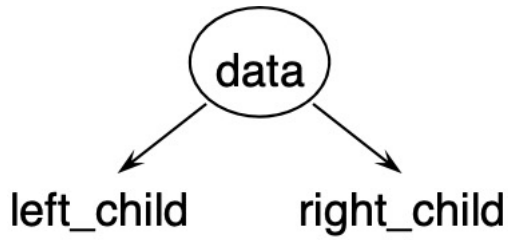

1
A
2
B
3
4
C
5
6
7
8
D
9
10
11
12
13
14
15
[0]
[1]
A
[2]
B
[3]
[4]
C
[5]
[6]
[7]
[8]
D

연결 자료구조를 이용한 이진트리 구현


```
struct Node {
    Node *left;
    Node *right;
}
```

연결 자료구조를 이용한 이진트리 구현

```
struct tnode {
            int data;
            struct tnode * left_child;
            struct tnode * right_child;
};
typedef struct tnode node;
typedef node * tree_ptr;
```

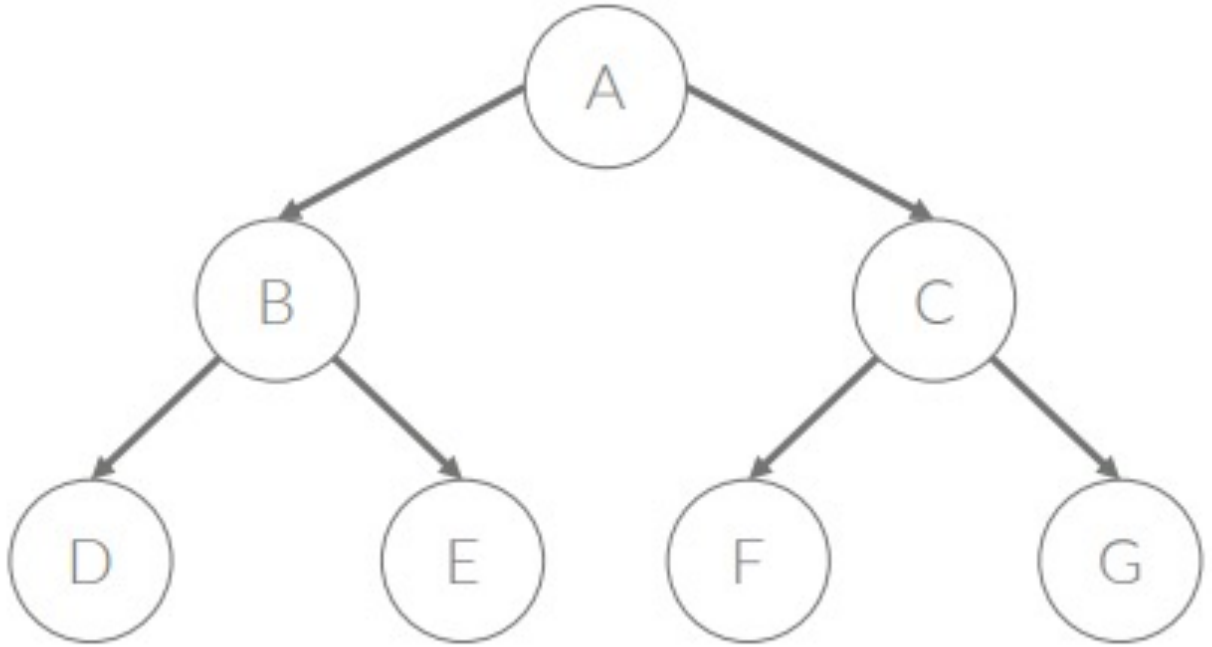


프리오더

Pre-order

- 노드 방문
- 왼쪽 자식 프리오더
- 오른쪽 자식 프리오더

```
preorder(T)
    if (T≠null) then {
        visit T.data;
        preorder(T.left);
        preorder(T.right);
    }
end preorder()
```



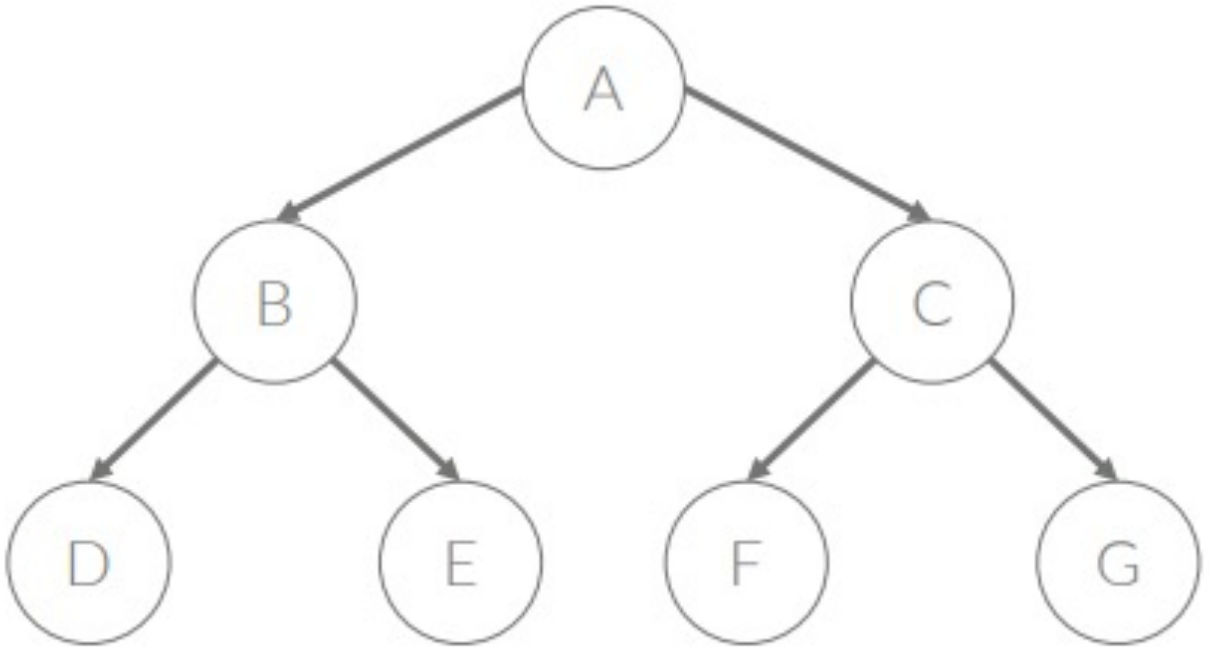
- ABDECFG

인오더

In-order

- **왼쪽 자식 인오더**
  - 왼쪽 자식 인오더
  - 노드 방문
  - 오른쪽 자식 인오더
- 노드 방문
- 오른쪽 자식 인오더

```
inorder(T)
    if (T≠null) then {
        inorder(T.left);
        visit T.data;
        inorder(T.right);
    }
end inorder()
```



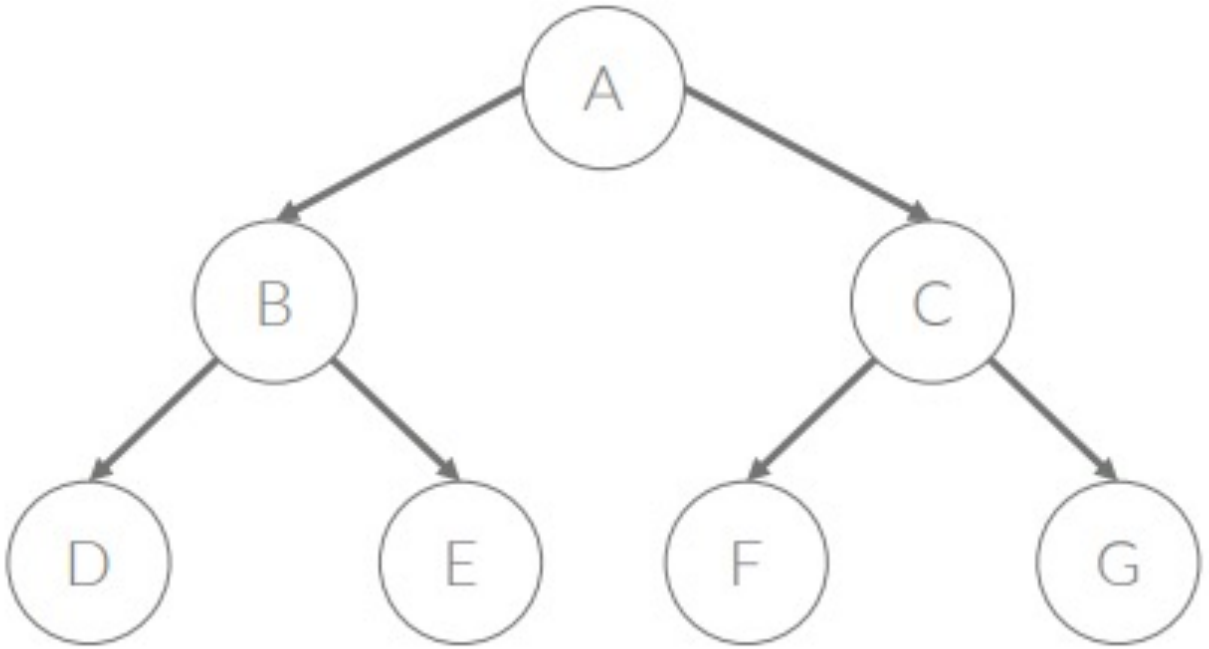
- DBEAFCG

포스트오더

Postorder

- 왼쪽 자식 포스트오더
- 오른쪽 자식 포스트오더
- 노드 방문

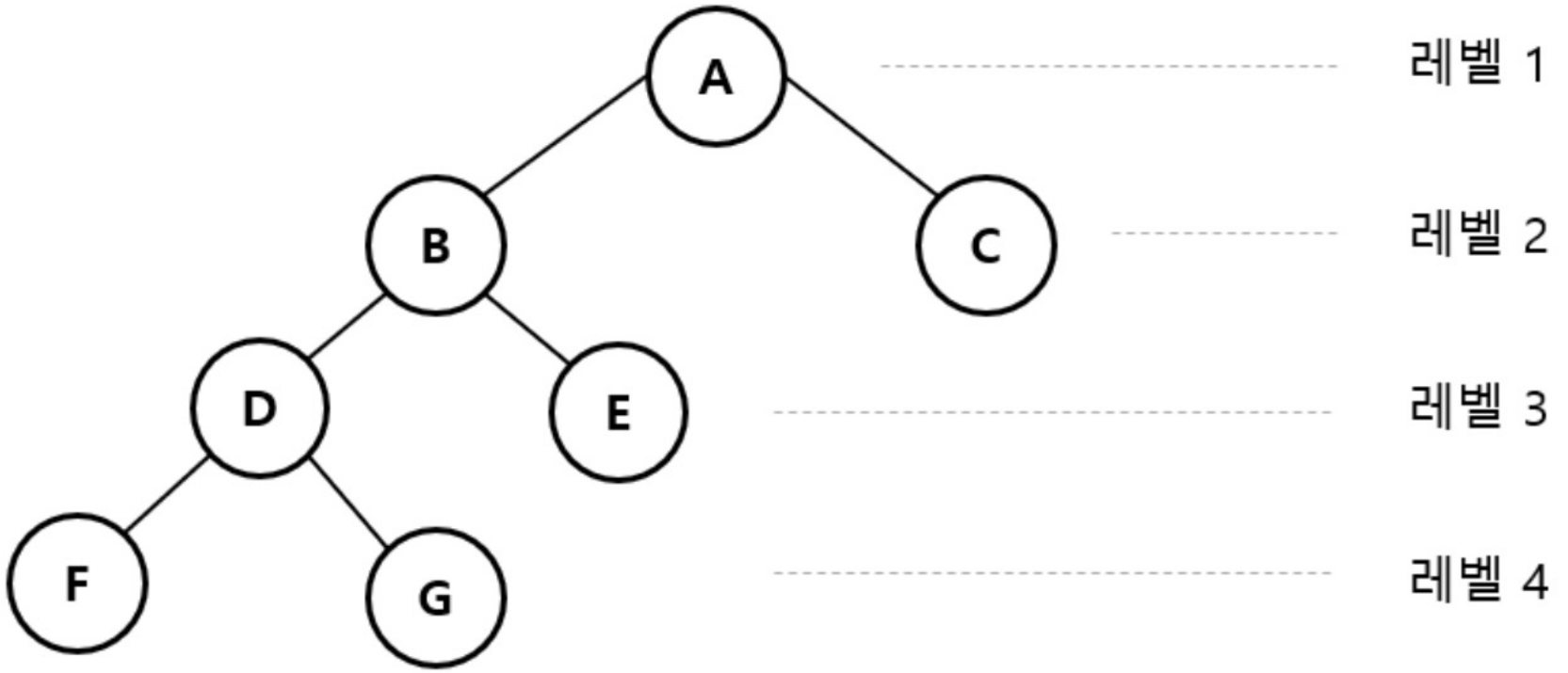
```
postorder(T)
      if (T≠null) then {
            postorder(T.left);
            postorder(T.right);
            visit T.data;
      }
end postorder()
```



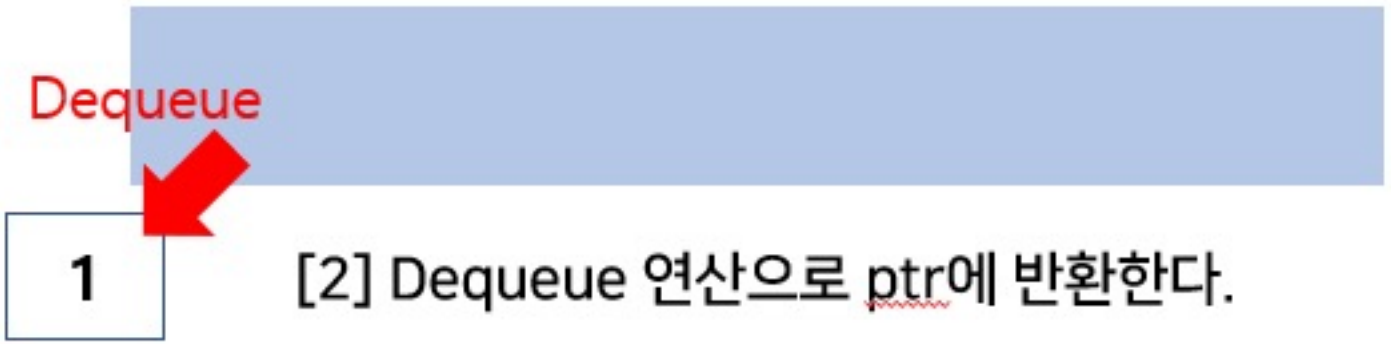
- DEBFGCA

레벨 오더

1) 루트 노드를 방문한다.
2) 루트 노드의 Left Child 를 방문한다.
3) 루트 노드의 Right Child를 방문한다.



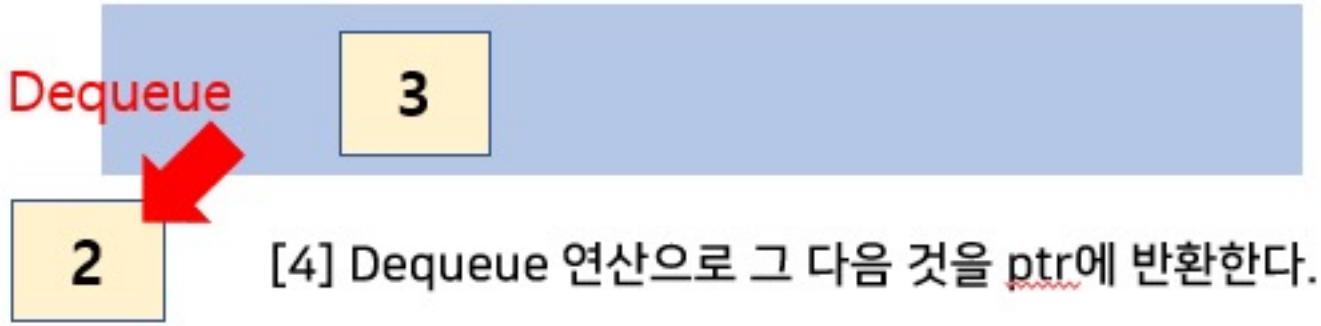
[1] 맨 먼저 큐에 루트 노드를 넣는다.



[2] Dequeue 연산으로 ptr에 반환한다.

[3] 반환한 ptr의 데이터를 출력하고, dequeue했던 1의 left, right Child를 큐에 넣는다.

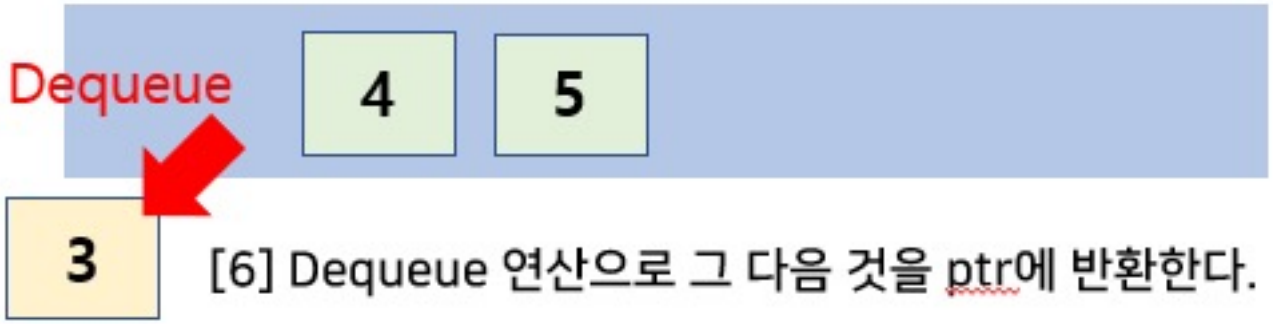
출력 : 1



[4] Dequeue 연산으로 그 다음 것을 ptr에 반환한다.

[5] 반환한 ptr의 데이터를 출력하고, dequeue했던 2의 left, right Child를 큐에 넣는다.

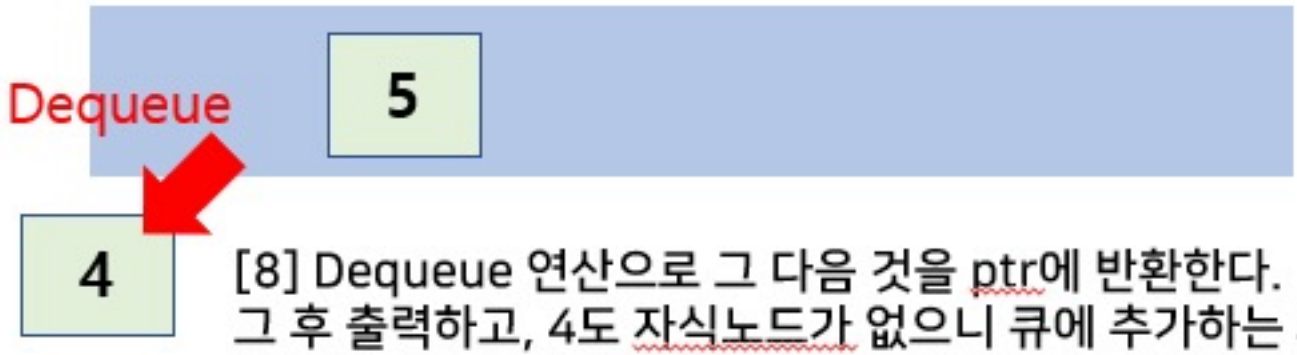
출력 : 1 2



[6] Dequeue 연산으로 그 다음 것을 ptr에 반환한다.

[7] 반환한 ptr의 데이터를 출력하고, dequeue했던 3의 left, right Child를 큐에 넣어야 하지만 3은 자식 노드가 없으므로 left, right Child가 NULL이다. 따라서 큐에 넣는 if문을 돌지 않는다.

출력 : 1 2 3



[8] Dequeue 연산으로 그 다음 것을 ptr에 반환한다.
그 후 출력하고, 4도 자식노드가 없으니 큐에 추가하는 과정이 없으며, 5도 출력한 뒤 자식 노드가 없으니 그 과정이 동일하다.

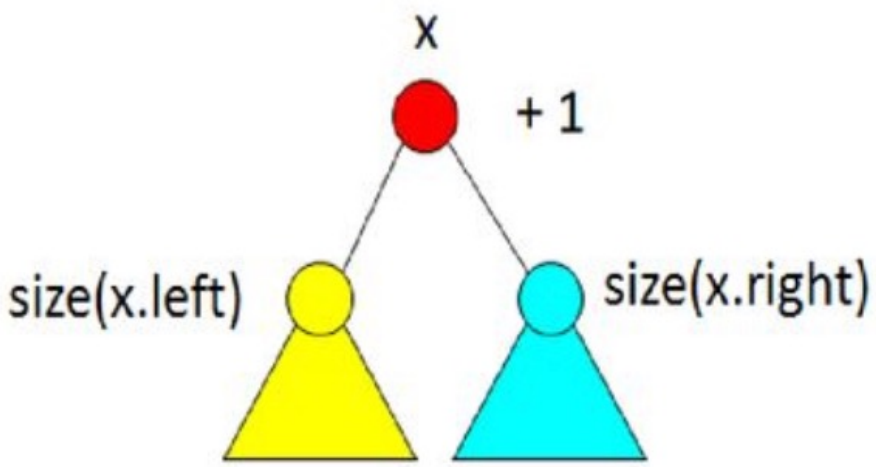
[9] 모든 것이 빠져나와 큐가 빈 상태면 Level Order 순회를 종료하게 된다.

루트 노드 넣기 -> dequeue로 반환해 출력 -> 반환한 노드의 자식 노드들 큐에 차례대로 넣기(왼쪽, 오른쪽 순) -> dequeue로 다음 노드 반환해 출력 -> 반환한 노드의 자식 노드를 큐에 넣기 -> .... 이 과정의 반복입니다.

```c
void levelOrder(treePointer ptr) {
    if (!ptr)
        return;

    Enqueue(ptr);

    while (1) {
        ptr = Dequeue();
        if (ptr)
        {
            printf("%d ", ptr->data);

            if (ptr->leftChild)
                Enqueue(ptr->leftChild);
            if (ptr->rightChild)
                Enqueue(ptr->rightChild);
        }
        else
            break;
    }
}
```

트리의 노드 수 구하기

트리의 노드 수를 계산하는 것은 트리의 아래에서 위로 각 자식의 후손노드 수를 합하며 올라가는 과정을 통해 수행되며, 최종적으로 루트노드에서 총 합을 구함

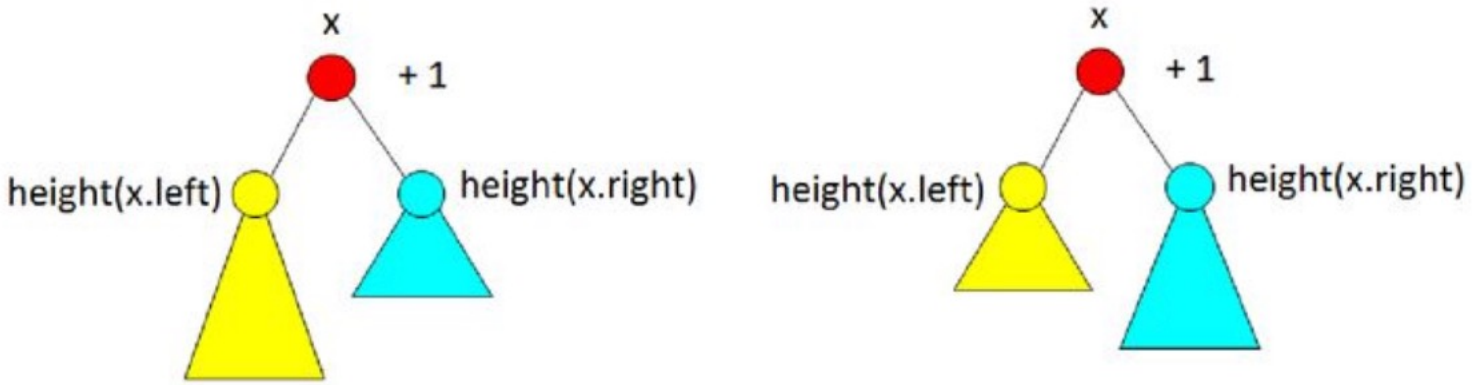


```
int size(Node n){
    if(n==NULL)
        return 0;
    else
        return (1+size(n.getLeft())+ size(n.geetRight()));

}
```

노드수: 1+ (루트노드의 왼쪽 서브트리에 있는 노드 수) + (루트노드의 오른쪽 서브트리에 있는 노드 수)

트리의 높이 구하기

트리의 높이도 아래에서 위로 두 자식을 각각
루트노드로 하는 서브트리의 높이를 비교하여 보다 큰
높이에 1을 더하는 것으로 자신의 높이를 계산하며,
최종적으로 루트노드의 높이가 트리의 높이가 됨

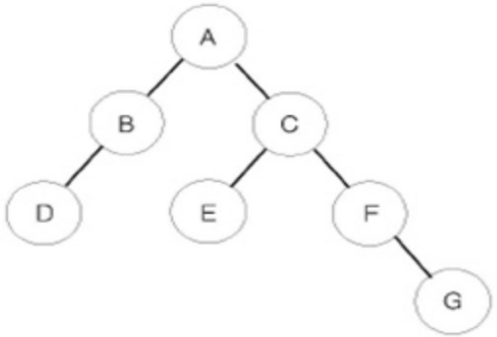


```
int height(Node n){
    if(n==NULL)
        return 0;
    else
      return (1+ max(heigth(n.getLeft()),height(n.getRight())))

}
```

트리의 높이 : 1 + max(루트의 왼쪽 서브트리의 높이. 루트의 오른쪽 서브트리의 높이)

문제

이진 트리를 입력받아 전위 순회(preorder traversal), 중위 순회(inorder traversal), 후위 순회(postorder traversal)한 결과를 출력하는 프로그램을 작성하시오.



입력

첫째 줄에는 이진 트리의 노드의 개수 N(1 ≤ N ≤ 26)이 주어진다. 둘째 줄부터 N개의 줄에 걸쳐 각 노드와 그의 왼쪽 자식 노드, 오른쪽 자식 노드가 주어진다. 노드의 이름은 A부터 차례대로 알파벳 대문자로 매겨지며, 항상 A가 루트 노드가 된다. 자식 노드가 없는 경우에는 .으로 표현한다.

출력

첫째 줄에 전위 순회, 둘째 줄에 중위 순회, 셋째 줄에 후위 순회한 결과를 출력한다. 각 줄에 N개의 알파벳을 공백 없이 출력하면 된다.

예제 입력 1 복사

```
7
A B C
B D .
C E F
E . .
F . G
D . .
G . .
```

예제 출력 1 복사

```
ABDCEFG
DBAECFG
DBEGFCA
```

https://www.acmicpc.net/problem/1991

```
int main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        char x, y, z;
        cin >> x >> y >> z;
        x = x - 'A';
        if (y == '.') {
            a[x].left = -1;
        }
        else {
            a[x].left = y - 'A';
        }
        if (z == '.') {
            a[x].right = -1;
        }
        else {
            a[x].right = z - 'A';
        }
    }
```

```cpp
struct Node {
    int left;
    int right;
};

Node a[50];

void preorder(int x) {
    if (x == -1) return;
    cout << (char)(x + 'A');
    preorder(a[x].left);
    preorder(a[x].right);
}
```

Thank you

나눔스퀘어