이진트리

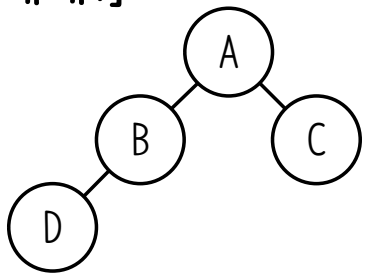
2018818716 컴퓨터공학부 김용현

```
1   bool isfull() {
2       return getcount() == pow(2, getheight()) - 1;
3   }
```

[코드]

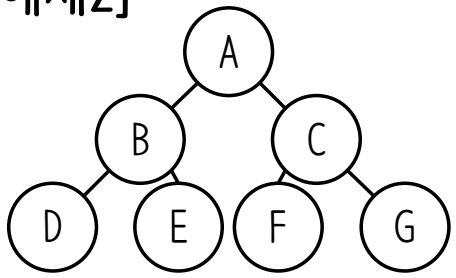
1. 트리의 높이가 h일 때,
   트리는 최소 h개 ~ 최대 2^h – 1개의 노드를 가진다.
2. 이때, 노드의 개수가 0, 1, 3, 7, ..., 2^h – 1개 일 때,
   (즉, 노드의 개수가 해당 높이 h에서 지닐 수 있는
   최대 개수일 때) '포화이진트리'가 된다.
3. 노드의 개수가 0개일때도 포화이진트리가 성립하는
   것으로 생각하고 코드를 작성하였다.

[예제1]



getcount() = 4
getheight() = 3
=> 2^3-1 = 7
∴ 포화이진트리 false

[예제2]



getcount() = 7
getheight() = 3
=> 2^3-1 = 7
∴ 포화이진트리 true

```
1   bool isbalanced() {
2       return isbalanced(root);
3   }
4   bool isbalanced(binarynode* node) {
5       /* Base Case */
6       if (node == NULL)
7           return true;
8       /* Inductive Step */
9       else
10          return
11          abs(getheight(node->getleft()) -
12              getheight(node->getright())) < 2 &&
13          isbalanced(node->getleft()) &&
14          isbalanced(node->getright());
15  }
```

[코드 : isbalanced()]

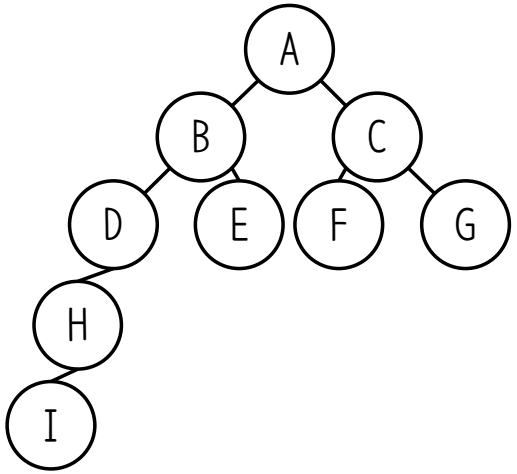
1. return isempty() ? true : isbalanced(root); 에서 isempty() 메소드는 결국 root == NULL 여부를 확인하는 것이므로, return isbalanced(root); 와 그 구조가 동일하다.
2. 노드의 개수가 0개일때도 balanced가 true가 된다 생각하고 코드를 작성하였다.

[코드 : isbalanced(binarynode* node)]

1. 재귀 방식으로 문제를 해결하여, 재귀의 탈출 조건이 되는 Base Case와, 재귀의 귀납 단계가 되는 Inductive Step으로 구분하였다.
2. [Base Case] : node == NULL 인 경우, true 값을 반환한다.
3. [Inductive Step] : ① 좌 · 우측 서브트리의 높이차가 2미만인지 확인한 후, ② 좌측 서브트리에 대해 isbalanced여부를 확인하고, ③ 우측 서브트리에 대해서도 isbalanced여부를 확인한다. ④ 마지막으로 &&연산을 통해 true · false 여부를 반환한다.

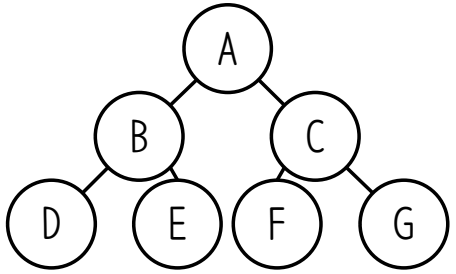
```
1   bool isbalanced() {
2       return isbalanced(root);
3   }
4   bool isbalanced(binarynode* node) {
5       /* Base Case */
6       if (node == NULL)
7           return true;
8       /* Inductive Step */
9       else
10          return
11          abs(getheight(node->getleft()) -
12              getheight(node->getright())) < 2 &&
13          isbalanced(node->getleft()) &&
14          isbalanced(node->getright());
15  }
```

[예제1]



[A기준]
좌측 서브트리 높이 : 4
우측 서브트리 높이 : 2
4 – 2 = 2 이므로, false
[B기준] : 참고용
좌측 서브트리 높이 : 3
우측 서브트리 높이 : 1
3 – 1 = 2 이므로, false

[예제2]



[A기준]
좌측 서브트리 높이 : 2
우측 서브트리 높이 : 2
2 – 2 = 0 이므로, true
[좌측 서브트리]
isbalanced = true
[우측 서브트리]
Isbalanced = true
∴ true

```
1   void reverse() {
2       return reverse(root);
3   }
4   void reverse(binarynode* node) {
5       /* Base Case */
6       if (node == NULL)
7           return;
8       /* Inductive Step */
9       //Recursion
10      reverse(node->getleft());
11      reverse(node->getright());
12      //Swap
13      binarynode* temp = node->getleft();
14      node->setleft(node->getright());
15      node->setright(temp);
16  }
```

[코드 : reverse()]

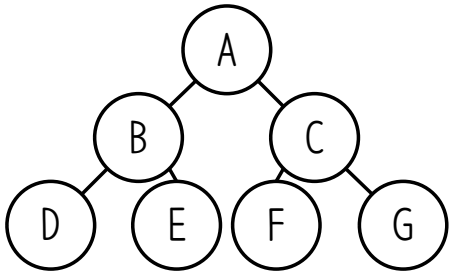
1. reverse(root)값을 반환한다.

[코드 : reverse(binarynode* node)]

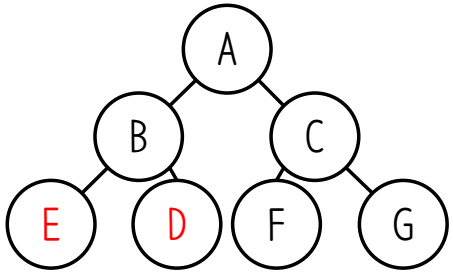
1. 재귀 방식으로 문제를 해결하여, 재귀의 탈출 조건이 되는 Base Case와, 재귀의 귀납 단계가 되는 Inductive Step으로 구분하였다.
2. [Base Case] : node == NULL 인 경우, 반환한다. (함수의 반환값이 void 이므로, 함수의 종료를 위해 return 만 삽입하였다.)
3. [Inductive Step] : ① 좌측 서브트리에 대해 reverse를 수행한다. ② 우측 서브트리에 대해 reverse를 수행한다. ③ 마지막으로 자기 자신에 대해 좌측 서브트리와 우측 서브트리의 위치를 교환한다. (후위 순회방식과 동일한 순서로 진행된다.)

```
1   void reverse() {
2       return reverse(root);
3   }
4   void reverse(binarynode* node) {
5       /* Base Case */
6       if (node == NULL)
7           return;
8       /* Inductive Step */
9       //Recursion
10      reverse(node->getleft());
11      reverse(node->getright());
12      //Swap
13      binarynode* temp = node->getleft();
14      node->setleft(node->getright());
15      node->setright(temp);
16  }
```

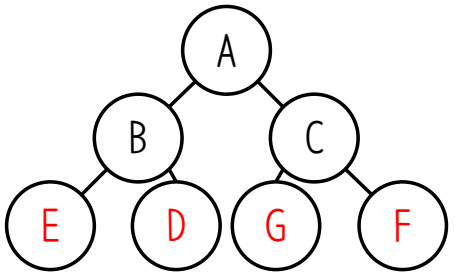
[step 0]



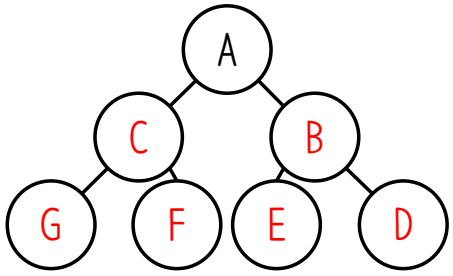
[step 1]



[step 2]



[step 3]



감사합니다!