

Homework #4

(Introduction to Data Structures)

Due date: May 17, 2018

학번: 2014310407

이름: 이 준 혁(JunHyuk Lee)

1. Sum of Leaves

1.1. Solution

가장 먼저, root 가 NULL 인 경우에 -1 을 반환하도록 했다.

그리고 다음과 같이 세 가지 상태를 파악하는 함수를 만들었다.

1. 왼쪽 자식이 있는지 여부
2. 오른쪽 자식이 있는지 여부
3. 해당 노드가 리프인지 여부

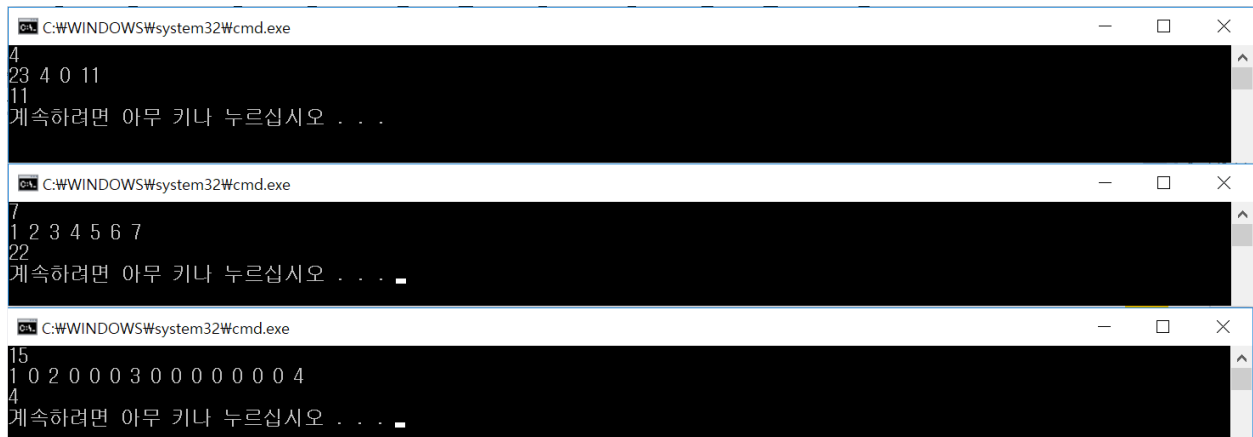
1,2 번 함수는 각각 왼쪽, 오른쪽 자식이 NULL 인지 아닌지 판단해서 있으면 true, 아니면 false 를 반환했다.

3 번 함수는 왼쪽과 오른쪽 자식이 둘 다 없는 경우 리프이므로, 그 경우 true 를, 아닌 경우 false 를 반환했다.

그리고는 함수를 재귀적으로 구성했는데, 만약 입력 노드가 리프이면 그 값을 반환하고, 아닌 경우 왼쪽 자식이 있는 경우 왼쪽 자식을 대상으로 다시 함수를 호출해서 그 반환 값을 더했고, 오른쪽 자식이 있는 경우에도 역시 같은 방법으로 함수를 호출해서 더했다. 그리고 그 더한 값을

반환했다. 이렇게 함수를 구성하면 모든 노드들을 방문하면서, 각 리프인 노드들의 값을 모두 더할 수 있게 된다.

1.2. Result (snapshot)



```
C:\WINDOWS\system32\cmd.exe
4
23 4 0 11
11
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
7
1 2 3 4 5 6 7
22
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
15
1 0 2 0 0 0 3 0 0 0 0 0 0 4
4
계속하려면 아무 키나 누르십시오 . . .
```

2. The Last Leaf

2.1. Solution

역시 마찬가지로 root 가 NULL 인 경우는 -1 을 반환하도록 했다.

이 문제는 우선 입력 받은 time 번 만큼 트리의 모든 leaf 들을 제거한 후에, 남은 트리의 리프들을 다 더하는 방식으로 진행했다. 이를 위해 1 번에서 구현한 모든 리프들의 값의 합을 구하는 함수와 리프인지 여부를 판별하는 함수를 가져왔다.

그리고 트리의 모든 노드들을 방문하며 노드가 리프인 경우에는 지우도록 했다. 이 때 level order traversal 이 가장 탐색이 용이해 보여서, Queue 를 이용해 모든 노드들을 탐색했다. 큐에 root 를 넣고, 원소를 하나 뺄 때 마다 왼쪽과 오른쪽 자식들을 방문했다. 이 때 자식이 NULL 인 경우에는 무시하고, 자식이 leaf 인 경우에는 해당 자식 노드를 트리에서 지웠다. 둘 다 아닌 경우,

즉 자식 밑에 자식이 더 있는 경우에는 그 자식을 큐에 넣었다. 이렇게 큐가 모두 빌 때까지, 즉, 모든 노드를 다 방문할 때까지 반복했다.

이렇게 노드를 지우는 과정을 time 번 반복하는데, 만약 그 전에 root 만 남은 트리 모양이 된다면 반드시 트리가 비게 되므로, -1 을 반환했다.

이 과정을 모두 반복한 후에, 남은 트리에서 리프들의 합을 구했다.(1 번 문제 함수 이용)

2.2. Result (snapshot)

```
C:\WINDOWS\system32\cmd.exe
10
2 1 1 3 4 2 5 0 0 1
1
5
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
3
2 1 1
2
-1
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
15
6 7 2 2 3 0 9 0 0 0 4 0 0 0 8
2
9
계속하려면 아무 키나 누르십시오 . . .
```

3. Balancing Weights

3.1. Solution

역시 마찬가지로 root 가 NULL 인 경우는 -1 을 반환하도록 했다.

우선 어떤 주어진 노드 아래의 모든 노드의 합을 구하는 함수가 필요하다고 생각해서 재귀적으로 Sum 이라는 함수를 정의했다. 이 함수는 recursion 을 이용해, 왼쪽 노드들의 합과 오른쪽 노드들의 합으로 총 트리의 합을 구하는 함수이다.

그리고 주어진 함수를 구성하는데, 다음과 같은 단계를 거쳤다. 주어진 함수가 무게 차이를 반환하는 함수라는 점에 유의해 다음과 같이 재귀적으로 함수를 구성했다.

1. leaf 인 경우에는, balance 를 맞출 필요가 없으므로 0 을 반환했다.
2. 무게 차이를 나타내는 변수, 왼쪽과 오른쪽 subtree 의 모든 값의 합을 구하는 변수를 선언했다
3. 왼쪽 자식이 있는 경우에는, 왼쪽에서 얻은 무게의 차이 값을 더하고(재귀적으로 함수 호출) 왼쪽 subtree 의 합을 구했다.
4. 오른쪽 자식이 있는 경우에도 3 과 같은 방식으로 구했다
5. left subtree 의 합이 right subtree 의 합보다 큰 경우, 그 무게 차이만큼 더한 후 오른쪽 자식에게 그 차이만큼 무게를 더해준다.
6. 5 번의 반대의 경우도 마찬가지로 더해준다.
7. 총 구한 모든 무게 차이를 반환한다.

이렇게 함수를 구성하면 주어진 노드에 대해 해당 노드 아래의 모든 무게 차이를 반환하는 함수를 얻게 된다.

3.2. Result (snapshot)

```
C:\WINDOWS\system32\cmd.exe
5
1 3 2 1 2
6
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
7
1 2 2 4 4 4 4
0
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
15
2 1 2 2 1 1 1 1 2 2 2 0 0 0 0
11
계속하려면 아무 키나 누르십시오 . . .
```

4. Invisible Nodes

4.1. Solution

역시 마찬가지로 root 가 NULL 인 경우는 -1 을 반환하도록 했다.

우선 주어진 레벨에서 가장 첫번째 노드 혹은 마지막 노드를 구해야 하므로, level order traversal 을 이용하기 위해 큐를 이용했다. 그리고 어떤 노드의 level 을 구해야 하므로, 레벨을 구하기 위해, $\text{floor}(\log_2(x))$ 함수를 tail recursion 을 이용 해 구현했다.

그리고 level order traversal 을 해야 하는데, 트리가 complete 혹은 full binary tree 가 아닌 임의의 binary tree 이므로, level 을 구하기 어렵게 된다. 따라서, 자식이 없는 경우에는 NULL 을 큐에 넣기로 했다. 다만 이 경우 traversal 을 끝내기 힘들어지므로, 최대 입력 값인 31 번까지만 시행하도록 했다.

이렇게 모든 노드들을 레벨 순서로 방문하는데, 만약 방문한 노드가 NULL 인 경우 큐에 NULL 을 두 개 넣고 다음 노드를 확인했다. 노드가 NULL 이 아닌 경우, left 가 있으면 left 를 큐에 넣었고, 없으면 NULL 을 큐에 넣었다. 그리고 이 노드가 visible 인지 여부를 확인해야 했다.

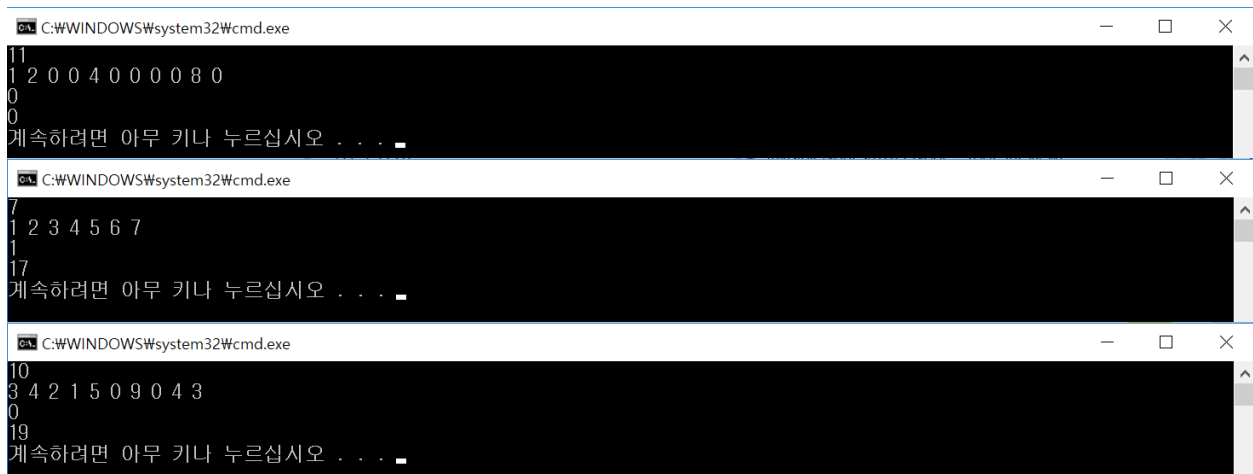
우선 각 레벨별로 visible 노드들을 다 저장할 배열을 선언했다. 최대 레벨이 5 이므로, 크기가 5 인 배열을 선언 후 0 으로 초기화했다. 그리고 view 가 오른쪽, 왼쪽인 경우로 나뉘었다.

view 가 왼쪽인 경우, 해당 레벨에서 가장 먼저 등장하는 NULL 이 아닌 노드가 visible 이므로, NULL 이 아닌 노드를 발견했을 때, 해당 레벨에서 먼저 발견한 노드가 없으면 그 값을 배열에 저장했다.

view 가 오른쪽인 경우, 해당 레벨에서 가장 마지막에 등장하는 NULL 이 아닌 노드가 visible 이므로, 해당 레벨에서 노드를 발견할 때 마다 그 값을 배열에 저장했다.

이렇게 하면 배열에 모든 visible 한 노드를 저장할 수 있다. 총 트리의 합에서 visible 한 노드의 합을 빼면 invisible 한 노드의 합이 된다. 이 값을 반환했다.

4.2. Result (snapshot)



The image shows three sequential screenshots of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". Each screenshot displays a binary tree structure with nodes containing integers. The first screenshot shows a tree with root 11, left child 1, and right child 0. The second screenshot shows a tree with root 7, left child 1, and right child 17. The third screenshot shows a tree with root 10, left child 3, and right child 19. Each screenshot also shows a prompt "계속하려면 아무 키나 누르십시오 . . ."

5. Least Common Ancestor

5.1. Solution

역시 마찬가지로 root 가 NULL 인 경우는 -1 을 반환하도록 했다.

아닌 경우에는, tail recursion 을 이용해 LCA 의 level 을 구하도록 했다. 다음과 같은 경우로 나누었다.

1. 주어진 노드의 아이템이 2 인 경우
2. 노드의 아이템이 2 가 아니면서, 리프인 경우
3. 아이템이 2 가 아니면서, 리프도 아닌 경우

1 번의 경우에는, 그 레벨을 알아야 하므로, 노드의 깊이를 반환했다.

2 번의 경우에는 해당 위치에 2 가 없다는 말이므로 이를 알리기 위해 0 을 반환했다.

3 번의 경우에는, 아래에 자식이 있다는 말이므로 왼쪽, 오른쪽 깊이를 나타내는 변수를 선언한 후 0 으로 초기화했다. 이후, 왼쪽과 오른쪽의 깊이를 재귀적으로 구했다. (단, 자식이 있는 경우에만) 이 때 레벨이 하나 증가하므로, depth+1 을 입력으로 주었다. 그리고 다음과 같이 세 가지 경우로 나누었다.

3-1. 왼쪽 깊이와 오른쪽 깊이가 모두 0 인 경우

3-2. 두 쪽의 깊이가 모두 0 이 아닌 경우

3-3 한쪽 깊이만 0 인 경우

3-1 번의 경우에는 두 자식 모두 2 를 가지고 있지 않으므로, 이 노드 아래에는 2 가 없다는 뜻이다. 이를 알리기 위해 2 번의 경우와 같이 0 을 반환했다.

3-2 번의 경우에는 이 노드가 LCA 가 된다. 따라서 현재 깊이를 반환했다.

3-3 번의 경우에는, 한 쪽에만 2 가 있다는 뜻이 된다. 이 때 2 를 하나만 포함할 수도, LCA 를 포함할 수도 있지만, 어떤 경우 든 0 이 아닌 쪽, 즉 2 가 있는 쪽의 깊이를 반환하면 된다.

이 행위를 루트 노드에 대해, depth 를 0 을 주고 시작하면 LCA 의 깊이를 반환 받게 된다.

5.2. Result (snapshot)



The image shows three sequential screenshots of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". Each screenshot displays a binary string on the first line, a number on the second line, and a Korean prompt on the third line.

```
7  
1 2 1 0 0 2 1  
0  
계속하려면 아무 키나 누르십시오 . . .
```

```
15  
1 1 1 0 0 1 2 0 0 0 0 2 0 1 0  
1  
계속하려면 아무 키나 누르십시오 . . .
```

```
17  
2 1 1 1 1 0 0 1 0 1 0 0 0 0 0 1 2  
0  
계속하려면 아무 키나 누르십시오 . . .
```