주어진 이진트리에서 깊이별로 연결 리스트를 만들어 내는 알고리즘을 작성하라.(트리의 깊이가 D라면, 알고리즘 수행 결과로 D개의 연결 리스트가 만들어져야 한다.)

해법

척 보면 깊이별 순회가 필요할 것 같지만, 실제로는 그렇지 않다. 아무 순회 방법이나 사용해도 된다. 현재 탐색중인 노드의 깊이만 추적할 수 있으면 된다.

전순회pre-order traversal 알고리즘을 살짝 변형하여 풀어보자. 재귀 함수를 호출할 때에는 level + 1을 인자로 넘기도록 한다. 아래의 코드는 깊이 우선 탐색 기법을 사용한 구현 결과다.

```
1   void createLevelLinkedList(TreeNode root,
2   ArrayList<LinkedList<TreeNode>> lists, int level) {
```

```
  if (root == null) return; // 초기 사례

  LinkedList<TreeNode> list = null;
  if (lists.size() == level) { // 해당 높이가 리스트 안에 없다
    list = new LinkedList<TreeNode>();
    /* 정순회 방법을 사용하였다는 것에 유의하자.
     * 그러므로, 깊이 #i를 처음 방문한 거라면, 깊이 #0부터 i-1까지는
     * 이미 방문한 상태다. 그러므로 깊이 #i는 마지막에 추가해도
     * 안전하다. */
    lists.add(list);
  } else {
    list = lists.get(level);
  }
  list.add(root);
  createLevelLinkedList(root.left, lists, level + 1);
  createLevelLinkedList(root.right, lists, level + 1);
}

ArrayList<LinkedList<TreeNode>> createLevelLinkedList(
        TreeNode root) {
  ArrayList<LinkedList<TreeNode>> lists =
    new ArrayList<LinkedList<TreeNode>>();
  createLevelLinkedList(root, lists, 0);
  return lists;
}
```

이렇게 하는 대신, 너비 우선 탐색 기법을 변경하여 구현할 수도 있다. 그렇게 할 경우 루트를 먼저 방문하고, 그 다음에 깊이 #2에 해당하는 노드를, 그리고 깊이 #3에 해당하는 노드를 방문해 나가게 될 것이다.

따라서 깊이 #i에 도달했을 때, 깊이 #i - 1에 해당하는 노드들은 전부 방문한 상태가 된다. 즉, 깊이 #i에 어떤 노드들이 있는지 알아내려면, #i-1에 있는 노드의 모든 자식 노드를 검사하기만 하면 된다.

이 알고리즘을 구현한 코드를 아래에 보였다.

```
1  ArrayList<LinkedList<TreeNode>> createLevelLinkedList(
2        TreeNode root) {
3     ArrayList<LinkedList<TreeNode>> result =
```

```
    new ArrayList<LinkedList<TreeNode>>();
    /* 루트 '방문' */
    LinkedList<TreeNode> current = new LinkedList<TreeNode>();
    if (root != null) {
      current.add(root);
    }

    while (current.size() > 0) {
      result.add(current); // 이전 깊이 추가
      LinkedList<TreeNode> parents = current; // 다음 깊이로 진행
      current = new LinkedList<TreeNode>();
      for (TreeNode parent : parents) {
        /* 자식 노드들 방문 */
        if (parent.left != null) {
          current.add(parent.left);
        }
        if (parent.right != null) {
          current.add(parent.right);
        }
      }
    }
    return result;
}
```