

알고리즘 10주차 11/09 201902694 박재우

RANK	TEAM	SCORE	1-FindRoot ○ [1 row]	2-TreeTraversal ● [2 rows]	3-LowestCommonAncestor ○ [3 rows]	4-DeepestSubtree ● [4 rows]
17	201902694	10 1003	73 2 tries	119 2 tries	292 4 tries	359 4 tries

Submissions			
time	problem	lang	result
11/09/20-20:08	4-DeepestSubtree	JAVA	CORRECT
11/09/20-19:54	4-DeepestSubtree	JAVA	WRONG-ANSWER
11/09/20-19:44	4-DeepestSubtree	JAVA	WRONG-ANSWER
11/09/20-19:01	3-LowestCommonAncestor	JAVA	CORRECT
11/09/20-19:00	3-LowestCommonAncestor	JAVA	WRONG-ANSWER
11/09/20-18:43	3-LowestCommonAncestor	JAVA	WRONG-ANSWER
11/09/20-18:31	3-LowestCommonAncestor	JAVA	WRONG-ANSWER
11/09/20-18:08	4-DeepestSubtree	JAVA	WRONG-ANSWER
11/09/20-16:08	2-TreeTraversal	JAVA	CORRECT
11/09/20-16:06	2-TreeTraversal	JAVA	WRONG-ANSWER
11/09/20-16:05	1-FindRoot	JAVA	CORRECT
11/09/20-15:22	1-FindRoot	JAVA	CORRECT
11/09/20-14:52	1-FindRoot	JAVA	WRONG-ANSWER

Clarifications

No clarifications.

Clarifications

No clarification request.

request clarification

문제 해결방법중 node class는 공통임으로 상위에 한번만 기록했다.

```
class Node {
    private String data;
    private Node leftChild = null;
    private Node rightChild = null;

    public Node(String data) {
        this.data = data;
        third.box.add(this);
    }

    public void setData(String data) {
        this.data = data;
    }

    public String getData() {
        return data;
    }

    public void setLeftChild(Node leftChild) {
        this.leftChild = leftChild;
    }

    public Node getLeftChild() {
        return leftChild;
    }

    public Node getRightChild() {
        return rightChild;
    }

    public void setRightChild(Node rightChild) {
        this.rightChild = rightChild;
    }
}
```

문제 1

- 문제/목표

입력받은 값으로 tree를 구축하고 꼭대기 찾기

- 해결방법

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int a;
    int b = 1;
    int c = 0;

    Node node, leftNode, rightNode;
    String[] input;

    a = sc.nextInt();
    sc.nextLine();
    for(int i = 0 ; i < a ; i++) {
        input = sc.nextLine().split(" ");
        node = findeNode(input[0]);
        if(input[1].compareTo(".") != 0) {
            leftNode = findeNode(input[1]);
            node.setLeftChild(leftNode);
        }
        if(input[2].compareTo(".") != 0) {
            rightNode = findeNode(input[2]);
            node.setRightChild(rightNode);
        }
    }

    String top = find_top(box);
    System.out.print(top);
}

public static String find_top(ArrayList<Node> box) {
    int[] count = new int[box.size()];
    for(int i=0;i<box.size();i++) {
        for(int j=0;j<box.size();j++) {
            if(box.get(j).getLeftChild()!=null &&
box.get(i).getData().compareTo(box.get(j).getLeftChild().getData()) == 0) {
                count[i]++;
            }
            if(box.get(j).getRightChild()!=null &&
box.get(i).getData().compareTo(box.get(j).getRightChild().getData()) == 0) {
                count[i]++;
            }
        }
    }
    int min = Integer.MAX_VALUE;
    int index = -1;
    for(int i=0;i<box.size();i++) {
        if(count[i] < min) {
            min = count[i];
            index = i;
        }
    }
    return box.get(index).getData();
}
```

```

- public static Node findeNode(String key) {
-     for(int i = 0 ; i < box.size() ; i++) {
-         if(box.get(i).getData().compareTo(key) == 0) {
-             return box.get(i);
-         }
-     }
-     return new Node(key);
- }
-
- }

```

1차적으로 2진트리 생성을 위한 node를 만든다(node는 최상단에 적어두었다)

그후 입력값에 따라 node의 data와 left, right child를 생성한다.

Find_top이라는 함수를 통하여 깊이 0의 node를 반환한뒤 출력한다.

Find_top이라는 함수는 우선 box에 저장되어있는 각각의 node들중 첫번째부터

각 node의 left, right child에 해당이 되는지를 count한다.

만약 box안에 있는 node중 반복중에 단한번도 left, right child가 되지 않는 node가 깊이 0의 최상위 node이다.

- 결과

```

<terminated> first (7) [Java Application]
6
D . .
E . .
F . .
C F .
B D E
A B C
A

```

- 시간복잡도 : find_top 함수에서 2중 for문으로 순차 탐색한다 -> $O(n^2)$

- 자신만의 생각

이문제를 arrayList로 한번, node를 생성해서 한번 이렇게 2번을 풀었다.

처음에는 list로 푸는것이 편한것 같았지만 이후에는 node를 사용해 이진트리를 만든뒤 문제를 풀어나가는 것이 훨씬 편할것 같다고 생각이 되어 수정을 했다.

- 난이도 2/5

문제 2

- 문제/목표

해당 트리를 Preorder / Inorder / Postorder 순으로 순회하고 출력하기

- 해결방법

```
public class second {  
  
    public static ArrayList<Node> box = new ArrayList<>();  
    public static int count = 0;  
  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        int a;  
        int b = 1;  
        int c = 0;  
  
        Node node, leftNode, rightNode;  
        String[] input;  
  
        a = sc.nextInt();  
        sc.nextLine();  
        for(int i = 0 ; i < a ; i++) {  
            input = sc.nextLine().split(" ");  
            node = findeNode(input[0]);  
            if(input[1].compareTo(".") != 0) {  
                leftNode = findeNode(input[1]);  
                node.setLeftChild(leftNode);  
            }  
            if(input[2].compareTo(".") != 0) {  
                rightNode = findeNode(input[2]);  
                node.setRightChild(rightNode);  
            }  
        }  
  
        String top = find_top(box);  
        pre(findeNode(top));  
        System.out.println("");  
        in(findeNode(top));  
        System.out.println("");  
        post(findeNode(top));  
    }  
  
    public static String find_top(ArrayList<Node> box) {  
        int[] count = new int[box.size()];  
        for(int i=0; i<box.size(); i++) {  
            for(int j=0; j<box.size(); j++) {  
                if(box.get(j).getLeftChild() != null &&  
box.get(i).getData().compareTo(box.get(j).getLeftChild().getData()) == 0) {  
                    count[i]++;  
                }  
                if(box.get(j).getRightChild() != null &&  
box.get(i).getData().compareTo(box.get(j).getRightChild().getData()) == 0) {  
                    count[i]++;  
                }  
            }  
        }  
    }  
}
```

```

        int min = Integer.MAX_VALUE;
        int index = -1;
        for(int i=0;i<box.size();i++) {
            if(count[i] < min) {
                min = count[i];
                index = i;
            }
        }
        return box.get(index).getData();
    }

    public static void pre(Node root) {
        System.out.print(root.getData()+" ");
        if(root.getLeftChild() != null) pre(root.getLeftChild());
        if(root.getRightChild() != null) pre(root.getRightChild());
    }

    public static void in(Node root) {

        if(root.getLeftChild() != null) in(root.getLeftChild());
        System.out.print(root.getData()+" ");
        if(root.getRightChild() != null) in(root.getRightChild());
    }

    public static void post(Node root) {

        if(root.getLeftChild() != null) post(root.getLeftChild());
        if(root.getRightChild() != null) post(root.getRightChild());
        System.out.print(root.getData()+" ");
    }

    public static Node findeNode(String key) {

        for(int i = 0 ; i < box.size() ; i++) {
            if(box.get(i).getData().compareTo(key) == 0) {
                return box.get(i);
            }
        }
        return new Node(key);
    }

}

```

재귀를 사용해서 각각 pre, in, post 탐색으로 출력하도록 했다.

기본 프로그램은 1번과 동일하나 pre, in, post 라는 함수를 추가해서 문제를 해결했다.

Pre는 먼저 방문한 node의 data를 출력, 왼쪽,오른쪽을 방문해 재귀를 통해 data를 출력한다.

In은 입력받은 node의 왼쪽을 전부 방문한 뒤에 뒤에 끝부터 출력을 하고 오른쪽을 방문하여 data를 출력하는 순으로 반복한다.

Post는 pre와는 다르게 왼쪽 오른쪽을 전부 방문한 후 뒤부터 출력을 한다.

- 결과

```
Console x Problems Debug Shell
<terminated> second (6) [Java Application] /Library/Java/JavaVirtual
6
D . .
E . .
F . .
C F .
B D E
A B C
A B D E C F
D B E A F C
D E B F C A
```

- 시간복잡도 : 재귀를 사용하므로
- 자신만의 생각
이문제는 이전 과목 자료구조에서 한번 배웠던 것이므로 떠올리는데 시간이 걸린것 이외
에는 문제가 되지 않았다고 생각한다.
- 난이도 1/5

문제 3

- 문제/목표
질의 노드의 최소 공통 조상 노드를 출력
- 해결방법

```
public class third {

    public static ArrayList<Node> box = new ArrayList<>();
    public static int count = 0;
    static LinkedList<Node>[] d;

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a;
        int b = 1;
        int c = 0;
        int index = 0;

        Node node, leftNode, rightNode;
        String[] input;

        a = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < a; i++) {
            input = sc.nextLine().split(" ");
            node = findeNode(input[0]);
            if (input[1].compareTo(".") != 0) {
                leftNode = findeNode(input[1]);
                node.setLeftChild(leftNode);
            }
            if (input[2].compareTo(".") != 0) {
                rightNode = findeNode(input[2]);
                node.setRightChild(rightNode);
            }
        }
        d = new LinkedList[a];
        for (int i = 0; i < a; i++) {
            d[i] = new LinkedList<>();
        }
    }
}
```

```

        String n1 = sc.next();
        String n2 = sc.next();
        String top = find_top(box);

        System.out.print(findLCA(findeNode(top), findeNode(n1),
findeNode(n2)).getData());

    }

    public static Node findLCA(Node root, Node node1, Node node2) {

        if (root == node1 || node2 == root)
            return root;

        if(node1 == node2)
            return node1;

        if (root == null) {
            return null;
        }

        Node left = root.getLeftChild();
        Node right = root.getRightChild();
        Node lcaNode1;
        Node lcaNode2;

        if (left != null && (left == node1 || left == node2)) {
            if(left.getLeftChild() != null && left.getRightChild() !=null &&(
node2
            left.getLeftChild() == node1 || left.getLeftChild() ==
== node2))
                || left.getRightChild() == node1 || left.getRightChild()
                return left;
            return root;
        }

        if (right != null && (right == node1 || right == node2)) {
            if(right.getLeftChild() != null && right.getRightChild() !=null
&&(
                right.getLeftChild() == node1 || right.getLeftChild() ==
node2
                || right.getRightChild() == node1 ||
right.getRightChild() == node2))
                return right;
            return root;
        }

        lcaNode1 = findLCA(left, node1, node2);
        lcaNode2 = findLCA(right, node1, node2);

        if ((lcaNode1 != null) && lcaNode2 != null) {
            return root;
        }

        if (lcaNode1 != null) {
            return lcaNode1;
        }

        if (lcaNode2 != null) {
            return lcaNode2;
        }

        return null;
    }

```

```

- public static String find_top(ArrayList<Node> box) {
-     int[] count = new int[box.size()];
-     for (int i = 0; i < box.size(); i++) {
-         for (int j = 0; j < box.size(); j++) {
-             if (box.get(j).getLeftChild() != null
-                 &&
- box.get(i).getData().compareTo(box.get(j).getLeftChild().getData()) == 0) {
-                 count[i]++;
-             }
-             if (box.get(j).getRightChild() != null
-                 &&
- box.get(i).getData().compareTo(box.get(j).getRightChild().getData()) == 0) {
-                 count[i]++;
-             }
-         }
-     }
-     int min = Integer.MAX_VALUE;
-     int index = -1;
-     for (int i = 0; i < box.size(); i++) {
-         if (count[i] < min) {
-             min = count[i];
-             index = i;
-         }
-     }
-     return box.get(index).getData();
- }

- public static Node findeNode(String key) {
-     for (int i = 0; i < box.size(); i++) {
-         if (box.get(i).getData().compareTo(key) == 0) {
-             return box.get(i);
-         }
-     }
-     return new Node(key);
- }
- }

```

기본적인 프로그램은 2번과 동일하나 문제해결을 위해 findLCA라는 함수가 추가되었다.

FindLCA는 우선 문제의 조건을 만족하기 위해 최상단 root를 입력받으며

이 root가 질이랑 같으면 그대로 root를 반환한다. 또한 질 두개가 동일하다면 둘중 하나를 리턴시킨다.

이후 예외처리를 해주는데 왼쪽노드가 질 둘중 하나랑 동일하다면

그 왼쪽노드의 child가 남은 질이랑 동일할경우 left를 return한다.

오른쪽도 역시 마찬가지로 매커니즘으로 예외처리를 해준다.

아닐경우 root를 리턴한다.

아닐경우 재귀를 통해 다음 노드로 넘어간다.

이후 그값들이 null이 나닐경우 root를 반환한다.

- 결과


```

<terminated> third (6) [Java Applet]
5
4 . .
3 . .
2 4 5
1 2 3
5 . .
3 4
1

```

- 시간복잡도 : $O(\log N)$
- 자신만의 생각
문제를 처음에 잘못 이해하고 함수를 만들어서 예외처리할것이 매우 늘어났다.
잘못만든 상태에서 유지한채로 예외를 처리하니라 if문의 가독성이 떨어졌다고 생각한다.
- 난이도 4/5

문제 4

- 문제/목표
가장 깊은 노드를 모두 포함하는 서브 트리의 '루트'를 출력하시오.
- 해결방법

```

public class fourth {

    public static ArrayList<Node> box = new ArrayList<>();
    public static int count = 0;
    static LinkedList<Node>[] d;
    static LinkedList<String>[] dd;

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a;
        int b = 1;
        int c = 0;
        int index = -1;

        Node node, leftNode, rightNode;
        String[] input;

        a = sc.nextInt();
        sc.nextLine();
        for(int i = 0 ; i < a ; i++) {
            input = sc.nextLine().split(" ");
            node = findeNode(input[0]);
            if(input[1].compareTo(".") != 0) {
                leftNode = findeNode(input[1]);
                node.setLeftChild(leftNode);
            }
            if(input[2].compareTo(".") != 0) {
                rightNode = findeNode(input[2]);
                node.setRightChild(rightNode);
            }
        }

        d = new LinkedList[a];
        dd = new LinkedList[a];
    }
}

```

```

        int min = -1;
        for(int i=0;i<a;i++) {
            d[i] = new LinkedList<>();
            dd[i] = new LinkedList<>();
        }
        String top = find_top(box);
        find_last(findeNode(top),0);

        for(LinkedList k : dd) {
            if( k.size()>0)
                min++;
        }
        Node[] check = R_LCA(findeNode(top),d[min].toArray(new
Node[d[min].size()]),min);
        System.out.print(check[0].getData());
    }
    public static Node findLCA(Node root, Node node1, Node node2) {

        if (root == node1 || node2 == root)
            return root;

        if(node1 == node2)
            return node1;

        if (root == null) {
            return null;
        }

        Node left = root.getLeftChild();
        Node right = root.getRightChild();
        Node lcaNode1;
        Node lcaNode2;

        if (left != null && (left == node1 || left == node2)) {
            if(left.getLeftChild()!= null && left.getRightChild() !=null &&
                left.getLeftChild() == node1 || left.getLeftChild() ==
node2
                || left.getRightChild() == node1 || left.getRightChild()
== node2))
                return left;
            return root;
        }

        if (right != null && (right == node1 || right == node2)) {
            if(right.getLeftChild()!= null && right.getRightChild() !=null
&&
                right.getLeftChild() == node1 || right.getLeftChild() ==
node2
                || right.getRightChild() == node1 ||
right.getRightChild() == node2))
                return right;
            return root;
        }
        lcaNode1 = findLCA(left, node1, node2);
        lcaNode2 = findLCA(right, node1, node2);

        if ((lcaNode1 != null) && lcaNode2 != null) {
            return root;
        }

        if (lcaNode1 != null) {
            return lcaNode1;
        }

        if (lcaNode2 != null) {

```

```

        return lcaNode2;
    }
    return null;
}

public static void find_last(Node root,int depth) {
    d[depth].add(root);
    dd[depth].add(root.getData());
    if(root.getLeftChild() != null)
        find_last(root.getLeftChild(),depth+1);
    if(root.getRightChild() != null)
        find_last(root.getRightChild(),depth+1);
}

public static Node[] R_LCA(Node root, Node[] ds,int depth) {
    if(ds.length <= 1)
        return ds;
    int n = 0;
    if(ds.length%2 != 0)
        n= 1;
    Node[] test = new Node[ds.length/2+n];
    int j = 0;
    for(int i = 0;i<ds.length/2;i++) {
        test[i] = findLCA(root,ds[j],ds[j+1]);
        j+=2;
    }
    if(n != 0)
        test[test.length-1] = d[depth-1].getLast();
    test = R_LCA(root,test,depth-1);
    return test;
}

public static String find_top(ArrayList<Node> box) {
    int[] count = new int[box.size()];
    for(int i=0;i<box.size();i++) {
        for(int j=0;j<box.size();j++) {
            if(box.get(j).getLeftChild()!=null &&
box.get(i).getData().compareTo(box.get(j).getLeftChild().getData()) == 0) {
                count[i]++;
            }
            if(box.get(j).getRightChild()!=null &&
box.get(i).getData().compareTo(box.get(j).getRightChild().getData()) == 0) {
                count[i]++;
            }
        }
    }
    int min = Integer.MAX_VALUE;
    int index = -1;
    for(int i=0;i<box.size();i++) {
        if(count[i] < min) {
            min = count[i];
            index = i;
        }
    }
    return box.get(index).getData();
}

public static Node findeNode(String key) {
    for(int i = 0 ; i < box.size() ; i++) {
        if(box.get(i).getData().compareTo(key) == 0) {

```

```

-         return box.get(i);
-     }
- }
-     return new Node(key);
- }

```

프로그램의 구조는 3번과 동일하나 linkedList[] d와 find_last함수, R_LCA함수가 추가되었다.

우선 가장 깊은 node들을 찾기위해 preorder방식으로 탐색하여 d[0]는 깊이0을, d[1]에는 깊이 1의 노드를 추가하는 방법으로 가장 깊은 node를 찾아낸다.

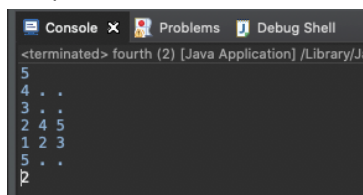
그후 R_LCA함수로 d중 가장 깊은 node가 저장되어있는 linked리스트를 배열로 변경하여 인자로 주고 그 깊이와 최상위 루트를 인자로 전달한다.

그후 입력받은 배열의 길이의 절반만큼 배열을 다시 생성하는데 만약 길이가 홀수라면 길이 +1을 하여 생성한다.

그후 문제 3번에서 사용했던 findLCA를 2개씩 묶어서 사용한다. 이는 가장 깊은 node들의 부모를 찾기 위함이다. 찾고난뒤에는 생성했던 배열에 저장한다.

홀수라면 한개의 node가 남는데 이 node는 바로 위 부모를 찾아 새로 생성했던 node의 가장 끝에 저장한다. 이를 반복하여 배열의 길이가 1이 될때까지 재귀를 사용한다.

- 결과



```

5
4 . .
3 . .
2 4 5
1 2 3
5 . .
2

```

- 시간복잡도 $T(n)$

- 자신만의 생각

3번을 이용해서 풀어야한다는것은 알았지만 이를 적용하는데에 있어서 진짜 많이 어려웠다. 최하위 node를 찾는 아이디어는 위에 설명한 방법밖에는 잘 모르겠다.

교수님의 방법으로는 어떻게 해야할지 감이 잡히질 않는다.

- 난이도 10/5