자료구조 및 실습
보고서

[제07주] DS_07_201502273_김현종

2018.05.08.

201502273/김현종

1.내용

```
import java.util.LinkedList;
import java.util.Queue;

public class Tree<T> {
	private T value;
	private Tree<T> leftChild;
	private Tree<T> rightChild;

	/*
	 * Tree의 생성자
	 * 모두 null로 초기화
	 */
	public Tree() {
		this.value = null;
		this.leftChild = null;
		this.rightChild = null;
	}

	/*
	 * 원소, leftChild, rightChild 모두 입력값으로 초기화하는 생성자
	 */
	public Tree(T anElement, Tree<T> leftChild, Tree<T> rightChild) {
		this.value = anElement;
		this.leftChild = leftChild;
		this.rightChild = rightChild;
	}

	// 왼쪽 자식을 가지고 있는지 확인
	public boolean hasLeftChild() {
		return (this.leftChild != null);
	}

	// 오른쪽 자식을 가지고 있는지 확인
	public boolean hasRightChild() {
		return (this.rightChild != null);
	}

	// Leaf인지 확인
	public boolean isLeaf() {
		return (this.leftChild == null) && (this.rightChild == null);
	}

	/*
	 * 왼쪽 자식이 있으면 해당 트리의 height값을 가져오고
	 * 오른쪽 자식도 있으면 해당 트리의 height값을 가져와 둘 중 큰 값을 반환하는 메소드
	 * 트리의 height를 알 수 있다.
	 */
	public int height() {
		int leftHeight = 0;
		if (this.hasLeftChild()) {
			leftHeight = this.leftChild.height();
		}
		int rightHeight = 0;
		if (this.hasRightChild()) {
			rightHeight = this.rightChild.height();
		}
		if (leftHeight > rightHeight)
			return (leftHeight + 1);
		else
			return (rightHeight + 1);
	}

	/*
	 * 왼쪽 자식의 트리수를 가져오고
	 * 오른쪽 자식의 트리수도 가져와 둘이 합쳐
	 * 총 트리에 있는 노드의 수를 반환하는 함수
	 */
	public int numberOfTrees() {
		int numberOfLeftTrees = 0;
```

```
                if (this.hasLeftChild()) {
                        numberOfLeftTrees = this.leftChild.numberOfTrees();
                }
                int numberOfRightTrees = 0;
                if (this.hasRightChild()) {
                        numberOfRightTrees = this.rightChild.numberOfTrees();
                }
                return (1 + numberOfLeftTrees + numberOfRightTrees);
        }

        //해당 트리의 노드의 값을 반환하는 함수
        public T element() {
                return this.value;
        }

        // 원소 설정하기: setter for element
        public void setElement(T anElement) {
                this.value = anElement;
        }

        /*
         * 해당 트리의 왼쪽 자식을 반환하는 함수
         */
        public Tree<T> leftChild() {
                return this.leftChild;
        }

        // left의 자식을 설정함: setter for leftChild
        public void setLeftChild(Tree<T> leftChild) {
                this.leftChild = leftChild;
        }

        /*
         * 해당 트리의 오른쪽 자식을 반환하는 함수
         */
        public Tree<T> rightChild() {
                return this.rightChild;
        }

        // right의 자식을 설정함: setter for rightChild
        public void setRightChild(Tree<T> rightChild) {
                this.rightChild = rightChild;
        }

        /*
         * inorder로 Tree를 검사하는 함수
         * 왼쪽 자식, root, 오른쪽 자식순으로 검사한다.
         */
        public void inorder(Tree<T> root) {
                if (root != null) {
                        inorder(root.leftChild());
                        visit(root);
                        inorder(root.rightChild());
                }
        }

        /*
         * preorder로 Tree를 검사하는 함수
         * root, 왼쪽 자식, 오른쪽 자식순으로 검사한다.
         */
        public void preorder(Tree<T> root) {
                if (root != null) {
                        visit(root);
                        preorder(root.leftChild());
                        preorder(root.rightChild());
                }
        }

        /*
         * postorder로 Tree를 검사하는 함수
         * 왼쪽 자식, 오른쪽 자식, root순으로 검사한다.
         */
```

```
	public void postorder(Tree<T> root) {
		if (root != null) {
			postorder(root.leftChild());
			postorder(root.rightChild());
			visit(root);
		}
	}

	/*
	 * postorder로 Tree를 검사하는 함수
	 * Queue에 root를 넣고 q에서 하나씩 꺼내며 검사하고
	 * 자식이 있을 경우 q에 넣는 것을 q가 빌때까지 반복한다.
	 */
	public void levelorder(Tree<T> root) {
		Queue<Tree<T>> q = new LinkedList<Tree<T>>();

		q.offer(root);

		while (!q.isEmpty()) {
			Tree<T> temp = q.poll();
			if (temp != null) {
				visit(temp);
				q.offer(temp.leftChild);
				q.offer(temp.rightChild);
			}
		}
	}

	/*
	 * 노드를 검사하는 함수
	 * 노드값을 터미널에 출력한다.
	 */
	private void visit(Tree<T> element) {
		// TODO Auto-generated method stub
		System.out.println(element.element());
	}
}
```

Tree.java

```
import java.util.LinkedList;
import java.util.Queue;

public class MainClass_07_201502273 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		int j = -1;

		Tree<String> tree = new Tree<String>();	//트리 생성

		Queue<Tree<String>> q = new LinkedList<Tree<String>>();	//편리하게
Tree를 만들기위한 queue

		q.offer(tree);	//q에 root를 enQueue

		for(int i = 0; i < 10; i++){
			Tree<String> temp = q.poll();	//q에서 deQueue해옴
			temp.setElement("Node num " + i);	//deQueue해온 Tree의 값
지정
			temp.setLeftChild(new Tree<String>());	//왼쪽 자식 생성
			temp.setRightChild(new Tree<String>());	//오른쪽 자식 생성
			q.offer(temp.leftChild());	//왼쪽 자식 enQueue
			q.offer(temp.rightChild());	//오른쪽 자식 enQueue
			j = i;	//j를 i로 업데이트
		}
```

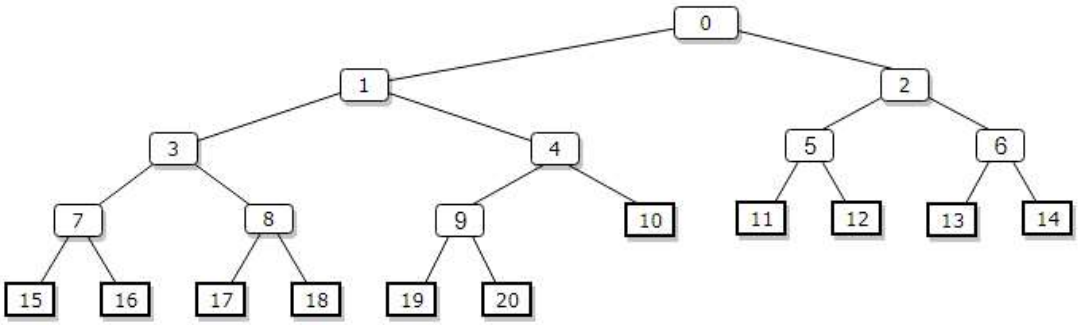
```
                    while(!q.isEmpty()){            //q가 빌때까지
                        j++;        //j를 하나씩 늘려가며
                        Tree<String> temp = q.poll();          //q에서 deQueue해옴
                        temp.setElement("Node num " + j);  //해당 트리의 값 지정
                    }


                    System.out.println("\nPreorder\n");
                    tree.preorder(tree);            //preorder 출력
                    System.out.println("\nInorder\n");
                    tree.inorder(tree); //inorder 출력
                    System.out.println("\nPostorder\n");
                    tree.postorder(tree);           //postorder 출력
                    System.out.println("\nLevelorder\n");
                    tree.levelorder(tree);          //levelorder 출력

            }

}
```

MainClass_07_201502273.java

2.결과

Preorder	Inorder	Postorder	Levelorder
Node num 0	Node num 15	Node num 15	Node num 0
Node num 1	Node num 7	Node num 16	Node num 1
Node num 3	Node num 16	Node num 7	Node num 2
Node num 7	Node num 3	Node num 17	Node num 3
Node num 15	Node num 17	Node num 18	Node num 4
Node num 16	Node num 8	Node num 8	Node num 5
Node num 8	Node num 18	Node num 3	Node num 6
Node num 17	Node num 1	Node num 19	Node num 7
Node num 18	Node num 19	Node num 20	Node num 8
Node num 4	Node num 9	Node num 9	Node num 9
Node num 9	Node num 20	Node num 10	Node num 10
Node num 19	Node num 4	Node num 4	Node num 11
Node num 20	Node num 10	Node num 1	Node num 12
Node num 10	Node num 0	Node num 11	Node num 13
Node num 2	Node num 11	Node num 12	Node num 14
Node num 5	Node num 5	Node num 5	Node num 15
Node num 11	Node num 12	Node num 13	Node num 16
Node num 12	Node num 2	Node num 14	Node num 17
Node num 6	Node num 13	Node num 6	Node num 18
Node num 13	Node num 6	Node num 2	Node num 19
Node num 14	Node num 14	Node num 0	Node num 20

먼저 Level Order로 Tree를 탐색한 결과를 보고 현재 Tree가 아래와 같은 모습으로 있는 것을 볼 수 있다,



이 Tree를 In-order로 탐색하게 되면 15, 7, 16, 3, 17, 8, 18, 1, 19, 9, 20, 4, 10, 0, 11, 5, 12, 2, 13, 6, 14가 나오며 Preorder로 탐색을 하면 0, 1, 3, 7, 15, 16, 8, 17, 18, 4, 9, 19, 20, 10, 2, 5, 11, 12, 6, 13, 14가 나온다는 것을 볼 수 있다. 마지막으로 해당 Tree를 Postorder로 탐색하게 되면 15, 16, 7, 17, 18, 8, 3, 19, 20, 9, 10, 4, 1, 11, 12, 5, 13, 14, 6, 2, 0의 순서로 탐색하게 됨을 볼 수 있다.

깨달은 점 및 결론

재귀를 통해서 Tree를 간단하게 탐색할 수 있다는 것을 알게 되었고 Preorder, In-order, Postorder, Level-order가 depth first search(DFS)와 breadth first search(BFS)의 개념이 된다는 것을 알 수 있었다.
그리고 이진트리 모습으로 터미널에 출력하는 것을 생각해보았는데 현재 코드상태로는 어렵고 파라미터를 추가해야할 것 같다는 생각이 들었다.