자료구조설계

제 출 일	2017.11.12.
과 제 번 호	08
분 반	01
학 과	컴퓨터공학과
학 번	201602038
이 름	이 미 진

1. 구현 내용 설명

1) AVLTree

```
	public AVLTree(int key){		// AVL 트리 생성자
		this.key = key;
		left = right = NIL;
	}
	private AVLTree(){		// AVL 트리 생성자
		left = right = this;
		height = -1;
	}
	private AVLTree(int key, AVLTree left, AVLTree right){
		this.key = key;
		this.left = left;
		this.right = right;
		height = 1 + Math.max(left.height, right.height);
	}
	public boolean add(int key){		// AVL 트리 삽입 메소드
		int oldSize = size(); //성공하면 true 리턴
		grow(key);
		return size() > oldSize;
	}
	public AVLTree grow(int key){	// AVL 트리 확장 메소드
		if (this == NIL) return new AVLTree(key);
		if (key == this.key) return this; // prevent key duplication
		if (key < this.key) left = left.grow(key);
		else right = right.grow(key);
		rebalance();
		height = 1 + Math.max( left.height, right.height );
		return this;
	}
	public int size(){
		if ( this == NIL ) return 0;
		return 1 + left.size() + right.size();
	}
	public String toString(){ // AVL 트리 프린트 메소드
		if ( this.search(key) == false ) return "";
		int bf = left.height - right.height;
		return left + "(" + this.key + "," + bf + ") " + right;
	}
	private void rebalance(){		// AVL 트리 재정렬 메소드
		if ( right.height > left.height+1 ){
			if( right.left.height > right.right.height )
				right.rotateRight();
			rotateLeft();
		}
		else if( left.height > right.height+1 ){
			if( left.right.height > left.left.height )
				left.rotateLeft();
			rotateRight();
		}
	}
	private void rotateLeft(){ // AVL Left rotate 메소드
		left = new AVLTree(key, left, right.left);
		key = right.key;
		right = right.right;
	}
	private void rotateRight(){ // AVL Right rotate 메소드
		right = new AVLTree(key, left.right, right);
		key = left.key;
		left = left.left;
	}
```

public AVLTree remove(int key) 메소드와 remove를 순환적으로 하기 위해 추가적으로 구현한 private int removeCyc() 메소드, public boolean search(int key) 메소드를 제외한 나머지 메소드들은 교재의 리스팅 코드를 참고하였다.
toString() 메소드는 키값과 함께 balance factor(hl – hr)를 같이 출력하도록 수정해야 하므로 int형 변수 bf에 this.left.height – this.right.height 결과 값을 가지도록 하고, left,right,key 값과 함께 return 되도록 하였다.

```
public AVLTree remove(int key){         // AVL 트리 삭제 메소드
        if ( this.search(key) == false ) return NIL;       // remove 하려는 것이 존재하지 않을 경우

        if ( key == this.key ) {  // key가 현재 key일 경우
                if( left == NIL ) {         // 오른쪽 자식만 존재할 경우
                        if( right != NIL) {
                                this.key = right.key;

                                if( right.left != NIL ) {
                                        left = right.left;
                                }
                                else if( right.right != NIL ) {
                                        right = right.right;
                                }
                                else this.right = NIL;
                                rebalance();      // rebalance 메소드 호출
                        }
                        else return NIL;// reaf 노드이면 NIL return
                }
                else if( left != NIL ) {    // 왼쪽 자식만 존재할 경우
                        if( right == NIL ) {
                                this.key = left.key;
                                        if( left.left != NIL ) {
                                        left = left.left;
                                }
                                else if( left.right != NIL ) {
                                        right = left.right;
                                }
                                else this.left = NIL;
                                rebalance();      // rebalance 메소드 호출
                        }
                        else this.key = removeCyc();
                }
                return this;
        }
        else if ( key < this.key ){         // 삭제하려는 key가 현재 key보다 작을 경우
                left = left.remove(key);  // 왼쪽자식 remove
        }
        else if( key > this.key ) {
                right = right.remove(key);        // 오른쪽자식 remove
        }
        rebalance();      // rebalance 메소드 호출
        height = 1 + Math.max( left.height, right.height );        // 높이 업데이트

        return this;
}

private int removeCyc() {         // 순환적인 remove
        AVLTree parent = this; // 부모 노드
        if( right.left != NIL && right.right == NIL ) {
                parent = parent.right;   // parent 노드를 후속자의 부모 노드로 이동시킴
                right = right.left;         // 후속자를 왼쪽 자식으로 이동

                for( ; right.left != NIL; ) {         // 후속자를 마지막 왼쪽 자식으로 만듦
```

```
                        right = right.left;
                        parent = parent.left;
                }

                this.key = right.key;
                parent.left = NIL;          // 후속자를 NIL로 설정
        }
        else if( right.left != NIL && right.right != NIL ) {
                parent = parent.right;   // parent 노드를 후속자의 부모 노드로
이동시킴
                right = right.left;         // 후속자를 왼쪽 자식으로 이동

                for( ; right.left != NIL; ) {       // 후속자를 마지막 왼쪽 자식으로
만듦
                        right = right.left;
                        parent = parent.left;
                }

                this.key = right.key;
                parent.left.key = right.right.key;
                parent.left = right.right;
        }
        else if( (right.left == NIL && right.right == NIL) || (right.left == NIL &&
right.right != NIL) ){
                        // 후속자에 왼쪽 자식이 없을 경우
                this.key = right.key;
                this.right = right.right;
                return this.key;
        }
        return this.key;
}
```

remove() 메소드에서는 삭제하려는 값이 존재하지 않으면 NIL을 return 해주고,
만약 key값이 this.key와 일치하면 다시 조건 검사를 한다.

1) left가 NIL 이고, right 가 NIL이 아니면(오른쪽 자식만 존재하면) this.key를 right.key로 바꾼 후, 조건 검사를 통해 right.left가 NIL 이 아니면 left에 right.left 값을 저장한다.
반대로 right.right가 NIL 이 아니면 right에 right.right 값을 저장하고, right.left 가 NIL이거나 right.right가 NIL이면 this.right 는 NIL이다.
그 후에 rebalance() 메소드 호출을 통해 트리를 재정렬한다. 그리고 reaf 노드이면 NIL을 return 해준다.

2) left가 NIL이 아니고, right가 NIL이면 this.key를 left.key로 바꿔주고 조건검사를 한다.
left.left가 NIL이 아니면 left는 left.left가 되고, left.right가 NIL이 아니면 right는 left.right가 된다.
둘 중 하나라도 NIL이면, this.left는 NIL이 되고, 트리 재정렬을 한다.
그리고 left가 NIL이 아니고 right가 NIL이 아니면 removeCyc() 메소드를 호출하여 this.key에 저장한다.

3) key가 this.key보다 작을 경우, left = left.remove(key)로 왼쪽 자식이 삭제된다.

4) key가 this.key보다 클 경우, right = right.remove(key)로 오른쪽 자식이 삭제된다.

모든 조건 검사가 끝나면, rebalance() 메소드로 트리를 재정렬해주고, grow() 메소드에서 사용했던 것처럼 height를 업데이트해준다.

removeCyc() 메소드는 remove() 메소드를 순환적으로 하기 위해 추가적으로 구현된 메소드이다.
부모 노드 parent 객체를 생성해주고, 조건검사를 거쳐 this.key를 return 해준다.

1) right.left가 NIL이 아니고, right.right가 NIL이면 parent는 parent.right가 된다.
즉, parent를 후속자의 부모 노드로 이동시키고, right에 right.left를 저장시킴으로써 후속자를 왼쪽 자식으로 이동시킨다.
후속자를 왼쪽 자식의 마지막 노드로 만들기 위해 반복을 통해 right에는 right.left를, parent에는 parent.left를 저장시킨다.

this.key는 right.key가 되고 parent의 left는 NIL이 된다.

2) right.left가 NIL이 아니고 right.right 가 NIL이 아닌 경우는 위의 조건과 비슷한 과정을 거치지만 parent의 left를 NIL로 설정하지 않고 parent.left.key는 right.right.key로, parent.left는 right.right가 된다.

3) right.left가 NIL이고 right.right가 NIL인 경우이거나, right.left가 NIL이고 right.right는 NIL이 아닌 경우, this.key는 right.key가 되고 this.right는 right.right가 되고 this.key를 return 해준다.

```
public boolean search(int key){ // AVL 트리 검색 메소드
        if( this == NIL ) return false; // search fail
        else {
                if( key == this.key ) return true;        // search success
                if( key < this.key ) return left.search(key);      // 찾은 key가
현재 key보다 작을 경우 left search
                if( key > this.key ) return right.search(key);     // 찾은 key가
현재 key보다 클 경우 right search
        }
        return true;
        }
```

search() 메소드에서는 this가 NIL일 경우 false를 return해서 검색에 실패했음을 알리고 this가 NIL이 아닐 경우, 조건에 따라 검색을 달리 한다.
1) key가 this.key와 일치하면 true를 return.
2) key가 this.key보다 작을 경우, left.search(key)로 왼쪽을 검색.
3) key가 this.key보다 클 경우, right.search(key)로 오른쪽을 검색.

2) TestAVLTree

```
public class TestAVLTree {
        public static void main(String args[]) {
        AVLTree avl = new AVLTree(1);
        avl.grow(2);
        avl.grow(3);
        avl.grow(4);
        avl.grow(5);
        avl.grow(6);
        avl.grow(7);
        avl.grow(8);
        avl.grow(9);
        avl.grow(10);
        System.out.println("***** Search *****");
        System.out.println("1 검색결과 : " + avl.search(1));
        System.out.println("3 검색결과 : " + avl.search(3));
        System.out.println("5 검색결과 : " + avl.search(5));
        System.out.println("11 검색결과 : " + avl.search(11));
        System.out.println();
        // Remove
        System.out.println("***** Remove *****");
        avl.remove(1);
        System.out.println("1 삭제");
        System.out.println(avl);
        avl.remove(3);
        System.out.println("3 삭제");
        System.out.println(avl);
        avl.remove(5);
        System.out.println("5 삭제");
        System.out.println(avl);
        avl.remove(11);
        System.out.println("11 삭제");
```

```
        System.out.println(avl);
        }
}
```

2. 실행 결과 화면

```
<terminated> TestAVLTree [Java Application] C:₩Program Files₩Java₩jre1.8.0_1
***** Search *****
1 검색결과 : true
3 검색결과 : true
5 검색결과 : true
11 검색결과 : false

***** Remove *****
1 삭제
(2,-1) (3,0) (4,-1) (5,0) (6,0) (7,0) (8,0) (9,-1) (10,0)
3 삭제
(2,0) (4,-1) (5,0) (6,0) (7,0) (8,1) (9,-1) (10,0)
5 삭제
(2,0) (4,-1) (6,-1) (7,0) (8,1) (9,-1) (10,0)
11 삭제
(2,0) (4,-1) (6,-1) (7,0) (8,1) (9,-1) (10,0)
```