자료구조 실습 보고서

[제 13주]

제출일: 06.19
학번/이름: 201603867/조성환

1. 프로그램 설명서

1) 주요 알고리즘/ 자료구조/ 기타

DictionaryByBST를 이용해 이진 트리를 검색, callback과 iterator를 이용해 출력

2) 함수 설명서

run()
함수의 실질적인 시작 및 전체적인 제어
dictionary, visitdelegate, List를 set함
dictionary를 추가, 이를 보여줌
call back과 iterator를 통해 보여줌
list를 set함
list를 삭제하면서 보여줌

setList()
run에서는 datagenerator.randomListWithoutDuplication을 통해 set함

addToDictionaryAndShowShape()
삽입하기 전 모습을 보여준 후 list에서 하나씩 가져와 dictionary에 추가한 후 해당 위치에 맞게 출력을 함.

showDictionary()
현재 모드의 제목을 출력한 후 scanInReverseOfSortedOrder()을 통해 출력을 함. 이를 이용하는 이유는 저장은 inorder이며 left->right 식으로 저장을 함.
출력을 위해서 right-> left로 하나씩 가져와 출력하기 위해 reverse를 이용

showDictionaryInSortedORderByIterator()
iterator를 이용해 순서대로 하나씩 보여줌

BST
removeRightMostElementOfLeftSubTree()
left를 통해 left의 root를 가져온다.
만약 이 leftRoot의 right가 없다면 이 자체가 삭제할 element이므로 leftRoot의 left를 root의 left로 set한 후 해당 element를 리턴함.
leftRoot의 right가 있다면 parentOfRightMost를 leftOfRoot로, rightMost를 leftOfRoot.right로 set, rightMost.right()가 없을 때 까지 parent를 rightMost로 설정하면 마지막 parent는 right를 하나만 가지고 있는 노드임, 이 parent의 right를 rightMost로 설정, rightMost의 left가 있다면 paret의 left를 rightMost의 left로, left가 없다면 null로 설정함. rightMost를 리턴함
결국 rightMost는 트리의 맨 오른쪽 노드를 의미함.

DataGenerator
randomListWithoutDuplication()
randomList를 생성함 이전 randomList와 같이 list를 만든 후 random으로 순서를 바꿈 이를 통해 key의 값에 변화를 줌

3) 종합 설명서

이 프로그램은 bst를 이용해 이진트리를 만들고 삭제함. 다만 저번주 과제에서는 visit을 구현하지 않았음. visit을 할 때 bst와 사용자(appcontroller)가 서로 간섭을 하면 안되는데, 왜냐하면 설계자는 사용자가 이를 어떻게 할지 알 수 없고 만약 설계자 마음대로 출력 및 제어를 한다면 mvc모드에 어긋남. 이는 효율적인 코드가 되지 못함. 이를 해결하기 위해 callback을 이용해 출력을 하도록 설계.

bst를 통해 dictionary의 이진트리를 만듬. add할 때 마다 현재 상태를 scan을 통해 보여줌 다 보여준 뒤 callback과 iterator를 통해 전체 모습을 출력, 마지막으로 삭제를 하면서 모습을 보여주는 것으로 프로그램은 종료

2. 프로그램 장단점/ 특이점 분석

앞서 서술했다시피 model은 controller의 역할을 할 수 없음. 예를 들어 노트북에서 a를 눌렀는데 알아서 단어인 apple을 작성하는 것과 마찬가지. 또한 사용자가 직접 model을 제어할 수 없음. 이런 모순상황을 해결하기 위한 기술이 바로 callback. 이 callback을 넣음으로써 mvc모델을 위반하지 않고 controller는 원하는 visit을 구현할 수 있음. 이것이 바로 이 프로그램의 장점

3. 실행 결과 분석

1) 입력과 출력

입력은 없다.

2) 결과 분석

```
"C:₩Program Files₩Java₩jdk1.8.0_201₩bin₩java.exe" ...
<<< 이진검색트리로 구현된 사전에서의 삽입과 삭제 >>>

[삽입 과정에서의 이진검색트리 사전의 변화]
> 삽입을 시작하기 전의이진검색트리 사전:
  Empty

> Key=8 (Object=0) 원소를 삽입한 후의이진검색트리 사전:
 Root:   8 ( 0)

> Key=3 (Object=1) 원소를 삽입한 후의이진검색트리 사전:
 Root:   8 ( 0)
                3 ( 1)

> Key=9 (Object=2) 원소를 삽입한 후의이진검색트리 사전:
                9 ( 2)
 Root:   8 ( 0)
                3 ( 1)

> Key=5 (Object=3) 원소를 삽입한 후의이진검색트리 사전:
                9 ( 2)
 Root:   8 ( 0)
                      5 ( 3)
                3 ( 1)

> Key=2 (Object=4) 원소를 삽입한 후의이진검색트리 사전:
                9 ( 2)
 Root:   8 ( 0)
                      5 ( 3)
                3 ( 1)
                      2 ( 4)
```

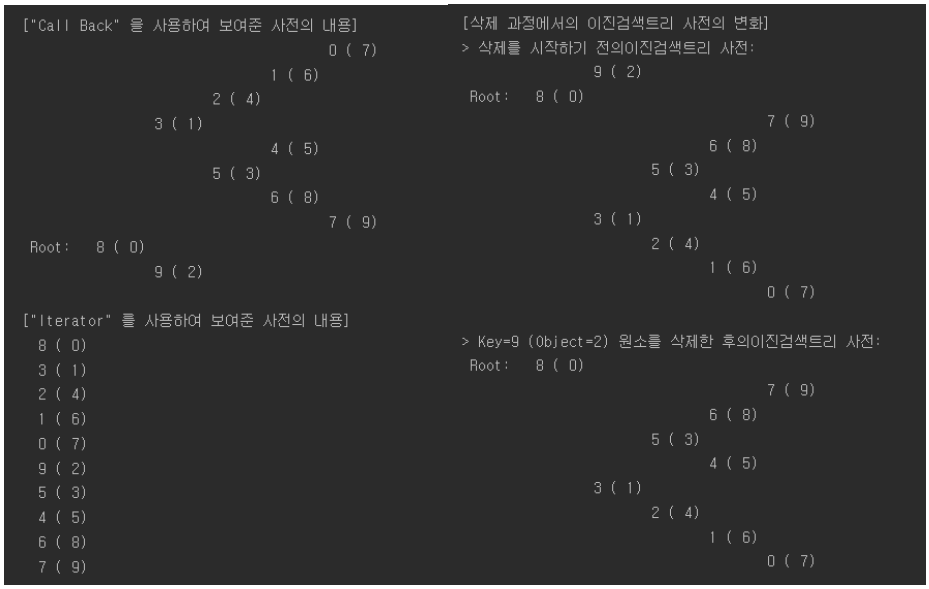
```
> Key=4 (Object=5) 원소를 삽입한 후의이진검색트리 사전:
                9 ( 2)
 Root:   8 ( 0)
                      5 ( 3)
                            4 ( 5)
                3 ( 1)
                      2 ( 4)

> Key=1 (Object=6) 원소를 삽입한 후의이진검색트리 사전:
                9 ( 2)
 Root:   8 ( 0)
                      5 ( 3)
                            4 ( 5)
                3 ( 1)
                      2 ( 4)
                            1 ( 6)

> Key=0 (Object=7) 원소를 삽입한 후의이진검색트리 사전:
                9 ( 2)
 Root:   8 ( 0)
                      5 ( 3)
                            4 ( 5)
                3 ( 1)
                      2 ( 4)
                            1 ( 6)
                                  0 ( 7)
```

```
> Key=6 (Object=8) 원소를 삽입한 후의이진검색트리 사전:
                9 ( 2)
 Root:   8 ( 0)
                                  6 ( 8)
                      5 ( 3)
                                  4 ( 5)
                3 ( 1)
                      2 ( 4)
                                  1 ( 6)
                                        0 ( 7)

> Key=7 (Object=9) 원소를 삽입한 후의이진검색트리 사전:
                9 ( 2)
 Root:   8 ( 0)
                                        7 ( 9)
                                  6 ( 8)
                      5 ( 3)
                                  4 ( 5)
                3 ( 1)
                      2 ( 4)
                                  1 ( 6)
                                        0 ( 7)
```



4. 소스 코드

AppController

```java
package controller;

import model.*;
import view.AppView;

public class AppController implements VisitDelegate<Integer, Integer> {
    // Constants
    private static final int  DEFAULT_DATA_SIZE = 10 ;
    // Private instance variables
    private Dictionary<Integer,Integer> _dictionary ;
    private Integer[]  _list ;

    public Dictionary<Integer, Integer> dictionary() { return _dictionary; }
    public void setDictionary(Dictionary<Integer, Integer> newDictionary) { this._dictionary = newDictionary; }
    public Integer[] list() { return _list; }
    public void setList(Integer[] newList) { this._list = newList; }

    public void  run() {
        AppView.outputLine ( aString: "<<< 이진검색트리로 구현된 사전에서의 삽입과 삭제 >>>") ;
        AppView.outputLine ( aString: "") ;

        this.setDictionary (new DictionaryByBinarySearchTree<Integer,Integer>()) ;
        this.dictionary().setVisitDelegate (this) ;
        this.setList (DataGenerator.randomListWithoutDuplication(DEFAULT_DATA_SIZE)) ;
        this.addToDictionaryAndShowShape() ;
        this.showDictionaryInSortedOrderByCallBack() ;
        this.showDictionaryInSortedOrderByIterator() ;
        this.setList (DataGenerator.randomListWithoutDuplication(DEFAULT_DATA_SIZE)) ;
        this.removeFromDictionaryAndShowShape() ;
        AppView.outputLine ( aString: "<<< 종료 >>>") ;

    }

    private void showDictionary (String aTitlePrefix) {
```

```
43          AppView.outputLine
44                  ( aString: "> " + aTitlePrefix + "이진검색트리 사전:") ;
45          if ( this.dictionary().isEmpty() ) {
46              AppView.outputLine ( aString: "  Empty") ;
47          } else {
48              this.dictionary().scanInReverseOfSortedOrder() ;
49          } AppView.outputLine( aString: "") ;
50      }
51
52      private void addToDictionaryAndShowShape() {
53          AppView.outputLine
54                  ( aString: "[삽입 과정에서의 이진검색트리 사전의 변화]") ;
55          this.showDictionary ( aTitlePrefix: "삽입을 시작하기 전의") ;
56          for ( int i = 0 ; i < this.list().length ; i++ ) {
57              Integer currentKey = this.list()[i] ;
58              Integer currentObj = Integer.valueOf(i) ;
59              this.dictionary().addKeyAndObject (currentKey, currentObj) ;
60              this.showDictionary
61                      ( String.format ("Key=%d (Object=%d) 원소를 삽입한 후의"
62                              , currentKey, currentObj) ) ;
63          }
64      }
65
66      private void removeFromDictionaryAndShowShape() {
67          AppView.outputLine
68                  ( aString: "[삭제 과정에서의 이진검색트리 사전의 변화]") ;
69          this.showDictionary ( aTitlePrefix: "삭제를 시작하기 전의") ;
70          for ( int i = 0 ; i < this.list().length ; i++ ) {
71              Integer currentKey = this.list()[i] ;
72              Integer currentObj = this.dictionary().removeObjectForKey (currentKey) ;
73              this.showDictionary
74                      ( String.format("Key=%d (Object=%d) 원소를 삭제한 후의",
75                              currentKey, currentObj) ) ;
```

```
                                currentKey, currentObj) ) ;
        }
    }

    @Override
    public void  visitForSortedOrder
            (DictionaryElement<Integer,Integer> anElement, int aLevel) {
        AppView.outputLine
                ( String.format("%3d (%2d)", anElement.key(), anElement.object() ) ) ;
    }

    @Override public void  visitForReverseOfSortedOrder
            (DictionaryElement<Integer,Integer> anElement, int aLevel) {
        if ( aLevel == 1 ) {
            AppView.output (String.format("%7s", "Root: ") );
        } else { AppView.output (String.format("%7s", "")) ;
        }
        for (int i = 1; i < aLevel ; i++ ) {
            AppView.output (String.format("%7s", ""));
        }
        AppView.outputLine
                ( String.format("%3d (%2d)", anElement.key(), anElement.object() ) ) ;
    }

    private void showDictionaryInSortedOrderByCallBack () {
        AppView.outputLine
                ( aString: "[#"Call Back#" 을 사용하여 보여준 사전의 내용]") ;
        this.dictionary().scanInSortedOrder() ;
        AppView.outputLine ( aString: "") ;
    }
    private void showDictionaryInSortedOrderByIterator () {
        AppView.outputLine
                ( aString: "[#"Iterator#" 를 사용하여 보여준 사전의 내용]") ;
        Iterator<DictionaryElement<Integer,Integer>> iterator = this.dictionary().iterator() ;
        while (iterator.hasNext() ) {
            DictionaryElement<Integer,Integer> dictionaryElement = iterator.next() ;
            AppView.outputLine
                    ( String.format("%3d (%2d)", dictionaryElement.key(), dictionaryElement.object()) ) ;
        }
        AppView.outputLine ( aString: "") ;
    }
}
```

BST

```
@Override public Obj  removeObjectForKey (Key aKey) {
    if ( aKey == null ) { return null ; }
    if ( this.root() == null ) { return null ; }
    if ( aKey.compareTo(this.root().element().key()) == 0 ) {
        //  this.root() is the node to be removed.
        Obj objectForRemove = this.root().element().object();
        if ((this.root().left() == null) && (this.root().right() == null)) {
            this.setRoot(null); // Root-only tree
        } else if (this.root().left() == null) {
            this.setRoot(this.root().right());
        } else if (this.root().right() == null) {
            this.setRoot(this.root().left());
        } else {
            this.root().setElement(this.removeRightMostElementOfLeftSubTree(this.root()));
        }
        this.setSize(this.size() - 1);
        return objectForRemove;
    }
    BinaryNode<DictionaryElement<Key,Obj>> current = this.root() ;
    BinaryNode<DictionaryElement<Key,Obj>> child = null ;
    do {
        if ( aKey.compareTo(current.element().key()) < 0 ) {
            child = current.left() ;
            if ( child == null ) {
                return null ; // "aKey" does not exist//
            }
            if ( aKey.compareTo(child.element().key()) == 0 ) {
                // Found. "child" is to be removed.
                Obj  objectForRemove = child.element().object() ;
                if ( child.left() == null && child.right() == null ) {
                    current.setLeft(null) ; // "child" is a leaf.
                } else if (child.left() == null) {
                    // "child" has no left.
                    current.setLeft(child.right()) ;
```

```
                    } else if (child.right() == null) {
                        // "child has no right.
                        current.setLeft(child.left()) ;
                    } else  {
                        child.setElement (this.removeRightMostElementOfLeftSubTree(child)) ;
                    }
                    this.setSize (this.size()-1) ;
                    return  objectForRemove;
                }
            } else {
                child = current.right() ;
                if ( child == null ) {
                    return null ; // "aKey" does not exist.
                } if ( aKey.compareTo(child.element().key()) == 0 ) {
                    // Found. "child" is to be removed.
                    Obj  objectForRemove = child.element().object() ;
                    if ( child.left() == null && child.right() == null ) {
                        current.setRight(null) ;
                    } else if (child.left() == null) {
                        current.setRight(child.right()) ;
                    } else if (child.right() == null) {
                        current.setRight(child.left()) ;
                    } else  {
                        child.setElement (this.removeRightMostElementOfLeftSubTree(child)) ;
                    }
                    this.setSize (this.size()-1) ;
                    return  objectForRemove;
                }
            }
            current = child;
        } while (true) ;
    } // End of "removeObjectForKey()"
```

```java
    private DictionaryElement<Key,Obj> removeRightMostElementOfLeftSubTree
            (BinaryNode<DictionaryElement<Key,Obj>> root) {
        // At this point, "root" has non-empty left subtree.
        BinaryNode<DictionaryElement<Key,Obj>> leftOfRoot = root.left() ;
        if ( leftOfRoot.right() == null ) {
            // "leftOfRoot" has no right subtree, so "leftOfRoot" itself is the rightmost node.
            root.setLeft (leftOfRoot.left()) ;
            return  leftOfRoot.element() ;
        } else {
            // There exist at least one node in the right subtree of the left of the root.
            BinaryNode<DictionaryElement<Key,Obj>> parentOfRightMost = leftOfRoot ;
            BinaryNode<DictionaryElement<Key,Obj>> rightMost = parentOfRightMost.right() ;
            while ( rightMost.right() != null ) {
                parentOfRightMost = rightMost ;
            } rightMost = rightMost.right() ;
        // We have found the right-most node so that it is owned by "rightMost".
        parentOfRightMost.setRight (rightMost.left()) ;
        return  rightMost.element() ;
        }
    } // End of "removeRightMostElementOfLeftSubTree()"

    @Override
    public void scanInSortedOrder() { this.inorderRecursively(this.root(),  aLevel: 1); }

    private void inorderRecursively
            (BinaryNode<DictionaryElement<Key,Obj>> aRootOfSubtree, int aLevel){
        if (aRootOfSubtree!= null){
            this.inorderRecursively(aRootOfSubtree.left(),  aLevel: aLevel+1);
            DictionaryElement<Key,Obj> visitedElement= aRootOfSubtree.element();
            this.visitDelegate().visitForReverseOfSortedOrder(visitedElement, aLevel);
            this.inorderRecursively(aRootOfSubtree.right(),  aLevel: aLevel+1);
        }
    }
```

```java
    public void scanInReverseOfSortedOrder() { this.reverseOfInorderRecursively(this.root(),  aLevel: 1); }

    private void reverseOfInorderRecursively
            (BinaryNode<DictionaryElement<Key,Obj>> aRootOfSubtree, int aLevel){
        if (aRootOfSubtree!= null){
            this.reverseOfInorderRecursively(aRootOfSubtree.right(),  aLevel: aLevel+1);
            DictionaryElement<Key,Obj> visitedElement= aRootOfSubtree.element();
            this.visitDelegate().visitForReverseOfSortedOrder(visitedElement, aLevel);
            this.reverseOfInorderRecursively(aRootOfSubtree.left(),  aLevel: aLevel+1);
        }
    }
```

Dictionary

```java
package model;

public abstract class Dictionary<Key extends Comparable<Key>, Obj> {
    //var
    private int _size;
    public int size () { return this._size; }
    protected void setSize(int newSize) { this._size = newSize; }

    //constructor
    public Dictionary() { this.setSize(0); }

    public boolean isEmpty() { return (this.size() == 0); }

    public abstract boolean isFull();
    public abstract boolean keyDoesExist(Key aKey);
    public abstract Obj objectForKey(Key aKey);
    public abstract boolean addKeyAndObject(Key aKey, Obj anObject);
    public abstract Obj removeObjectForKey(Key aKey);
    public abstract void clear();
    public abstract void scanInSortedOrder();
    public abstract void scanInReverseOfSortedOrder();

    private VisitDelegate<Key,Obj> _visitDelegate;

    public VisitDelegate<Key, Obj> visitDelegate() {
        return _visitDelegate;
    }
    public void setVisitDelegate(VisitDelegate<Key, Obj> newVisitDelegate) {
        this._visitDelegate = newVisitDelegate;
    }

    public abstract Iterator<DictionaryElement<Key, Obj>> iterator();
}
```

visit

```java
package model;

public interface  VisitDelegate<Key extends Comparable<Key>, Obj> {
    public void visitForSortedOrder
            (DictionaryElement<Key,Obj> anElement, int aLevel) ;
    public void visitForReverseOfSortedOrder
            (DictionaryElement<Key,Obj> anElement, int aLevel) ;
}
```