자료구조: 2022년 1학기 [강의]

Dictionary (1)

강지훈
jhkang@cnu.ac.kr
충남대학교 컴퓨터융합학부

사전 (Dictionary) 이란?

개요

- 사전이란?
  - (단어, 설명) 의 쌍들을 모아놓은 것.
  - 단어가 주어지면 그 단어의 설명을 찾는다.

- 또 다른 예는?
  - (학번, 성적)

- 일반화 하면?
  - (key, object) 의 쌍을 모아 놓은 집합.
  - Key 가 주어지면, 그에 해당하는 object 를 찾는다.
  - key 값은 유일하다.

Class

"Dictionary<Key,Obj>"

Dictionary<Key,Obj> 의 공개함수

Dictionary<Key,Obj> 객체 사용법

- public Dictionary () ;

- public boolean isEmpty ( ) ;
- public boolean isFull () ;
- public int size () ;

- public boolean keyDoesExist (Key aKey) ;
  - 주어진 aKey 가 사전에 존재하는지 여부를 확인한다
- public Obj objectForKey (Key aKey) ;
  - 주어진 aKey 를 갖는 객체를 얻는다

- public boolean addKeyAndObject (Key aKey, Obj anObject) ;
  - <aKey, anObject> 쌍을 사전에 추가한다.
- public Obj removeObjectForKey (Key aKey) ;
  - 주어진 aKey와 쌍을 이루는 객체를 함께 삭제한다.
- public boolean replaceObjectForKey (Key aKey, Obj objectForReplace) ;
  - 주어진 aKey 와 쌍을 이루는 객체를 새로운 objectForReplace 로 대체한다.
- public void clear ( ) ;
  - 사전의 모든 <key, object> 쌍을 삭제하여, 사전을 비운다.

Dictionary 의 구현은?

Dictionary 를 구현하는 방법

- List
  - ArrayList
  - LinkedList
- Tree
  - **Binary Search Tree**
  - Multi-way Search Tree
- Hash Table

이진검색트리
(Binary Search Tree)

이진 검색 트리

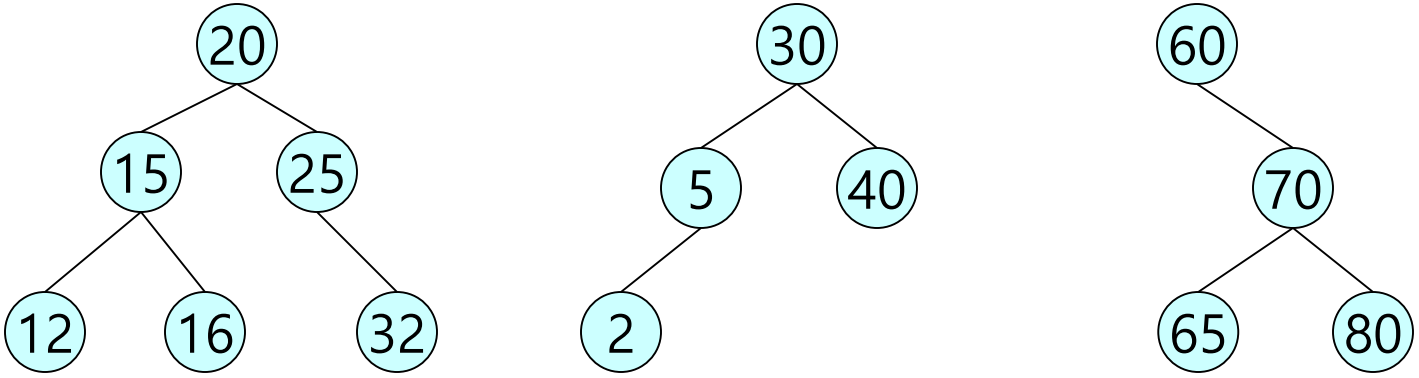
- 임의의 원소의 삽입/삭제/검색

- 시간복잡도 비교

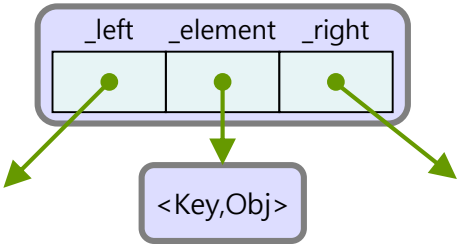
	정렬되지 않은 배열	정렬된 배열	이진검색트리
삽입	O(1)	O(n)	O(log n)
삭제	O(n)	O(n)	O(log n)
검색	O(n)	O(log n)	O(log n)

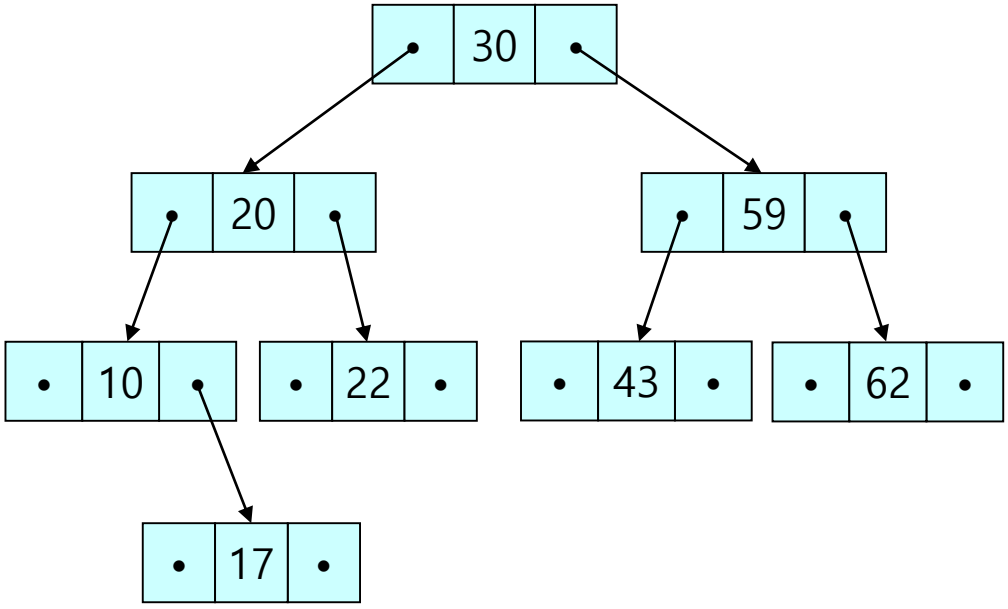
이진검색트리 (Binary Search Tree)

- 이진검색트리는 이진트리이다. 비어 있을 수 있으며, 만일 비어 있지 않다면 다음의 조건을 만족해야 한다:
  1. 모든 원소는 키를 가지고 있으며, 어떠한 원소도 동일한 키를 가지고 있지 않다. 즉, 키는 유일(unique) 하다.
  2. 트리의 루트의 키는 비어있지 않은 왼쪽 부트리에 있는 키들보다 크다.
  3. 트리의 루트의 키는 비어있지 않은 오른쪽 부트리에 있는 키들보다 작다.
  4. 왼쪽 부트리와 오른쪽 부트리는 또한 이진검색트리이다.



연결 체인을 이용한 이진 검색 트리





검색 알고리즘: [재귀적으로]

```
// 주어진 Key 를 갖는 노드를 찾는다. 없으면 null 을 얻는다
BinaryNode<E>  search (BinaryNode<E> currentRoot, Key keyForSearch)
{
    if ( currentRoot != null ) {
        if ( keyForSearch.compareTo(currentRoot.element().key()) == 0 ) {
            return  currentRoot ;
        }
        else if ( keyForSearch.compareTo(currentRoot.element().key()) < 0 ) {
            return  search (currentRoot.left(), keyForSearch) ;
        }
        else {
            return  search (currentRoot.right(), keyForSearch) ;
        }
     }
    else {
        return  null ;
    }
}
```

- 검색 시간 복잡도: O(h)
  - h: 이진검색트리의 높이
  - 높이와 노드 수와의 관계는?

```
public class DictionaryElement<Key, Obj> {
        private Key _key ;
        private Obj _object ;
        // Getter/Setter
        public Key   key() {...} ;
        public void  setKey() {...} ;
        ..... // 다른  공개함수들
}

public class  BinaryNode<E> {
        private E                   _element ;
        private BinaryNode<E>   _left ;
        private BinaryNode<E>   _right ;

        public E                    element() {...} ;
        public void                 setElement() {...} ;
        public BinaryNode<E>    left() {...} ;
        public void                 setLeft() {...} ;
        public BinaryNode<E>    right() {...} ;
        public void                 setRight() {...} ;
}
```

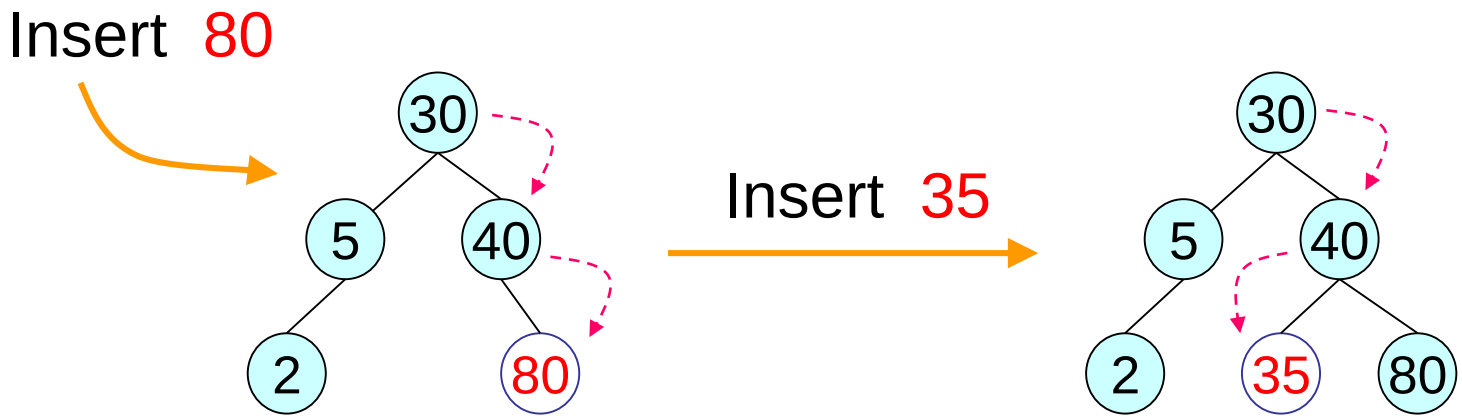
검색 알고리즘: [반복적으로]

```
// 주어진 Key 를 갖는 노드를 찾는다. 없으면 null 을 얻는다
BinaryNode<E>  search (BinaryNode<E> currentRoot, Key keyForSearch)
{
      while ( currentRoot != null ) {
            if ( keyForSearch.compareTo(currentRoot.element().key()) == 0 ) {
                  return  currentRoot ;
            }
            else if ( keyForSearch.compareTo(currentRoot.element().key()) < 0 ) {
                  currentRoot = currentRoot.left() ;
            }
            else {
                  currentRoot = currentRoot.right() ;
            }
       }
      return  null ;
}
```

- 검색 시간 복잡도: O(h)
  - h: 이진검색트리의 높이
  - 높이와 노드 수와의 관계는?

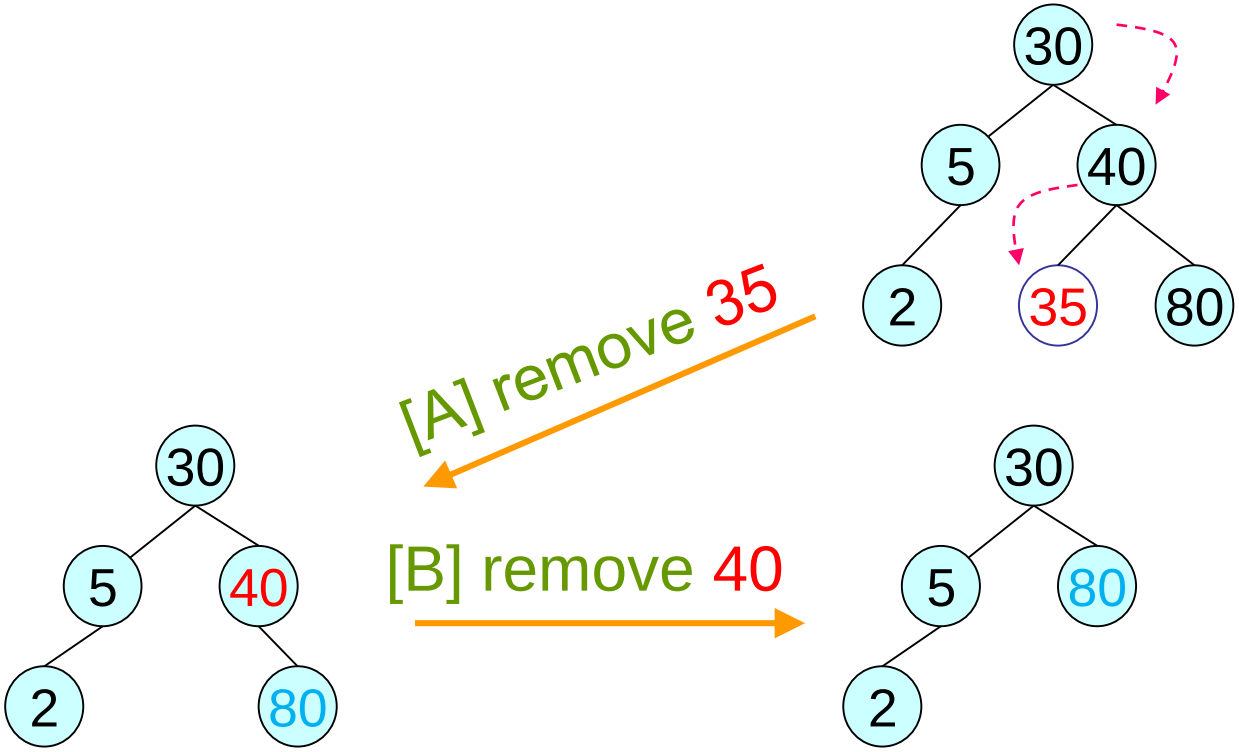
```
public class DictionaryElement<Key, Obj> {
        private Key _key ;
        private Obj _object ;
        // Getter/Setter
        public Key   key() {...} ;
        public void  setKey() {...} ;
        ..... // 다른  공개함수들
}

public class  BinaryNode<E> {
        private E                   _element ;
        private BinaryNode<E>    _left ;
        private BinaryNode<E>    _right ;

        public E                    element() {...} ;
        public void                 setElement() {...} ;
        public BinaryNode<E>     left() {...} ;
        public void                 setLeft() {...} ;
        public BinaryNode<E>     right() {...} ;
        public void                 setRight() {...} ;
}
```

삽입



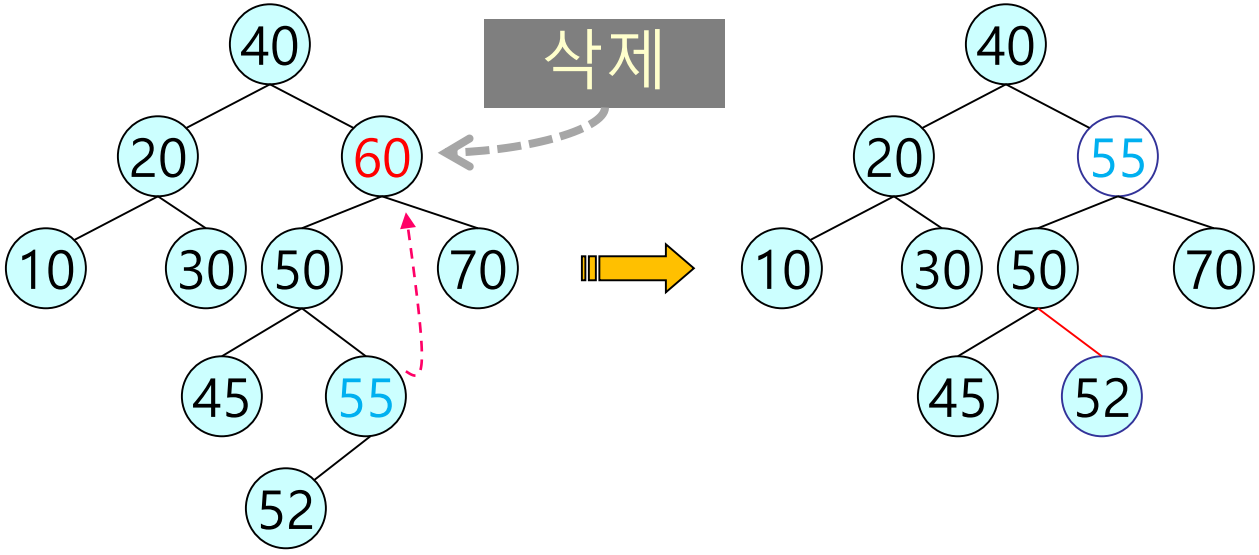
- 삽입의 시간복잡도: O(h)
  - h: 이진검색트리의 높이

삭제



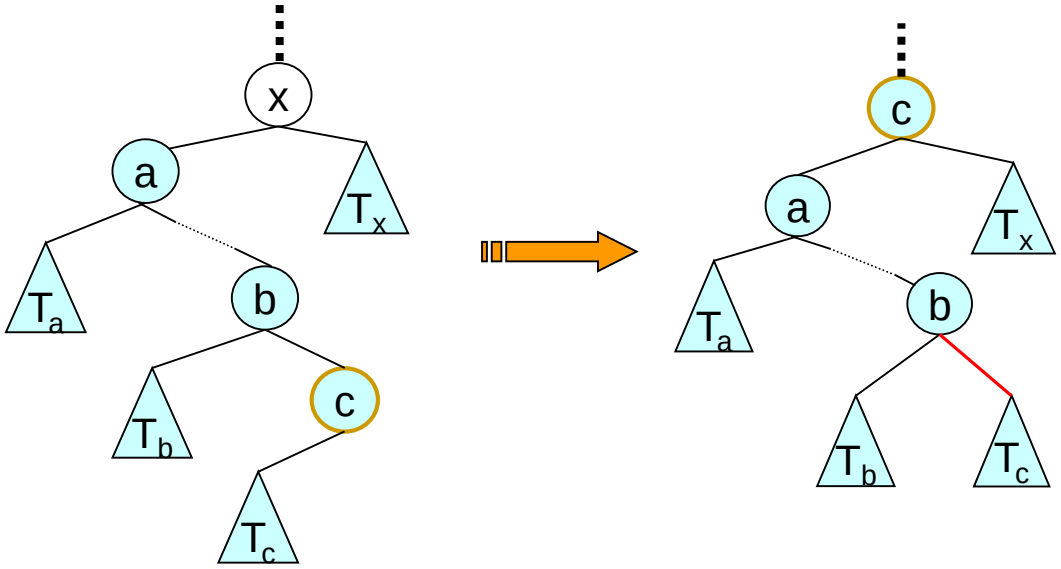
삭제

- 잎 노드: 바로 삭제
- 자식이 하나인 노드: 해당 노드는 삭제하고 그 자리에 자식 노드를 갖다 놓는다
- 자식이 둘인 노드: 잎 노드나 자식이 하나인 노드의 삭제의 문제로 변환



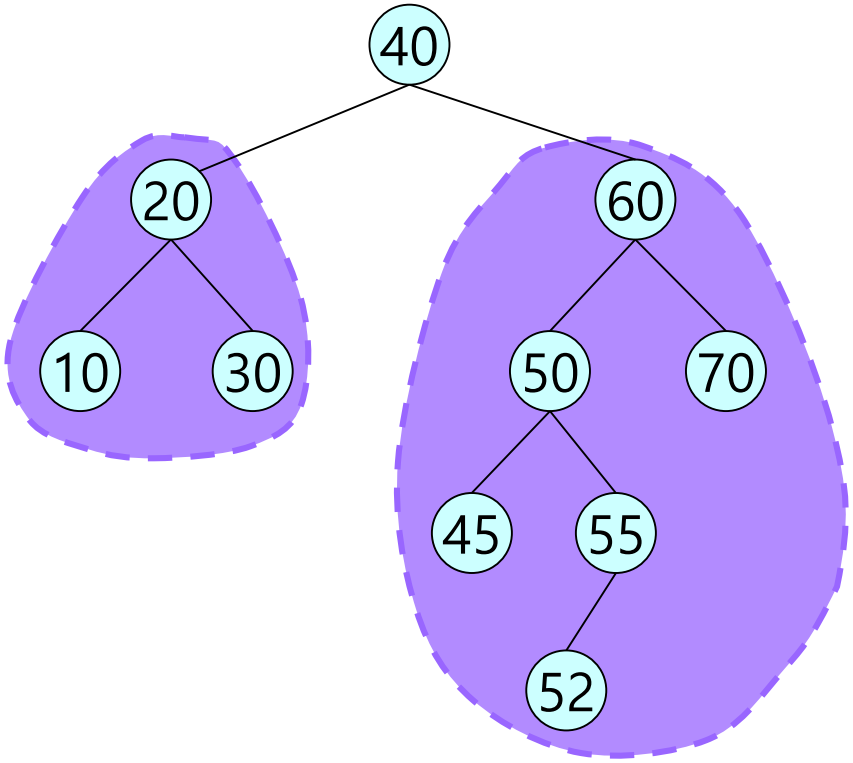
내부 노드 X 의 삭제

- X 의 왼쪽 부트리에서 가장 큰 값을 갖는 노드로 대체 (또는 X 의 오른쪽부트리에서 가장 작은 값을 갖는 노드로 대체)

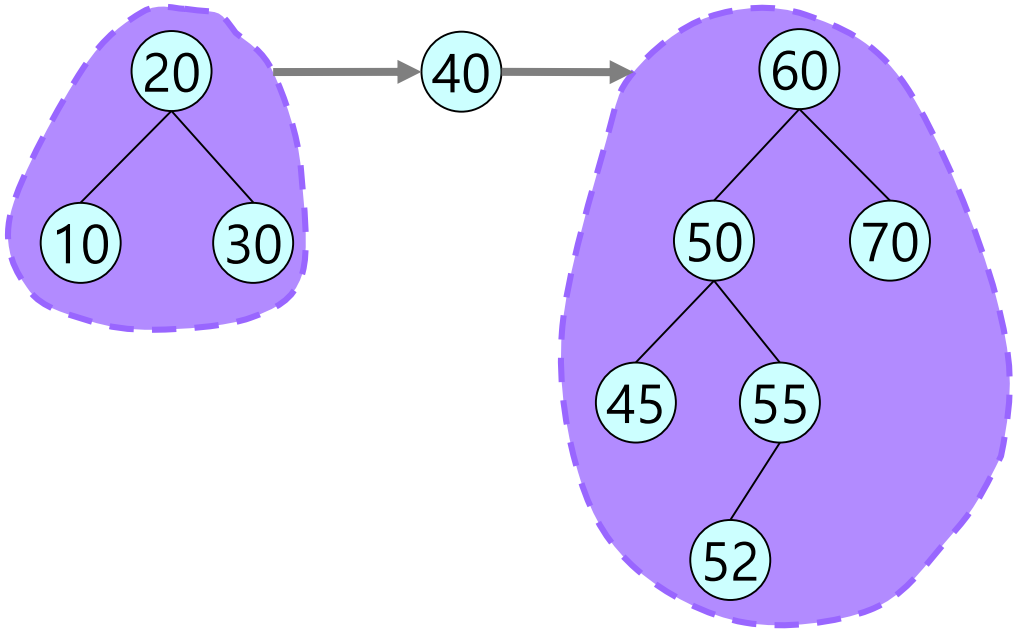


- 삭제의 시간복잡도: O(log n)
  - 트리의 높이가 h 일 때, O(h)
  - 삽입이나 삭제가 무작위로 발생하면, h = O(log n)

BST 에서 중위 탐색을 하면 ?



BST 에서 중위 탐색을 하면 ?



⇒ 키 값의 오름차순, 즉 정렬된 순서로 원소들을 방문

사전의 (Key, Object) 쌍을 위한
Class "DictionaryElement"

DictionaryElement: 공개함수

```
public class  DictionaryElement<Key,Obj>
{
      ……

      public  DictionaryElement<Key,Obj> () {…}
      public  DictionaryElement<Key,Obj> (Key givenKey, Obj  givenObject) {…}

      public Key     key () {…}
      public void    setKey (Key newKey) {…}

      public Obj     object () {…}
      public void    setObject (Obj newObject) {…}
}
```

DictionaryElement: 비공개 인스턴스 변수

```
public class  DictionaryElement<Key, Obj>
{
    // 비공개 인스턴스 변수
    private Key      _key ;
    private Obj      _object ;
```

DictionaryElement: Getter/Setter

```
public class  DictionaryElement<Key,Obj>
{
    ......
    public Key  key () ;
    {
        return  this._key ;
    }

    public void  setKey (Key newKey) ;
    {
         this._key =  newKey ;
    }

    public Obj  object () ;
    {
        return  this._object ;
    }

    public void  setObject (Obj newObject) ;
    {
         this._object =  newObject ;
    }
```

DictionaryElement: 생성자 [1]

```
public class  DictionaryElement<Key,Obj>
{
    ......
    public  DictionaryElement ()
    {
        this.setKey (null) ;
        this.setObject (null) ;
    }

    public  DictionaryElement (Key givenKey, Obj givenObject) ;
    {
        this.setKey (givenKey) ;
        this.setObject (givenObject) ;
    }
```

DictionaryElement: 생성자 [2]

```
public class  DictionaryElement<Key,Obj>
{
    ......
    public  DictionaryElement ()
    {
        this.setKey (null) ;
        this.setObject (null) ;
    }

    public  DictionaryElement (Key givenKey, Obj givenObject) ;
    {
        this.setKey (givenKey) ;
        this.setObject (givenObject) ;
    }
```

```
// 또는 간단하게
this (null, null) ;
```

사전 구현에 사용할 이진검색트리의 노드:

Class “BinaryNode<E>”

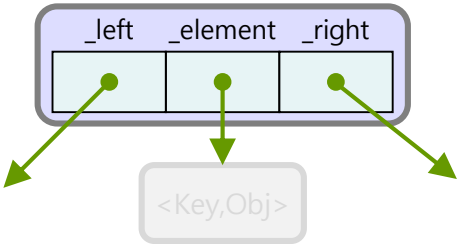
Class "BinaryNode<E>" 의 공개함수

```
pubic class  BinaryNode<E>
{
      ......

      public  BinaryNode () {...}
      public  BinaryNode
           (E givenElement, BinaryNode<E> givenLeft, BinaryNode<E> givenRight) {...}

      public E                    element () {...}
      public void                 setElement (E newElement) {...}

      public BinaryNode<E>        left () {...}
      public void                 setLeft (BinaryNode<E> newLeft) {...}

      public BinaryNode<E>        right () {...}
      public void                 setRight (BinaryNode<E> newRight) {...}
}
```

Class “BinaryNode<E>” 의 구현

BinaryNode: 비공개 인스턴스 변수

```
pubic class  BinaryNode<E>
{
      // 비공개 인스턴스 변수
      private E                    _element ;
      private BinaryNode<E>        _left ;
      private BinaryNode<E>        _right ;
```



BinaryNode: Getter/Setter

```
pubic class  BinaryNode<E>
{
    // 비공개 인스턴스 변수
    private E                   _element ;
    private BinaryNode<E>       _left ;
    private BinaryNode<E>       _right ;

    // Getter/Setter
    public E    element() {
        return  this._element ;
    }
    public void  setElement (E newElement) {
        this._element = newElement ;
    }

    public BinaryNode<E>  left () {
        return  this._left ;
    }
    public void  setLeft (BinaryNode<E> newLeft) {
        this._left = newLeft ;
    }

    public BinaryNode<E>  right () {
        return  this._right ;
    }
    public void  setRight (BinaryNode<E> newRight) {
        this._right = newRight ;
    }
```

BinaryNode: 생성자

```
public class  BinaryNode<E>
{
    ……

    // 생성자
    public  BinaryNode ()
    {
        this.setElement (null) ;
        this.setLeft (null) ;
        this.setRight (null) ;
    }

    public  BinaryNode
        (E givenElement, BinaryNode<E> givenLeft, BinaryNode<E> givenRight)
    {
        this.setElement (givenElement) ;
        this.setLeft (givenLeft) ;
        this.setRight (givenRight) ;
    }
```

```
// 또는 간단하게
this (null, null, null) ;
```

Class

“Dictionary<Key, Obj>”

Dictionary<Key,Obj> 의 공개함수

- Dictionary 객체 사용법
  - public Dictionary () ;
  - public boolean isEmpty ( ) ;
  - public boolean isFull () ;
  - public int size() ;
  - public boolean keyDoesExist (Key aKey) ;
  - public Obj objectForKey (Key aKey) ;
  - public boolean addKeyAndObject (Key aKey, Obj anObject) ;
  - public Obj removeObjectForKey (Key aKey) ;
  - public boolean replaceObjectForKey (Key aKey, Obj objectForReplace) ;
  - public void clear ( ) ;

Class

“Dictionary<Key, Obj>”의 구현

End of

"Dictionary (1)"