데이터구조입문

트리

김영민

2019. 6. 13

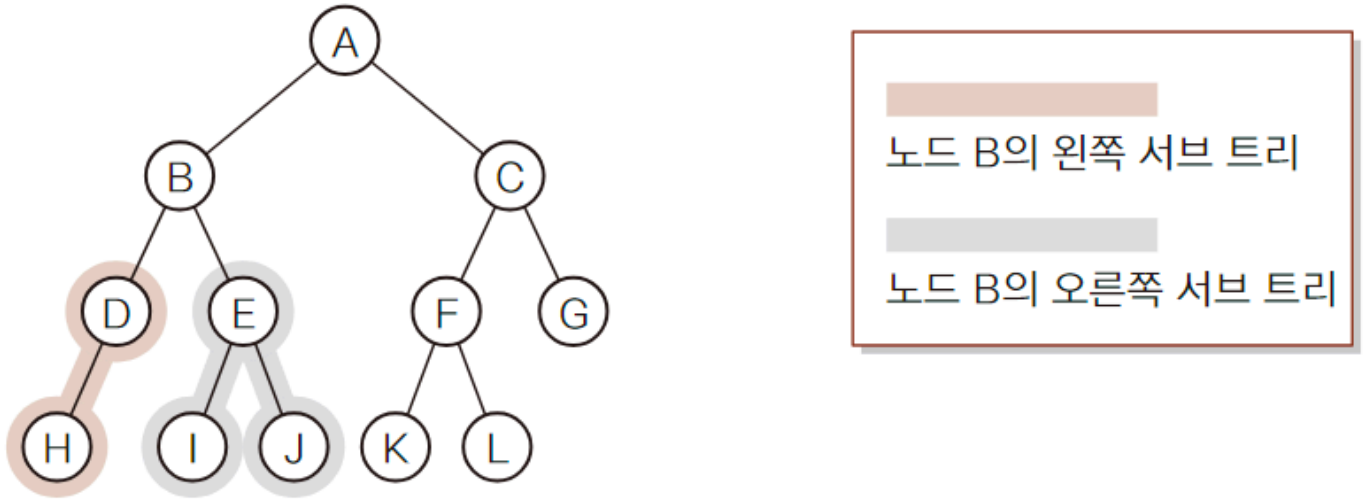
목차

- 이진 트리
- 이진 검색 트리 구현
- 이진 검색 트리 사용 프로그램

이진 트리

이진 트리

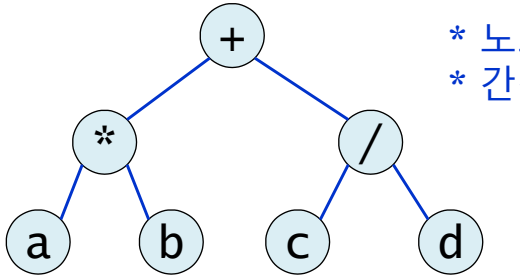
- 이진 트리(binary tree)
  - 노드가 왼쪽 자식과 오른쪽 자식을 갖는 트리. 각 노드의 자식은 2명 이하만 유지해야 함
  - 즉, 모든 노드가 2개의 서브 트리를 가지고 있는 트리. 서브 트리는 공집합일 수 있음
  - 모든 노드의 차수가 2 이하가 됨 → 구현하기 편리
  - 서브 트리간의 순서가 존재 → 왼쪽, 오른쪽



[그림 10-6] 이진트리

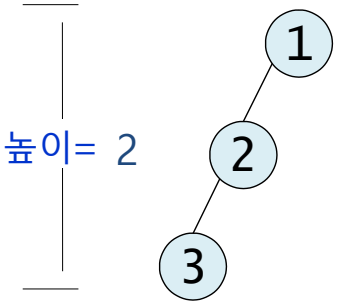
이진 트리 성질

- 노드의 개수가 n개이면 가지의 개수는 n-1

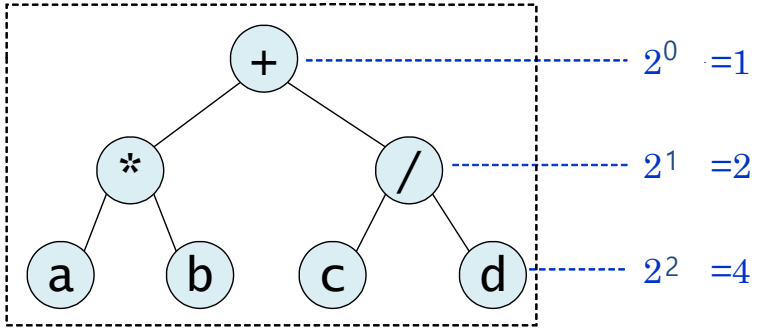


* 노드의 개수: 7
* 간선의 개수: 6

- 높이가 h인 이진 트리의 경우, 최소 h+1개의 노드를 가지며 최대 $2^{h+1}-1$ 개의 노드를 가짐



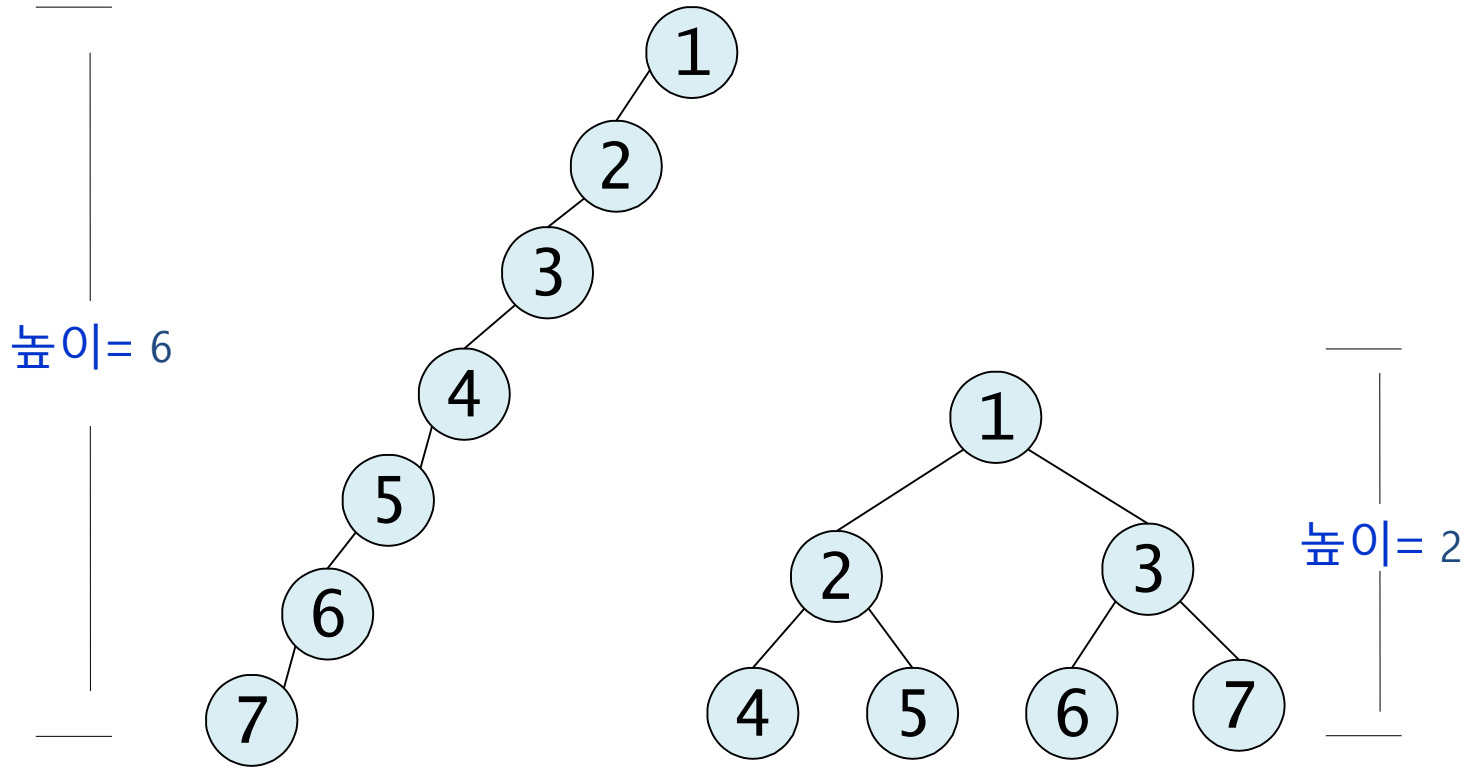
최소 노드 개수=3



최대 노드 개수= $2^0 + 2^1 + 2^2 = 1+2+4 = 7$

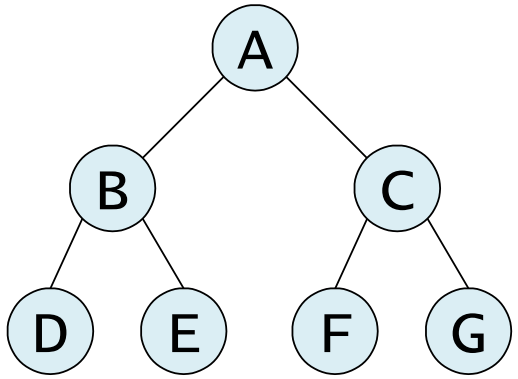
이진 트리 성질

- n개의 노드를 가지는 이진 트리의 높이는 최대 n-1이거나 최소 $log_2(n+1)$ -1



이진 트리의 분류

- 포화 이진 트리(full binary tree)
  - 트리의 각 레벨에 노드가 꽉 차있는 이진 트리



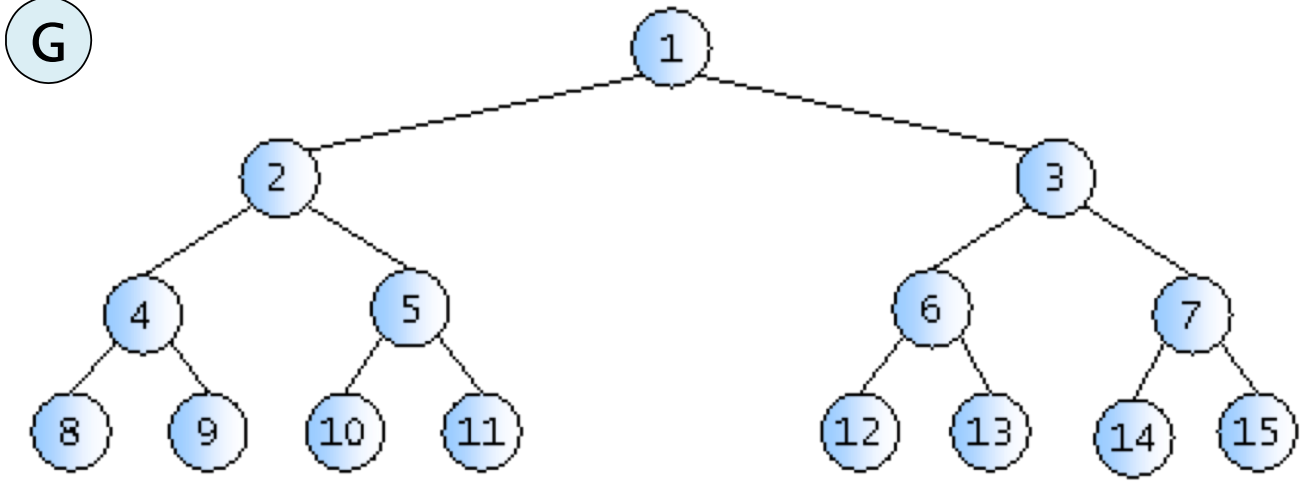
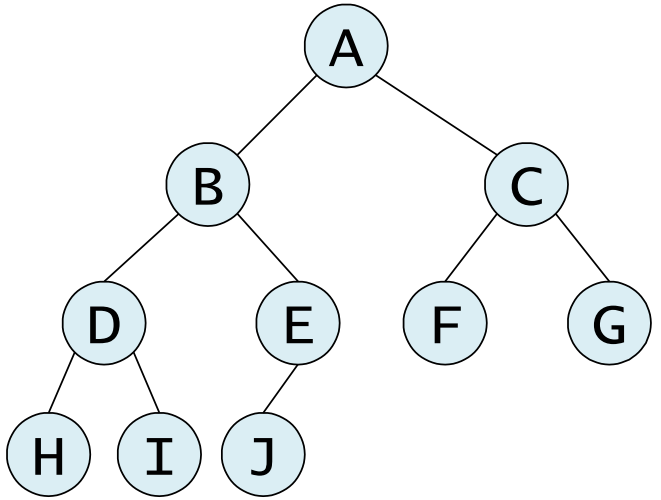


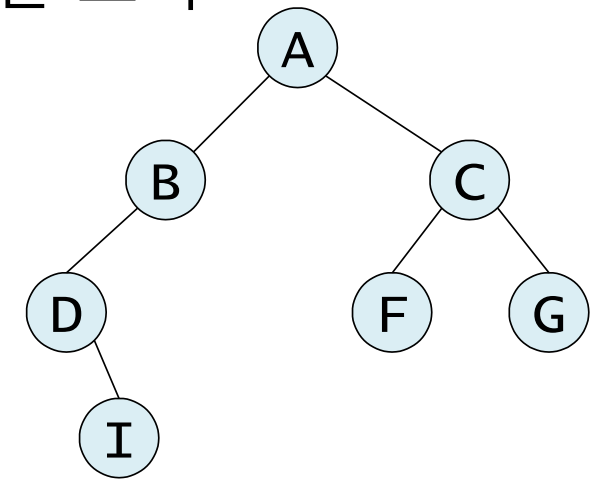
그림 7.15 포화이진트리에서의 노드의 번호

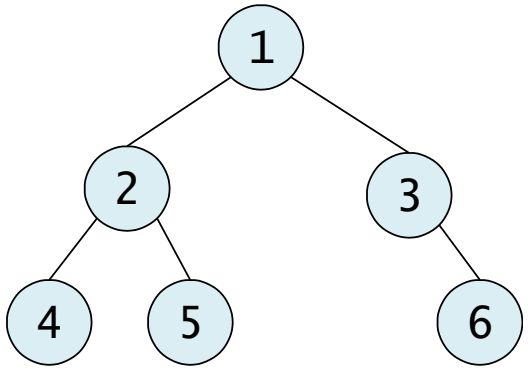
이진 트리의 분류

- 완전 이진 트리(complete binary tree)
  - 높이가 h일 때 레벨 0부터 h-1까지는 노드가 모두 채워짐
  - 마지막 레벨 h에서는 노드가 순서대로 채워짐
  - n개의 노드를 저장할 수 있는 완전 이진 트리의 높이는 $log_2 n$ 보다 작거나 같은 정수



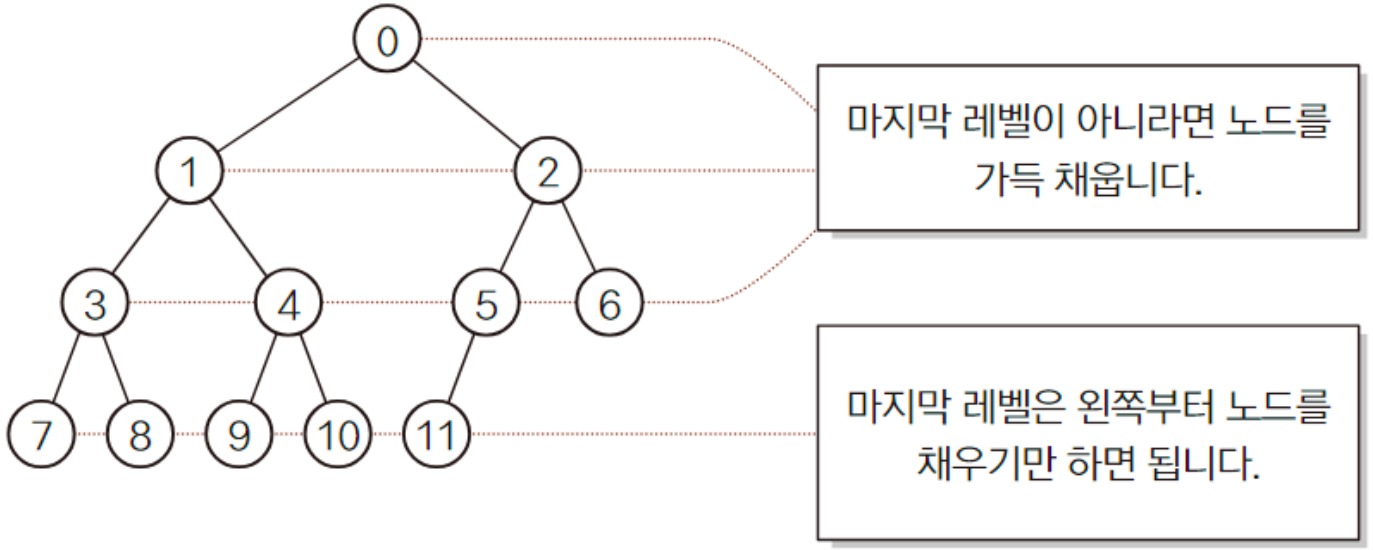
- 기타 이진 트리





완전 이진 트리

- 완전 이진 트리(complete binary tree): 루트부터 노드가 채워져 있으면서 같은 레벨에서는 왼쪽에서 오른쪽으로 노드가 채워져 있는 이진 트리



1. 마지막 레벨을 제외한 레벨은 노드를 가득 채웁니다.
2. 마지막 레벨은 왼쪽부터 오른쪽 방향으로 노드를 채우되 반드시 끝까지 채울 필요는 없습니다.

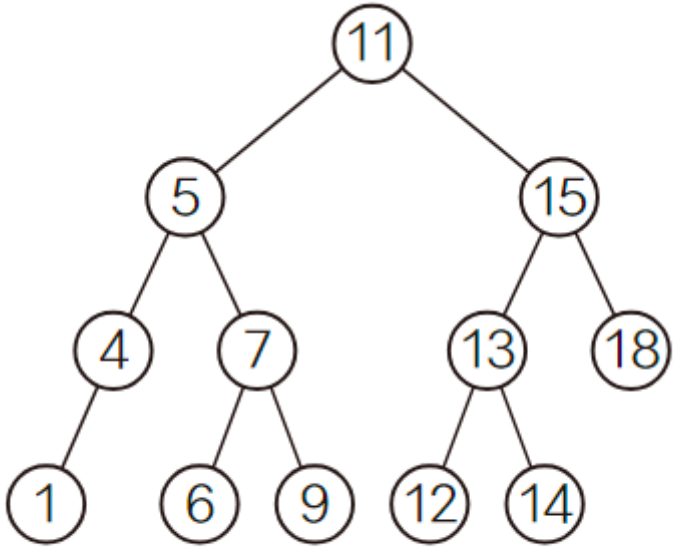
이진 검색 트리

- 이진 검색 트리(binary search tree)
  - 다음 조건을 만족하는 이진 트리

1. 어떤 노드 N을 기준으로 왼쪽 서브 트리 노드의 모든 키 값은 노드 N의 키 값보다 작아야 합니다.
2. 오른쪽 서브 트리 노드의 키 값은 노드 N의 키 값보다 커야 합니다.
3. 같은 키 값을 갖는 노드는 없습니다.

이진 검색 트리

1. 어떤 노드 N을 기준으로 왼쪽 서브 트리 노드의 모든 키 값은 노드 N의 키 값보다 작아야 합니다.
2. 오른쪽 서브 트리 노드의 키 값은 노드 N의 키 값보다 커야 합니다.
3. 같은 키 값을 갖는 노드는 없습니다.



[그림 10-8] 이진검색트리의 구현

이진 검색 트리를 중위 순회(Inorder) 시 노드의 키 값이 오름차순으로 정렬됨

1 → 4 → 5 → 6 → 7 → 9 → 11 → 12 → 13 → 14 → 15 → 18

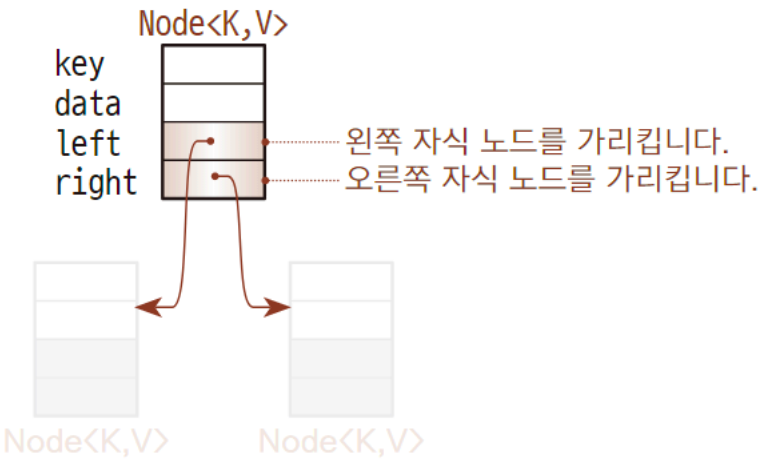
이진 검색 트리 구현

노드 클래스

- 노드 클래스 Node<K, V>
  - 이진 검색 트리의 개별 노드를 나타내는 클래스. 다음 4개의 필드로 구성.

- key … 키 값
- data … 데이터
- left … 왼쪽 자식 노드에 대한 참조(왼쪽 포인터라고 부릅니다).
- right … 오른쪽 자식 노드에 대한 참조(오른쪽 포인터라고 부릅니다).

```
class Node<K, V>{
   K key;                  // 키 값
   V data;                 // 데이터
   Node<K, V> left;        // 왼쪽 자식 노드
   Node<K, V> right;       // 오른쪽 자식 노드
}
```



생성자와 메서드

- 생성자 : 각 필드에 넣을 4개의 값을 전달받아 그대로 설정
- getKey method: 키 값 key를 반환하는 메서드
- getValue method: 데이터 data를 반환하는 메서드
- print method: 데이터 data를 출력하는 메서드

구현


```
01  package chap10;
02  import java.util.Comparator;
03  // 이진검색트리
04
05  public class BinTree<K,V> {
06    // 노드
07    static class Node<K,V> {
08      private K key;                  // 키 값
09      private V data;                 // 데이터
10      private Node<K,V> left;         // 왼쪽 자식 노드
11      private Node<K,V> right;        // 오른쪽 자식 노드
12
13      // 생성자
14      Node(K key, V data, Node<K,V> left, Node<K,V> right) {
15        this.key   = key;
16        this.data  = data;
17        this.left  = left;
18        this.right = right;
19      }
20
21      // 키 값을 반환
22      K getKey() {
23        return key;
24      }
25
26      // 데이터를 반환
27      V getValue() {
28        return data;
29      }
30
31      // 데이터를 출력
32      void print() {
33        System.out.println(data);
34      }
35    }
36
37    private Node<K,V> root;
38    private Comparator<? super K> comparator = null;
```

이진 검색 트리 클래스

- 이진 검색 트리 클래스 BinTree<K, V>
  - 다음 두 개의 필드로 구성됨

  - root … 루트에 대한 참조를 보존·유지하는 필드입니다.
  - comparator … 키 값의 대소 관계를 비교하는 비교자입니다.

생성자

- 비어 있는 이진 검색 트리 생성

```
40    // 생성자
41    public BinTree() {
42       root = null;
43    }
44
45    // 생성자
46    public BinTree(Comparator<? super K> c) {
47       this(); ─1
48       comparator = c; ─2
49    }
```

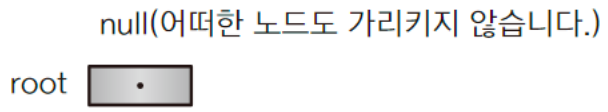
A : 자연 순서에 따라 키 값을 비교

B : 비교자로 키 값을 비교

comparator 값은 널이 됨

생성자

- BinTree()



[그림 10-10] 빈 상태의 이진검색트리

comparator 값은 널이 됨

- BinTree(Comparator<? Super K> c)
  - 인수로 비교자를 전달받는 생성자
  - 노드의 대소 관계 판단할 때 비교자에 의해 다음 수행

1. this()에 의해 인수를 전달받지 않는 생성자 BinTree()를 호출합니다. root가 널인(비어 있는) 이진검색 트리를 생성합니다.
2. 필드 comparator에 전달받은 c를 설정합니다.

Comp method

- comp method
  - 두 개의 키 값을 비교
  - 검색, 삽입, 삭제를 하는 메서드에서 호출하는 비공개 메서드(private method)

```
51      // 두 키 값을 비교
52      private int comp(K key1, K key2) {
53        return (comparator == null) ? ((Comparable<K>)key1).compareTo(key2)
54                                    : comparator.compare(key1, key2);
55      }
```

  - 반환값
    - key1 > key2면 양수
    - key1 < key2면 음수
    - Key1 == key2면 0

Comp method

- 비교자 comparator가 null 일 때
  - 생성자 BinTree()로 이진 검색 트리를 생성한 경우 필드 comparator의 값은 널이고 비교자는 설정되어 있지 않음
  - Key1을 Comparable<K> 인터페이스형으로 형변환(cast)하고 compareTo 메서드를 호출하여 key2와 비교

```
((Comparable<K>)key1).compareTo(key2)
```

- 비교자 comparator가 null이 아닌 경우
  - 생성자 BinTree(Comparator<? Super K> c)로 이진 검색 트리를 생성한 경우 comparator에는 비교자가 설정되어 있음
  - 설정된 comparator의 compare 메서드를 호출하여 두 키 값 key1, key2 비교

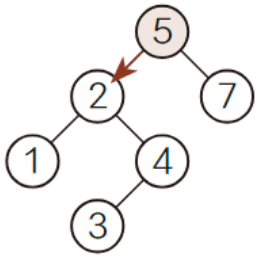
```
comparator.compare(key1, key2)
```

Search method

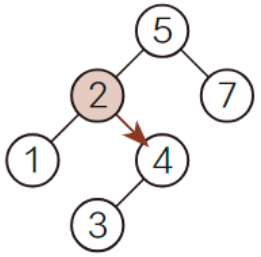
- 키 값으로 검색하는 method

a 3을 검색합니다(검색 성공).

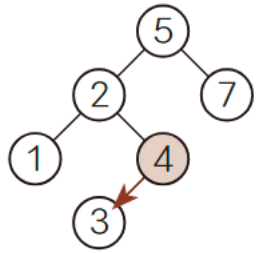
1 5를 선택합니다. 3은 5보다 작기 때문에 왼쪽으로 검색을 진행합니다.



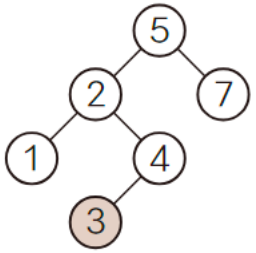
2 2를 선택합니다. 3은 2보다 크기 때문에 오른쪽으로 검색을 진행합니다.



3 4를 선택합니다. 3은 4보다 작기 때문에 왼쪽으로 검색을 진행합니다.



4 3을 찾았습니다. 검색에 성공합니다.

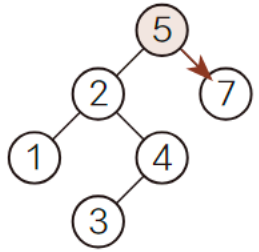


Search method

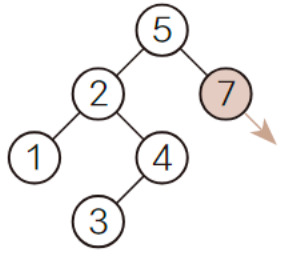
- 키 값으로 검색하는 method

b 8을 검색합니다(검색 실패).

1 5를 선택합니다. 8은 5보다 크기 때문에 오른쪽으로 검색을 진행합니다.



2 7을 선택합니다. 8은 7보다 크기 때문에 오른쪽으로 검색을 진행합니다. 하지만 오른쪽에는 자식이 없어 검색에 실패합니다.



Search method

- 알고리즘

1. 루트부터 선택하여 검색을 진행합니다. 여기서 선택하는 노드를 p라고 합니다.
2. p가 널이면 검색에 실패합니다.
3. 검색하는 값 key와 선택한 노드 p의 키 값을 비교하여
   - 값이 같으면 검색에 성공(검색 종료)합니다.
   - key가 작으면 선택한 노드에 왼쪽 자식 노드를 대입합니다(왼쪽으로 검색 진행).
   - key가 크면 선택한 노드에 오른쪽 자식 노드를 대입합니다(오른쪽으로 검색 진행).
4. 2번 과정으로 되돌아갑니다.

구현


```
57    // 키에 의한 검색
58    public V search(K key)  {
59       Node<K,V> p = root;                          // 루트에 주목
60
61       while (true) {
62          if (p == null)                            // 더 이상 진행하지 않으면
63             return null;                           // 검색 실패
64          int cond = comp(key, p.getKey());         // key와 노드 p의 키를 비교
65          if (cond == 0)                            // 같으면
66             return p.getValue();                   // 검색 성공
67          else if (cond < 0)                        // key 쪽이 작으면
68             p = p.left;                            // 왼쪽 서브 트리에서 검색
69          else                                      // key 쪽이 크면
70             p = p.right;                           // 오른쪽 서브 트리에서 검색
71       }
72    }
```

키 값이 key인 노드를 검색하고 검색에 성공하면 그 노드의 데이터에 대한 참조를 반환

add method

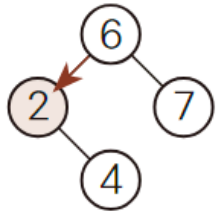
- 노드를 삽입하는 메서드
  - 노드를 삽입한 후에 트리의 형태가 이진 검색 트리의 조건을 유지해야 함
  - 즉, 위치 선정이 중요
  - 이미 같은 키 값을 갖는 노드가 트리에 있다면 노드 삽입 불가

add method

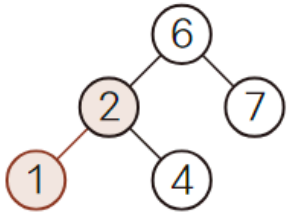
- 예제

a 1을 삽입하는 과정

검색하는 방법과 마찬가지로 삽입할 위치를 찾습니다. 추가할 값인 1은 2보다 작고, 2의 왼쪽 자식 노드는 비어 있으므로 해당 위치를 삽입 위치로 선택합니다.

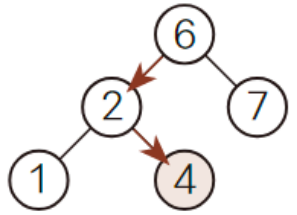


2의 왼쪽 자식 노드로 1을 삽입합니다.

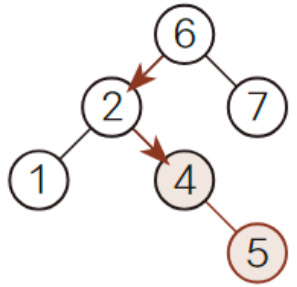


b 5를 삽입하는 과정

검색하는 방법과 마찬가지로 삽입할 위치를 찾습니다. 추가할 값인 5는 4보다 크고, 4의 오른쪽 자식 노드는 비어 있으므로 해당 위치를 삽입 위치로 선택합니다.



4의 오른쪽 자식 노드로 5를 삽입합니다.



[그림 10-12] 이진검색트리에 노드를 삽입하는 과정

add method

- 루트가 node인 서브 트리에 대해 키 값이 key인 데이터를 삽입하는 알고리즘

1. 루트를 선택합니다. 여기서 선택하는 노드를 node로 합니다.
2. 삽입할 키 key와 선택 노드 node의 키 값을 비교합니다. 값이 같다면 삽입에 실패합니다(종료).
   - **값이 같지 않은 경우 key 값이 삽입할 값보다 작으면**

     왼쪽 자식 노드가 없는 경우에는(a) 노드를 삽입합니다(종료).

     왼쪽 자식 노드가 있는 경우에는 선택한 노드를 왼쪽 자식 노드로 옮깁니다.
   - **key 값이 삽입할 값보다 크면**

     오른쪽 자식 노드가 없는 경우에는(b) 노드를 삽입합니다(종료).

     오른쪽 자식 노드가 있는 경우에는 선택한 노드를 오른쪽 자식 노드로 옮깁니다.
3. 2로 되돌아갑니다.

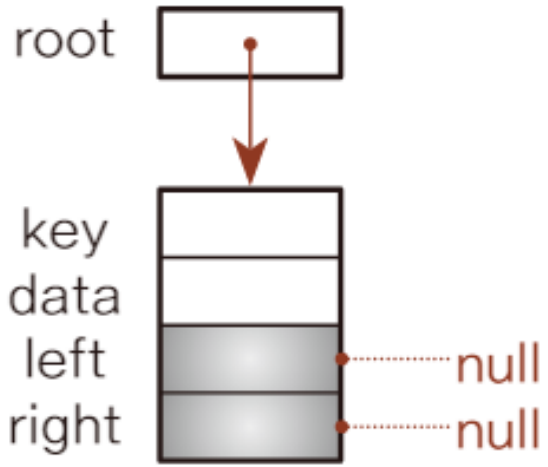
구현


```
74    // node를 루트로 하는 서브 트리에 노드<K,V>를 삽입
75    private void addNode(Node<K,V> node, K key, V data) {
76       int cond = comp(key, node.getKey());
77       if (cond == 0)
78          return;                                    // key가 이진검
79       else if (cond < 0) {
80          if (node.left == null)
81             node.left = new Node<K,V>(key, data, null, null);
82          else
83             addNode(node.left, key, data);          // 왼쪽 서브 트
84       } else {
85          if (node.right == null)
86             node.right = new Node<K,V>(key, data, null, null);
87          else
88             addNode(node.right, key, data);         // 오른쪽 서브 트
89       }
90    }
```

```
92    // 노드를 삽입
93    public void add(K key, V data) {
94       if (root == null)
95          root = new Node<K,V>(key, data, null, null);   ← 1
96       else
97          addNode(root, key, data);                      ← 2
98    }
```

구현

- root가 null인 경우
  - 트리가 비어 있는 상태이므로 루트만으로 구성된 트리를 만들어야 함
  - new Node<K,V>(key, data, null, null)을 이용하여 키값이 key, 데이터가 data, 왼쪽과 오른쪽 포인터가 모두 null인 노드를 생성하고 root가 그것을 참조하도록 함



[그림 10-13] 루트만 있는 이진검색트리

구현

- root가 null이 아닌 경우
  - 트리가 비어있지 않은 상태이므로 메서드 addNode를 호출하여 노드를 삽입

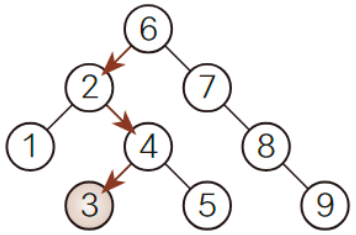
```
addNode(root, key, data);
```
2

remove method

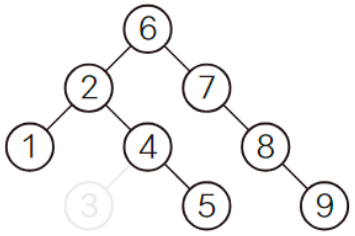
- 노드를 삭제하는 메서드
- 자식 노드가 없는 경우

a 3을 삭제하는 경우

검색할 때와 마찬가지로 3을 먼저 찾아갑니다.
그런 다음 삭제할 노드 3에서 멈춥니다.

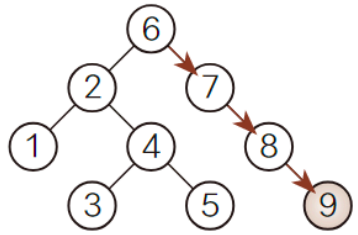


부모의 왼쪽 포인터에 null을 대입합니다.

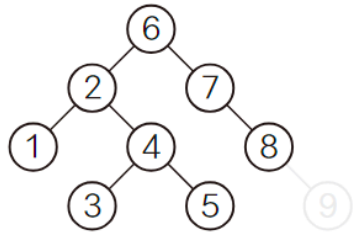


b 9를 삭제하는 경우

검색할 때와 마찬가지로 9를 먼저 찾아갑니다.
그런 다음 삭제할 노드 9에서 멈춥니다.



부모의 오른쪽 포인터에 null을 대입합니다.



- 삭제할 노드가 부모 노드의 왼쪽 자식이면 부모의 왼쪽 포인터를 null로 합니다.
- 삭제할 노드가 부모 노드의 오른쪽 자식이면 부모의 오른쪽 포인터를 null로 합니다.

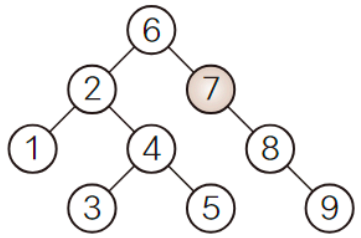
[그림 10-14] 자식 노드가 없는 노드를 삭제하는 과정

remove method

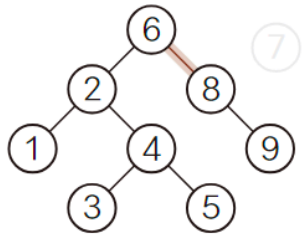
- 자식 노드가 1개인 노드 삭제하는 경우

a 7을 삭제하는 경우

검색할 때와 마찬가지로 7을 찾아갑니다.
그런 다음 삭제할 노드 7에서 멈춥니다.

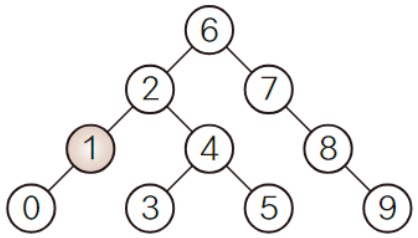


부모 노드인 6의 오른쪽 포인터가 7의 자식 노드인 8을 가리키도록 업데이트합니다.

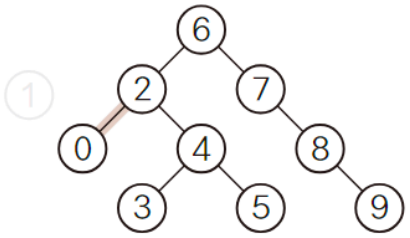


b 1을 삭제하는 경우

검색할 때와 마찬가지로 1을 찾아갑니다.
그런 다음 삭제할 노드 1에서 멈춥니다.



부모 노드인 2의 왼쪽 포인터가 1의 자식 노드인 0을 가리키도록 업데이트합니다.



[그림 10-15] 자식 노드가 1개인 노드를 삭제하는 과정

remove method

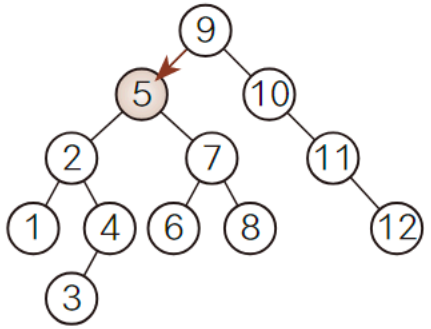
- 자식 노드가 1개인 노드 삭제하는 경우

  - 삭제 대상 노드가 부모 노드의 왼쪽 자식이면 부모의 왼쪽 포인터가 삭제 대상 노드의 자식을 가리키도록 합니다.
  - 삭제 대상 노드가 부모 노드의 오른쪽 자식이면 부모의 오른쪽 포인터가 삭제 대상 노드의 자식을 가리키도록 합니다.

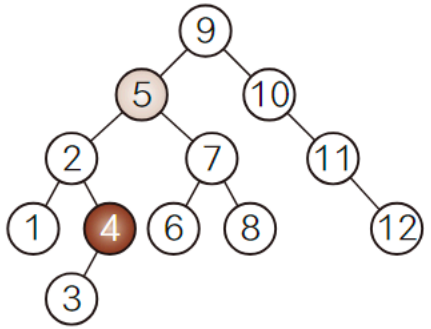
remove method

- 자식 노드가 2개인 노드 삭제하는 경우

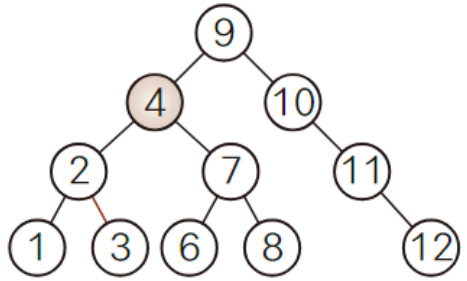
a 5를 삭제하는 경우



5의 왼쪽 서브 트리에서 가장 큰 키 값을 갖는 노드를 검색합니다. 현재는 4가 가장 큰 값을 가지는 노드이므로 여기에서 멈춥니다.



5가 있는 곳으로 4를 옮기면 삭제 과정이 끝납니다. 4를 옮기려면 먼저 4의 데이터를 5의 위치로 복사해야 합니다.



그런 다음 원래의 4를 트리에서 떼어내면 됩니다.

remove method

- 자식 노드가 2개인 노드 삭제하는 경우

1. 삭제할 노드의 왼쪽 서브 트리에서 키 값이 가장 큰 노드를 검색합니다.
2. 검색한 노드를 삭제 위치로 옮깁니다(검색한 노드의 데이터를 삭제 대상 노드 위치로 복사합니다).
3. 옮긴 노드를 삭제합니다. 이때
   - **옮긴 노드에 자식이 없으면**
     '자식 노드가 없는 노드의 삭제 순서'에 따라 노드를 삭제합니다.
   - **옮긴 노드에 자식이 1개만 있으면**
     '자식 노드가 1개 있는 노드의 삭제 순서'에 따라 노드를 삭제합니다.

구현


```
100     // 키 값이 key인 노드를 삭제
101     public boolean remove(K key) {
102         Node<K,V> p = root;                    // 스캔 중인 노드
103         Node<K,V> parent = null;               // 스캔 중인 노드의 부모 노드
104         boolean isLeftChild = true;            // p는 부모의 왼쪽 자식 노드인가?
105
```

구현

```
106   while (true) {
107      if (p == null)                    // 더 이상 진행하지 않으면
108         return false;                  // 그 키 값은 없습니다.
109      int cond = comp(key, p.getKey());  // key와 노드 p의 키 값을 비교
110      if (cond == 0)                    // 같으면
111         break;                         // 검색 성공
112      else {
113         parent = p;                    // 가지로 내려가기 전에 부모를 설정
114         if (cond < 0) {                // key 쪽이 작으면
115            isLeftChild = true;         // 왼쪽 자식으로 내려감
116            p = p.left;                 // 왼쪽 서브 트리에서 검색
117         } else {                       // key 쪽이 크면
118            isLeftChild = false;        // 오른쪽 자식으로 내려감
119            p = p.right;                // 오른쪽 서브 트리에서 검색
120         }
121      }
122   }
```

1

```
124     if (p.left == null) {              // p에는 왼쪽 자식이 없음
125       if (p == root)
126         root = p.right;
127       else if (isLeftChild)
128         parent.left  = p.right;        // 부모의 왼쪽 포인터가 오른쪽 자식을 가리킴
129       else
130         parent.right = p.right;        // 부모의 오른쪽 포인터가 오른쪽 자식을 가리킴
131     } else if (p.right == null) {      // p에는 오른쪽 자식이 없음
132       if (p == root)
133         root = p.left;
134       else if (isLeftChild)
135         parent.left  = p.left;         // 부모의 왼쪽 포인터가 왼쪽 자식을 가리킴
136       else
137         parent.right = p.left;         // 부모의 오른쪽 포인터가 왼쪽 자식을 가리킴
138     } else {
```

2

구현

```
139         parent = p;
140         Node<K,V> left = p.left;        // 서브 트리 가운데 가장 큰 노드
141         isLeftChild = true;
142         while (left.right != null) {    // 가장 큰 노드 left를 검색
143           parent = left;
144           left = left.right;
145           isLeftChild = false;
146         }
147         p.key  = left.key;              // left의 키 값을 p로 옮김
148         p.data = left.data;             // left의 데이터를 p로 옮김
149         if (isLeftChild)
150           parent.left  = left.left;     // left를 삭제
151         else
152           parent.right = left.left;     // left를 삭제
153       }
154       return true;
155     }
```

3

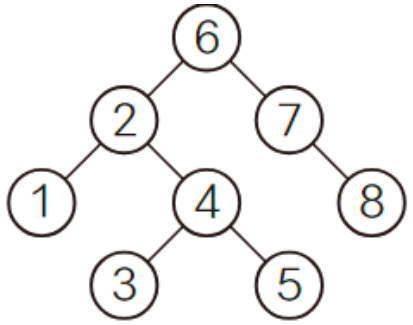
print method

- 모든 노드를 출력하는 메서드 – 오름차순 정렬 위해 중위 순회 사용

```
157    // node를 루트로 하는 서브 트리의 노드를 키 값의 오름차순으로 출력
158    private void printSubTree(Node node) {
159      if (node != null) {
160        printSubTree(node.left);        // 왼쪽 서브 트리를 키 값의 오름차순으로 출력
161        System.out.println(node.key + " " + node.data);      // node를 출력
162        printSubTree(node.right);       // 오른쪽 서브 트리를 키 값의 오름차순으로 출력
163      }
164    }
165
166    // 모든 노드를 키 값의 오름차순으로 출력
167    public void print() {
168      printSubTree(root);
169    }
170  }
```

print method

- Root를 매개변수로 하여 printSubTree 메서드를 호출
- 재귀를 이용하여 노드를 키 값의 오름차순으로 출력
- 예)



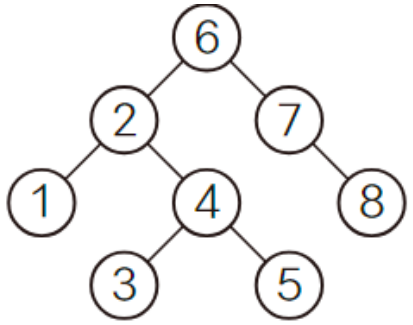
[그림 10-17] 이진검색트리

```
private void printSubTree(Node node) {
   if (node != null) {
      printSubTree(node.left);        // 왼쪽 서브 트리를 ㅊ
      System.out.println(node.key + " " + node.data);
      printSubTree(node.right);       // 오른쪽 서브 트리를
   }
}
```

1 노드 2를 가리키는 왼쪽 포인터를 printSubTree 메서드에 전달하면서 재귀적으로 호출합니다(재귀).
2 현재 노드의 데이터인 6을 출력합니다.
3 노드 7을 가리키는 오른쪽 포인터를 printSubTree 메서드에 전달하면서 재귀적으로 호출합니다(재귀).

print method

- Root를 매개변수로 하여 printSubTree 메서드를 호출
- 재귀를 이용하여 노드를 키 값의 오름차순으로 출력
- 예)



[그림 10-17] 이진검색트리

```
private void printSubTree(Node node) {
   if (node != null) {
      printSubTree(node.left);        // 왼쪽 서브 트리를 ㅊ
      System.out.println(node.key + " " + node.data);
      printSubTree(node.right);       // 오른쪽 서브 트리를
   }
}
```

1 *에 대한 실행*

a 노드 1을 가리키는 왼쪽 포인터를 printSubTree 메서드에 전달하면서 재귀적으로 호출합니다(재귀).

b 현재 노드의 데이터인 2를 출력합니다.

c 노드 4를 가리키는 오른쪽 포인터를 printSubTree 메서드에 전달하면서 재귀적으로 호출합니다(재귀).

이진 검색 트리 사용 프로그램

이진 검색 트리를 사용하는 프로그램 구현


```
01  package chap10;
02  import java.util.Scanner;
03  // 이진검색트리 클래스 BinTree<K,V>의 이용 예
04
05  class BinTreeTester {
06    static Scanner stdIn = new Scanner(System.in);
07
08    // 데이터 (회원번호 + 이름)
09    static class Data {
10      public static final int NO   = 1;      // 번호를 입력 받습니까?
11      public static final int NAME = 2;      // 이름을 입력 받습니까?
12
13      private Integer no;                    // 회원번호 (키 값)
14      private String  name;                  // 이름
```

```
16        // 키 값
17        Integer keyCode() {
18          return no;
19        }
20
21        // 문자열을 반환합니다.
22        public String toString() {
23          return name;
24        }
25
26        // 데이터를 입력합니다.
27        void scanData(String guide, int sw) {
28          System.out.println(guide + "할 데이터를 입력하세요.");
29
30          if ((sw & NO) == NO) {
31            System.out.print("번호 : ");
32            no = stdIn.nextInt();
33          }
34          if ((sw & NAME) == NAME) {
35            System.out.print("이름 : ");
36            name = stdIn.next();
37          }
38        }
39      }
```

구현


```
41      // 메뉴 열거형
42      enum Menu {
43         ADD(      "삽입"),
44         REMOVE(   "삭제"),
45         SEARCH(   "검색"),
46         PRINT(    "표시"),
47         TERMINATE("종료");
48
49         private final String message;       // 출력할 문자열
50
51         static Menu MenuAt(int idx) {       // 서수가 idx인 열거를 반환
52            for (Menu m : Menu.values())
53               if (m.ordinal() == idx)
54                  return m;
55            return null;
56         }
57
58         Menu(String string) {                        // 생성자
59            message = string;
60         }
61
62         String getMessage() {                        // 출력할 문자열을
63            return message;
64         }
65      }
```

구현

```
67      // 메뉴 선택
68      static Menu SelectMenu() {
69        int key;
70        do {
71          for (Menu m : Menu.values())
72            System.out.printf("(%d) %s  ", m.ordinal(), m.getMessage());
73          System.out.print(" : ");
74          key = stdIn.nextInt();
75        } while (key < Menu.ADD.ordinal() || key > Menu.TERMINATE.ordinal());
76
77        return Menu.MenuAt(key);
78      }
```

구현


```
80    public static void main(String[] args) {
81      Menu menu;                                  // 메뉴
82      Data data;                                  // 추가용 데이터 참조
83      Data ptr;                                   // 검색용 데이터 참조
84      Data temp = new Data();                     // 입력용 데이터
85      BinTree<Integer, Data> tree = new BinTree<Integer, Data>();
86
87      do {
88        switch (menu = SelectMenu()) {
89          case ADD :                              // 노드를 삽입
90              data = new Data();
91              data.scanData("삽입", Data.NO | Data.NAME);
92              tree.add(data.keyCode(), data);
93              break;
94
95          case REMOVE :                           // 노드를 삭제
96              temp.scanData("삭제", Data.NO);
97              tree.remove(temp.keyCode());
98              break;
100         case SEARCH :                           // 노드를 검색
101             temp.scanData("검색", Data.NO);
102             ptr = tree.search(temp.keyCode());
103             if (ptr != null)
104               System.out.println("그 번호의 이름은 " + ptr + "입니다.");
105             else
106               System.out.println("해당 데이터가 없습니다.");
107             break;
108
109         case PRINT :                            // 모든 노드를 키 값의 오름차순으로 출력
110             tree.print();
111             break;
112       }
113     } while (menu != Menu.TERMINATE);
114   }
115 }
```

- 키 값 … Integer형의 회원번호
- 데이터 … Integer형의 회원번호와 String형의 이름을 갖는 클래스 Data

실행 결과

실행 결과

(0) 삽입 (1) 삭제 (2) 검색 (3) 출력 (4) 종료 : 0
삽입할 데이터를 입력하세요.
번호 : 1 ..... {1, 현규}를 삽입
이름 : 현규

(0) 삽입 (1) 삭제 (2) 검색 (3) 출력 (4) 종료 : 0
삽입할 데이터를 입력하세요.
번호 : 10 ..... {10, 진아}를 삽입
이름 : 진아

(0) 삽입 (1) 삭제 (2) 검색 (3) 출력 (4) 종료 : 0
삽입할 데이터를 입력하세요.
번호 : 5 ..... {5, 영준}을 삽입
이름 : 영준

(0) 삽입 (1) 삭제 (2) 검색 (3) 출력 (4) 종료 : 0
삽입할 데이터를 입력하세요.
번호 : 12 ..... {12, 윤미}를 삽입
이름 : 윤미

(0) 삽입 (1) 삭제 (2) 검색 (3) 출력 (4) 종료 : 0
삽입할 데이터를 입력하세요.
번호 : 14 ..... {14, 연의}를 삽입
이름 : 연의

(0) 삽입 (1) 삭제 (2) 검색 (3) 출력 (4) 종료 : 2
검색할 데이터를 입력하세요.
번호 : 5 ..... {5}를 검색
그 번호의 이름은 영준입니다.

(0) 삽입 (1) 삭제 (2) 검색 (3) 출력 (4) 종료 : 3
1 현규
5 영준
10 진아
12 윤미
14 연의 ..... 키 값의 오름차순으로 모든 노드를 출력

(0) 삽입 (1) 삭제 (2) 검색 (3) 출력 (4) 종료 : 1
삭제할 데이터를 입력하세요.
번호 : 10 ..... {10}을 삭제

(0) 삽입 (1) 삭제 (2) 검색 (3) 출력 (4) 종료 : 3
1 현규
5 영준
12 윤미
14 연의 ..... 키 값의 오름차순으로 모든 노드를 출력

(0) 삽입 (1) 삭제 (2) 검색 (3) 출력 (4) 종료 : 4