Nonlinear Data Structure

http://smartlead.hallym.ac.kr

Instructor: **Jin Kim**
010-6267-8189(033-248-2318)

jinkim@hallym.ac.kr
Office Hours:

Lab4(이진탐색트리)

http://smartlead.hallym.ac.kr

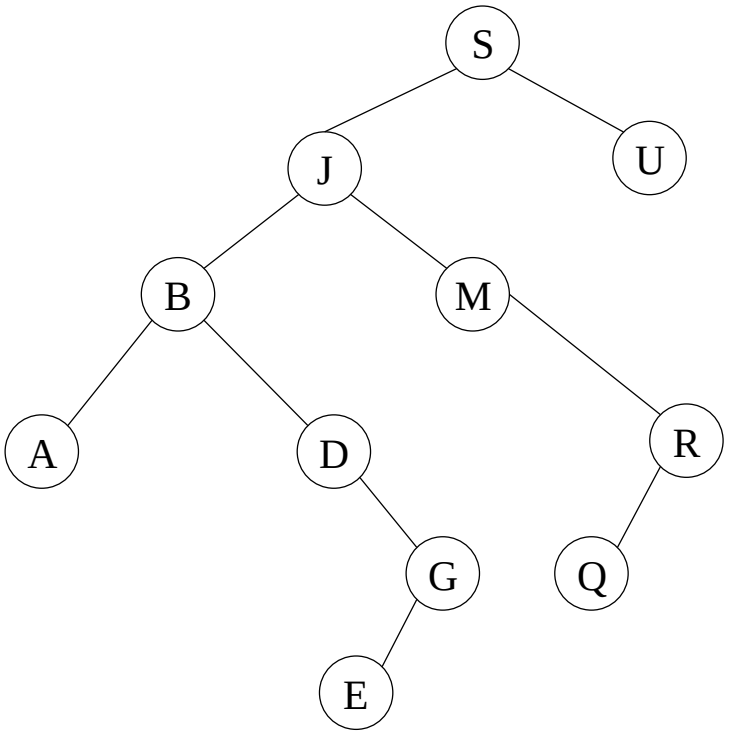
Instructor: **Jin Kim**
010-6267-8189(033-248-2318)

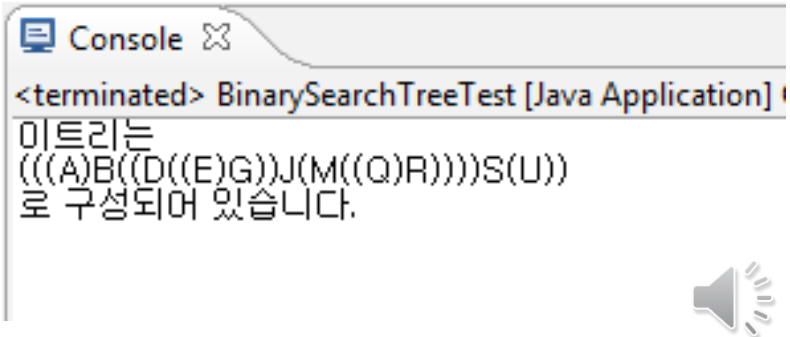
jinkim@hallym.ac.kr
Office Hours:

Binary Search Tree

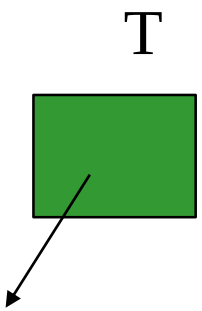
예제 1(Insertion)

- 연결리스트를 이용하여 다음과 같은 이원 탐색 트리를 구현하라.

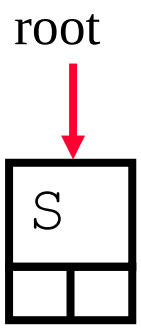




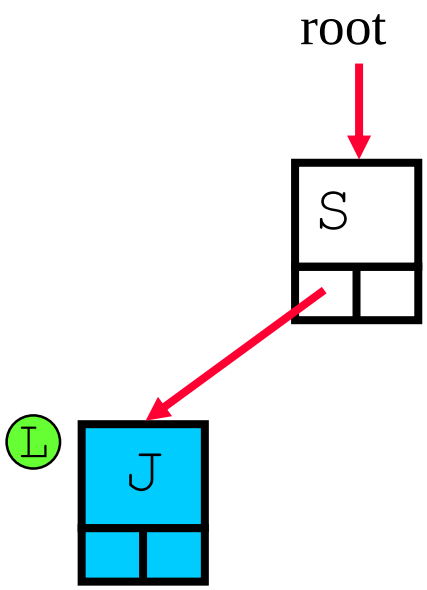
BinarySearchTree T=BinarySearchTree();



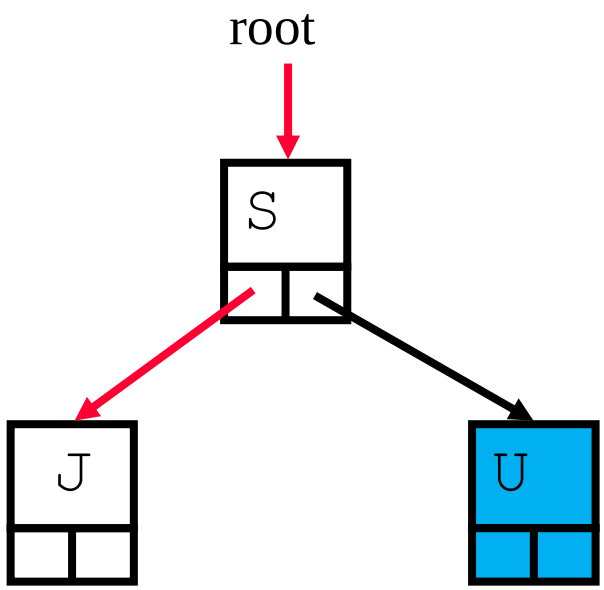
T.insert(“S”)



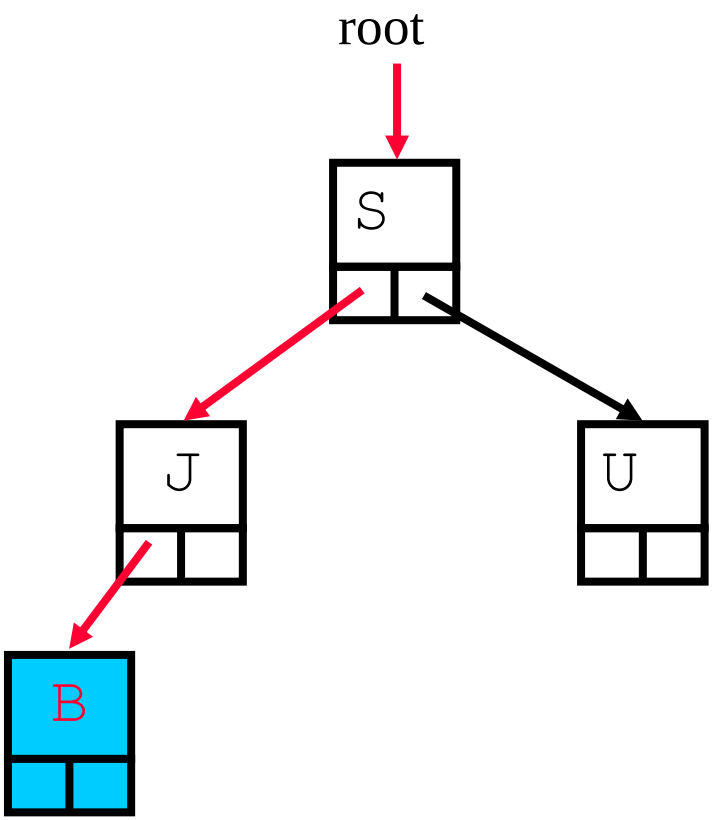
T.insert("J")



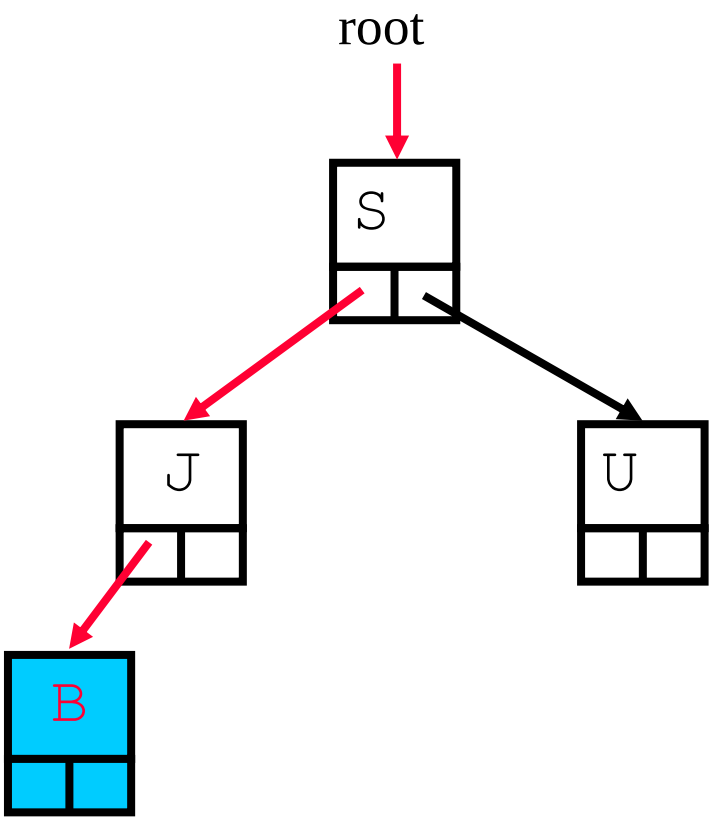
T.insert("U")



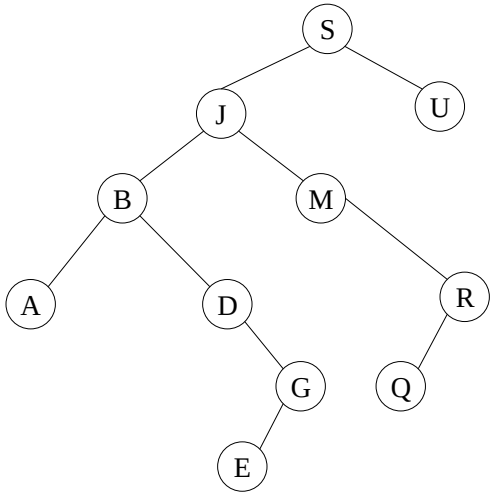
T.insert("B")



T.insert("B")



BST



예제 1(다음을 이해하라)

BinarySearchTreeTest.java

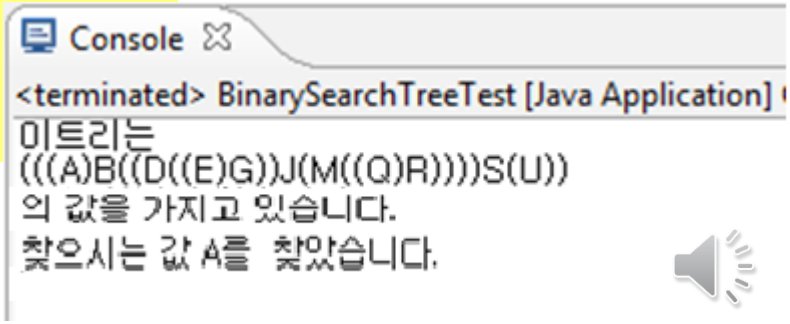
```
class TreeNode {
   String key;
   TreeNode left;
   TreeNode right;
}

class BinarySearchTree {
   private TreeNode rootNode;

   private TreeNode insertKey(TreeNode T, String x) {
      //insert()메소드에 의해 사용되는 보조 순환 메소드
      if (T == null) {
         TreeNode newNode = new TreeNode();
         newNode.key = x;
         return newNode;
      } else if (x.compareTo(T.key) < 0) {//x<T.key이면 x를
         T.left = insertKey(T.left, x);          //T의 왼쪽서브트리에 삽입
         return T;
      } else if (x.compareTo(T.key) > 0) {//x>T.key이면 x를
         T.right = insertKey(T.right, x);        //T의 오른쪽서브트리에 삽입
         return T;
      } else {                                   //key값 x가 이미 T에 있는 경우
         return T;
      }
   }//삽입보조끝

   void insert(String x) {
      rootNode = insertKey(rootNode, x);
   }//삽입끝

   private void printNode(TreeNode N) {
      //printBST() 메소드에 의해 사용되는 순환 메소드
      if (N != null) {
         System.out.print("(");
         printNode(N.left);
         System.out.print(N.key);
         printNode(N.right);
         System.out.print(")");
      }
   }//출력보조끝

   void printBST() {
      //트리를 괄호를 사용하여 출력
      printNode(rootNode);
      System.out.println();
   }//출력끝
}//BinarySearchTree 끝
```

```
public class BinarySearchTreeTest {

   public static void main(String[] args) {
      BinarySearchTree T = new BinarySearchTree();
      T.insert("S");
      T.insert("J");
      T.insert("U");
      T.insert("B");
      T.insert("M");
      T.insert("A");
      T.insert("D");
      T.insert("R");
      T.insert("G");
      T.insert("Q");
      T.insert("E");
      //값을 삽입하여 ppt 3쪽의 트리를 구현

      System.out.println("이트리는");
      T.printBST();
      System.out.println("로 구성되어 있습니다.");
      System.out.println();
      //구축된 이진탐색트리 출력
   }//메인 끝
}//BinarySearchTreeTest 끝
```

과제 1(이해하라)

- 다음의 메소드를 구현하라
  - PrintNode()
    - 중위순회방식으로 트리를 출력한다.

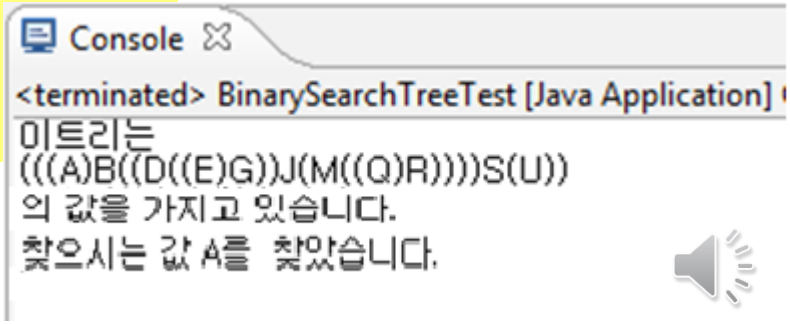
```
private void printNode(TreeNode N) {
                    // printBST() 메소드에 의해 사용되는 순환 메소드
                    if (N != null) {
                                System.out.print("(");
                                printNode(N.left);
                                //중위순회방식으로 출력, 이 부분을 채워라
                                printNode(N.right);
                                System.out.print(")");
                    }
}
```



과제 2(이미구현됨)

- 다음의 메소드를 구현하라
  - find()
    - 매개변수로 받은 문자를 검색 후 해당 노드 반환한다.

```
searchBST(B, x)
        // B는 이원 탐색 트리
        // x는 탐색 키값
        p ← B;
        if (p = null) then                    // 공백 이진 트리로 실패
            return null;
        if (p.key = x) then                   // 탐색 성공
            return p;
        if (p.key < x) then                   // 오른쪽 서브트리 탐색
            return searchBST(p.right, x);
        else return searchBST(p.left, x); // 왼쪽 서브트리 탐색
  end searchBST()
```



과제 1, 2 (계속)

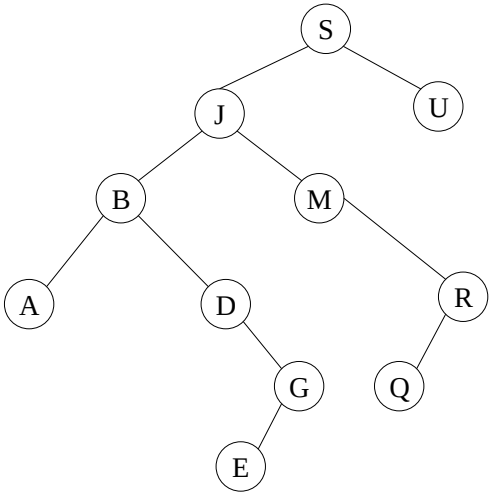
BinarySearchTreeTest.java

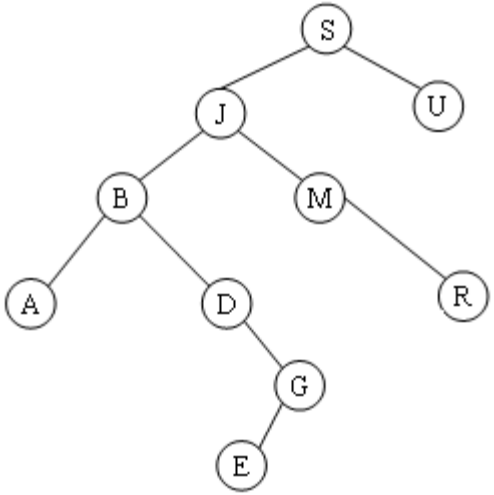
```
class TreeNode {
   String key;
   TreeNode left;
   TreeNode right;
}

class BinarySearchTree {
   private TreeNode rootNode;

   private TreeNode insertKey(TreeNode T, String x) {
      //insert()메소드에 의해 사용되는 보조 순환 메소드
      if (T == null) {
         TreeNode newNode = new TreeNode();
         newNode.key = x;
         return newNode;
      } else if (x.compareTo(T.key) < 0) {//x<T.key이면 x를
         T.left = insertKey(T.left, x);          //T의 왼쪽서브트리에 삽입
         return T;
      } else if (x.compareTo(T.key) > 0) {//x>T.key이면 x를
         T.right = insertKey(T.right, x);        //T의 오른쪽서브트리에 삽입
         return T;
      } else {                                   //key값 x가 이미 T에 있는 경우
         return T;
      }
   }//삽입보조끝

   void insert(String x) {
      rootNode = insertKey(rootNode, x);
   }//삽입끝

   private void printNode(TreeNode N) {
      //printBST() 메소드에 의해 사용되는 순환 메소드
      if (N != null) {
         System.out.print("(");
         printNode(N.left);
         System.out.print(N.key);
         printNode(N.right);
         System.out.print(")");
      }
   }//출력보조끝

   void printBST() {
      //트리를 괄호를 사용하여 출력
      printNode(rootNode);
      System.out.println();
   }//출력끝
```

```
   TreeNode find(String x) {              //키값 x를 가지고 있는 TreeNode의
      TreeNode T = rootNode;              //포인터를 반환
      int result;
      while (T != null) {
         if ((result = x.compareTo(T.key)) < 0) {
            T = T.left;
         } else if (result == 0) {
            return T;
         } else {
            T = T.right;
         }
      }
      return T;
   }//찾기끝
}//BinarySearchTree 끝

public class BinarySearchTreeTest {

   public static void main(String[] args) {
      BinarySearchTree T = new BinarySearchTree();
      T.insert("S");
      T.insert("B");
      T.insert("A");
      T.insert("J");
      T.insert("E");
      T.insert("D");
      T.insert("R");
      T.insert("Q");
      T.insert("A");
      T.insert("G");
      T.insert("E");
      //값을 삽입하여 ppt 2쪽의 트리를 구현

      System.out.println("이트리는");
      T.printBST();
      System.out.println("의 값을 가지고 있습니다.");
      System.out.println();
      //구축된 이진탐색트리 출력

      String key = "A";
      TreeNode P = T.find(key);
      if (P != null) {
         System.out.println("찾으시는 값 " + key + "를  찾았습니다.");
      } else {
         System.out.println("찾으시는 값 " + key + "를 못찾았습니다.");
      }
      System.out.println();
      //key값을 탐색하고 출력
   }//메인 끝
}//BinarySearchTreeTest 끝
```

과제 3(주어진 프로그램에추가)

- 다음의 메소드를 구현하라
  - delete()
    - 주어진 키 값을 가진 원소를 삭제





Problems @ Javadoc Declaration Console

<terminated> BinarySearchTreeTest [Java Application] C:\Program Files\Java\jre7\bi

이트리는
((((A)B(D((E)G)))J(M((Q)R)))S(U))
의 값을 가지고 있습니다.
값을 검색합니다.
찾으시는 값 Q를 찾았습니다.

Q값을 삭제합니다.
((((A)B(D((E)G)))J(M((Q)R)))S(U))
((((A)B(D((E)G)))J(M(R)))S(U))
다시 값을 검색합니다.
찾으시는 값 Q를 못찾았습니다.

삭제 알고리즘

```
deleteBST(B, x)
    p ← the node to be deleted;                  //주어진 키값 x를 가진 노드
    parent ← the parent node of p;               // 삭제할 노드의 부모 노드
    if p = null then return false;     // 삭제할 원소가 없음
    case {
        p.left = null and p.right = null :   // 삭제할 노드가 리프 노드인 경우
                if parent.left = p then parent.left ← null;
                else parent.right ← null;
        p.left = null or p.right = null :            // 삭제할 노드의 차수가 1인 경우
                if p.left ≠ null then {
                    if parent.left = p then parent.left ← p.left;
                    else parent.right ← p.left;
                } else {
                    if parent.left = p then parent.left ← p.right;
                    else parent.right ← p.right;
                  }
        p.left ≠ null and p.right ≠ null :       // 삭제할 노드의 차수가 2인 경우
                            q ← maxNode(p.left);                    // 왼쪽 서브트리에서
                                                                    // 최대 키값을 가진 원소를 탐색
                            p.key ← q.key;
                            deleteBST(p.left, p.key);
    }
end deleteBST()
```

구현 순서

- 삭제하려는 키 값이 주어질 때
  - 키 값을 가진 노드를 검색 구현
  - 원소를 찾으면 삭제 연산 구현

  - 삭제 연산은 해당 노드의 자식 수에 따라 세가지
    - 자식이 없는 리프 노드의 삭제
    - 자식이 하나인 노드의 삭제
    - 자식이 둘인 노드의 삭제

삭제 메소드

- 삭제 메소드 시작

```
deleteBST(B, x)
    p ← the node to be deleted;              //주어진 키값 x를 가진 노드
    parent ← the parent node of p;           // 삭제할 노드의 부모 노드
```

```
void delete(String x) {
    rootNode = deleteKey(rootNode, x);
}

TreeNode deleteKey(TreeNode T, String x) {
    TreeNode p;
    TreeNode parent;
    TreeNode q;
    p = T;
    parent = null;
```

원소 탐색

- 키 값이 주어졌을 때 키 값을 가진 검색
  - find() 함수와 동일

```
while (p != null && p.key != x) {
    parent = p;
    if (( x.compareTo(p.key)) < 0) {
        p = p.left;
    } else {
        p = p.right;
    }
}// 루트로부터 탐색하여 key값과 같은 노드를 찾아 p에 저장
    // parent에 찾은 노드 상위값을 저장해둠
```

- 삭제할 원소가 없을 경우

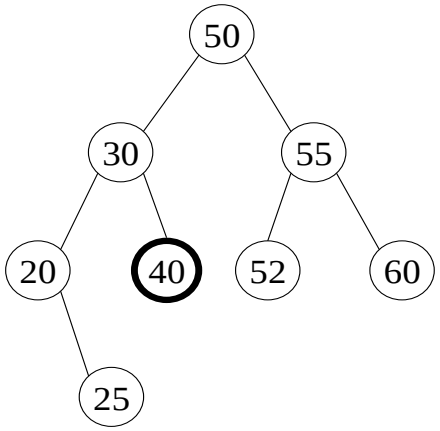
```
if p = null then return false;    // 삭제할 원소가 없음
```

```
if (p == null)
    return null; //삭제할 원소가 없음
```

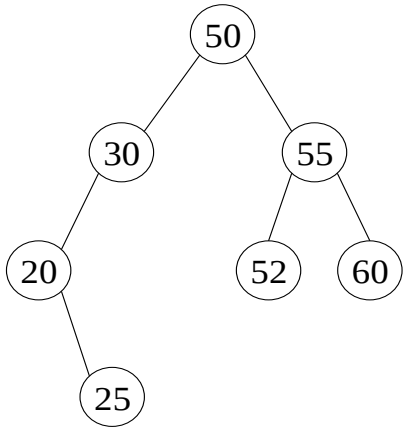
삭제 연산

- 노드의 자식 수에 따른 연산
  - 자식이 없는 리프 노드의 삭제

  - 부모 노드가 가리키고 있는 곳을 널(null)



(a) 삭제전



(b) 삭제후

삭제 연산

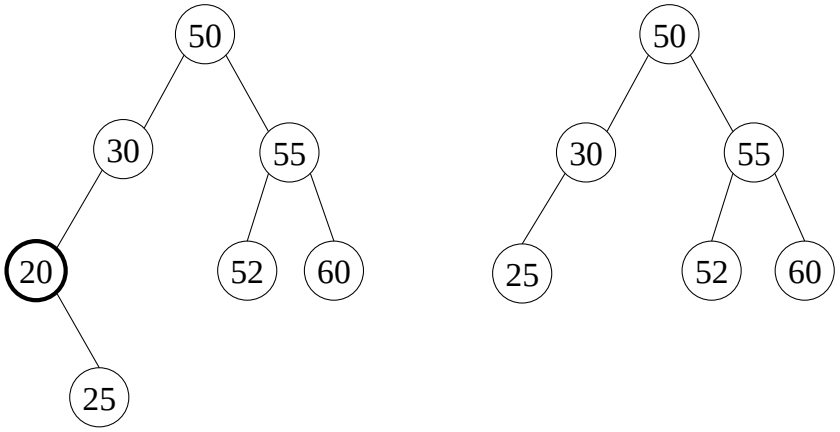
- 노드의 자식 수에 따른 연산
  - 자식이 없는 리프 노드의 삭제

```
p.left = null and p.right = null :  // 삭제할 노드가 리프 노드인 경우
      if parent.left = p then parent.left ← null;
      else parent.right ← null;
```

```
if (p.left == null && p.right == null) {
    if (parent.left == p) {
        parent.left = null;
    } else
        parent.right = null;
}// 자식이 없는 경우 삭제
```

삭제 연산

- 노드의 자식 수에 따른 연산
  - 자식이 하나인 노드의 삭제
  - 삭제되는 노드의 자리에 자식 노드를 위치시킨다.



(a) 삭제전 (b) 삭제후

삭제 연산

- 노드의 자식 수에 따른 연산
  - 자식이 하나인 노드의 삭제

```
p.left = null or p.right = null :
      if p.left ≠ null then {
          if parent.left = p then parent.left ← p.left;
          else parent.right ← p.left;
       } else {
          if parent.left = p then parent.left ← p.right;
          else parent.right ← p.right;
        }
```

```
else if (p.left == null || p.right == null) {
   if (p.left != null) {
      if (parent.left == p) {
         parent.left = p.left;
      } else {
         parent.right = p.left;
      }
   } else {
      if (parent.left == p) {
         parent.left = p.right;
      } else {
         parent.right = p.right;
      }
   }
}// 자식이 하나 있는 경우
```

삭제 연산

- 노드의 자식 수에 따른 연산
  - 자식이 둘인 노드의 삭제

  - 삭제되는 노드 자리에 좌측 서브 트리 중 제일 큰 값이나 오른쪽 서브 트리 중 제일 작은 값을 위치 시킨다.
  - 그 후 서브 트리에서 해당 노드를 삭제 시킨다.

(a) 삭제전

(b) 왼쪽 서브트리의 최대 원소로 대체

(c) 오른쪽 서브트리의 최소 원소로 대체

삭제 연산

- 노드의 자식 수에 따른 연산
  - 자식이 둘인 노드의 삭제

```
p.left ≠ null and p.right ≠ null :
                    q ← maxNode(p.left);

                    p.key ← q.key;
                    deleteBST(p.left, p.key);
```

```
    else if (p.left != null && p.right != null) {
        q = maxNode(p.left);
        p.key = q.key;
        deleteKey(p.left, p.key);
    }// 자식이 둘다 있는 경우
    return T;
}
```

삭제 연산

- 노드의 자식 수에 따른 연산
  - 자식이 둘인 노드의 삭제에서 필요한 maxNode()메소드

```
TreeNode maxNode(TreeNode B) { // 서브트리의 최대원소를 리턴
    TreeNode p;
    p = B;
    if (p == null)
        return p;
    else if (p.right == null)
        return p;
    else {
        while (p.right != null) {
            p = p.right;
        }
        return p;
    }
}
```

삭제 연산

```
String key = "Q";
System.out.println(key + "값을 삭제합니다.");
T.printBST();
P = T.find(key);
if (P != null) {
    T.delete(key);
} else {
    System.out.println("삭제하려는 값 " + key + "를 못찾았습니다.");
}// key값을 탐색하고 삭제
T.printBST();

System.out.println("다시 값을 검색합니다.");
P = T.find(key);
if (P != null) {
    System.out.println("찾으시는 값 " + key + "를 찾았습니다.");
} else {
    System.out.println("찾으시는 값 " + key + "를 못찾았습니다.");
}
System.out.println();
```

BinarySearchTreeTest에 추가

제출

- 완성된 BinarySearchTree.java와 실행화면을 업로드하라. ex) 20209999김진bst.zip

- 기한 :
- 제출장소 : smartlead.hallym.ac.kr