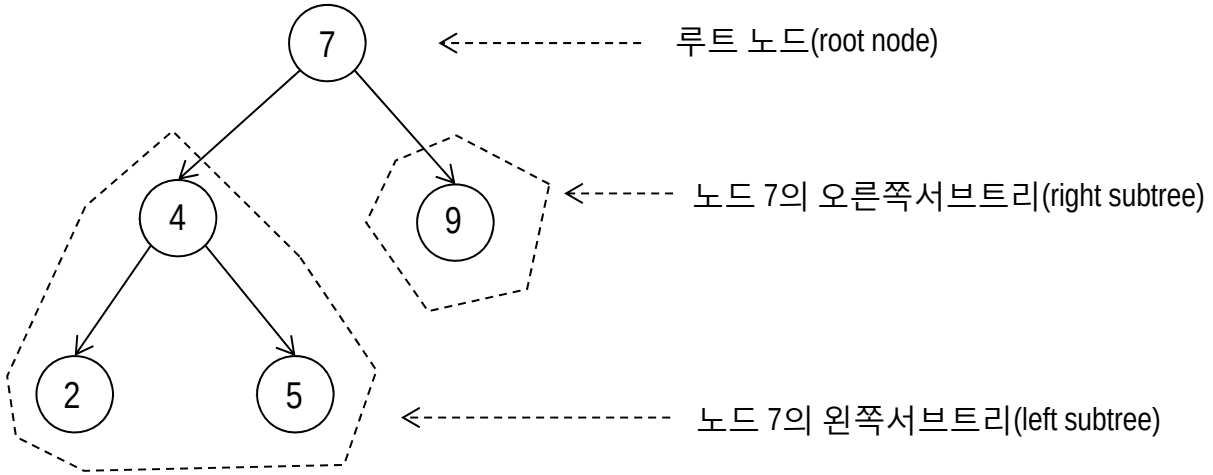
이진탐색트리

이진트리 (binary tree)

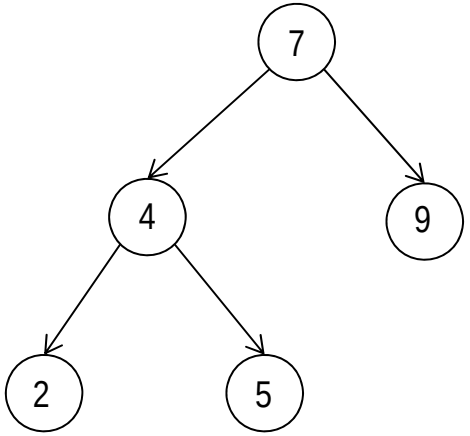
이진트리는 0개 이상 노드(node)들의 모음이며,
각 노드는 다른 노드로의 참조를 0개 이상 최대 2개까지 가지며,
모든 참조는 유일하고(사이클이 없음),
루트로의 참조는 없다.

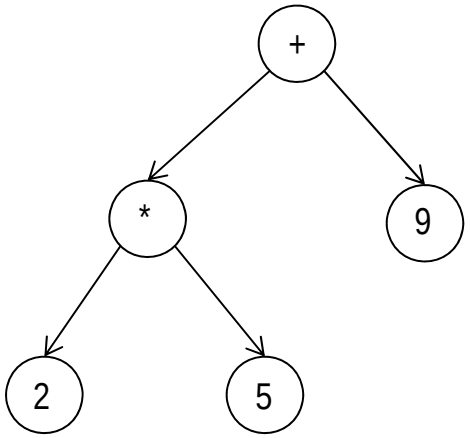


- 노드 7은 이진트리의 루트(root) 노드
- 노드 4와 노드 9는 노드 7의 자식(child) 노드들
- 노드 4는 노드 7의 왼쪽 자식(left child) 노드
- 노드 9는 노드 7의 오른쪽 자식(right child) 노드
- 노드 7은 노드 4 혹은 노드 9의 부모(parent) 노드
- 노드 2, 노드 5, 노드 9는 단말(terminal 혹은 left) 노드 (자식 노드 없음)

이진트리: 순회(traversal)

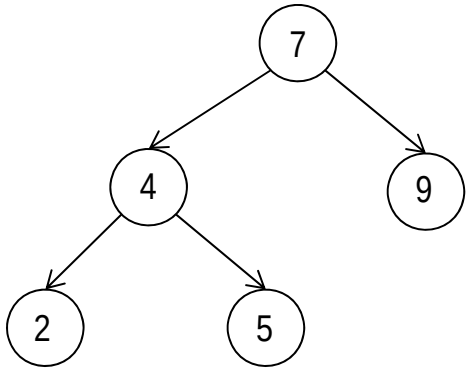
- 전위순회(pre-order traversal)는 루트 노드, 왼쪽 서브트리, 오른쪽 서브트리 순으로 방문한다.
- 중위순회(in-order traversal)는 왼쪽 서브트리, 루트 노드, 오른쪽 서브트리 순으로 방문한다.
- 후위순회(post-order traversal)는 왼쪽 서브트리, 오른쪽 서브트리, 루트 노드 순으로 방문한다.
- 레벨순회(level-order traversal)는 낮은 레벨부터 높은 레벨 순으로 동일 레벨인 경우 좌에서 우로 방문한다.

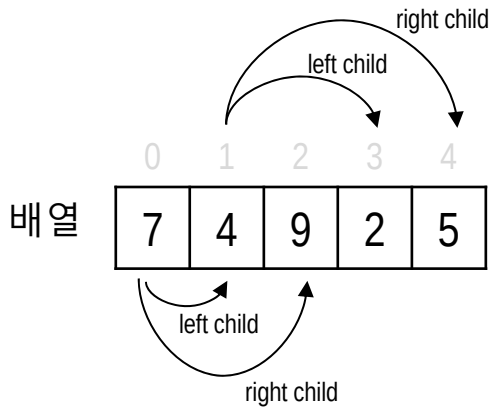




- Pre-order traversal: 7 4 2 5 9
- In-order traversal: 2 4 5 7 9
- Post-order traversal: 2 5 4 9 7
- Level-order traversal: 7 4 9 2 5

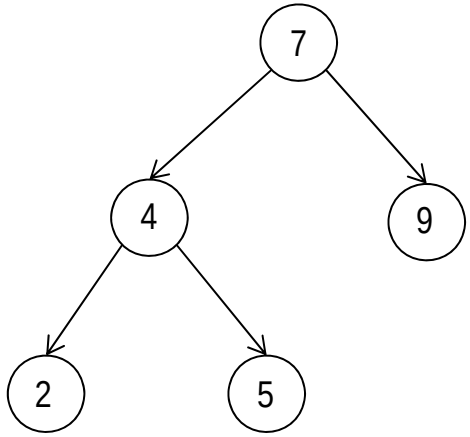
이진트리 (binary tree): 배열 기반 표현법





- 배열 i번째 위치 노드의 왼쪽자식노드 위치 → (2 * i) + 1
- 배열 i번째 위치 노드의 오른쪽자식노드 위치 → (2 * i) + 2
- 배열 i번째 위치 노드의 부모노드 위치 → (i - 1) / 2

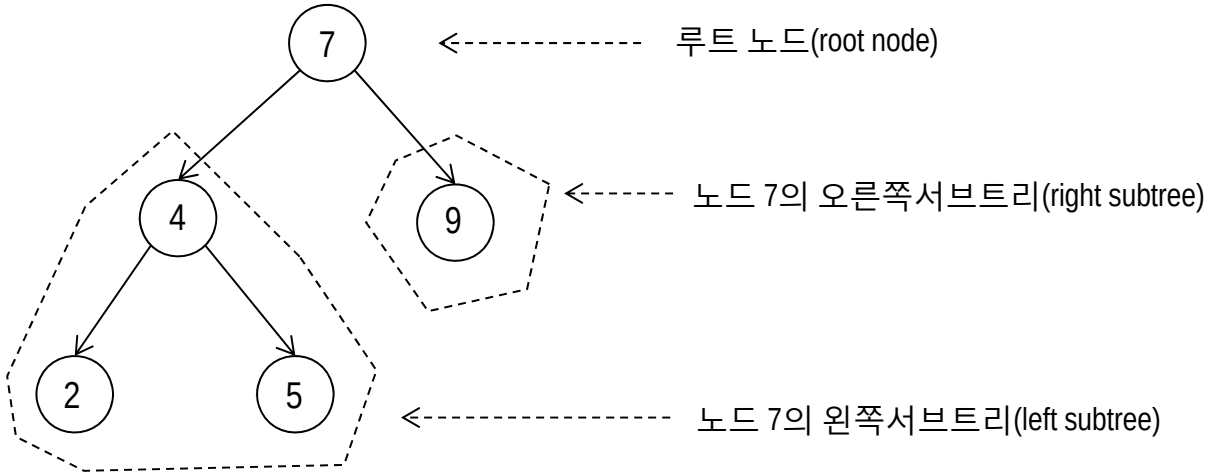
이진트리 (binary tree): 링크 기반 표현법




```
public class Test {
        public static void main(String[] args) {

                class BinaryTree {
                        int     key;
                        BinaryTree left, right;
                        public BinaryTree(int key) {  this.key=key;  }
                }

                BinaryTree  root;
                root=new BinaryTree(7);
                root.left=new BinaryTree(4);
                root.right=new BinaryTree(9);
                root.left.left=new BinaryTree(2);
                root.left.right=new BinaryTree(5);

                LinkedList<BinaryTree>    queue=new LinkedList<>();
                queue.add(root);
                while(!queue.isEmpty()){
                        BinaryTree    node=queue.remove();
                        System.out.print(node.key+" ");
                        if(node.left!=null) queue.add(node.left);
                        if(node.right!=null) queue.add(node.right);
                }
        }
}
```

이진탐색트리 (binary search tree)

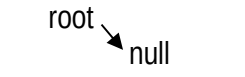
이진탐색트리는 이진트리이며,
이진탐색트리 내 각 노드의 키(key)는 유일하며 그 노드의 왼쪽서브트리 내 모든 노드의 키보다 크고,
그 노드의 오른쪽서브트리 내 모든 노드의 키보다 작다.

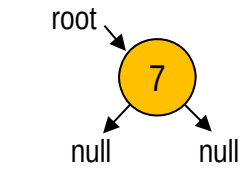


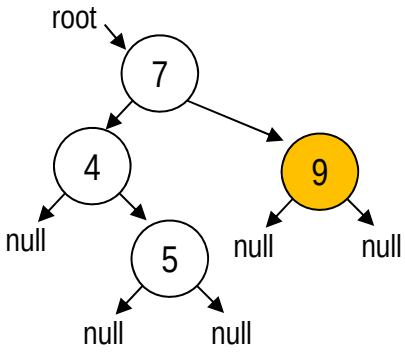
- 2, 4, 5 < 7 < 9
- 2 < 4 < 5

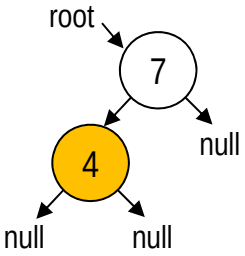
이진탐색트리 (binary search tree): 삽입

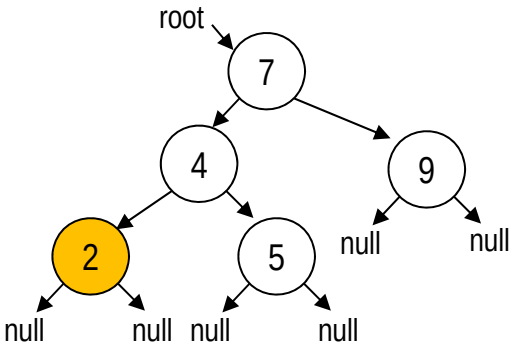
Reference: https://algs4.cs.princeton.edu/32bst/BST.java.html, GPLv3

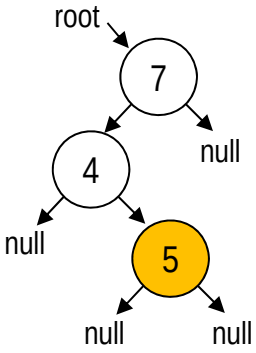













```
public class Test {
	public static void main(String[] args) {
		BinarySearchTree  tree=new BinarySearchTree();
		int	n[]={7, 4, 5, 9, 2};
		//int	n[]={50,20,70,10,30,5,15,25,60,90,62,65,64,35};
		for (int i = 0; i < n.length; i++) tree.add(n[i]);
	}
}

class BinarySearchTree {
	class BinaryTree {
		int	key;
		BinaryTree	left, right;
		public BinaryTree(int key) {  this.key=key;  }
		 public String toString() { return Integer.toString(key); }
	}
	BinaryTree	root;

	public void add(int key) {  root=add(root, key);  }

	private BinaryTree add(BinaryTree tree, int key) {
		if(tree==null) return new BinaryTree(key);
		if(tree.key<key) tree.right=add(tree.right, key);
		else if(tree.key>key) tree.left=add(tree.left, key);
		else ; // value 삽입 시| else tree.value=value;
		return tree;
	}
}
```

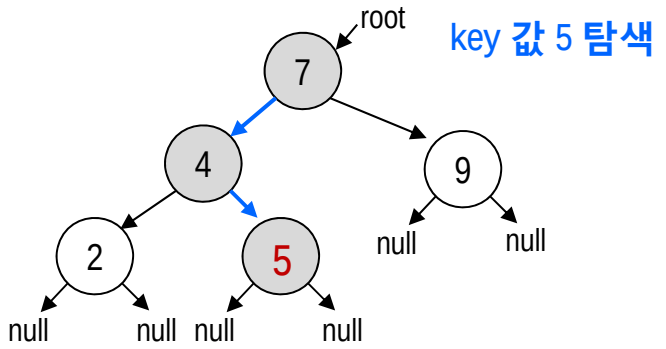
이진탐색트리 (binary search tree): 탐색

Reference: https://en.wikipedia.org/wiki/Binary_search_tree, CC-BY-SA

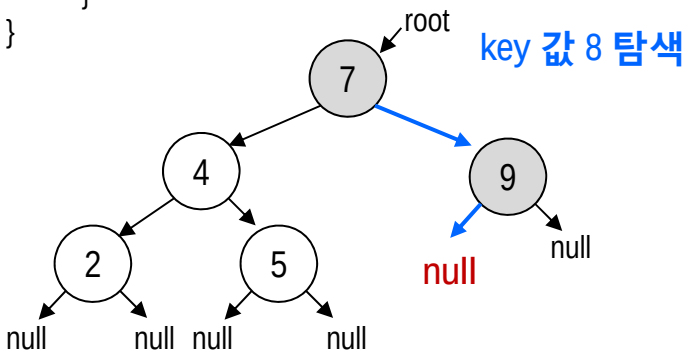
```
class BinarySearchTree {
        class BinaryTree {
                int      key;
                BinaryTree   left, right;
                public BinaryTree(int key) {  this.key=key;  }
                 public String toString() { return Integer.toString(key); }
        }
        BinaryTree   root;
        public void add(int key) { … }
        private BinaryTree add(BinaryTree tree, int key) { … }

        public BinaryTree search(int key) {
                BinaryTree   node=root;
                while(node!=null){
                        if(node.key==key) return node;
                        if(node.key<key) node=node.right;
                        else node=node.left;
                }
                return node;
        }
}
```

실습: 이진탐색트리를 재귀적으로 탐색하는 아래 search 함수가 동작하도록 searchRecur()를 구현하시오.

```
public BinaryTree search(int key) { return searchRecur(root, key); }
private BinaryTree searchRecur(BinaryTree node, int key) { … }
```

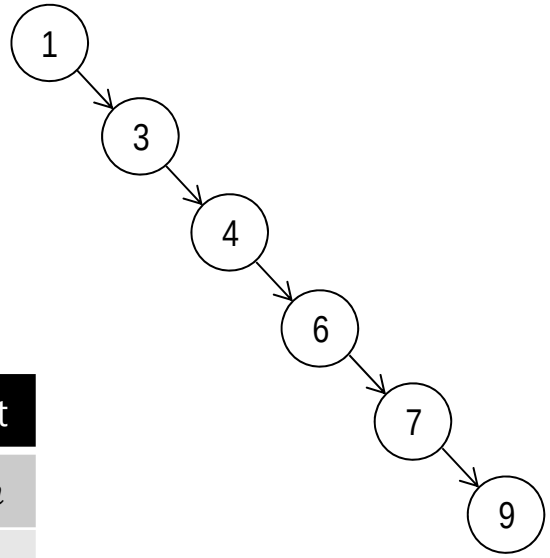



```
public class Test {
        public static void main(String[] args) {
                BinarySearchTree  tree=new BinarySearchTree();
                int      n[]={7,4,5,9,2};
                for (int i = 0; i < n.length; i++) tree.add(n[i]);
                System.out.println(tree.search(5));
                System.out.println(tree.search(8));
        }
}
```



균형이진탐색트리

- (unbalanced) 이진탐색트리
  - 삽입(insert), 삭제(delete), 탐색(search) 시간복잡도
    - 평균 → $O(\log n)$
    - 최악의 경우 → $O(n)$



- 균형탐색트리(self-balancing search tree)

분류	균형탐색트리	Average	Worst
삽입 삭제 탐색	Red-black tree (이진탐색트리)	$n \log n$	$n \log n$
	AVL tree (이진탐색트리)	$n \log n$	$n \log n$
	2-3 tree	$n \log n$	$n \log n$
	B-tree	$n \log n$	$n \log n$

References

- C로 쓴 자료구조론 (Fundamentals of Data Structures in C, Horowitz et al.). 이석호 역. 사이텍미디어. 1993.
- 쉽게 배우는 알고리즘: 관계 중심의 사고법. 문병로. 한빛아카데미. 2013.
- C언어로 쉽게 풀어 쓴 자료구조. 천인국 외 2인. 생능출판사. 2017.
- 프로그래밍 콘테스트 챌린징, Akiba 등 공저, 로드북, 2011.
- https://introcs.cs.princeton.edu/
- Introduction to Algorithms, Cormen et al., 3rd Edition (The MIT Press)