Data Structure #6

Binary Search Tree

2020년 1학기

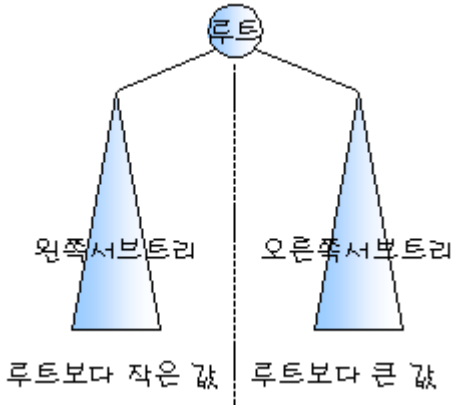
Intro.

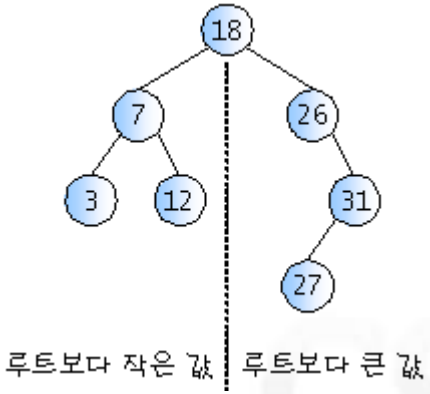
- **실습주제 소개**
  - BST
    - search
    - insert_node
    - delete_node

이진탐색트리

- **정의**
  - 모든 노드는 키를 가지며 동일한 키를 갖지 않는다.
  - 왼쪽 서브트리에 있는 키들은(만약 있다면) 그 루트의 키보다 작다.
  - 오른쪽 서브트리에 있는 키들은(만약 있다면) 그 루트의 키보다 크다.
  - 왼쪽과 오른쪽 서브트리도 모두 이진탐색트리다.

- **탐색작업을 효율적으로 하기 위한 자료구조**





이진탐색트리(개인실습 #1)

- **이진탐색트리 구현시 필요한 요소**
  - (반복적인 방법과 순환적인 방법이 있음)
  - 탐색
  - 새로운 노드 삽입 연산
  - 노드 삭제 연산

이진탐색트리 구현

- 이진탐색트리 구조체

```
typedef struct TreeNode {
        int key;
        struct TreeNode *left, *right;
} TreeNode;
```

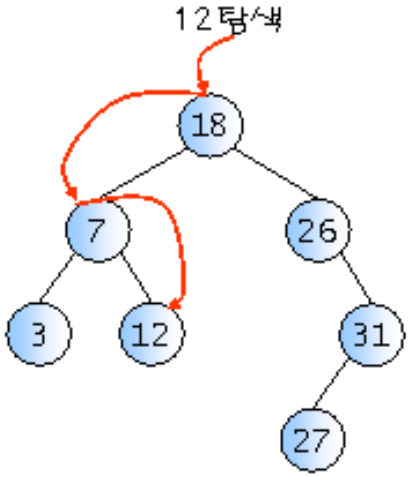
이진탐색트리 구현

– 출력 함수

```
void display(TreeNode *p)
{
        if( p != NULL ) {
                display(p->left);
                printf("%d", p->key);
                display(p->right);
        }
}
```

이진탐색트리 탐색연산

- **탐색연산**
  - 탐색은 루트부터 시작한다.
  - 비교한 결과가 같으면 탐색이 성공적으로 끝난다.
  - 주어진 키 값이 루트 노드의 키 값보다 작으면 탐색은 이 루트 노드의 왼쪽 자식을 기준으로 다시 시작한다.
  - 주어진 키 값이 루트 노드의 키 값보다 크면 탐색은 이 루트 노드의 오른쪽 자식을 기준으로 다시 시작한다.



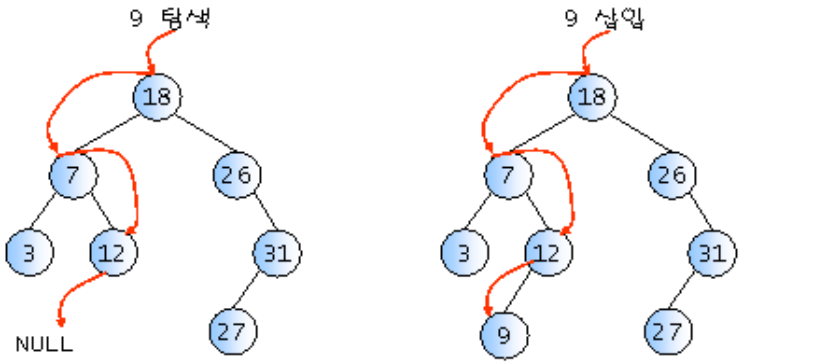
이진탐색트리 구현

– 노드 탐색 함수(iterator, 반복적인 방법)

```
//순환적인 탐색 함수
TreeNode  *search(TreeNode  *node,  int  key)
{
        while(node != NULL){
            if( key == node->key ) return node;
            else if( key < node->key )
                    //fill in the blank
            else
                    //fill in the blank
        }
    return NULL;       // 탐색에 실패했을 경우 NULL 반환
}
```

이진탐색트리 삽입연산

- **삽입연산(반복적 탐색 후 삽입)**
  - 이진탐색트리에 노드를 삽입하기 위해서는 먼저 탐색을 수행하는 것이 필요
  - 탐색에 실패한 위치 = 새로운 노드를 삽입할 위치



(a) 탐색을 먼저 수행 (b) 탐색이 실패한 위치에 9를 삽입

이진탐색트리 구현

- 노드 삽입 함수 (iterator, 반복적인 방법)

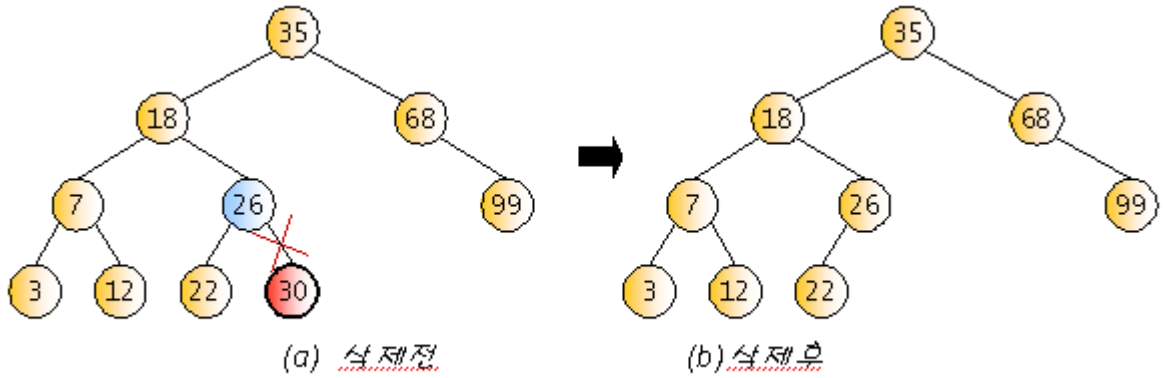
```
void insert_node(TreeNode **root, int key)
{
    TreeNode *parNode, *curNode;
    TreeNode *newNode;
    curNode = *root;
    parNode = NULL;
    // 매개변수로 받은 key 탐색 수행

    // key가 트리 안에 없으므로 삽입 가능
        // new = malloc

   // 데이터 복사
        //n->key = key, n->left, right = NULL

    // 부모 노드와 링크 연결
        // if(부모노드의 key보다 작은경우)
        // else 부모노드의 key보다 큰경우;
        // 부모노드가 없는경우
}
```

이진탐색트리 삭제

- **3가지의 경우**
  - 삭제하려는 노드가 단말 노드일 경우
  - 삭제하려는 노드가 왼쪽이나 오른쪽 서브 트리중 하나만 가지고 있는 경우
  - 삭제하려는 노드가 두 개의 서브 트리를 모두 가지고 있는 경우

이진탐색트리 삭제

- **CASE 1:**
  - 삭제하려는 노드가 단말 노드일 경우,
    단말 노드의 부모 노드를 찾아서 연결을 끊으면 된다.

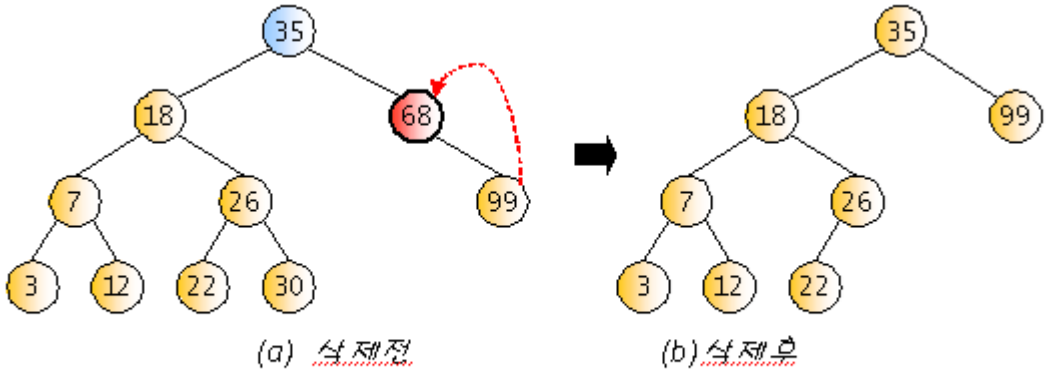


(a) 삭제전 (b) 삭제후

그림 7.42 이진탐색트리의 삭제연산: 삭제노드가 단말노드인 경우

이진탐색트리 삭제

- **CASE 2:**
  - 삭제하려는 노드가 하나의 서브트리만 가지고 있는 경우,
    삭제되는 노드가 왼쪽이나 오른쪽 서브 트리중 하나만 가지고 있는 경우에는 노드는 삭제하고 서브 트리는 부모 노드에 붙여준다.

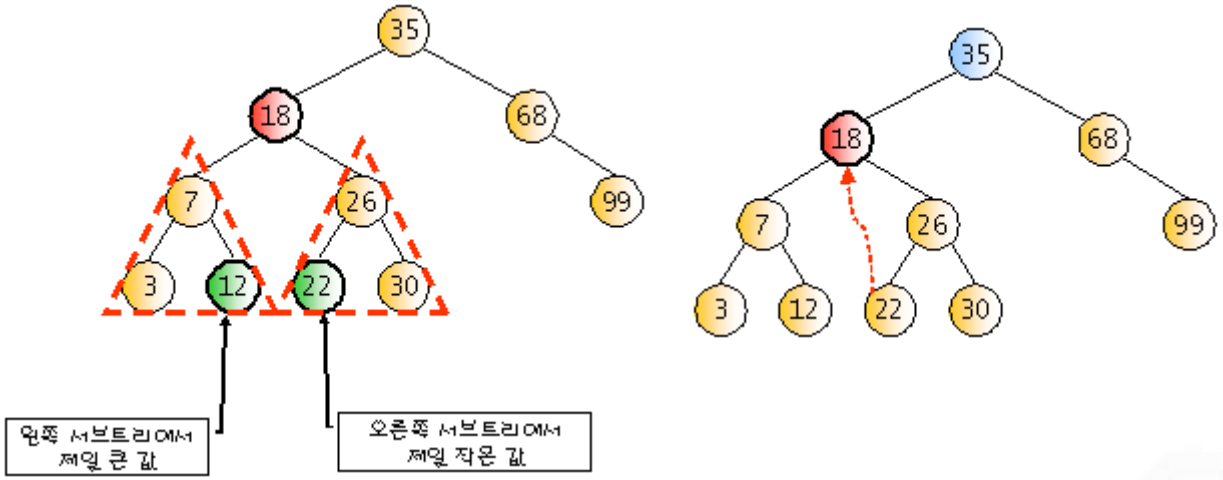


(a) 삭제전 (b) 삭제후

그림 7.43 이진탐색트리의 삭제연산: 삭제노드가 하나의 서브트리를 가지고 있는 경우

이진탐색트리 삭제

- **CASE 3:**
  - 삭제하려는 노드가 두 개의 서브트리를 가지고 있는 경우,
    삭제할 노드의 오른쪽 서브트리에서 가장 작은 값을 가진 노드를 삭제노드 위치로 가져온다.



이진탐색트리 구현

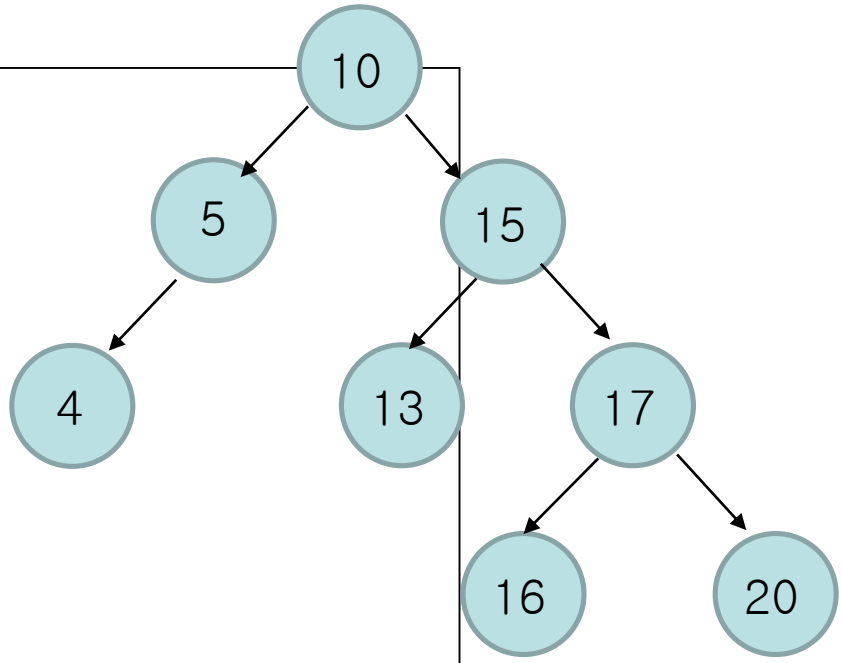
– 노드 삭제 함수 (iterator, 반복적인 방법)

```
void delete_node(TreeNode *node, int key)
{
    TreeNode *parNode, *curNode, *childNode, *succ, *succ_p
    parNode = NULL;
    curNode = node;
    // 삭제할 key 탐색 수행

    // case 1

    // case2

    // case3

}
```

이진탐색트리 구현

- 이진탐색트리 메인함수

```
void main(){
TreeNode *root = NULL;
insert_node(&root, 10);
insert_node(&root, 5);
insert_node(&root, 15);
insert_node(&root, 4);
insert_node(&root, 13);
insert_node(&root, 17);
insert_node(&root, 16);
insert_node(&root, 20);

display(root);
if(search(root, 5) != NULL)
printf("search success: %d₩n", search(root, 5)->key);

delete_node(root, 5);
display(root);
delete_node(root, 15);
display(root);
delete_node(root, 20);
display(root);
}
```



제출

- **제출**
  - 개인실습 (#1)
    - 오늘 자정까지 제출 (~ 2020/4/24 23:59)
    - Binary Search Tree
  - 과제
    - Binary Search Tree (&traversal)구현 (lab6.docx)
    - 다음 주 목요일 자정까지 제출 (~ 2020/4/30 23:59)

과제

- **Lab6.docx**
  - insertNode (i x)
    - 새로운 key x를 binary search tree에 삽입
    - x가 이미 tree에 존재할 경우, 에러메시지 출력
  - deleteNode (d x)
    - binary search tree에 있는 key x를 삭제
    - x가 tree에 존재하지 않는 경우, 에러메시지 출력
  - findNode (f x)
    - tree 내 key x 존재 여부 확인
  - printInorder (pi)
  - printPreorder (pr)
  - printPostorder (po)

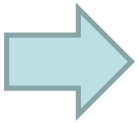
과제

- 함수
  - Tree* insertNode(Tree *root, int key)
  - Tree* deleteNode(Tree *root, int key)
  - Tree* findNode(Tree *root, int key)
  - void printInorder(Tree *root)
  - void printPreorder(Tree *root)
  - void printPostorder(Tree *root)

과제

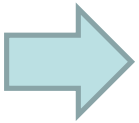
- 예시

```
i 4
i 2
i 6
i 1
i 3
i 5
pi
pr
po
f 3
f 7
```



```
1 2 3 4 5 6
4 2 1 3 6 5
1 3 2 5 6 4
3 is in the tree
7 is not in the tree
```

```
i 10
i 5
i 15
i 4
i 13
i 17
i 16
i 20
pi
d 5
d 15
pi
```



```
4 5 10 13 15 16 17 20
4 10 13 16 17 20
```