DATA
STRUCTURES
USING C

- 탐색의 개념을 이해한다.
- 이진탐색트리의 개념을 이해한다.
- 이진탐색트리의 연산들을 이해한다.
- 이진탐색트리의 효율성을 이해한다.
- 이진탐색트리를 이용한 문제해결 능력을 배양한다.
- 이진탐색트리 균형화의 의미를 이해한다.

CHAPTER 09

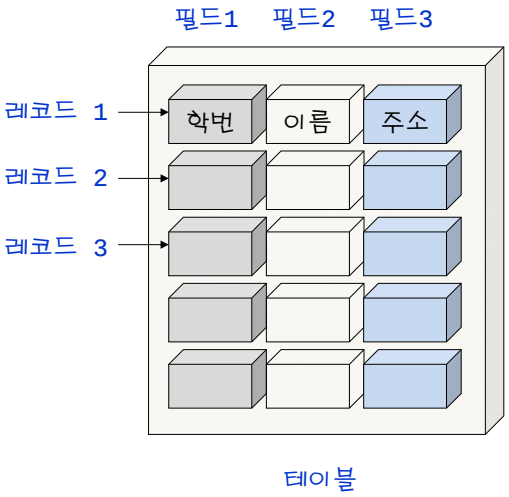
이진 탐색 트리

9.1 이진탐색트리

- **탐색(search)**은 가장 중요한 컴퓨터 응용의 하나
- **이진 탐색 트리(BST, Binary Search Tree)**
  - 이진트리 기반의 탐색을 위한 자료 구조
  - 효율적인 탐색 작업을 위한 자료 구조

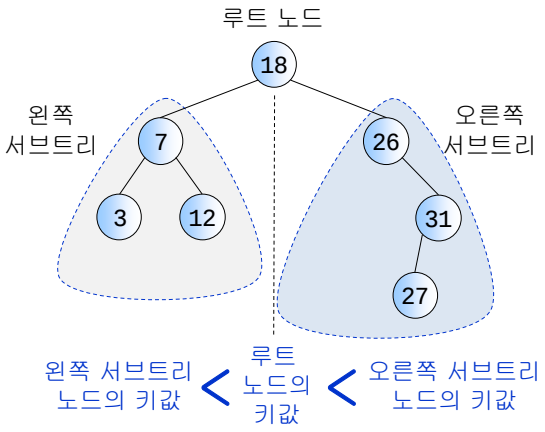
탐색 관련 용어

- 레코드(record)
- 필드(field)
- 테이블(table)
- 키(key)
- 주요키(primary key)



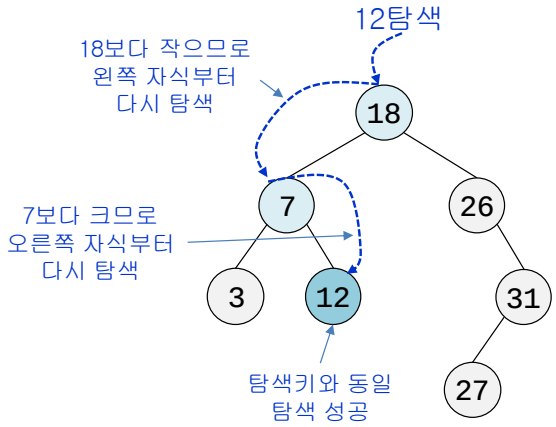
이진탐색트리의 정의

- 탐색작업을 효율적으로 하기 위한 자료구조
  - key(왼쪽서브트리)<key(루트노드)<key(오른쪽서브트리)
  - 이진탐색를 중위순회하면 오름차순으로 정렬된 값을 얻을 수 있다.



9.2 이진탐색트리의 연산

- 탐색 연산
  - 비교한 결과가 같으면 탐색이 성공적으로 끝난다.
  - 키 값이 루트보다 작으면 ➔ 왼쪽 자식을 기준으로 다시 탐색
  - 키 값이 루트보다 크면 ➔ 오른쪽 자식을 기준으로 다시 탐색



이진탐색트리의 탐색연산

- 순환적인 구현

```
TNode* search( TNode *n, int key )
{
        if (n == NULL ) return NULL;
        else if (key== n->data) return n;
        else if (key < n->data) return search( n->left, key );
        else return search( n->right, key );
}
```

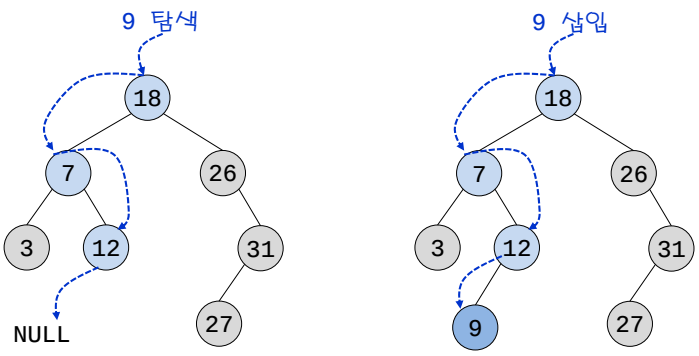
- 반복적인 구현

```
TNode* search_iter( TNode *n, int key )
{
        while(n != NULL){
                if( key == n->data ) return n;
                else if( key < n->data ) n = node->left;
                else n = node->right;
        }
        return NULL;
}
```

- 이진탐색트리를 사용하여 우선순위 큐를 구현할 수도 있다. 우선순위 큐란 항목들이 우선순위를 가지고 있고 우선순위가 가장 큰 항목이 먼저 삭제되는 큐이다. 이진탐색트리에서 가장 큰 값을 찾으려면 어떻게 해야 하는가?

이진탐색트리의 삽입연산

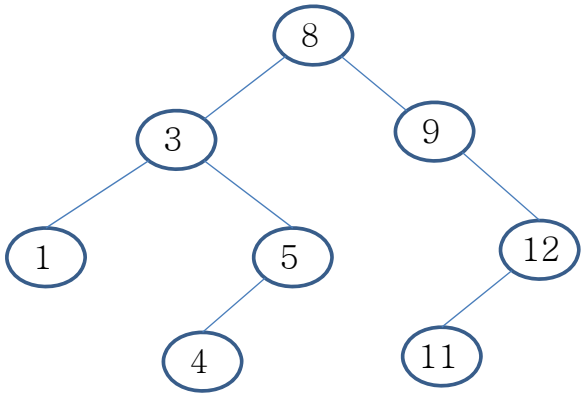
- 먼저 탐색을 수행
  - 탐색에 실패한 위치가 바로 새로운 노드를 삽입하는 위치



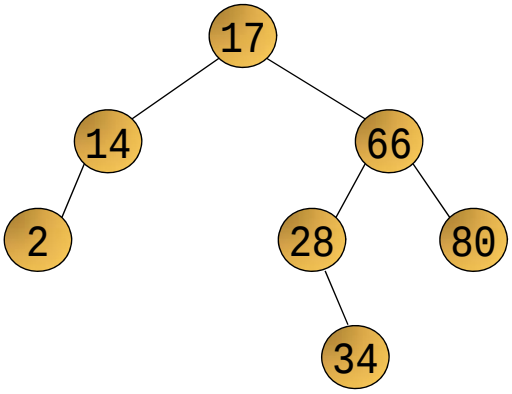
삽입연산 알고리즘 구현

```
void insert( TNode* r, TNode* n )
{
    if( n->data == r->data ) return;
    else if( n->data < r->data ) {
        if( r->left == NULL ) r->left = n;
        else insert( r->left, n );
    }
    else {
        if( r->right== NULL ) r->right = n;
        else insert( r->right, n );
    }
}
```


- 다음의 이진탐색트리에 7을 추가한 후에 재구성되는 트리를 그려라.



- 다음의 이진 탐색 트리에서 입력한 데이터의 순서는?



- 이진탐색트리가 공백상태로부터 다음과 같은 순서로 노드들이 추가된다. 생성되는 이진탐색트리를 그려라.

  11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31

- 다음 데이터들이 어떤 순서로 이진탐색트리에 입력되었을 경우, 가장 균형잡힌 트리가 되는가?

  10, 5, 6, 13, 15, 8, 14, 7, 12, 4

- 또, 어떤 순서로 입력되었을 경우에 가장 불균형한 이진탐색트리가 되는가?

이진탐색트리의 삭제연산

- 노드 삭제의 3가지 경우

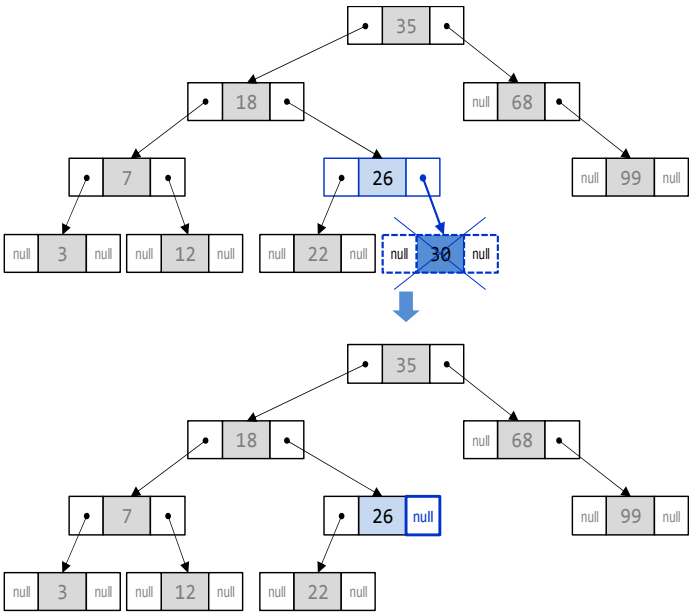
  1. 삭제하려는 노드가 단말 노드일 경우

  2. 삭제하려는 노드가 하나의 왼쪽이나 오른쪽 서브 트리중 하나만 가지고 있는 경우

  3. 삭제하려는 노드가 두개의 서브 트리 모두 가지고 있는 경우

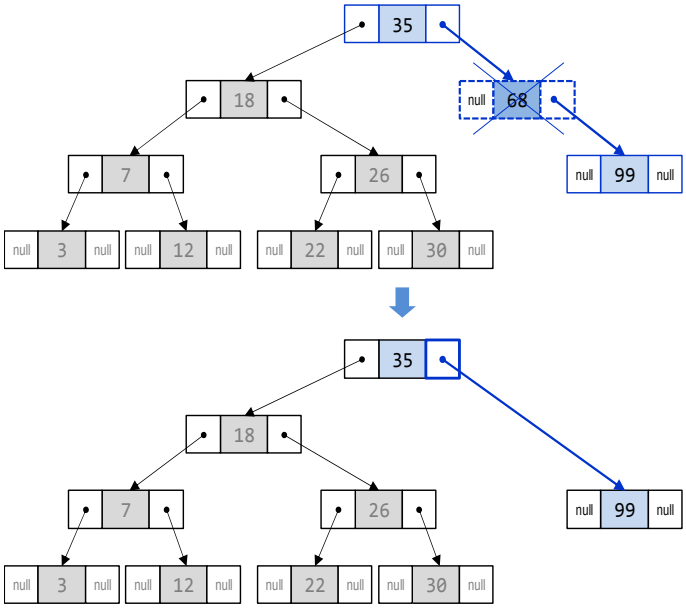
Case 1: 단말 노드 삭제

- 단말노드의 부모노드를 찾아서 연결을 끊으면 된다.



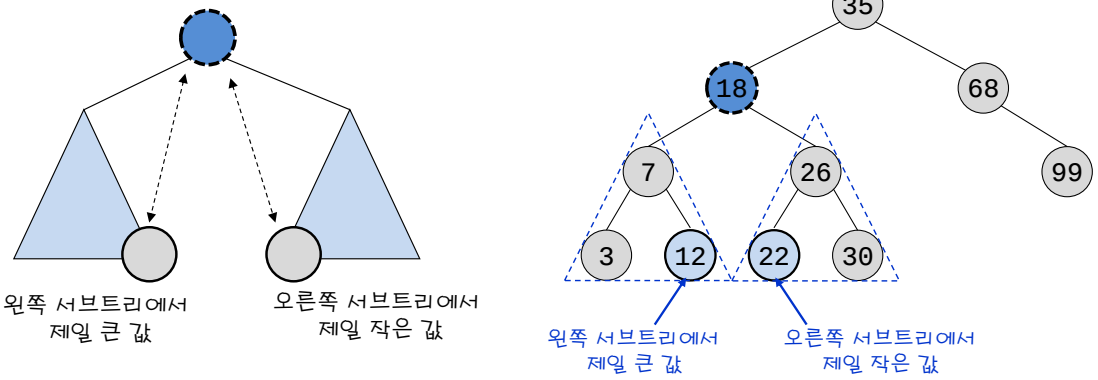
Case 2: 자식이 하나인 노드 삭제

- 노드는 삭제하고 서브 트리는 부모 노드에 붙여줌

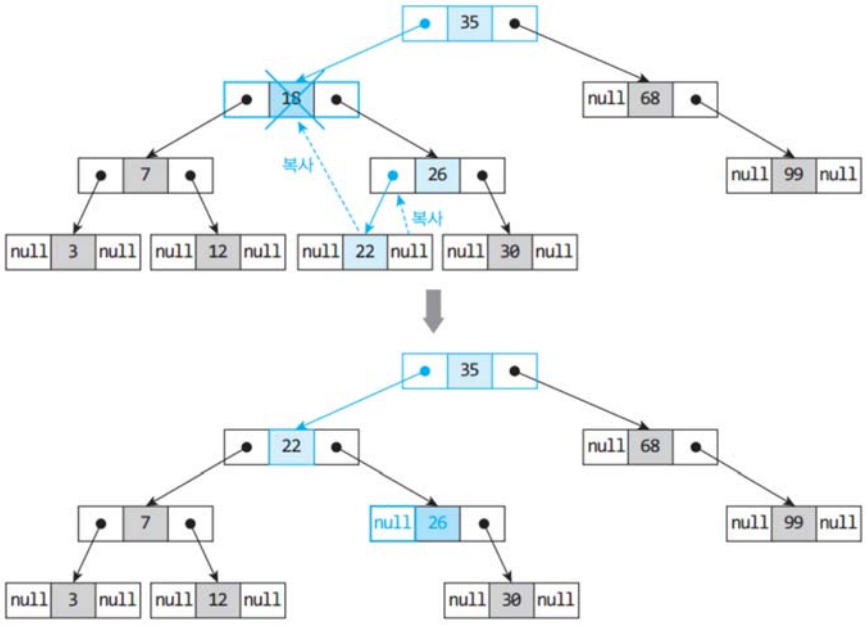


Case 3: 두 개의 자식을 가진 노드 삭제

- 가장 비슷한 값을 가진 노드를 삭제노드 위치로 가져옴
- 후계 노드의 선택



- Case3: 노드 18 삭제 과정



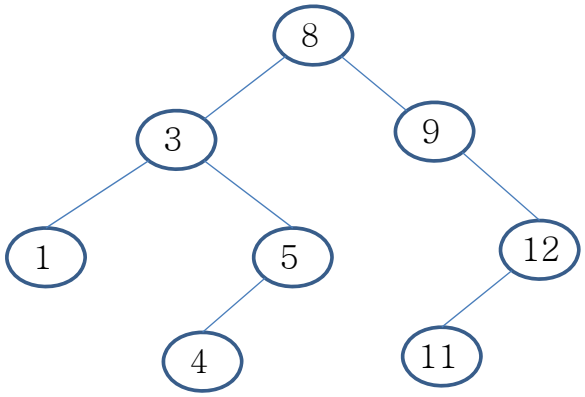
```
// 이진탐색트리 삭제
void delete (TNode *parent, TNode *node)
{
TNode *child, *succ, *succp;

// case 1
if ((node->left == NULL && node->right == NULL)) {
    if (parent == NULL) root = NULL;
    else {
        if (parent->left == node)
                parent->left = NULL;
        else parent->right = NULL;
    }
}
```



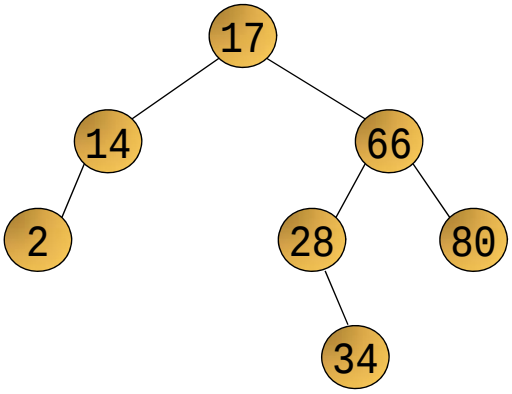
```
// case 2
else if (node->left == NULL || node->right == NULL) {
child = (node->left != NULL) ? node->left : node->right;
    if (node == root) root = child;
    else {
        if (parent->left == node)
                parent->left = child;
        else parent->right = child;
    }
}
```

```
// case 3
else {
succp = node;  // 후계자 부모 시작점
succ = node->right; // 후계자 찾기 시작점
while (succ->left != NULL) { // 시작점의 왼쪽 끝까지 찾기
        succp = succ;
        succ = succ->left;
}
// 후계자의 우측 자식을 연결시키는 부분
if (succp->left == succ) // 후계자가 왼쪽에서 찾아진 경우
        succp->left = succ->right; // 후계자 우자식을 연결
else succp->right = succ->right; // 후계자가 바로 우자식일때

node->data = succ->data;
node = succ;
}
free(node);
}
```


- 다음의 이진탐색트리에서 8을 가지고 있는 노드를 삭제한 후에 재구성되는 트리를 그려라.



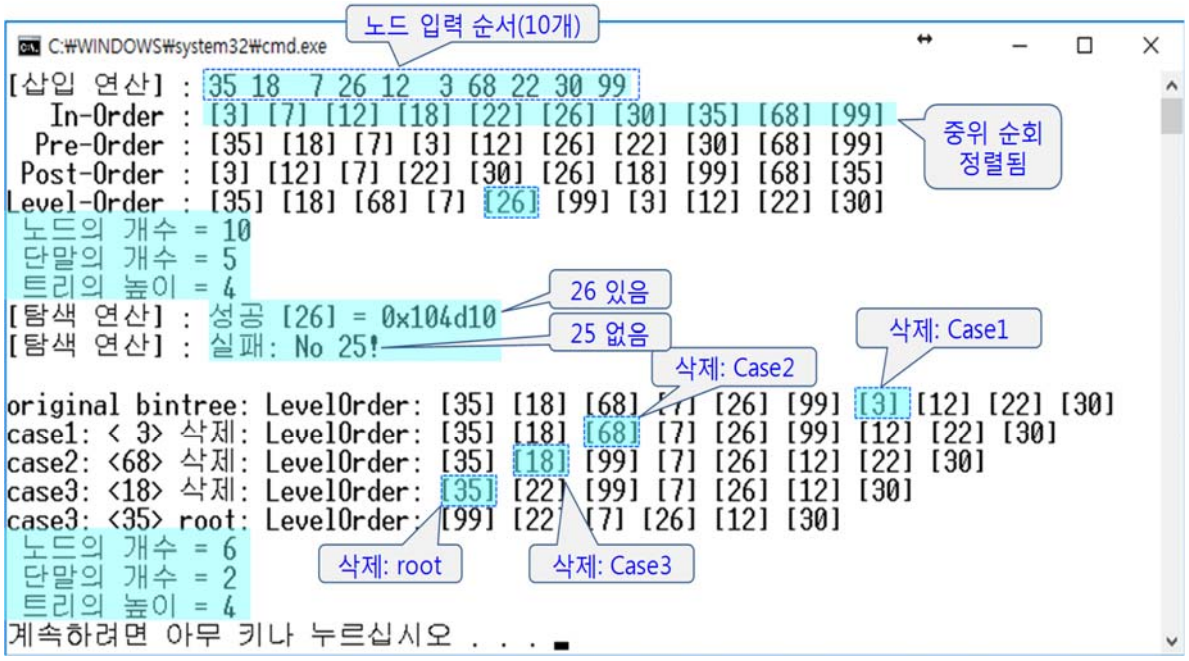
- 다음의 이진 탐색 트리에서 17을 삭제했을 경우를 그리시오



- 정수 데이터가 이진 탐색 트리에 저장되어 있다. 이러한 이진 탐색 트리가 공백 상태로부터 다음과 같은 순서로 연산이 실행된다. 연산들에 의해 생성되는 이진 탐색 트리를 순서대로 그려라.

1. 삽입 5, 7, 2, 8, 3
2. 삭제 3
3. 삽입 4, 3
4. 삭제 7
5. 삭제 5

전체 프로그램 실행결과

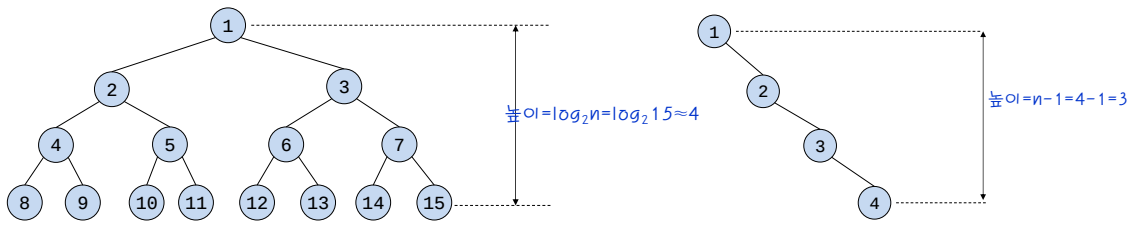


9.3 이진탐색트리의 성능

- 이진 탐색 트리에서의 탐색, 삽입, 삭제 연산의 시간 복잡도는 트리의 높이를 h라고 했을때 h에 비례한다

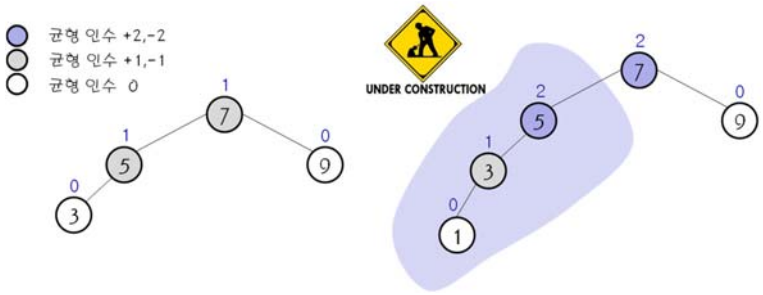
□ **최선의 경우**
- 이진 트리가 균형적으로 생성되어 있는 경우: h=log2n
- 시간복잡도: **O(logn)**

□ **최악의 경우**
- 경사이진트리: h=n
- 시간복잡도: **O(n)**



9.4 균형 이진탐색트리

- AVL 트리
  - Adelson-Velskii와 Landis에 의해 1962년에 제안된 트리
  - **모든 노드의 왼쪽과 오른쪽 서브트리의 높이 차가 1이하**
  - 비균형 상태로 되면 스스로 노드들을 재배치하여 균형상태 유지
  - 평균, 최선, 최악 시간복잡도: O(log(n))
  - 균형 인수
    - balance factor =왼쪽서브트리 높이 - 오른쪽서브트리 높이
  - 모든 노드의 균형 인수가 ±1 이하이면 AVL 트리

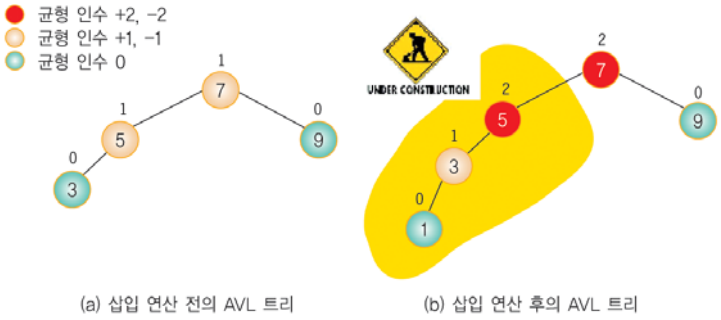


AVL 트리의 연산

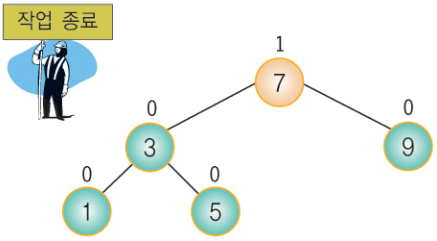
- 탐색연산: 이진탐색트리와 동일
- 삽입 연산과 삭제 연산 시 균형 상태가 깨질 수 있음
- 삽입 연산
  - 삽입 위치에서 루트까지의 경로에 있는 조상 노드들의 균형 인수에 영향을 미침
  - 삽입 후에 불균형 상태로 변한 가장 가까운 조상 노드(균형 인수가 ±2가 된 가장 가까운 조상 노드)의 서브 트리들에 대하여 다시 재균형
  - 삽입 노드부터 균형 인수가 ±2가 된 가장 가까운 조상 노드까지 회전

AVL 트리의 삽입연산

- 삽입 연산의 예
  - 노드 1을 트리에 추가



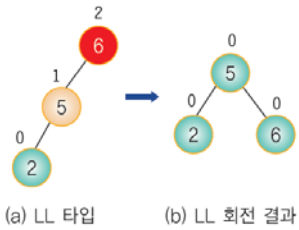
(a) 삽입 연산 전의 AVL 트리 (b) 삽입 연산 후의 AVL 트리



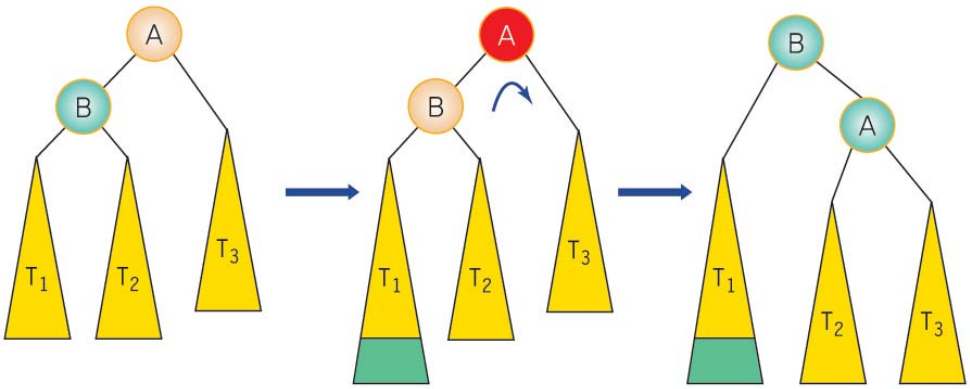
AVL 트리의 삽입연산

- AVL 트리의 균형이 깨지는 4가지 경우
  - 삽입된 노드 N으로부터 가장 가까우면서 균형 인수가 ±2가 된 조상 노드가 A라면
    - LL 타입: N이 A의 왼쪽서브트리의 왼쪽서브트리에 삽입
    - LR 타입: N이 A의 왼쪽서브트리의 오른쪽서브트리에 삽입
    - RR 타입: N이 A의 오른쪽서브트리의 오른쪽서브트리에 삽입
    - RL 타입: N이 A의 오른쪽서브트리의 왼쪽서브트리에 삽입
  - 각 타입별 재균형 방법
    - LL 회전: A부터 N까지의 경로상 노드의 오른쪽 회전
    - LR 회전: A부터 N까지의 경로상 노드의 왼쪽-오른쪽 회전
    - RR 회전: A부터 N까지의 경로상 노드의 왼쪽 회전
    - RL 회전: A부터 N까지의 경로상 노드의 오른쪽-왼쪽 회전

LL 회전 방법

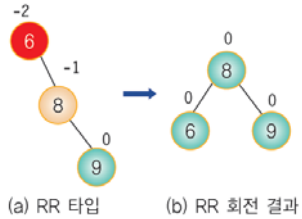


(a) LL 타입 (b) LL 회전 결과

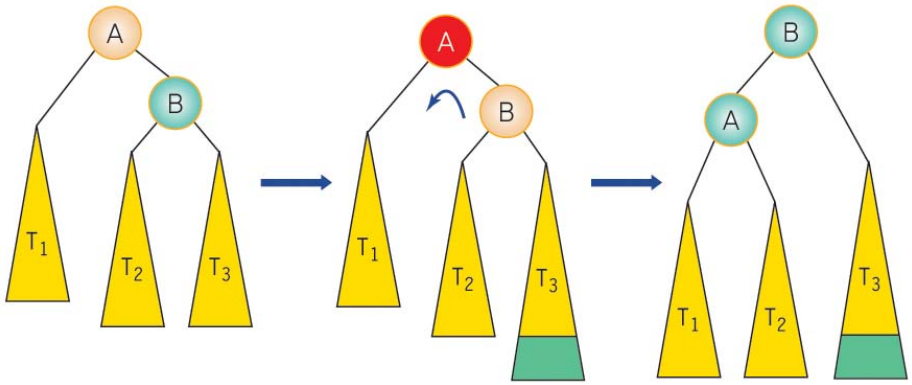



```
TNode *rotateLL(TNode* A)
{
        TNode* B = A->left;
        A->left = B->right;
        B->right = A;
        return B;
}
```

RR 회전 방법

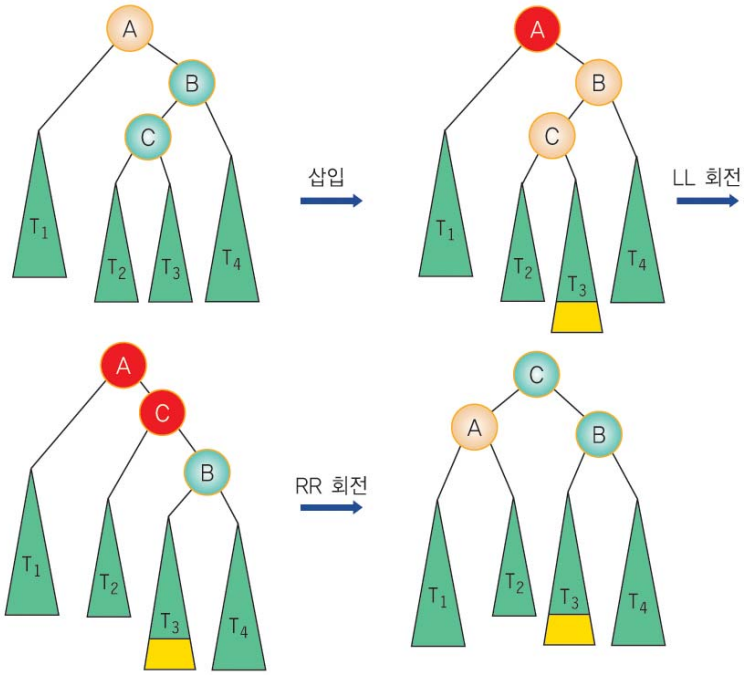


(a) RR 타입 (b) RR 회전 결과

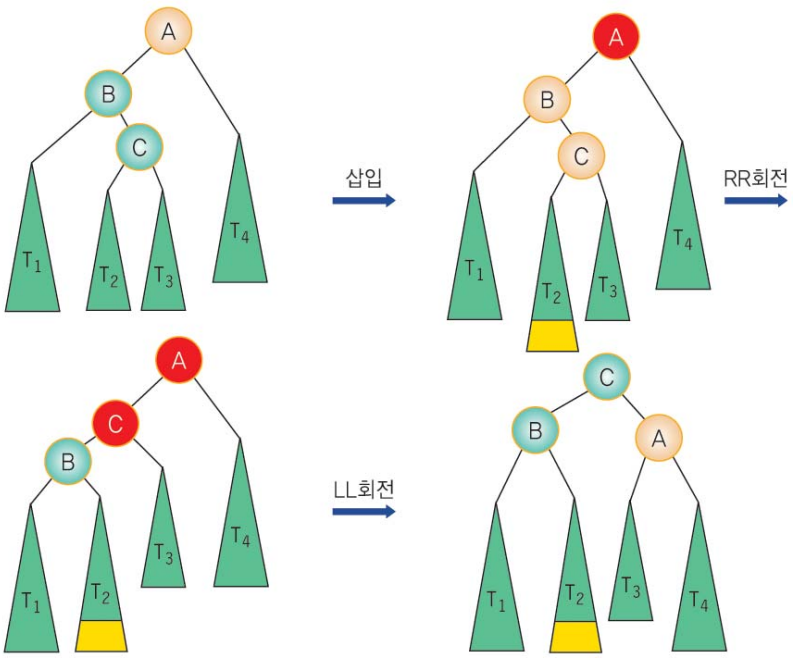



```
TNode *rotateRR(TNode* A) {
        TNode* B = A->right;
        A->right = B->left;
        B->left = A;
        return B;
}
```

RL 회전 방법



LR 회전 방법



AVL 트리 회전 알고리즘

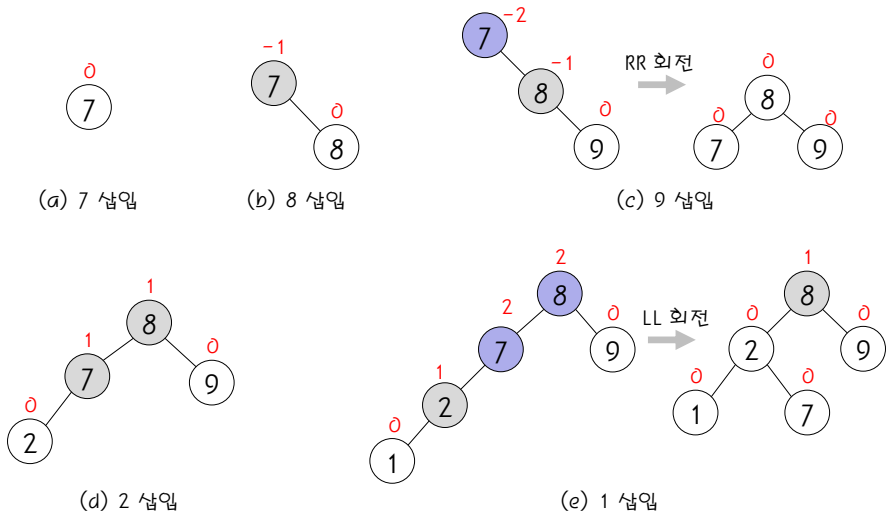
- 단순 회전 알고리즘

```
rotate_LL(A)
        B의 오른쪽 자식을 A의 왼쪽 자식으로 만든다
        A를 B의 오른쪽 자식 노드로 만든다.

rotate_RR(A)
        B의 왼쪽 자식을 A의 오른쪽 자식으로 만든다
        A를 B의 왼쪽 자식 노드로 만든다.
```

- 이중 회전 알고리즘

```
rotate_RL(A)
        rotate_LL(B)가 반환하는 노드를 A의 오른쪽 자식으로 만든다
        rotate_RR(A)

rotate_LR(A)
        rotate_RR(B)가 반환하는 노드를 A의 왼쪽 자식으로 만든다
        rotate_LL(A)
```

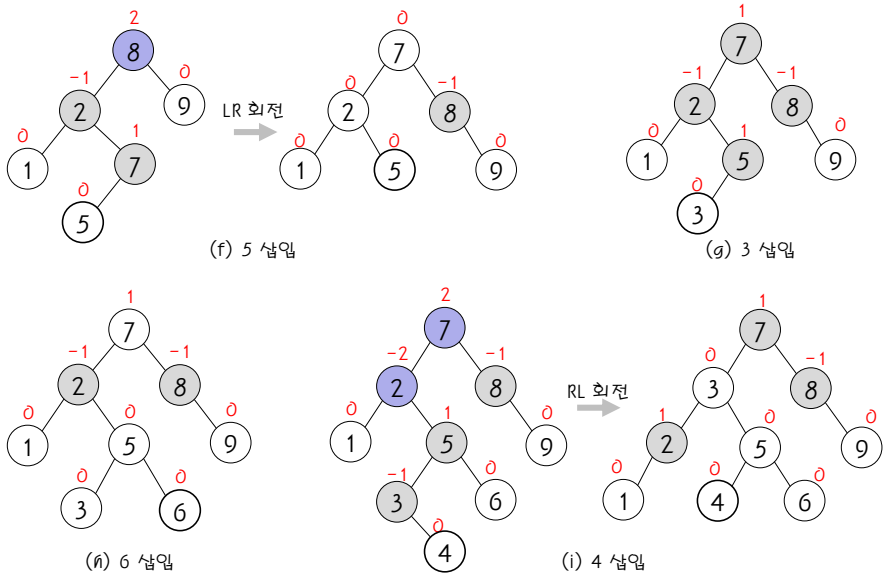

AVL 트리 생성 예



(a) 7 삽입 (b) 8 삽입 (c) 9 삽입

(d) 2 삽입 (e) 1 삽입

AVL 트리 생성 예 (계속)



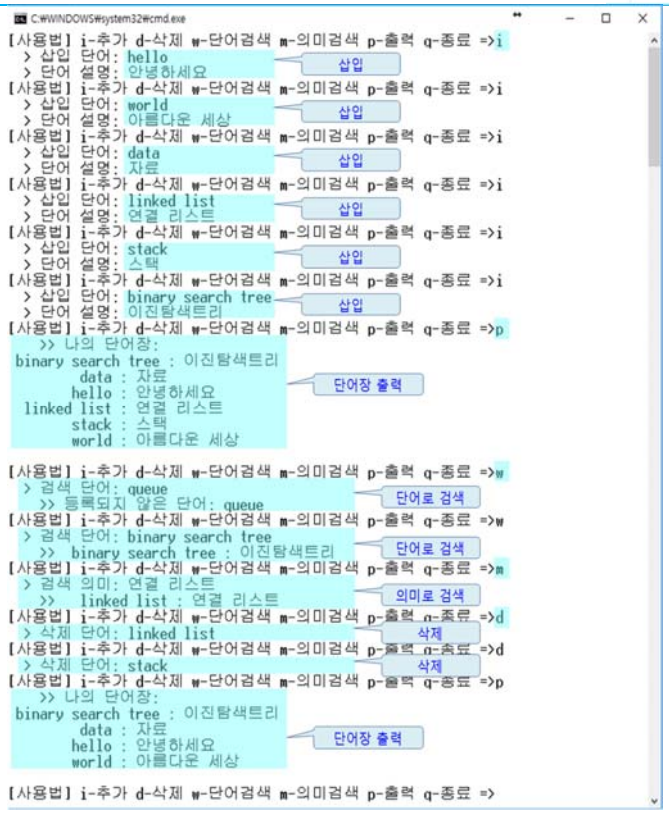
이진탐색트리의 응용 : 나의 단어장

- 기능
  - 입력(i) : 단어와 의미를 입력하여 하나의 노드 추가
  - 삭제(d) : 단어를 입력하면 해당 노드를 찾아 트리에서 제거
  - 단어 탐색(w) : 단어를 입력하면 해당 "단어"의 노드를 찾아 "의미"를 출력
  - 의미 탐색(m) : 의미를 입력하면 해당 "의미"의 노드를 찾아 "단어"를 출력
  - 사전 출력(p) : 사전의 모든 단어를 알파벳 순서대로 화면에 출력
  - 종료(q) : 프로그램을 종료

```
typedef struct DicRecord {
        char word[MAX_WORD_SIZE];
        char meaning[MAX_MEANING_SIZE];
} Record;
typedef Record TElement;
```

이진탐색트리 실행결과

- 교재코드 참조



실습

- 이진탐색트리를 이용하여 친구들의 연락처를 저장하고 탐색하는 프로그램을 작성하라.

  삽입(i), 탐색(s), 삭제(d): i

  친구의 이름: 홍길동

  친구의 전화번호: 010-1234-5678

  …