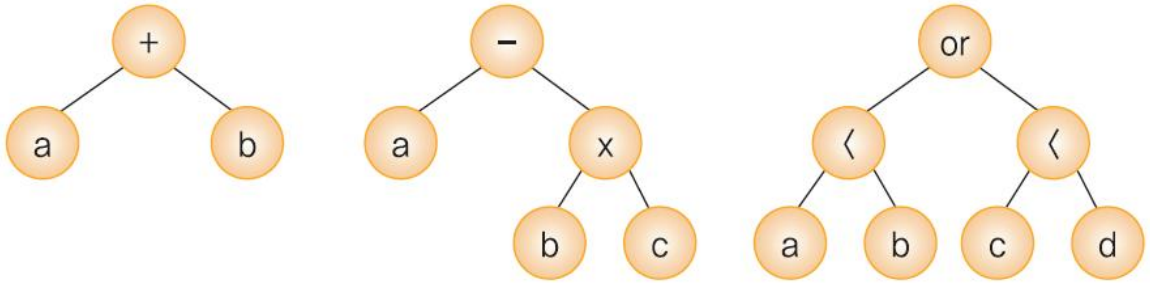
8장 트리 2부

트리 응용, 이진 트리 추가 연산, 스레드 이진 트리, 이진 탐색 트리

8.7 트리의 응용: 수식 트리 처리

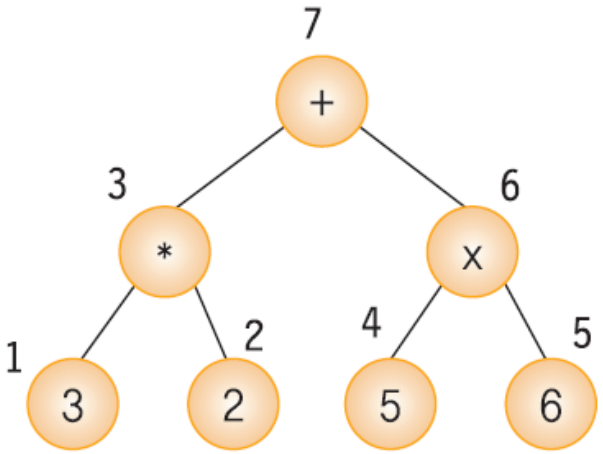
- 수식 트리: 산술식을 트리형태로 표현한 것
  - 비단말노드: 연산자(operator)
  - 단말노드: 피연산자(operand)
- 예)



수식	a + b	a − (b × c)	(a ⟨ b) or (c ⟨ d)
전위순회	+ a b	− a × b c	or < a b < c d
중위순회	a + b	a − (b × c)	(a < b) or (c < d)
후위순회	a b +	a b c × −	a b < c d < or

수식 트리를 이용 수식 계산

- 후위순회를 사용
- 서브 트리의 값을 순환호출로 계산
- 비단말노드를 방문할 때 양쪽 서브 트리의 값을 노드에 저장된 연산자를 이용하여 계산한다



수식 트리 알고리즘 (순환 호출 사용)


```
evaluate(exp)

1.  if exp == NULL then
2.          return 0;
3.  if (exp->left == NULL) and (exp->right == NULL) then
4.          return exp->data;
5.  else
6.          x←evaluate(exp->left);
7.          y←evaluate(exp->right);
8.          op←exp->data;
9.          return (x op y);
```

교재 수정

수식 트리 계산 프로그램: exp_eval.c (1/2)


```c
#include <stdio.h>
#include <stdlib.h>

typedef struct TreeNode {
    int data;
    struct TreeNode *left, *right;
} TreeNode;

//            +
//        *       +
//    1     4   16  25
TreeNode n1 = { 1,   NULL, NULL };
TreeNode n2 = { 4,   NULL,  NULL };
TreeNode n3 = { '*',  &n1,  &n2 };
TreeNode n4 = { 16, NULL, NULL };
TreeNode n5 = { 25, NULL, NULL };
TreeNode n6 = { '+', &n4,  &n5 };
TreeNode n7 = { '+', &n3,  &n6 };
TreeNode *exp = &n7;
```

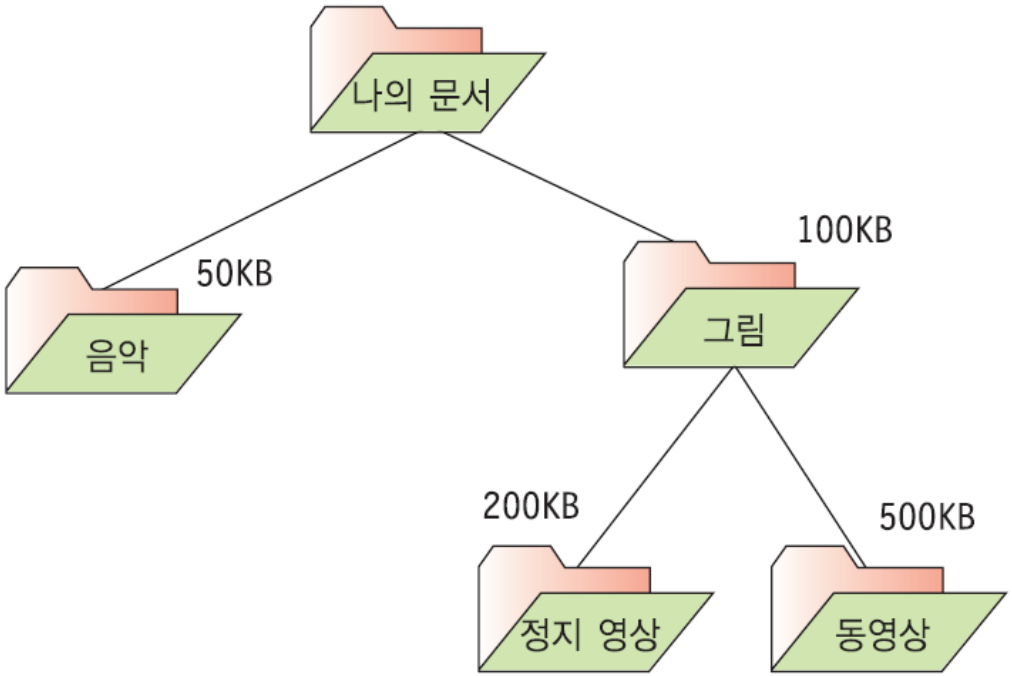
```
1 * 4을 계산합니다.
16 + 25을 계산합니다.
4 + 41을 계산합니다.
수식의 값은 45입니다.

--------------------------------
Process exited after 0.02026 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

수식 트리 계산 프로그램: exp_eval.c (2/2)


```
21  // 수식 계산 함수
22  int evaluate(TreeNode *root)
23  {
24      if (root == NULL)
25          return 0;
26      if (root->left == NULL && root->right == NULL)
27          return root->data;
28      else {
29          int op1 = evaluate(root->left);
30          int op2 = evaluate(root->right);
31          printf("%d %c %d을 계산합니다.\n", op1, root->data, op2);
32          switch (root->data) {
33          case '+':
34              return op1 + op2;
35          case '-':
36              return op1 - op2;
37          case '*':
38              return op1 * op2;
39          case '/':
40              return op1 / op2;
41          }
42      }
43      return 0;
44  }
45
46
47  int main(void)
48  {
49      printf("수식의 값은 %d입니다. \n", evaluate(exp));
50      return 0;
51  }
```

8.8 트리의 응용: 디렉토리 용량 계산

- 디렉토리의 용량을 계산하는데 후위 트리 순회 사용



디렉토리 용량 계산 프로그램: cal_direc.c


```c
#include <stdio.h>
#include <stdlib.h>

typedef struct TreeNode {
    int data;
    struct TreeNode *left, *right;
} TreeNode;

int calc_dir_size(TreeNode *root)
{
    int left_size, right_size;
    if (root == NULL) return 0;

    left_size = calc_dir_size(root->left);
    right_size = calc_dir_size(root->right);
    return (root->data + left_size + right_size);
}
//
int main(void)
{
    TreeNode n4 = { 500, NULL, NULL };
    TreeNode n5 = { 200, NULL, NULL };
    TreeNode n3 = { 100, &n4, &n5 };
    TreeNode n2 = { 50, NULL, NULL };
    TreeNode n1 = { 0, &n2, &n3 };

    printf("디 렉 토 리 의  크 기 =%d\n", calc_dir_size(&n1));
}
```

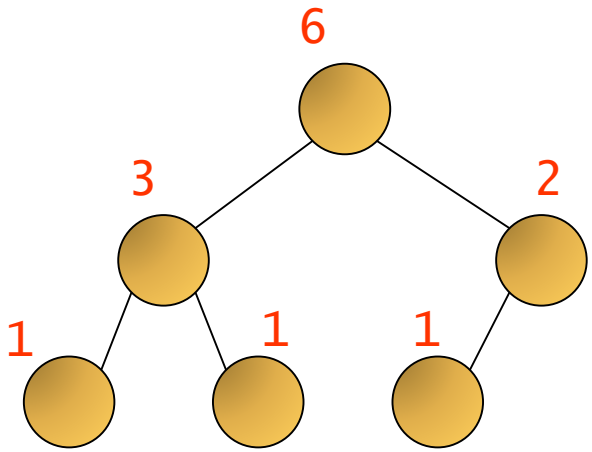
```
디렉토리의 크기=850

--------------------------------
Process exited after 0.01299 seconds with return value 20
계속하려면 아무 키나 누르십시오 . . .
```

8.9 이진 트리의 추가 연산

1) 트리의 노드 개수 구하기
2) 단말 노드 개수 구하기
3) 트리 높이 구하기

이진 트리 연산: 노드 개수

- 탐색 트리안의 노드의 개수를 계산
- 각각의 서브트리에 대하여 순환 호출한 다음, 반환되는 값에 1을 더하여 반환




```
1   int get_node_count(TreeNode *node)
2   {
3       int count = 0;
4
5       if (node != NULL)
6           count = 1 + get_node_count(node->left) +
7           get_node_count(node->right);
8
9       return count;
10  }
```

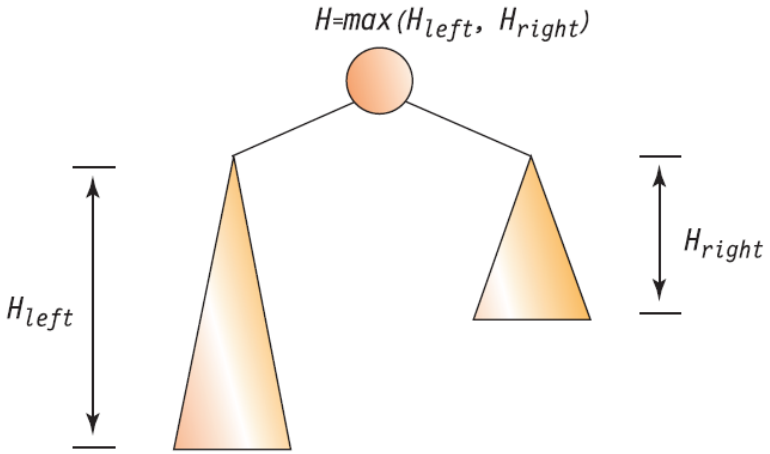
이진 트리 연산: 단말 노드 개수

- 트리의 모든 노드를 순회하여,
  - 왼쪽 링크와 오른쪽 링크 모두가 NULL이면 단말 노드이므로 return 1;
  - 아닐 경우(비단말 노드)에는 왼쪽, 오른쪽 링크에 대해 순환 호출 후, 반환 값들을 합산

```
12  int get_leaf_count(TreeNode *node)
13  {
14      int count = 0;
15
16      if (node != NULL) {
17          if (node->left == NULL && node->right == NULL)
18              return 1;
19          else
20              count = get_leaf_count(node->left) +
21              get_leaf_count(node->right);
22      }
23      return count;
24  }
```

이진 트리 연산: 높이 구하기

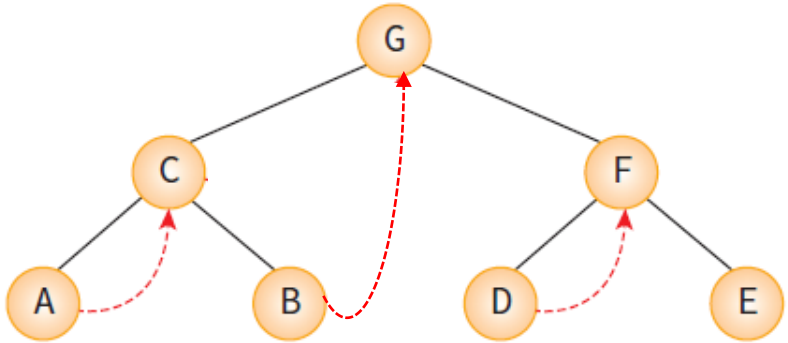
- 서브트리에 대하여 순환호출하고 서브 트리들의 반환값 중에서 최대값을 구하여 반환




```
26  int get_height(TreeNode *node)
27  {
28      int height = 0;
29
30      if (node != NULL)
31          height = 1 + max(get_height(node->left),
32              get_height(node->right));
33
34      return height;
35  }
```

8.10 스레드 이진 트리

- 이진 트리의 링크 중 많은 수의 링크들이 NULL 값을 가짐
  - n개의 노드 트리는 2n개의 링크를 가짐, 이중 최대 n+1개의 링크는 NULL로 설정될 수 있음
- **스레드 이진 트리(Threaded Binary Tree)**
  - 이진 트리의 NULL 링크를 이용하여 순환 호출 없이도 트리의 노드들을 순회
  - NULL 링크에 중위 순회 시에 후속 노드인 **중위 후속자(inorder successor)**를 저장
  - 연산 비용이 비교적 큰 recursion 대신 중위 순회를 간단히 구현할 수 있음



교재 그림 8-36 수정

스레드 이진 트리의 노드 정의

- 단말노드와 비단말노드의 구별을 위하여 is_thread 필드 사용
  - is_thread가 참: right 링크는 중위 순위의 후속 노드를 포인팅
  - is_thread가 거짓: right 링크는 오른쪽 자식 노드를 포인팅

```
5  typedef struct TreeNode {
6      int data;
7      struct TreeNode *left, *right;
8      int is_thread;  // 스레드이면 TRUE
9  } TreeNode;
```

스레드 이진 트리의 순회: thread_tree.c (1/3)


```c
#include <stdio.h>
#define TRUE 1
#define FALSE 0

typedef struct TreeNode {
    int data;
    struct TreeNode *left, *right;
    int is_thread;  // 스레드이면 TRUE
} TreeNode;


//            G
//        C         F
//    A     B   D     E
TreeNode n1 = { 'A',  NULL, NULL, 1 };
TreeNode n2 = { 'B',  NULL,  NULL, 1 };
TreeNode n3 = { 'C',  &n1,  &n2, 0 };
TreeNode n4 = { 'D', NULL, NULL, 1 };
TreeNode n5 = { 'E', NULL, NULL, 0 };
TreeNode n6 = { 'F', &n4,  &n5, 0 };
TreeNode n7 = { 'G', &n3,  &n6, 0 };
TreeNode * exp = &n7;
```

스레드 이진 트리의 순회: thread_tree.c (2/3)


```
24  TreeNode *find_successor(TreeNode * p)
25  {
26      // q는 p의 오른쪽 포인터
27      TreeNode * q = p->right;
28      // 만약 오른쪽 포인터가 NULL이거나 스레드이면 오른쪽 포인터를 반환
29      if (q == NULL || p->is_thread == TRUE)
30          return q;
31
32      // 만약 오른쪽 자식이면 다시 가장 왼쪽 노드로 이동
33      while (q->left != NULL) q = q->left;
34      return q;
35  }
```

- find_successor() 함수
  - 현재 노드의 right 링크인 q가 NULL 이면, (더 이상 후속자 없음) NULL 리턴
  - q가 스레드로 사용된다면 q를 리턴
  - q가 오른쪽 자식을 포인팅한다면, 왼쪽 서브 트리의 제일 왼쪽 노드를 리턴

스레드 이진 트리의 순회: thread_tree.c (3/3)

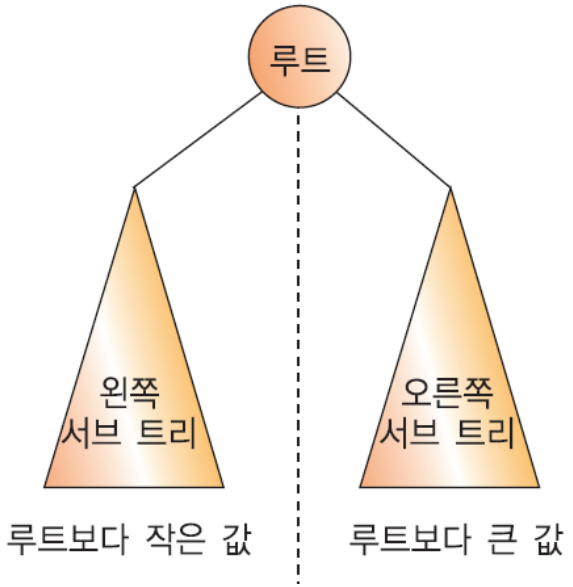

```c
void thread_inorder(TreeNode * t)
{
    TreeNode * q;

    q = t;
    while (q->left) q = q->left;     // 가장 왼쪽 노드로 간다.
    do {
        printf("%c -> ", q->data);   // 데이터 출력
        q = find_successor(q);       // 후속자 함수 호출
    } while (q);                     // NULL이 아니면
}

int main(void)
{
    // 스레드 설정
    n1.right = &n3;
    n2.right = &n7;
    n4.right = &n6;
    // 중위 순회
    thread_inorder(root);
    printf("\n");
    return 0;
}
```

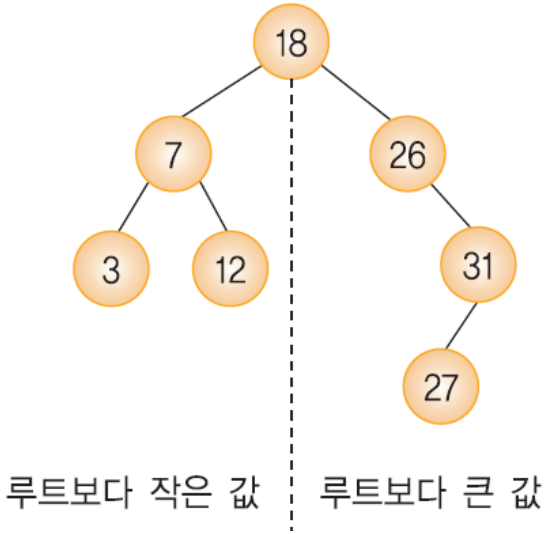
```
A -> C -> B -> G -> D -> F -> E ->

--------------------------------
Process exited after 1.044 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

```c
//           G
//       C       F
//    A    B   D    E
TreeNode n1 = { 'A',  NULL, NULL, 1 };
TreeNode n2 = { 'B',  NULL,  NULL, 1 };
TreeNode n3 = { 'C',  &n1,  &n2, 0 };
TreeNode n4 = { 'D', NULL, NULL, 1 };
TreeNode n5 = { 'E', NULL, NULL, 0 };
TreeNode n6 = { 'F', &n4,  &n5, 0 };
TreeNode n7 = { 'G', &n3,  &n6, 0 };
TreeNode * exp = &n7;
```

8.11 이진 탐색 트리

- 탐색작업을 효율적으로 하기 위한 자료구조
- key(왼쪽서브트리) ≤ key(루트노드) ≤ key(오른쪽서브트리)
- 이진 탐색 트리를 중위순회하면 오름차순으로 정렬된 값을 얻을 수 있다.



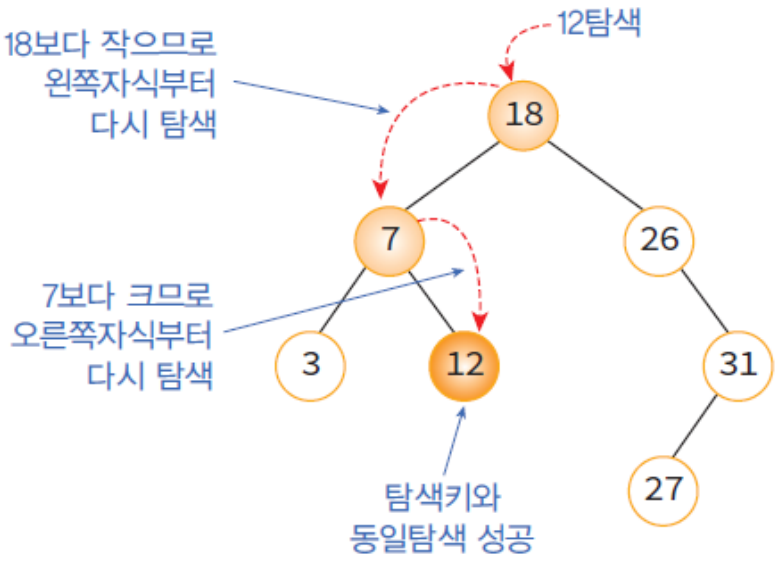


이진 탐색 트리 정의

- 모든 원소들은 유일한 키를 가진다.
- 왼쪽 서브 트리의 키들은 루트의 키보다 작다.
- 오른쪽 서브 트리의 키들은 루트의 키보다 크다.
- 왼쪽과 오른쪽 서브 트리들도 이진 탐색 트리이다.

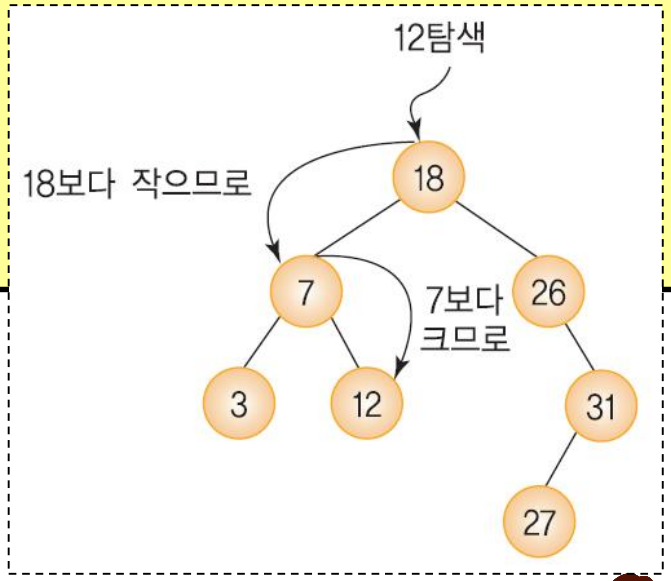
이진 탐색 트리에서의 탐색 방법

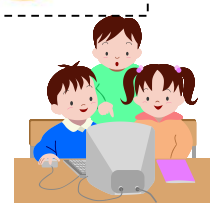
- 비교한 결과가 같으면 탐색이 성공적으로 끝난다.
- 비교한 결과, 주어진 키 값이 루트 노드의 키 값보다 작으면 탐색은 이 루트 노드의 왼쪽 자식을 기준으로 다시 시작한다.
- 비교한 결과, 주어진 키 값이 루트 노드의 키 값보다 크면 탐색은 이 루트 노드의 오른쪽 자식을 기준으로 다시 시작한다.



이진 탐색 트리에서의 탐색(순환) 알고리즘


```
search (root, key):

if root == NULL then              // 탐색 실패
            return NULL;
if key == KEY(root) then
            return root;
else if key < KEY(root) then
            return search(LEFT(root), k);
else
            return search(RIGHT(root), k);
```





탐색을 구현하는 두가지 방법

1) 순환적 방법
2) 반복적 방법

- 이진 탐색 트리 구조

```
4   typedef int element;
5   typedef struct TreeNode {
6       element key;
7       struct TreeNode *left, *right;
8   } TreeNode;
```

순환적인 방법 구현 함수

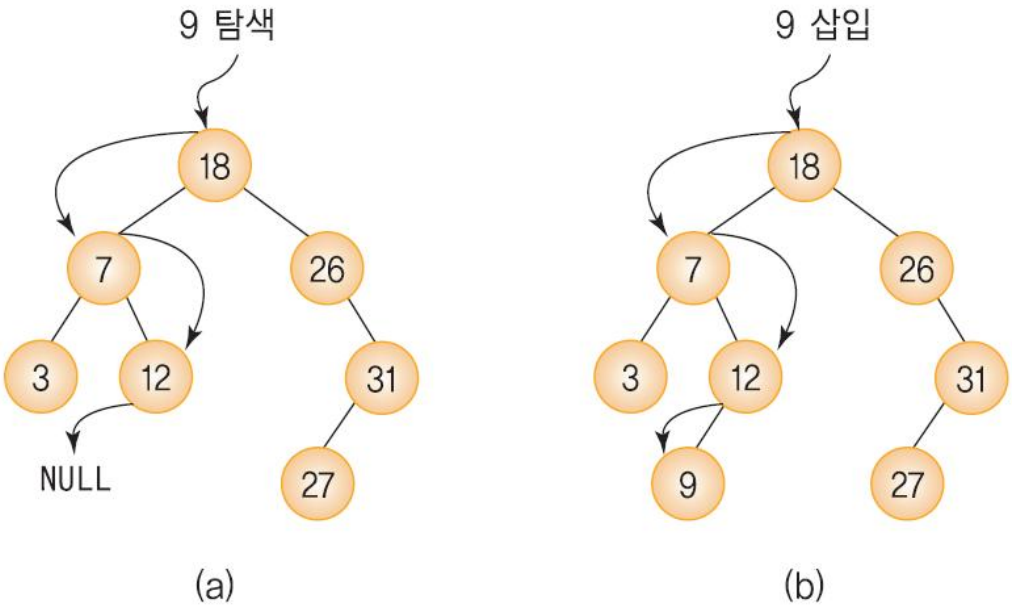

```
10  // 순환적인 탐색 함수
11  TreeNode *search(TreeNode * node, int key)
12  {
13      if (node == NULL)    // 탐색 실패
14          return NULL;
15
16      if (key == node->key)
17          return node;
18      else if (key < node->key)
19          return search(node->left, key);
20      else
21          return search(node->right, key);
22  }
```

반복적인 방법 구현 함수


```
25  // 반복적인 탐색 함수
26  TreeNode * search(TreeNode *node, int key)
27  {
28      while (node)
29      {
30          if (key == node->key)
31              return node;
32          else if (key < node->key)
33              node = node->left;
34          else
35              node = node->right
36      }
37      return NULL;     // 탐색 실패
38
39  }
```

이진 탐색 트리에서의 삽입 연산

- 이진 탐색 트리에 원소를 삽입하기 위해서는 먼저 탐색을 수행하는 것이 필요
- 탐색에 실패한 위치가 바로 새로운 노드를 삽입하는 위치



삽입 연산 알고리즘


```
insert (root, n):

if KEY(n) == KEY(root) then                  // root와 키가 같으면 return
        return;

else if KEY(n) < KEY(root) then              // root보다 키가 작으면
        if LEFT(root) == NULL then           // root의 왼쪽 자식이 없으면
                LEFT(root) ← n;              // root의 왼쪽 자식을 n으로 설정(삽입)
        else
                insert(LEFT(root),n);                // 있으면 순환 호출

else                                         // root보다 키가 크면
        if RIGHT(root) == NULL then          // root의 오른쪽 자식이 없으면
                RIGHT(root) ← n;             // root의 오른쪽 자식을 n으로 설정(삽입)
        else
                insert(RIGHT(root),n);
```

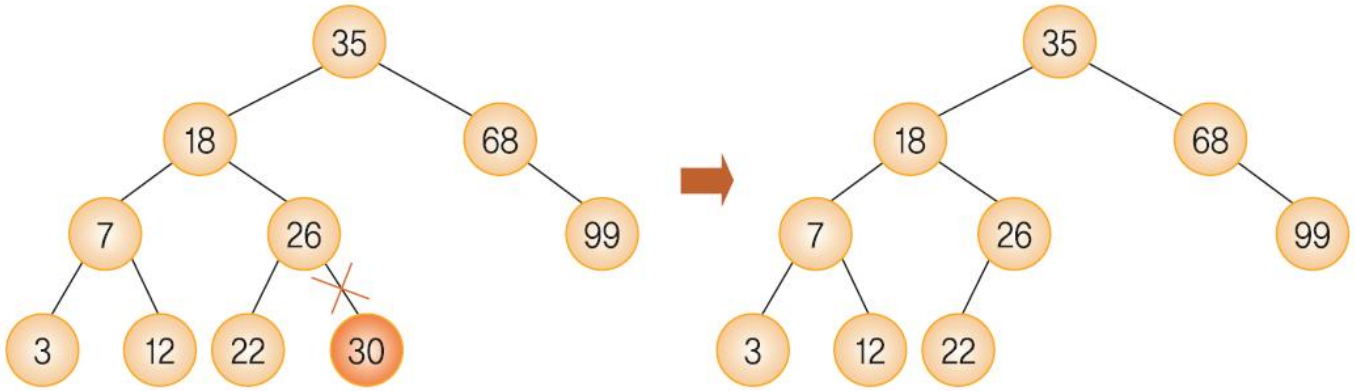
삽입 연산 구현


```
41  TreeNode *new_node(int item)
42  {
43      TreeNode * temp = (TreeNode *)malloc(sizeof(TreeNode));
44      temp->key = item;
45      temp->left = temp->right = NULL;
46      return temp;
47  }
48
49  TreeNode *insert_node(TreeNode *node, int key)
50  {
51      // 트리가 공백이면 새로운 노드를 반환한다.
52      if (node == NULL)
53          return new_node(key);
54
55      // 그렇지 않으면 순환적으로 트리를 내려간다.
56      if (key < node->key)
57          node->left = insert_node(node->left, key);
58      else if (key > node->key)
59          node->right = insert_node(node->right, key);
60
61      // 변경된 루트 포인터를 반환한다.
62      return node;
63  }
```

이진 탐색 트리에서의 삭제 연산

- 3가지의 경우
  1. 삭제하려는 노드가 단말 노드 일 경우
  2. 삭제하려는 노드가 하나의 왼쪽이나 오른쪽 서브 트리 중 하나만 가지고 있는 경우
  3. 삭제하려는 노드가 두개의 서브 트리 모두 가지고 있는 경우

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

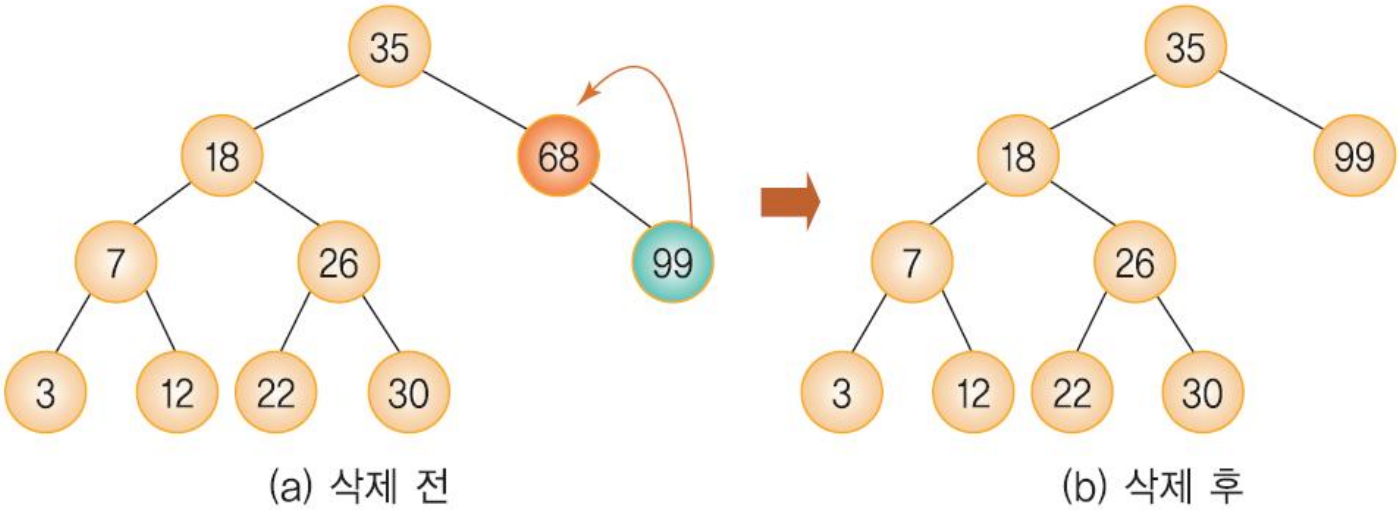


(a) 삭제 전 (b) 삭제 후

이진 탐색 트리에서의 삭제 연산

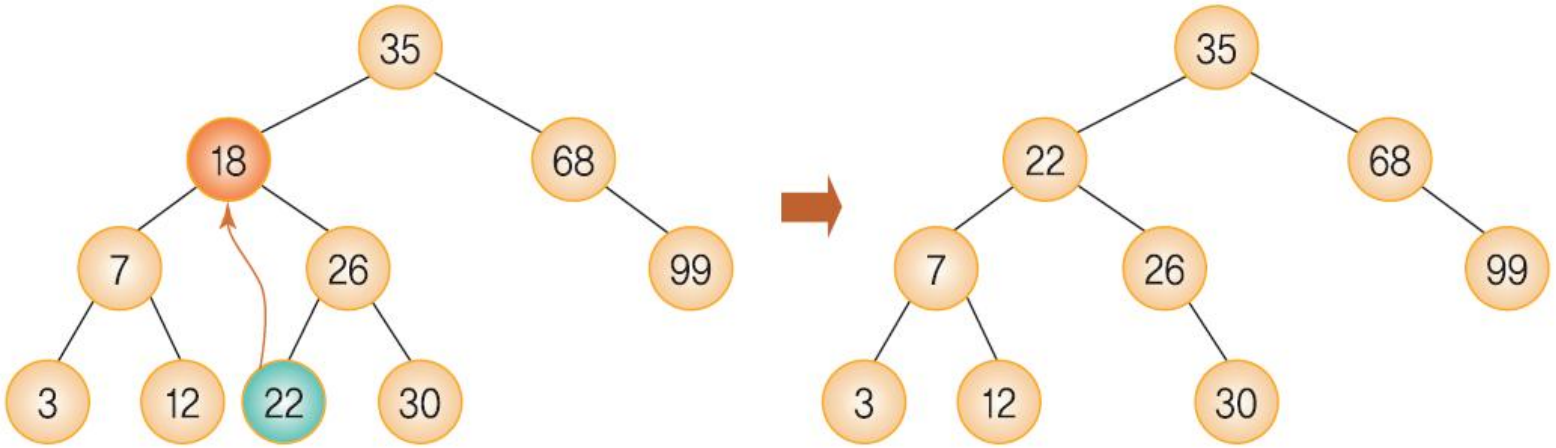
- CASE 2:삭제하려는 노드가 하나의 서브트리만 갖고 있는 경우 :

  삭제되는 노드가 왼쪽이나 오른쪽 서브 트리중 하나만 갖고 있을 때, 그 노드는 삭제하고 서브 트리는 부모 노드에 붙여준다.

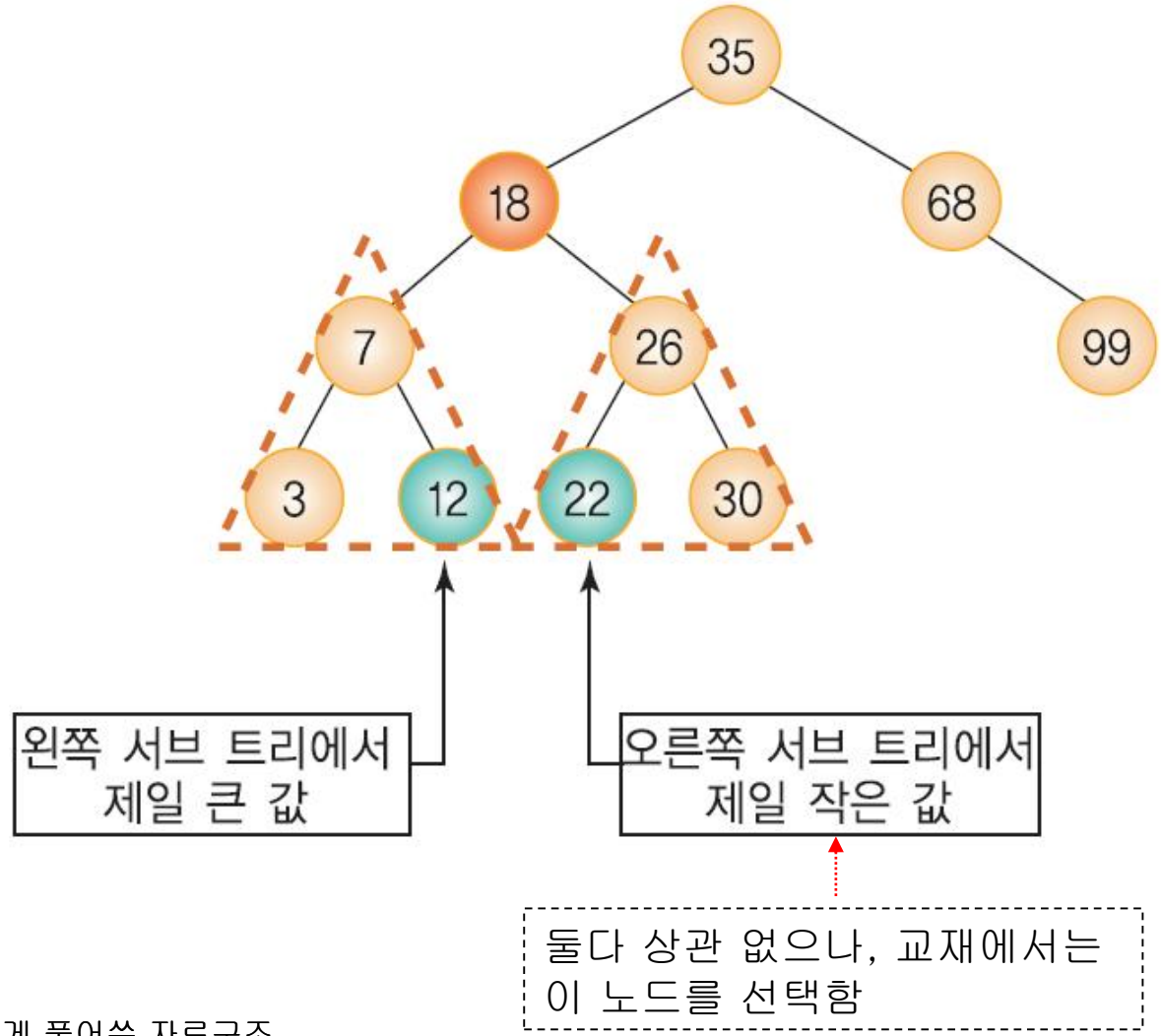


이진 탐색 트리에서의 삭제 연산

- CASE 3:삭제하려는 노드가 두 개의 서브 트리를 갖고 있는 경우: 삭제 노드와 가장 비슷한 값을 가진 노드를 삭제 노드 위치로 가져온다.



CASE 3: 가장 비슷한 값은 어디에 있을까?



삭제 연산 구현 (1/2)


```
65  // 주어진 이진 탐색 트리에서 최소 키값을 가지는 노드를 찾아 반환
66  TreeNode *min_value_node(TreeNode *node)
67  {
68      TreeNode * current = node;
69
70      // 맨 왼쪽 단말 노드를 찾으러 내려감
71      while (current->left != NULL)
72          current = current->left;
73
74      return current;
75  }
76
77  // 이진 탐색 트리와 키가 주어지면 키가 저장된 노드를 삭제하고
78  // 새로운 루트 노드를 반환한다.
79  TreeNode *delete_node(TreeNode *root, int key)
80  {
81      if (root == NULL) return root;
82
83      // 만약 키가 루트보다 작으면 왼쪽 서브 트리에 있는 것임
84      if (key < root->key)
85          root->left = delete_node(root->left, key);
86      // 만약 키가 루트보다 크면 오른쪽 서브 트리에 있는 것임
87      else if (key > root->key)
88          root->right = delete_node(root->right, key);
```

삭제 연산 구현 (2/2)


```
90      // 키가 루트와 같으면 이 노드를 삭제하면 됨
91      else
92      {
93          // 첫 번째나 두 번째 경우
94          if (root->left == NULL) {
95              TreeNode * temp = root->right;
96              free(root);
97              return temp;
98          }
99          else if (root->right == NULL) {
100             TreeNode * temp = root->left;
101             free(root);
102             return temp;
103         }
104         // 세 번째 경우
105         TreeNode * temp = min_value_node(root->right);
106         // 중위 순회시 후계 노드를 복사한다.
107         root->key = temp->key;
108         // 중위 순회시 후계 노드를 삭제한다.
109         root->right = delete_node(root->right, temp->key);
110     }
111     return root;
112 }
```

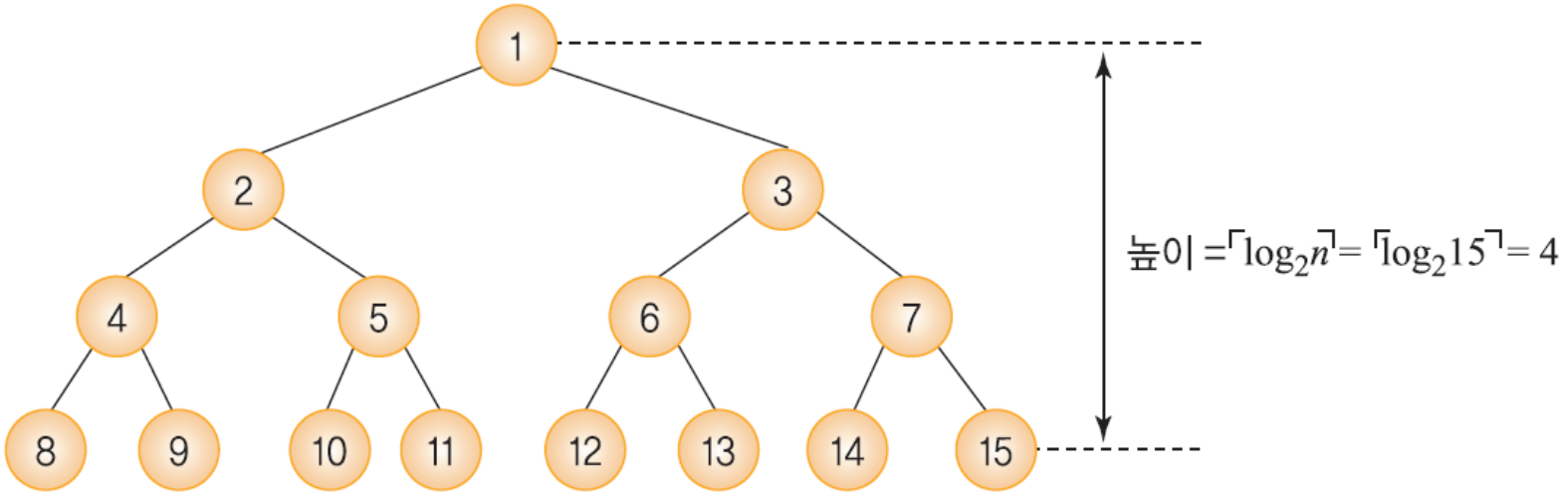
이진 탐색 트리 main() 함수


```c
// 중위 순회
void inorder(TreeNode * root) {
    if (root) {
        inorder(root->left);// 왼쪽 서브트리 순회
        printf("[%d] ", root->key);  // 노드 방문
        inorder(root->right);// 오른쪽 서브트리 순회
    }
}

int main(void)
{
    TreeNode * root = NULL;
    TreeNode * tmp = NULL;

    root = insert_node(root, 30);
    root = insert_node(root, 20);
    root = insert_node(root, 10);
    root = insert_node(root, 40);
    root = insert_node(root, 50);
    root = insert_node(root, 60);

    printf("이진 탐색 트리 중위 순회 결과 \n");
    inorder(root);
    printf("\n\n");
    if (search(root, 30) != NULL)
        printf("이진 탐색 트리에서 30을 발견함 \n");
    else
        printf("이진 탐색 트리에서 30을 발견못함 \n");
    return 0;
}
```

```
이진 탐색 트리 중위 순회 결과
[10] [20] [30] [40] [50] [60]

이진 탐색 트리에서 30을 발견함

--------------------------------
Process exited after 1.421 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

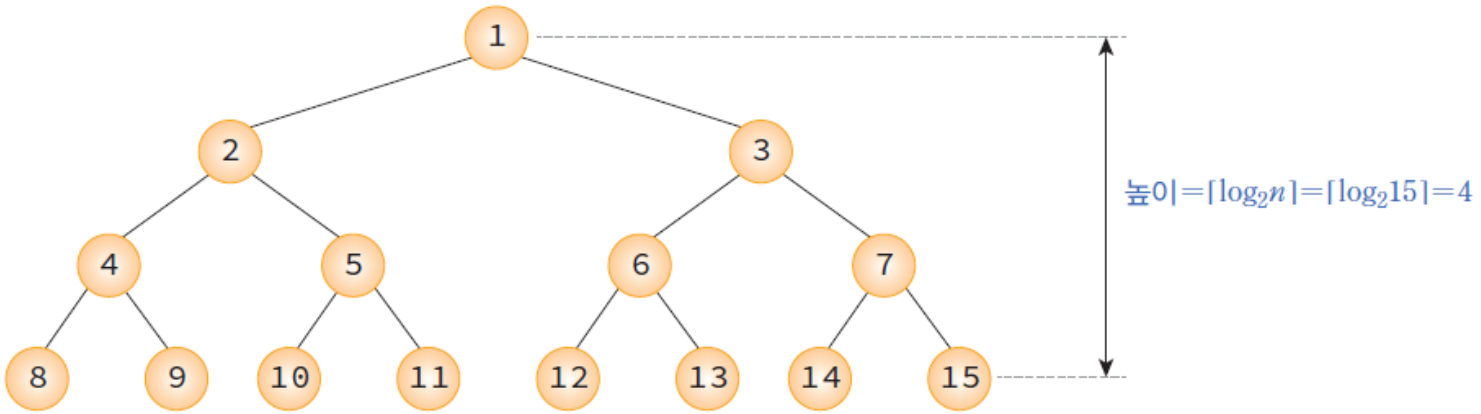
이진 탐색 트리의 성능 분석

- 이진 탐색 트리에서의 탐색, 삽입, 삭제 연산의 시간 복잡도는 트리의 높이를 **h**라고 했을때 **h**에 비례한다



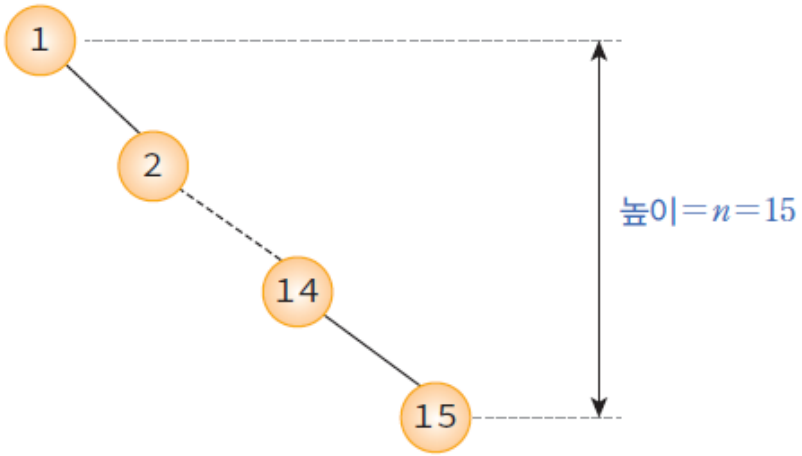
이진 탐색 트리의 성능 분석

- 최선의 경우
  - 이진 트리가 균형적으로 생성되어 있는 경우
  - $h=\log_2 n$ -> $O(\log_2 n)$



이진 탐색 트리의 성능 분석

- 최악의 경우
  - 한쪽으로 치우친 경사이진트리의 경우
  - h=n -> *O(n)*
  - 순차 탐색과 시간복잡도가 같다.



- 해결 방법: AVL 형태의 트리 구성 (13장)

8.12 이진 탐색 트리 응용: 영어사전

- 이진 탐색 트리 응용에 의한 영어사전 구현
  - 메뉴: **** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****
- 실행 예

```
**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****: i
단어:name
의미:이름

**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****: i
단어:picture
의미:그림

**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****: i
단어:free
의미:자유

**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****: i
단어:car
의미:자동차

**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****: p
(((car:자동차)free:자유)name:이름(picture:그림))

**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****: i
단어:student
의미:학생

**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****: s
단어:car
의미:자동차

**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****: q
```

english_dic.c (1/6)

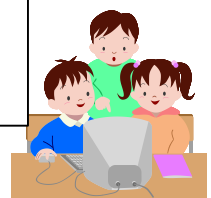

```c
// 이진 탐색 트리를 사용한 영어 사전
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>

#define MAX_WORD_SIZE    100
#define MAX_MEANING_SIZE 200

// 데이터 형식
typedef struct {
    char word[MAX_WORD_SIZE];       // 키 필드
    char meaning[MAX_MEANING_SIZE];
} element;

// 노드의 구조
typedef struct TreeNode {
    element key;
    struct TreeNode* left, * right;
} TreeNode;

// 만약 e1 < e2 이면 -1 반환
// 만약 e1 == e2 이면 0 반환
// 만약 e1 > e2 이면 1 반환
int compare(element e1, element e2)
{
    return strcmp(e1.word, e2.word);
}
```

english_dic.c (2/6)


```
30  // 이진 탐색 트리 출력 함수
31  void display(TreeNode * p)
32  {
33      if (p != NULL) {
34          printf("(");
35          display(p->left);
36          printf("%s:%s", p->key.word, p->key.meaning);
37          display(p->right);
38          printf(")");
39      }
40  }
41
42  // 이진 탐색 트리 탐색 함수
43  TreeNode * search(TreeNode * root, element key)
44  {
45      TreeNode * p = root;
46      while (p != NULL) {
47          if (compare(key, p->key) == 0)
48              return p;
49          else if (compare(key, p->key) < 0)
50              p = p->left;
51          else if (compare(key, p->key) > 0)
52              p = p->right;
53      }
54      return p;    // 탐색에 실패했을 경우 NULL 반환
55  }
```

english_dic.c (3/6)


```
57  TreeNode * new_node(element item)
58  {
59      TreeNode * temp = (TreeNode *)malloc(sizeof(TreeNode));
60      temp->key = item;
61      temp->left = temp->right = NULL;
62      return temp;
63  }
64
65  TreeNode * insert_node(TreeNode * node, element key)
66  {
67      // 트리가 공백이면 새로운 노드를 반환한다.
68      if (node == NULL) return new_node(key);
69
70      // 그렇지 않으면 순환적으로 트리를 내려간다.
71      if (compare(key, node->key)<0)
72          node->left = insert_node(node->left, key);
73      else if (compare(key, node->key)>0)
74          node->right = insert_node(node->right, key);
75      // 루트 포인터를 반환한다.
76      return node;
77  }
78
79  TreeNode * min_value_node(TreeNode * node)
80  {
81      TreeNode * current = node;
82      // 맨 왼쪽 단말 노드를 찾으러 내려감
83      while (current->left != NULL)
84          current = current->left;
85      return current;
86  }
```



english_dic.c (4/6)


```
88  // 이진 탐색 트리와 키가 주어지면 키가 저장된 노드를 삭제하고
89  // 새로운 루트 노드를 반환한다.
90  TreeNode * delete_node(TreeNode * root, element key)
91  {
92      if (root == NULL) return root;
93
94      // 만약 키가 루트보다 작으면 왼쪽 서브 트리에 있는 것임
95      if (compare(key, root->key)<0)
96          root->left = delete_node(root->left, key);
97      // 만약 키가 루트보다 크면 오른쪽 서브 트리에 있는 것임
98      if (compare(key, root->key)>0)
99          root->right = delete_node(root->right, key);
100     // 키가 루트와 같으면 이 노드를 삭제하면 됨
101     else {
102         // 첫 번째나 두 번째 경우
103         if (root->left == NULL) {
104             TreeNode * temp = root->right;
105             free(root);
106             return temp;
107         }
108         else if (root->right == NULL) {
109             TreeNode * temp = root->left;
110             free(root);
111             return temp;
112         }
113         // 세 번째 경우.
114         TreeNode * temp = min_value_node(root->right);
115
116         // 중외 순회시 후계 노드를 복사한다.
117         root->key = temp->key;
118         // 중외 순회시 후계 노드를 삭제한다.
119         root->right = delete_node(root->right, temp->key);
120     }
121     return root;
122 }
```

english_dic.c (5/6)


```
124  //
125  void help()
126  {
127      printf("\n**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료  ****: ");
128  }
129
130  // 이진 탐색 트리를 사용하는 영어 사전 프로그램
131  int main(void)
132  {
133      char command;
134      element e;
135      TreeNode * root = NULL;
136      TreeNode * tmp;
137
138      do {
139          help();
140          command = getchar();
141          getchar();       // 엔터키 제거
142          switch (command) {
143          case 'i':
144              printf("단어:");
145              gets(e.word);
146              printf("의미:");
147              gets(e.meaning);
148              root = insert_node(root, e);
149              break;
150          case 'd':
151              printf("단어:");
152              gets(e.word);
153              root=delete_node(root, e);
154              break;
155          case 'p':
156              display(root);
157              printf("\n");
158              break;
```

교재 소스의 gets_s() 대신 gets()로 수정

english_dic.c (6/6)


```
159             case 's':
160                 printf("단 어 :");
161                 gets(e.word);
162                 tmp = search(root, e);
163                 if (tmp != NULL)
164                     printf("의 미 :%s\n", tmp->key.meaning);
165                 break;
166             }
167     //      display(root);
168
169         } while (command != 'q');
170         return 0;
171 }
```

교재 소스 코드 수정

```
**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****: i
단어:people
의미:사람

**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****: i
단어:korea
의미:대한민국

**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****: i
단어:teacher
의미:선생님

**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****: i
단어:year
의미:년도

**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****: p
((korea:대한민국)people:사람(teacher:선생님(year:년도)))

**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****: d
단어:teacher

**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****: p
((korea:대한민국)people:사람(year:년도))

**** i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 ****:
```