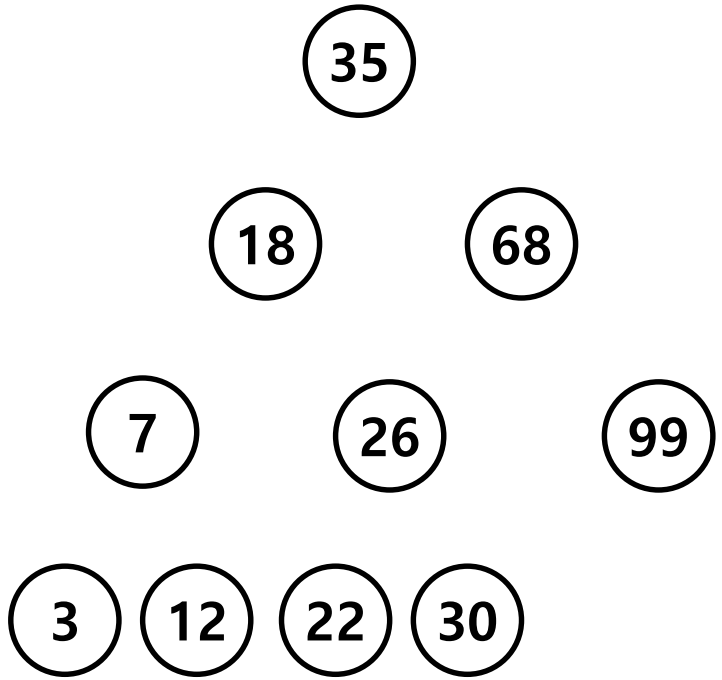
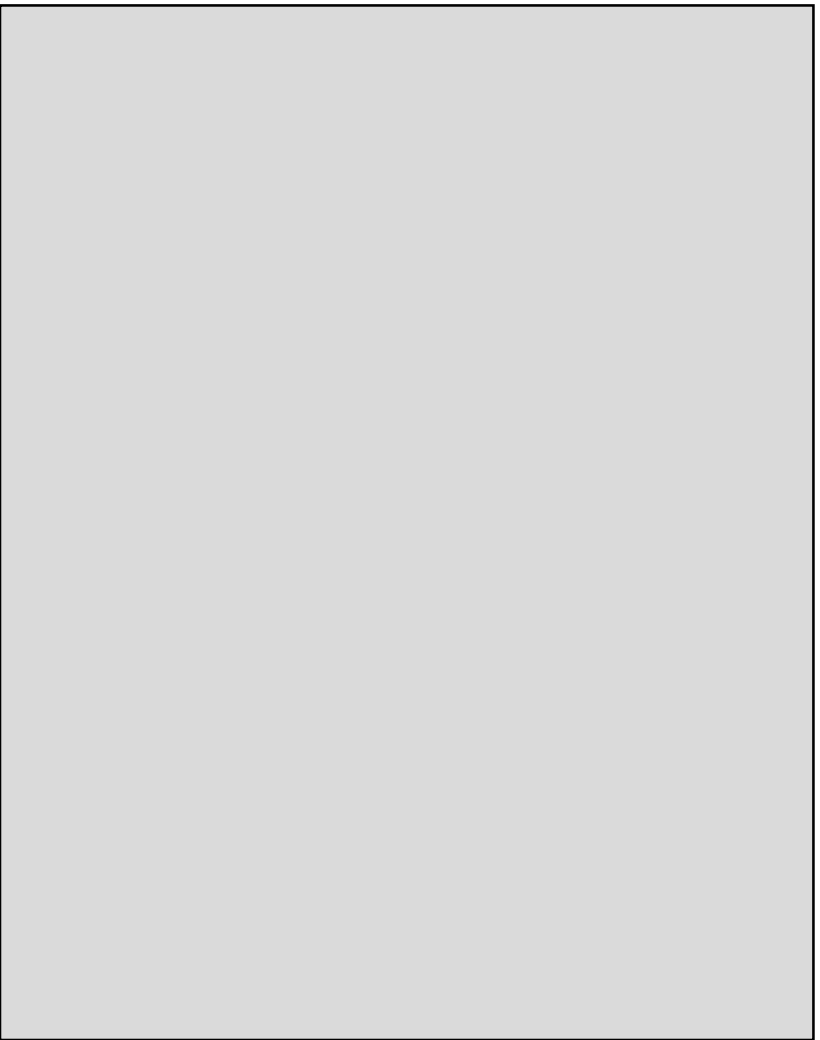

35
18
68
7
26
99
3
12
22
30

35

NULL

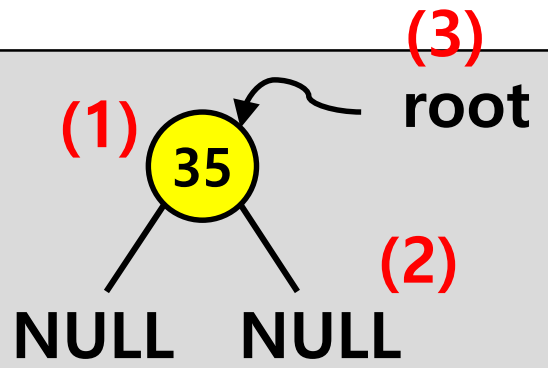
root



```
void insert_node(TreeNode **root, int
key) {
    TreeNode *p, *t;
    TreeNode *n;  // n은 새로운 노드
    t = *root;
    p = NULL;
    // 탐색을 먼저 수행
    while (t != NULL) {
        if( key == t->key ) return;
        p = t;
        if( key < t->key ) t = t->left;
        else t = t->right;
    }
```

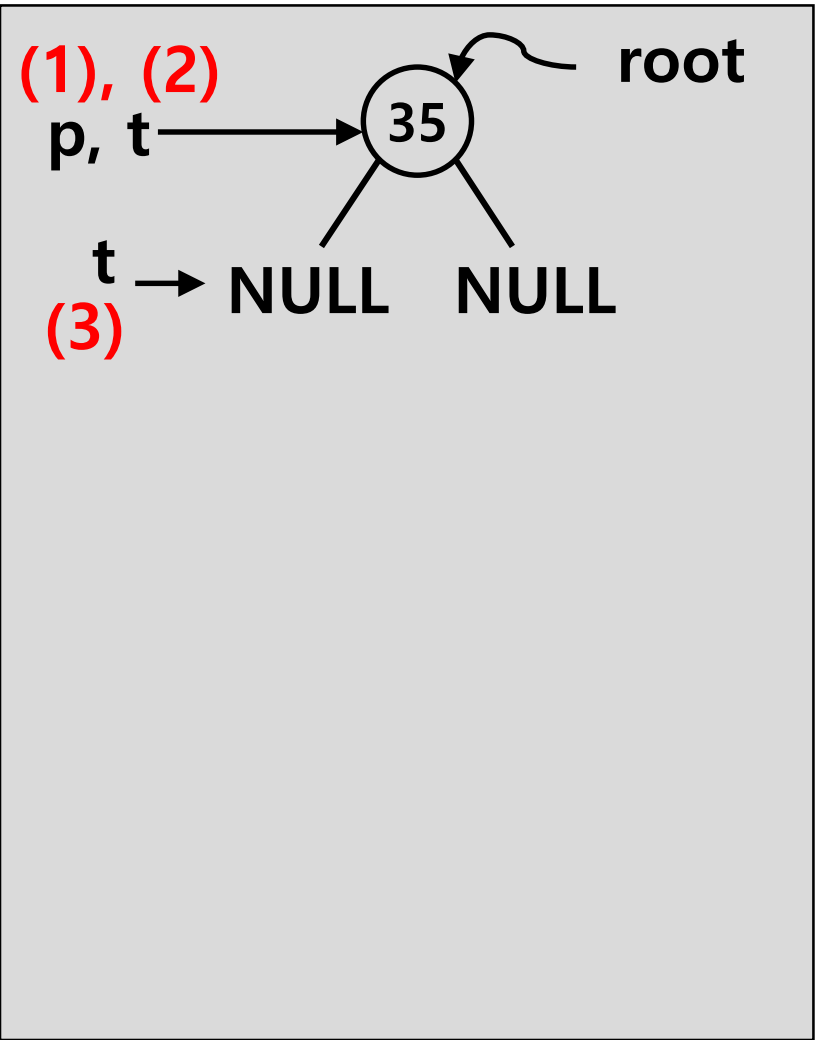
t == NULL 이므로 while문 pass

35




```
n = (TreeNode *)malloc(sizeof(TreeNode));
if( n == NULL ) return;
// 데이터 복사  ----------(1)
n->key = key;              ------(2)
n->left = n->right = NULL;
// 부모 노드와 링크 연결
if( p != NULL )
      if( key < p->key )
            p->left = n;
      else p->right = n;  ----------(3)
else *root = n;
}
```

p == NULL 이므로 else문 실행




```
void insert_node(TreeNode **root, int
key) {
    TreeNode *p, *t;
    TreeNode *n; // n은 새로운 노드
    t = *root;
    p = NULL;
    // 탐색을 먼저 수행
    while (t != NULL) {
        if( key == t->key ) return;
        p = t;
        if( key < t->key ) t = t->left;
        else t = t->right;
    }
```

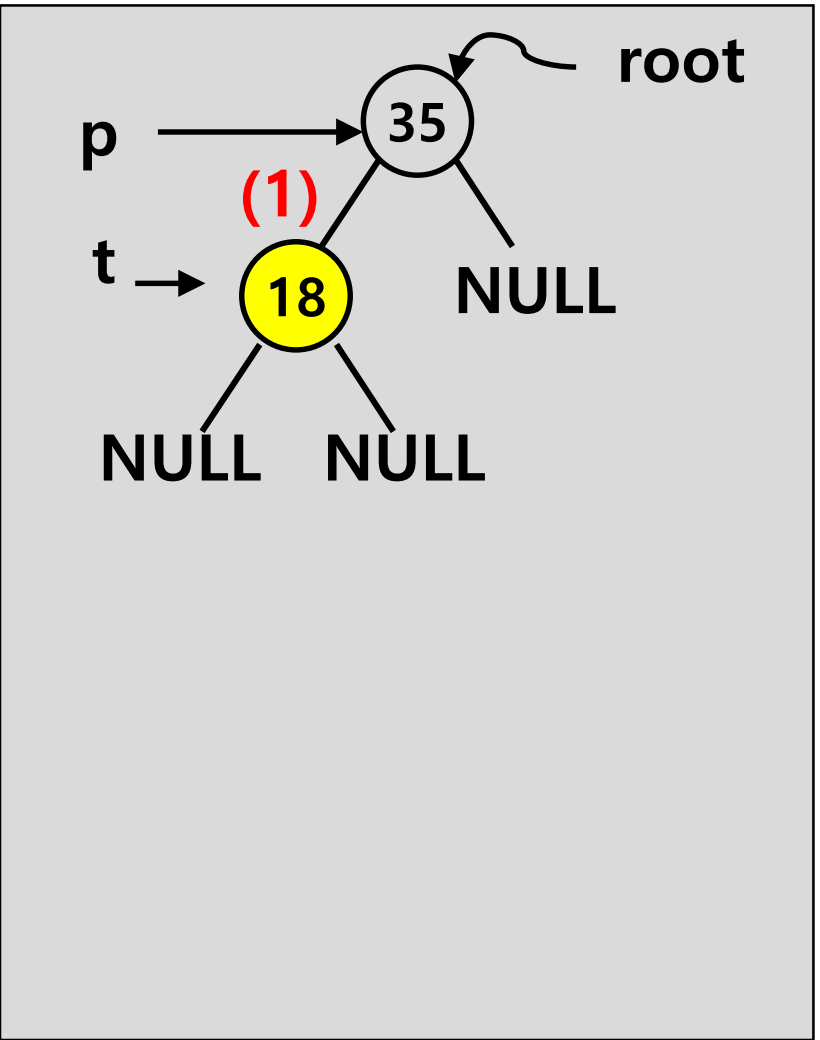
(1)

(2)

---- (3)

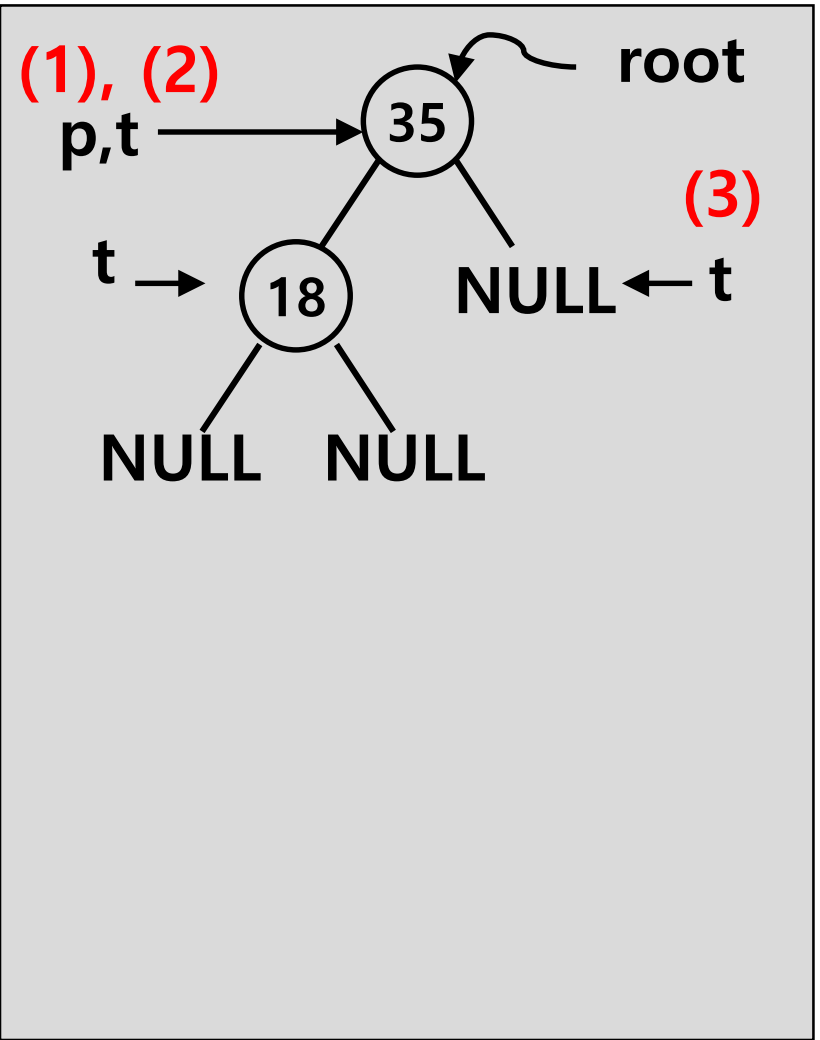
t == NULL 이 되므로 while문 종료

35 18




```
n = (TreeNode *)malloc(sizeof(TreeNode));
if( n == NULL ) return;
// 데이터 복사
n->key = key;
n->left = n->right = NULL;
// 부모 노드와 링크 연결
if( p != NULL )
      if( key < p->key )----------------(1)
            p->left = n;
      else p->right = n;
else *root = n;
}
```

35 18 68




```
void insert_node(TreeNode **root, int
key) {
    TreeNode *p, *t;
    TreeNode *n; // n은 새로운 노드
    t = *root;
    p = NULL;
    // 탐색을 먼저 수행
    while (t != NULL) {
        if( key == t->key ) return;
        p = t;
        if( key < t->key ) t = t->left;
        else t = t->right;
    }
```

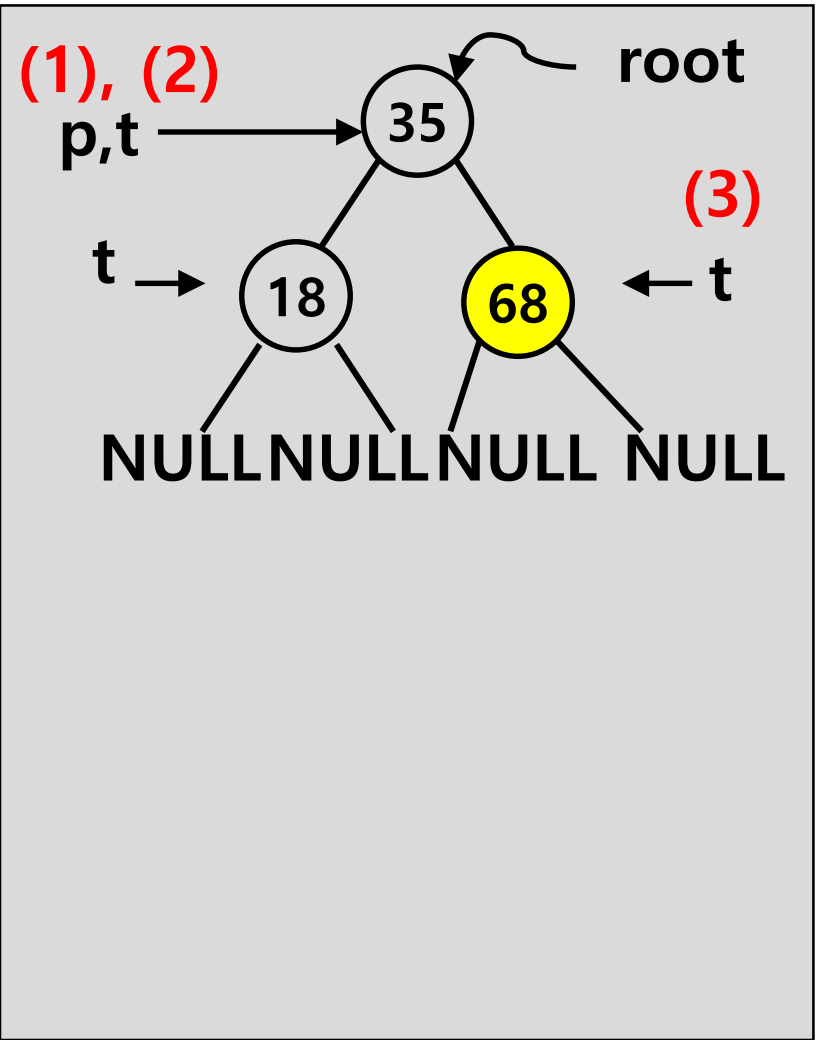
(1)

(2)

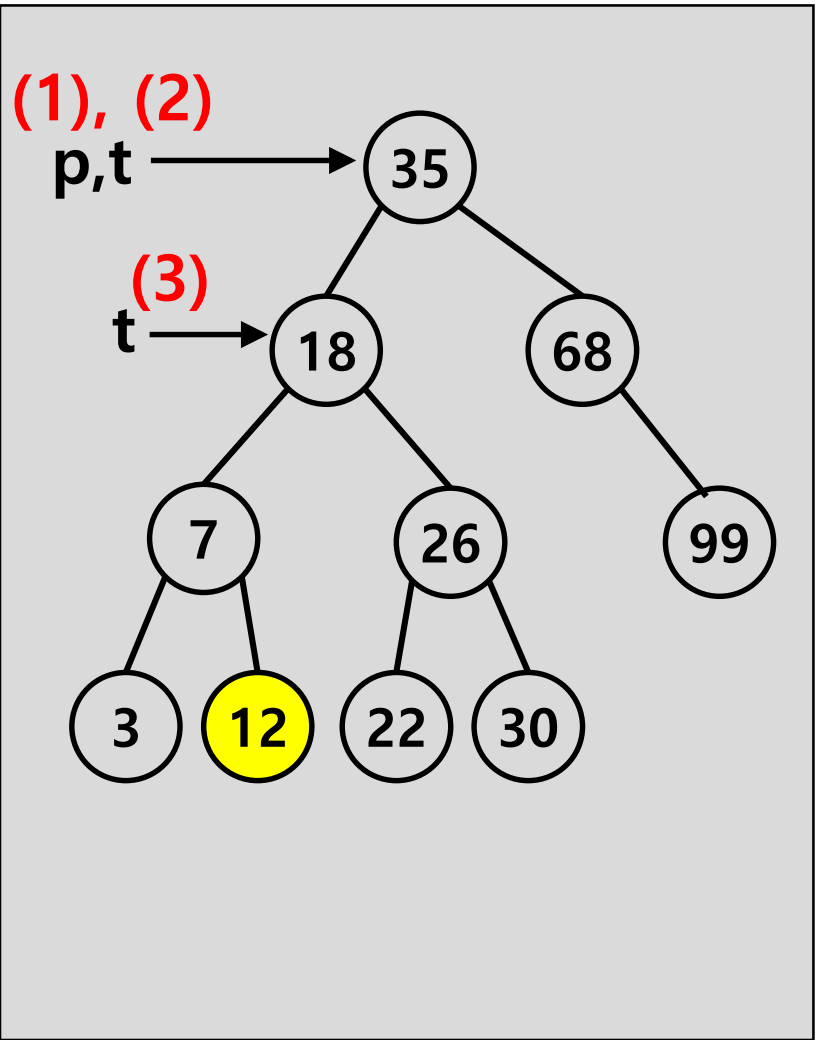
(3)

t == NULL 이 되므로 while문 종료

35 18 68




```
n = (TreeNode *)malloc(sizeof(TreeNode));
if( n == NULL ) return;
// 데이터 복사
n->key = key;
n->left = n->right = NULL;
// 부모 노드와 링크 연결
if( p != NULL )
      if( key < p->key )
            p->left = n;----------------(1)
      else p->right = n;
else *root = n;
}
```

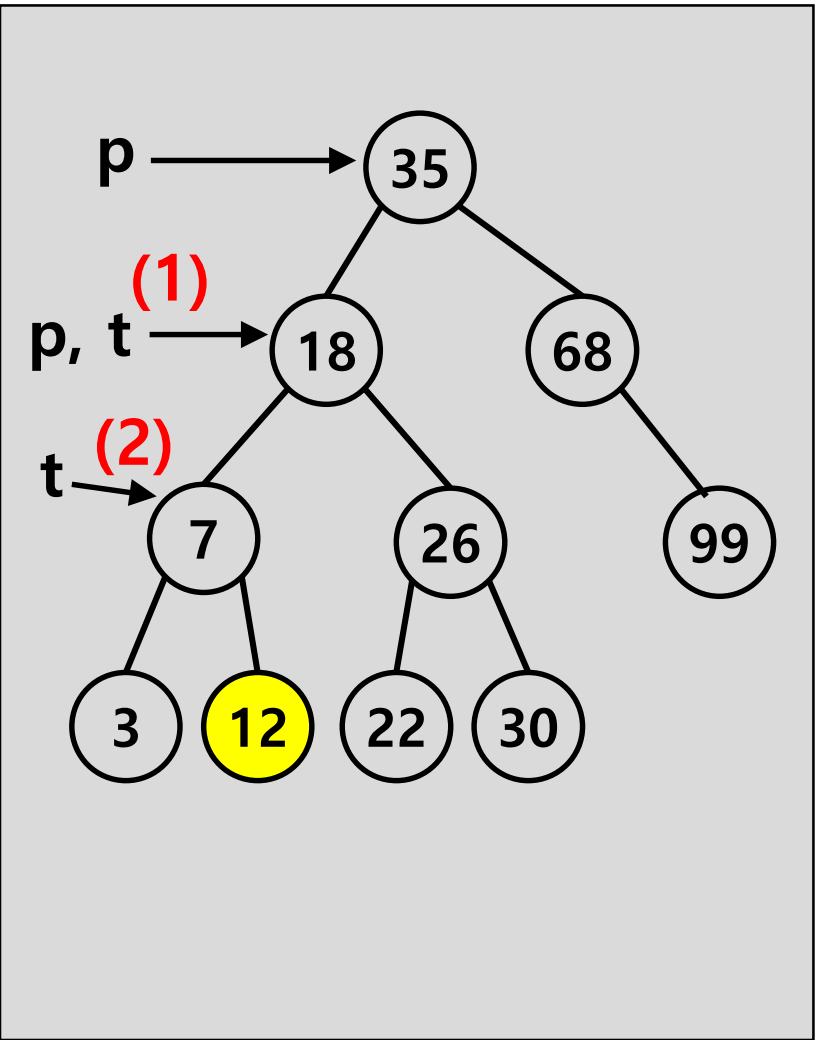



```
void  delete_node(TreeNode  **root,  int
key) {
   TreeNode *p, *child, *succ, *succ_p,
*t;
   p = NULL;          t = *root;
   while( t != NULL && t->key != key )
{
      p = t;
      t = ( key < t->key ) ? t->left :
t->right;
   }
   if( t == NULL ) {     // 탐색트리에 없
는 키
      printf("key is not in the tree");
      return;
}
```

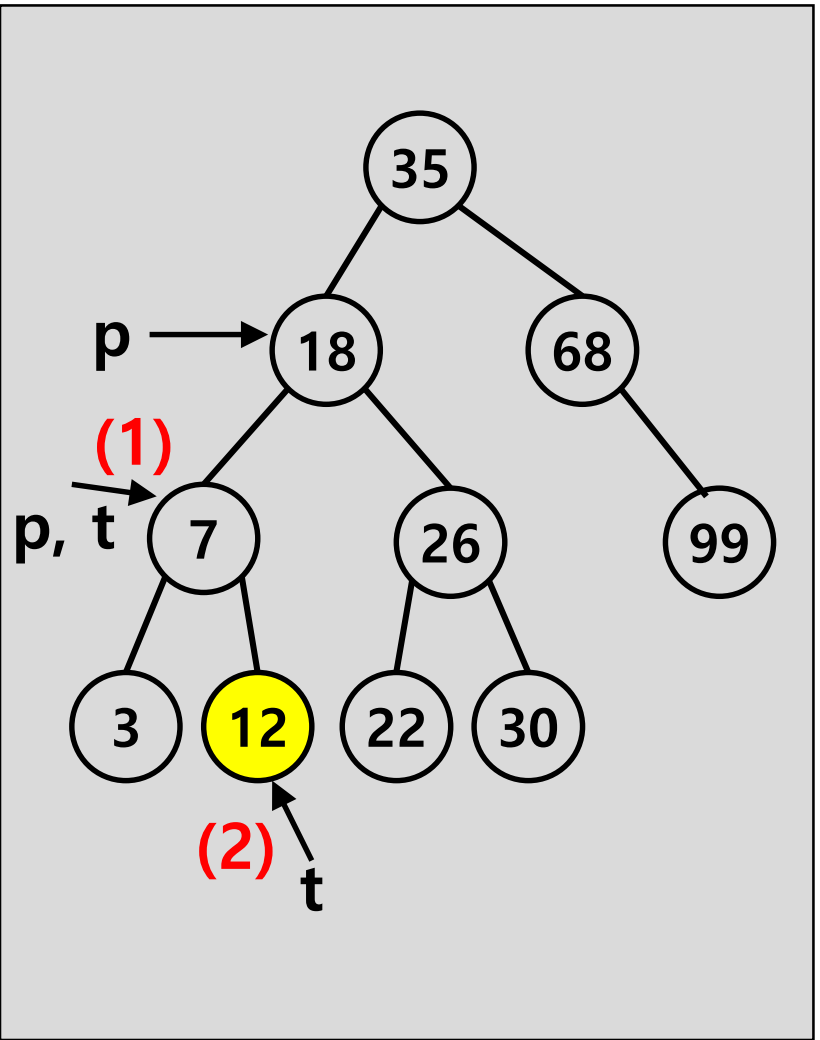
(1)

(2)

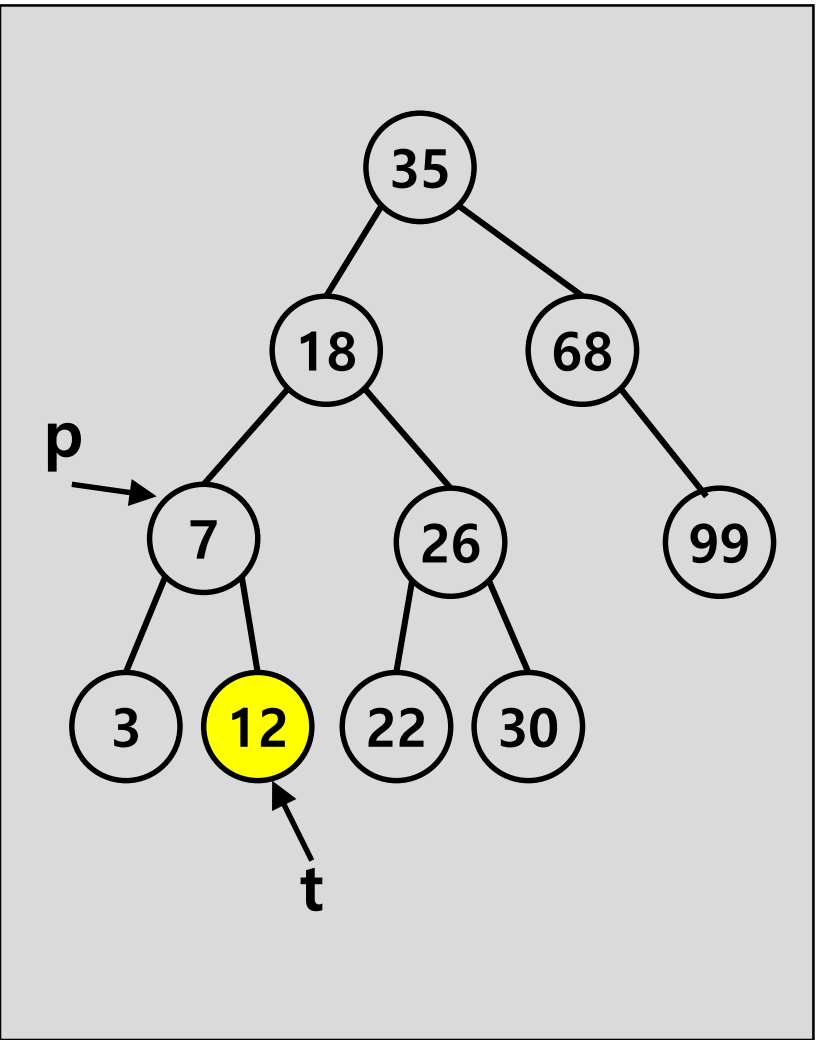
(3)




```
void  delete_node(TreeNode  **root,  int
key) {
   TreeNode *p, *child, *succ, *succ_p,
*t;
   p = NULL;  t = *root;          (1)
   while( t != NULL && t->key != key )
{                  ------------------(2)
      p = t;
      t = ( key < t->key ) ? t->left :
t->right;
   }
   if( t == NULL ) {     // 탐색트리에 없
는 키
      printf("key is not in the tree");
      return;
}
```

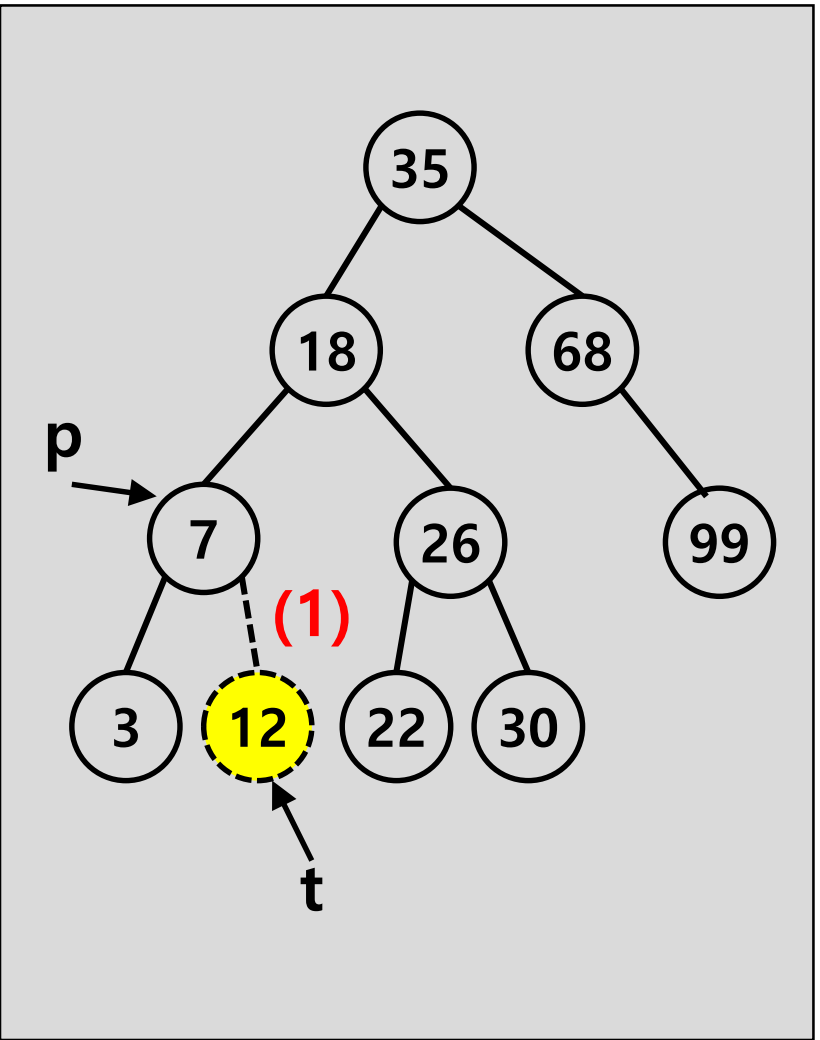



```
void  delete_node(TreeNode  **root,  int
key) {
   TreeNode *p, *child, *succ, *succ_p,
*t;
   p = NULL;  t = *root;          (1)
   while( t != NULL && t->key != key )
{                  ------------------(2)
      p = t;
      t = ( key < t->key ) ? t->left :
t->right;
   }
   if( t == NULL ) {     // 탐색트리에 없
는 키
      printf("key is not in the tree");
      return;
}
```

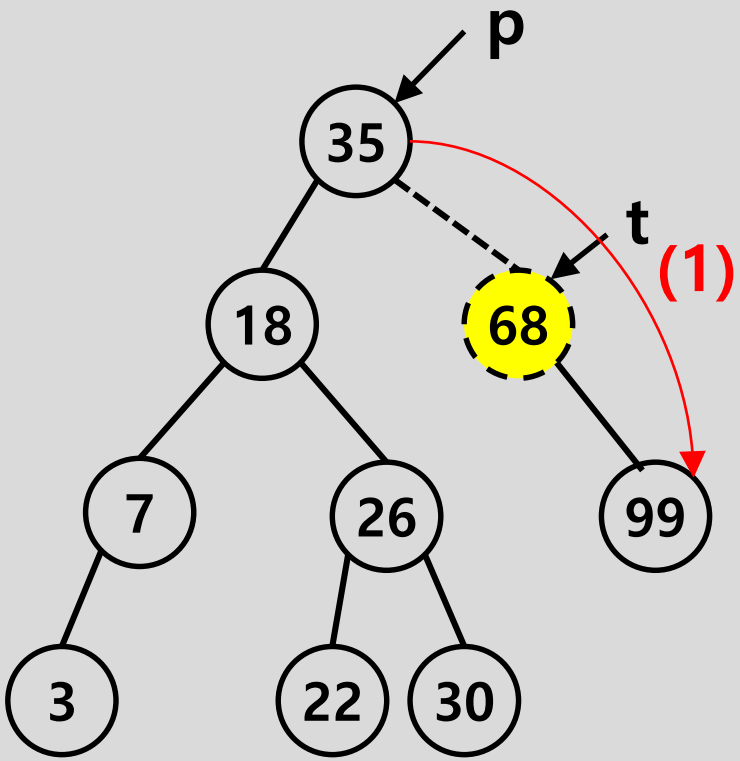



```
void  delete_node(TreeNode  **root,  int
key) {
   TreeNode *p, *child, *succ, *succ_p,
*t;
   p = NULL;   t = *root;
   while( t != NULL && t->key != key )
{
      p = t;
      t = ( key < t->key ) ? t->left :
t->right;
   }
   if( t == NULL ) {     // 탐색트리에 없
는 키
      printf("key is not in the tree");
      return;
}
```

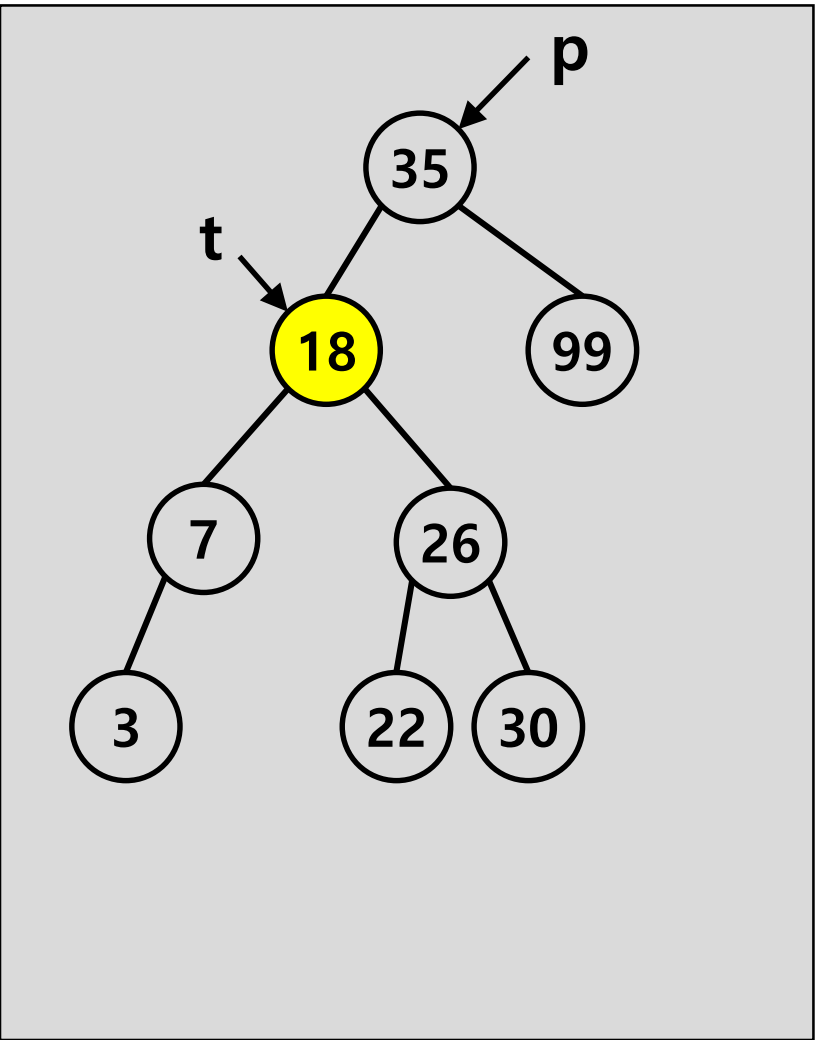
t->key == key이므로 while 종료




```
// 첫번째 경우: 단말노드인 경우
if(      (t->left==NULL)      &&      (t->right==NULL) ){
    if( p != NULL ){
        // 부모노드의 자식필드를 NULL
        if( p->left == t )
              p->left = NULL;---------------(1)
        else p->right = NULL;
    }
    else // 부모노드가 NULL이면 삭제 노드는 루트
              *root = NULL;
}
```

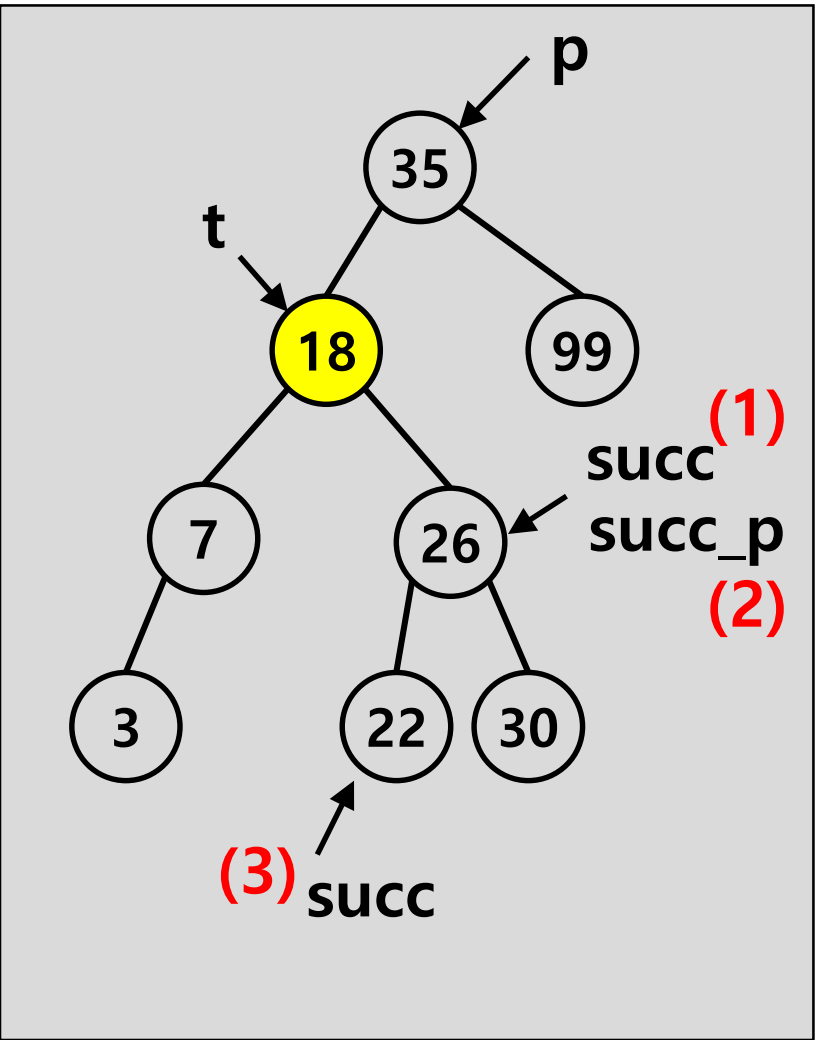



```
// 두번째 경우: 하나의 자식만 가지는 경우
else                    if((t->left==NULL)||(t-
>right==NULL)){
   child = (t->left != NULL) ? t->left :
t->right;
   if( p != NULL ){
     if( p->left == t ) ----// 부모를 자식과 (1)
연결
           p->left = child;
     else p->right = child;
   }
   else // 부모노드가 NULL이면 루트 삭제
     *root = child;
```




```
// 세번째 경우: 두개의 자식을 가지는 경우
else {
   succ_p = t; //오른쪽 서브트리에서 후계자 탐색
   succ = t->right;
   while(succ->left != NULL){// 왼쪽으로 이동
      succ_p = succ;
      succ = succ->left;      }
   // 후속자의 부모와 자식을 연결
   if( succ_p->left == succ )
      succ_p->left = succ->right;
   else
      succ_p->right = succ->right;
   t->key = succ->key; // 후속자 키 값 복사
```

(1)
(1)

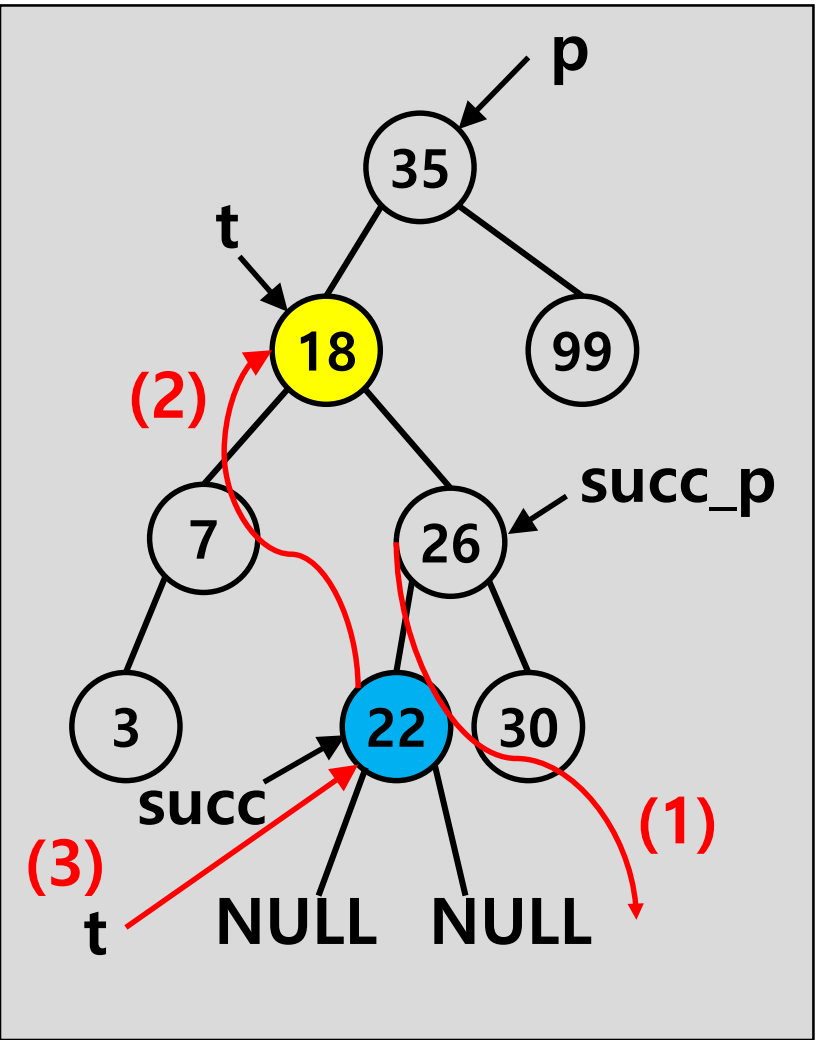



```
// 세번째 경우: 두개의 자식을 가지는 경우
else {
   succ_p = t; //오른쪽 서브트리에서 후계자 탐색
   succ = t->right;
   while(succ->left != NULL){ // 왼쪽으로 이동
      succ_p = succ;
      succ = succ->left;      }
   // 후속자의 부모와 자식을 연결
   if( succ_p->left == succ )
      succ_p->left = succ->right;
   else
      succ_p->right = succ->right;
   t->key = succ->key; // 후속자 키 값 복사
```

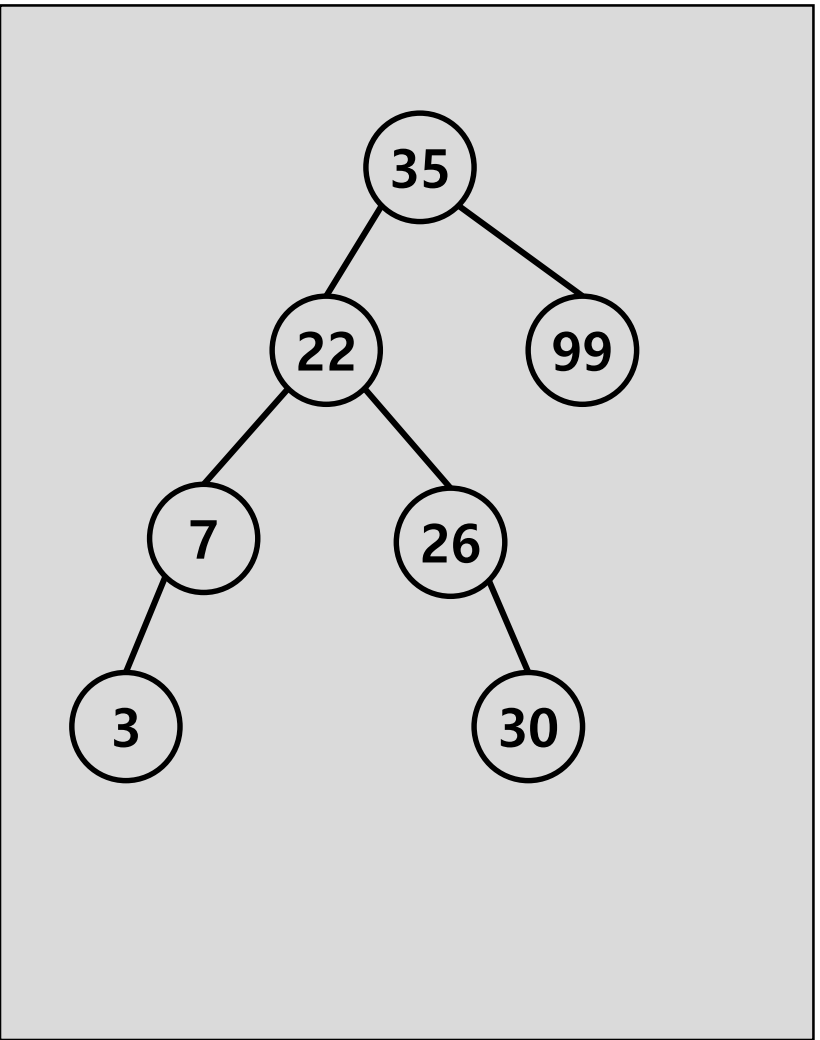
(1)

(2)

(3)




```
// 세번째 경우: 두개의 자식을 가지는 경우
else {
   succ_p = t; //오른쪽 서브트리에서 후계자 탐색
   succ = t->right;
   while(succ->left != NULL){// 왼쪽으로 이동
      succ_p = succ;
      succ = succ->left;      }
   // 후속자의 부모와 자식을 연결----------(1)
   if( succ_p->left == succ )
      succ_p->left = succ->right;
   else                       ----------(2)
      succ_p->right = succ->right;
   t->key = succ->key;                 ----------(3)
   t = succ;
```


35
22
99
7
26
3
30