데이터 구조

강의 노트 5 트

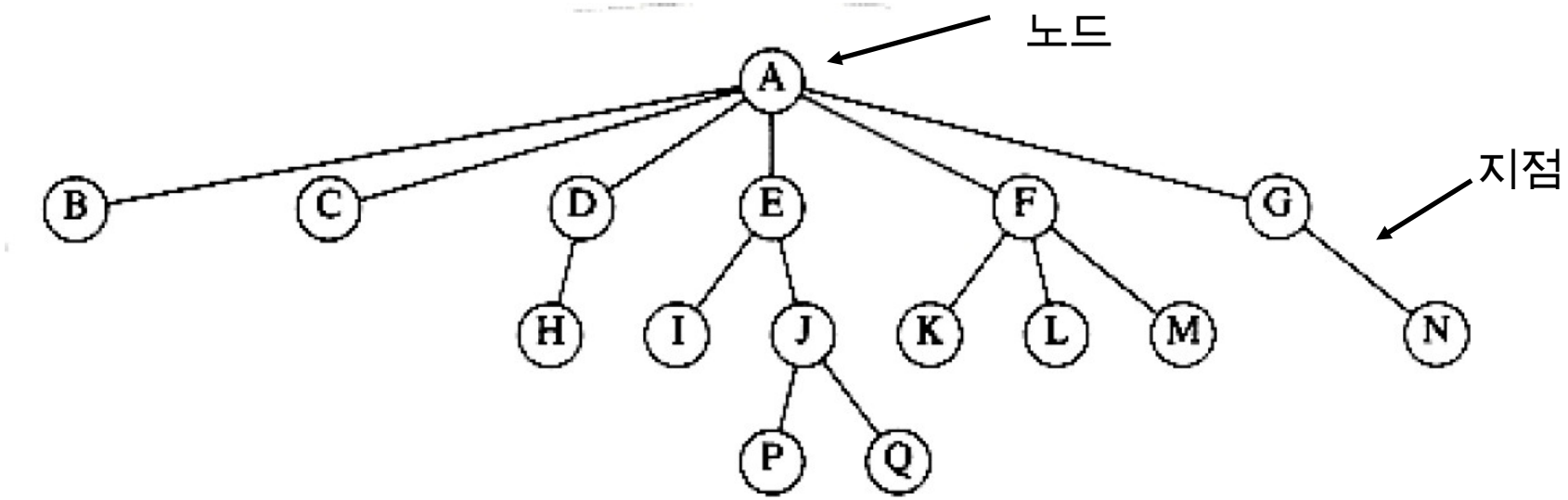
리

김유증, 박사 조교수
컴퓨터 과학 및 정보 공학부 가톨릭 대학교, 대한민국

트

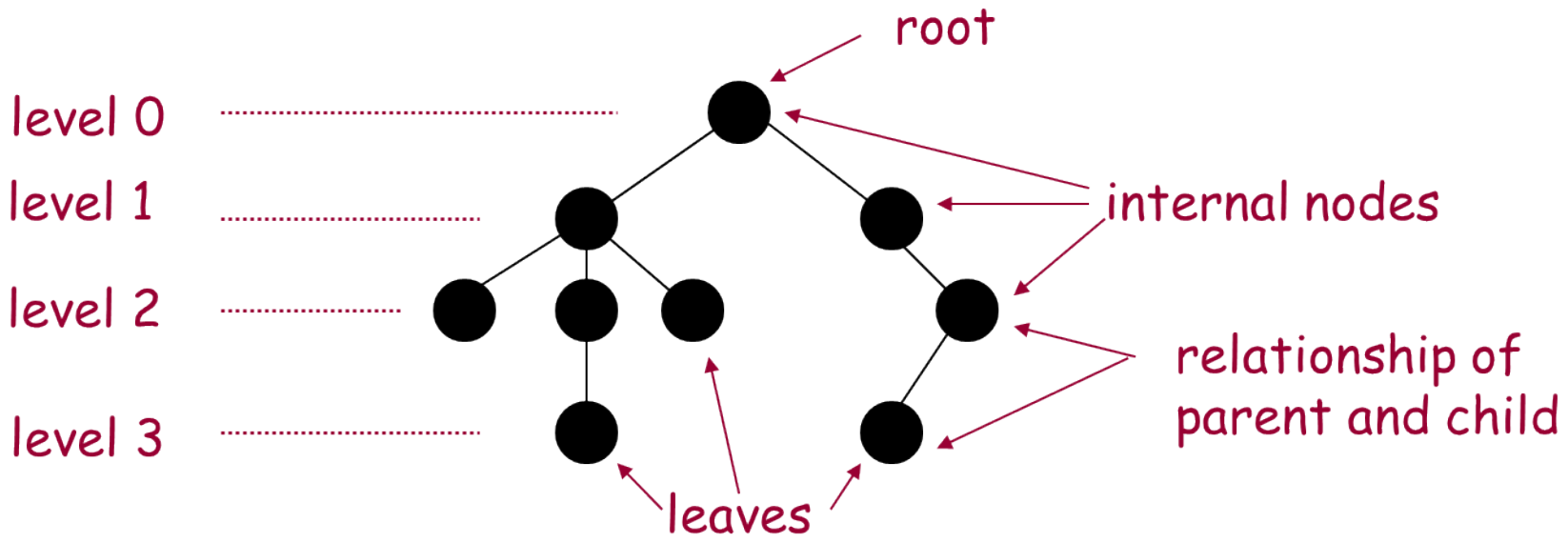
- Array: O(1) 액세스 시간, 확장 가능하지 않음
- 연결된 목록: 확장 가능하지만 액세스 시간이 O(N)입니다.
- 대량의 데이터의 경우 확장성과 짧은 액세스 시간을 원합니다.

- **트리** : 확장 가능하고 O(logN) 액세스 시간
  - 일반적으로 노드와 브랜치(또는 에지)가 있는 거꾸로 된 트리로 표시됩니다.



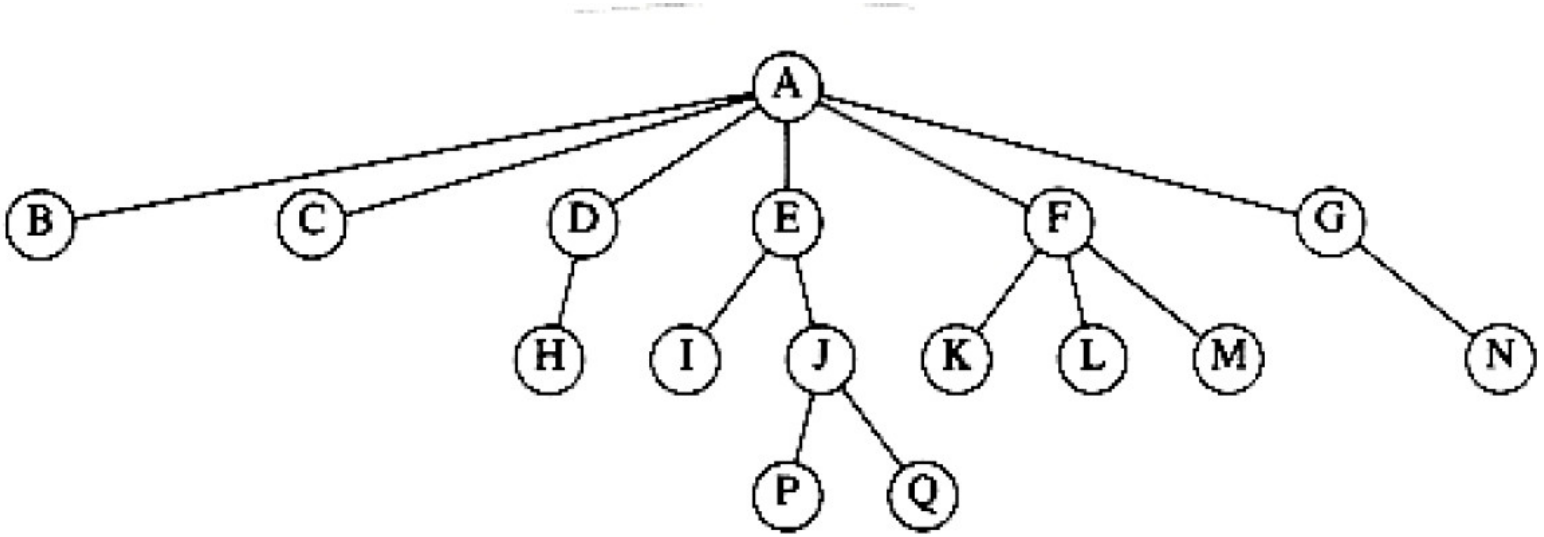
트

- 노드에는 "하나의" 부모 노드와 자식 노드가 있습니다.
  - 루트 노드: 부모 노드가 없는 노드
  - 리프 노드(터미널 노드): 자식 노드가 없는 노드
- 각 노드에는 고유한 깊이가 있으며 조상으로 계산할 수 있습니다.
  - 트리 깊이(높이)는 종종 노드의 최대 깊이를 의미합니다.
- 노드 수는 종종 트리 크기로 간주됩니다.
- 한 노드에서 다른 노드로 가는 경로는 하나뿐입니다.
- 하위 트리: 주어진 트리의 일부로 하위 집합이 있는 트리입니다.



트

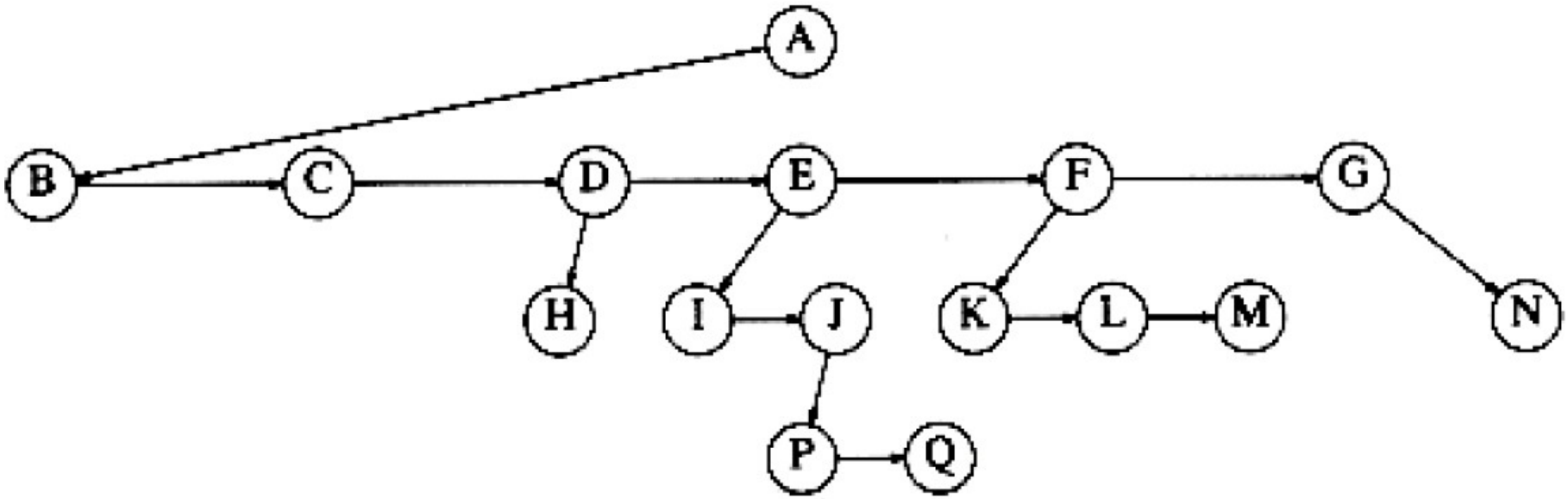
- 루트 노드란 무엇인가요?
- 트리 크기는 어떻게 되나요?
- 최대 수심은 얼마인가요?
- J의 깊이는 어떻게 되나요?



트리 구현

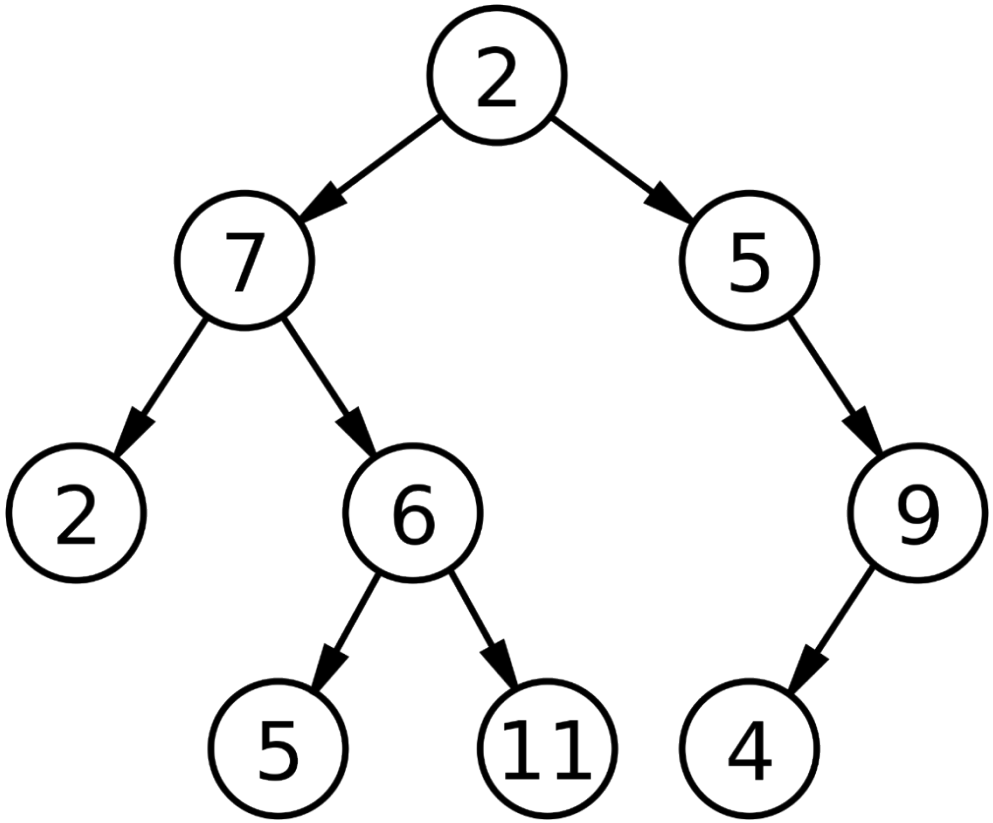
- 트리는 종종 다중 목록으로 구현됩니다.

```
1  typedef struct TreeNode *PtrToNode;
2
3  struct TreeNode
4  {
5      ElementType Element;
6      PtrToNode   FirstChild;
7      PtrToNode   NextSibling;
8  };
```



이진 트리

- 이진 트리는 최대 2개의 자식 노드가 있는 트리입니다.
  - N 입력의 경우 최악의 깊이는 N-1이지만 일반적으로 평균 깊이는 다음과 같습니다. □(√□) 또는(로그).



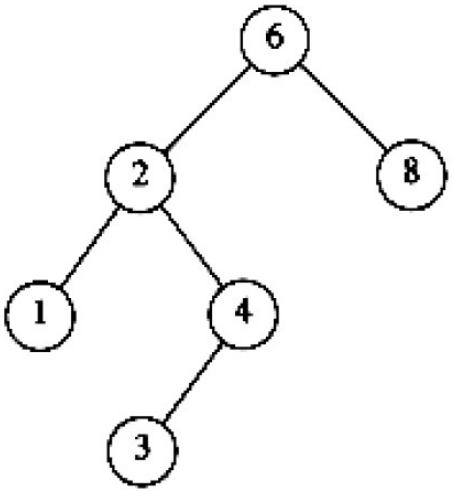
이진 트리

- 이진 트리는 최대 2개의 자식 노드가 있는 트리입니다.
  - N 입력의 경우 최악의 깊이는 N-1이지만 일반적으로 평균 깊이는 다음과 같습니다. □(√□) 또는(로그).

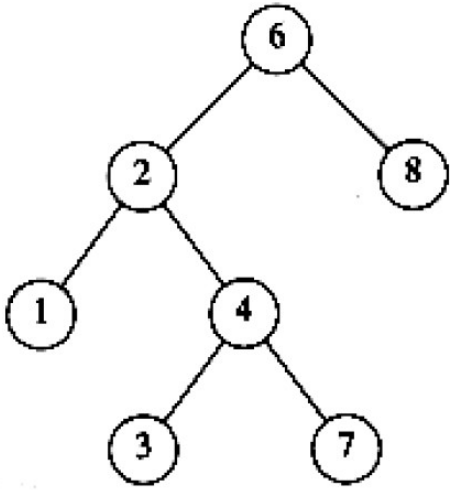
```
1  typedef struct TreeNode *PtrToNode;
2  typedef struct PtrToNode Tree;
3
4  struct TreeNode
5  {
6      ElementType Element;
7      Tree        Left;
8      Tree        Right;
9  };
```

이진 검색 트리

- 바이너리 트리로 검색을 효율적으로 수행할 수 있습니다.
  - 키에는 순서(예: 숫자, 문자열)가 있습니다.
  - 각 트리 노드에는 키 값이 있고, 주어진 키 값이 2진수인 노드가 있습니다.
    트리는 □ (로그).
  - 목록을 검색하려면 O (□).
  - 키는 중복될 수 있지만(예: 성) 여기서는 고유 키를 가정합니다.
- 이진 검색 트리
  - (왼쪽 자식의 키) < (주어진 노드의 키)
  - (오른쪽 자식의 키) > (주어진 노드의 키)



유효한 이진 검색 트리



잘못된 이진 검색 트리

이진 검색 트리: 구현

- 요소는 키라고 가정하지만 키와 값을 분리할 수 있습니다.

```
typedef TreeNode;
typedef struct TreeNode *Position;
typedef struct TreeNode *SearchTree;

SearchTree MakeEMpty(SearchTree T);
Position Find(ElementType X,SearchTree T);
Position FindMin(SearchTree X);
Position FindMax(SearchTree X);
SearchTree Insert(ElementType X,SearchTree T);
SearchTree Delete(ElementType X,SearchTree T);

struct TreeNode
{
    ElementType Element;
    SearchTree  Left;
    SearchTree  Right;
};
```

이진 검색 트리: 구현

- 이 구현에서는 더미 노드가 없습니다.
  - 엔티티가 없으면 노드가 없습니다.
  - 더미 헤드 노드가 있는 링크드 리스트 구현에는 엔티티가 0인 경우에도 더미 노드가 있습니다.
  - 이 경우 루트 노드를 업데이트해야 합니다(루트 삭제 고려).

- `MakeEmpty()`
  - 나무를 비우고 비워두기

```
20  SearchTree MakeEmpty(SearchTree T)
21  {
22      if( T != NULL ) {
23          MakeEmpty(T->Left);
24          MakeEmpty(T->Right);
25          free(T);
26      }
27      return NULL;
28  }
```

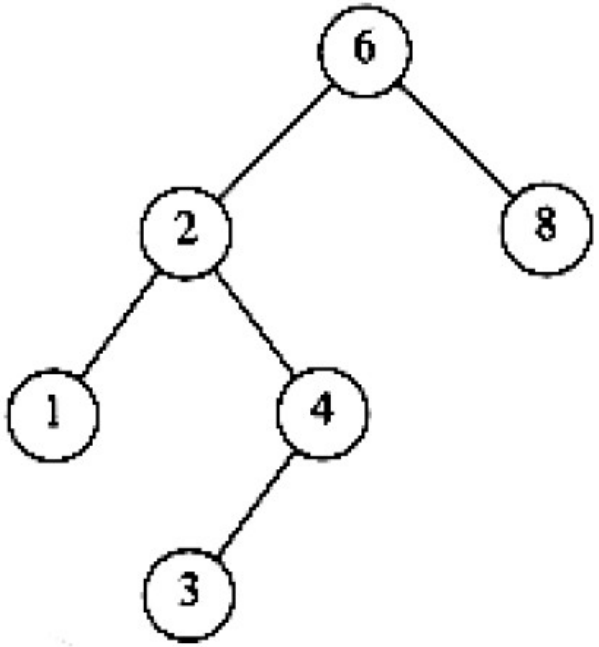
이진 검색 트리: 구현

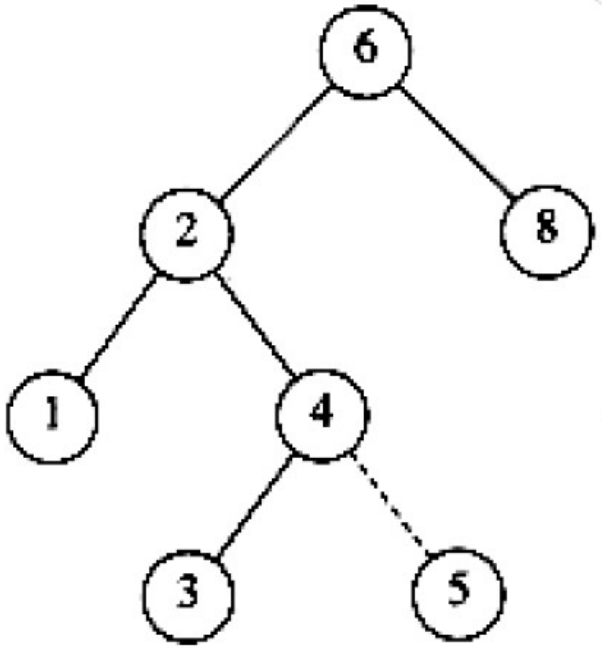
- 찾기(), 찾기 최소(), 찾기 최대()
  - 주어진 키를 가진 노드, 최소/최대 키를 가진 노드 찾기
  - 이는 재귀적으로 구현되지만 재귀 없이도 가능하고 효율적입니다.

```
30  Position Find(ElementType X,SearchTree T)
31  {
32      if( T == NULL ) return NULL;
33      else if( X < T->Element ) return Find(X,T->Left);
34      else if( X > T->Element ) return Find(X,T->Right);
35      else return T;
36  }
37
38  Position FindMin(SearchTree T)
39  {
40      if( T == NULL ) return NULL;
41      else if( T->Left == NULL ) return T;
42      else return FindMin(T->Left);
43  }
44
45  Position FindMax(SearchTree T)
46  {
47      if( T == NULL ) return NULL;
48      else if( T->Right == NULL ) return T;
49      else return FindMax(T->Right);
50  }
```

이진 검색 트리: 구현

- 삽입 ( )
  - 키가 있는 노드를 삽입하고, 주어진 키를 가진 노드가 있으면 업데이트만 하면 됩니다.
  - 삽입된 노드를 반환하여 새 트리를 생성하거나 기존 트리 내에 할당해야 합니다.





이진 검색 트리: 구현

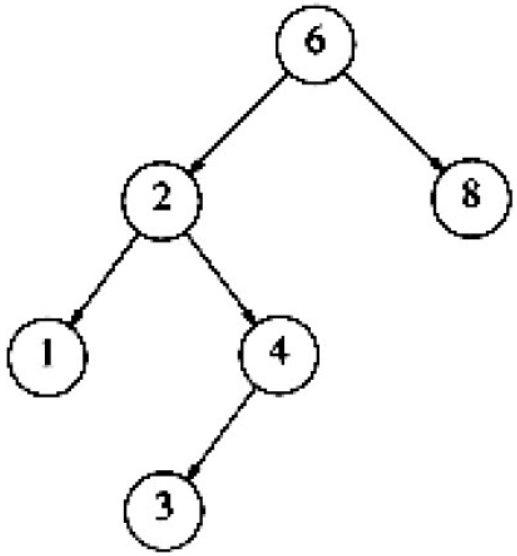
- 삽입 ( )
  - 키가 있는 노드를 삽입하고, 주어진 키를 가진 노드가 있으면 업데이트만 하면 됩니다.
  - 삽입된 노드를 반환하여 새 트리를 생성하거나 기존 트리 내에 할당해야 합니다.

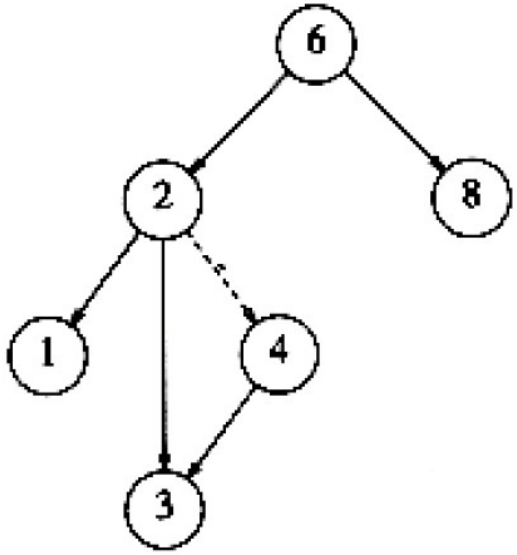
```
52  SearchTree Insert(ElementType X,SearchTree T)
53  {
54      if( T == NULL ) { // creating a new node
55          T = (SearchTree)malloc(sizeof(struct TreeNode));
56          if( T == NULL ) FetalError("Out of Memory");
57          else {
58              T->Element = X;
59              T->Left = T->Right = NULL;
60          }
61      } else if( X < T->Element ) {
62          T->Left = Insert(X,T->Left);
63      } else if( X > T->Element ) {
64          T->Right = Insert(X,T->Right);
65      }
66      return T;
67  }
```

이진 검색 트리: 구현

- `Delete()`
  - 주어진 키를 가진 노드를 찾아 삭제하는 경우, 네 가지 경우가 있습니다.
  - (1) 찾을 수 없음: 아무것도 하지 않음
  - (2) 자식이 없는 노드: 삭제하기만 하면 됩니다.
  - (3) 자식이 하나 있는 노드: 자식을 부모에 연결한 후 삭제
  - (4) 두 개의 자식이 있는 노드: 오른쪽 트리에서 최소 키를 가진 노드를 찾고, 찾은 노드로 대상 노드를 업데이트한 후, 찾은 노드를 삭제합니다.

- 예: 4 삭제

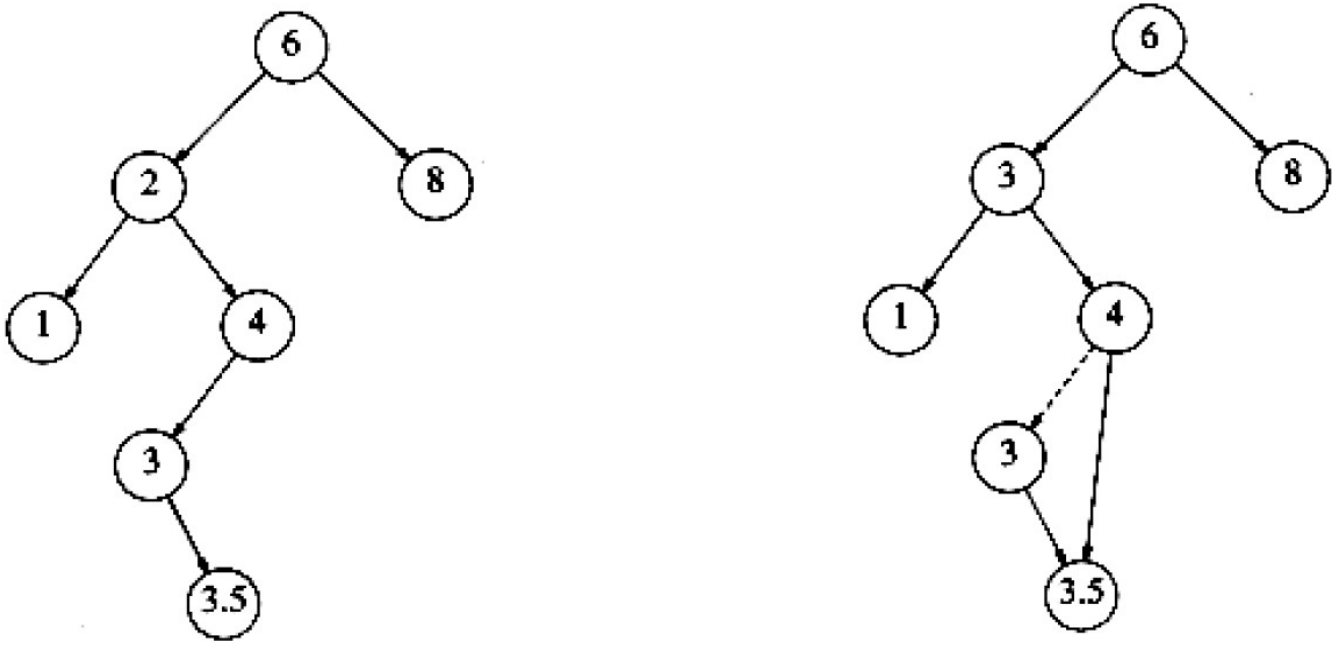




이진 검색 트리: 구현

- `Delete()`
  - 주어진 키를 가진 노드를 찾아 삭제하는 경우, 네 가지 경우가 있습니다.
  - (1) 찾을 수 없음: 아무것도 하지 않음
  - (2) 자식이 없는 노드: 삭제하기만 하면 됩니다.
  - (3) 자식이 하나 있는 노드: 자식을 부모에 연결한 후 삭제
  - (4) 두 개의 자식이 있는 노드: 오른쪽 트리에서 최소 키를 가진 노드를 찾고, 찾은 노드로 대상 노드를 업데이트한 후, 찾은 노드를 삭제합니다.

- 예: 2 삭제



이진 검색 트리: 구현

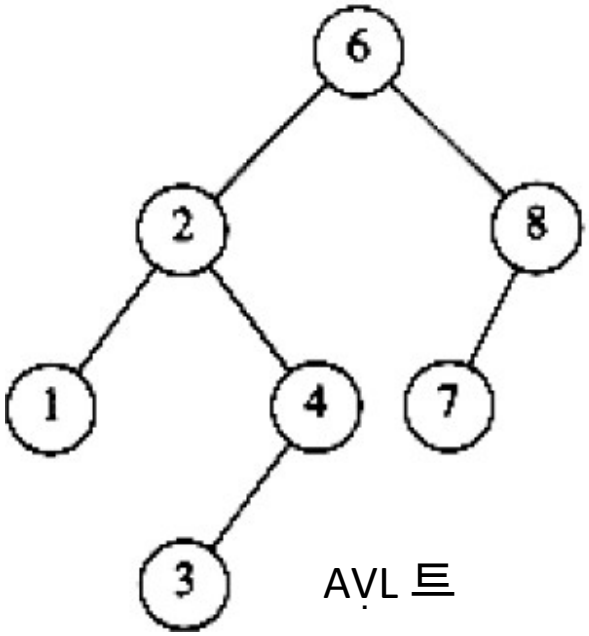
- Delete()

```
69  SearchTree Delete(ElementType X,SearchTree T)
70  {
71      Position Tmp;
72      if( T == NULL ) { // do nothing
73      } else if( X < T->Element ) {
74          T->Left = Delete(X,T->Left);
75      } else if( X > T->Element ) {
76          T->Right = Delete(X,T->Right);
77      } else if( T->Left && T->Right ) { // two children
78          Tmp = FindMin(T->Right);
79          T->Element = Tmp->Element;
80          T->Right = Delete(T->Element,T->Right);
81      } else { // zero or one child
82          Tmp = T;
83          if( T->Left == NULL ) T = T->Right;
84          else if( T->Right == NULL ) T = T->Left;
85          free(Tmp);
86      }
87      return T;
88  }
```

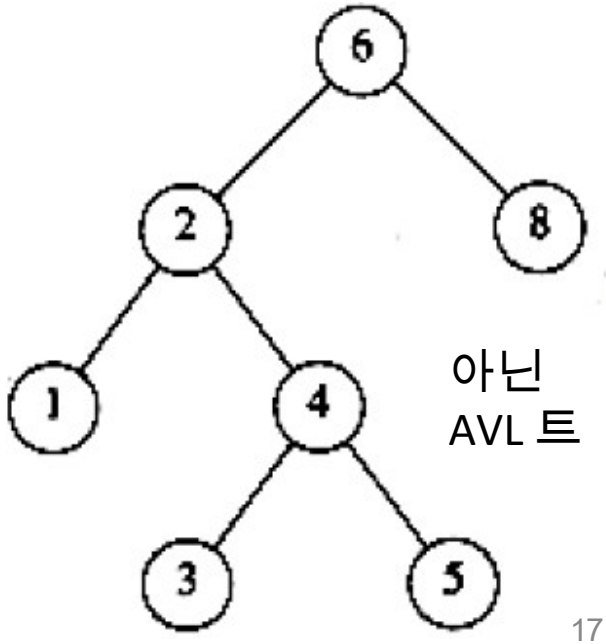
AVL 트

- AVL(Adelson -Velskii and Landis) 트리는 균형 잡힌 조건의 이진 검색 트리입니다.
  - 트리의 모든 노드에 대해 왼쪽 및 오른쪽 하위 트리의 높이는 다음과 같이 다를 수 있습니다.
    최대 1.
  - 지연 삭제가 가정됩니다. 삽입을 제외한 모든 작업은 다음에서 수행할 수 있습니다. □(log □). 실제로 삽입에도 □(log □)가 필요합니다.
  - AVL 트리의 모든 연산은 O(로그 N)가 걸리는 반면, BST는 O (□).
  - 삽입하려면 트리의 밸런스를 재조정해야 하는데, 이 작업은 $O(\log N)$로 수행할 수 있습니다.

AVL 트



AVL 트

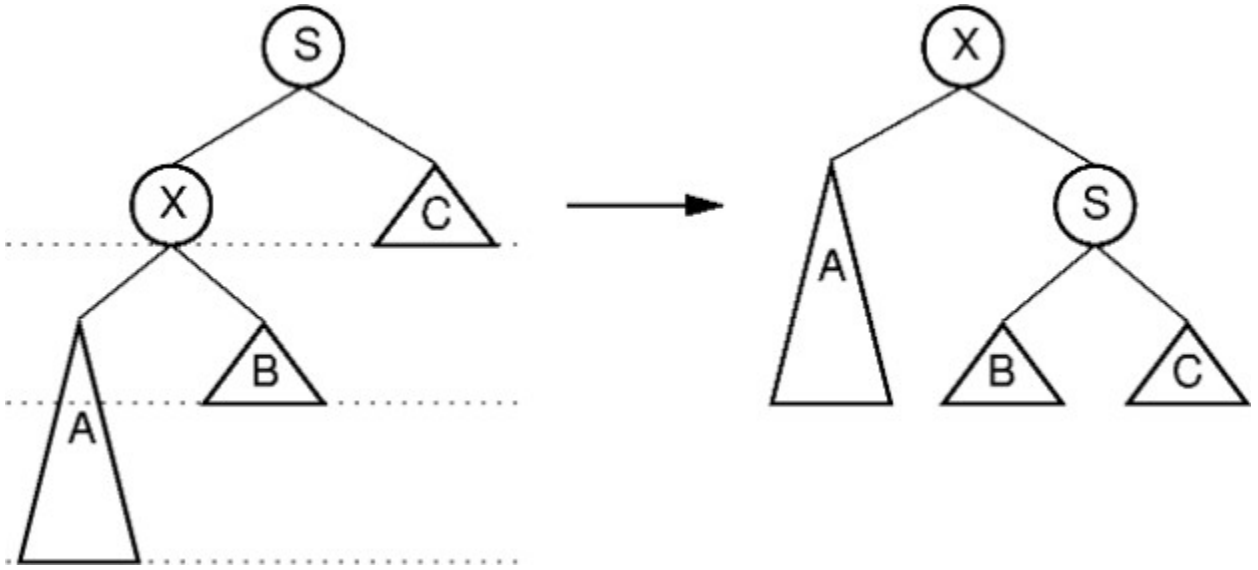


아닌
AVL 트

AVL 트리: 삽입

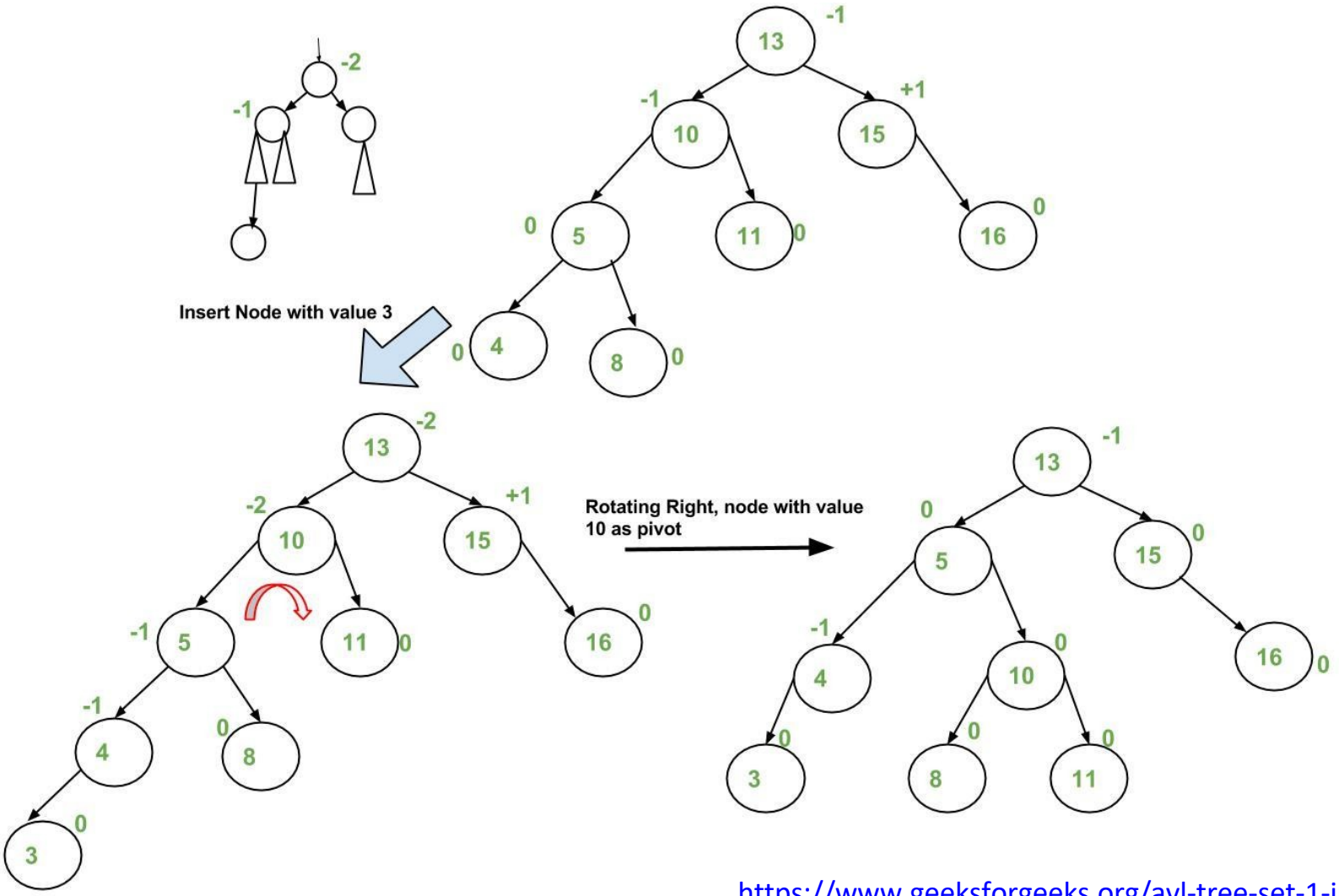
- 삽입 절차
  - 이진 검색 트리처럼 삽입하기
  - AVL 조건을 위반한 경우 회전 기술을 사용하여 트리의 밸런스를 재조정합니다.

- 단일 회전
  - 포인터를 세 번만 움직이면 됩니다.



AVL 트리: 삽입

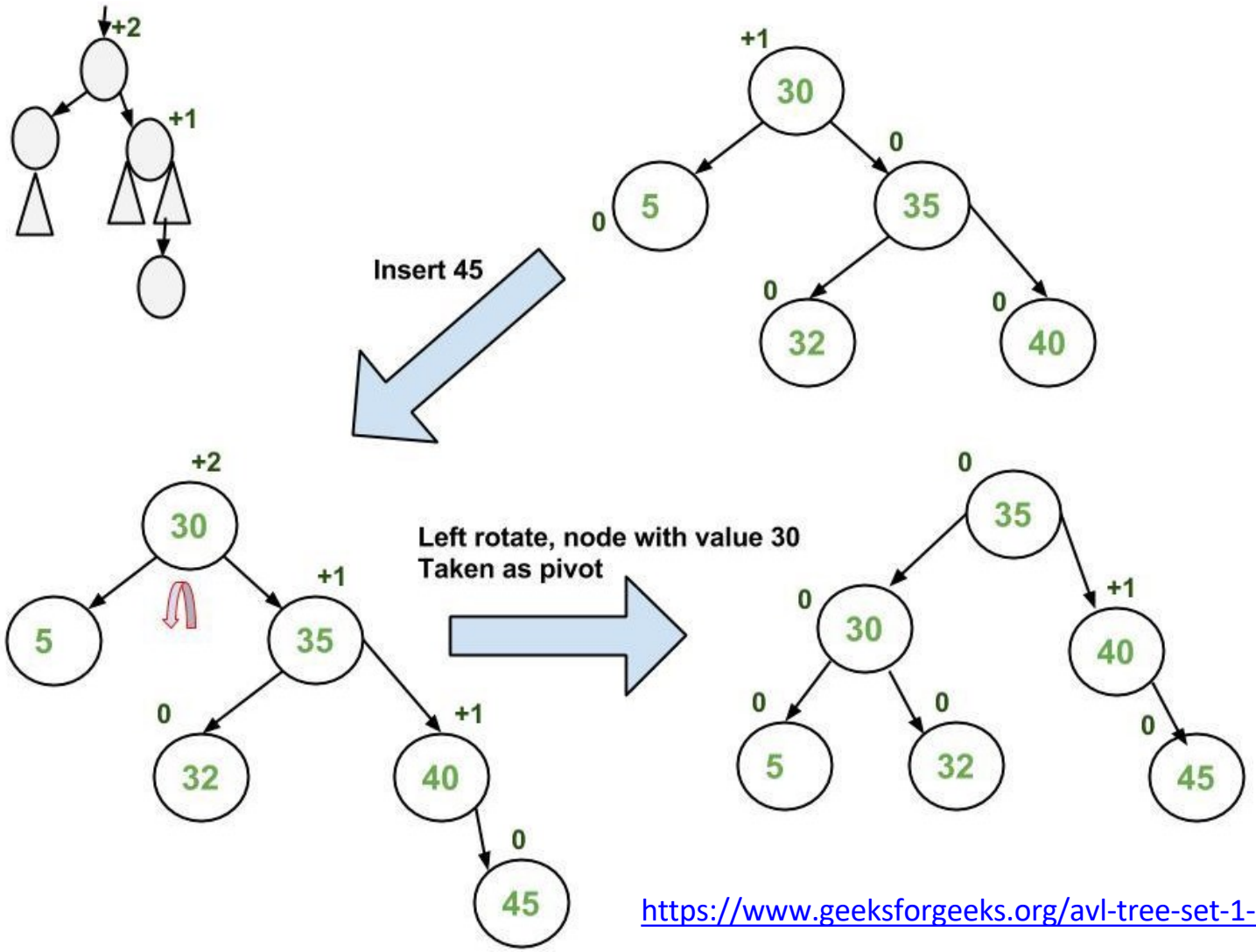
- 단일 회전 예시



https://www.geeksforgeeks.org/avl-tree-set-1-insertion/

AVL 트리: 삽입

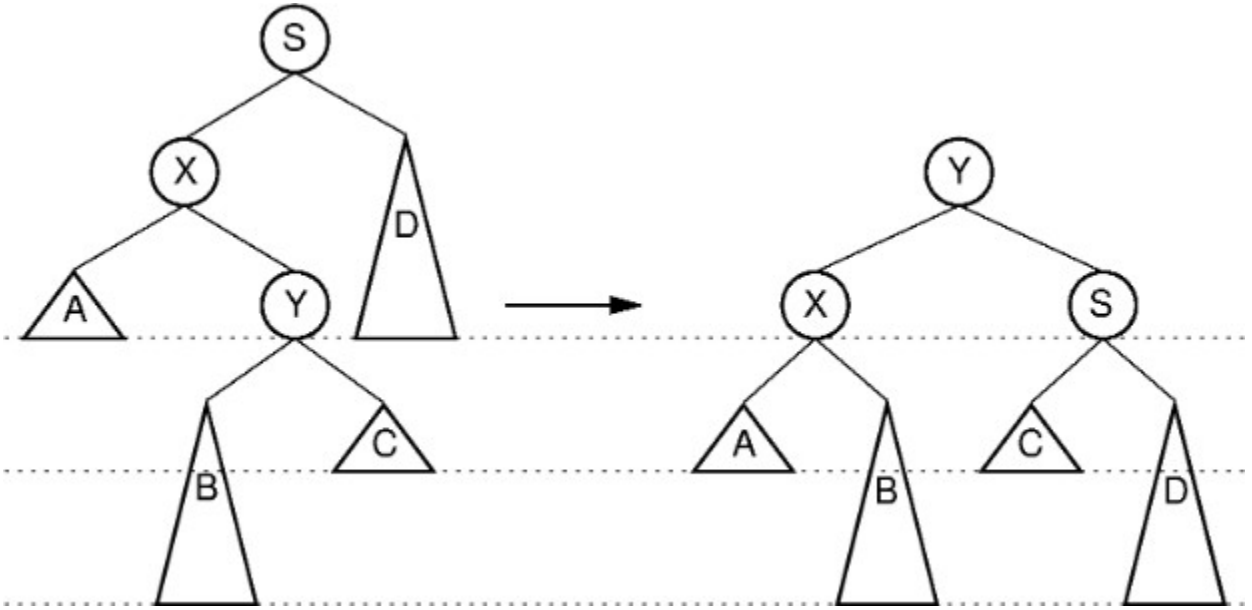
- 단일 회전 예시



https://www.geeksforgeeks.org/avl-tree-set-1-insertion/

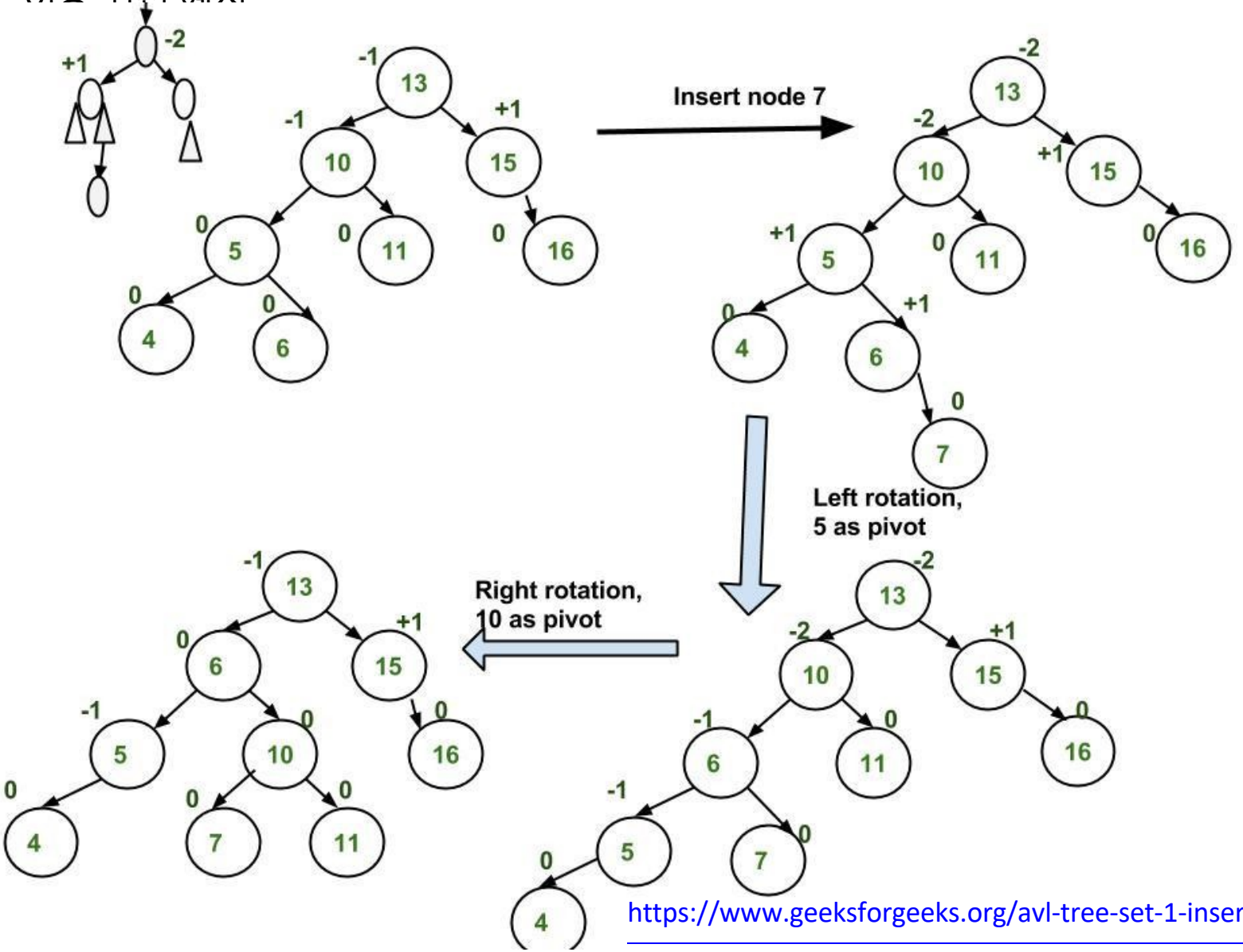
AVL 트리: 삽입

- 이중 회전
  - 두 번의 단일 회전으로 구현 가능



AVL 트리: 삽입

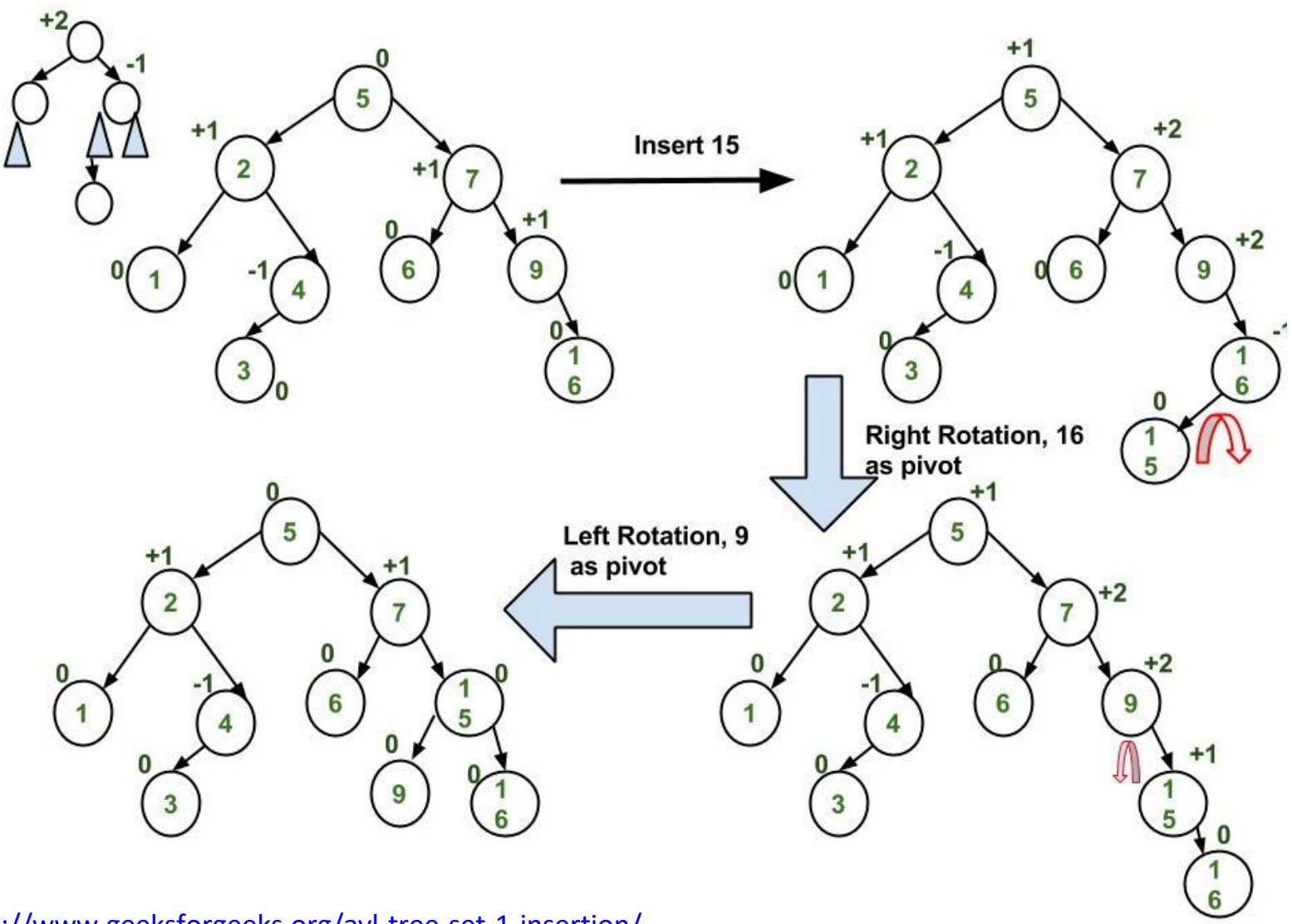
- 이중 회전 예시



https://www.geeksforgeeks.org/avl-tree-set-1-insertion22/

AVL 트리: 삽입

- 이중 회전 예시



https://www.geeksforgeeks.org/avl-tree-set-1-insertion/

트리 트래버스

- 프리오더 트래버스: 부모 노드에서 먼저 작업한 다음 자식 노드로 이동합니다.
- 후순위 순회: 자식 노드에서 먼저 작업한 다음 부모 노드로 이동합니다.
- 순서 순회: 키 순서대로 작업(예: 왼쪽 자식, 부모, 오른쪽 자식)
- 레벨 순서 순회: 깊이 D의 노드에서 작업한 다음 깊이 D+1의 노드에서 작업하거나 그 반대의 방식으로 작업합니다.

운동

- 이진 검색 트리에 [8, 4, 12, 1, 6, 15, 5, 7]의 요소를 가진 노드를 삽입합니다. 다음 함수를 구현합니다.
  - 최대 깊이
  - 사전 주문 트래버
  - 순서 순회
  - 포스트 오더 트래버스

최대 깊이: 3

선주문 통과: 8 4 1 6 5 7 12 15

순서 순회: 1 4 5 6 7 8 12 15