

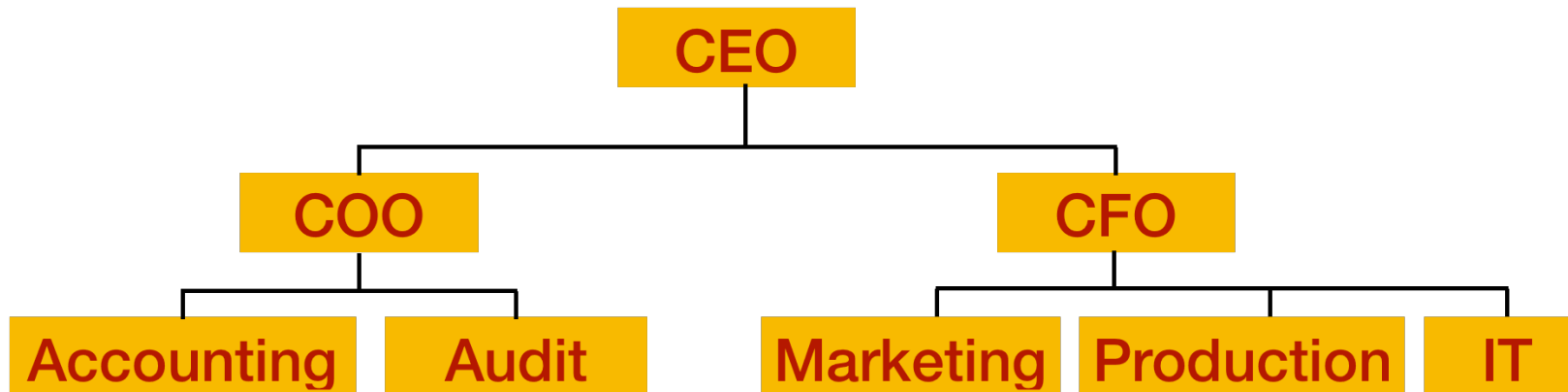
# Tree

즐겁고 알찬 자료구조 튜터링

**Tree**

# Tree

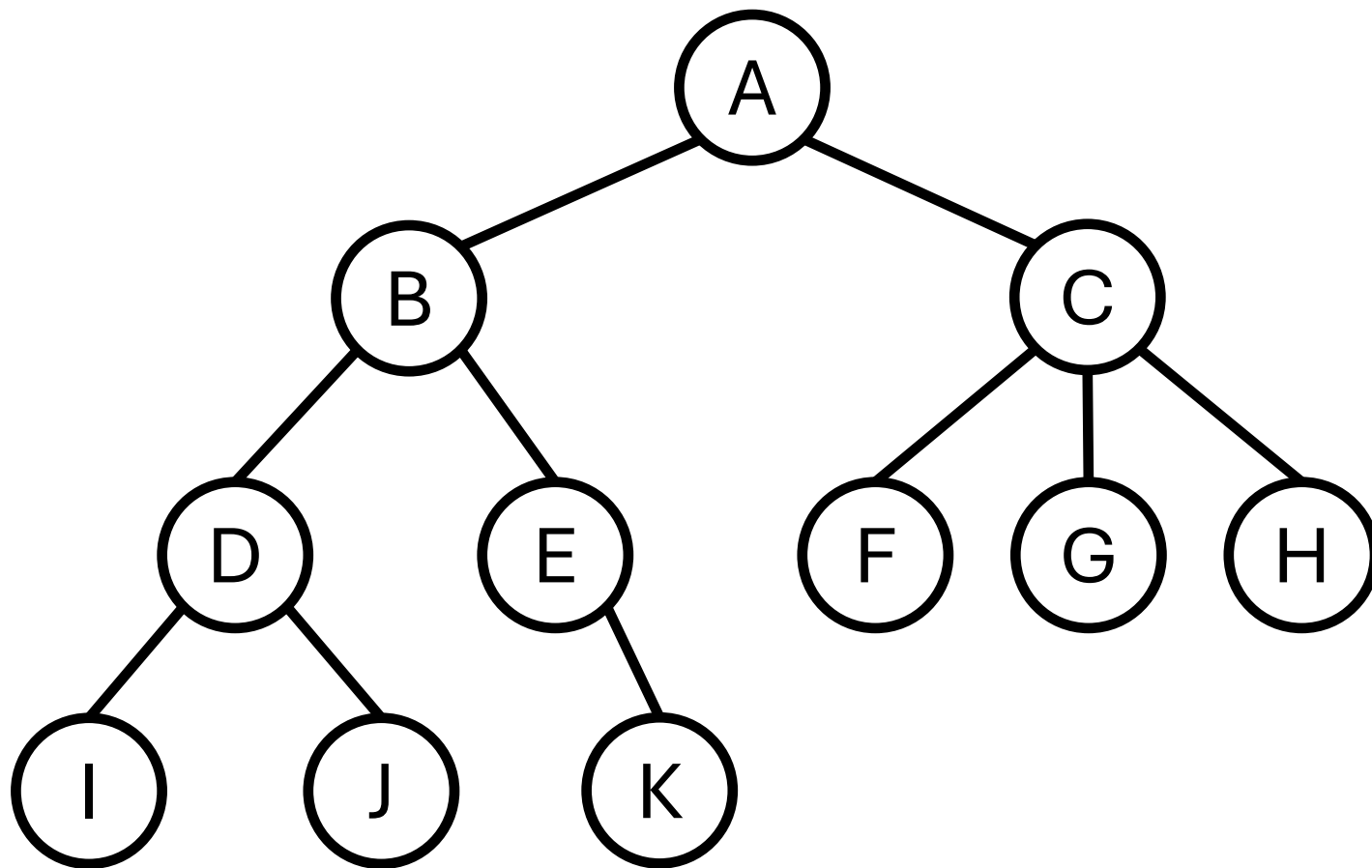
- 계층적 관계(Hierarchical Relationship)을 표현하는 비선형 자료구조



A Corporate Tree

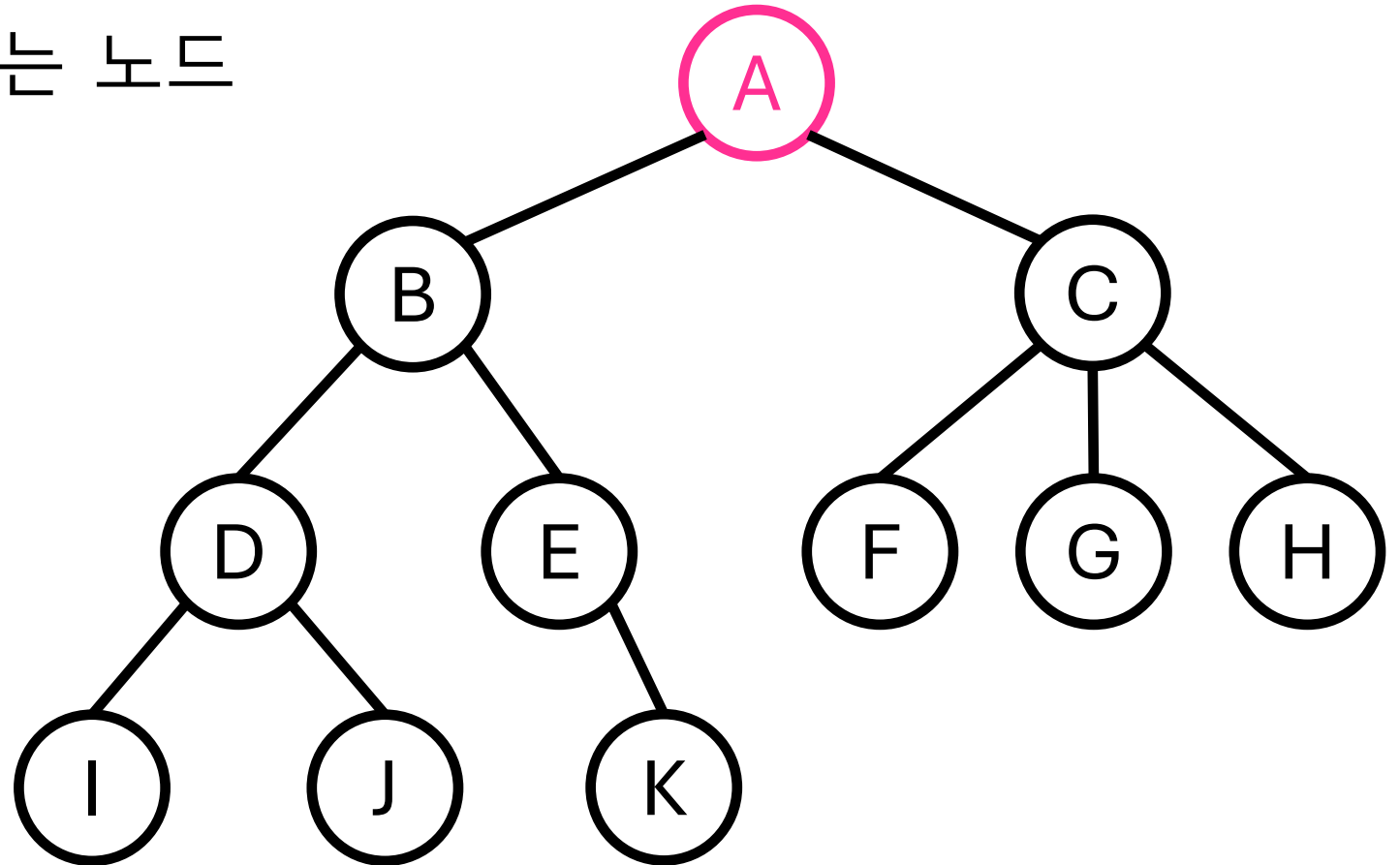
# Tree의 용어

- root
- internal node
- external node
- ancestors
- depth
- height
- descendant
- subtree



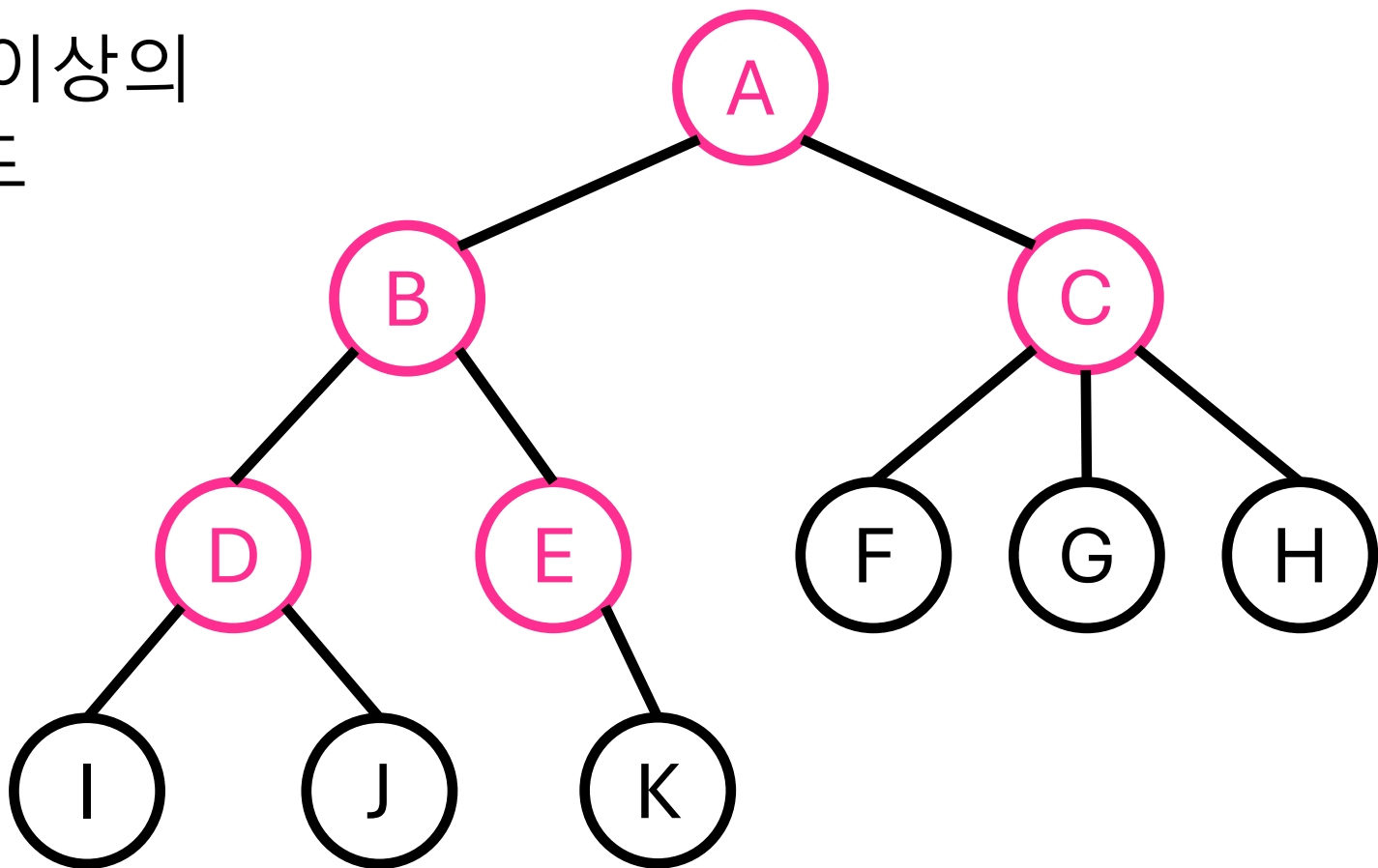
# Tree의 용어

- root : 부모 노드가 없는 노드



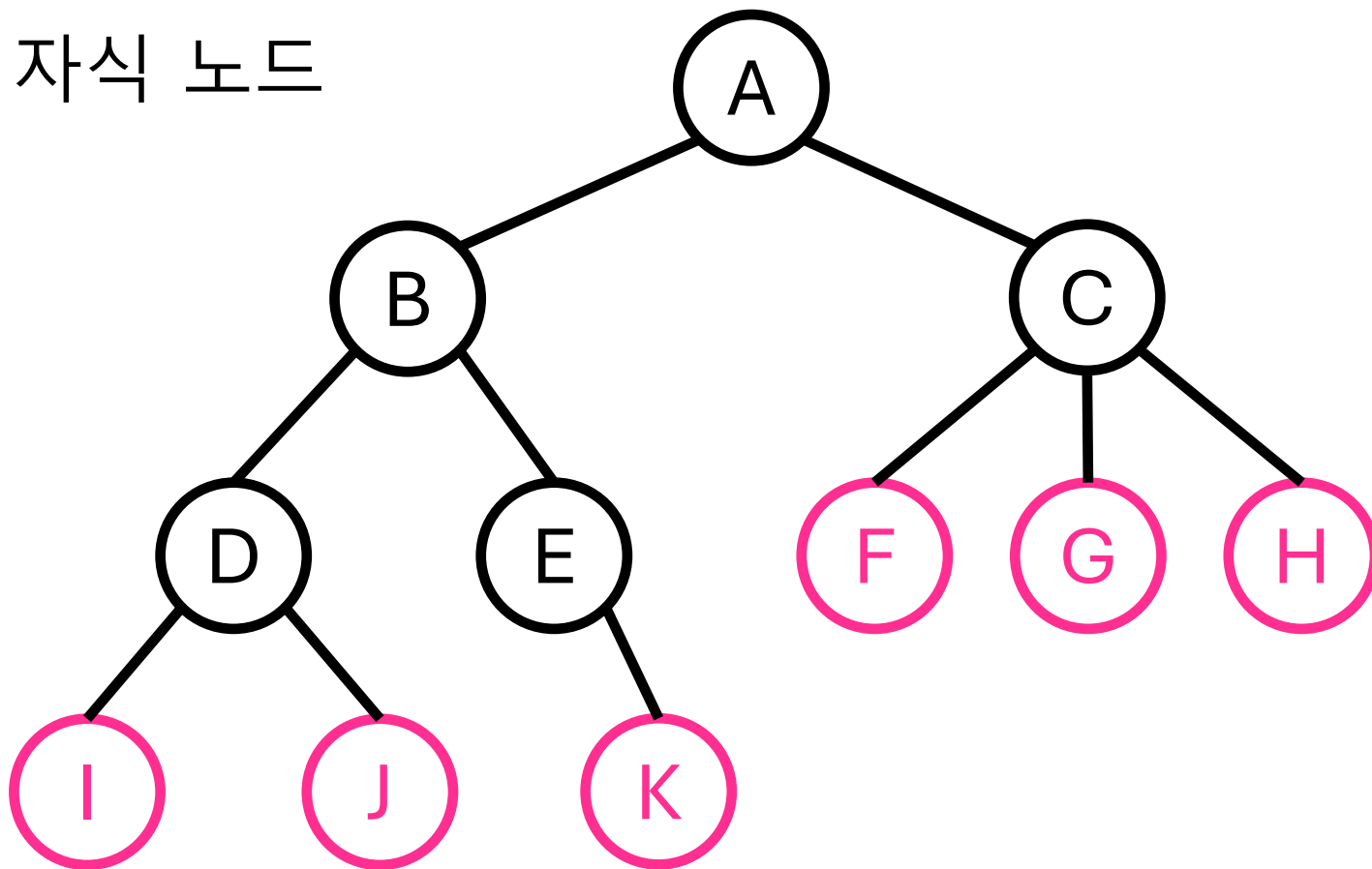
# Tree의 용어

- internal node : 하나 이상의 자식 노드를 가진 노드



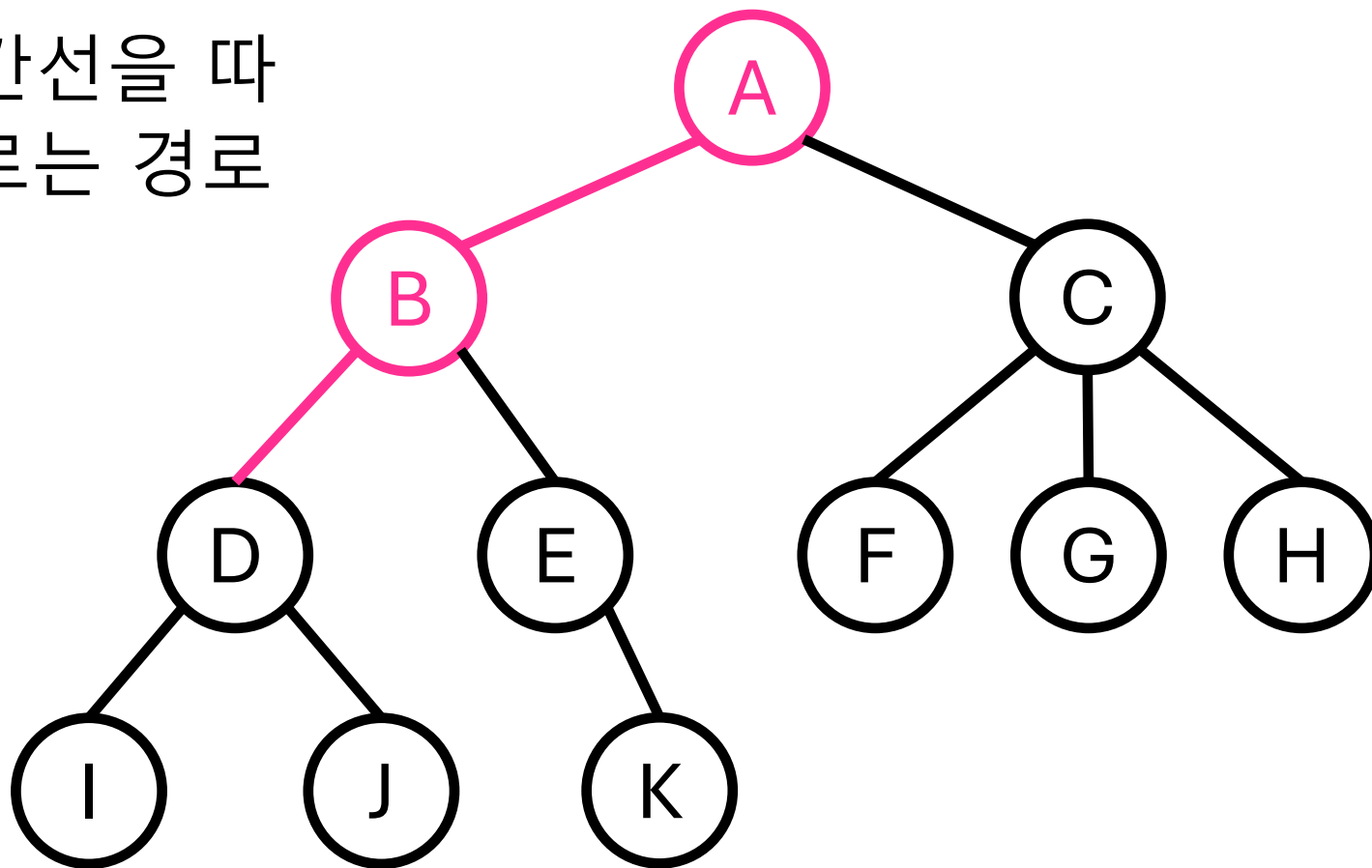
# Tree의 용어

- external node (leaf) : 자식 노드가 없는 노드



# Tree의 용어

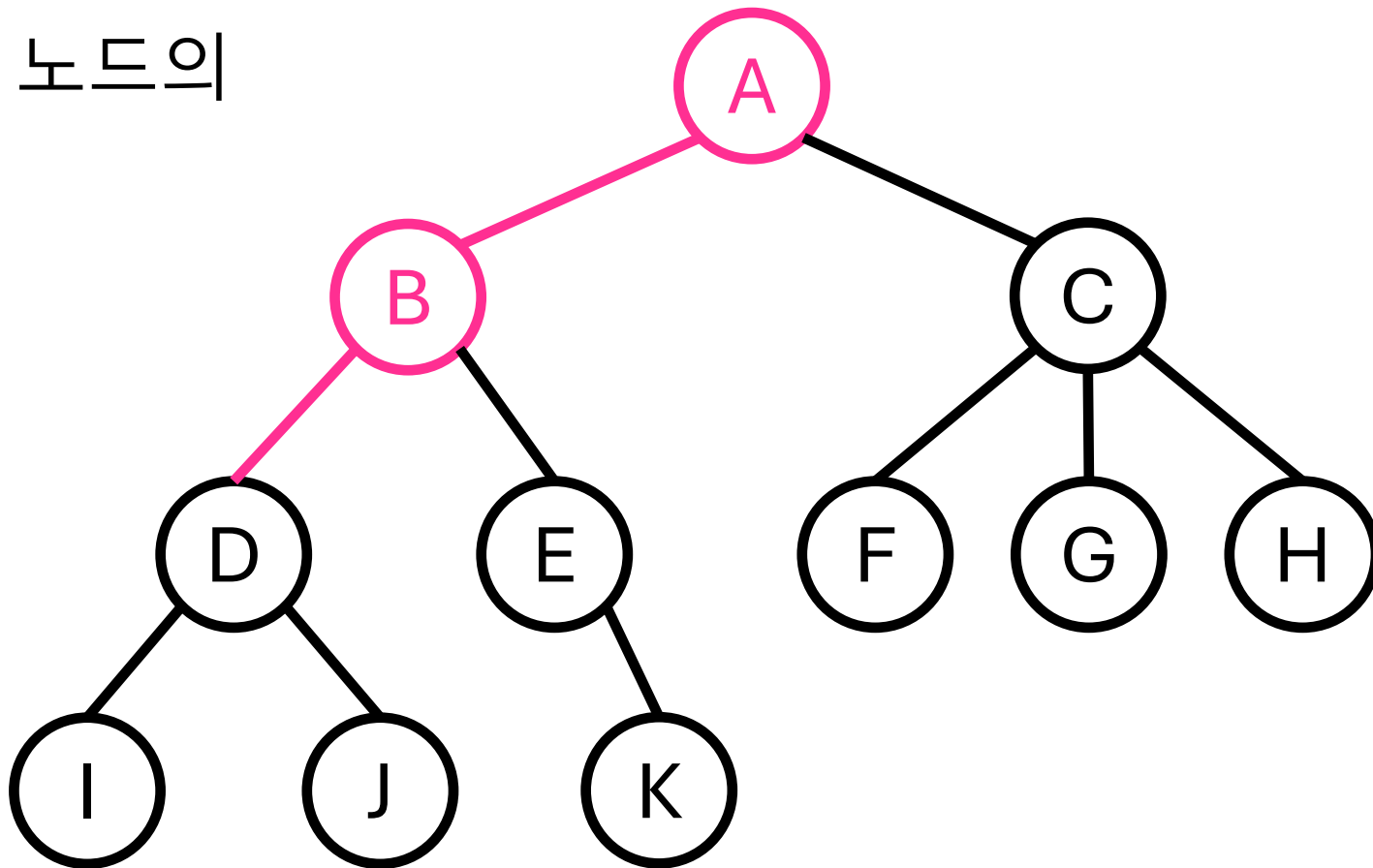
- ancestors of node : 간선을 따라 루트 노드까지 이르는 경로에 있는 모든 노드
- ancestors of D : A, B





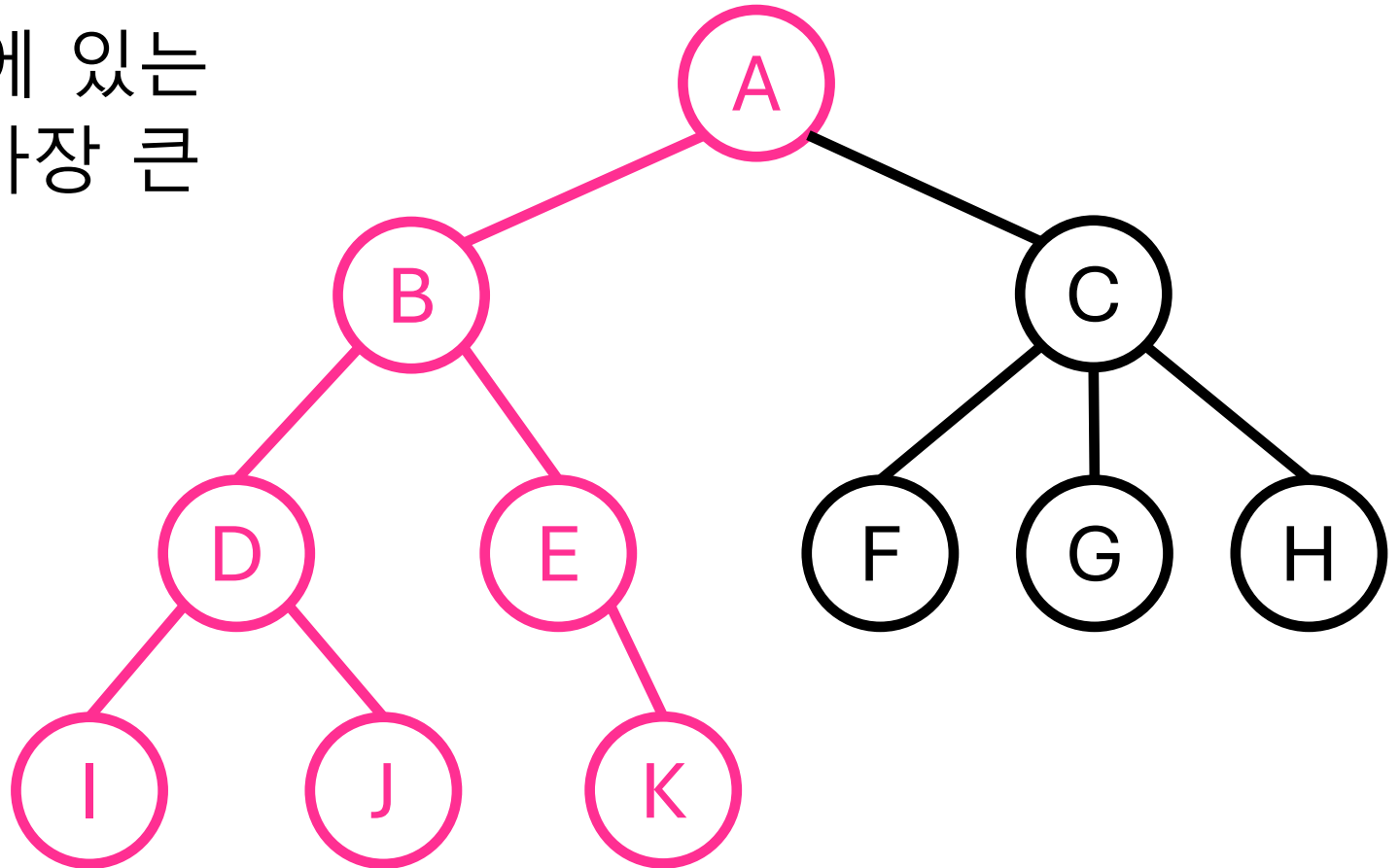
# Tree의 용어

- depth of node : 조상 노드의 개수
- ex) depth of D : 2



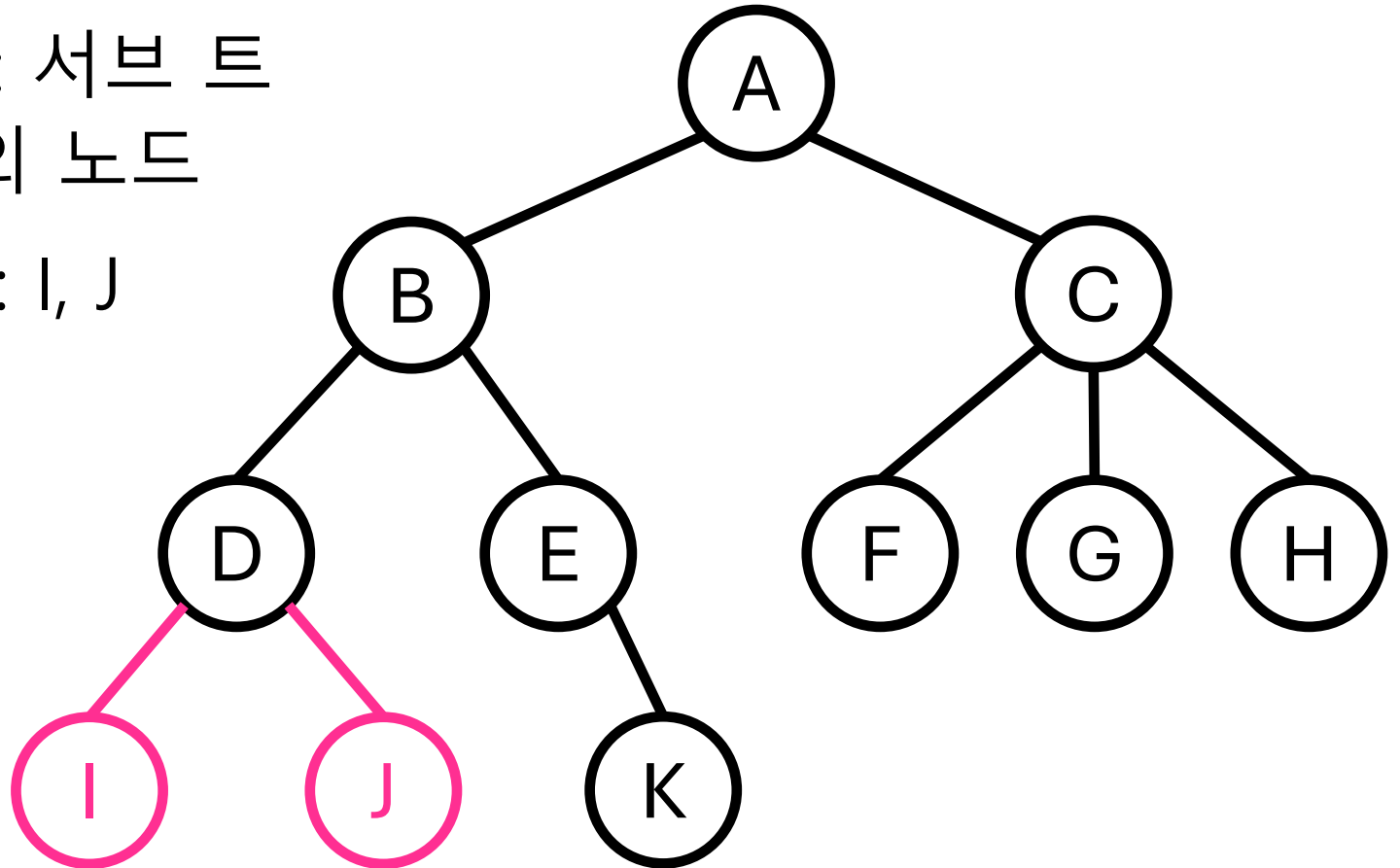
# Tree의 용어

- height of tree : 트리에 있는 노드의 높이 중에서 가장 큰 값



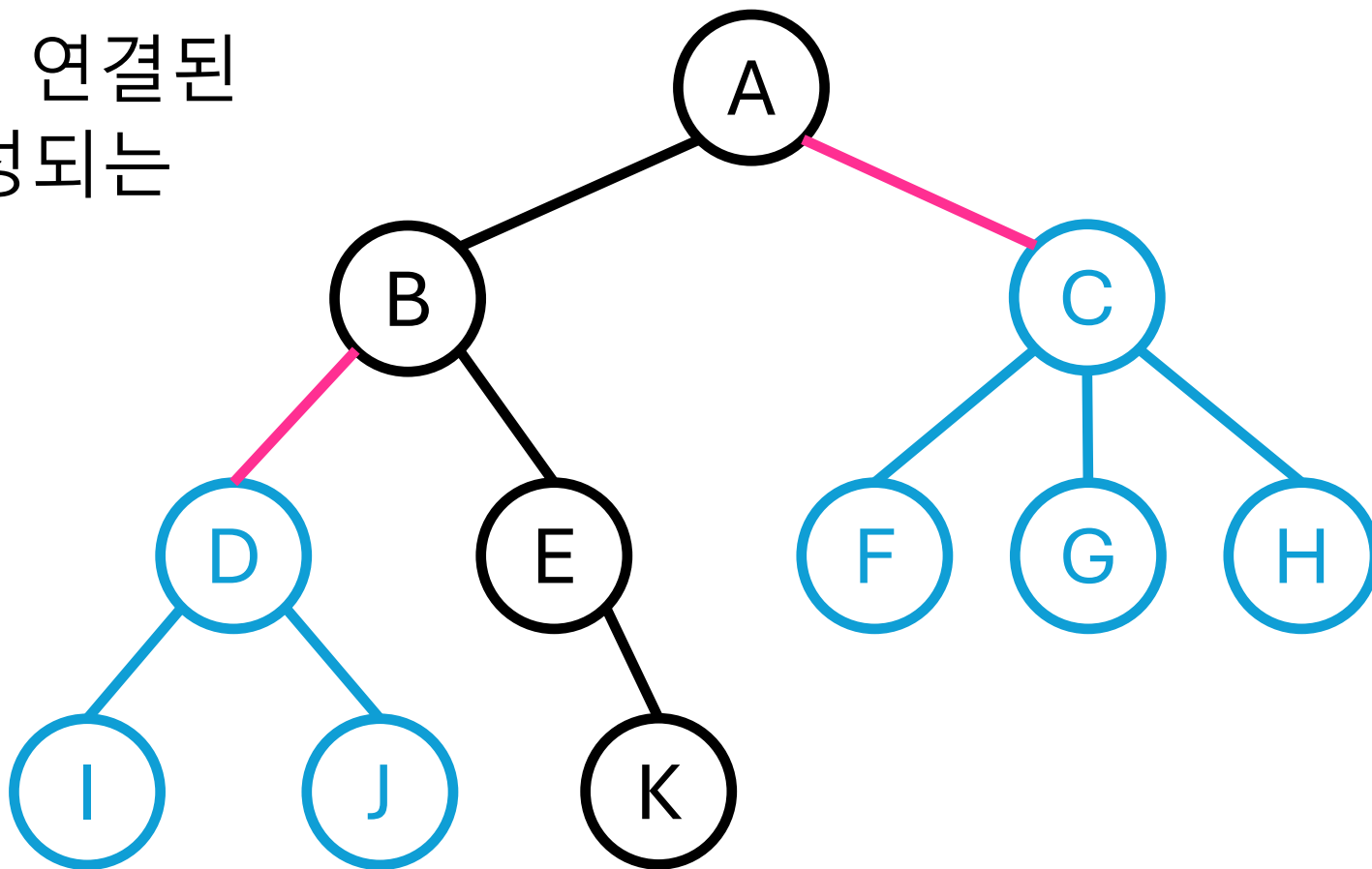
# Tree의 용어

- descendant of node : 서브 트리에 있는 하위 레벨의 노드
- ex) descendant of D : I, J



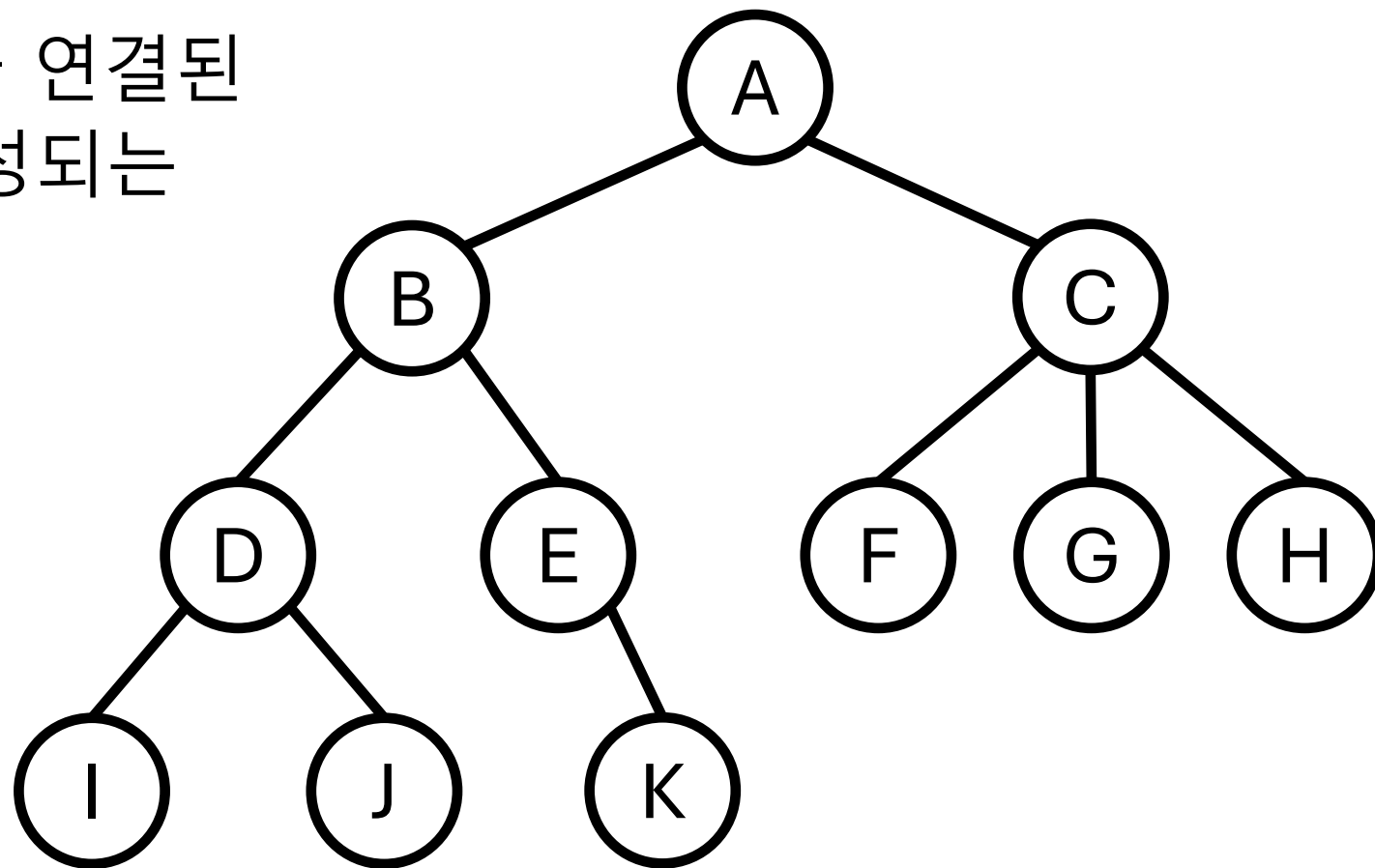
# Tree의 용어

- subtree : 부모 노드와 연결된 간선을 끊었을 때 생성되는 트리



# Tree의 구현

- subtree : 부모 노드와 연결된 간선을 끊었을 때 생성되는 트리

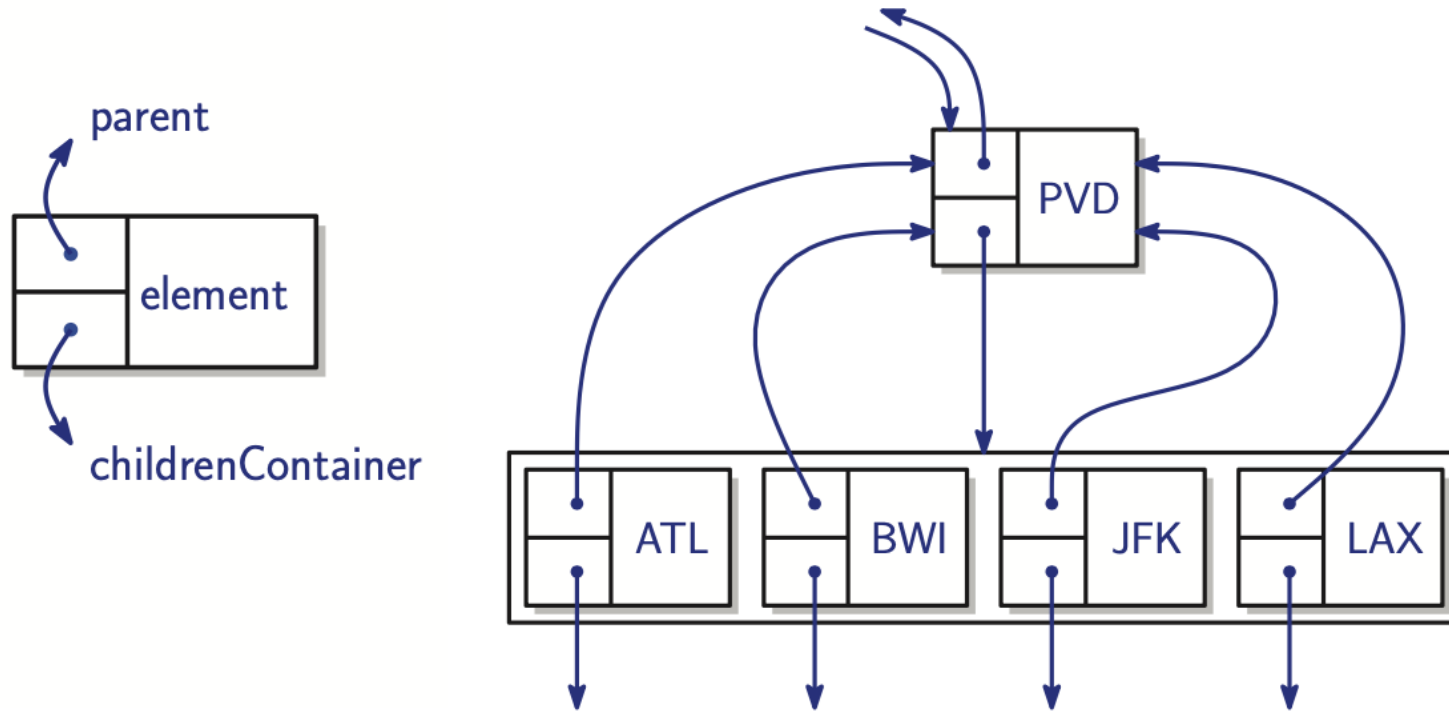


# Tree의 ADT

- `int size()` : 노드의 개수를 반환
- `bool empty()` : 트리가 비었는지 반환
- `Position root()` : 루트 노드를 반환
- `PositionList positions` : 트리에 있는 모든 노드들을 반환
- `Position p.parent()` : p의 부모 노드를 반환
- `PositionList p.children()` : p의 자식 노드 리스트를 반환
- `bool p.isRoot()` : p가 루트 노드인지 반환
- `bool p.isExternal()` : p가 리프 노드인지 반환

# Tree의 구현

- 노드가 가지는 정보 : 노드가 가지는 값, 부모 노드의 포인터, 자식 노드 리스트



# Position Class



```
1 class PositionList;  
2 template <typename T>  
3 class Position {  
4 private:  
5     T element;  
6     Position<T> *parent;  
7     PositionList children;
```



# Position Class



```
1 class PositionList;  
2 template <typename T>  
3 class Position {  
4 private:  
5     T element;  
6     Position<T> *parent;  
7     PositionList children;
```

# Position Class



```
1 T& operator*() {  
2     return element;  
3 }  
4 Position<T> parent() const {  
5     return *parent;  
6 }  
7 PositionList children() const {  
8     return children;  
9 }
```



```
1 bool isRoot() const {  
2     return parent == nullptr;  
3 }  
4 bool isExternal() const {  
5     return children.empty();  
6 }
```

# Tree Class



```
1 template <typename T>
2 class Tree {
3 private:
4     PositionList _positions;
```



```
1 int size() const {
2     return _positions.size();
3 }
4
5 bool empty() const {
6     return size() == 0;
7 }
```

# Tree Class



```
1 Position<T> root() const {  
2     for (Position<T> p : _positions) {  
3         if (p.isRoot()) {  
4             return p;  
5         }  
6     }  
7  
8     throw "Can not find root";  
9 }
```

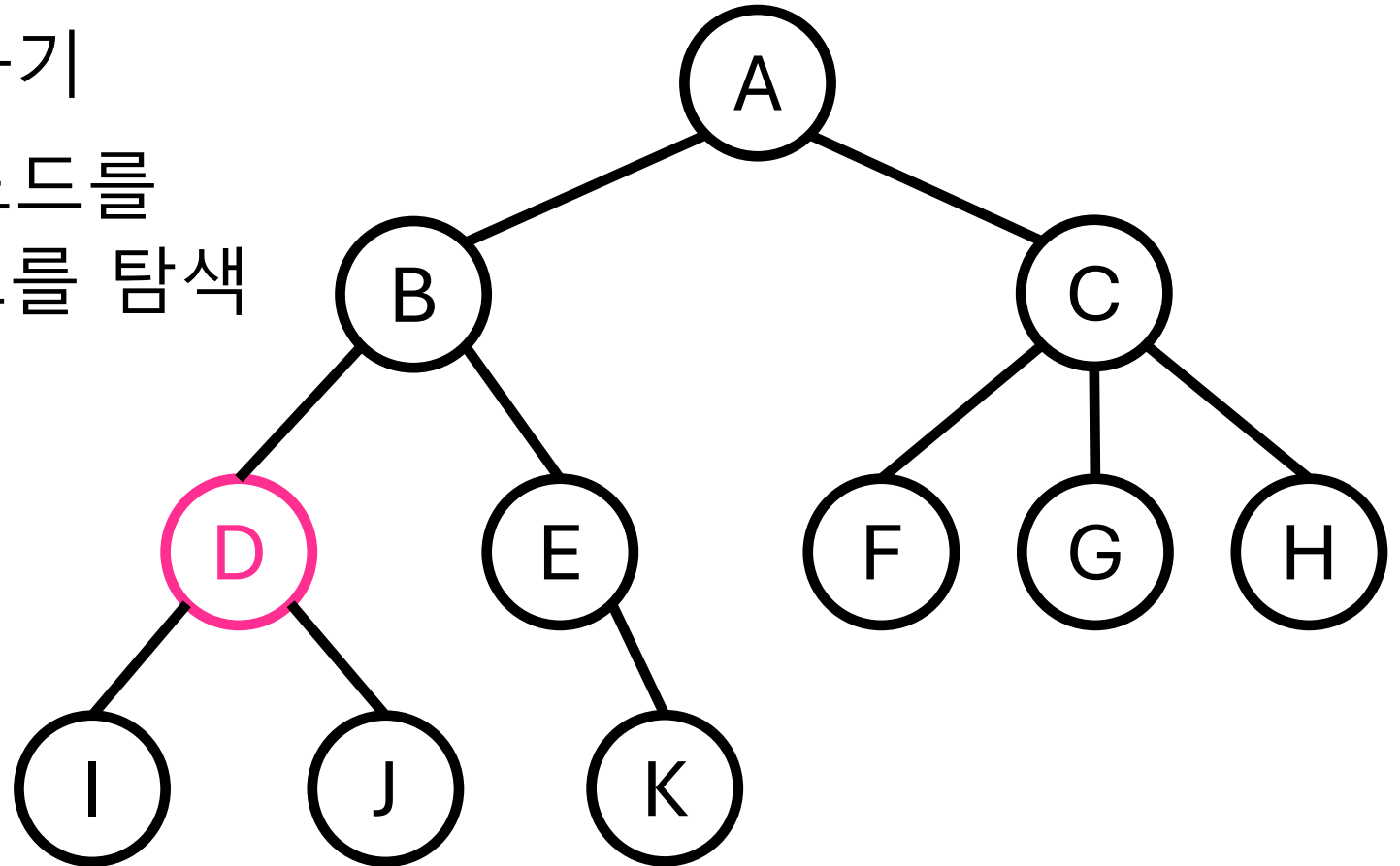


```
1 PositionList positions() {  
2     return _positions;  
3 }
```

# Tree Traversal

# Depth of a node

- 노드 D의 높이를 구하기
- D부터 시작해 루트 노드를 만나기까지 부모 노드를 탐색



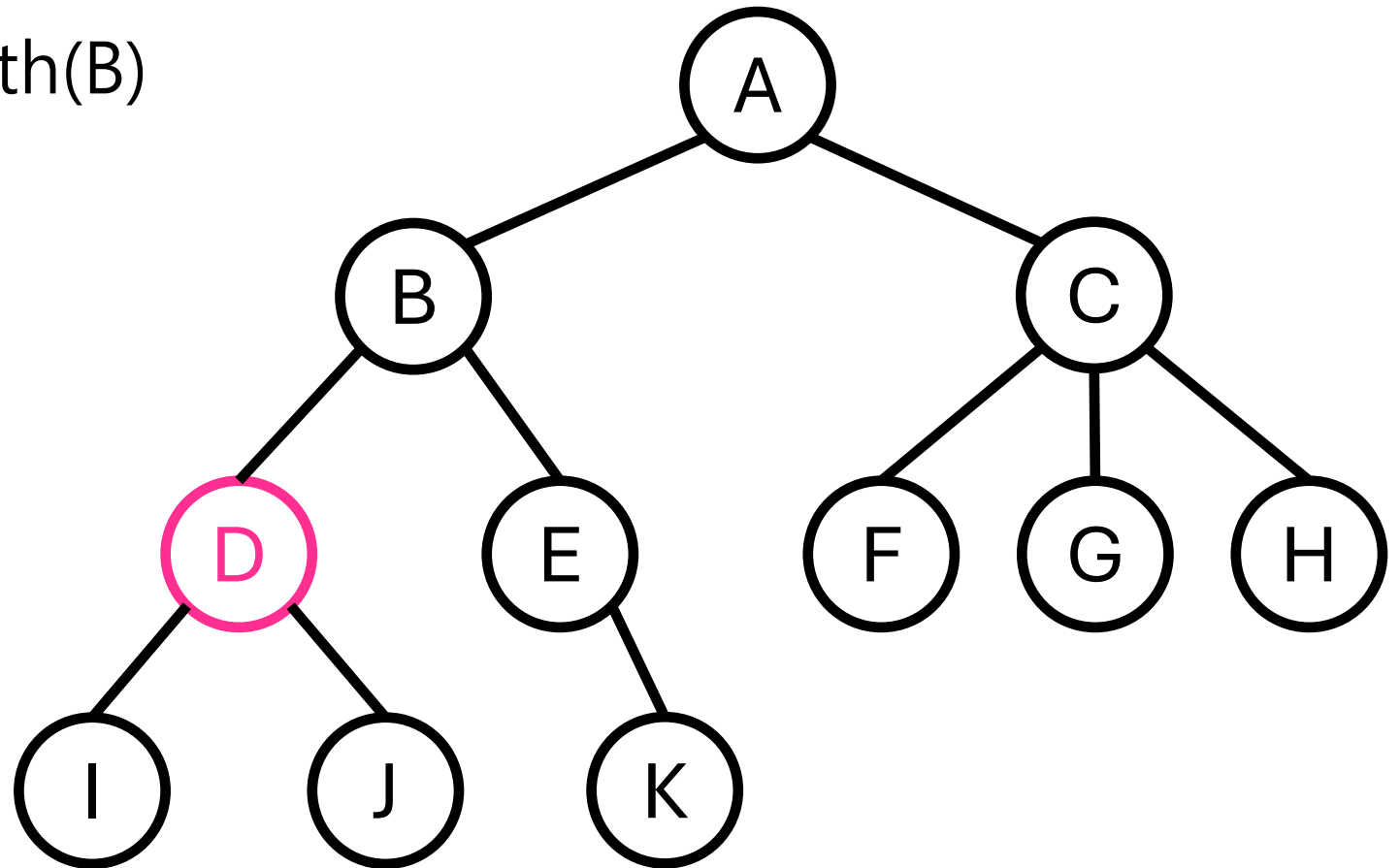
# Depth of a node



```
1 int depth(const Tree<int> &t, const Position<int> &p) {  
2     if (p.isRoot())  
3         return 0;  
4     else  
5         return 1 + depth(t, p.parent());  
6 }
```

# Depth of a node

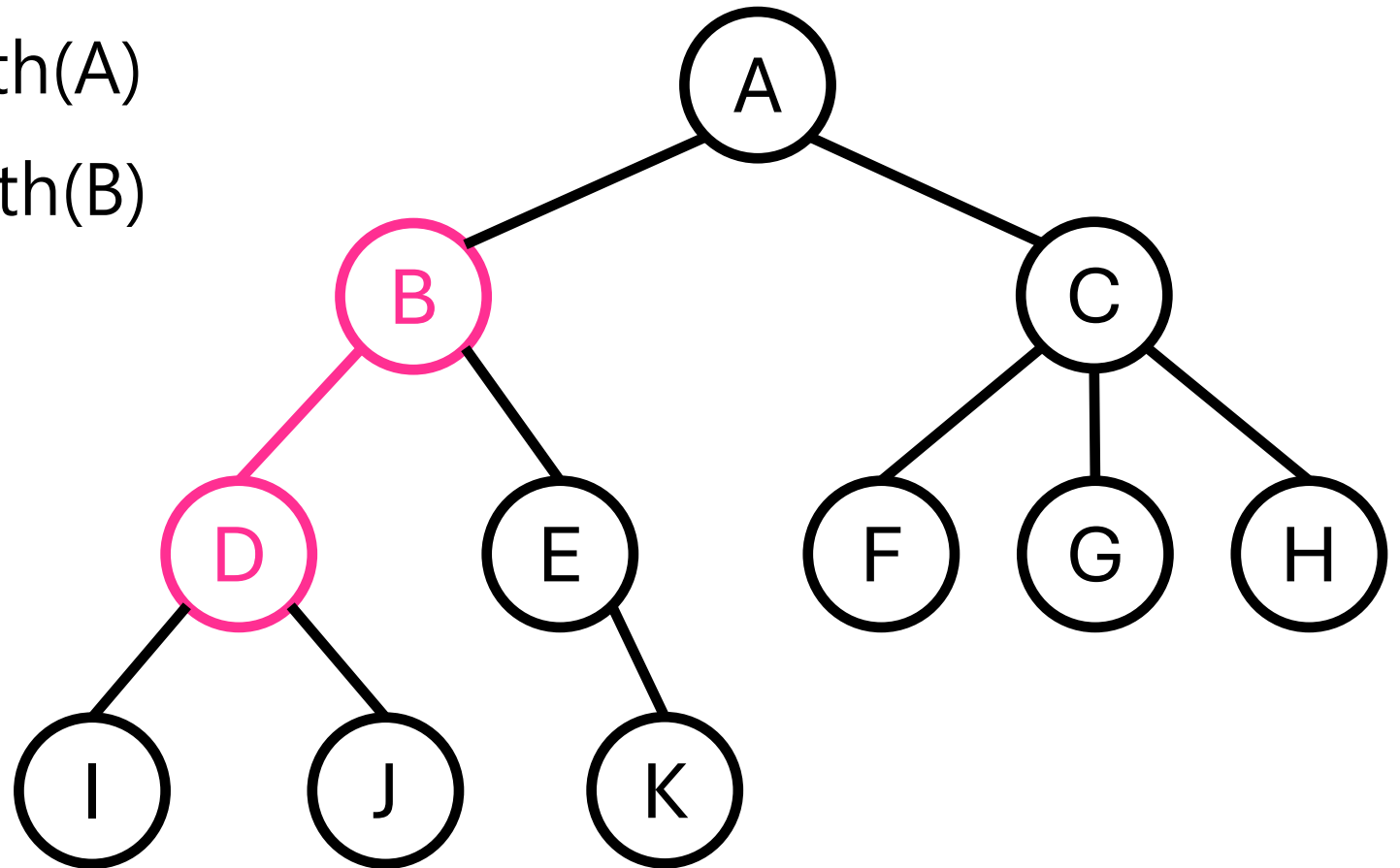
- $\text{depth}(D) \rightarrow 1 + \text{depth}(B)$





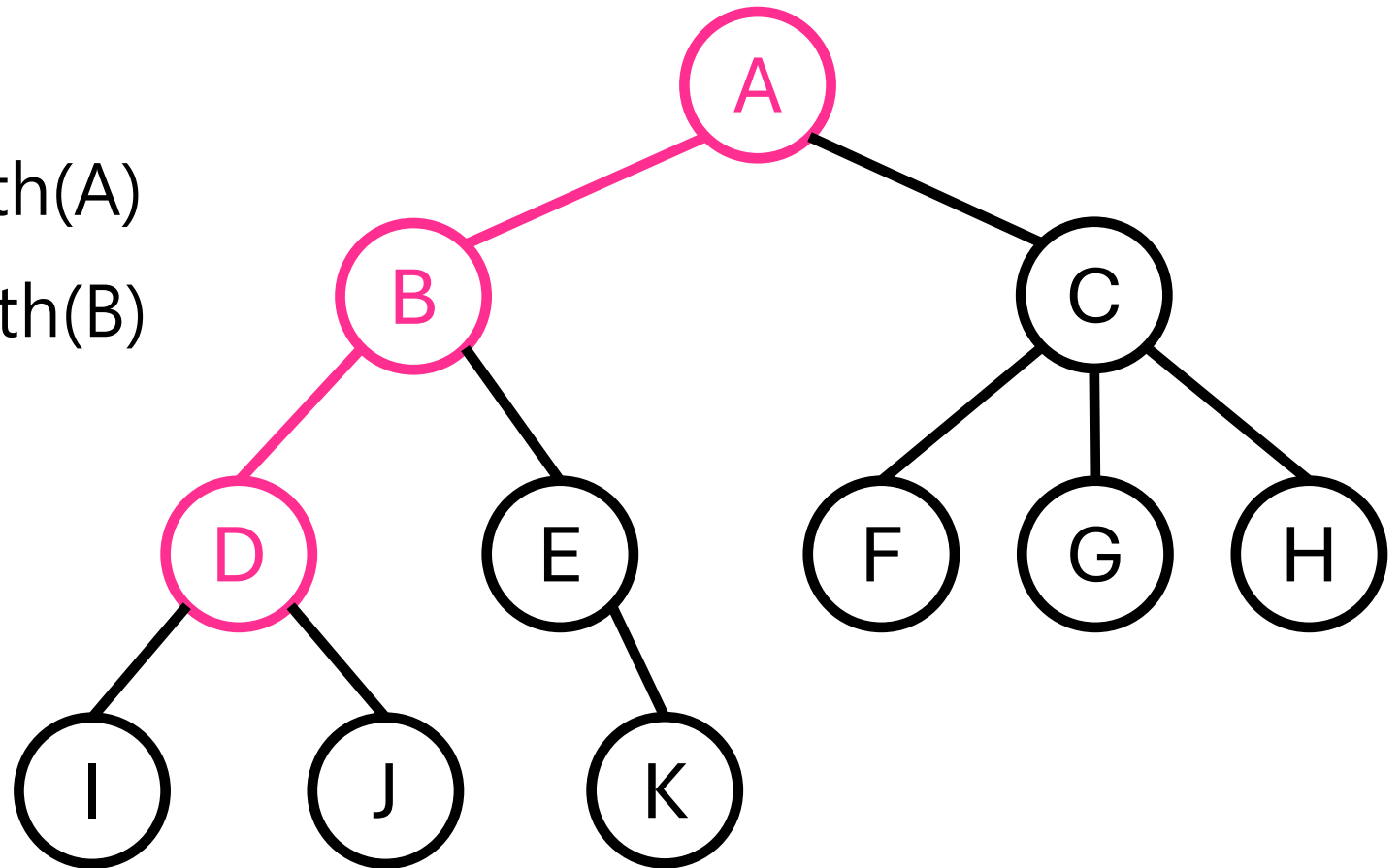
# Depth of a node

- $\text{depth}(B) \rightarrow 1 + \text{depth}(A)$
- $\text{depth}(D) \rightarrow 1 + \text{depth}(B)$



# Depth of a node

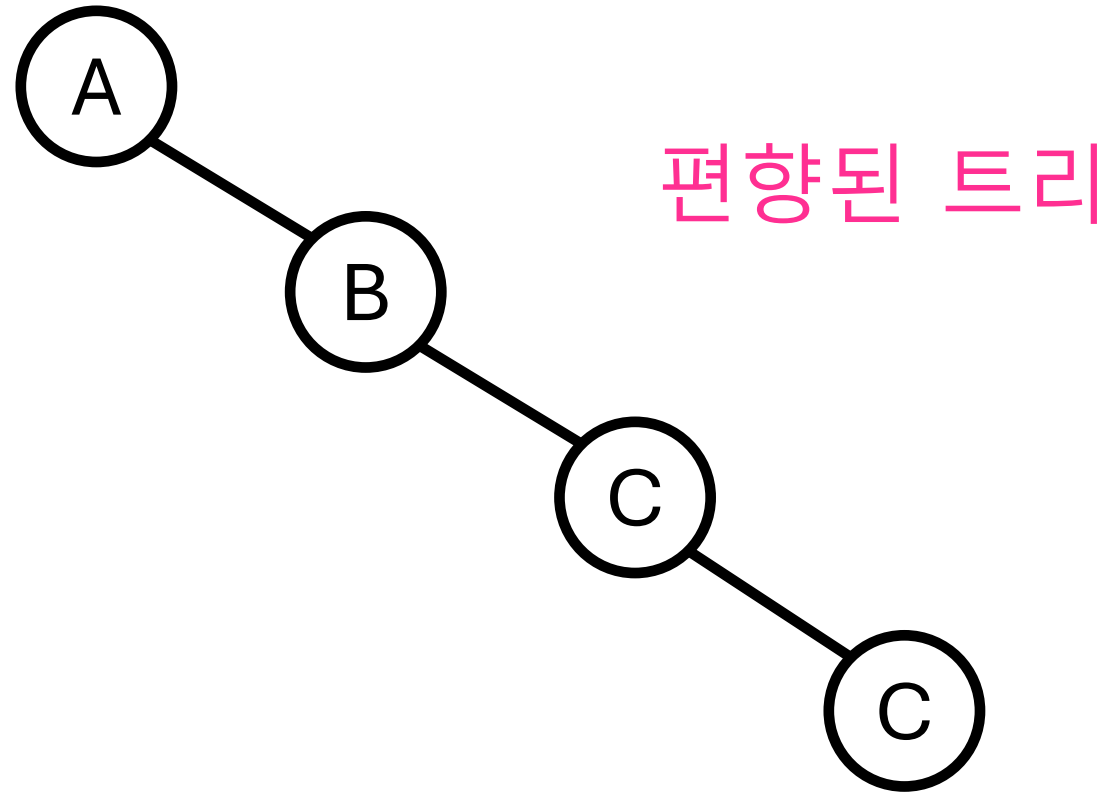
- $\text{depth}(A) \rightarrow 0$
- $\text{depth}(B) \rightarrow 1 + \text{depth}(A)$
- $\text{depth}(D) \rightarrow 1 + \text{depth}(B)$



# depth 함수의 복잡도

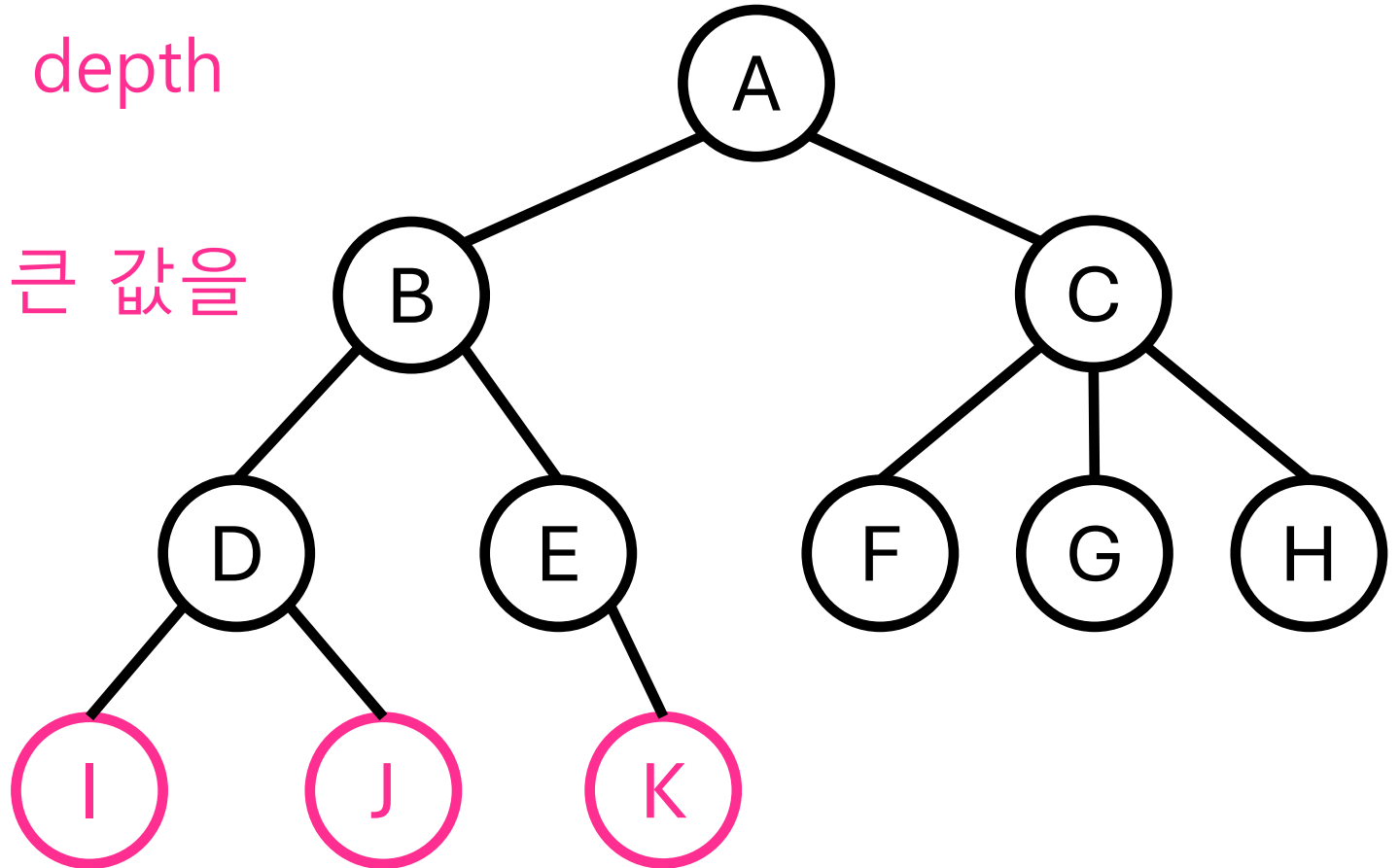
- 최악의 경우 분석
  - 단위연산 : 노드가 루트 노드인지 확인
  - 입력크기 : 노드의 개수  $n$
  - 최악의 경우 트리가 편향된 경우(트리가 한쪽으로 치우쳐짐)에서 리프 노드의 depth를 계산하면 모든 노드에 대해서 함수가 호출
  - $W(n) = n$

# depth 함수의 복잡도



# Height of a tree

- 트리에서 모든 노드의 depth를 계산
- 노드의 depth중 가장 큰 값을 height로 계산



# Height of a tree



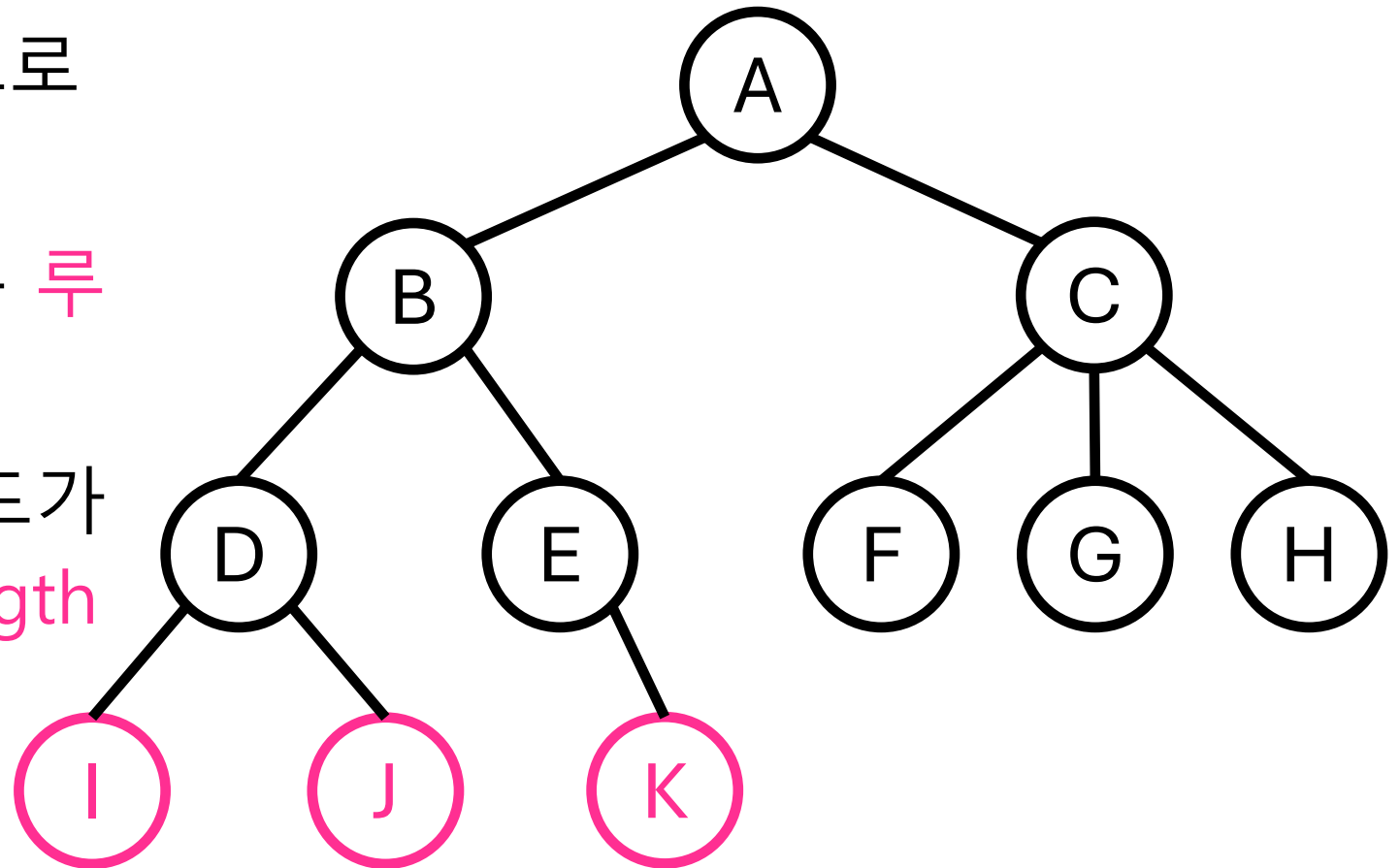
```
1  int height1(const Tree<int> &t) {  
2      int h = 0;  
3      PositionList nodes = t.positions();  
4      for (auto q = nodes.begin(); q != nodes.end(); ++q) {  
5          if (q->isExternal()) {  
6              h = max(h, depth(t, q));  
7          }  
8      }  
9      return h;  
10 }
```

# height1 함수의 복잡도

- 최악의 경우 분석
  - 단위연산 : depth 함수 호출
  - 입력크기 : 노드의 개수  $n$
  - depth 함수의 최악의 경우 복잡도는  $n$ 이고, depth 함수는 모든 노드에 대해서 호출
  - $W(n) = n^2$

# 개선된 height 함수

- 노드에 대해 재귀적으로 계산
- 최초 호출되는 노드는 루트 노드
- 해당 노드가 루트 노드가 되는 서브 트리의 height를 계산





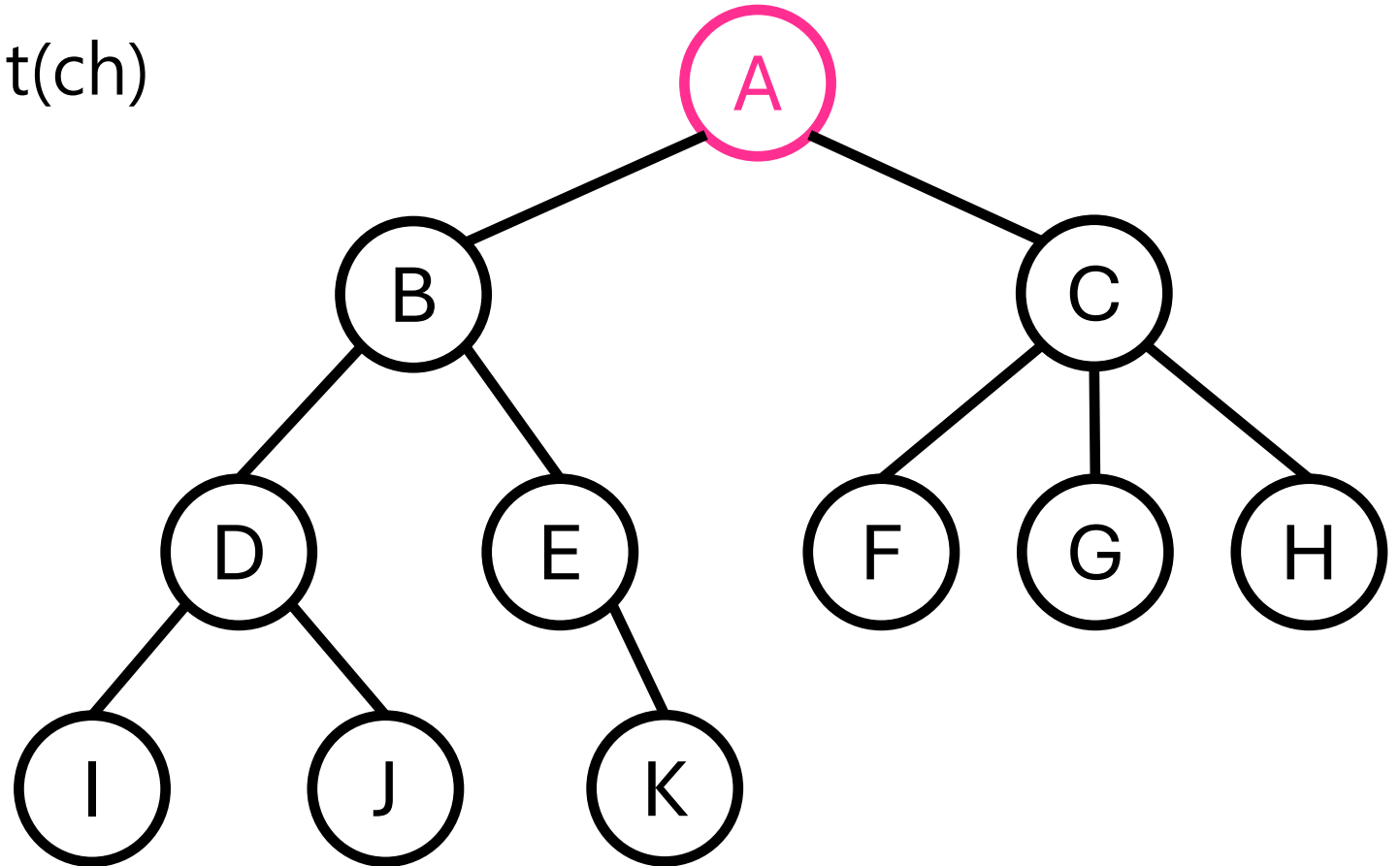
# 개선된 height 함수



```
1  int height2(const Tree<int> &t, const Position<int> &p) {
2      if (p.isExternal())
3          return 0;
4
5      int h = 0;
6      PositionList ch = p.children();
7      for (auto q = ch.begin(); q != ch.end(); ++q)
8          h = max(h, height2(t, *q));
9      return 1 + h;
10 }
```

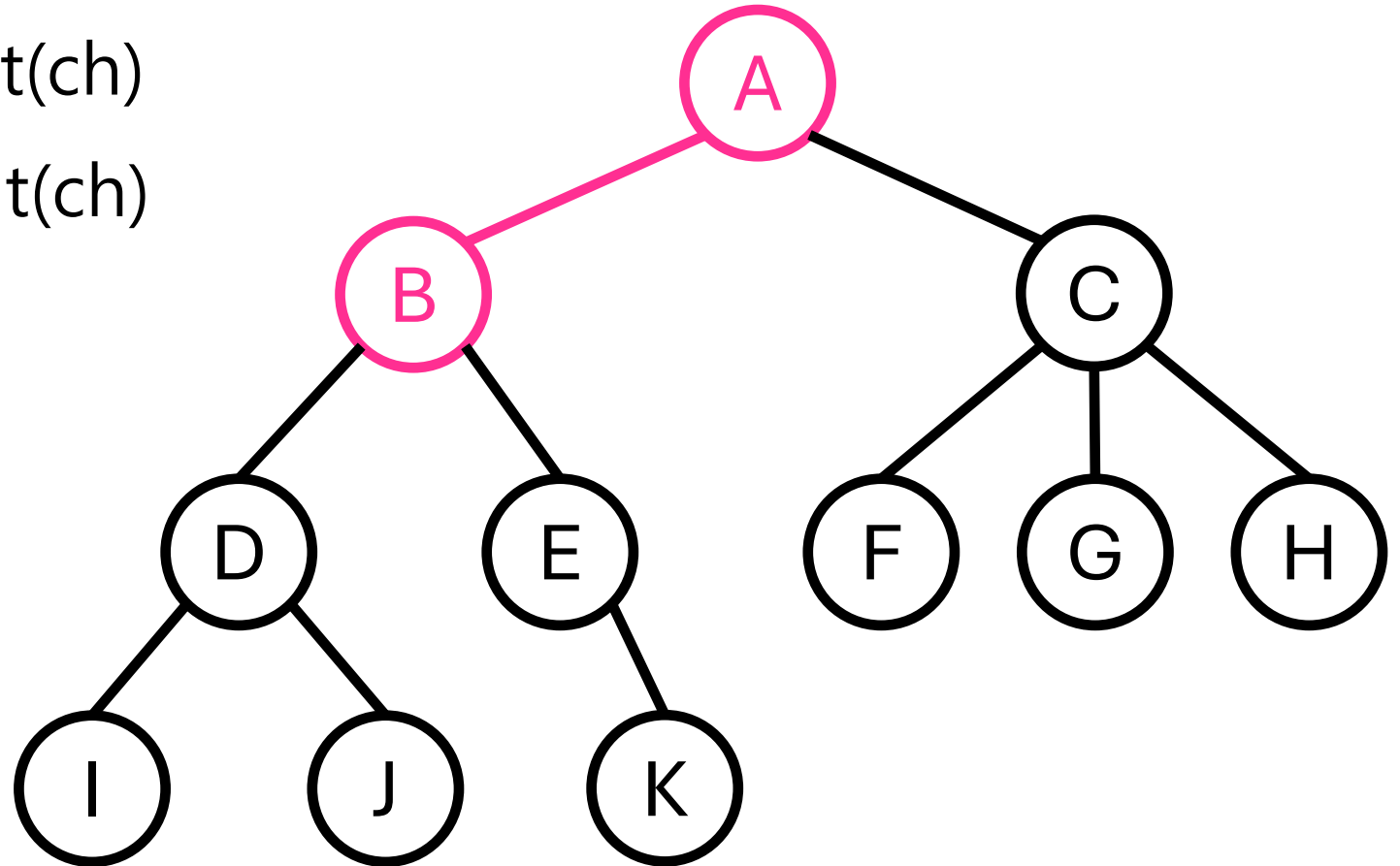
# 개선된 height 함수

- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$



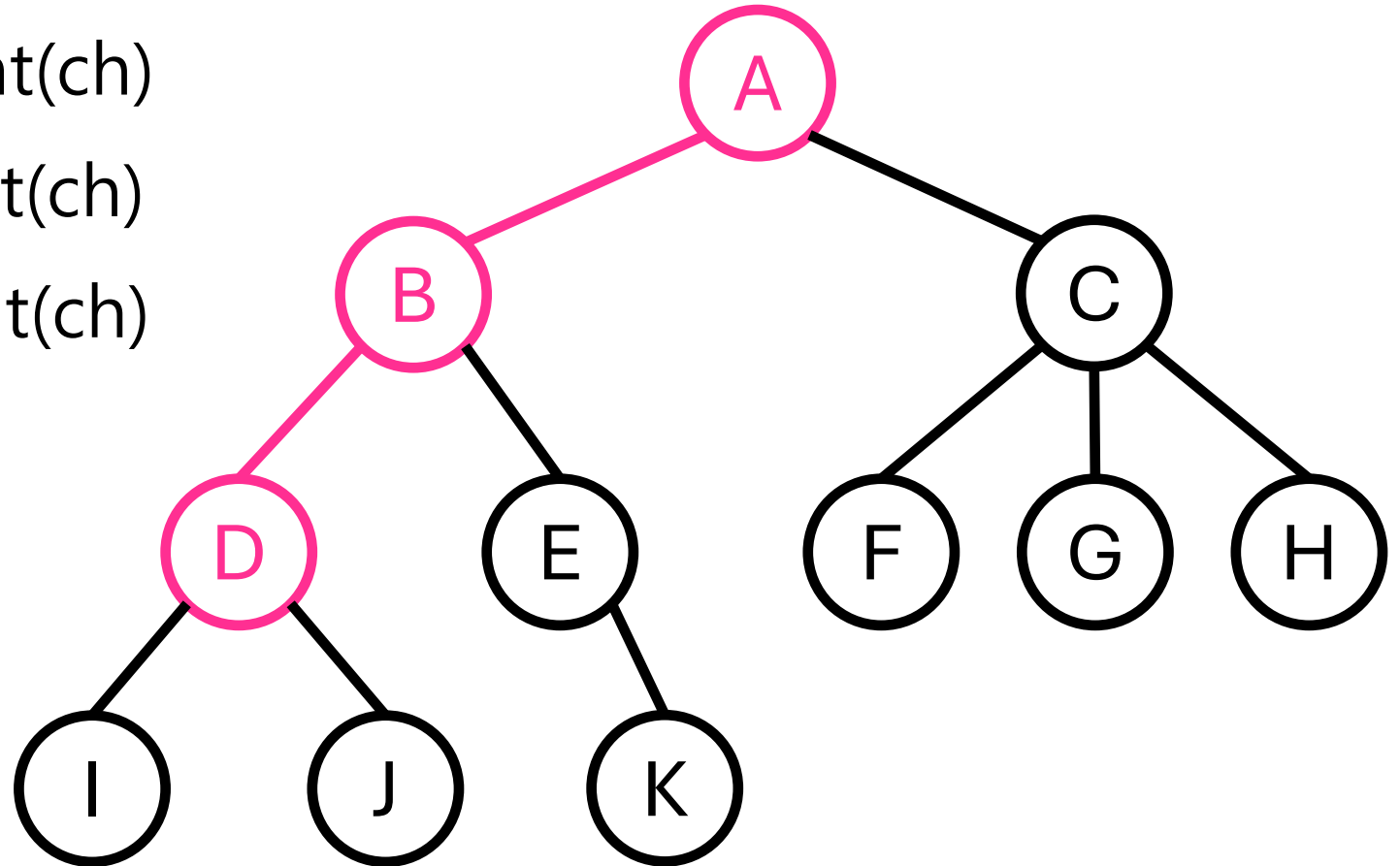
# 개선된 height 함수

- $\text{height}(B) \rightarrow 1 + \text{height}(\text{ch})$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$



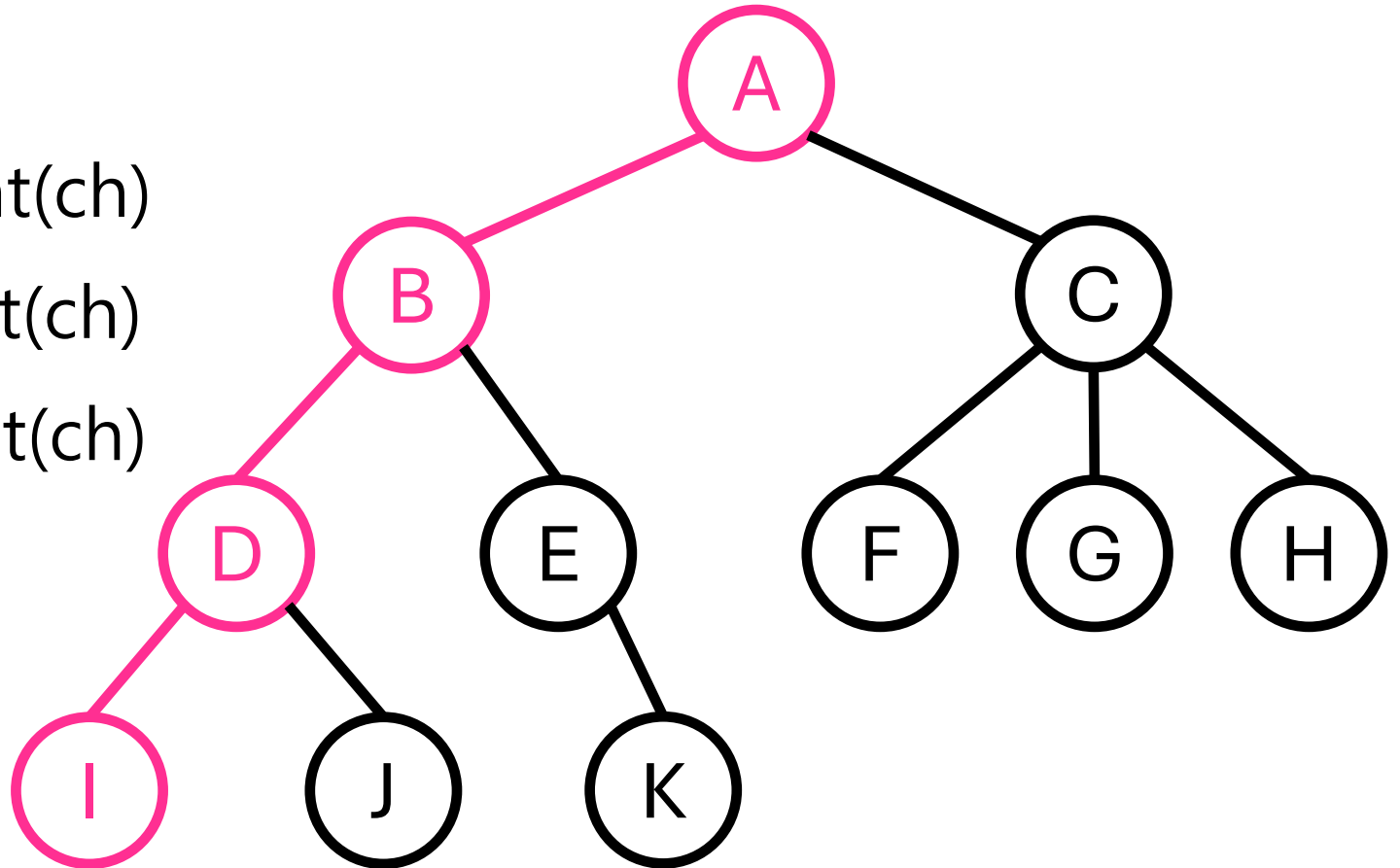
# 개선된 height 함수

- $\text{height}(D) \rightarrow 1 + \text{height}(\text{ch})$
- $\text{height}(B) \rightarrow 1 + \text{height}(\text{ch})$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$



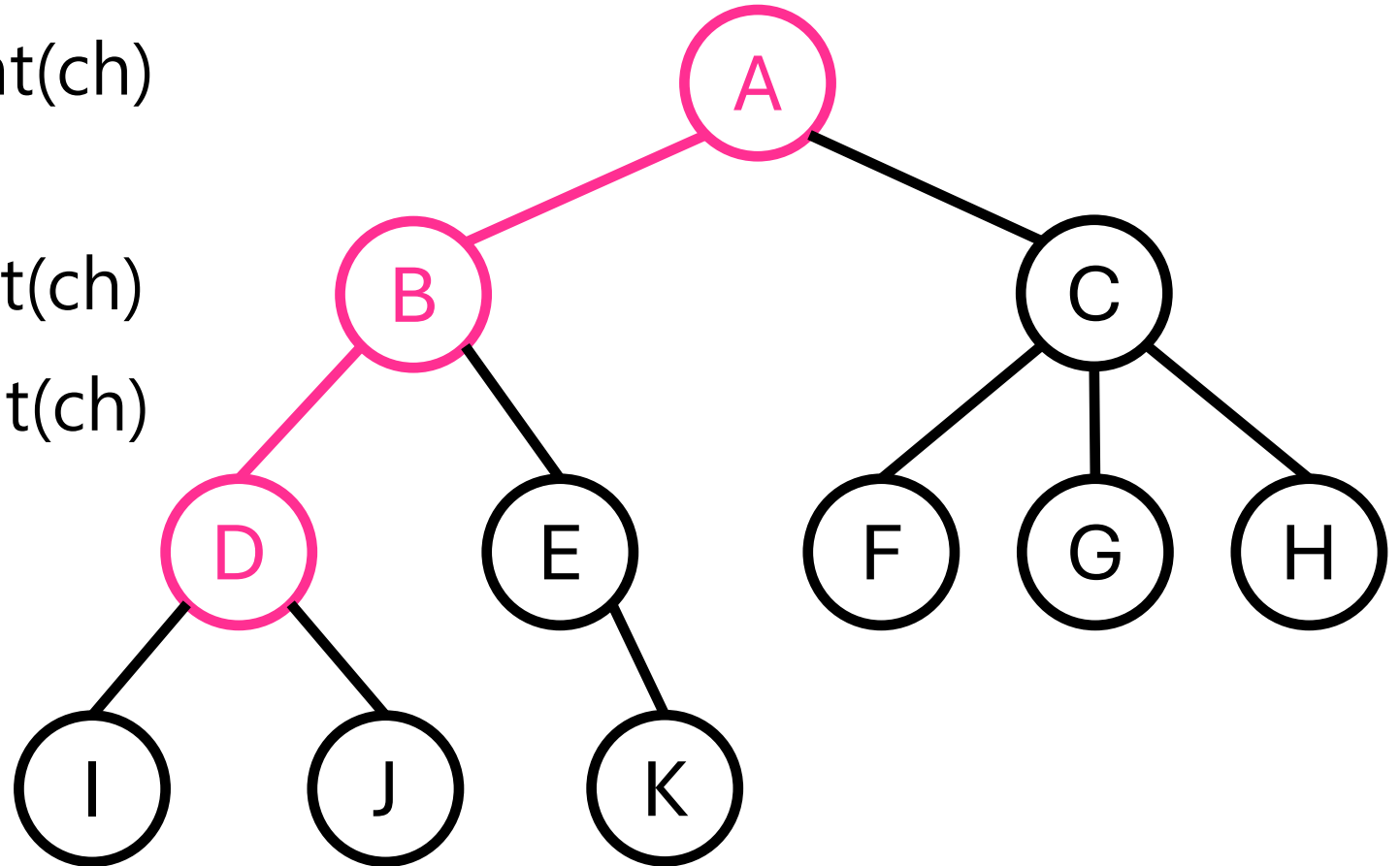
# 개선된 height 함수

- $\text{height}(I) \rightarrow 0$
- $\text{height}(D) \rightarrow 1 + \text{height}(ch)$
- $\text{height}(B) \rightarrow 1 + \text{height}(ch)$
- $\text{height}(A) \rightarrow 1 + \text{height}(ch)$



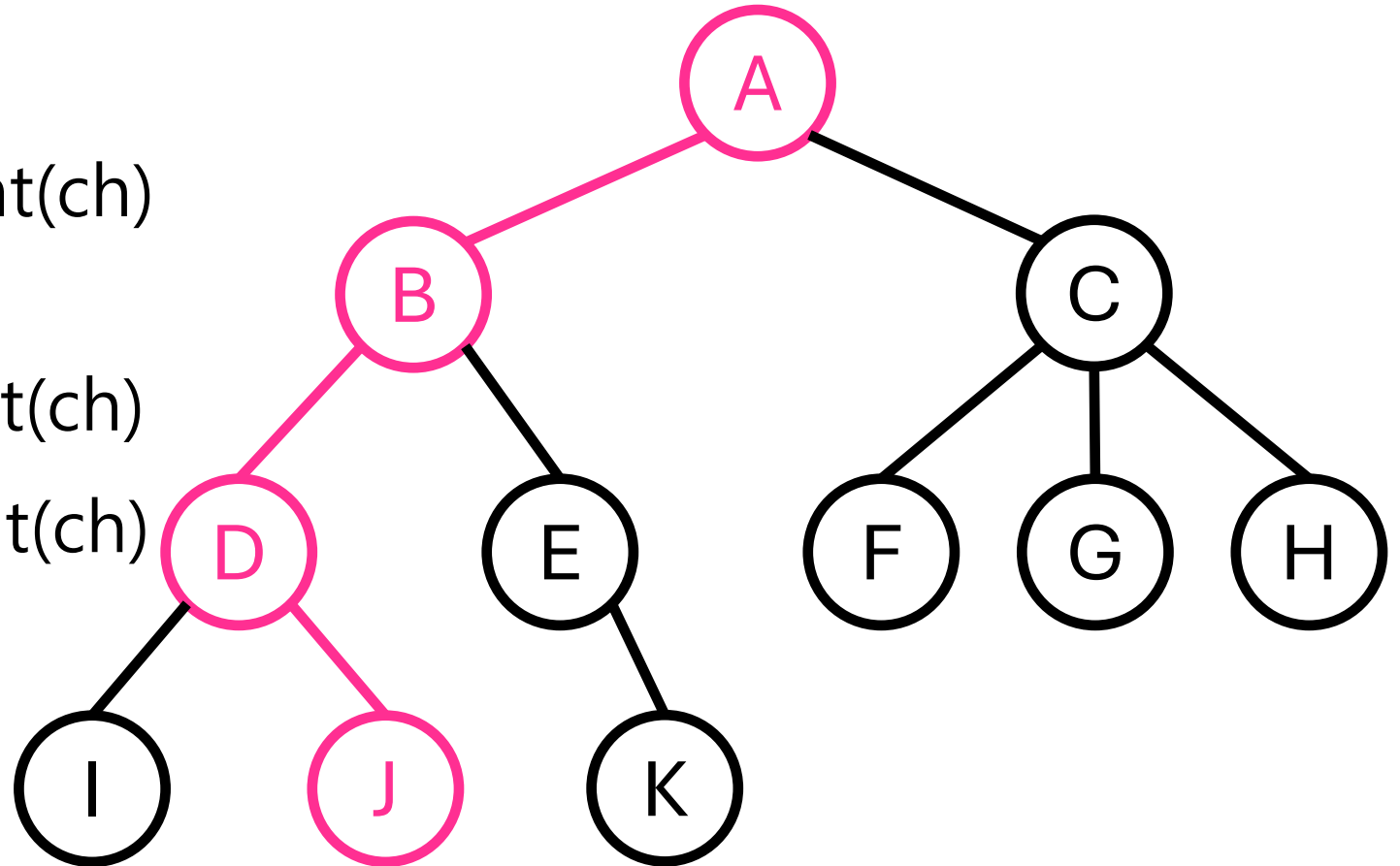
# 개선된 height 함수

- $\text{height}(D) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 1$
- $\text{height}(B) \rightarrow 1 + \text{height}(\text{ch})$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$



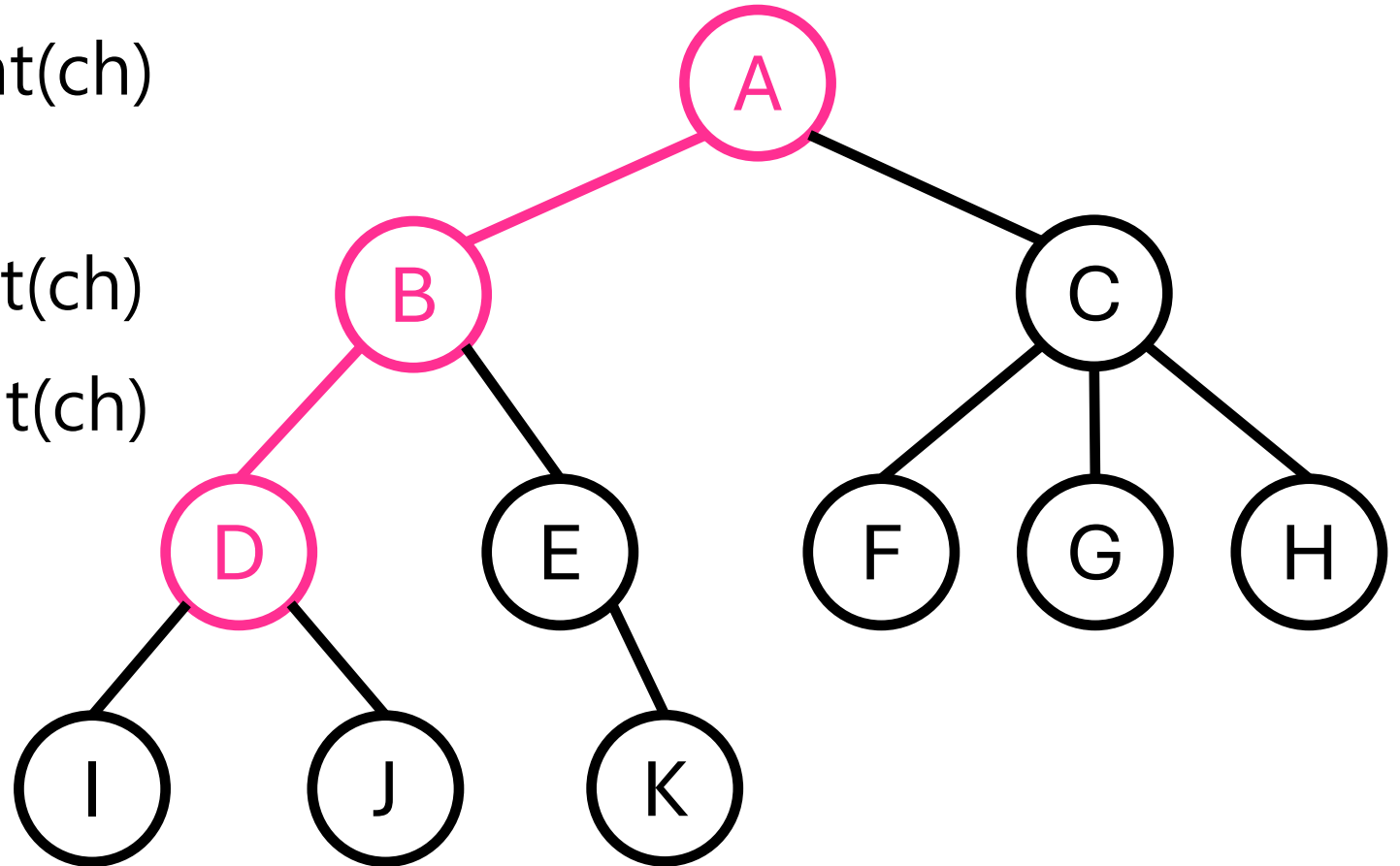
# 개선된 height 함수

- $\text{height}(J) \rightarrow 0$
- $\text{height}(D) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 1$
- $\text{height}(B) \rightarrow 1 + \text{height}(\text{ch})$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$



# 개선된 height 함수

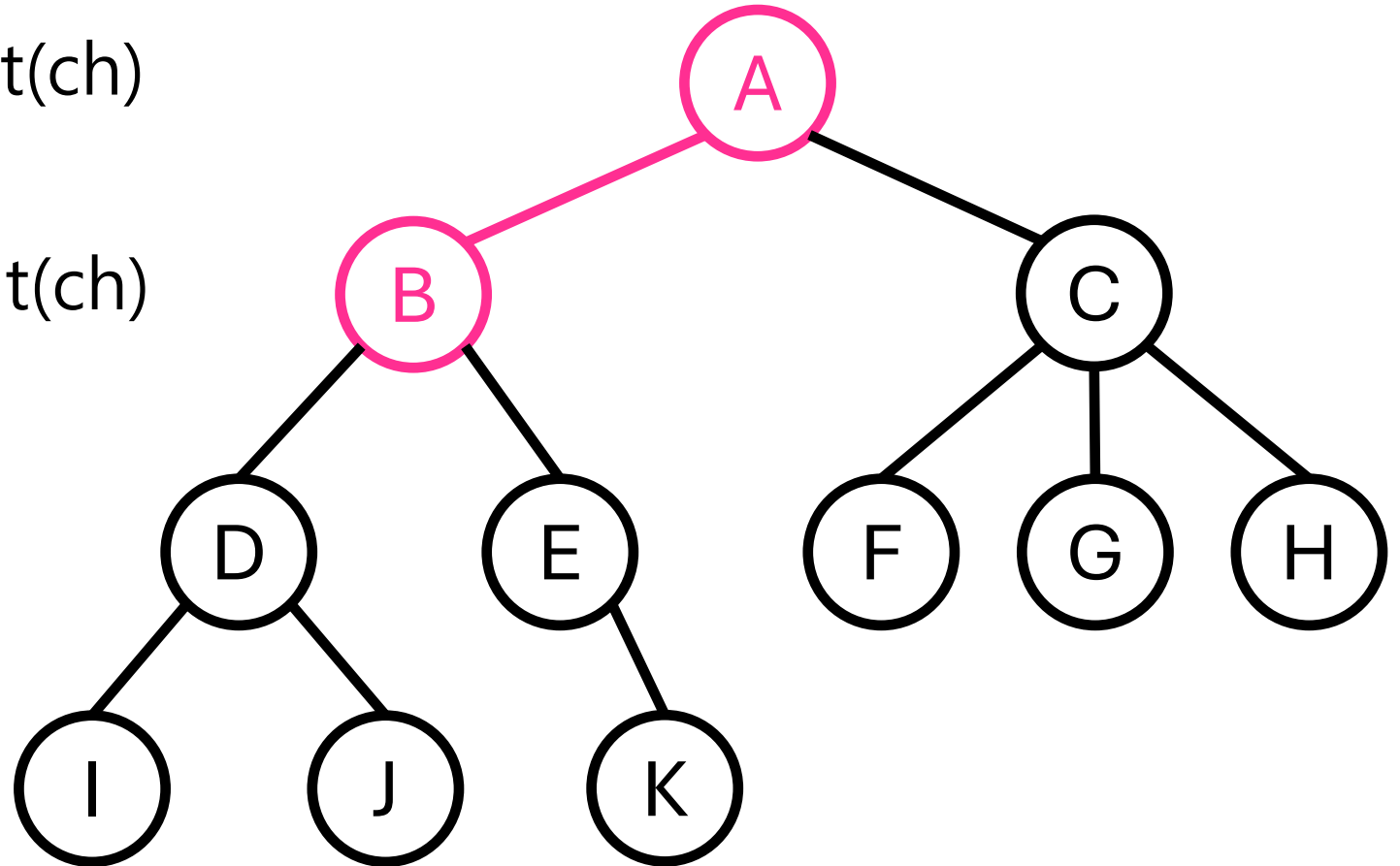
- $\text{height}(D) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 1$
- $\text{height}(B) \rightarrow 1 + \text{height}(\text{ch})$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$





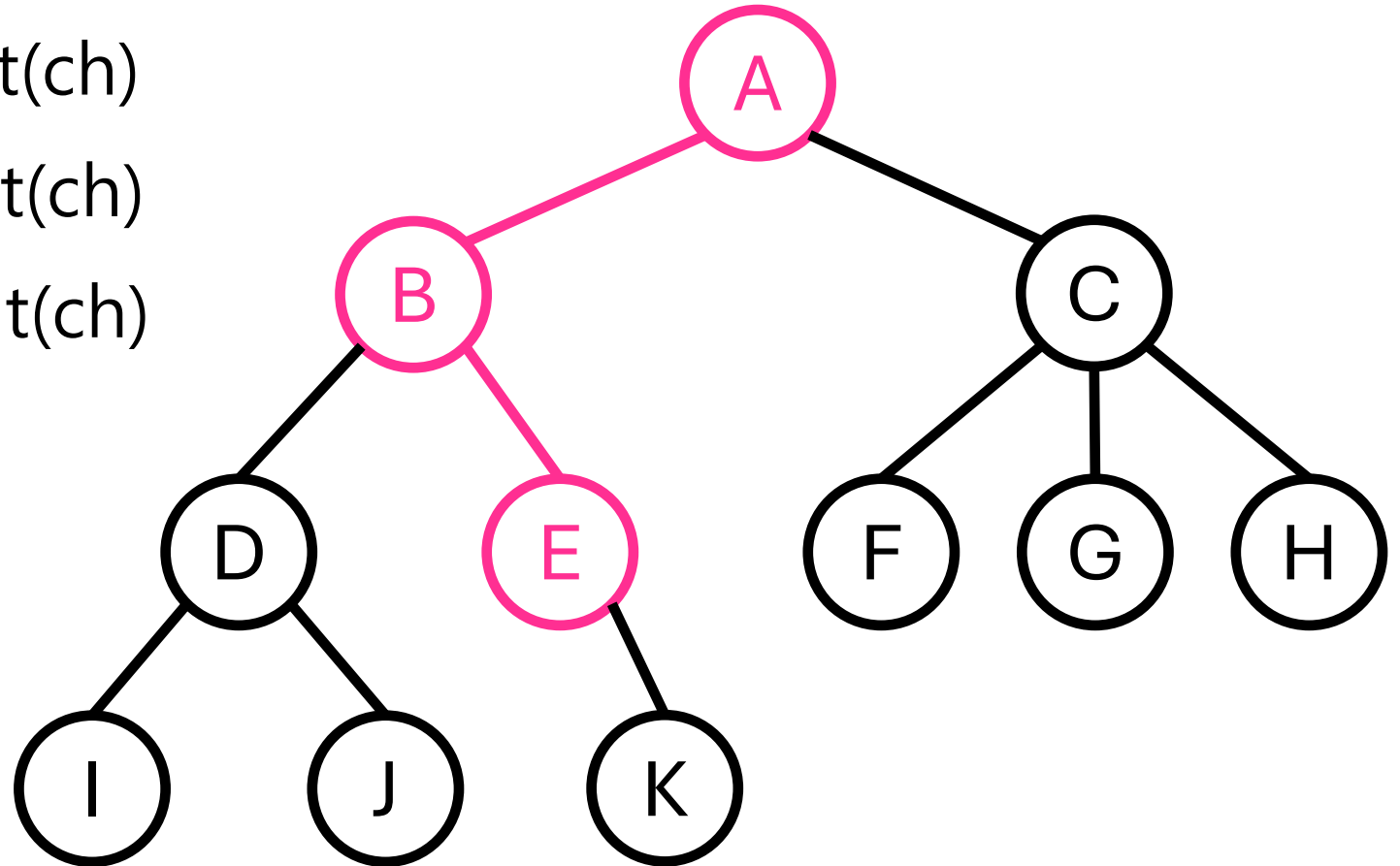
# 개선된 height 함수

- $\text{height}(B) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 2$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$



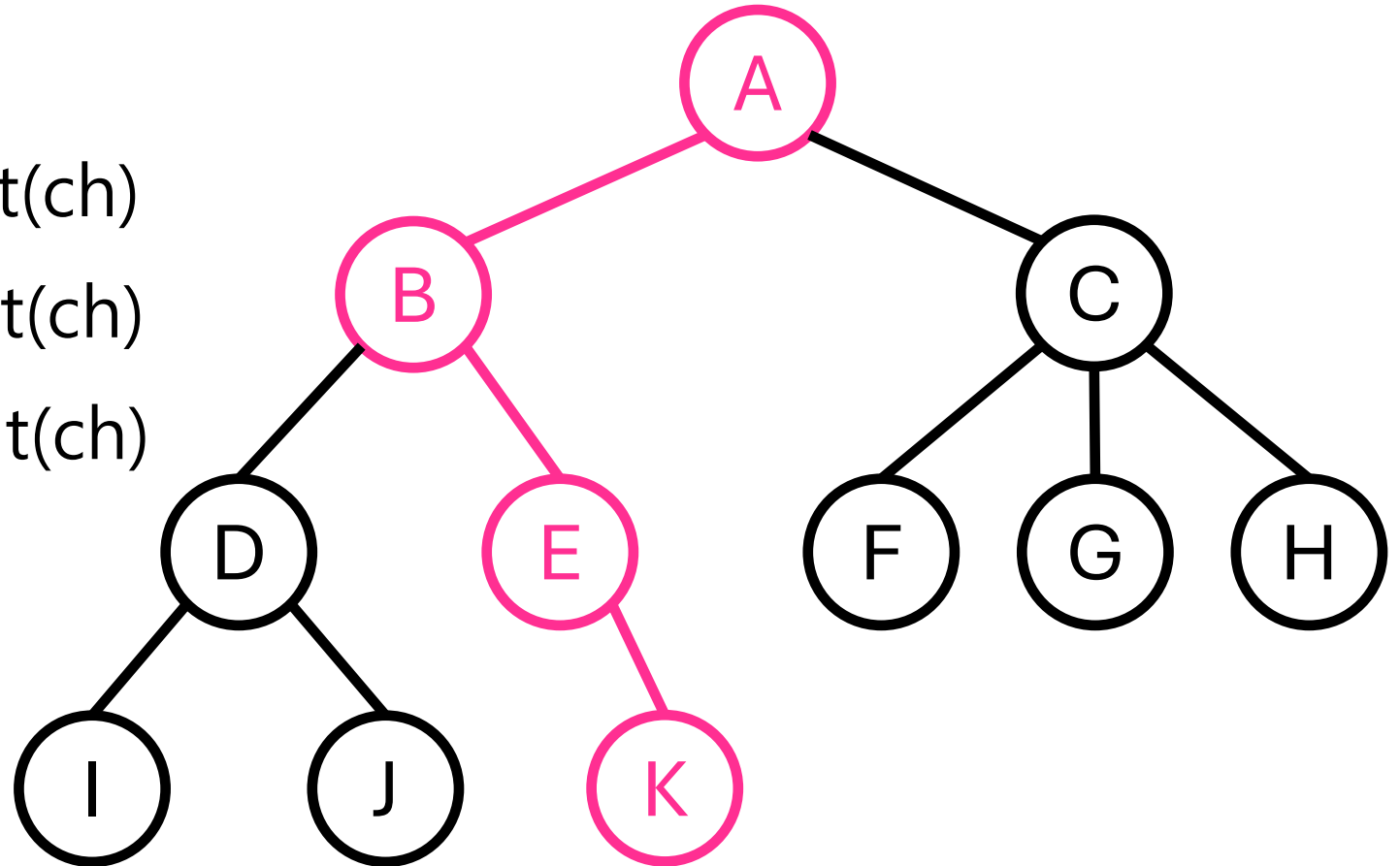
# 개선된 height 함수

- $\text{height}(E) \rightarrow 1 + \text{height}(\text{ch})$
- $\text{height}(B) \rightarrow 1 + \text{height}(\text{ch})$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$



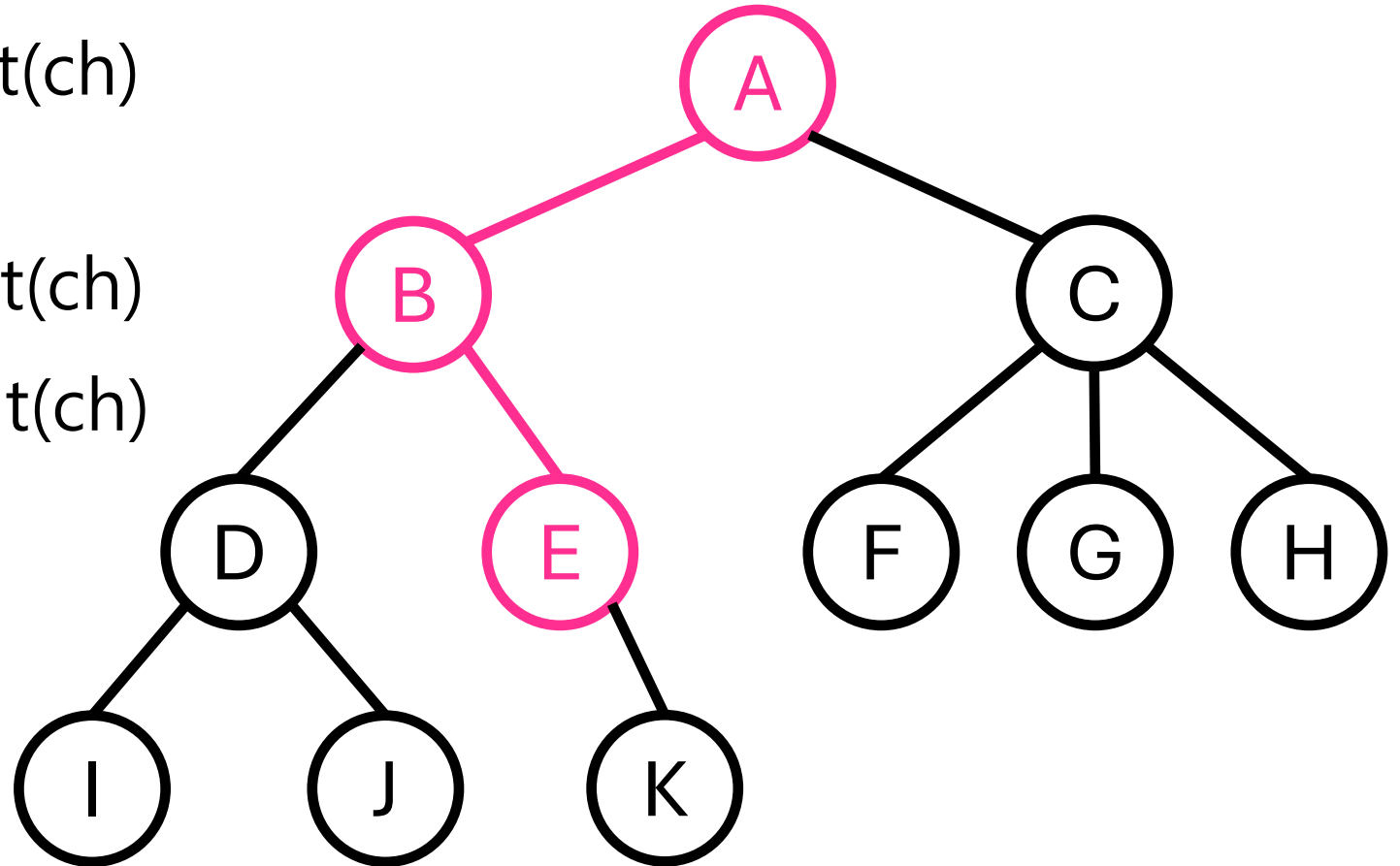
# 개선된 height 함수

- $\text{height}(K) \rightarrow 0$
- $\text{height}(E) \rightarrow 1 + \text{height}(\text{ch})$
- $\text{height}(B) \rightarrow 1 + \text{height}(\text{ch})$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$



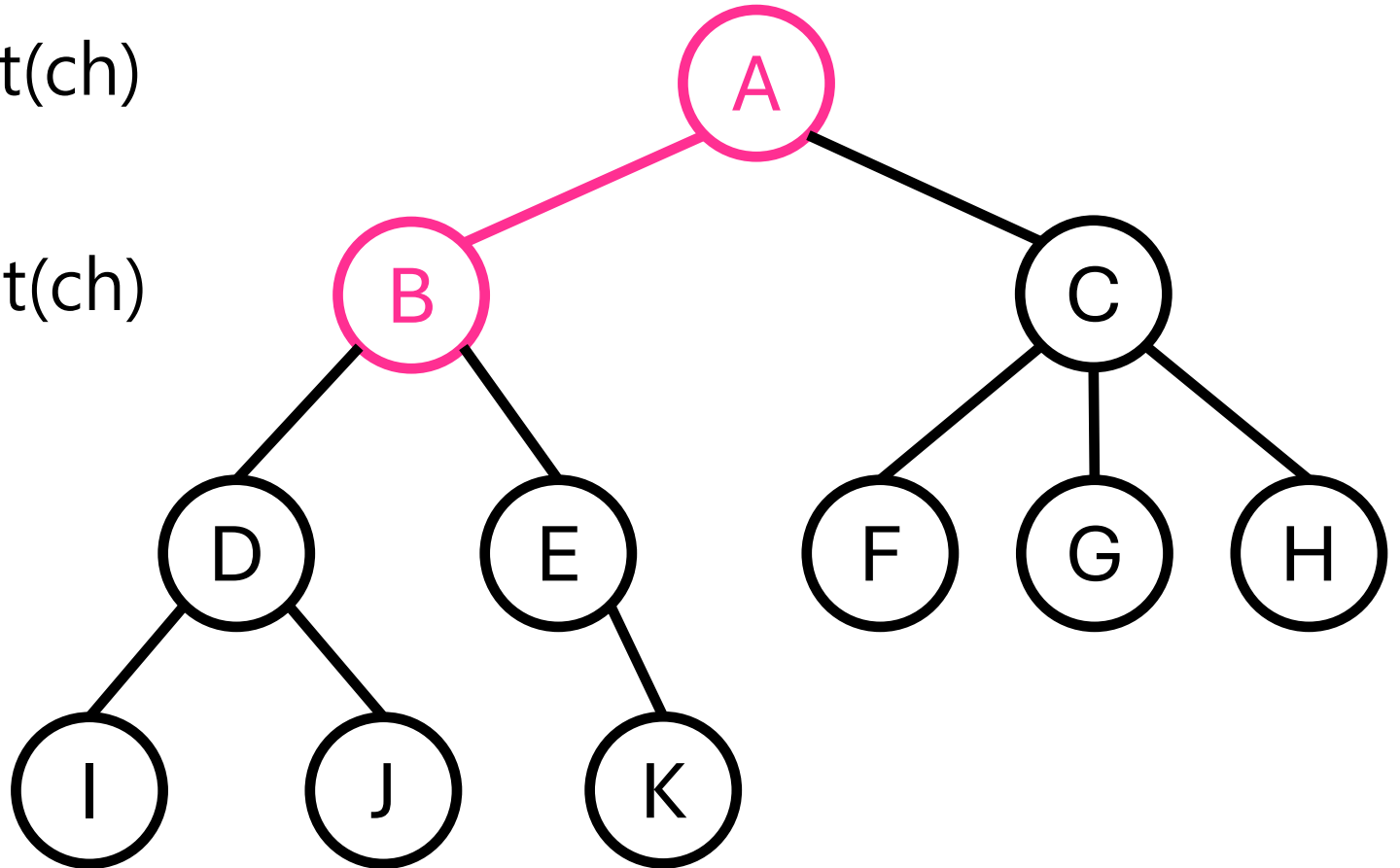
# 개선된 height 함수

- $\text{height}(E) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 1$
- $\text{height}(B) \rightarrow 1 + \text{height}(\text{ch})$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$



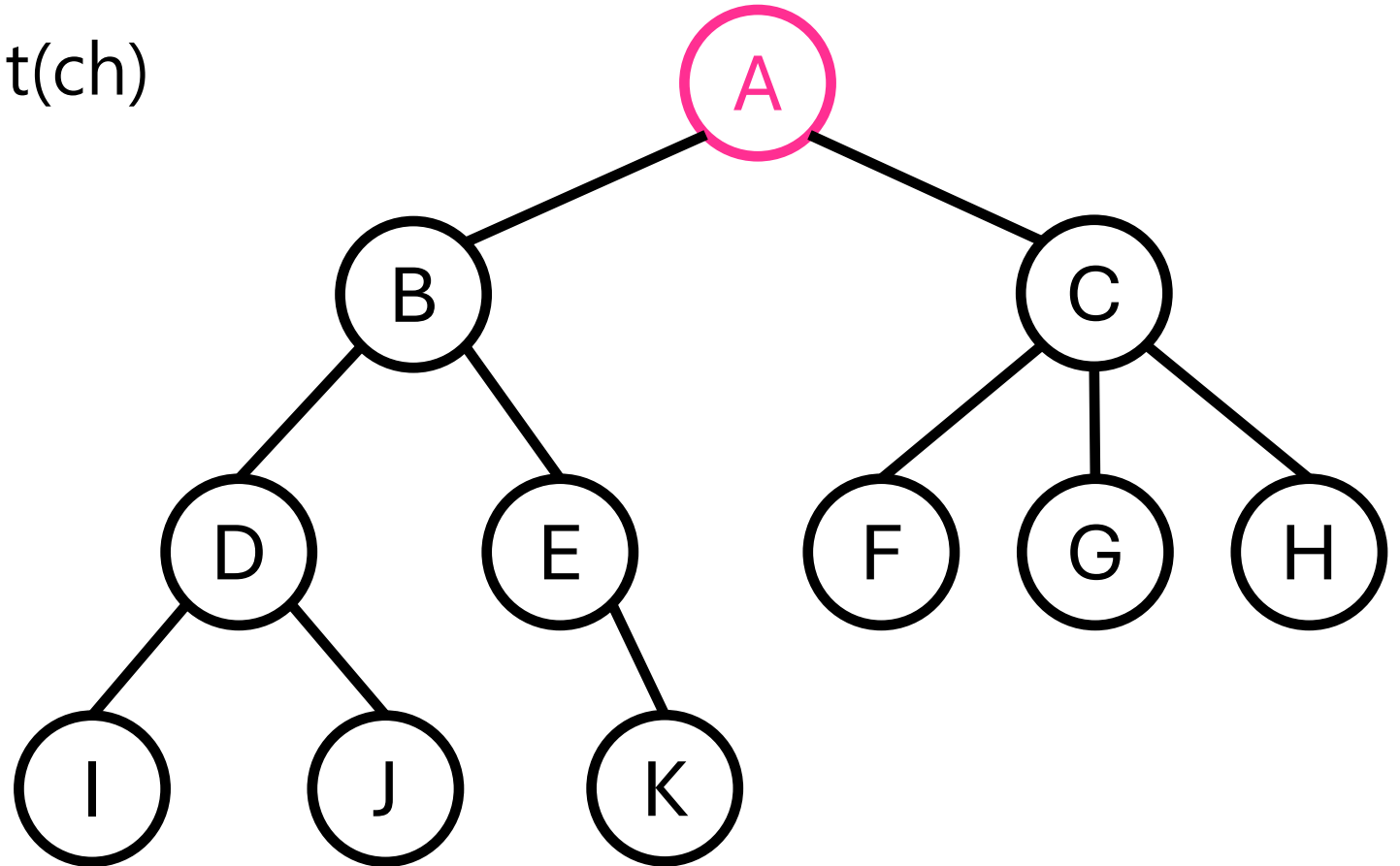
# 개선된 height 함수

- $\text{height}(B) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 2$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$



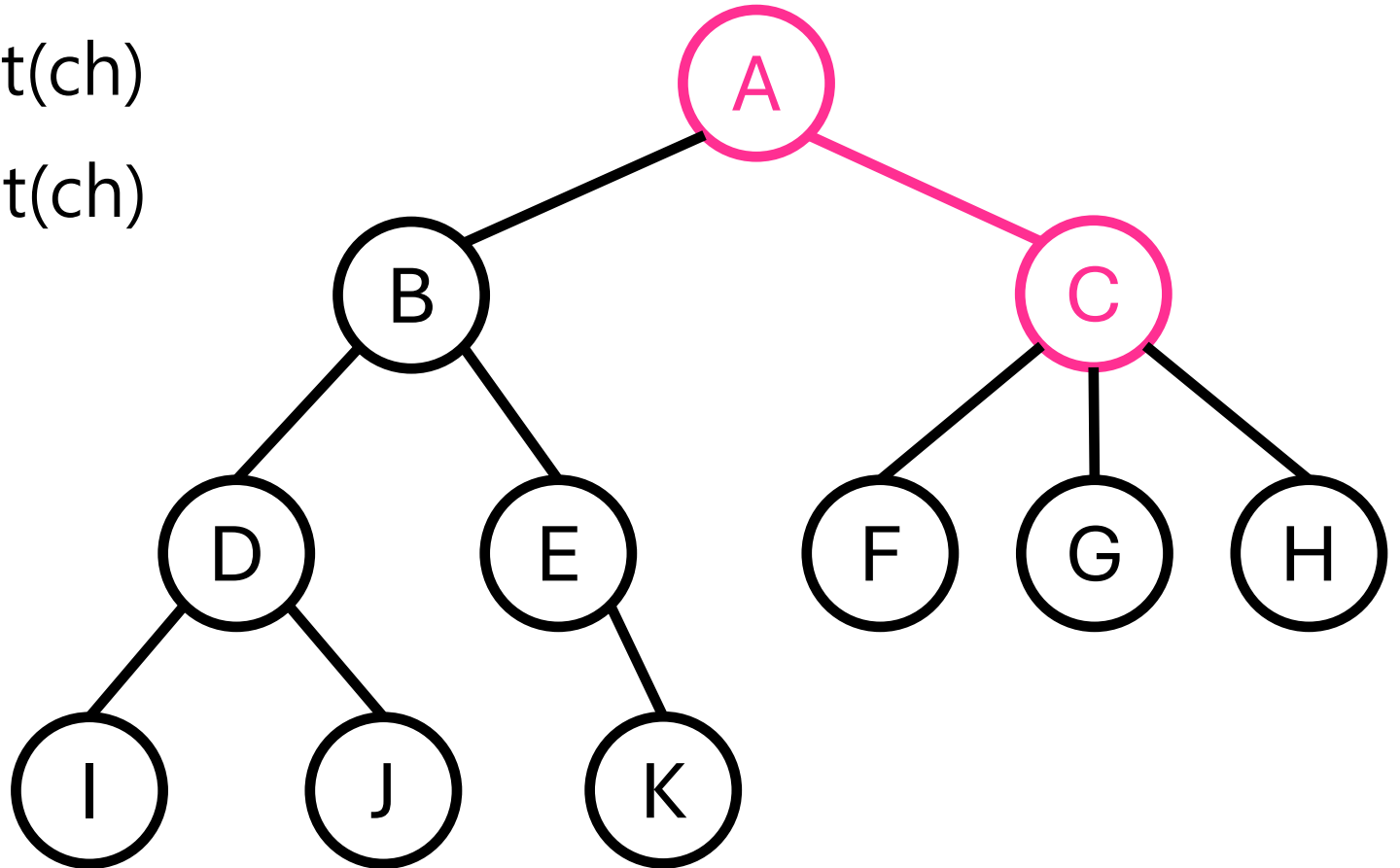
# 개선된 height 함수

- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 3$



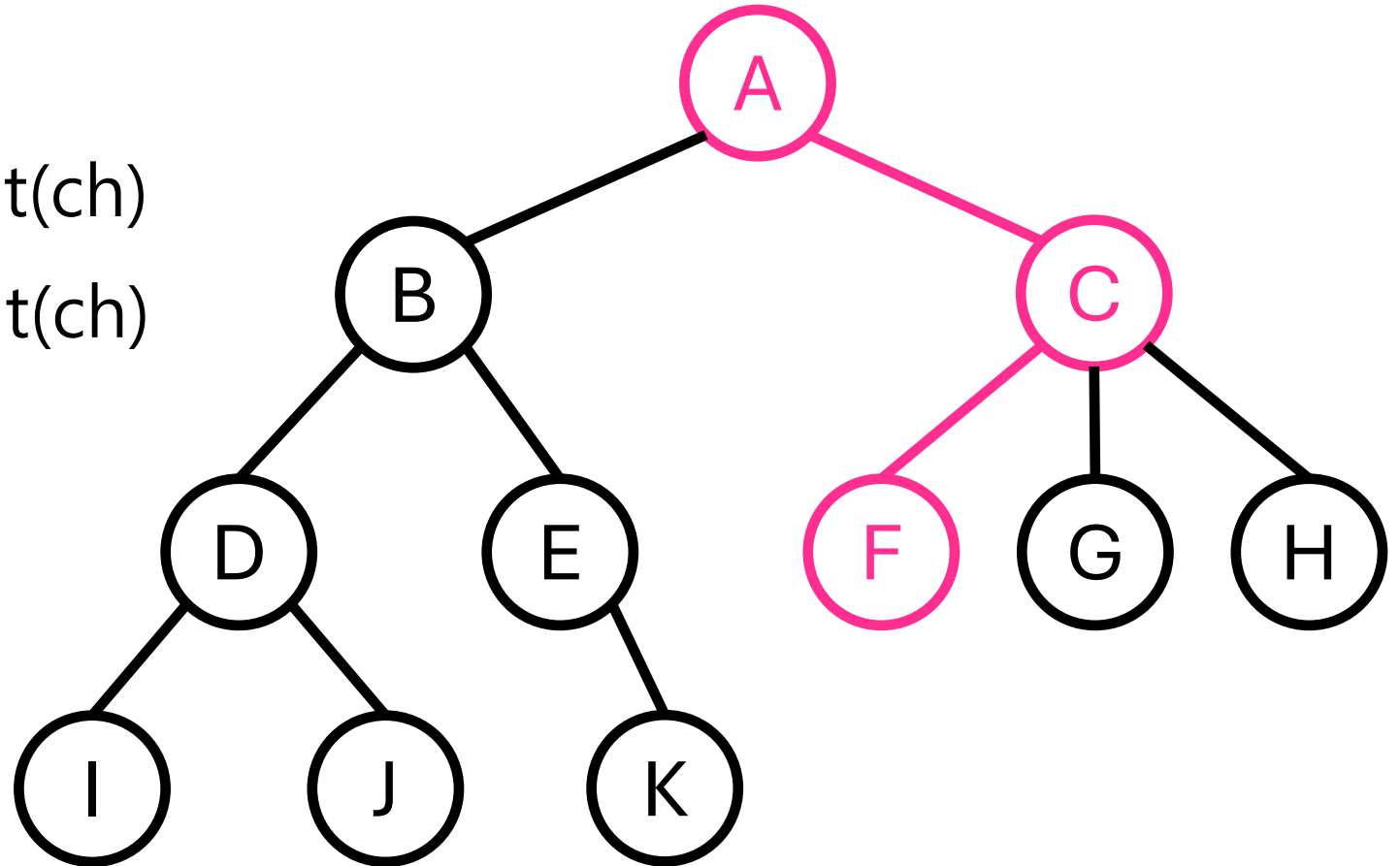
# 개선된 height 함수

- $\text{height}(C) \rightarrow 1 + \text{height}(\text{ch})$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 3$



# 개선된 height 함수

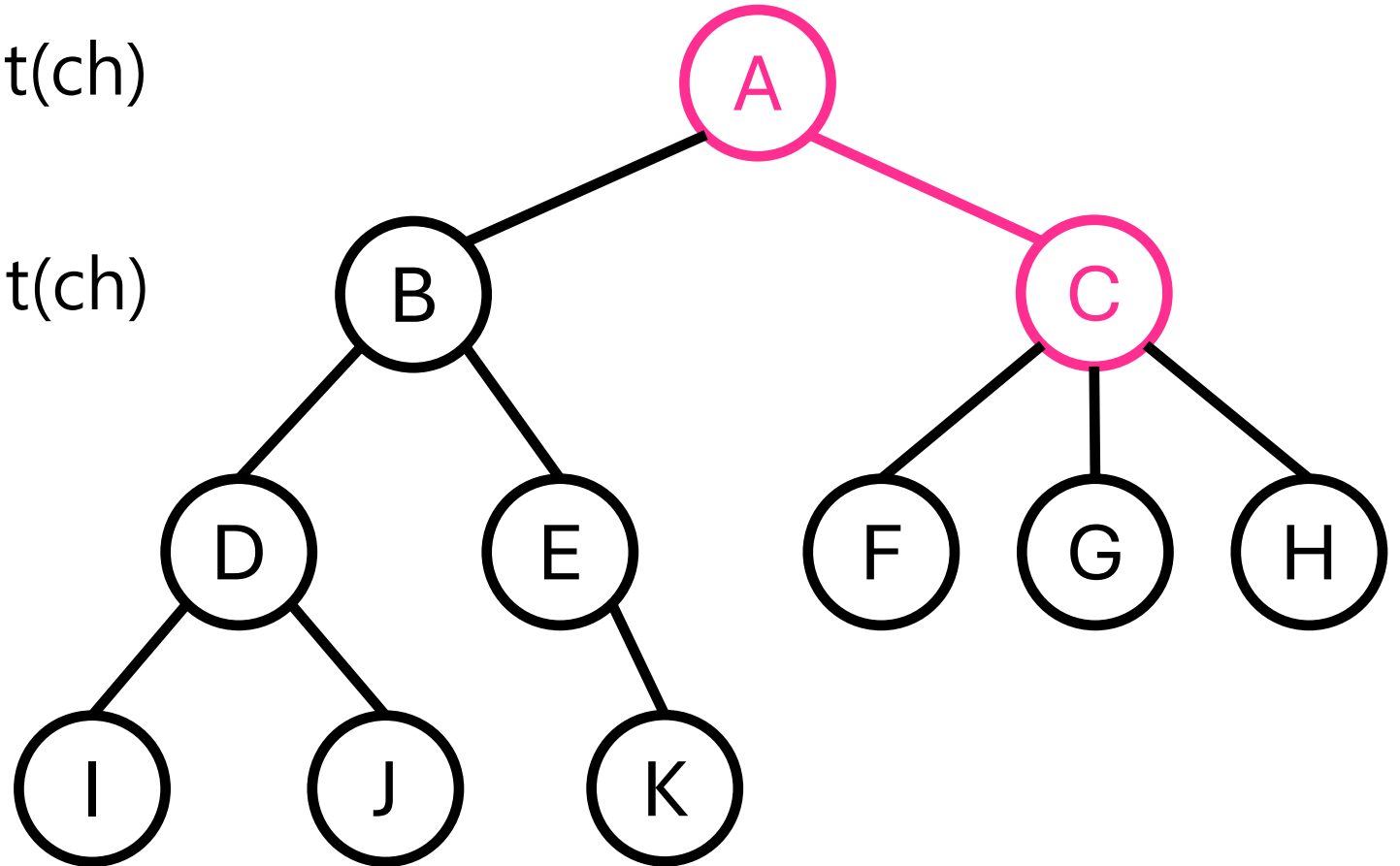
- $\text{height}(F) \rightarrow 0$
- $\text{height}(C) \rightarrow 1 + \text{height}(\text{ch})$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 3$





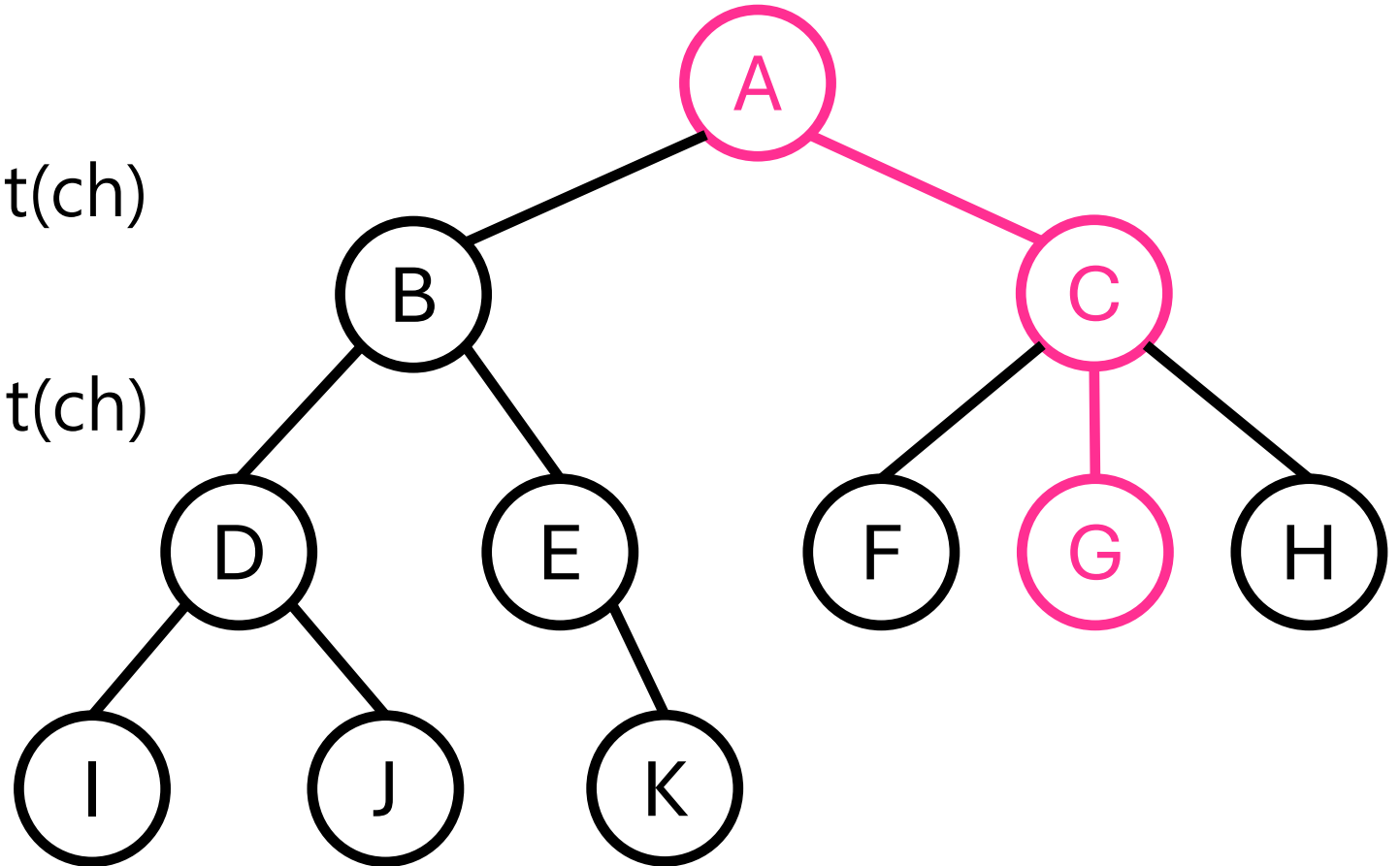
# 개선된 height 함수

- $\text{height}(C) \rightarrow 1 + \text{height}(ch)$ 
  - $h = 1$
- $\text{height}(A) \rightarrow 1 + \text{height}(ch)$ 
  - $h = 3$



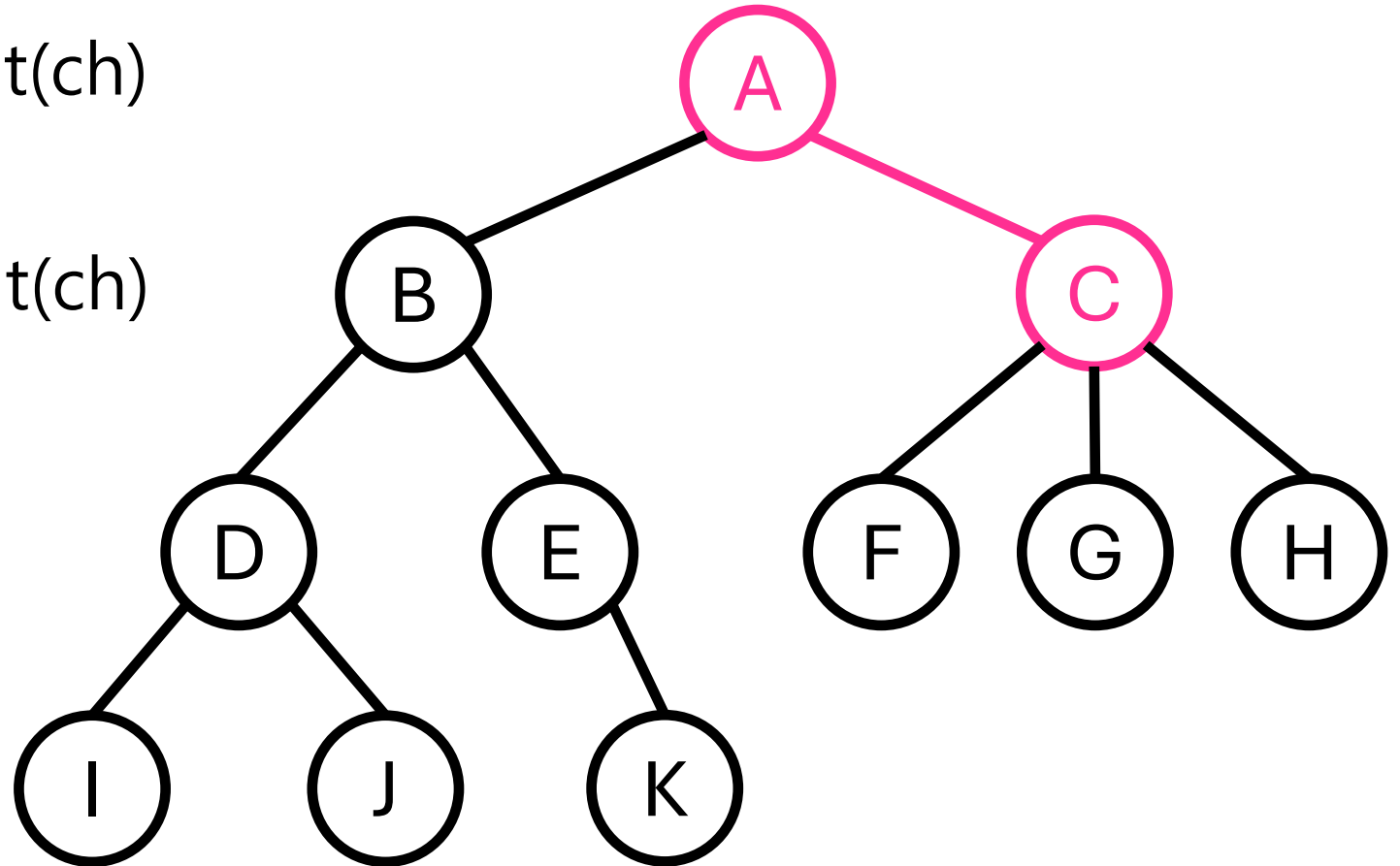
# 개선된 height 함수

- $\text{height}(G) \rightarrow 0$
- $\text{height}(C) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 1$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 3$



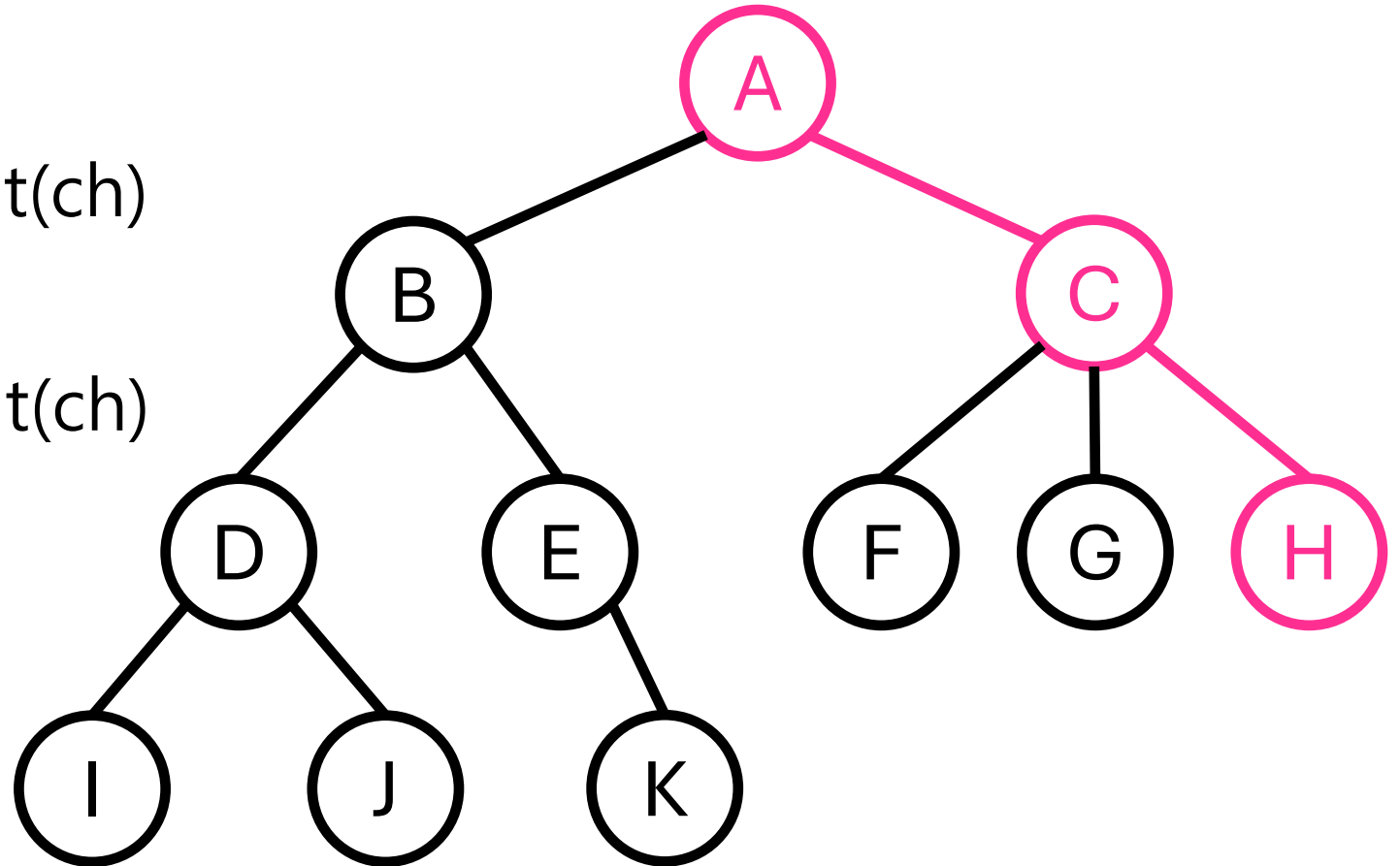
# 개선된 height 함수

- $\text{height}(C) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 1$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 3$



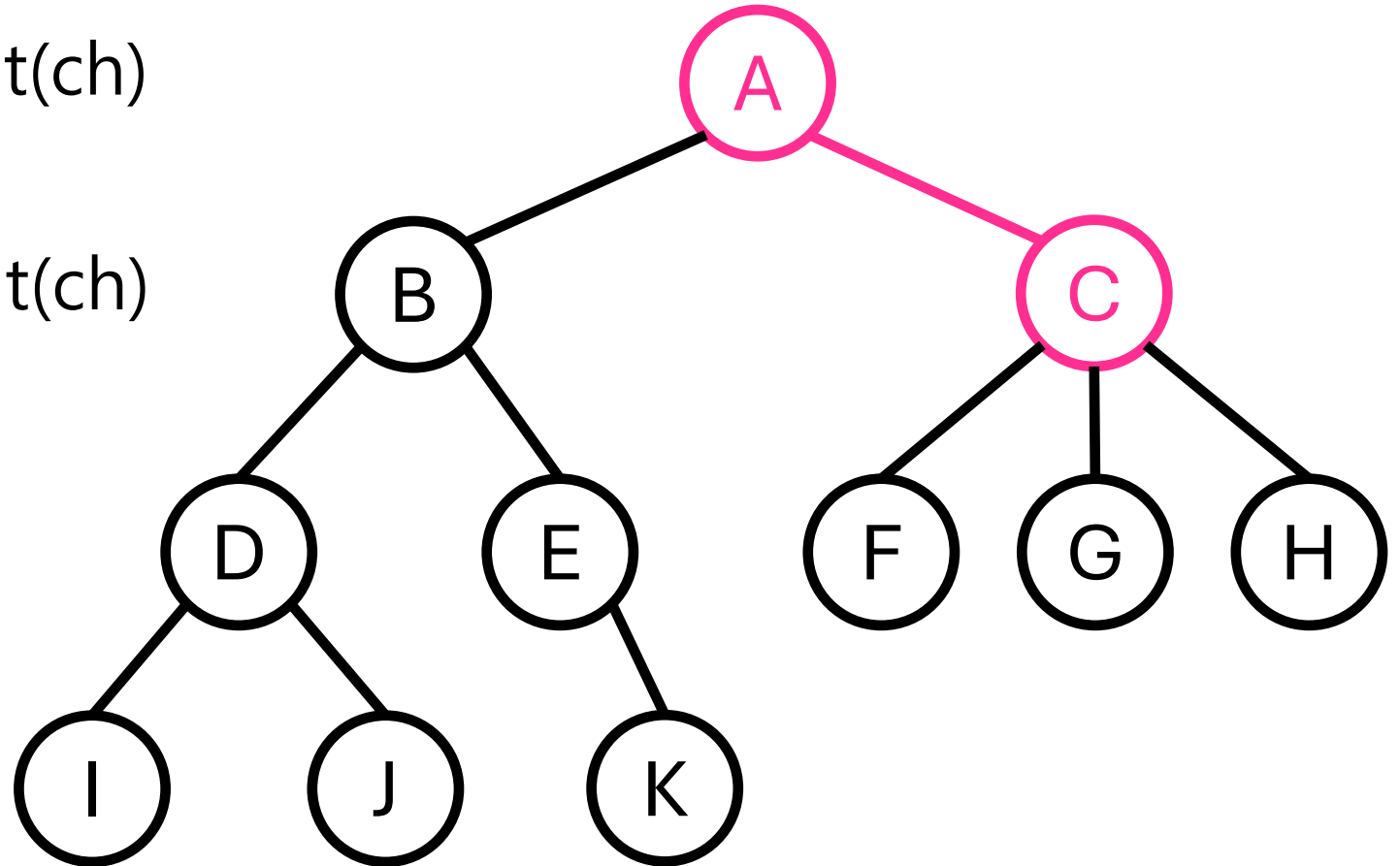
# 개선된 height 함수

- $\text{height}(H) \rightarrow 0$
- $\text{height}(C) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 1$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 3$



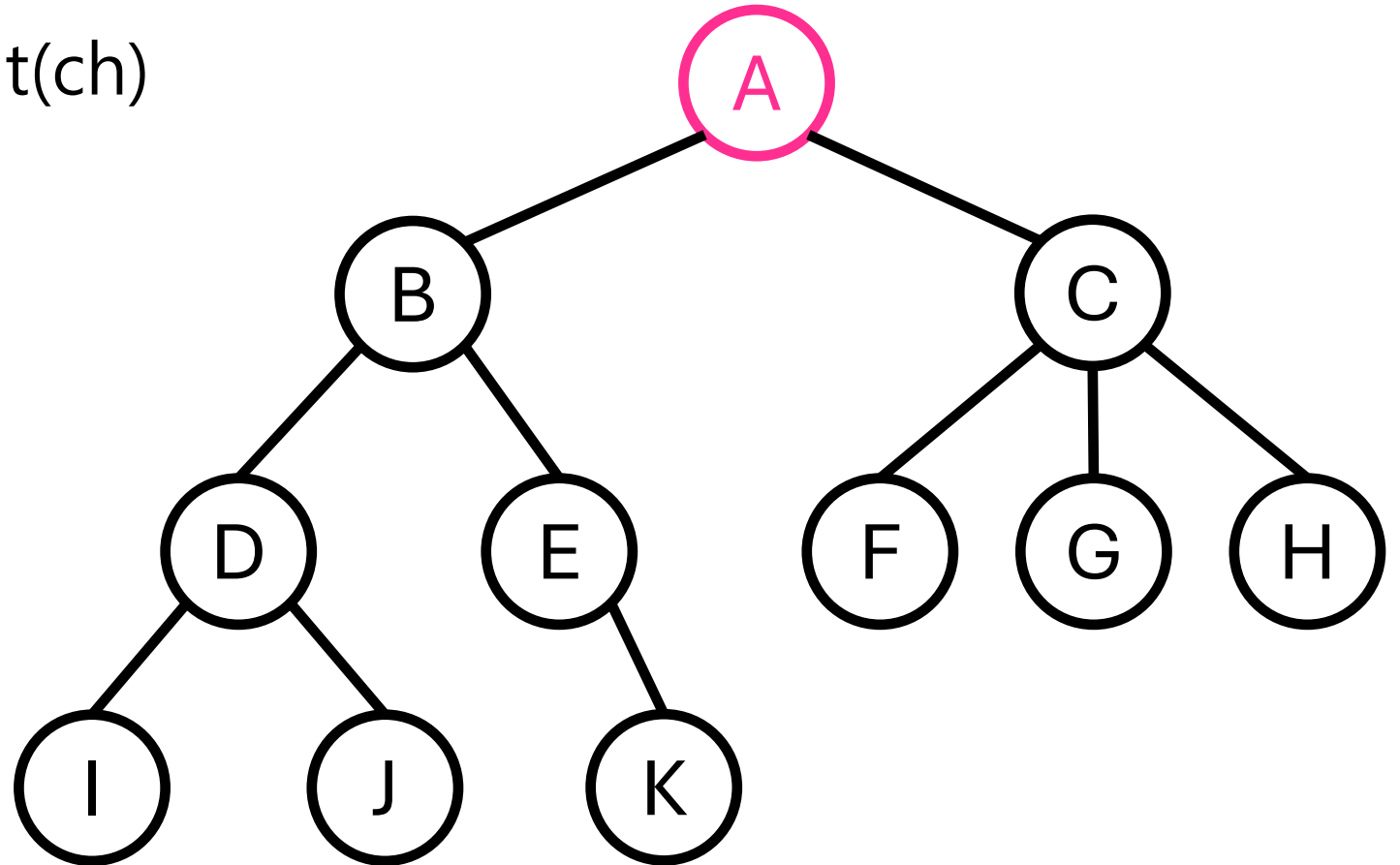
# 개선된 height 함수

- $\text{height}(C) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 1$
- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 3$



# 개선된 height 함수

- $\text{height}(A) \rightarrow 1 + \text{height}(\text{ch})$ 
  - $h = 3$



# height2 함수의 복잡도

- 모든 경우에 대한 분석
  - 단위연산 : height2 함수 호출
  - 입력크기 : 자식 노드의 개수  $c_p$
  - 재귀 함수에서 height2 함수 호출은 자식 노드의 개수 만큼
  - 루트 노드에서 시작해 각 노드의 자식 노드에 대해 함수를 호출하고, 한번 호출된 노드에 대해서는 다시 계산이 이루어지지 않음
  - 즉 모든 노드에 대해 한 번씩만 함수가 호출이 됨
  - $T(n) = n$