BOJ

#15681. 트리와 쿼리

https://www.acmicpc.net/problem/15681

24.12.30

Problem 시간: 1시간 15분

트리와 쿼리 성공 #15681. 트리와 쿼리

☆

골드 V

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	128 MB	17958	8553	6464	45.336%

문제

간선에 가중치와 방향성이 없는 임의의 루트 있는 트리가 주어졌을 때, 아래의 쿼리에 답해보도록 하자.

- 정점 U를 루트로 하는 서브트리에 속한 정점의 수를 출력한다.

만약 이 문제를 해결하는 데에 어려움이 있다면, 하단의 힌트에 첨부한 문서를 참고하자.

입력

트리의 정점의 수 N과 루트의 번호 R, 쿼리의 수 Q가 주어진다. ($2 \le N \le 10^5$, $1 \le R \le N$, $1 \le Q \le 10^5$)

이어 N-1줄에 걸쳐, U V의 형태로 트리에 속한 간선의 정보가 주어진다. ($1 \le U, V \le N$, $U \ne V$)

이는 U와 V를 양 끝점으로 하는 간선이 트리에 속함을 의미한다.

이어 Q줄에 걸쳐, 문제에 설명한 U가 하나씩 주어진다. ($1 \le U \le N$)

입력으로 주어지는 트리는 항상 올바른 트리임이 보장된다.

출력

Q줄에 걸쳐 각 쿼리의 답을 정수 하나로 출력한다.

예제 입력 1 복사

N, R, Q

```
9 5 3
1 3
4 3
5 4
5 6
6 7
2 3
9 6
6 8
5
4
8
```

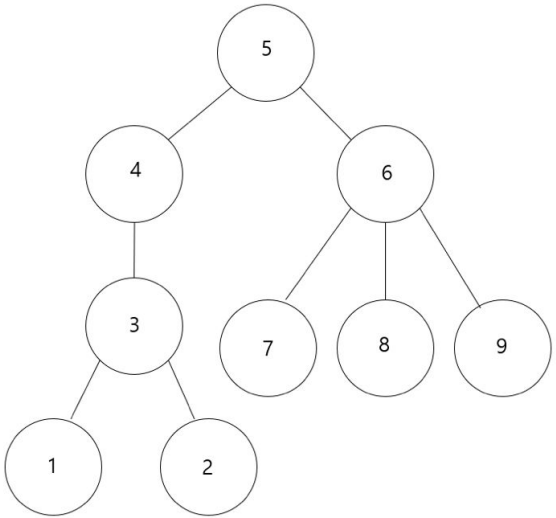
트리

쿼리 U

예제 출력 1 복사

```
9
4
1
```

정점 U를 루트로 하는 서브 트리에 대한 정점의 수

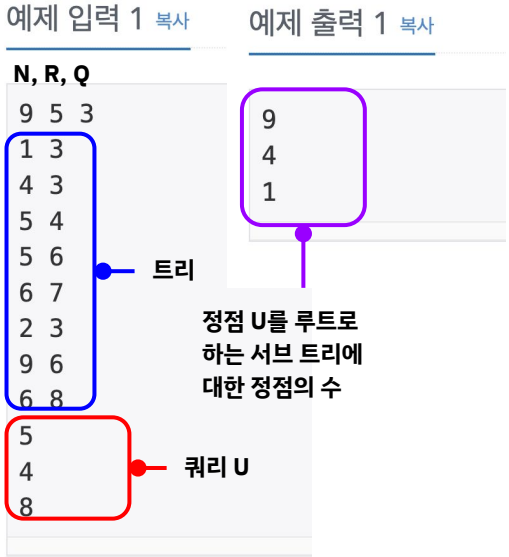


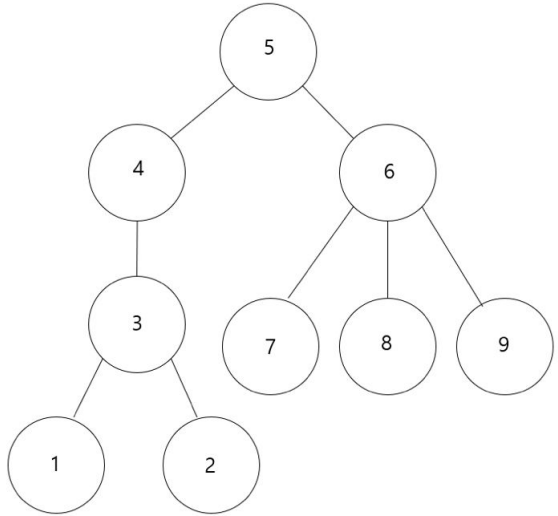
📝 Problem

Step 1. 문제 핵심 요약

> *각 쿼리에서, **특정 정점 U를 루트로 하는 서브트리에 속한 정점의 수**를 구하라.*

- **문제 유형**: 트리, 그래프, DFS
- **목표**:
  - 루트가 있는 트리에서,
    쿼리를 통해 특정 정점 U를 루트로
    하는 서브트리에 속한 정점의
    개수를 구한다.
- **입력**:
  - 정점 수 N, 루트 R, 쿼리 수 Q
  - N-1개의 간선 (U V)
  - Q개의 쿼리 (각각 U)
- **출력**:
  - 각 쿼리 U에 대해, U를 루트로
    하는 서브트리에 속한 정점의 수를
    출력





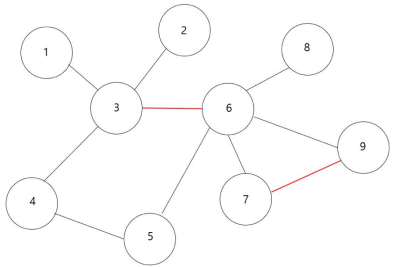
- **5**를 루트로 하는 서브트리: 9개의 정점
- **4**를 루트로 하는 서브트리: 4개의 정점 (4, 3, 1, 2)
- **8**을 루트로 하는 서브트리: 1개의 정점

📝 Problem

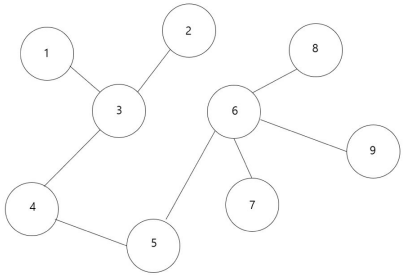
Step 1. 문제 힌트 요약

> 💡 **트리의 특성과 서브트리 이해**

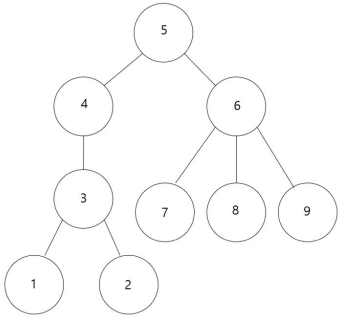
- **그래프:** 정점들과 정점 둘을 잇는 간선들로 이루어진 집합
  - **무향 그래프, 유향 그래프**
- **사이클:** 어떤 정점에서 출발해 시작점을 제외한 어떤 정점도, 어떤 간선도 두 번 이상 방문하지 않고 시작점으로 돌아올 수 있는 경로
- **트리:** 그래프에 **단 하나의 사이클이 없는 그래프**
  - 임의의 두 정점 U와 V에 대해, **U에서 V로 가는 최단경로는 유일**하다.
  - 아무 정점이나 잡고 부모와의 연결을 끊었을 때, 해당 정점과 그 자식들, 그 자식들의 자식들 … 로 이루어진 부분그래프는 트리가 된다. 이를 '**서브트리**'라고 부른다.
  - **서브트리**: **특정 정점 U와 그 자식 노드들로 구성된 부분 트리**



그래프



트리



5번 정점이
루트일 때 트리

💡 Hint

Step 1. 들어가기 앞서

🚀 1. 트리와 그래프

- 트리는 **사이클이 없는 연결 그래프**이다.
- 루트에서 출발해 모든 정점을 탐색하는 **DFS/BFS** 방식이 주로 사용된다.

🚀 2. 재귀 깊이 설정

- 트리 문제에서 **DFS**는 재귀로 구현되는 경우가 많다.
- 트리의 최대 정점 개수가 100,000개까지 주어질 수 있으므로, **재귀 깊이 제한**을 늘려야 한다.

```
import sys
sys.setrecursionlimit(10**6)  # 재귀 깊이 제한 설정(10만)
```

💡 Hint

Step 2. Tree Dynamic Programming

트리 DP는 **트리 구조**에서 **동적 계획법(DP)**을 사용하는 기법이다.

- **트리에서 DP를 구현한다는 것은,**
  - 예를 들면 특정한 **i번째 노드를 루트로 하는 서브 트리**에 대해서 **i번째 루트 노드를 포함 했을 때와 포함하지 않았을 때** 중 조건에 맞는 답을 정의하는 것이다.
  - 이 문제에서는 트리의 각 노드에 대해 **“서브트리 크기”** 값을 DP처럼 저장한다.
  - 핵심은, 각 노드를 탐색하며 자식 노드에서 계산된 값을 **"부모 노드로 올려 보내는"** 방식입니다.
- **접근법**
  - **DFS + DP**
  - 루트 노드에서 시작하여 트리의 자식 노드들을 탐색한다.
  - 각 노드의 서브트리 크기를 계산한 후, 그 값을 부모 노드로 전달한다.
- 이 접근법은 모든 노드를 정확히 한 번만 방문하므로 시간 복잡도는 **O(N)**이다.

Hint

Step 3. 문제에서 주어진 힌트

현재 정점을 루트로 하는 서브트리에 속한 정점의 수를 계산하는 함수

```
def countSubtreeNodes(currentNode) :
    size[currentNode] = 1 // 자신도 자신을 루트로 하는 서브트리에 포함되므로 0이 아닌 1에서
시작한다.
    for Node in currentNode’s child:
        countSubtreeNode(Node)
         size[currentNode] += size[Node]
```

💡 Hint

Step 4. 구현 힌트

> *** size 리스트가 DP 라고 이해하면 된다.**

- **트리 구성**
  - 입력으로 주어진 간선 정보를 활용하여 트리를 리스트 형태로 저장한다.
  - 무향 그래프이므로 각 간선을 양방향으로 저장한다.
- **DFS 탐색을 활용한 서브트리 크기 계산**
  1. 루트 노드(R)부터 시작해 DFS를 수행한다.
  2. 방문한 노드는 배열 size[]에 서브트리 크기를 저장한다.
  3. 자식 노드에서 반환된 값을 부모 노드의 서브트리 크기에 더한다.
- **쿼리 처리**
  - 서브트리 크기는 size[]에 미리 저장되므로, 각 쿼리에 대해 O(1) 시간에 응답할 수 있다.

Hint

Step 5. 수도 코드

1. 트리 그래프 생성:

1. N+1개의 빈 리스트를 갖는 graph 배열을 만든다.
    - 각 정점(노드)마다 연결된 노드를 저장할 리스트이다.
2. 모든 간선 (u, v)를 반복하며:
    - u와 v를 서로 연결한다.
    - 이는 무향 그래프이므로 양방향으로 간선을 저장해야 한다.

2. DFS(깊이 우선 탐색) 함수 정의:

1. 현재 노드 current를 기준으로 탐색을 시작한다.
2. size[current]에 1을 저장한다.
3. 현재 노드에 연결된 모든 노드를 순회하며:
    - 만약 해당 노드가 아직 방문되지 않았다면:
        - 해당 노드로 이동해 재귀적으로 서브트리 크기를 계산한다.
        - 이후, 자식 노드의 서브트리 크기를 size[current]에 더한다.
        - 즉, 자식 노드에서 반환된 서브트리 크기를 부모 노드의 서브트리 크기에 합산한다.

3. 쿼리 처리:

1. 쿼리에서 요청한 노드 U에 대해 size[U]를 출력한다.

✨Solution

```python
import sys
input = sys.stdin.readline
sys.setrecursionlimit(10**6)  # 재귀 깊이 제한 설정(10만)

# 입력 처리
N, R, Q = map(int, input().split())  # 정점 수, 루트 번호, 쿼리 수
graph = [[] for _ in range(N + 1)]  # 트리 그래프

# 간선 정보 입력
for _ in range(N - 1):
    u, v = map(int, input().split())
    graph[u].append(v)
    graph[v].append(u)

# 서브트리 크기 기록 배열
size = [0] * (N + 1)

# DFS를 이용한 서브트리 크기 계산
def countSubtreeNodes(current):
    size[current] = 1  # 자기 자신 포함
    for node in graph[current]:
        if size[node] == 0:  # 아직 방문하지 않은 경우
            countSubtreeNodes(node)
            size[current] += size[node]  # 자식 서브트리 크기 추가

# 루트에서 시작해 서브트리 크기 계산
countSubtreeNodes(R)

# 쿼리 처리
for _ in range(Q):
    U = int(input())
    print(size[U])
```

https://jainn.tistory.com/74

시간 복잡도

- DFS 탐색: O(N) – 트리의 모든 정점을 방문
- 쿼리 처리: O(Q) – 각 쿼리당 O(1)
- 총 시간 복잡도: O(N + Q)

💻 Assignment

백준 #1949. 우수 마을 (골드2)

Tree Dynamic Programing

발제 문제에서 사용한 개념을 응용한 문제

- N개의 마을로 이루어진 **트리** 구조에서 **'우수 마을'**을 선정해 **주민 수의 총합을 최대화**해야 한다.
- 우수 마을은 서로 인접할 수 없으며, 선정되지 않은 마을은 최소 하나의 우수 마을과 인접해야 한다.

💻 Assignment

백준 #1949. 우수 마을 (골드2)

동작 방식과 힌트 안내

1. **동작 방식**
   - 예제에 대한 동작 방식을 슬라이드로 만들었습니다. 가장 마지막 슬라이드를 참조하면 됩니다.
2. **힌트**
   - 문제 풀이 중 힌트가 필요할 경우 힌트 페이지를 참고해주시면 됩니다.

💻 Assignment

백준 #1949. 우수 마을 (골드2)

힌트 1단계: 문제의 핵심 조건 이해하기

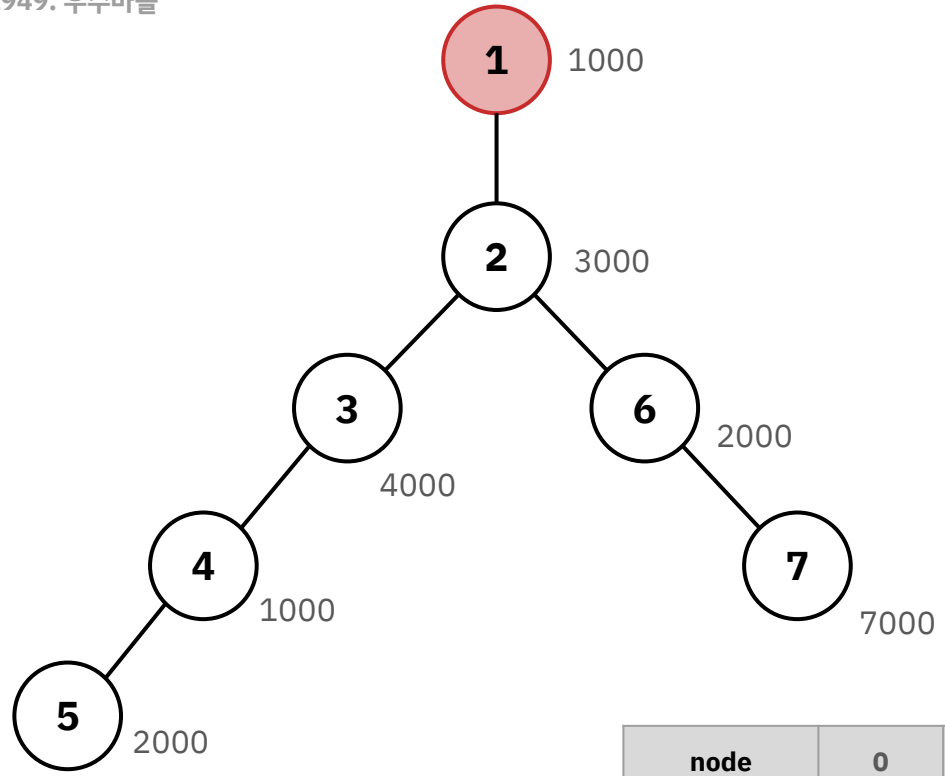
- **서로 인접한 마을이 동시에 우수 마을이 될 수 없다.**
  - 트리에서 두 노드가 동시에 선택될 수 없는 것은, 부모-자식 관계에 있는 두 노드가 함께 선택될 수 없다는 의미이다.
  - 즉, 특정 마을을 '우수 마을'로 선정하려면, 자식 노드는 우수 마을에서 제외해야 한다.
- **선정되지 않은 마을은 적어도 하나의 인접 마을이 우수 마을이어야 한다.**
  - 이는 부모 마을이 '우수 마을'이 아니라면 자식 중 최소 하나는 우수 마을이어야 한다는 조건이다.
- **'우수 마을'의 주민 수 합을 최대화하는 것이 목표이다.**
  - 각 마을의 주민 수가 다르기 때문에, 더 많은 주민이 있는 마을을 우선적으로 선택하는 방식으로 접근해야 한다.

💻 Assignment

백준 #1949. 우수 마을 (골드2)

힌트 2단계: 트리 DP의 필요성 파악

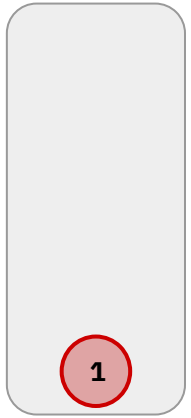
1. **트리 구조에서 DP가 필요한 이유**
   - 각 마을(노드)에 대해 두 가지 경우를 고려해야 한다:
     - 해당 마을이 '우수 마을'로 **선정된 경우**
     - 해당 마을이 '우수 마을'에서 **제외된 경우**
   - 각 경우에 대해 주민 수의 최댓값을 기록해야 합니다.
2. **DP 테이블 정의**
   - dp[i][0]: i번 마을이 우수 마을이 **아닌 경우** 최대 주민 수
   - dp[i][1]: i번 마을이 우수 마을로 **선정된 경우** 최대 주민 수



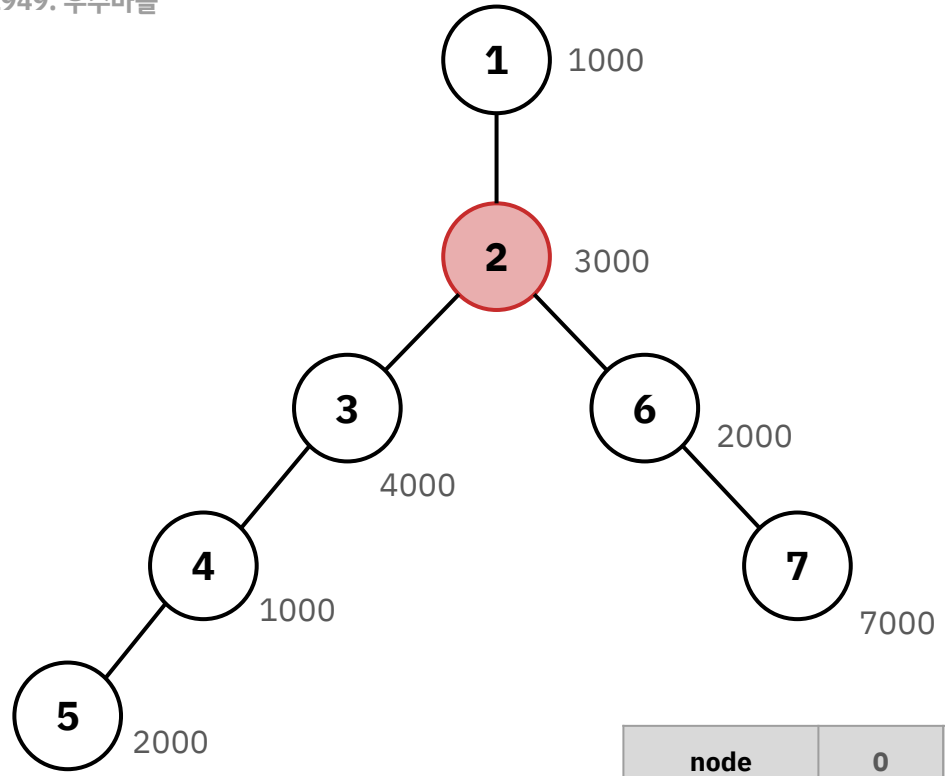
현재 방문 노드(current)	1
자식 노드 (child)	2
처음 방문할 경우 ➡ 우수마을에 지정될 경우에 대해 업데이트 dp[current][1] += people[current]	1000

➡️ 자식노드 [2] 탐색

DFS



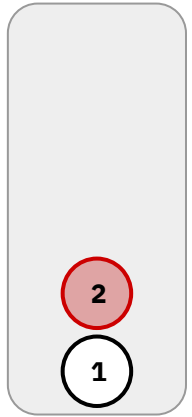
node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	0	0	0	0	0	0	0
dp[node][1] 우수마을O	0	1000	0	0	0	0	0	0



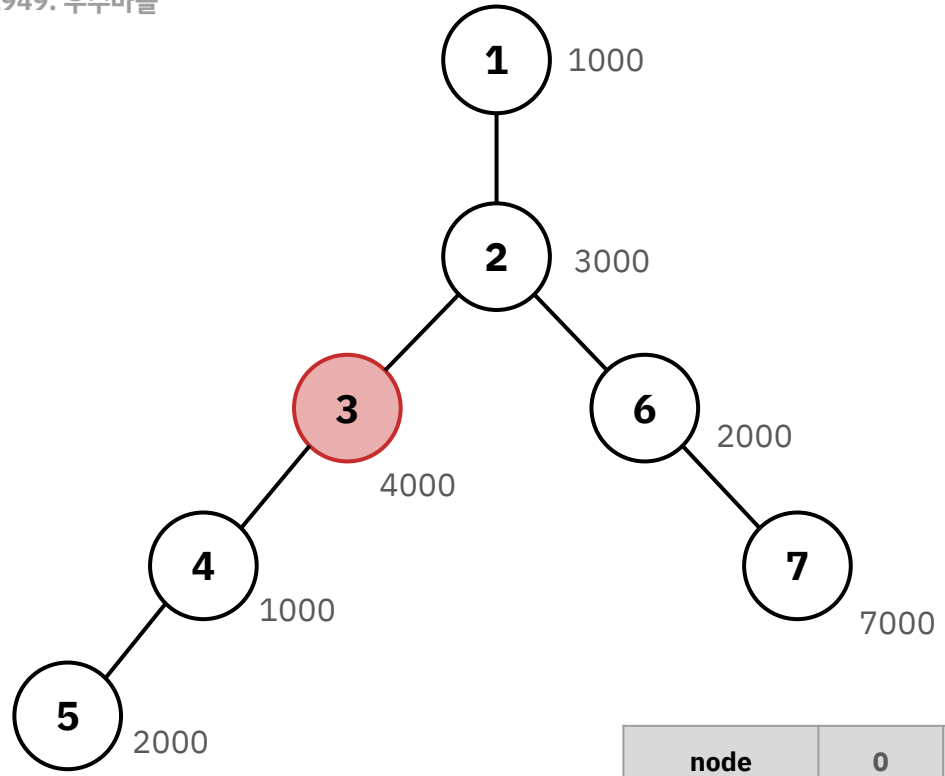
현재 방문 노드(current)	2
자식 노드 (child)	1,3,6
처음 방문할 경우 ➡ 우수마을에 지정될 경우에 대해 업데이트 dp[current][1] += people[current]	3000

➡️ 자식노드 [3] 탐색

DFS



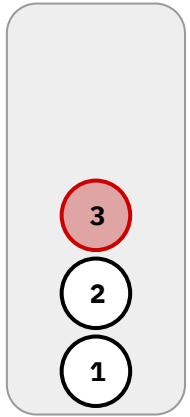
node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	0	0	0	0	0	0	0
dp[node][1] 우수마을O	0	1000	**3000**	0	0	0	0	0



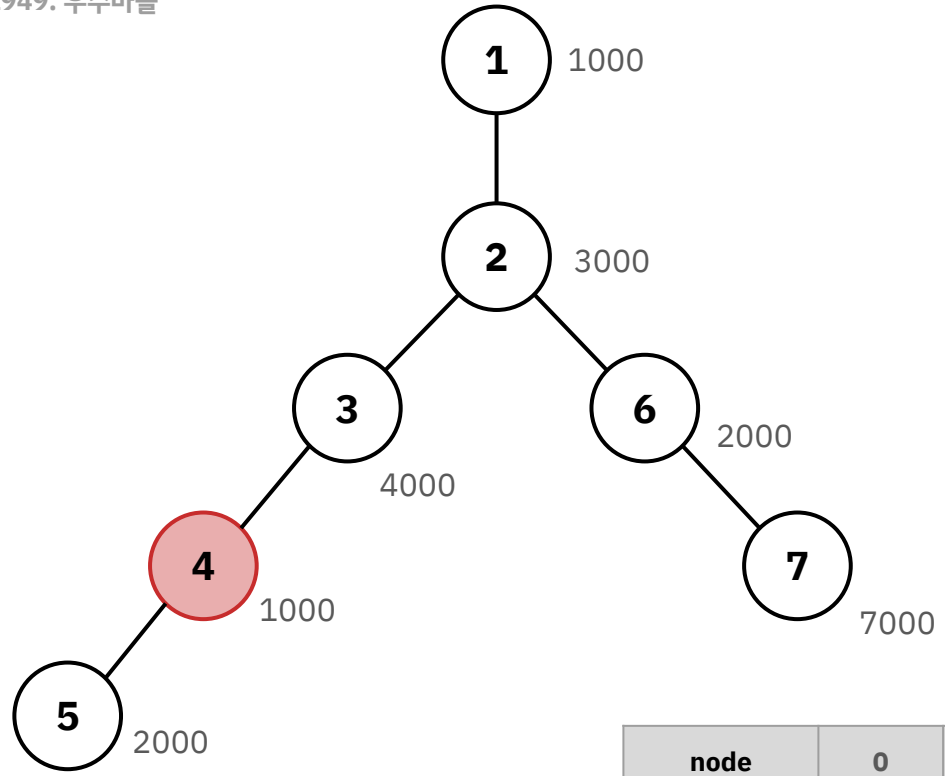
현재 방문 노드(current)	3
자식 노드 (child)	4
처음 방문할 경우 ➡ 우수마을에 지정될 경우에 대해 업데이트 dp[current][1] += people[current]	4000

➡️ 자식노드 [4] 탐색

DFS



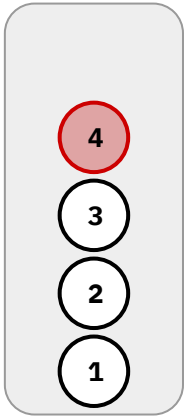
node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	0	0	0	0	0	0	0
dp[node][1] 우수마을O	0	1000	3000	**4000**	0	0	0	0



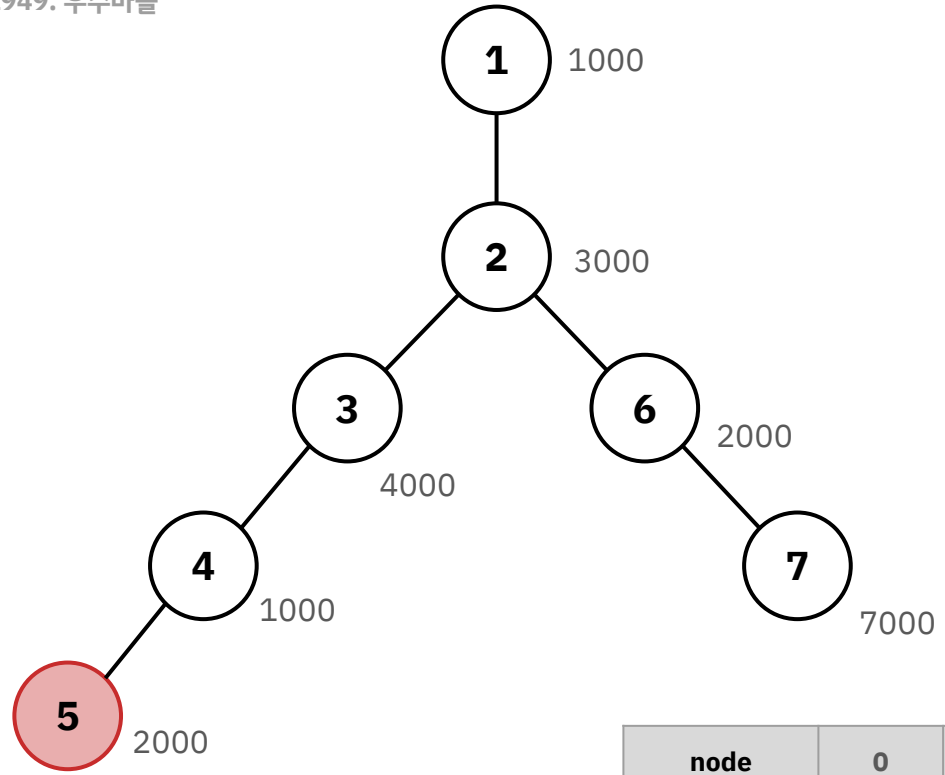
현재 방문 노드(current)	4
자식 노드 (child)	5
처음 방문할 경우 ➡ 우수마을에 지정될 경우에 대해 업데이트 dp[current][1] += people[current]	1000

➡️ 자식노드 [5] 탐색

DFS

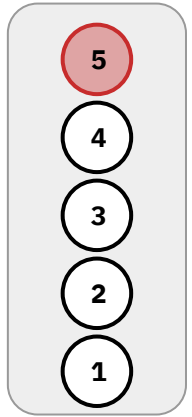


node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	0	0	0	0	0	0	0
dp[node][1] 우수마을O	0	1000	3000	4000	**1000**	0	0	0

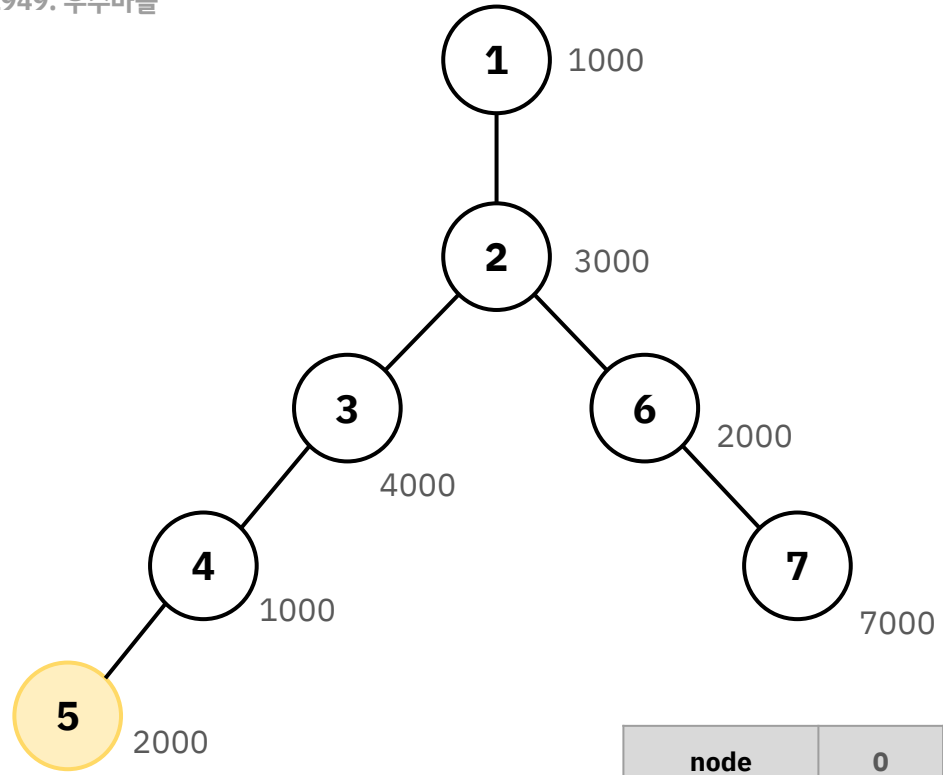


현재 방문 노드(current)	5
자식 노드 (child)	X
처음 방문할 경우 ➡ 우수마을에 지정될 경우에 대해 업데이트 dp[current][1] += people[current]	2000

DFS

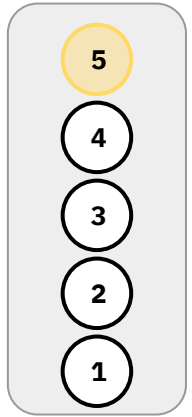


node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	0	0	0	0	0	0	0
dp[node][1] 우수마을O	0	1000	3000	4000	1000	**2000**	0	0

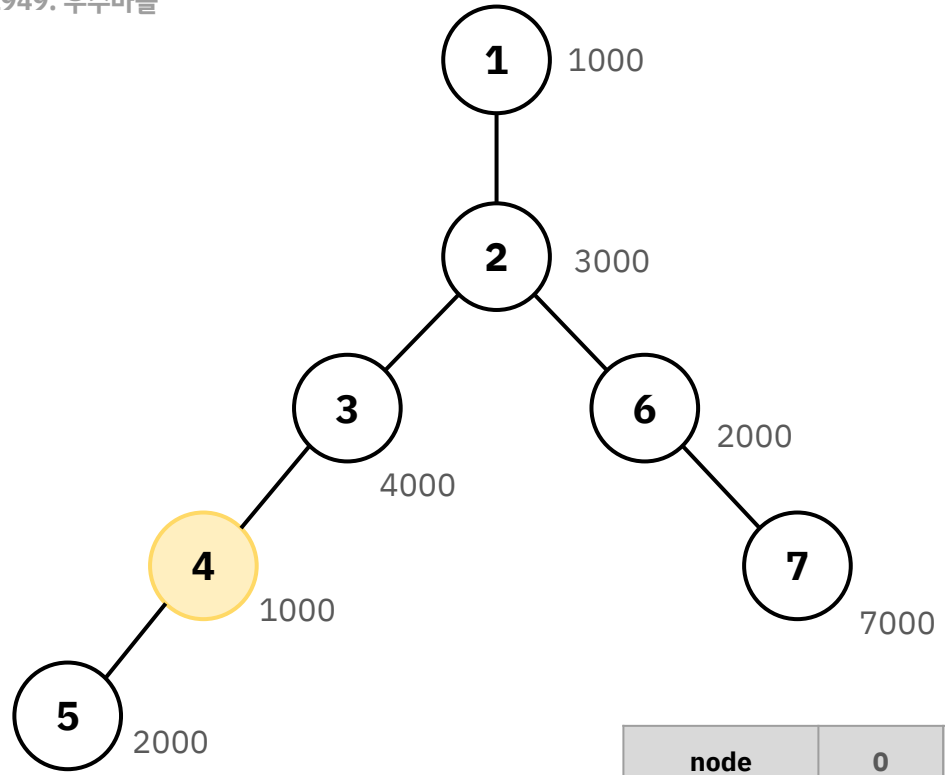


현재 방문 노드(current)	5
자식 노드 (child)	X
처음 방문할 경우 ➡ 우수마을에 지정될 경우에 대해 업데이트 dp[current][1] += people[current]	2000

DFS

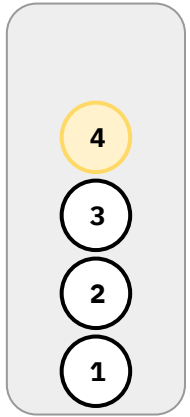


node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	0	0	0	0	**0**	0	0
dp[node][1] 우수마을O	0	1000	3000	4000	1000	**2000**	0	0



현재 방문 노드(current)	4
자식 노드 (child)	5
첫 방문에 대해 업데이트함	-

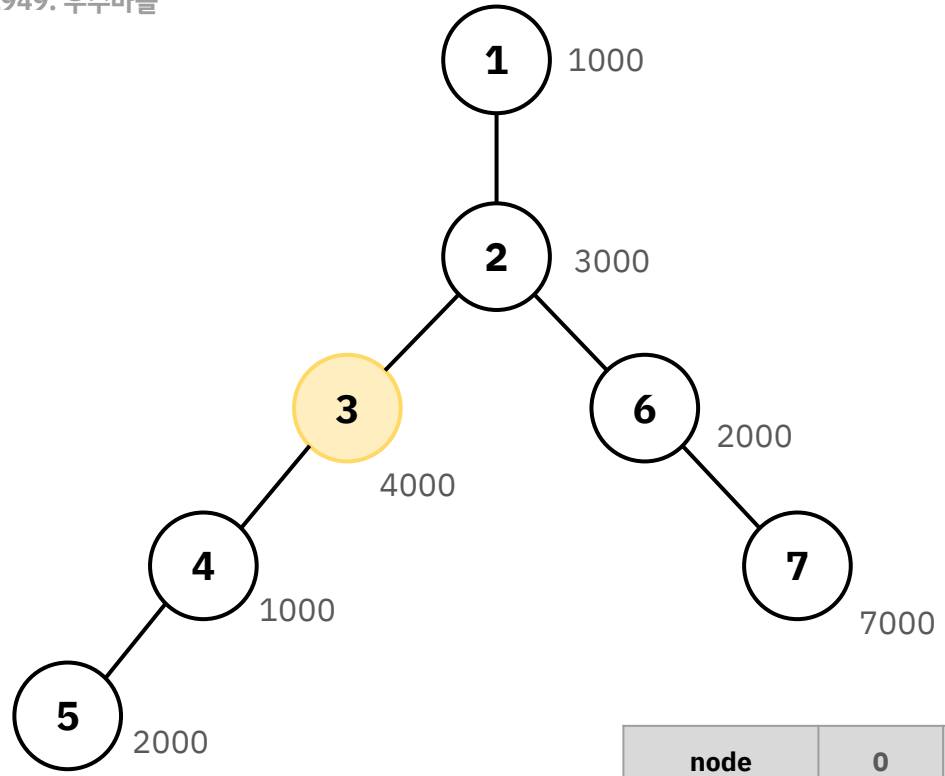
DFS



DP 업데이트

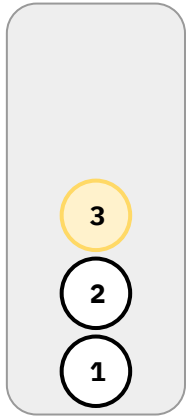
```
dp[current][0] += max(dp[child][0], dp[child][1])
dp[current][1] += dp[child][0]
```

node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	0	0	0	**2000**	0	0	0
dp[node][1] 우수마을O	0	1000	3000	4000	**1000**	2000	0	0



현재 방문 노드(current)	3
자식 노드 (child)	4
첫 방문에 대해 업데이트함	-

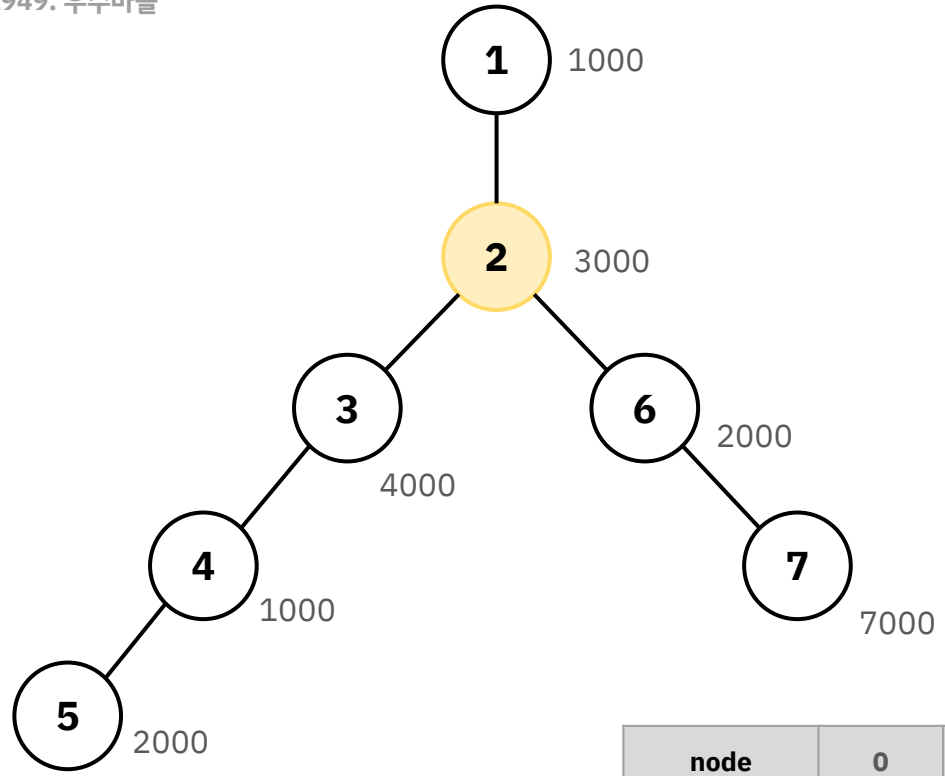
DFS



DP 업데이트

dp[current][0] += max(dp[child][0], dp[child][1])
dp[current][1] += dp[child][0]

node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	0	0	**2000**	2000	0	0	0
dp[node][1] 우수마을O	0	1000	3000	**6000**	1000	2000	0	0

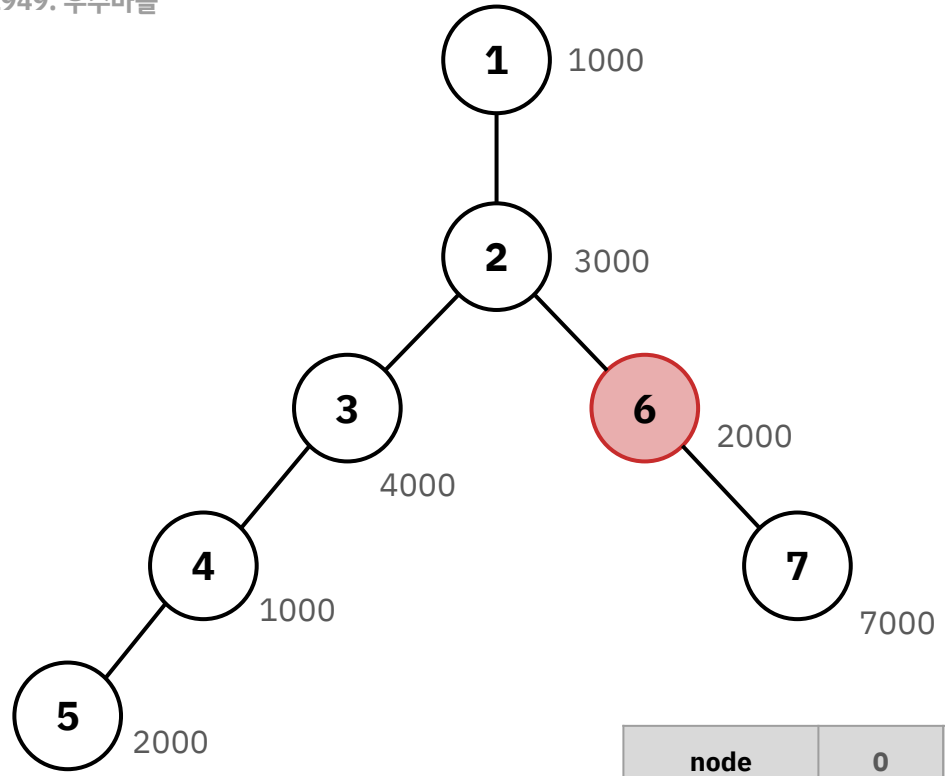


현재 방문 노드(current)	2
자식 노드 (child)	1,3,6
첫 방문에 대해 업데이트함	-

➡️ 방문하지 않은 자식노드 [6] 탐색

DFS

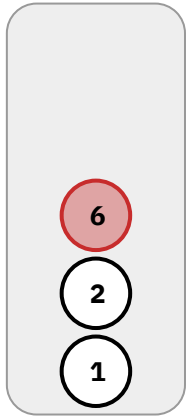
node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	0	**6000**	2000	2000	0	0	0
dp[node][1] 우수마을O	0	1000	**5000**	6000	1000	2000	0	0



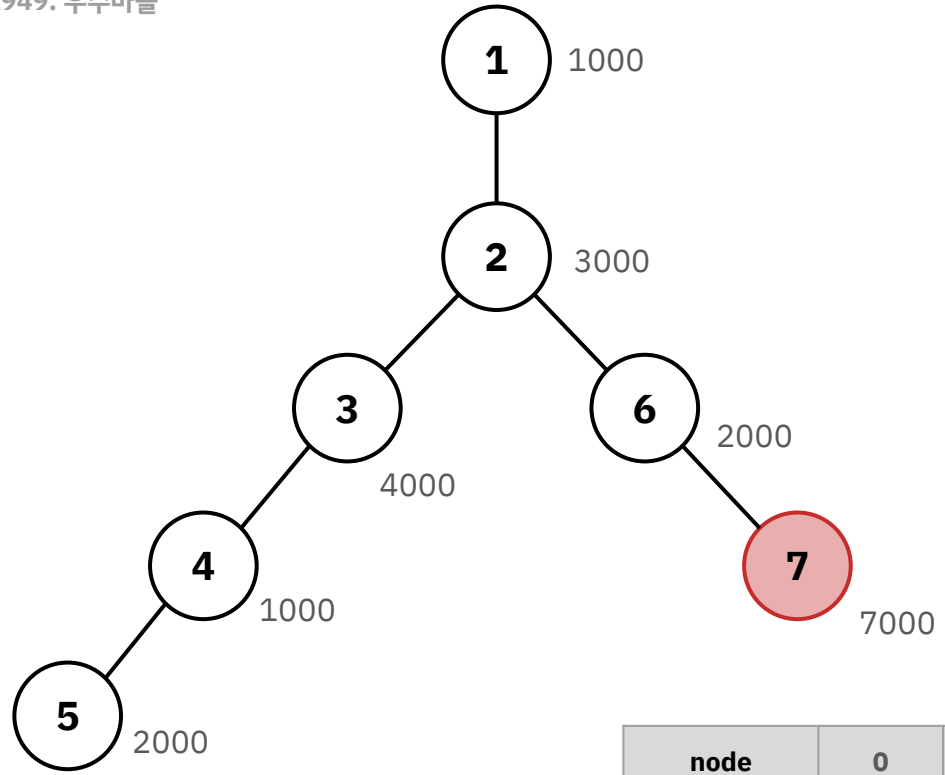
현재 방문 노드(current)	6
자식 노드 (child)	7
처음 방문할 경우 ➡ 우수마을에 지정될 경우에 대해 업데이트 dp[current][1] += people[current]	2000

➡️ 자식노드 [7] 탐색

DFS

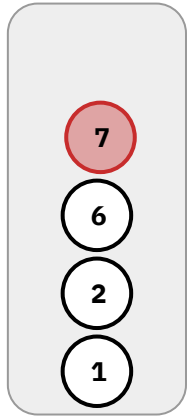


node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	0	6000	2000	2000	0	0	0
dp[node][1] 우수마을O	0	1000	5000	6000	1000	2000	2000	0

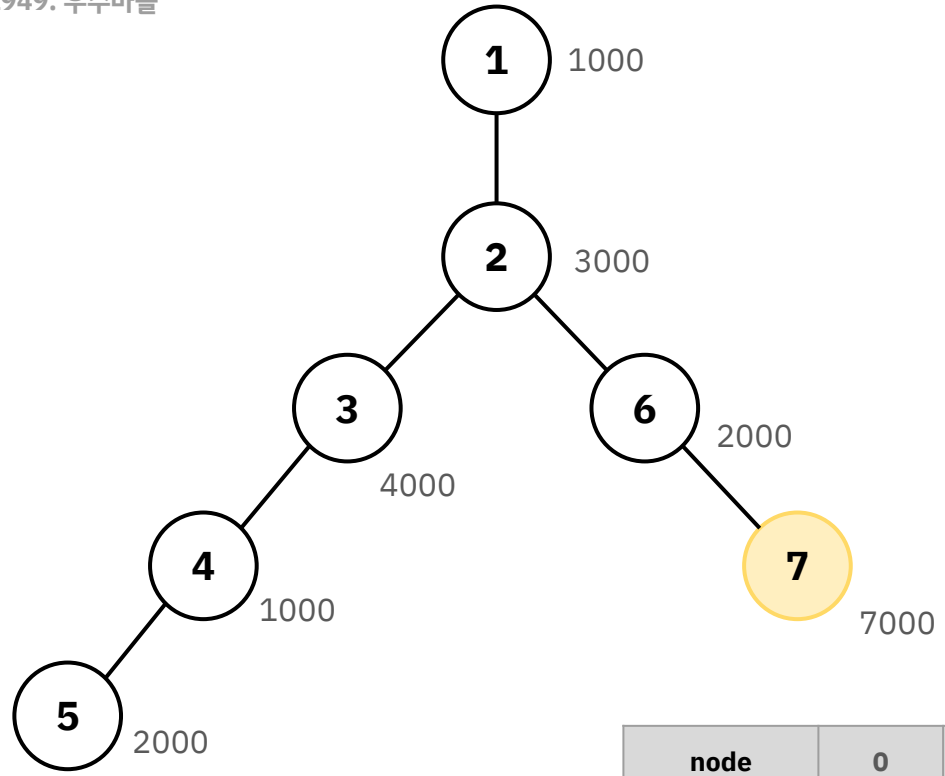


현재 방문 노드(current)	7
자식 노드 (child)	X
처음 방문할 경우 ➡ 우수마을에 지정될 경우에 대해 업데이트 dp[current][1] += people[current]	7000

DFS



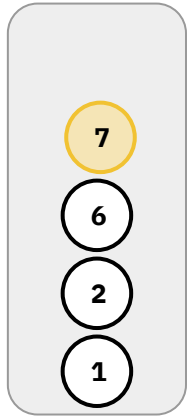
node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	0	6000	2000	2000	0	0	**0**
dp[node][1] 우수마을O	0	1000	5000	6000	1000	2000	2000	**7000**



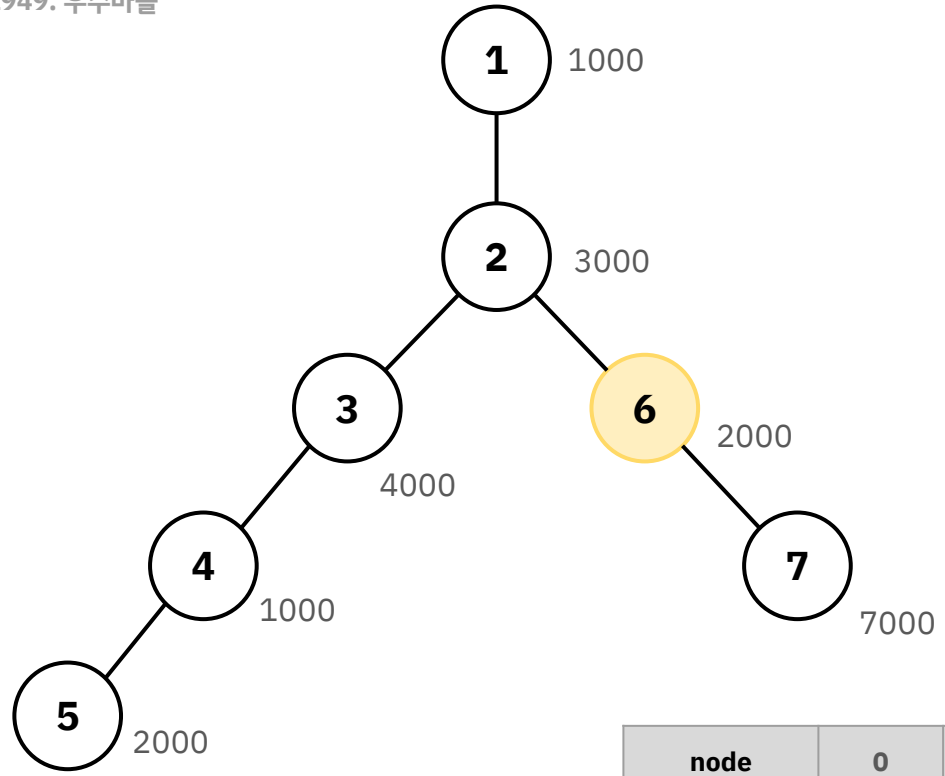
현재 방문 노드(current)	7
자식 노드 (child)	X
처음 방문할 경우 ➡ 우수마을에 지정될 경우에 대해 업데이트 dp[current][1] += people[current]	7000

DP 업데이트

DFS

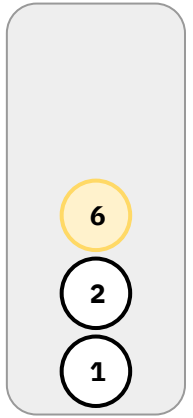


node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	0	6000	2000	2000	0	0	**0**
dp[node][1] 우수마을O	0	1000	5000	6000	1000	2000	2000	**7000**



현재 방문 노드(current)	6
자식 노드 (child)	7
첫 방문에 대해 업데이트함	-

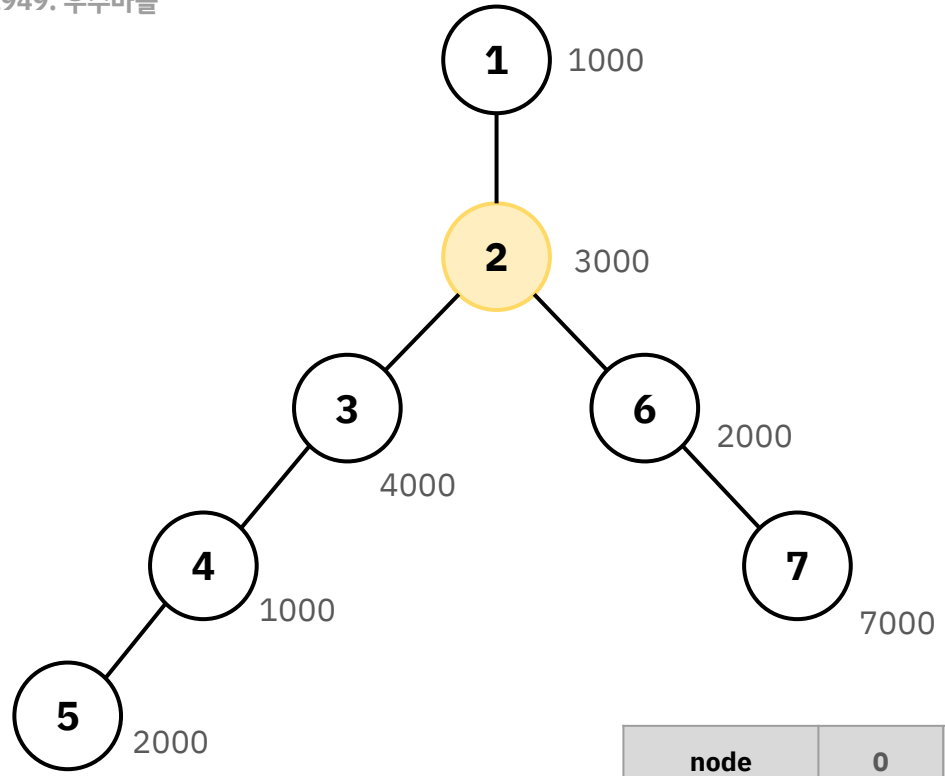
DFS



DP 업데이트

```
dp[current][0] += max(dp[child][0], dp[child][1])
dp[current][1] += dp[child][0]
```

node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	0	6000	2000	2000	0	**7000**	0
dp[node][1] 우수마을O	0	1000	5000	6000	1000	2000	**2000**	7000



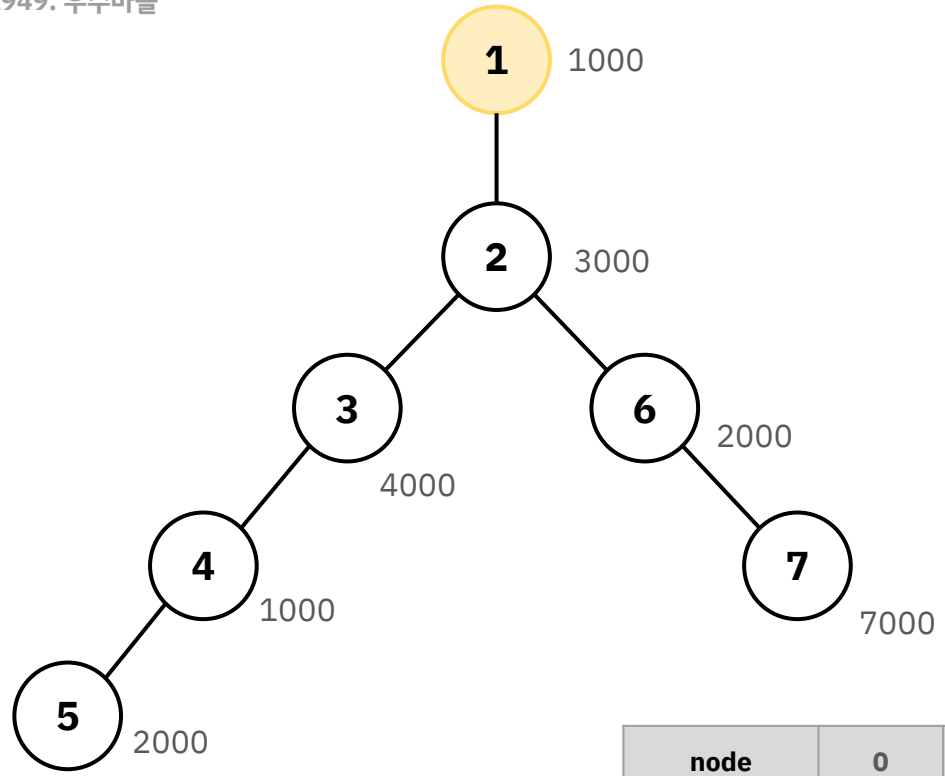
현재 방문 노드(current)	2
자식 노드 (child)	1,3,6
첫 방문에 대해 업데이트함	-

DP 업데이트

dp[current][0] += max(dp[child][0], dp[child][1])
dp[current][1] += dp[child][0]

DFS

node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	0	**13000**	2000	2000	0	7000	0
dp[node][1] 우수마을O	0	1000	**12000**	6000	1000	2000	2000	7000



현재 방문 노드(current)	1
자식 노드 (child)	2
첫 방문에 대해 업데이트함	-

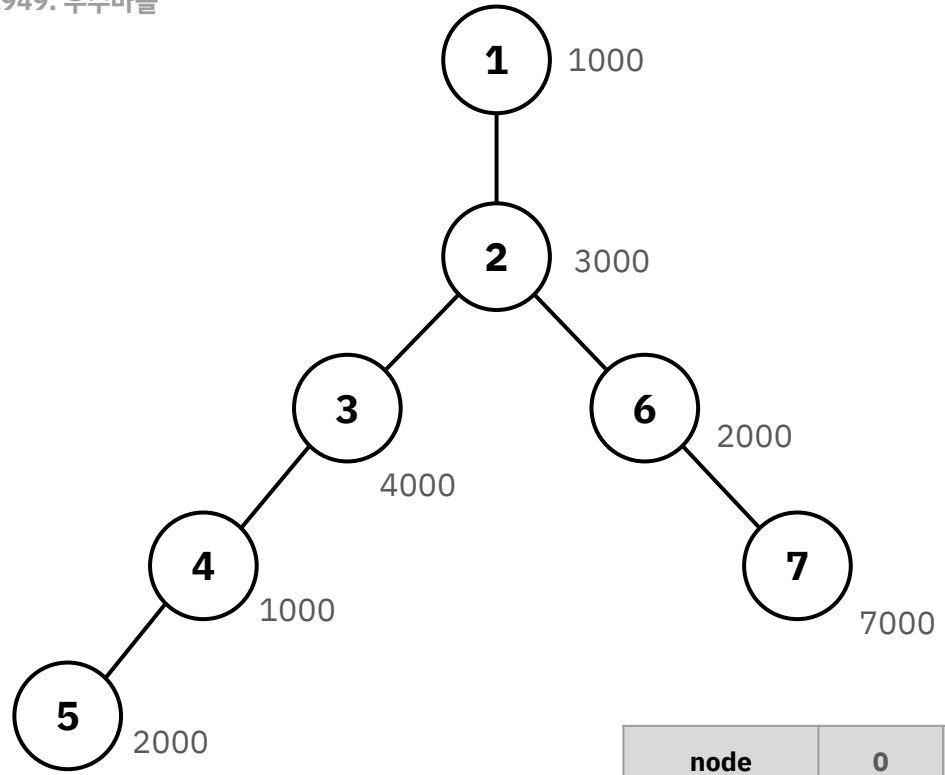
DFS

1

DP 업데이트

dp[current][0] += max(dp[child][0], dp[child][1])
dp[current][1] += dp[child][0]

node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	**13000**	13000	2000	2000	0	7000	0
dp[node][1] 우수마을O	0	**14000**	12000	6000	1000	2000	2000	7000



현재 방문 노드(current)	-
자식 노드 (child)	-
첫 방문에 대해 업데이트함	-

DFS

DFS 탐색 종료

dp[1][0]과 dp[1][1] 중에서 더 큰 값을 출력!
=> 정답: 14000

node	0	1	2	3	4	5	6	7
dp[node][0] 우수마을X	0	**13000**	13000	2000	2000	0	7000	0
dp[node][1] 우수마을O	0	**14000**	12000	6000	1000	2000	2000	7000