BOJ 2056

작업

소프트웨어학과 201921017 이지우

1. 접근

문제

수행해야 할 작업 N개 (3 ≤ N ≤ 10000)가 있다. 각각의 작업마다 걸리는 시간(1 ≤ 시간 ≤ 100)이 정수로 주어진다.

몇몇 작업들 사이에는 선행 관계라는 게 있어서, 어떤 작업을 수행하기 위해 반드시 먼저 완료되어야 할 작업들이 있다. 이 작업들은 번호가 아주 예쁘게 매겨져 있어서, K번 작업에 대해 선행 관계에 있는(즉, K번 작업을 시작하기 전에 반드시 먼저 완료되어야 하는) 작업들의 번호는 모두 1 이상 (K-1) 이하이다. 작업들 중에는, 그것에 대해 선행 관계에 있는 작업이 하나도 없는 작업이 반드시 하나 이상 존재한다. (1번 작업이 항상 그러하다)

즉, 무조건 1번 작업부터 시작

모든 작업을 완료하기 위해 필요한 최소 시간을 구하여라. 물론, 서로 선행 관계가 없는 작업들은 동시에 수행 가능하다.

입력

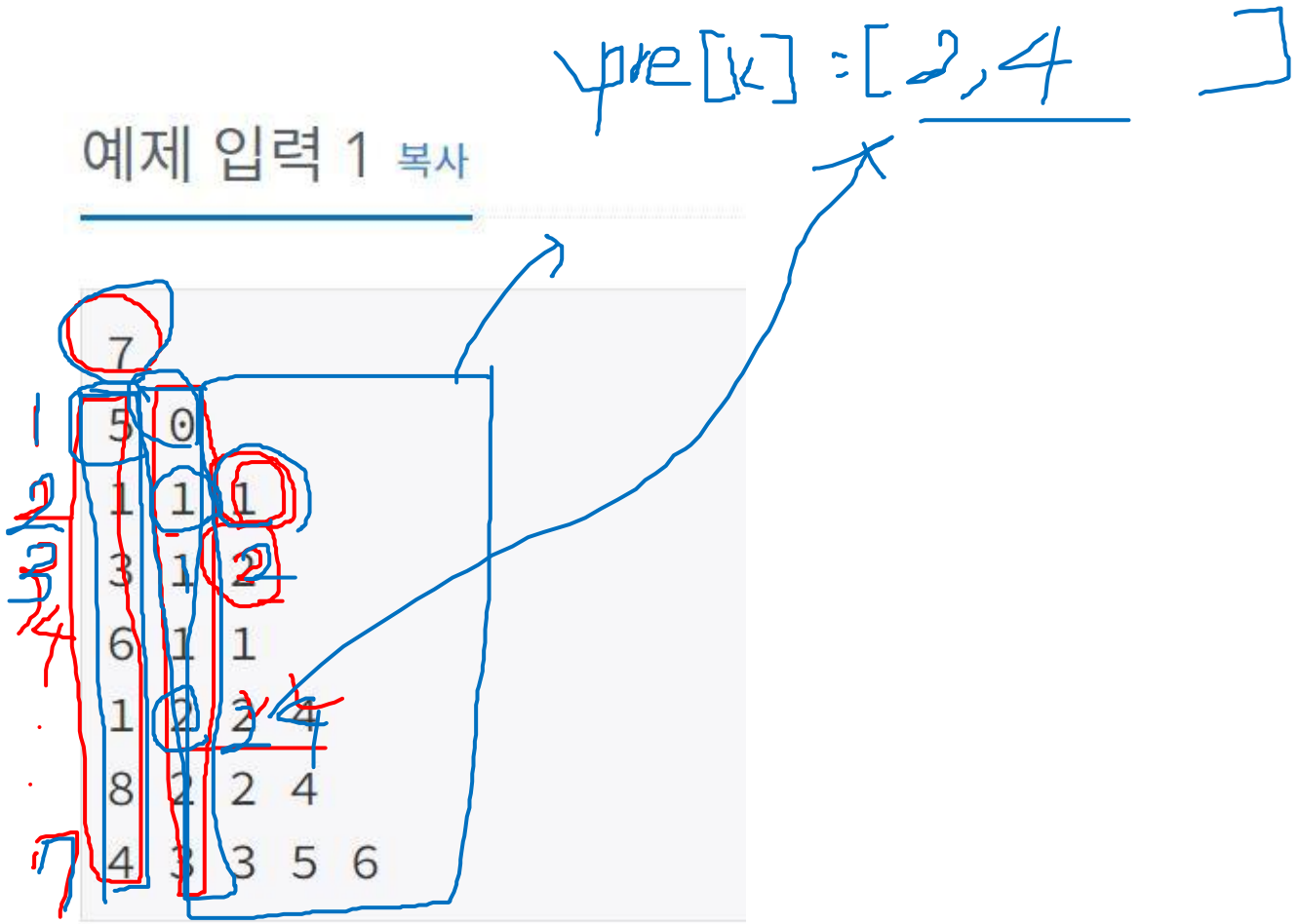
첫째 줄에 N이 주어진다.

두 번째 줄부터 N+1번째 줄까지 N개의 줄이 주어진다. 2번째 줄은 1번 작업, 3번째 줄은 2번 작업, ..., N+1번째 줄은 N번 작업을 각각 나타낸다. 각 줄의 처음에는, 해당 작업에 걸리는 시간이 먼저 주어지고, 그 다음에 그 작업에 대해 선행 관계에 있는 작업들의 개수(0 ≤ 개수 ≤ 100)가 주어진다. 그리고 선행 관계에 있는 작업들의 번호가 주어진다.

출력

첫째 줄에 모든 작업을 완료하기 위한 최소 시간을 출력한다.

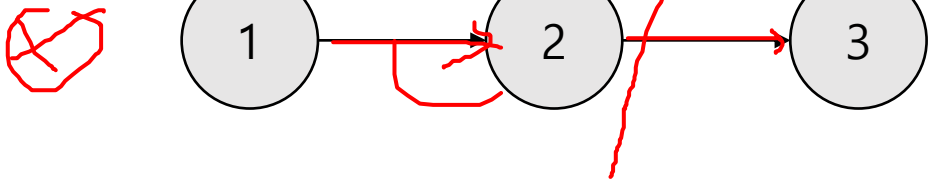
1. 접근



1. 접근

	0	1	2	3
time		5	7	2

[CASE 1]

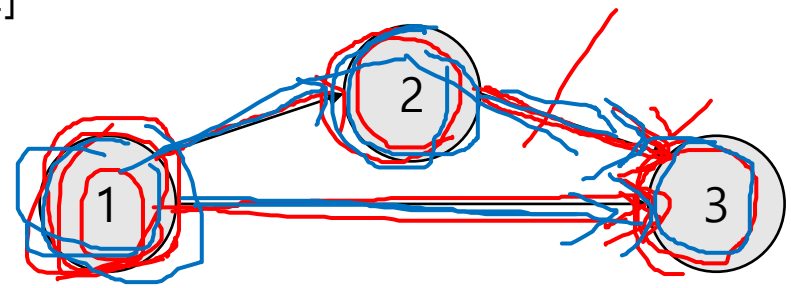


① 1번 작업을 완료하는 데 걸리는 시간 = time[1] = 5
② 2번 작업을 완료하는 데 걸리는 시간 = ① + time[2] = 5+7 = 12
③ 3번 작업을 완료하는 데 걸리는 시간 = ② + time[3] = 5+7+2 = 14

1. 접근

	0	1	2	3
time		5	7	2

[CASE 2]



① 1번 작업을 완료하는 데 걸리는 시간 = time[1] = 5
② 2번 작업을 완료하는 데 걸리는 시간 = ① + time[2] = 5+7 = 12
③-1 3번 작업을 완료하는 데 걸리는 시간 = ② + time[3] = 5+7+2 = 14
③-2 3번 작업을 완료하는 데 걸리는 시간 = ① + time[3] = 5+2 = 7

1. 접근

DP[k] : k번째 작업을 마치기 위해 필요한 최소한의 시간

DP[K] = max(DP[p] for each p in *Precedes*[K]) + time[K]

Top-down

```
def solve(curr) {
    if DP[curr] != 0, then return DP[curr];

    for each p ∈ Precedes[curr]
        DP[curr] ← max(DP[curr], solve(p)];
    DP[curr] += time[curr];

    return DP[curr];
```

Bottom-up

```
def solve() {
    total_time ← 0;

    for i ← 1 to N
        for each p ∈ Precedes[i]
            DP[i] ← max(DP[i], DP[p]);
        DP[i] += time[i];
        total_time ← max(total_time, DP[i]);

    return total_time;
```

2. 구현 – getMinTime()

```
int getMinTime(int n, vector<vector<int>> &pre, vector<int> &time) {
    vector<int> dp(n + 1, 0);

    int total_time = 0;
    for (int i = 1; i <= n; i++) {
        for (int p : pre[i]) {
            dp[i] = max(dp[i], dp[p]);
        }
        dp[i] += time[i];
        total_time = max(total_time, dp[i]);
    }

    return total_time;
}
```

2. 구현 - 입출력

```
int n;
cin >> n;

vector<vector<int>> pre(n + 1, vector<int>());
vector<int> time(n + 1, 0);

for (int i = 1; i <= n; i++) {
    cin >> time[i];

    int p_nums;
    cin >> p_nums;

    for (int j = 0; j < p_nums; j++) {
        int p; cin >> p;
        pre[i].push_back(p);
    }
}

cout << getMinTime(n, pre, time) << "\n";
```

Thx!

소프트웨어학과 201921017 이지우