Dynamic Programming with Data Structures

2020 Spring 중급

다이나믹 프로그래밍?

→ 작은 문제의 답을 구해 두고 이를 이용해 큰 문제의 답을 계산하는 기법

기본적으로

'나는 10번째 피보나치 수를 어떻게 구하는지는 잘 모르겠지만

아무튼 8번째랑 9번째 수가 구해져 있다고 가정하면

그걸로 10번째 수를 구할 수 있을 거 같음'

뭐 이런 거

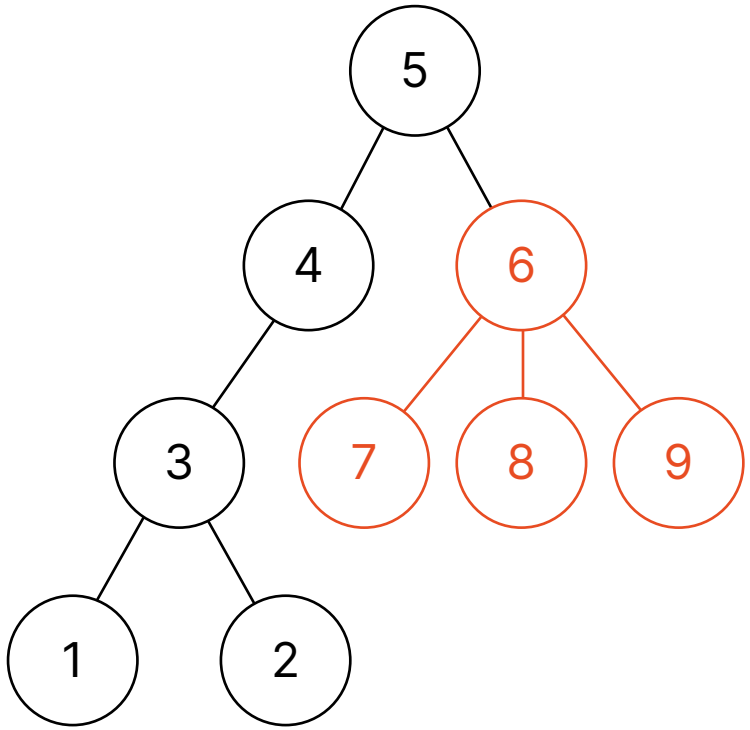
#15681 – 트리와 쿼리

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

- 정점 U를 루트로 하는 서브트리에 속한 정점의 수를 출력한다.

정점, 쿼리 모두 100,000개

- 정점 *U*를 루트로 하는 서브트리에 속한 정점의 수를 출력한다.



U = 6일 때 정점 수는 4

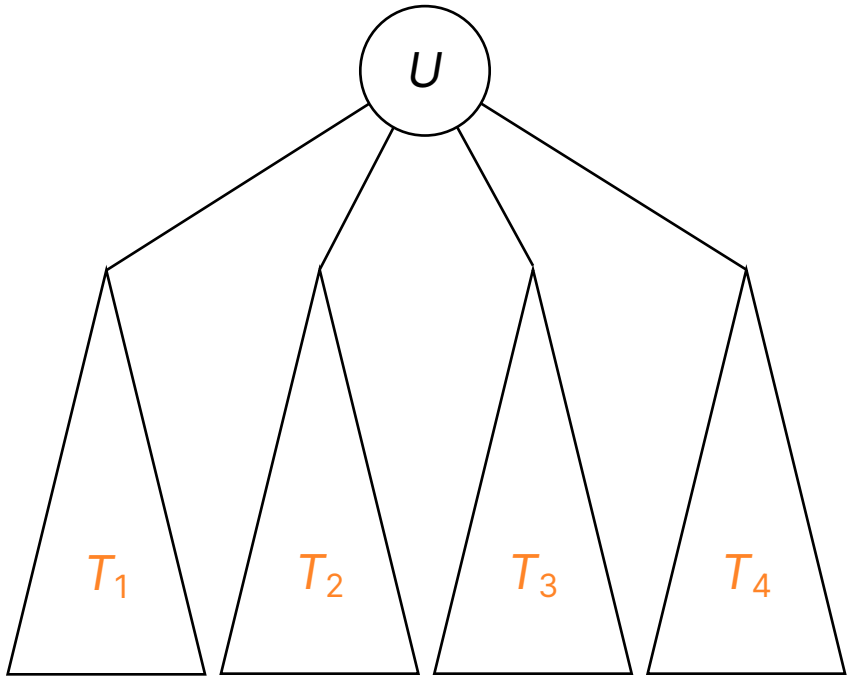
쿼리 하나 처리하는 방법은

1. DFS로 각 노드들의 부모 노드를 전처리한 후 – O(*N*)
2. U에서부터 다시 DFS를 해서 서브트리 노드 개수 세기 – O(*N*)

...인데 쿼리가 100,000개라서 더 효율적인 방법이 필요

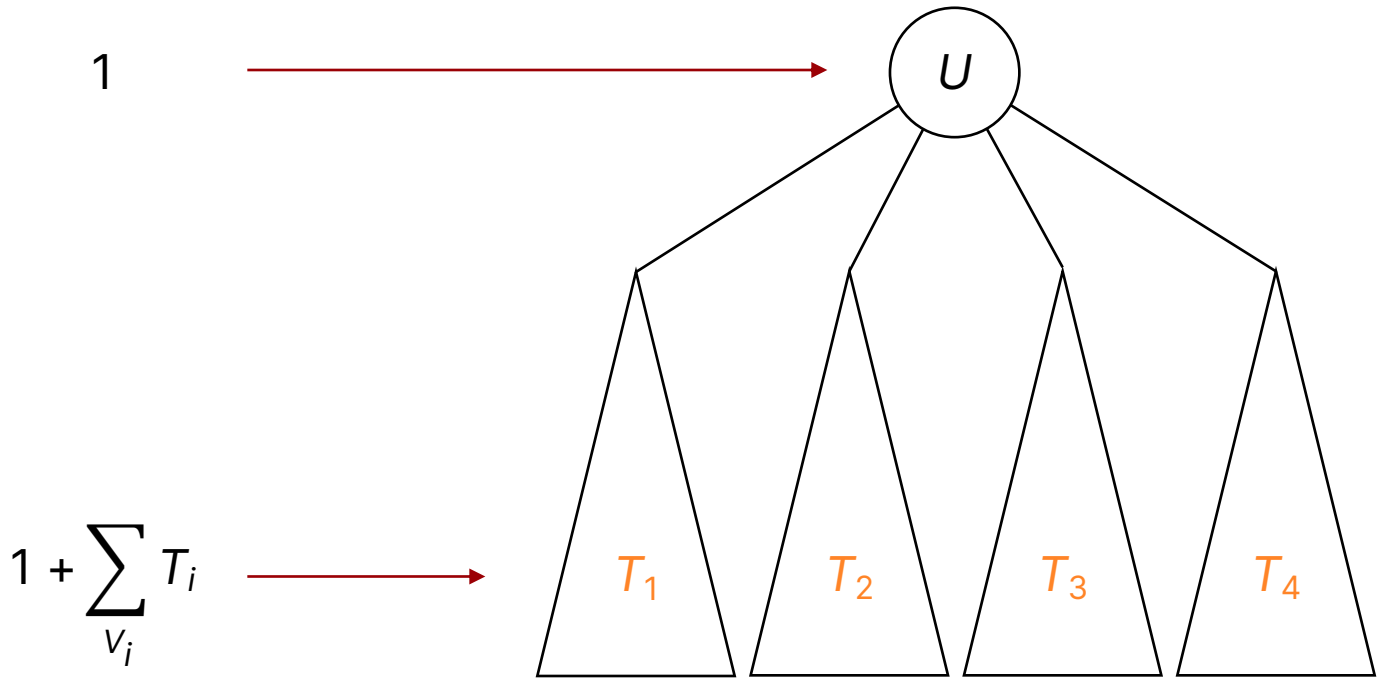
- 어떤 노드 U의 자식 노드direct child V_i들에 대해 서브트리 크기가 모두 계산되어 있다고 가정한다면 어떨까?

암튼 서브트리 크기들이
다 구해져 있고 그 값은 순서대로



라면 어떨까

$$U\text{의 서브트리 크기} = \left(1 + \sum_{V_i} T_i\right)$$

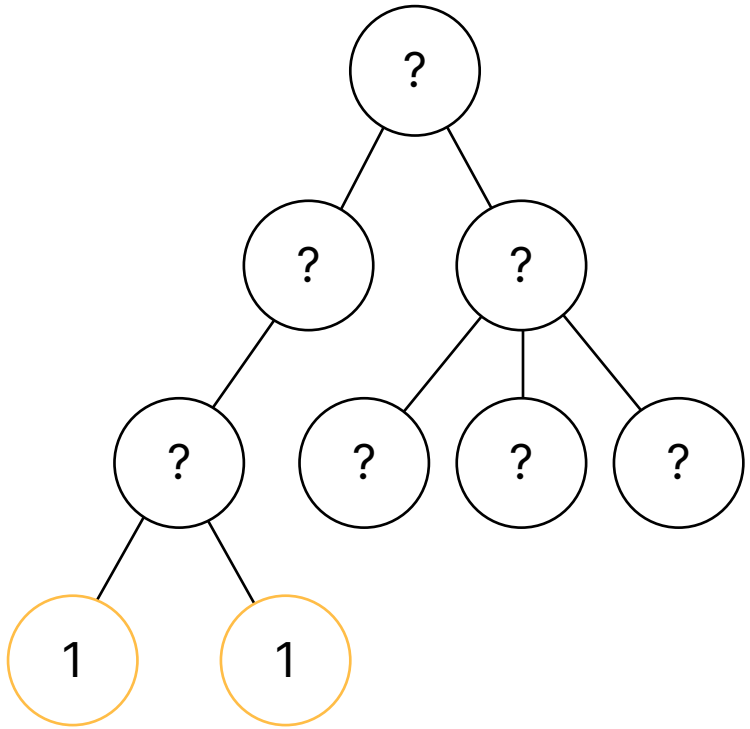


결과적으로 이 문제는

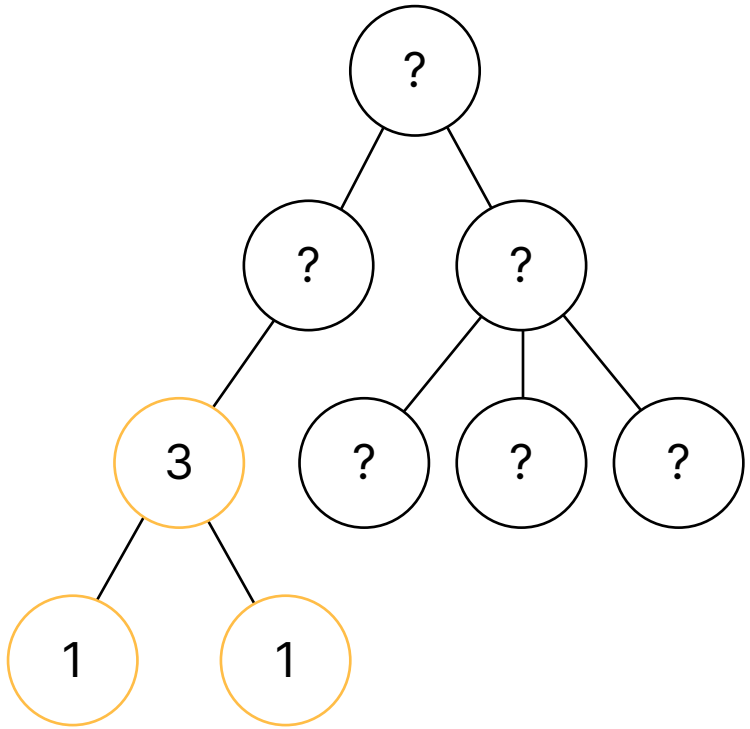
$$f(U) = \begin{cases} \left(1 + \sum_{V_i} f(V_i)\right) & U\text{가 리프 노드가 아님} \\ 1 & U\text{가 리프 노드} \end{cases}$$

를 점화식으로 하는 다이나믹 프로그래밍으로 생각해 볼 수 있다!

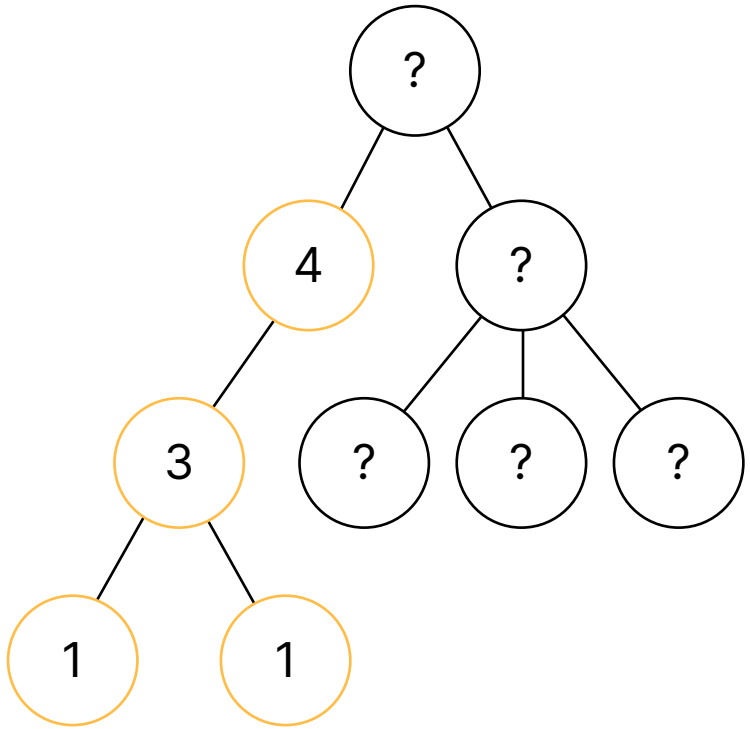
- DFS를 하는 식으로, 리프 노드부터 위로 가면서 서브트리의 크기를 전처리



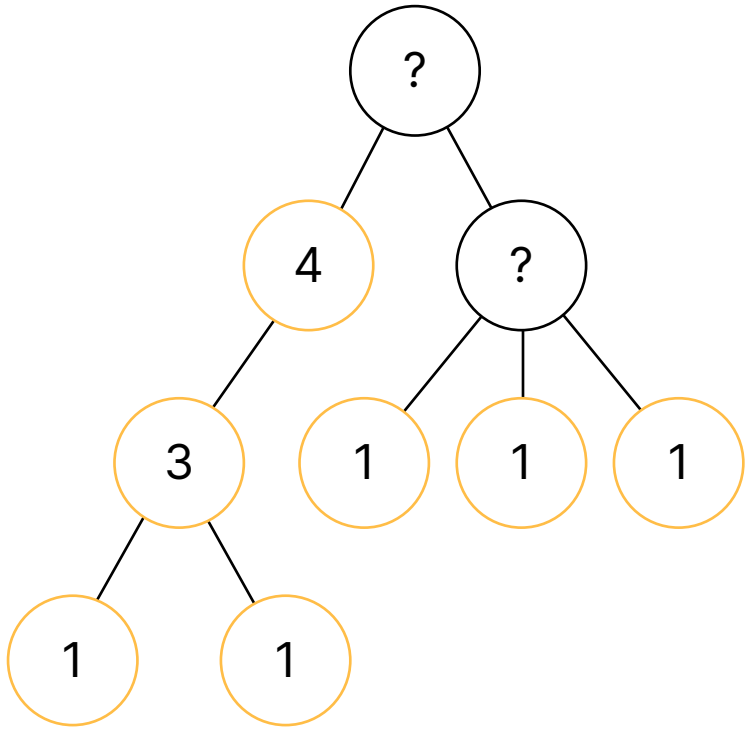
- DFS를 하는 식으로, 리프 노드부터 위로 가면서 서브트리의 크기를 전처리



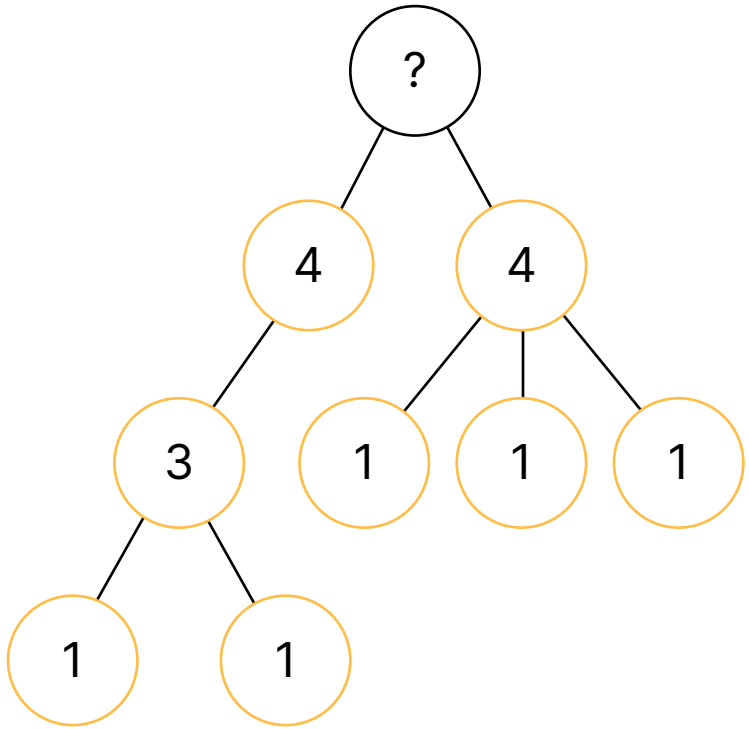
- DFS를 하는 식으로, 리프 노드부터 위로 가면서 서브트리의 크기를 전처리



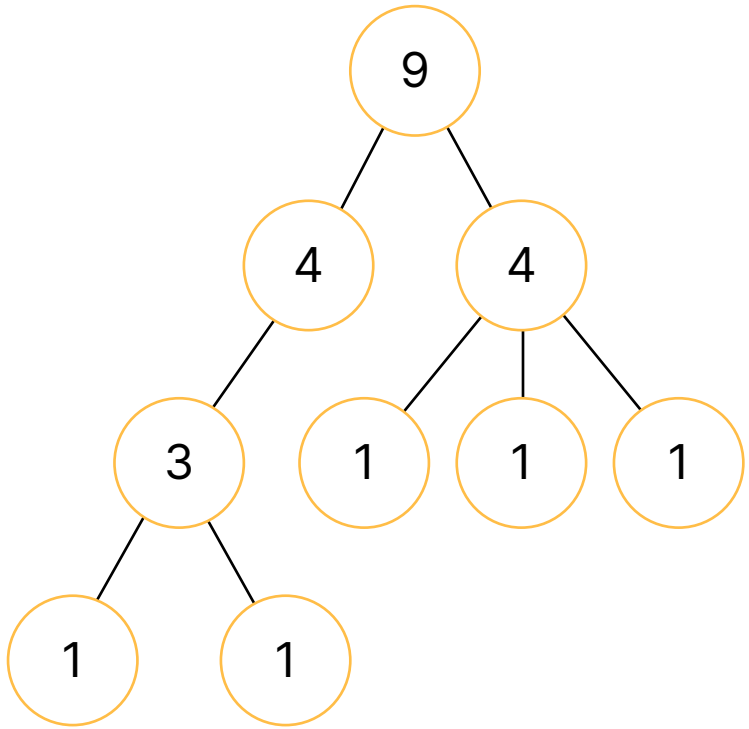
- DFS를 하는 식으로, 리프 노드부터 위로 가면서 서브트리의 크기를 전처리



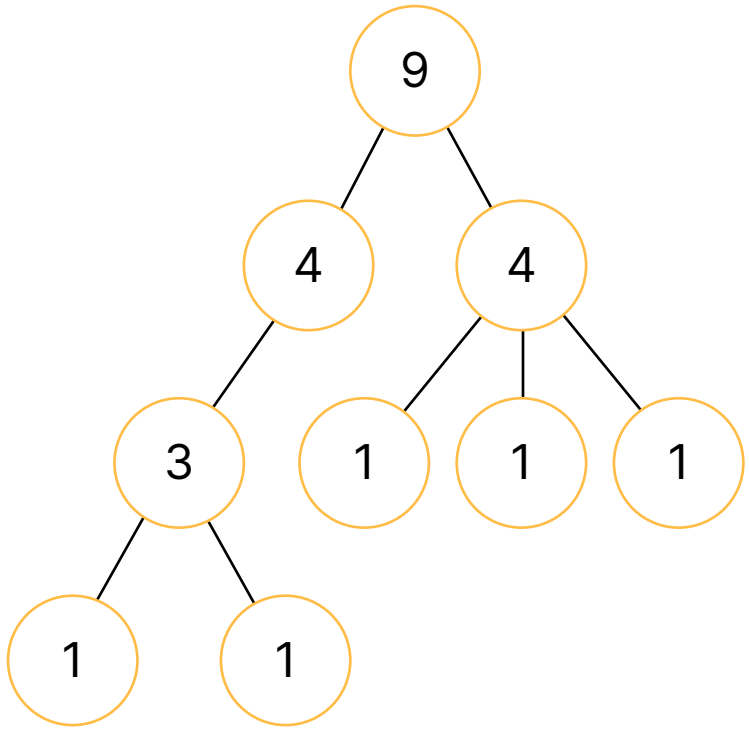
- DFS를 하는 식으로, 리프 노드부터 위로 가면서 서브트리의 크기를 전처리



- DFS를 하는 식으로, 리프 노드부터 위로 가면서 서브트리의 크기를 전처리



- 쿼리가 들어올 때마다 전처리된 값을 출력만 해주면 끝!
- 전처리 DFS O(*N*), 쿼리 O(*Q*), 합쳐서 O(*N* + *Q*)



```cpp
vector<vector<int>> tree(n + 1);
for (int i = 1; i < n; i++) {
    int u, v;
    cin >> u >> v;
    tree[u].emplace_back(v);
    tree[v].emplace_back(u);
}

dfs(r, -1, tree);
```

```cpp
int sz[100001];

int dfs(int u, int p, const vector<vector<int>> &tree) {
    int c = 0;
    for (int v : tree[u]) {
        if (v == p) continue;
        sz[u] += dfs(v, u, tree);
        c++;
    }
    return ++sz[u];
}
```

트리를 입력받아 서브트리 크기를 다이나믹 프로그래밍 전처리

트리와 쿼리

boj.kr/5895be659f2344419b3f8b4a245ec76e

```
while (q--) {
    int u;
    cin >> u;
    cout << sz[u] << '\n';
}
```

끝. 이게 다예요.

이렇게 트리에서 DFS를 하는 것으로 다이나믹 프로그래밍을 할 수 있다!

#1949 – 우수 마을

N개의 마을로 이루어진 나라가 있다. 이 나라는 트리 구조로 이루어져 있다. 또, 모든 마을은 연결되어 있다.

다음 세 가지 조건을 만족하면서 N개의 마을 중 몇 개의 마을을 '우수 마을'로 선정하려고 한다.

- '우수 마을'로 선정된 마을 주민 수의 총 합을 최대로 해야 한다.
- '우수 마을'끼리는 서로 인접해 있을 수 없다.
- '우수 마을'로 선정되지 못한 마을은 적어도 하나의 '우수 마을'과는 인접해 있어야 한다.

$1 \le N \le 10,000$.

이 문제를 작은 문제로 쪼개서 풀 수 있을까? 어떻게 풀 수 있을까?

일단 주민 수의 합을 최대화해야 하는 게 목표고 나머지는 조건이므로...

u번 노드를 루트로 하는 서브트리를 T_u라 하면

dp[u][1]: u번 노드가 우수 마을일 때
dp[u][0]: u번 노드가 우수 마을이 아닐 때

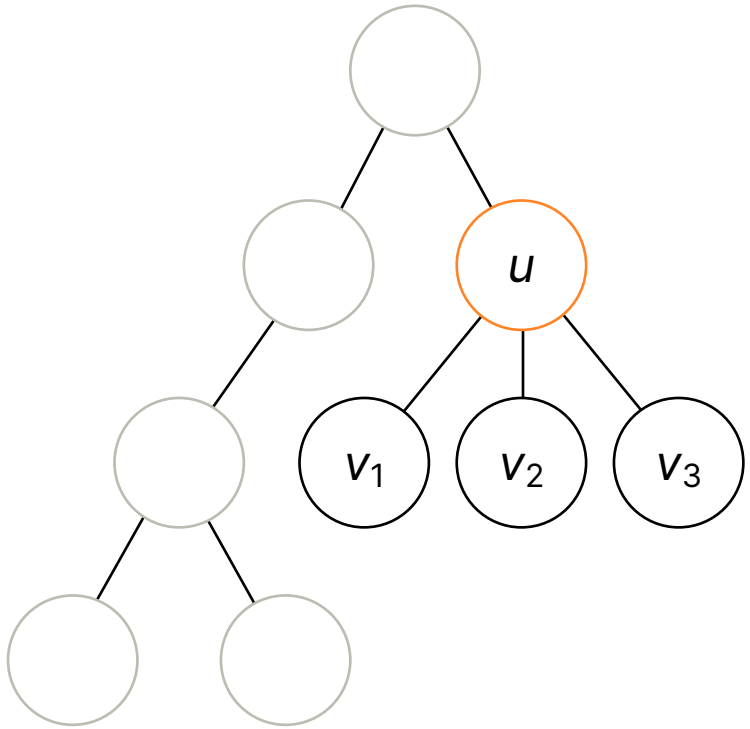
...의 T_u에서의 우수 마을들의 합의 최댓값

#1949 – 우수 마을

dp[u][1]: u번 노드가 우수 마을일 때

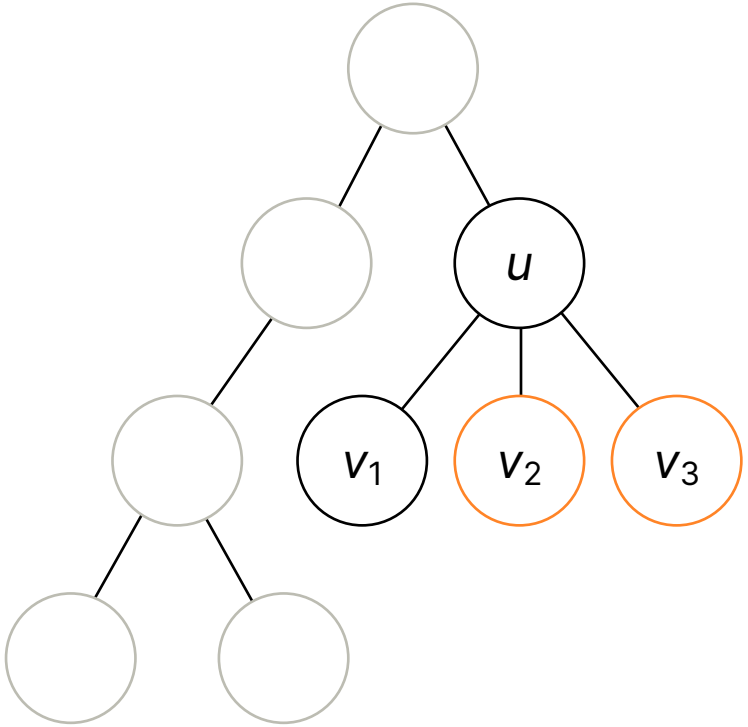
→자식 노드들은 모두 우수 마을이 아니어야 함

→그러므로, dp[u][1] = $a[u] + \sum_{v_i}$ dp[v_i][0]



dp[u][0]: u번 노드가 우수 마을이 아닐 때

→dp[u][0] = $\sum_{v_i}$ max{dp[v_i][0], dp[v_i][1]}



이런 식으로 트리를 사용한 DP에서는

- 현재 보고 있는 서브트리의 루트를 포함하는 경우
- 현재 보고 있는 서브트리의 루트를 포함하지 않는 경우

로 케이스를 나눠보는 것이 일반적

덱을 사용한 DP를 이야기하기 전에...

#11003 – 최솟값 찾기

N개의 수 $A_1, A_2, \ldots, A_N$과 L이 주어진다.

$D_i = A_{i-L+1} \sim A_i$ 중의 최솟값이라고 할 때, D에 저장된 수를 출력하는 프로그램을 작성하시오.

- $1 \le L \le N \le 5{,}000{,}000$

- 고정된 크기의 구간 내에서 뭔가를 하고 싶을 때 활용 – 슬라이딩 윈도
- 이번 문제의 경우에는 고정된 구간 안의 최솟값을 유지할 거
  - 덱 내부가 단조롭게 유지되면서
  - 현재 덱 안에 있는 원소들은 우리가 관심을 갖고 있는 구간 내 원소들의 부분집합이 되도록 적절히 pop, push

#11003 – 최솟값 찾기

A =	1	5	2	3	6	2	3	7	3	5	2	6

D =												

- 덱 내부가 단조롭게 유지되면서
- 현재 덱 안에 있는 원소들은 우리가 관심을 갖고 있는 구간 내 원소들의 부분집합이 되도록 적절히 pop, push

#11003 – 최솟값 찾기

while (front().index ≤ 0 – 3) pop_front()

while (back().value > 1) pop_back()

emplace_back(1, 0)

1 [0]		

A =	1	5	2	3	6	2	3	7	3	5	2	6

D =	1											

- 덱 내부가 단조롭게 유지되면서
- 현재 덱 안에 있는 원소들은 우리가 관심을 갖고 있는 구간 내 원소들의 부분집합이 되도록 적절히 pop, push

while (front().index ≤ 1 – 3) pop_front()

while (back().value > 5) pop_back()

emplace_back(5, 1)

1 [0]	5 [1]	

A =	1	5	2	3	6	2	3	7	3	5	2	6

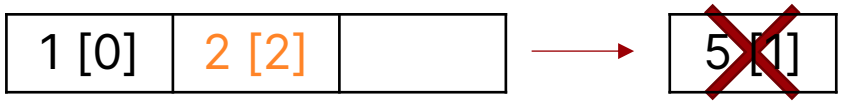
D =	1	1										

- 덱 내부가 단조롭게 유지되면서
- 현재 덱 안에 있는 원소들은 우리가 관심을 갖고 있는 구간 내 원소들의 부분집합이 되도록 적절히 pop, push

while (front().index ≤ 2 – 3) pop_front()

while (back().value > 5) pop_back()

emplace_back(2, 2)



A =	1	5	2	3	6	2	3	7	3	5	2	6

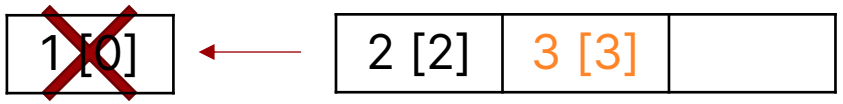
D =	1	1	1									

- 덱 내부가 단조롭게 유지되면서
- 현재 덱 안에 있는 원소들은 우리가 관심을 갖고 있는 구간 내 원소들의 부분집합이 되도록 적절히 pop, push

while (front().index ≤ 3 – 3) pop_front()

while (back().value > 3) pop_back()

emplace_back(3, 3)



A =	1	5	2	3	6	2	3	7	3	5	2	6

D =	1	1	1	2								

- 덱 내부가 단조롭게 유지되면서
- 현재 덱 안에 있는 원소들은 우리가 관심을 갖고 있는 구간 내 원소들의 부분집합이 되도록 적절히 pop, push

while (front().index ≤ 4 – 3) pop_front()

while (back().value > 6) pop_back()

emplace_back(6, 4)

2 [2]	3 [3]	6 [4]

A =	1	5	2	3	6	2	3	7	3	5	2	6

D =	1	1	1	2	2							

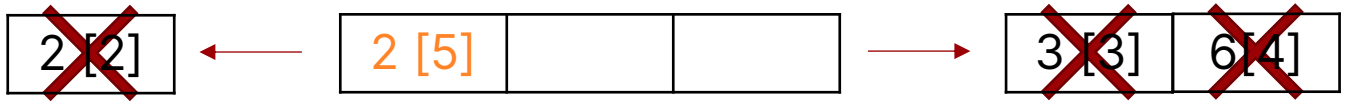
- 덱 내부가 단조롭게 유지되면서
- 현재 덱 안에 있는 원소들은 우리가 관심을 갖고 있는 구간 내 원소들의 부분집합이 되도록 적절히 pop, push

#11003 – 최솟값 찾기

while (front().index ≤ 5 – 3) pop_front()

while (back().value > 2) pop_back()

emplace_back(2, 5)



A =	1	5	2	3	6	2	3	7	3	5	2	6

D =	1	1	1	2	2	2						

- 덱 내부가 단조롭게 유지되면서
- 현재 덱 안에 있는 원소들은 우리가 관심을 갖고 있는 구간 내 원소들의 부분집합이 되도록 적절히 pop, push

while (front().index ≤ 6 – 3) pop_front()

while (back().value > 3) pop_back()

emplace_back(3, 6)

2 [5]	3 [6]	

A =

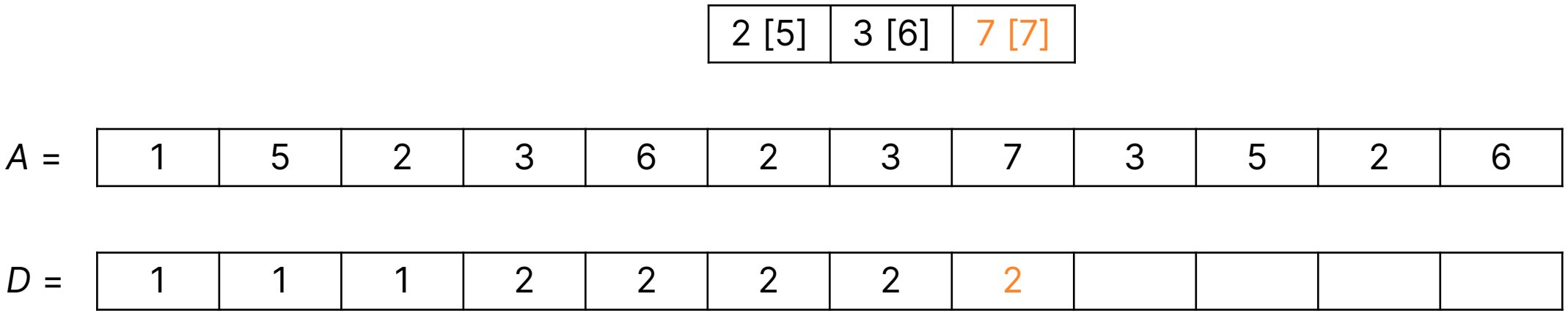
1	5	2	3	6	2	3	7	3	5	2	6

D =

1	1	1	2	2	2	2					

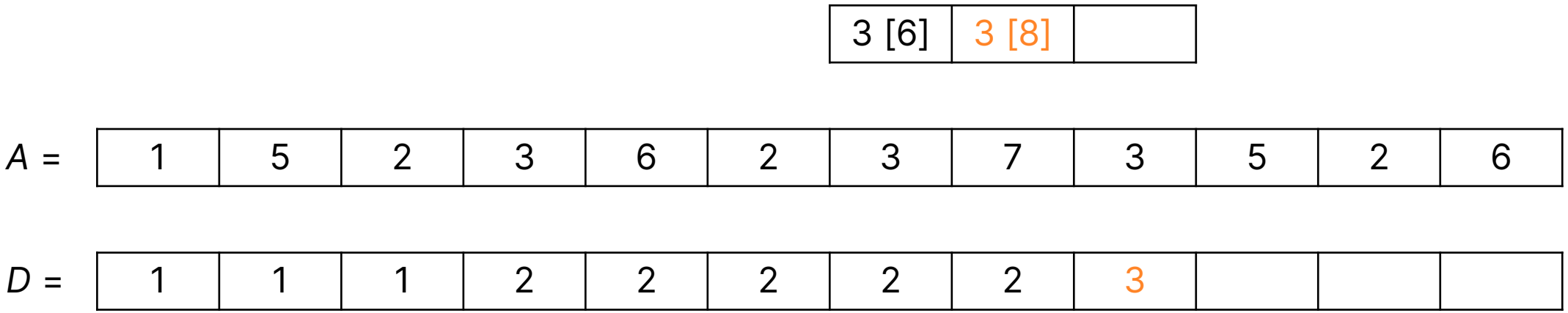
- 덱 내부가 단조롭게 유지되면서
- 현재 덱 안에 있는 원소들은 우리가 관심을 갖고 있는 구간 내 원소들의 부분집합이 되도록 적절히 pop, push

#11003 – 최솟값 찾기



- 덱 내부가 단조롭게 유지되면서
- 현재 덱 안에 있는 원소들은 우리가 관심을 갖고 있는 구간 내 원소들의 부분집합이 되도록 적절히 pop, push

#11003 – 최솟값 찾기



- 덱 내부가 단조롭게 유지되면서
- 현재 덱 안에 있는 원소들은 우리가 관심을 갖고 있는 구간 내 원소들의 부분집합이 되도록 적절히 pop, push

#11003 – 최솟값 찾기

3 [8]	5 [9]	

A =	1	5	2	3	6	2	3	7	3	5	2	6

D =	1	1	1	2	2	2	2	2	3	3		

- 덱 내부가 단조롭게 유지되면서
- 현재 덱 안에 있는 원소들은 우리가 관심을 갖고 있는 구간 내 원소들의 부분집합이 되도록 적절히 pop, push

#11003 – 최솟값 찾기

										2 [10]		

A =	1	5	2	3	6	2	3	7	3	5	2	6

D =	1	1	1	2	2	2	2	2	3	3	2	

- 덱 내부가 단조롭게 유지되면서
- 현재 덱 안에 있는 원소들은 우리가 관심을 갖고 있는 구간 내 원소들의 부분집합이 되도록 적절히 pop, push

#11003 – 최솟값 찾기

2 [10]	6 [11]	

A =	1	5	2	3	6	2	3	7	3	5	2	6

D =	1	1	1	2	2	2	2	2	3	3	2	2

- 덱 내부가 단조롭게 유지되면서
- 현재 덱 안에 있는 원소들은 우리가 관심을 갖고 있는 구간 내 원소들의 부분집합이 되도록 적절히 pop, push

모든 원소는 한 번씩 추가되고 한 번씩 제거하므로 결과적으로 시간 복잡도는 $O(N)$

```cpp
int n, k;
cin >> n >> k;

deque<pii> d; // value, index
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    while (d.size() && (d.front().second <= i - k)) d.pop_front();
    while (d.size() && (d.back().first > x)) d.pop_back();
    d.emplace_back(x, i);
    cout << d.front().first << ' ';
}
```

그대로 구현!

최솟값 찾기

boj.kr/5576a542d1ae47d3898d4e6fc2410362

#15678 – 연세워터파크

- 정수 K_i가 쓰여진 징검다리 $N \le 10^5$개
- 아무 시작점에서 시작해서 몇 개든 한 번씩만 마음대로 밟은 뒤, 나오고 싶을 때 나온다
- 거리가 D 이하인 징검다리로만 점프할 수 있다

시작점을 포함해, 밟은 모든 징검다리에 쓰여진 정수의 합을 최대화하자.

관찰

- 징검다리를 한 번만 밟을 수 있고 거리 D 이하로만 움직일 수 있으므로 왼쪽에서 오른쪽으로만 움직여도 되지 않을까?
- 그렇다면 DP 점화식은

$$\mathrm{dp}[u] = \mathrm{a}[u] + \max\left\{0, \max_{u-D \le v < u} \mathrm{dp}[v]\right\}$$

관찰

- 징검다리를 한 번만 밟을 수 있고 거리 D 이하로만 움직일 수 있으므로 왼쪽에서 오른쪽으로만 움직여도 되지 않을까?
- 그렇다면 DP 점화식은

$$\mathrm{dp}[u] = \mathrm{a}[u] + \max\left\{0, \max_{u-D \le v < u} \mathrm{dp}[v]\right\}$$

- 아까처럼 덱을 이용하면 밑줄 부분을 O(D)에서 amortized O(1)로 줄일 수 있다

```cpp
int n, d;
cin >> n >> d;

ll mx = -1e18;
deque<int> dq;

for (int i = 0; i < n; i++) {
    while (dq.size() && dq.front() < i - d) dq.pop_front();

    cin >> dp[i];
    if (dq.size()) dp[i] += max(0ll, dp[dq.front()]);
    mx = max(dp[i], mx);

    while (dq.size() && dp[dq.back()] <= dp[i]) dq.pop_back();
    dq.emplace_back(i);
}

cout << mx;
```

$$dp[u] = a[u] + \max\left\{0, \max_{u-D \le v < u} dp[v]\right\}$$

정답은 dp[u]들의 최댓값. 징검다리는 음수도 될 수 있으므로 최댓값을 음수 무한대로 초기화함에 주의

연세워터파크

boj.kr/8f3fddd04f314451bd4309fb31c41bef

이번 주의 문제 셋 – 트리에서의 다이나믹 프로그래밍

문제 난이도는 2020. 5. 16 기준

- 5 #15861 – 트리와 쿼리
- 1 #1949 – 우수 마을

- 2 #2533 – SNS
- 1 #2213 – 트리의 독립집합
- 5 #17831 – 대기업 승범이네
- 5 #10273 – 고대 동굴 탐사
- 4 #1814 – 지붕 색칠하기
- 3 #1289 – 트리의 가중치
- 1 #17969 – Gene Tree

https://solved.ac/problems/algorithms/92

이번 주의 문제 셋 – 덱을 활용한 다이나믹 프로그래밍

문제 난이도는 2020. 5. 16 기준

5 #15678 – 연세워터파크

---

2 #5977 – Mowing the Lawn

https://solved.ac/problems/algorithms/108