



# 2020 SKH Algorithm Camp

주관



후원



승고한 대회 풀이집  
Intermediate/Advanced



# SKH 엔터테인먼트

---

한양대 ALOHA  
한경수

Intermediate A

# SKH 엔터테인먼트

---

- 자료구조 SET을 이용하는 문제
- SET이란?
  - 중복을 허용하지 않는 자료구조
  - Set에 이미 들어있는 Element를 다시 집어넣어도, set에 들어있는 총 Elements의 개수는 변하지 않는다.
  - 각 월별로 스타들의 이름을 담는 set을 12개 만들고, 월에 맞게 스타들의 이름을 집어넣은 뒤, set에 들어있는 Element들의 개수를 세면 된다.

# 균형잡힌 수열

---

숭실대 SCCC  
이성서

Intermediate B

# 균형잡힌 수열

---

- 문제에서 요구하는 조건  $b = \frac{a + c}{2}$  이 등차중항이므로 균형잡힌 수열은 **등차수열**과 같습니다.
- 정렬한 뒤 모든  $2 \leq i \leq N$ 에 대해  $A[i-1]+A[i+1] == A[i]*2$ 인지 검사해 보면 됩니다.
- $O(N^2)$  정렬 알고리즘도 통과할 수 있게 데이터가 만들어졌습니다.

# 나무를 심어볼까요?

---

한양대 ALOHA  
박정호

Intermediate C  
Advanced A

# 나무를 심어볼까요?

---

- 정호가 나무를 심는 방식은 크게 두 단계로 나눠진다.

## 1. 나무를 심는다.

“나무 심기는 왼쪽에서 오른쪽으로 진행한다.”

“비어있는 칸 중 가장 왼쪽 칸에 나무를 심었을 때, 이 나무가 햇빛을 받을 수 있다면,  
바로 나무를 심는다.”

## → 왼쪽에서 오른쪽으로 진행

## 2. 나무를 뽑는다.

“가장 오른쪽의 나무가 아직 심어진 지 K일이 되지 않았다면,  
그 나무를 뽑아서 버리고, 1의 과정으로 돌아간다.”

## → 오른쪽에서 왼쪽으로 진행

# 나무를 심어볼까요?

---

- 나무가 심어지는 방향과 뽑히는 방향이 반대이다.  
즉, 현재 나무 중 가장 나중에 심어진 나무가 가장 먼저 뽑힌다.
- 여기서 **LIFO (Last In First Out)**를 연상할 수 있으며  
따라서 스택 구조를 사용해야 한다는 것을 알 수 있다.

# 나무를 심어볼까요?

---

- 나무의 정보를 스택에 넣고 빼면서 문제를 해결한다.  
여기서 나무의 정보란, 나무의 높이와 심어진 날짜이다.
- 새로 나무를 심는 것은 스택에 나무의 정보를 넣는 것으로,  
나무를 뽑는 것은 스택의 맨 위의 정보를 빼는 것으로 진행한다.
- 또한, 빈 칸을 하나 두는 것은 스택에 높이 0의 나무를 넣는 것으로 대체할 수 있다.  
모든 나무가 양의 정수의 높이를 가지기 때문에 높이 0의 나무는 어느 나무도 가리지 않고,  
빈 칸의 위치가 어디인지 확실하게 특정할 수 있다.

답은 스택에 넣어진 높이 0의 나무의 개수가 된다.

- 결론적으로 모든 나무는 스택에 최대 1회 넣어지고, 스택에서 최대 1회 빠져나온다.  
따라서 시간 복잡도는  $O(2^N) = O(N)$ 으로 해결 가능하다.

# 제가 재밌는 문제를 가져왔어요

---

숭실대 SCCC  
권욱제

Intermediate D

# 제가 재밌는 문제를 가져왔어요

---

별로 재밌는 문제는 아닙니다 ㅎㅎ;

주어지는 날짜가 100만 범위이므로, 배열에 쉬는 날과 일하는 날을 0, 1로 마킹할 수 있습니다.

Two pointer, inch worm, sliding window 등으로 불리는 ‘그 방법’으로 휴가를 최대한 써가면서 쉬는 구간의 최대 길이를 구하면 됩니다.

휴가를 최대 100만일까지 쓸 수 있으므로, 배열은 200만 이상으로 넉넉하게 잡아야 함에 유의합니다.

# 제곱 가중치

---

고려대 ALPS  
김민성

Intermediate E  
Advanced B

# 제곱 가중치

---

- Prefix sum을 연쇄적으로 사용하여 문제를 해결한다.
- FFT로 풀어도 되지만, 그러면 실수 오차에 신경을 많이 써줘야 한다.

$$ans[x] = 2ps(ps(ps(a)))[x] - ps(ps(a))[x]$$

# 제곱 가중치

---

- 어떤 수열  $a$ 에 대해서  $ps(a)$ 를  $a$ 의 prefix sum이라고 정의하자.

$$a = [a[1], a[2], \dots, a[n]]$$

$$ps(a) = [a[1], a[1] + a[2], \dots, \sum_{i=1}^n a[i]]$$

# 제곱 가중치

---

- prefix sum을 3번 구하면 다음과 같다.
- $ps^i(a)$  는  $a$ 의  $i$ -차 prefix sum을 의미한다.

$$a = [a[0], a[1], \dots, a[x-1], \dots, a[n-1]]$$

$$ps(a) = [a[0], a[0] + a[1], \dots, \sum_{i=0}^{x-1} a[i], \dots, \sum_{i=0}^{n-1} a[i]]$$

$$ps^2(a) = [a[0], 2a[0] + a[1], \dots, \sum_{i=0}^{x-1} (x-i)a[i], \dots, \sum_{i=0}^{n-1} (n-i)a[i]]$$

$$ps^3(a) = [a[0], 3a[0] + a[1], \dots, \sum_{i=0}^{x-1} (1+2+\dots+(x-i))a[i], \dots, \sum_{i=0}^{n-1} (1+2+\dots+(n-i))a[i]]$$

$$= [a[0], 3a[0] + a[1], \dots, \sum_{i=0}^{x-1} \frac{(x-i)(x-i+1)}{2}a[i], \dots, \sum_{i=0}^{n-1} \frac{(n-i)(n-i+1)}{2}a[i]]$$

# 제곱 가중치

---

$$ps^3(a)[x] = \sum_{i=0}^{x-1} \frac{(x-i)(x-i+1)}{2} a[i]$$

$$ps^2(a)[x] = \sum_{i=0}^{x-1} (x-i) a[i]$$

$$2ps^3(a)[x] - ps^2(a)[x] = \sum_{i=0}^{x-1} (x-i)^2 a[i]$$

# 전염병을 피해라

---

고려대 Alkor  
이동관

Intermediate F

# 전염병을 피해라

---

다익스트라 알고리즘 2번을 통해 최단거리를 구하여 푸는 문제입니다.

- 핵심 아이디어:

- 병원과 인접한 모든 도시들에 대하여, 전염병보다 빠르게 도착할 수 있는 도시가 있다면 동관이는 해당 도시를 거쳐 병원에 도착할 수 있다. 이런 도시가 하나도 없다면 동관이는 병원에 갈 수 없다.
- 간단하게 증명해봅시다!
- 위의 명제를 2개의 명제로 나누어 보겠습니다.
  1. 동관이가 전염병보다 빠르게 도착할 수 있는 병원 인접 도시  $x$ 가 있다면 동관이가 집에서  $x$ 로 전염병을 만나지 않고 가는 경로가 반드시 존재한다.
  2. 이런 도시가 하나도 없다면 동관이는 전염병을 피해 병원에 갈 수 없다.

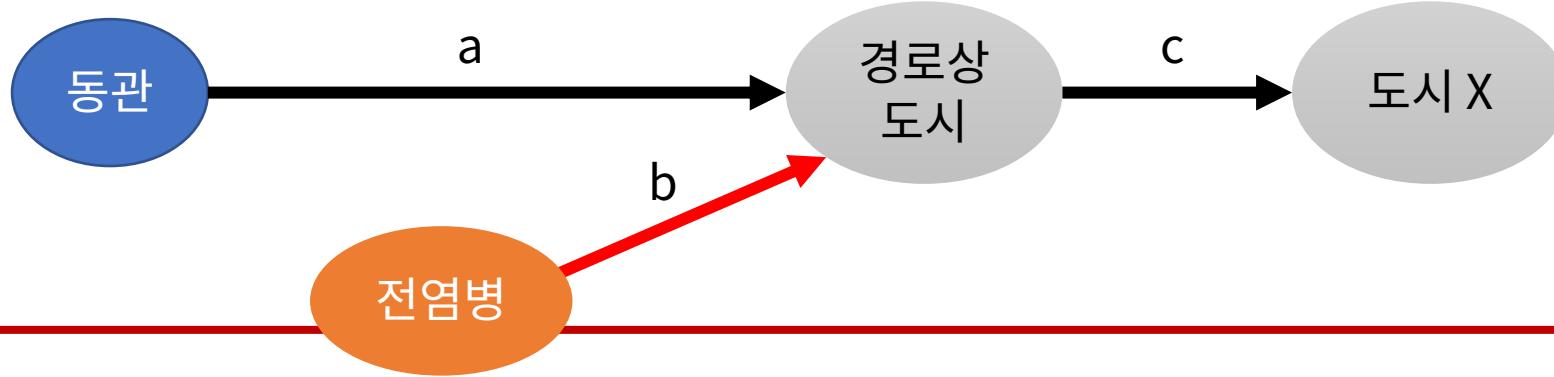
# 전염병을 피해라

1. “동관이가 전염병보다 빠르게 도착할 수 있는 병원 인접 도시 x가 있다면 동관이가 집에서 x로 전염병을 만나지 않고 가는 경로가 반드시 존재한다.” 부터 증명하겠습니다.

귀류법으로 생각해 봅시다.

경로가 존재하지 않는다는 말은, 동관이가 x로 가는 최단 경로에 전염병이 이미 도착했다는 말과 동치입니다.

하지만 이는 동관이가 전염병보다 빠르게 도착한다는 가정에 모순이므로 거짓이고, 증명되었습니다. ( $a \geq b$  이면  $a+c \geq b+c$ , 따라서 모순)



# 전염병을 피해라

---

2. “이런 도시가 하나도 없다면 동관이는 전염병을 피해 병원에 갈 수 없다.” 를 증명하겠습니다.

마찬가지로 귀류법으로 증명합니다.

동관이가 전염병을 피해 병원을 갈 수 있다면, 동관이는 병원에 인접한 어떤 도시  $x$ 가 봉쇄되기 전에  $x$ 를 들러 병원에 도착해야 합니다.

이는 가정에 모순이고, 따라서 증명되었습니다.

추가:

잘못 구현된 다익스트라가 인터넷이나 책에 많이 소개되어 있고,  
따라서 이를 저격하는 TLE 데이터를 넣어 두었습니다.

# 강남 건물주

---

한양대 ALOHA  
한경수

Intermediate G  
Advanced C

# 강남 건물주

---

- 기본적인 풀이 : 2차원 DP + 세그먼트 트리
- 2차원 DP를 이용해 각 층에 도달하기 위해 필요한 버튼의 클릭 횟수를 모두 구한 뒤, 최솟값을 저장하는 세그먼트 트리를 구현하여 각 구간에 대한 최솟값을 출력하면 된다.
- $DP[i][j] =$  마지막으로 올라가는 층수가  $2^j$ 가 되도록 하여(다음 번에 B버튼을 누르면  $2^{j+1}$  만큼 올라갈 수 있도록) i층까지 도달하는 버튼의 최소 클릭 횟수.

# 최소 대역폭 보장

---

고려대 Alkor  
오제노

Intermediate H  
Advanced D

# 최소 대역폭 보장

---

크게 2가지 풀이가 존재합니다.

1. 오프라인 쿼리 + 유니온 파인드 + 정렬
2. 최대 비용 신장 트리 + LCA

# 최소 대역폭 보장 - 첫 번째 풀이

---

- 1)  $Q = 1$ 일 경우
  - 주어진 쿼리를 A B C라고 하자.
  - M개의 간선 중에서 대역폭이 C 이상인 간선들만 남긴다.
  - 남은 간선들로 구성된 그래프에서 A와 B를 연결하는 경로가 있는가?
- 즉, 대역폭이 C 이상인 간선들의 두 끝점에 대해 union find를 진행했을 때, A와 B가 같은 집합인가?

# 최소 대역폭 보장 - 첫 번째 풀이

---

- 2) 쿼리들의 C 값이 모두 동일한 경우
  - M개의 간선 중에서 대역폭이 C 이상인 간선들만 남긴다.
  - 남은 간선들로 구성된 그래프에서 A와 B를 연결하는 경로가 있는가?
- 대역폭이 C 이상인 간선들의 두 끝점에 대해 union find를 진행했을 때 → A와 B가 같은 집합인가?
- 위의 질문을 Q번 하면 된다.

# 최소 대역폭 보장 - 첫 번째 풀이

---

- 1번과 2번처럼 문제를 해결하기 위해서는 어떻게 해야 할까?
  - 쿼리를 입력 받은 순서대로 처리한다면 → 쿼리에서 요구한 대역폭 이상인 간선들을 union find로 연결한 상태가 필요하다.
- $n$ 번째 질의 :  $A_n, B_n, C_n$
- $n + 1$ 번째 질의 :  $A_{n+1}, B_{n+1}, C_{n+1}$ 
  - Case 1)  $C_n < C_{n+1}$ 인 경우 :  $C_n$  이상  $C_{n+1}$  미만의 간선들을 모두 제거
  - Case 2)  $C_n = C_{n+1}$ 인 경우 : easy
  - Case 3)  $C_n > C_{n+1}$ 인 경우 :  $C_{n+1}$  이상  $C_n$  미만의 간선들을 모두 추가

# 최소 대역폭 보장 - 첫 번째 풀이

---

- Offline query
  - 질의를 받아서 순서대로 처리할 필요가 없다!
- Union find를 통해서 할 수 있는 것은?
  - 그래프에서 어떤 두 노드가 연결되어 있는가?
  - 간선 추가
- Union find가 할 수 없는 것은?
  - 간선 제거

# 최소 대역폭 보장 - 첫 번째 풀이

---

- $n$ 번째 질의 :  $A_n, B_n, C_n$
- $n + 1$ 번째 질의 :  $A_{n+1}, B_{n+1}, C_{n+1}$ 
  - Case 1)  $C_n < C_{n+1}$ 인 경우 :  $C_n$  이상  $C_{n+1}$  미만의 간선들을 모두 제거
  - Case 2)  $C_n = C_{n+1}$ 인 경우 : easy
  - Case 3)  $C_n > C_{n+1}$ 인 경우 :  $C_{n+1}$  이상  $C_n$  미만의 간선들을 모두 추가
- Union find로 처리할 수 있는 상태 : Case 2, 3
- Offline query이므로
  - 정렬을 통해서 case 2, 3의 상태만 있게 만들 수 있다!

# 최소 대역폭 보장 - 두 번째 풀이

---

- 간선들에 대해서 최대 비용 신장 트리를 구합니다.
  - Kruscal, Prim 알고리즘을 통해서 구할 수 있습니다.
- 구한 최대 비용 신장 트리를 T라고 정의합시다.
- T에서 쿼리 a, b, c를 다음과 같이 처리할 수 있습니다:
  - 정점 a, b의 LCA를 구한다.
  - LCA를 구하기 위한 sparse table 과, T에서 a와 b를 이어 주는 경로 위의 간선의 최소값을 구하기 위한 sparse table을 통해서 질의에 답이 가능.

# 최소 대역폭 보장 - 두 번째 풀이

---

- 풀이의 정당성
  - T에서 정점 a와 정점 b를 이어 주는 경로 위의 간선의 최소값  $T_{min}$
  - 과연 원래 그래프에서 정점 a와 정점 b를 이어 주는 최대 대역폭  $C_{max}$
  - 이 두 값이 일치할까?
- Kruscal, Prim 알고리즘의 정당성 증명하는 것과 유사하게
  - 귀납적으로 증명할 수 있습니다.
  - 특히 Kruscal 알고리즘의 경우, 풀이 1와 상당히 유사합니다.

# 최소 대역폭 보장 - 두 번째 풀이

---

- Kruscal 알고리즘(최대 비용 신장 트리)
  1. 간선을 비용 기준으로 내림차순 정렬.
  2. 트리 T는 처음에 공집합
  3. 간선들을 차례로 트리 T에 추가한다.
    1. 단, 간선을 추가하는 것으로서 cycle이 발생한다면, 간선 추가를 진행하지 않는다.
  4. 모든 간선에 대해 2.의 과정을 진행한다.
- Kruscal 알고리즘에서
  - 간선  $(a, b, c)$ 을 추가하는 것으로서 cycle이 발생한다는 것은
  - $T$ 에  $a$ 와  $b$ 를 이어주는 경로가 존재한다는 것이다.
  - 한편,  $T$ 에 존재하는 간선들은 전부 가중치가  $c$  이상이다.
  - 즉,  $a$ 와  $b$ 를 이어주는 최대 대역폭은 제거된 간선  $(a, b, c)$ 와 관련이 없다.

# 최소 대역폭 보장 - 두 번째 풀이

---

- <귀류법> 어떤 정점 쌍  $(a, b)$ 에 대하여,  $T_{min} \neq C_{max}$  인 경우가 존재한다면
  - 1)  $T_{min} > C_{max}$ 
    - T에 있는 모든 간선들은 C에 존재하므로,
    - T에서 정점 a와 정점 b를 이어 주는 경로 위의 간선들은 C에도 존재  $\rightarrow$  모순
  - 2)  $T_{min} < C_{max}$ 
    - C에서 정점 a와 정점 b를 이어 주는 어떤 경로 X가 존재하여,
    - X 위의 모든 간선들의 가중치는  $C_{max}$  이상입니다.
    - 여기에서, kruscal algorithm의 작동원리를 생각해 본다면,
    - T는 경로 X 위의 임의의 간선 e에 대해서 다음 두 가지 중 한 가지는 만족해야 한다.
      - 1. 간선 e가 T 위에 존재한다.
      - 2. 간선  $e = (e_a, e_b, e_c)$ 라면,  $e_a$  와  $e_b$  를 이어 주는 경로가 존재하여야 한다.

# 최소 대역폭 보장 - 두 번째 풀이

---

- 즉, T는 경로 X 위의 임의의 간선  $e = (e_a, e_b, e_c)$ 에 대해서
  - $e_a$ 와  $e_b$ 를 이어 주는 대역폭  $e_c$ 를 감당할 수 있는 경로가 존재한다.
  - $e_c \geq C$  이므로, 이는 다음과 같이 쓸 수 있습니다.
  - $e_a$ 와  $e_b$ 를 이어 주는 대역폭 C를 감당할 수 있는 경로가 존재한다.
- 경로 X를  $(v_1 - v_2 - \dots - v_k)$ 라고 둔다면, 다음이 성립합니다.
- $v_i$ 와  $v_{i+1}$ 을 이어 주는 대역폭 C를 감당할 수 있는 경로가 존재한다.

# 최소 대역폭 보장 - 두 번째 풀이

---

- $(V, W), (W, Y)$ 를 각각 이어 주는 대역폭 C 이상의 경로가 존재한다
  - $\rightarrow (V, Y)$  를 각각 이어 주는 대역폭 C 이상의 경로가 존재합니다.
  - 증명)  $V \rightarrow W, W \rightarrow Y$  경로를 더해 주면 됩니다.
- $v_i$ 와  $v_{i+1}$ 을 이어 주는 대역폭 C를 감당할 수 있는 경로의 존재성
- 이 두 가지를 합하면 트리 T에서 정점 a와 정점 b를 이어 주는 대역폭 C를 감당할 수 있는 경로의 존재성을 증명할 수 있습니다.

# 최소 대역폭 보장 - 두 번째 풀이

---

- Prim algorithm으로 구한 최대 비용 신장 트리의 정당성
- Prim algorithm
  - 1. 그래프  $G=(V, E)$
  - 2.  $V_1 = \{x\}$ , x는  $V$ 에 있는 임의의 꼭지점.  $E_1 = \{\emptyset\}$
  - 3.  $V_1 = V$ 가 될 때까지 다음을 계속 수행:
    - $E$ 에서 최대 가중치를 가진 변  $(u, v)$ 를 뽑아낸다.
    - 단,  $u$ 는  $V_1$ 에 속해야 하고,  $v$ 는  $V_1$ 에 속하지 말아야 한다.
    - $v$ 를  $V_1$ 에 추가하고,  $E_1$ 에  $(u, v)$ 를 추가한다.
  - 4. 알고리즘이 끝났을 때 만들어진 트리  $G=(V_1, E_1)$ 이 최대 비용 신장 트리

# 최소 대역폭 보장 - 두 번째 풀이

---

- $V_1$ 의 크기에 대해서 귀납법으로 정당성을 증명합니다.
  - 편의상  $V_1 = \{v_1, v_2, \dots, v_n\}$ ,  $v_i$ 는 i번째로  $V_1$ 에 추가된 정점으로 정의
  - 1)  $|V_1|=1 \rightarrow$  성립
  - 2)  $|V_1|=2 \rightarrow v_1$ 과  $v_2$ 를 이어 주는 임의의 경로는  $v_1, x$ 를 간선으로 포함
    - 그러나,  $(v_1, x)$  간선의 가중치  $\leq (v_1, v_2)$  간선의 가중치  $\rightarrow$  성립
  - 3)  $|V_1| = k \rightarrow$  성립한다고 가정
  - 4)  $|V_1| = k+1$ 
    - $1 \leq i < j \leq k$ 에 대해서,  $E_1$  내부에서  $v_i$ 와  $v_j$ 을 이어 주는 임의의 경로는 전체 그래프에서 최적입니다.
    - $1 \leq i \leq k$ 에 대해서,  $E_1$  내부에서  $v_i$ 와  $v_{k+1}$ 을 이어 주는 임의의 경로가 전체 그래프에서도 최적임을 증명하면 됩니다.

# 최소 대역폭 보장 - 두 번째 풀이

---

- $|V_1| = k$ 에서  $|V_1| = k + 1$ 로 바뀔 때,
- $v_g$ 와  $v_{k+1}$ 를 이어 주는 간선이 추가되었다고 정의합시다.
- $|V_1|=2$ 일 때와 마찬가지로, 전체 그래프에서  $v_i$ 와  $v_{k+1}$ 을 이어 주는 임의의 최적의 경로  $R$ 이 존재한다고 합시다.
- $R = \{R_1, R_2, \dots, R_r\}$ ,  $R_1 = v_{k+1}$ ,  $R_r = v_i$ 라고 둔다면,
- $R_1$ 은  $|V_1|=k$  상태의  $V_1$ 에 포함 되지 않고,
- $R_r$ 은  $|V_1|=k$  상태의  $V_1$ 에 포함 됩니다.
- mn을  $|V_1|=k$  상태의  $V_1$ 에 포함되지 않는 최대의 자연수로 정의한다면

# 최소 대역폭 보장 - 두 번째 풀이

---

- 1.  $R_{mn+1}$ 과  $R_r$ 을 이어 주는 최적의 경로는  $|V_1|=k$  상태의  $E_1$ 내부의 간선들로 구성된 경로만으로 충분합니다.
- 2.  $R_{mn}$ 과  $R_{mn+1}$ 을 이어 주는 최적의 경로 위의 간선의 가중치는  $v_g$ 와  $v_{k+1}$ 을 이어 주는 간선의 가중치보다 작거나 같습니다.
- 즉, 이를 결합하면 경로 R의 최대 대역폭은  $|V_1|=k+1$  상태의  $E_1$ 내부의 간선들로 구성된 경로만으로 충분하다는 사실을 확인할 수 있습니다.

# 최소 대역폭 보장 - 두 번째 풀이

---

- LCA → sparse table
  - LCA[a][b] : a의 ( $1 \ll b$ ) 번째 조상의 노드 번호
  - MIN\_NODE[a][b] : a와 LCA[a][b]를 연결하는 최대 대역폭
- $LCA[a][b] = LCA[LCA[a][b-1]][b-1]$
- $MIN\_NODE[a][b] =$
- $\min(MIN\_NODE[a][b-1], MIN\_NODE[LCA[a][b-1]][b-1])$
- 위의 두 가지 sparse table로 구할 수 있습니다.

# 최소 대역폭 보장

---

- 풀이 1의 시간복잡도
  - $O(Q \log Q + M \log M)$
- 풀이 2의 시간복잡도
  - $O(Q \log N + M \log M)$
- 풀이 1, 2 모두 제 시간 안에 나옵니다!

# 오렌지 수거

---

고려대 Alkor  
김태훈

Intermediate I

# 오렌지 수거 - 관찰 1

---

- 어떤 노드에서, 오렌지가 몇 개 있든지
  - 짹수 개 있으면, 오렌지가 없는 것과 동일하다.
  - 홀수 개 있으면, 오렌지가 1개 있는 것과 동일하다.
- 왜냐하면, 짹수 개의 오렌지는 0의 힘으로 없앨 수 있다.

# 오렌지 수거 - 관찰 2

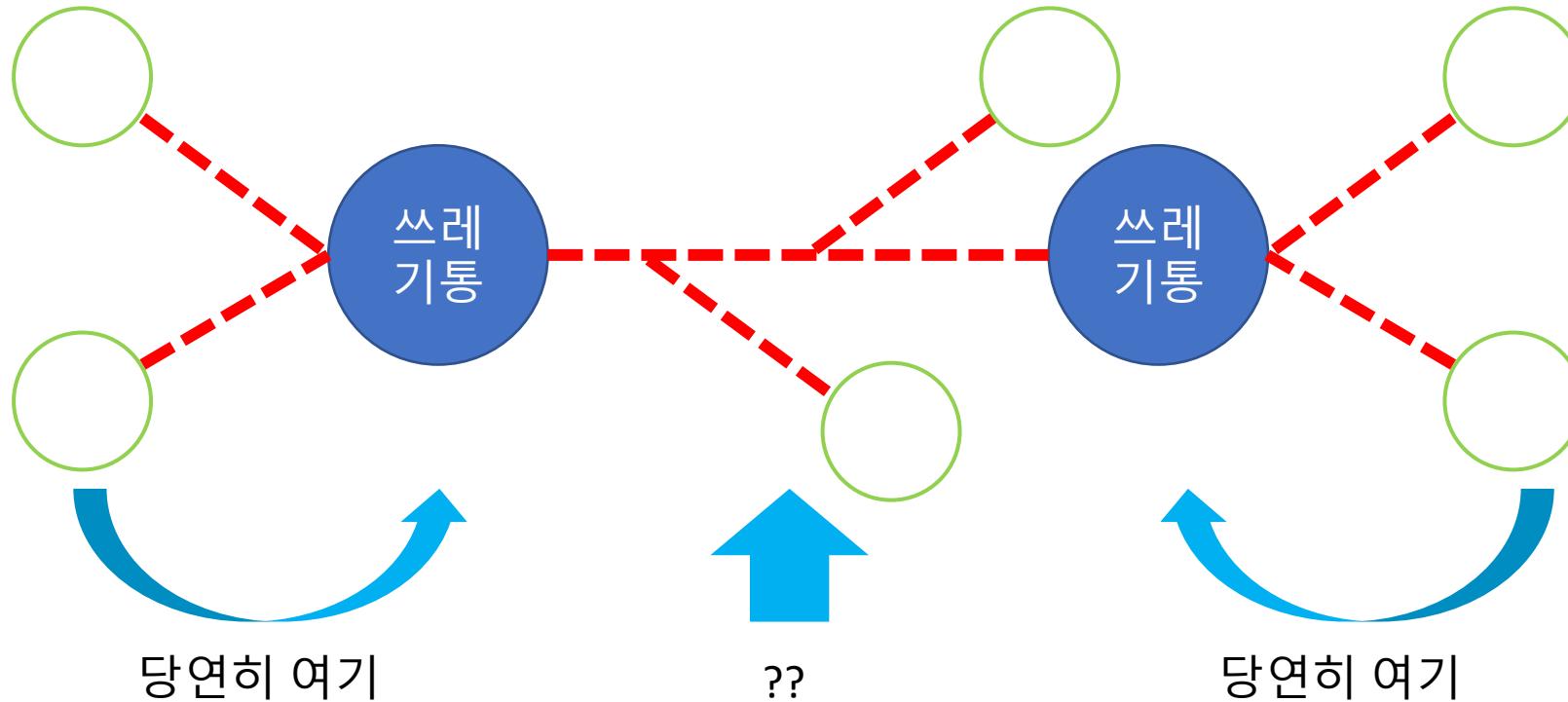
---

- 오렌지를 쓰레기통으로 옮길 때, 항상 최단경로를 따라 움직이는 것이 최선이다.
  - 이 때 하나씩 옮기는 것이 아니라, 각 노드에서 subtree의 오렌지가 모두 모일 때까지 기다렸다가 옮긴다.
  - 그러면 subtree에서 모인 오렌지가 홀수 개이면 1개로 취급, 짝수 개이면 없앨 수 있다.
- 다른 오렌지가 있는 노드로 옮겨서 짝수 개로 만든다면?
  - 어차피 최소 공통 조상으로 모으는 것으로 위의 방법과 동치이다.

# 오렌지 수거 - 관찰 3

---

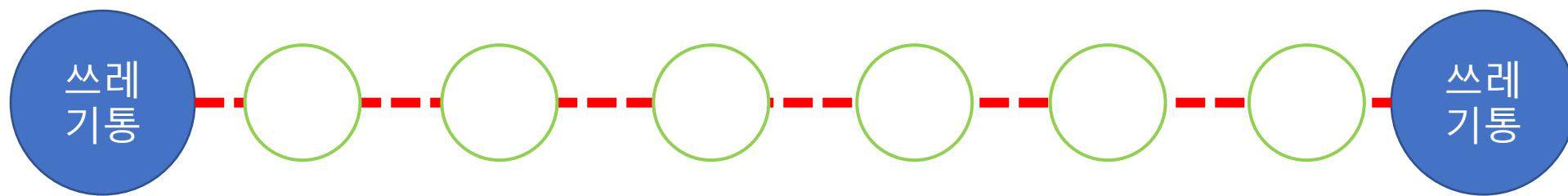
- 2 개의 쓰레기통 중 어느 곳으로 가야 할까?



# 오렌지 수거 - 관찰 4

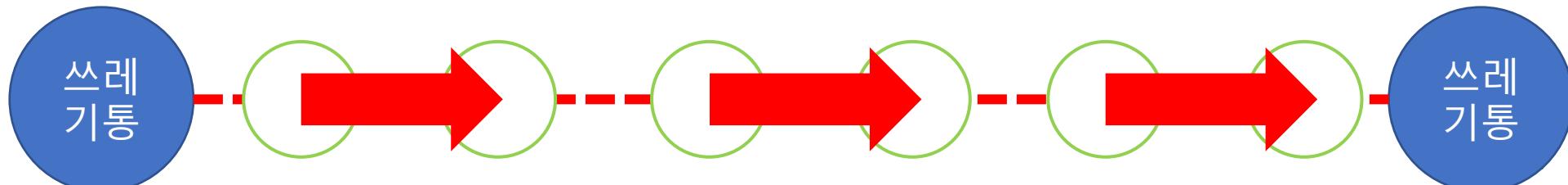
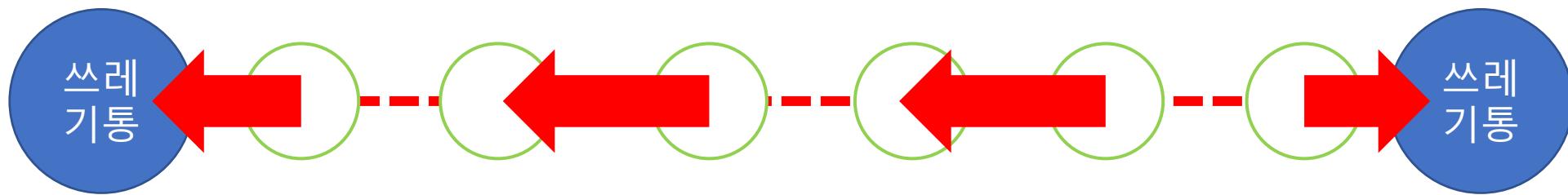
---

- 우선 두 쓰레기통을 연결하는 경로 사이에 놓인 노드까지는 오렌지를 모아야 한다.
- 오렌지가 홀수 개(1개)인 노드만 중요하기 때문에, 이러한 노드만 살리고 경로를 다시 그릴 수 있다.



# 오렌지 수거 - 관찰 5

- 그러면 다음 두 방법 중 하나가 최선이다.
  - 중간 노드 개수의 홀짝은 중요하지 않습니다.
  - 첫 노드에서 왼쪽으로 가는지 오른쪽으로 가는 지로 나뉩니다.



# 오렌지 수거

---

- 쓰레기통 사이 경로 찾기 → DFS 1회
- 무조건 양 끝 쓰레기통으로 들어가는 오렌지를 넣기 → DFS 2회 (합쳐서  $O(N)$ )
- 쓰레기통 사이에 대해, 경로 상의 노드로 모으기 → DFS ?회 (합쳐서  $O(N)$ )
- 두 가지 방법을 해보고 비교 →  $O(N)$
  
- 따라서  $O(N)$ 의 시간복잡도로 해결 가능!
  - 그러나 시간제한은 5초로  $O(N \log N)$ 도 통과할 수 있습니다.

# 동굴의 입구를 열어라

---

고려대 Alkor  
오제노

Advanced E

# 동굴의 입구를 열어라

---

- 아래와 같은 조건을 찾는 것이 핵심이었습니다.

$a_1x_1 + a_2x_2 + \cdots + a_Nx_N = M$  이면서,  $x_1 + x_2 + \cdots + x_N$ 이 최소가 되는

$(x_1, x_2, \dots, x_N)$  이 존재한다면, 아래와 같은 조건을 만족합니다.

$$x_1 < a_2, x_2 < a_3, \dots, x_{N-1} < a_N$$

# 동굴의 입구를 열어라

---

만일 어떠한  $1 \leq i \leq N - 1$ 에 대해,  $a_{i+1} \leq x_i$  라고 가정해 봅시다.

그렇다면,  $a_i x_i + a_{i+1} x_{i+1} = a_i(x_i - a_{i+1}) + a_{i+1}(x_{i+1} + a_i) \geq 0$ 이므로,

$(x_1, x_2, \dots, x_i - a_{i+1}, x_{i+1} + a_i, \dots, x_n)$  순서쌍에 대해서,

$$a_1 x_1 + a_2 x_2 + \cdots + a_i(x_i - a_{i+1}) + a_{i+1}(x_{i+1} + a_i) + \cdots + a_N x_N = M,$$

$$x_1 + x_2 + \cdots + (x_i - a_{i+1}) + (x_{i+1} + a_i) + \cdots + x_N =$$

$$x_1 + x_2 + \cdots + x_N + (a_i - a_{i+1}) < x_1 + x_2 + \cdots + x_N$$

즉,  $x_1 + x_2 + \cdots + x_N \geq 0$ 이 최소임에 모순입니다.

그러므로,  $x_1 < a_2, x_2 < a_3, \dots, x_{N-1} < a_N$ 이 성립합니다.

# 동굴의 입구를 열어라

---

그렇다면, 가능한  $(x_1, x_2, \dots, x_{N-2})$  쌍에 대해서, 즉  $a_2 \times a_3 \times \dots \times a_{N-1}$  개의 순서쌍에 대해서만 시행해 보면, 모든  $x_1 + x_2 + \dots + x_N$ 이 최소가 되는  $(x_1, x_2, \dots, x_N)$  후보군을 다 시행해 볼 수 있다는 것입니다.

$x_{N-1} a_{N-1} + x_N a_N = Y$ 이면서,  $x_{N-1} + x_N$ 이 최소가 되게 하는 것은 매우 쉽습니다.

1)  $\gcd(a_{N-1}, a_N)$  이  $Y$ 의 약수가 아닌 경우

불가능.

2)  $\gcd(a_{N-1}, a_N)$  이  $Y$ 의 약수인 경우

Let  $(b_{N-1}, b_N) = (a_{N-1}/\gcd(a_{N-1}, a_N), a_N/\gcd(a_{N-1}, a_N))$ ,  $Z = Y/\gcd(a_{N-1}, a_N)$

$$x_{N-1} b_{N-1} + x_N b_N = Z$$

$$x_{N-1} \equiv Z(b_{N-1})^{-1} (\text{mod } b_N)$$

$(b_{N-1})^{-1} (\text{mod } b_N)$  즉  $b_N$ 에 대한  $b_{N-1}$ 의 잉여약수를 구하는 것은 사전 작업으로 하면 됩니다.

# 동굴의 입구를 열어라

---

문제 조건에서,  $a_2 \times a_3 \times \dots \times a_{N-1} \leq 3 \times 10^7$  이라고 명시되어 있습니다.

따라서, 시간 복잡도는  $O(3 \times 10^7)$  가 됩니다.

# 동굴의 입구를 열어라

---

- 메모리가 8MB이기 때문에 순수한 DP로는 순서쌍 개수까지는 맞추기 힘들었을 것이라 예상합니다.
- N=2, 3일 때를 해 보셨다면, 충분히 풀이의 실마리를 잡으실 수 있을 것이라고 기대하였습니다.

# 숭고한 아이스크림

---

고려대 Alkor  
김태훈

Advanced F

# 승고한 아이스크림 - 관찰 1

---

- 지금 먹은 아이스크림의 쿨타임은,
  - 더 나중에 먹는 아이스크림의 맛에만 영향을 준다.
- 가장 마지막에 먹은 아이스크림의 쿨타임은, 어떤 것에도 영향을 주지 않는다.

# 숭고한 아이스크림 - 관찰 2

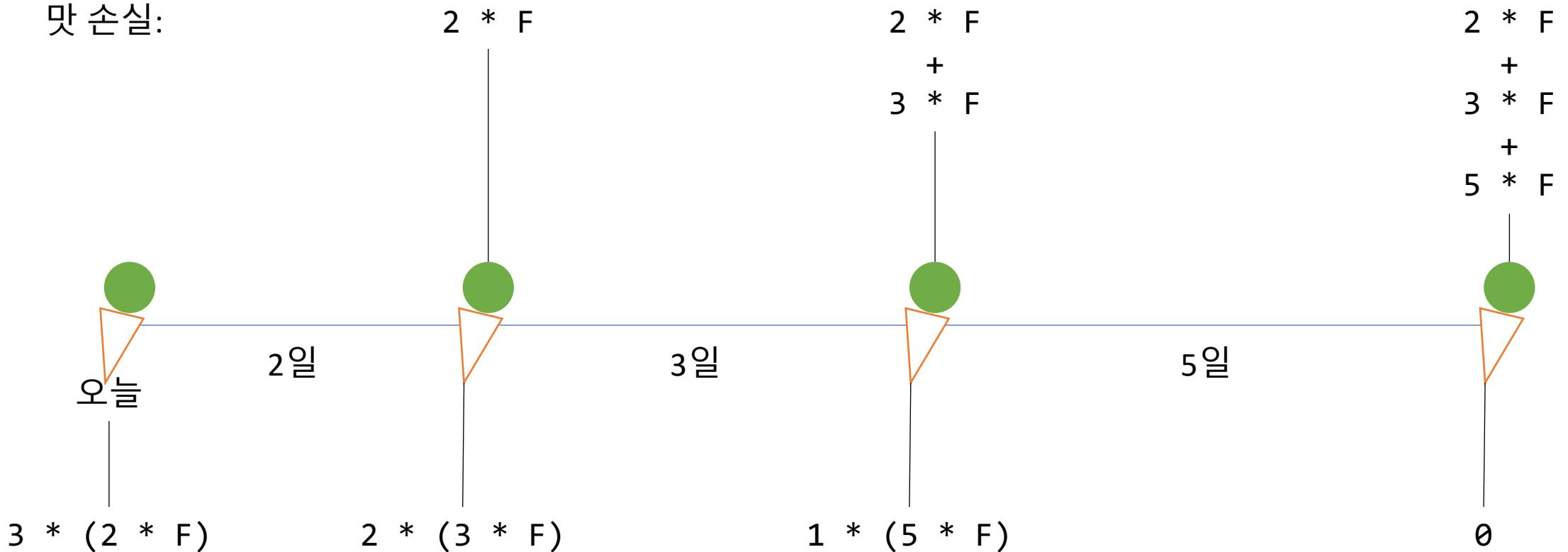
---

- 지금 먹은 아이스크림의 쿨타임이 C이고, 그 뒤에 아이스크림을 R개 먹었다고 하면,
- 지금 아이스크림이 떨어뜨린 맛의 총 합은  $R * (C * F)$  이다.
- 음?

# 승고한 아이스크림 - 관찰 2 (그림)

---

맛 손실:



# 승고한 아이스크림 - 관찰 3

---

- 따라서, 뒤에 R개의 아이스크림이 남아있다면,
  - 초기맛이 D이고 쿨타임이 C인 아이스크림에 대해,
  - 얻을 수 있는 맛은  $D - R * (C * F)$  이다.
- 
- 즉  $R = 0 \sim M - 1$  범위를 돌면서, 각각 R에 대해,
  - 위 식의 값이 최대인 아이스크림을 골라 먹으면 된다.
  - $\rightarrow O(MN)$

# 승고한 아이스크림 - 관찰 4

---

- $D - R * (C * F)$
- $y = (-C * F) * R + D$
- $y = a * x + b$
- 각  $x$ 에 대해 최대의  $y$ 값이 궁금하다.
- Convex Hull이 궁금하다.
  - Convex Hull 생성에  $O(N \log N)$ , 아이스크림 선택에  $O(M + N)$

# 숭고한 아이스크림

---

- 아이스크림을 입력받고 직선을 생성, Convex Hull 만들기
  - $O(N \log N)$
- $x = 0 \sim M - 1$  의 범위 안에서 Convex Hull의 값 더하기
  - Convex Hull이 음수가 되면, 더 이상 먹지 않고 break!
  - $O(M + N)$  : 직선 하나씩 보면서 교점이 범위 안에 있는지 보면 된다.
- 따라서 시간복잡도는  $O(N \log N + M)$

# 수열 복원

---

숭실대 SCCC  
김도현

Advanced G

# 수열 복원

---

- $a_i + a_j = k$ 는 “ $a_i$ 값을 알면  $a_j$ 값도 알 수 있고,  $a_j$ 값을 알면  $a_i$ 값도 알 수 있다”는 의미에서 정점i와 정점j를 가중치 c의 간선으로 연결한 그래프를 생각해봅시다.
- 이 그래프가 연결그래프라면, 한 정점의 값을 알 때 나머지 모든 정점의 값도 알 수 있습니다. 한 정점의 값은 약간의 선형대수적 직관을 활용하면, 홀수사이클로부터 구할 수 있습니다. 홀수사이클의 존재성은 그래프의 이분성과 동치이므로 dfs를 통해 쉽게 할 수 있습니다.
- 연결그래프가 아닐 때도 위의 전략을 확장하여 적용할 수 있습니다. 모든 컴포넌트에 홀수사이클이 존재하면 답을 구할 수 있고, 아니면 -1을 출력해야 합니다.

# 수열 복원 - 증명

---

그래프에 대해, 각 행은 한 간선을 나타내며, 각 행마다 간선이 연결하는 두 정점  $u, v$  위치는 1이고 아니면 0인  $e^*v$  행렬을 생각해봅시다. 이제 그래프에 대해 rank나 span등의 이야기를 할 수 있습니다. (NOTE: 이러한 행렬은 문제의 ‘기록’을 연립방정식으로 표현했을때와 같습니다.)

그래프를 임의의 스패닝 트리와 잔여간선으로 분해해 봅시다. (스패닝트리+잔여간선 1개)는 사이클을 만들며, 다음과 같은 성질이 있습니다.

- 크기  $c$ 의 사이클의 rank는 짹수사이클일때  $c-1$ , 홀수사이클일때  $c$ 입니다.
- 스패닝트리의 rank는  $v-1$ 입니다.
- 스패닝트리는 그 사이클의 서로 독립인  $c-1$ 개 간선을 포함합니다.

따라서 (스패닝트리+잔여간선1개)의  $\text{rank} = (\text{스패닝트리의 rank}) + (\text{사이클의 rank}) - (\text{교집합 rank})$ = 짹수사이클일때  $v-1$ , 홀수사이클일때  $v$ 입니다.

- 홀수사이클이 있다면  $\text{rank}=v$ 이고, 답은 유일해집니다. ( $\because$  미지수가  $v$ 개인 연립방정식의 행렬표현이 full rank임)
- 홀수사이클이 없다면  $\text{rank}=v-1$ 이므로 답이 유일하지 않습니다.

# 수열 복원 - 증명

---

full rank여도 모순이면 답이 존재하지 않을 수 있는데, “모순되는 입력은 없다”는 조건에 의해 그런 경우는 없습니다.

한가지 남은 경우의 수, 잔여간선끼리 span한 결과가 스패닝트리와 독립인 새로운 벡터를 만들 수 있는지 생각해보아야 하는데, 다음과 같이 생각하면 불가능함을 알 수 있습니다.

$a \in \text{span}(\text{basis}) \wedge b \in \text{span}(\text{basis}) \Rightarrow \text{span}(\{a,b\}) \subset \text{span}(\text{basis})$  이다. 이를 다시쓰면  
a가 basis에 종속  $\wedge$  b가 basis에 종속  $\Rightarrow \text{span}(\{a,b\})$ 는 basis에 종속이므로, 잔여간선들만으로  
스패닝트리와 독립인 벡터는 만들 수 없습니다.

QED

# 반사복제된 트리

---

고려대 Alkor  
김민성

Advanced H

# 반사복제된 트리 - Brute force?

---

- 만약 이 문제를 brute force로 풀게 된다면… 트리를 직접 k번 반사복제한 후, 다시 그 트리의 vertex의 개수에 대한 Quadratic한 DFS를 돌려야 하기 때문에.. (Cubic으로 더 심하게 느리게 풀 수도 있다)

$$\mathcal{O}(n^2 |L|^{2^k})$$

# 반사복제된 트리 - 풀이 요약

---

Step 1: 트리의 리프가 아닌 노드를 루트로 정하고, 초기 트리의 3가지 방식의 거리를 계산한다. (DP 사용,  $n = 2$ 는 예외처리)

$$D_{LL} = \sum_{l_1 \in L} \sum_{l_2 \in L} dist(l_1, l_2) \quad (|L|, |V|, D_{LL}, D_{LA}, D_{AA}) \rightarrow$$

$$D_{LA} = \sum_{l_1 \in L} \sum_{v_1 \in V} dist(l_1, v_1) \quad (|L|^*, |V|^*, D_{LL}^*, D_{LA}^*, D_{AA}^*)$$

$$D_{AA} = \sum_{v_1 \in V} \sum_{v_2 \in V} dist(v_1, v_2)$$

- L은 모든 리프의 집합이다.
- V는 모든 노드의 집합이다.
- 이 문제는 오직 DP만 빡세게 사용하는 문제이다.

# 반사복제된 트리 - 풀이 Step 1-1

---

- $C(v) = v$ 의 자식노드의 집합
- $UL(v) = v$ 의 서브트리에 있는 리프의 개수(본인 포함)
- $UA(v) = v$ 의 서브트리에 있는 모든 노드의 개수(본인 포함)

$$UL(l)_{l \in L} = 1, \quad UL(v)_{v \notin L} = \sum_{c \in C(v)} UL(c)$$

$$UA(v) = 1 + \sum_{c \in C(v)} UA(c)$$

# 반사복제된 트리 - 풀이 Step 1-1

---

- $DSL(v) = v$ 의 서브트리에 있는 모든 리프에서부터  $v$ 까지 가는 거리의 합
- $DSA(v) = v$ 의 서브트리에 있는 모든 노드에서부터  $v$ 까지 가는 거리의 합

$$DSL(l)_{l \in L} = 0, \quad DSL(v)_{v \notin L} = UL(v) + \sum_{c \in C(v)} DSL(c)$$

$$DSA(l)_{l \in L} = 0, \quad DSA(v)_{v \notin L} = UA(v) - 1 + \sum_{c \in C(v)} DSA(c)$$

- 각 수식에  $UL$ ,  $UA$ 를 더해주는 이유는 각 시작점 노드들이 기존에 머물렀던 child 노드에서 1칸씩을 더 올라와야 하기 때문이다.  $UA$ 는 자기 자신을 항상 포함하기 때문에 1을 빼준다.

# 반사복제된 트리 - 풀이 Step 1-2

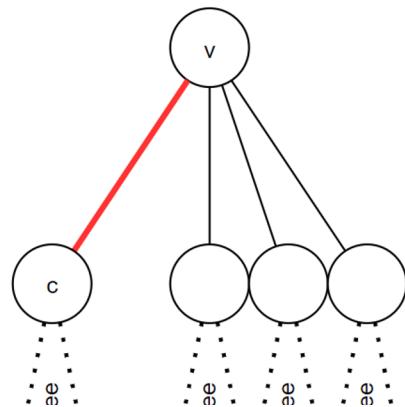
---

- 모든 노드  $v_0$ 에 대해서,  $v_0$ 를 지나가면서 동시에  $v_0$ 를 가장 높은 degree vertex로 갖는 임의의 경로  $v_1 - v_2$ 는 다음과 같은 두 종류 중 하나이다.
  - $v_0$ 가 해당 경로의 시작점/종착점이 아닌 경우. (경로 유형 1)
  - $v_0$ 가 해당 경로의 시작점/종착점인 경우. (경로 유형 2)
- 만약 트리의 루트노드를 리프노드가 아닌 노드로 잡는다면, 트리 DP를 할 때 (적어도 출제자의 접근에서는) D\_LL은 경로 유형 2에 대해서 고민할 필요가 없다.
  - $n=2$  인 경우,  
 $k$ 를 1 감소시키고 대신 기존의 트리를 노드가 4개인 일자형 트리로 대체한다.

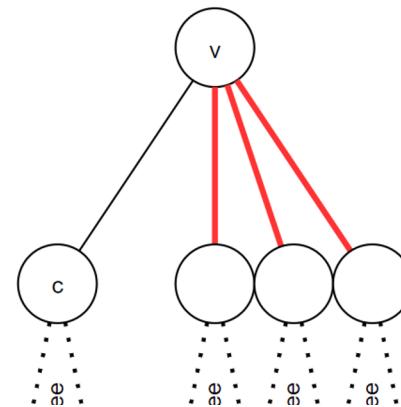
# 반사복제된 트리 - 풀이 Step 1-2

---

- D\_LL, D\_LA, D\_AA 경로 유형 1 계산의 핵심 아이디어: 각 노드  $v$ 와 그의 자식노드  $c$ 에 대해서 child를 시작점으로 잡고  $v$ 를 중간지점으로 삼아서 전반부 거리(child 서브트리의 시작점들부터  $v$  까지 오는 거리)와 후반부 거리( $v$ 부터 child 제외 서브트리들의 끝점들까지 가는 거리)를 따로 계산한다.
- D\_LL, D\_LA의 경우 경로 유형 2에 대한 계산도 따로 해줘야 한다.



전반부 거리



후반부 거리

# 반사복제된 트리 - 풀이 Step 1-2

---

- 큰 괄호 안의 윗줄은 전반부 거리, 아랫줄은 후반부 거리이다. 모든 수식은 “해당 거리들을 지나갈 노드의 개수” X “거리들의 거리의 총합” 형태이다.
- $c^C$ 는  $c$ 의 형제 노드들의 집합을 의미한다. 편의상  $f(\text{set})$ 은 set 내부의 모든 원소  $x$ 에 대해서  $f(x)$ 의 합으로 정의한다.

$$\begin{aligned} D_{LL} &= \sum_{v \in V} \sum_{c \in C(v)} \left( \begin{array}{l} UL(c^C)(DSL(c) + UL(c)) \\ UL(c)(DSL(c^C) + UL(c^C)) \end{array} \right) \\ &= \sum_{v \in V} \sum_{c \in C(v)} \left( \begin{array}{l} (UL(v) - UL(c))(DSL(c) + UL(c)) \\ UL(c)(DSL(v) - DSL(c) - UL(c)) \end{array} \right) \end{aligned}$$

# 반사복제된 트리 - 풀이 Step 1-2

---

- 큰 괄호 안의 윗줄은 전반부 거리, 아랫줄은 후반부 거리이다. 모든 수식은 “해당 거리들을 지나갈 노드의 개수” X “거리들의 거리의 총합” 형태이다.
- $c^C$ 는  $c$ 의 형제 노드들의 집합을 의미한다. 편의상  $f(\text{set})$ 은 set 내부의 모든 원소  $x$ 에 대해서  $f(x)$ 의 합으로 정의한다.
- 수식 중간의  $DSL(v)$  항은 경로 유형 2를 계산한 값이다.

$$\begin{aligned} D_{LA} &= \sum_{v \in V} \left( DSL(v) + \sum_{c \in C(v)} \left( \begin{array}{l} UA(c^C)(DSL(c) + UL(c)) \\ UL(c)(DSA(c^C) + UA(c^C)) \end{array} \right) \right) \\ &= \sum_{v \in V} \left( DSL(v) + \sum_{c \in C(v)} \left( \begin{array}{l} (UA(v) - UA(c) - 1)(DSL(c) + UL(c)) \\ UL(c)(DSA(v) - DSA(c) - UA(c)) \end{array} \right) \right) \end{aligned}$$

# 반사복제된 트리 - 풀이 Step 1-2

---

- 큰 괄호 안의 윗줄은 전반부 거리, 아랫줄은 후반부 거리이다. 모든 수식은 “해당 거리들을 지나갈 노드의 개수” X “거리들의 거리의 총합” 형태이다.
- $c^C$ 는  $c$ 의 형제 노드들의 집합을 의미한다. 편의상  $f(\text{set})$ 은 set 내부의 모든 원소  $x$ 에 대해서  $f(x)$ 의 합으로 정의한다.
- 수식 중간의  $DSA(v)$  항은 경로 유형 2를 계산한 값이다.

$$\begin{aligned} D_{AA} &= \sum_{v \in V} \left( 2DSA(v) + \sum_{c \in C(v)} \left( \begin{array}{l} UA(c^C)(DSA(c) + UA(c)) \\ UA(c)(DSA(c^C) + UA(c^C)) \end{array} \right) \right) \\ &= \sum_{v \in V} \left( 2DSA(v) + \sum_{c \in C(v)} \left( \begin{array}{l} (UA(v) - UA(c) - 1)(DSA(c) + UA(c)) \\ UA(c)(DSA(v) - DSA(c) - UA(c)) \end{array} \right) \right) \end{aligned}$$

# 반사복제된 트리 - 풀이 Step 2-1

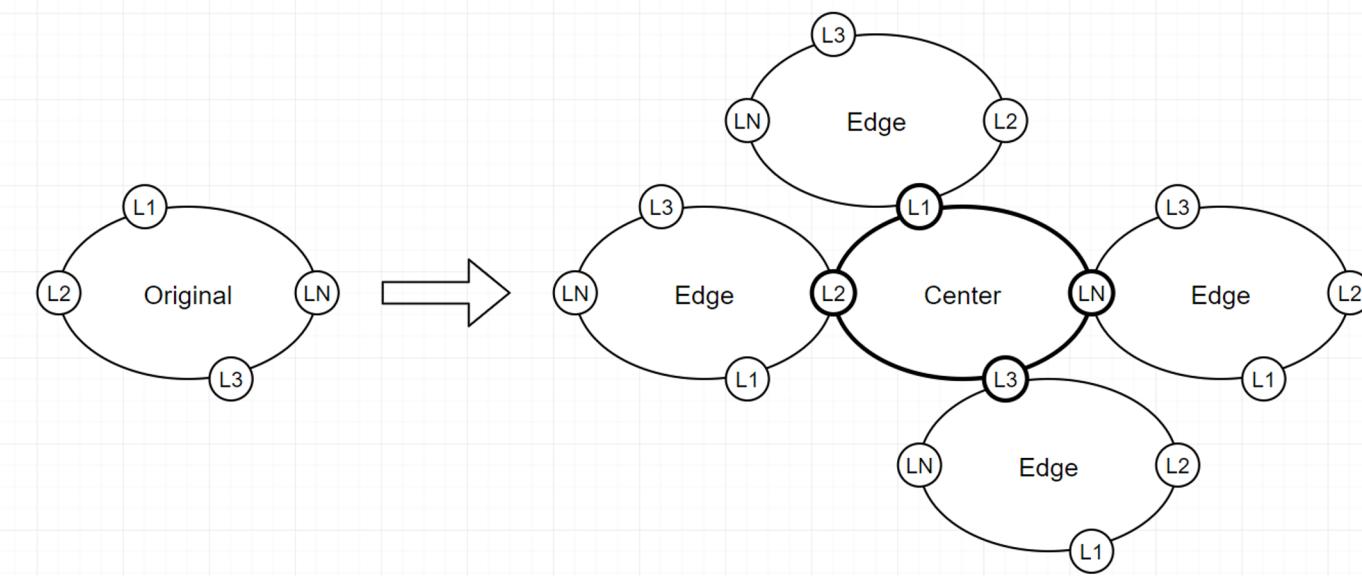
---

- 이제 우리는 반사복제를 하지 않은 초기 트리에 대한  $D_{LL}$ ,  $D_{LA}$ ,  $D_{AA}$ 를 구했다. 그런데 우리는 왜  $D_{AA}$ 만 구하지 않고 3가지 거리를 구했는가?
  - 이는 반사복제 과정이 리프노드를 통해서만 이루어지기 때문이다. 그래서 거리 계산식들을 리프 노드 oriented하게 짜면 비교적 수월하게  $D_{AA}$  값을 구할 수가 있다. (물론 이 수식들도 복잡하고, linear 하지도 않아서 행렬로 풀 수 없다.)
- 지금부터  $|L|$  (트리의 리프의 개수),  $|V|$  (트리의 모든 노드의 개수),  $D_{LL}$ ,  $D_{LA}$ ,  $D_{AA}$ 로 이루어진 5중 점화식을 짤 것이다.  
그 형태는 “ $x$ 번 반사복제된 트리의 성질  $\rightarrow (x+1)$ 번 반사복제된 트리의 성질”이다.

# 반사복제된 트리 - 풀이 Step 2-1

---

- 핵심 아이디어: 다음 그림은 트리가 반사복제하는 모습을 나타낸 것이다. 반사복제된 트리는 오리지널 파트에 해당하는 “Center Tree”와 그 곁가지로 형성된 “Edge Tree”로 영역을 구분할 수 있다.
- 추후 D\_LL, D\_LA, D\_AA는 각각의 경로를 Edge 또는 Center / Edge-Center / Edge-Center-Edge 3가지 형태로 따로 분리해서 계산하여 더한다.



# 반사복제된 트리 - 풀이 Step 2-1

---

- 또한,  $v_1$ 과  $v_2$ 가 같은 노드이면  $\text{dist}(v_1, v_2) = 0$ 이라는 점을 이용하여 다음 수식을 유추해낼 수 있다.

$$D_{LL} = \sum_{l_1 \in L} \sum_{l_2 \in L} \text{dist}(l_1, l_2) = \sum_{l_1 \in L} \sum_{(l_1 \neq l_2 \in L)} \text{dist}(l_1, l_2)$$

$$D_{LA} = \sum_{l_1 \in L} \sum_{v_1 \in V} \text{dist}(l_1, v_1) = \sum_{l_1 \in L} \sum_{(l_1 \neq v_1 \in V)} \text{dist}(l_1, v_1)$$

$$D_{AA} = \sum_{v_1 \in V} \sum_{v_2 \in V} \text{dist}(v_1, v_2) = \sum_{v_1 \in V} \sum_{(v_1 \neq v_2 \in V)} \text{dist}(v_1, v_2)$$

# 반사복제된 트리 - 풀이 Step 2-2

---

- 리프 노드의 개수, 전체 노드의 개수를 구하는 수식은 간단하다.
- $|L|$ 의 경우, 기존의 리프는 더 이상 리프가 아니게 되고, 새로 생긴 Edge Tree마다  $L-1$ 개씩 리프노드가 생기게 된다.
- $|V|$ 의 경우, 전체 노드 개수를 반사복제 후 서브트리의 개수만큼 곱하고, 겹쳐지는 부분만 빼주면 된다.

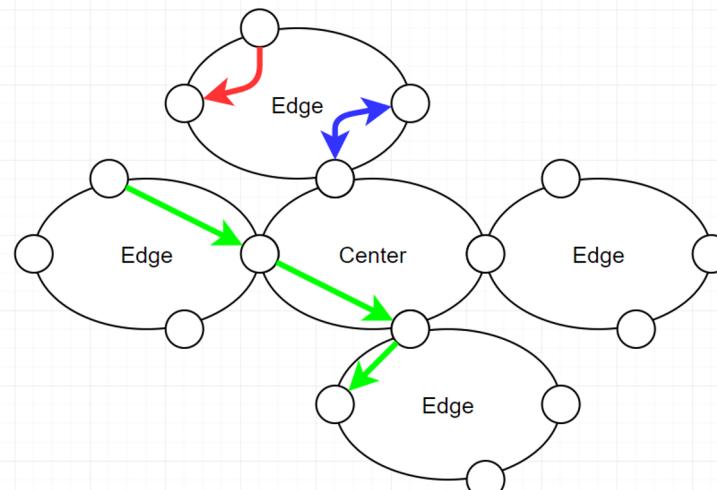
$$|L|_{i+1} = |L|_i^2 - |L|_i$$

$$|V|_{i+1} = |V|_i(|L|_i + 1) - |L|_i$$

# 반사복제된 트리 - 풀이 Step 2-3

---

- 다음 그림은 D\_LL을 구하는 아이디어이다.
- Edge 트리 안에서만 노는 경우(빨간색, 파란색)와 Edge-Center-Edge 경로(초록색)을 생각해준다. 파란색은 더하는 것이 아니라 빼줘야 한다.
- 빨간색, 초록색이 단방향인 이유는 어차피 모든 리프에 대해 생각하면 양방향을 모두 생각할 수 있기 때문이다. 하지만, 파란색은 양방향으로 생각하는 쪽이 더 편하다.



# 반사복제된 트리 - 풀이 Step 2-3

---

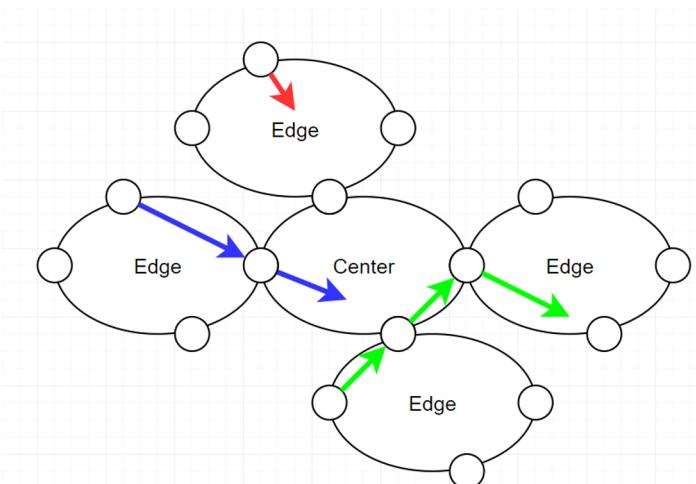
- 첫 번째 항은 Edge Tree 안에서만 노는 경우이다. Edge Tree와 Center Tree가 겹쳐지는 지점에 있던 기존의 리프노드가 출발점/도착점이 되는 경우를 뺀다.
  - 이때, 2를 곱해주는 이유는  $l_3$ 가 각 “겹쳐진 리프”이고  $l_4$ 가  $l_3$ 와 같은 Edge Tree에 속하는 리프 노드라고 생각하면 편하다.
- 두 번째 항은 Edge Tree - Center Tree - Edge Tree에 해당하는 경우이다.

$$\begin{aligned} D_{LL_{i+1}} &= \left( |L|_i \sum_{l_1 \in L_i} \sum_{(l_1 \neq l_2 \in L_i)} dist(l_1, l_2) - 2 \sum_{l_3 \in L_i} \sum_{(l_3 \neq l_4 \in L_i)} dist(l_3, l_4) \right) \\ &\quad + \left( \sum_{l_1 \in L_i} \sum_{(l_1 \neq l_2 \in L_i)} \sum_{(l_2 \neq l_3 \in L_i)} \sum_{(l_3 \neq l_4 \in L_i)} (dist(l_1, l_2) + dist(l_2, l_3) + dist(l_3, l_4)) \right) \\ &= (D_{LL_i}(|L|_i - 2)) + (3(|L|_i - 1)^2 D_{LL_i}) \end{aligned}$$

# 반사복제된 트리 - 풀이 Step 2-3

---

- 다음 그림은 D\_LA를 구하는 핵심 아이디어이다.
- 빨간색 경로: Edge Tree (겹쳐진 리프 포함) 안에서만 노는 경우
- 파란색 경로: Edge - Center (Edge - Center 사이의 겹쳐진 리프 미포함, 다른 겹쳐진 리프는 포함)
- 초록색: Edge - Center - Edge (모든 겹쳐진 리프 미포함)



# 반사복제된 트리 - 풀이 Step 2-3

---

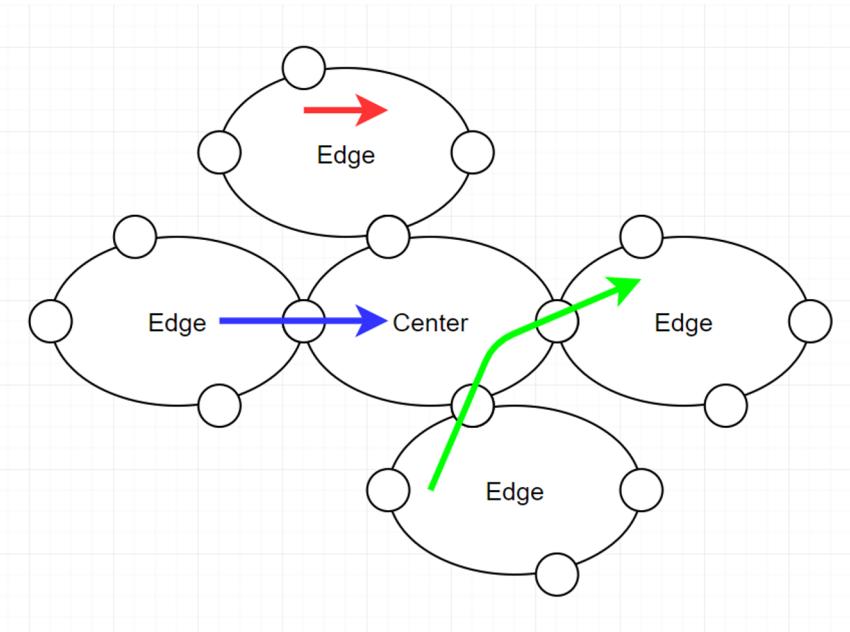
- 첫 번째 항은 Edge, 두 번째 항은 Edge - Center, 세 번째 항은 Edge - Center - Edge 경로를 의미한다.

$$\begin{aligned} D_{LA_{i+1}} &= \left( |L|_i \sum_{l_1 \in L_i} \sum_{(l_1 \neq v_1 \in V_i)} dist(l_1, v_1) - \sum_{l_2 \in L_i} \sum_{(l_2 \neq v_2 \in V_i)} dist(l_2, v_2) \right) \\ &\quad + \left( \sum_{l_1 \in L_i} \sum_{(l_1 \neq l_2 \in L_i)} \sum_{(l_1 \neq v_1 \in V_i)} (dist(l_2, l_1) + dist(l_1, v_1)) \right) \\ &\quad + \left( \sum_{l_1 \in L_i} \sum_{(l_1 \neq l_2 \in L_i)} \sum_{(l_1 \neq l_3 \in L_i)} \sum_{(l_2 \neq v_1 \in V_i)} (dist(l_3, l_1) + dist(l_1, l_2) + dist(l_2, v_1)) \right) \\ &= \left( D_{LA_i}(|L|_i - 1) \right) + \left( D_{LL_i}(|V|_i - 1) + D_{LA_i}(|L|_i - 1) \right) \\ &\quad + \left( D_{LA_i}(|L|_i - 1)^2 + 2D_{LL_i}(|L|_i - 1)(|V|_i - 1) \right) \end{aligned}$$

# 반사복제된 트리 - 풀이 Step 2-3

---

- 다음 그림은 D\_AA를 구하는 핵심 아이디어이다. 출발점이 리프노드에만 한정되지 않는다는 점을 제외하고, D\_LA와 세부사항이 완전 동일하다.



# 반사복제된 트리 - 풀이 Step 2-3

---

- 첫 번째 항은 Edge / Center (단일 영역 내 경로), 두 번째 항은 Edge - Center, 세 번째 항은 Edge - Center - Edge 경로를 의미한다.

$$\begin{aligned} D_{AA_{i+1}} &= \left( (|L|_i + 1) \sum_{v_1 \in V_i} \sum_{v_1 \neq v_2 \in V_i} dist(v_1, v_2) \right) \\ &\quad + \left( \sum_{v_1 \in V_i} \sum_{v_1 \neq l_1 \in L_i} \sum_{l_1 \neq v_2 \in V_i} (dist(v_1, l_1) + dist(l_1, v_2)) \right) \\ &\quad + \left( \sum_{v_1 \in V_i} \sum_{v_1 \neq l_1 \in L_i} \sum_{l_1 \neq l_2 \in L_i} \sum_{l_2 \neq v_2 \in V_i} (dist(v_1, l_1) + dist(l_1, l_2) + dist(l_2, v_2)) \right) \\ &= \left( D_{AA_i}(|L|_i + 1) \right) + \left( 2D_{LA_i}(|V|_i - 1) \right) \\ &\quad + \left( D_{LL_i}(|V|_i - 1)^2 + 2D_{LA_i}(|L|_i - 1)(|V|_i - 1) \right) \end{aligned}$$

# 반사복제된 트리 - 최종 결산

---

- 순수 DP만 빽세게 사용하는 문제!
- 시간복잡도는  $O(n + k)$  이다.
- Modular Arithmetic을 잘 처리해서 overflow가 나지 않도록 주의해야 한다.

# 찾아라 드래곤볼!

---

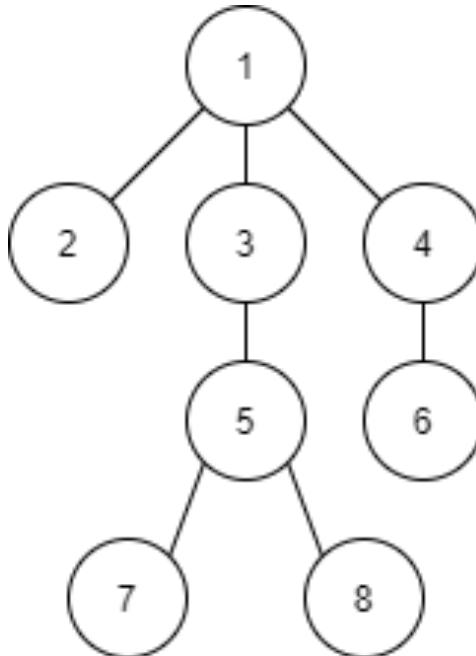
고려대 ALPS  
서태수

Advanced I

# 찾아라 드래곤볼!

---

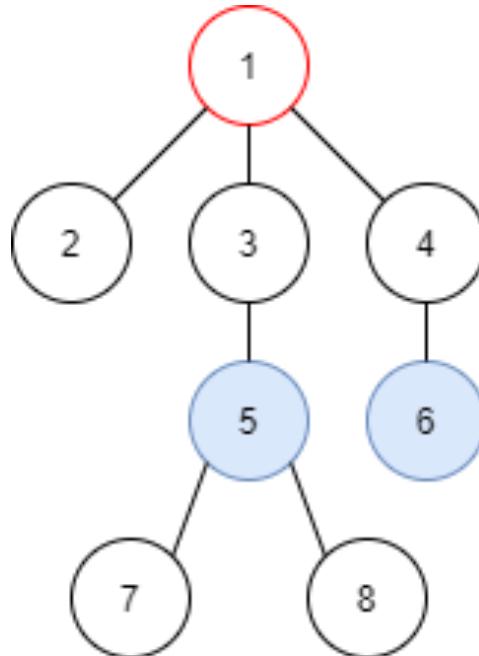
- 옥제의 마을이 아래의 트리처럼 생겼다고 생각해봅시다.



# 찾아라 드래곤볼!

---

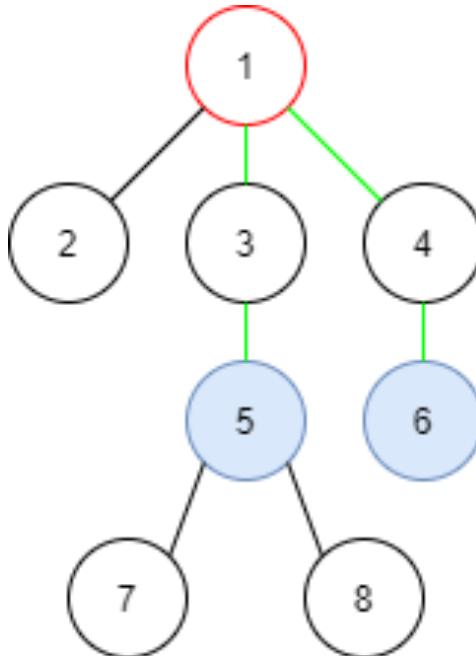
- 다음과 같이 파란 정점에 드래곤볼이 있다고 가정해봅시다. (1번 정점은 육제가 살고있는 마을)



# 찾아라 드래곤볼!

---

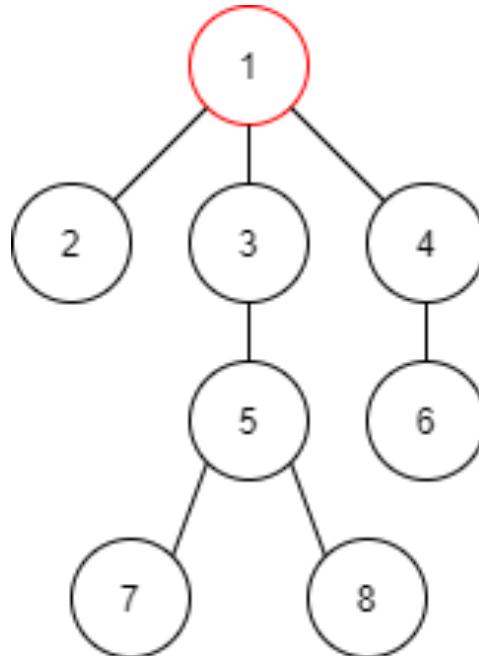
- 1번 정점에서 출발해서 모든 드래곤볼을 모아 다시 1번 정점으로 돌아오는 최단거리는 1번 정점과 드래곤볼이 있는 정점들로 구성된 가장 작은 subgraph의 간선의 개수 × 2입니다.



# 찾아라 드래곤볼!

---

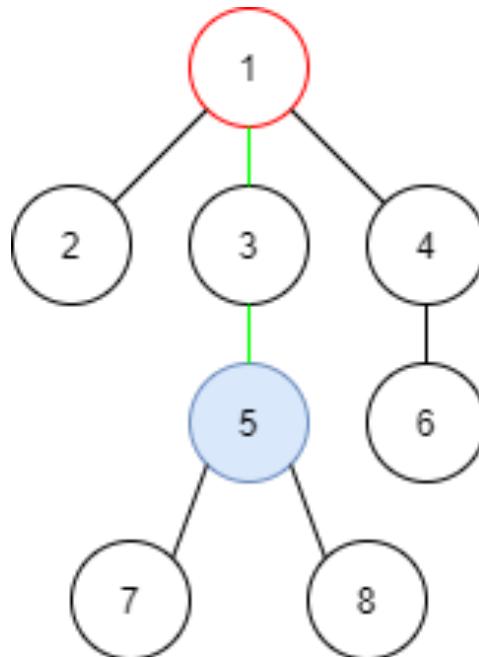
- 우선 경로에 대한 업데이트를 알아보기 전에 정점 1개씩 업데이트 할 때 어떻게 계산해야 할지 알아봅시다.



# 찾아라 드래곤볼!

---

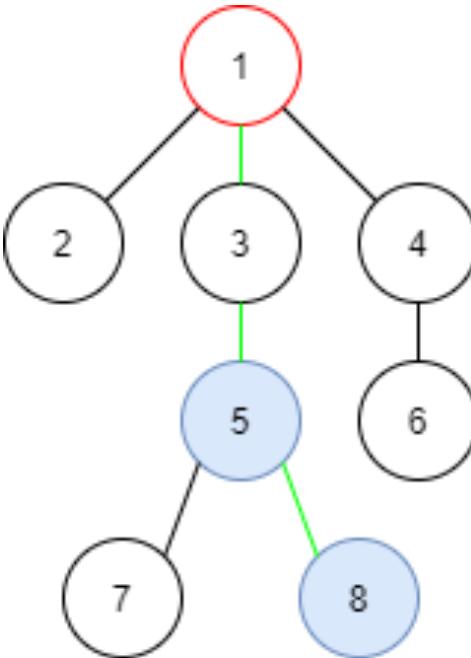
- 5번 정점에 드래곤볼이 추가된다면 간선의 개수가 2개 늘어납니다.



# 찾아라 드래곤볼!

---

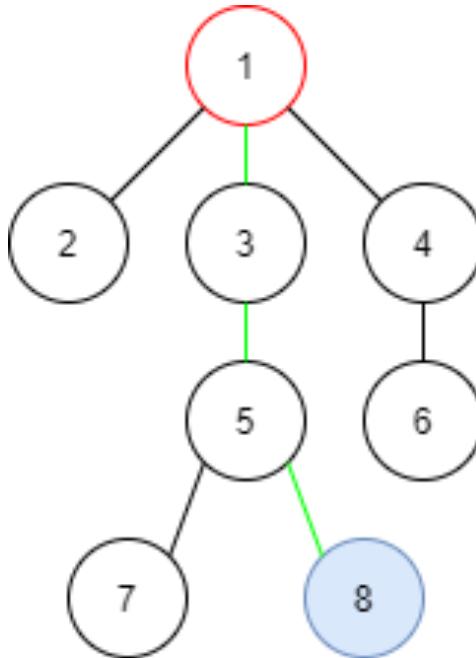
- 8번 정점에 드래곤볼이 추가된다면 간선의 개수가 1개 늘어납니다. 5번 정점에 이미 드래곤볼이 있기 때문에 그 위로는 안봐도 됩니다.



# 찾아라 드래곤볼!

---

- 여기서 다시 5번 정점의 드래곤볼을 없앴을 때, 5번 정점을 루트로 하는 서브트리에 속해있는 8번 정점에 드래곤볼이 있으므로 답에는 변화가 없습니다.



# 찾아라 드래곤볼!

---

- 정점에 드래곤볼이 추가될 때
  - 새 정점에 드래곤볼을 추가하기 전에 1번 정점까지 올라가면서 해당 정점을 루트로 하는 서브트리에 드래곤볼이 없었으면 새로운 간선 추가
- 정점에 있는 드래곤볼을 제거할 때
  - 해당 정점의 드래곤볼을 없애고 1번 정점까지 올라가면서 해당 정점을 루트로 하는 서브트리에 드래곤볼이 없으면 해당 간선 제거
- 스파스 테이블과 dfs ordering + 구간합 이용해 빠르게 계산 가능

# 찾아라 드래곤볼!

---

- 정점에 드래곤볼을 추가하는 것이 아니라 경로상의 모든 정점에 드래곤볼을 추가할 때도 똑같이 적용할 수 있습니다.
- 경로상의 정점들에 드래곤볼을 추가/제거하는 것은 HLD와 lazy propagation을 이용하면 처리할 수 있습니다.

# 찾아라 드래곤볼!

---

- 약간의 관찰을 하면 a-b 경로에 대한 쿼리가 주어지면 a, b에 대해서만 앞에서 언급했던 것을 적용하면 답을 업데이트 할 수 있다는 것을 알 수 있습니다.
- 다만 a, b에 대해서 2번 계산하기 때문에 몇몇 케이스들에 대해서 중복 처리를 해야합니다.
- $O(Q \log^2 N)$ 의 시간복잡도로 해결할 수 있습니다.
- Dynamic tree dp를 이용한  $O(Q \log^2 N \log \log N)$ 풀이도 존재합니다.