2021 GCC

Official Solution

2021 GBS Coding Contest by ALPS

Sponsored By NEXON STARTLINK

출제진

- 강희원(heeda0528)
- 권도현(gbs16_dohyun)
- 이채준(juney)
- 장래오(leo020630)
- 박수빈(psb0623)

검수진

- 나정휘(jhnah917, 숭실대학교)
- 신민철(fefe, KAIST)
- 오주원(kyo20111, 숭실대학교)
- 최유빈(sean9892, 충북과학고등학교)
- 최준석(stonejjun03, 고려대학교)

문제 이름	예상 난이도	출제자
비숍 투어	B2	강희원(heeda0528)
알프수	B1	강희원(heeda0528)
프린트 전달	G3	강희원(heeda0528)
등차수열과 쿼리	S2	강희원(heeda0528)
돌 굴러가유	S2	권도현(gbs16_dohyun)
달팽이팽이	G4	장래오(leo020630)
타이어 끌기	G2	장래오(leo020630)
바코드 찢기	G5	장래오(leo020630)
알프스의 힘	G1	이채준(juney)
합성함수와 쿼리 2	P4	박수빈(psb0623)

비숍 투어

- 알고리즘 분류: Case Work
- First Solve: admin";-- (4분)
- 해결한 팀의 수: 14팀
- 정답률: 24.675%
- 출제자: 강희원(heeda0528)

Solution (비숍 투어)

서브태스크 1

- 출발점과 도착점이 같다면 항상 이동할 수 있으므로 *YES* 를 출력하면 됩니다.

Solution (비숍 투어)

서브태스크 2

- 체스판의 각 칸을 검은색과 흰색 중 하나로 색칠하되, 변을 공유하는 두 칸은 서로 다른 색으로 칠했다고 해봅시다.

- 비숍이 대각선으로만 움직이므로 이동 중에 비숍이 위치한 칸의 색은 변하지 않습니다.

Solution (비숍 투어)

- 이제 출발점과 도착점의 색이 같은지만 판별하면 서브태스크 2를 해결할 수 있습니다.

- 이는 어떻게 확인할 수 있을까요?

- $(1, 1)$을 흰색으로 칠했다면, 흰색으로 칠해지는 모든 점은 (홀수, 홀수) 또는 (짝수, 짝수) 꼴이고 검은색으로 칠해지는 모든 점은 (홀수, 짝수) 또는 (짝수, 홀수) 꼴입니다 .

Solution (비숍 투어)

- 따라서 출발점의 X좌표와 Y좌표를 더한 값과 도착점의 X좌표와 Y좌표를 더한 값의 홀짝성이 같다면 두 점은 같은 색이라고 판단할 수 있습니다 .

- 같은 색이라면 *YES*, 다른 색이라면 *NO* 를 출력하면 됩니다.

Solution (비숍 투어)

서브태스크 3

- 사실, 앞에서 설명한 풀이에는 작은 반례가 있습니다.

- 과연 색이 같은 두 점 간에는 항상 이동이 가능할까요?

Solution (비숍 투어)

- 비숍은 대각선으로만 이동하기 때문에, 자유롭게 움직이기 위해서는 최소 가로 2칸, 세로 2칸이 확보되어야 합니다.

- 따라서 $N = 1$ 또는 $M = 1$ 이고 출발점과 도착점이 다르다면 NO 를 출력해야 합니다.

- 모든 서브태스크의 시간복잡도는 $O(1)$ 입니다.

알프수

- 알고리즘 분류: Implementation
- First Solve: py (12분)
- 해결한 팀의 수: 8팀
- 정답률: 3.587%
- 출제자: 강희원(heeda0528)

Solution (알프수)

서브태스크 1

- X의 크기가 매우 크므로 입력을 문자열로 받아야 합니다.

- X에 1과 2만 등장하므로 오르막 또는 내리막의 경사가 중간에 달라지는 일은 일어나지 않습니다.

- 처음 숫자와 마지막 숫자가 1이고 평지가 없다면 *YES*, 아니면 *NO* 를 출력하면 됩니다.

Solution (알프수)

서브태스크 2

- 잘 생각해보면 서브태스크 2와 상황이 크게 다르지 않습니다.

- 처음이 오르막, 마지막이 내리막이고 평지가 없다면 *YES*, 아니면 *NO* 를 출력하면 됩니다.

Solution (알프수)

서브태스크 3

- 서브태스크 2의 풀이를 포함하면서 오르막 또는 내리막의 경사가 중간에 달라지지 않는지를 추가로 확인해야 합니다.

- 바로 전 구간의 기울기를 저장하면서 현재 기울기와 부호가 같지만 절댓값이 다른 부분이 있는지 검사하고, 한 군데라도 존재한다면 *NO*, 아니면 *YES* 를 출력하면 됩니다.

- 모든 서브태스크의 시간복잡도는 $O(X$의 길이$)$ 입니다.

프린트 전달

- 알고리즘 분류: BFS, Tree DP(DFS)
- First Solve: 날강두 (80분)
- 해결한 팀의 수: 2팀
- 정답률: 15.385%
- 출제자: 강희원(heeda0528)

Solution (프린트 전달)

서브태스크 1

- 학생들이 한 줄로 앉아있는 상황입니다.

- 학생들이 중간에 떨어져서 앉아있다면 프린트가 반대쪽으로 넘어갈 수 없으므로 *-1* 을 출력합니다.

Solution (프린트 전달)

- S번 학생을 기준으로 왼쪽에 L명, 오른쪽에 R명 있다고 하면 각각 왼쪽과 오른쪽을 시작으로 1부터 L(또는 R)까지의 수만큼 프린트를 받게 됩니다.

- S번 학생은 프린트를 $K(=L+R+1)$개 받게 됩니다.

- 시간복잡도는 $O(M)$ 입니다.

Solution (프린트 전달)

서브태스크 2

- 학생들이 두 줄로 자리를 꽉 채워서 앉아있고, 선생님이 (1, 1)에 앉은 학생에게 프린트를 건네는 상황입니다.

- 항상 모든 학생이 프린트를 받을 수 있습니다.

Solution (프린트 전달)

- (1, 1)에 위치한 학생은 (1, 2), (2, 1), (2, 2)에 위치한 학생에게 프린트를 전달합니다.

- $i \geq 2$ 일 때, (1, i)와 (2, i)에 위치한 학생은 (1, i-1)와 (2, i-1)에 위치한 학생 중 번호가 더 작은 학생에게 프린트를 받습니다.

- 각 학생이 받아야 하는 프린트의 수는 어떻게 정할 수 있을까요?

Solution (프린트 전달)

- S번 학생을 루트로 하고, 프린트를 주는 학생이 부모, 받는 학생이 자식 관계가 되는 트리를 만들었다고 생각해봅시다.

- 이제 각 학생이 받아야 하는 프린트의 수는 그 학생의 번호를 루트로 하는 서브 트리에 속한 정점의 수와 같습니다.

- 이는 DFS를 돌면서 자식들의 크기를 부모의 크기에 모두 더해주는 식으로 구할 수 있습니다. (백준 15681과 같은 문제입니다)

Solution (프린트 전달)

- 비슷하지만, 오른쪽부터 보면서 부모에게 자식의 크기를 더해주는 식으로 구현하면 DFS를 쓰지 않고 해결할 수 있습니다.

- 시간복잡도는 $O(K)$ 입니다.

Solution (프린트 전달)

서브태스크 3

- 전체 문제에서 까다로운 조건 하나가 빠졌습니다.

- 8방향 BFS를 돌면서 프린트의 전달 관계를 나타내는 인접 리스트를 만들 수 있습니다.

- 이를 이용해 서브태스크 2에서 언급한 트리를 구성하고, 각 학생이 받아야 하는 프린트의 수를 서브트리의 크기로 구하면 됩니다.

Solution (프린트 전달)

- 시간복잡도는 $O(K)$ 입니다.

Solution (프린트 전달)

서브태스크 4

- BFS를 조금 변형해서 큐에서 원소를 하나 뽑을 때 가장 먼저 넣은 값이 아닌, 가장 가까우면서 번호가 가장 작은 학생이 준 원소가 나오게 만들면 좋을 것 같습니다.

- 그리고 이는 BFS에서 큐를 우선순위 큐로 대체하면 쉽게 해결 가능합니다.

- 시간복잡도는 $O(K \log K)$ 입니다.

Solution (프린트 전달)

- 우선순위 큐를 사용하지 않는 방법도 있습니다.

- BFS를 돌면서 부모-자식 관계를 그대로 저장하지 않고, 어떤 학생에 도달했을 때 그 학생과 인접한 8방향 중 본인보다 한 단계 가까우면서 번호가 가장 작은 학생에게 프린트를 받도록 구현하면 됩니다.

- 이 경우 시간복잡도는 $O(8K) = O(K)$ 입니다.

등차수열과 쿼리

- 알고리즘 분류: Math
- First Solve: **권도현짹짹이 (85분)**
- 해결한 팀의 수: **5팀**
- 정답률: 1.961%
- 출제자: **강희원(heeda0528)**

Solution (등차수열과 쿼리)

서브태스크 1

- 문제 그대로 구현하면 됩니다.
- $\gcd(x, y, z) = \gcd(\gcd(x, y), z)$ 입니다.
- gcd를 구하기 위해 수를 하나씩 다 나눠봐도 됩니다.
- 시간복잡도는 $O(QX^2)$ 입니다.

Solution (등차수열과 쿼리)

서브태스크 2

- gcd를 조금 더 빠르게 구해야 합니다.
- 유클리드 호제법을 이용하면 gcd를 $O(\log X)$에 구할 수 있습니다.
- 시간복잡도는 $O(QX \log X)$ 입니다.
- 오버플로우에 주의해야 합니다.

Solution (등차수열과 쿼리)

서브태스크 3

- 범위가 많이 커져서, 수열을 전부 탐색하면 시간초과를 받습니다.
- 등차수열의 합은 아주 유명한 공식으로 $O(1)$에 구할 수 있습니다.
- 수열의 연속한 원소들의 gcd는 어떻게 빠르게 구할 수 있을까요?

Solution (등차수열과 쿼리)

- 양의 정수 a, b에 대해 $a < b$ 라고 했을 때, $\gcd(a, b) = \gcd(a, b - a)$ 가 성립합니다 . 이는 유클리드 호제법에서도 이용하는 성질입니다 .

- A_l, A_{l+1} 의 최대공약수를 구해봅시다.

- $\gcd(a + (l - 1) * d, a + l * d) = \gcd(a + (l - 1) * d, d) = \gcd(a, d)$

- $\gcd(\gcd(A_l, A_{l+1}), A_{l+2}) = \gcd(\gcd(a, d), a + (l + 1) * d) = \gcd(a, d)$

Solution (등차수열과 쿼리)

- 이를 반복하면...

- $\gcd(A_l, \ldots, A_r) = \gcd(a, d)$ 입니다.

- 단, $l = r$ 일 때는 A_l 자체가 최대공약수가 됩니다.

- 이제 구간의 최대공약수도 $O(logX)$ 에 구할 수 있습니다.

- 따라서 전체 문제를 $O(QlogX)$ 에 해결할 수 있습니다.

Solution (등차수열과 쿼리)

- 세그먼트 트리나 희소 배열 등을 이용하면 같은 시간 복잡도에 해결할 수 있습니다.

- 다만, 머리가 고생하는 것보다 손이 훨씬 더 고생하므로 추천하지는 않습니다.

돌 굴러가유

- 알고리즘 분류: Greedy
- First Solve: 날강두 (97분)
- 해결한 팀의 수: 2팀
- 정답률: 8.000%
- 출제자: 권도현(gbs16_dohyun)

Solution (돌 굴러가유)

- 서브태스크 1

- M=K일 때, 모든 벽을 돌의 위치에 놓는 것이 가장 좋은 해임을 알 수 있습니다.

- 벽을 돌이 위치한 마을에 놓는다면, 돌이 아예 굴러가지 못하기 때문에 모래성을 하나도 부수지 못하기 때문입니다.

- M=K인 서브태스크는 단순히 입력받은 돌의 위치를 출력하면 되므로, O(M)=O(K)에 해결할 수 있습니다.

Solution (돌 굴러가유)

- 서브태스크 2

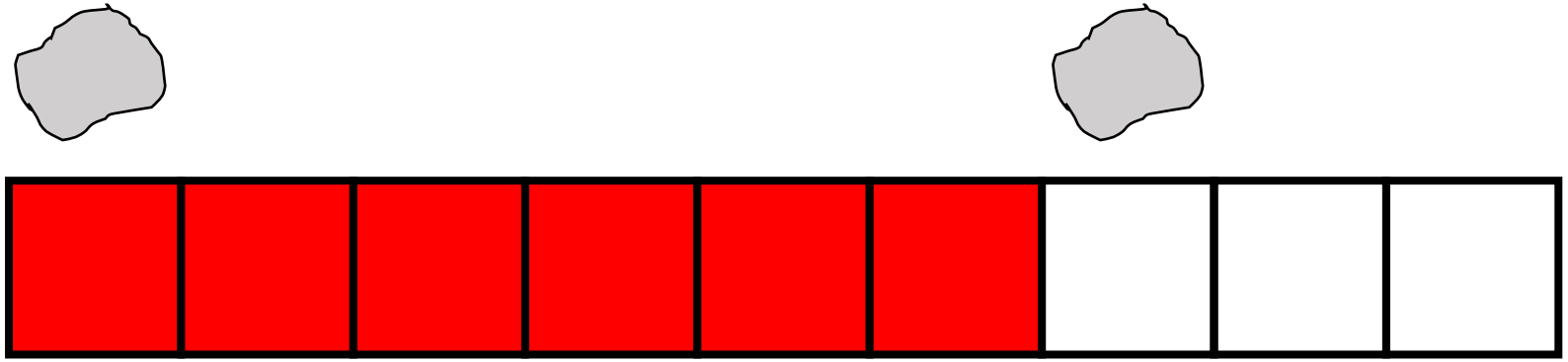
  - 두 번째 서브태스크에서는 완전 탐색을 시행할 수 있습니다.

  - 최대 가능한 돌의 개수가 25개이니, 최대 25개의 돌에 벽을 세울 수 있는 경우의 수를 모두 찾아볼 수 있습니다.

  - 시뮬레이션을 시행해 모든 경우의 수를 탐색할 수 도 있으나, $\binom{25}{12} \approx 5000000$이고, $\mathrm{N} = 100$이므로 최적화를 진행하거나, 시뮬레이션을 하지 않고 부서지는 모래성의 개수를 세는 방법을 찾아야 합니다.

Solution (돌 굴러가유)

- 서브태스크 3



- 위의 그림과 같이, 돌은 다음 돌 전까지의 모래성만 자신이 부술 수 있습니다. 그 이유는, 다음 돌의 위치에 벽이 있다면 다음 돌 이후의 모래성은 부수지 못하며, 다음 돌의 위치에 벽이 없다면 다음 돌 이후의 모래성은 다음 돌이 부쉈을 것이기 때문입니다.

Solution (돌 굴러가유)

- 서브태스크 3

- 따라서, 하나의 돌이 부술 수 있는 모래성은 그 돌의 위치부터 다음 돌 바로 이전의 위치까지 있는 모래성까지 입니다. 이를 통해 완전탐색을 하면 서브태스크 2를 해결할 수 있으나, 이미 이를 알아냈다면 서브태스크 3까지 해결할 수 있습니다.

- 각 돌이 부술 수 있는 모래성의 개수를 구하고, 부술 수 있는 모래성의 개수가 가장 큰 순서대로 벽을 놓으면 됩니다.

- 돌이 부술 수 있는 모래성의 개수를 정렬하는 총 시간 복잡도는 $O(KlogK)$가 됩니다. 이때 두 개의 변수를 함께 정렬해야 하는데, 이는 백준 11650(좌표 정렬하기)를 참고하시면 됩니다.

달팽이팽이

- 알고리즘 분류: Geometry, Ad-Hoc
- First Solve: **마인크래프트 (36분)**
- 해결한 팀의 수: **5팀**
- 정답률: **4.762%**
- 출제자: **장래오(leo020630)**

Solution (달팽이팽이)

- 영역 내부의 점 $P(a, b)$에 대해 테두리에 위치한 점 중 P와 가장 멀리 떨어진 점을 $Q(x, y)$라 합시다.

- Q는 $(0, R)$과 $(0, -R)$중 하나로, $b \geq 0$일 경우 $(0, -R)$, $b < 0$일 경우 $(0, R)$입니다.

Solution (달팽이팽이)

- 증명은 다음과 같습니다. 일반성을 잃지 않고 $b \geq 0$이라 할 때,

(i) Q가 y축 위의 점일 경우에는 Q를 $(0, -R)$으로 옮기는 것이 최적입니다.

(ii) Q가 반원 위의 점일 경우 조건에 따라 P와 Q사이의 거리 > P와 $(0, -R)$사이의 거리여야 하고, 이는 $(x-a)^2+(y-b)^2 > a^2+(b+R)^2$으로 나타낼 수 있습니다.

- 이를 정리하면 $x^2+y^2-2ax-2by > 2bR+R^2$이고, 조건에 따라 $x^2+y^2=R^2$이므로 $-ax > b(y+R)$을 만족해야 합니다. 이 때, 조건에 의해 좌변은 0 이하, 우변은 0 이상입니다. 따라서 주어진 부등식은 모순입니다.

Solution (달팽이팽이)

- 서브태스크 1, 2

- 일반성을 잃지 않고 $b \geq 0$이라 하면, 문제는 $(0, -R)$과 가장 멀리 떨어진 영역 내부의 점 $P(a, b)$를 찾는 것으로 바뀝니다. 두 점 사이의 거리를 d라 하면 문제에서 최대화해야 하는 답은 πd^2이기 때문입니다.

- $d^2 = a^2 + (b + R)^2 = a^2 + b^2 + 2bR + R$^2이며, R^2은 상수이므로 우리가 최대화해야 하는 값은 $a^2 + b^2 + 2bR = f(b)$입니다.

- 모든 점에 대해 이를 구해보는 방식으로 2번 서브태스크까지의 문제를 $O(R^2)$에 해결할 수 있습니다.

Solution (달팽이팽이)

- 서브태스크 3

- 각각의 y좌표 b에 대해 $a^2 + b^2 < R$^2을 만족하는 최대의 a를 K_b라 합시다.

- $f(b) = K_b^2 + b^2 + 2bR$이며, 이는 $f(b) < R^2 + 2bR$을 만족합니다.

- 각각의 b에 대해 K_b를 $O(logR)$ 또는 $O(\sqrt{R})$에 구할 수 있고, 이를 이용해 전체 문제를 $O(RlogR)$ 또는 $O(R\sqrt{R})$에 해결할 수 있으며, 이는 3번 서브태스크에 해당합니다.

Solution (달팽이팽이)

- 서브태스크 4

- 이 때, 만약 $f(R-1) > R\text{^}2 + 2(R-2)R$이라면 $0 \le b < R$에 대해 $f(R-1)$이 최대값을 가진다는 것을 알 수 있습니다. $0 \le b < R-1$에 대해 $f(b)$는 $R^2 + 2(R-2)R$이상으로 커질 수 없기 때문입니다. 그리고 이는 성립합니다.

- $f(R-1) = K_{R-1}^2 + (R-1)^2 + 2(R-1)R = K_{R-1}^2 + 3R^2 - 4R + 1 > R^2 + 2(R-2)R = 3R^2 - 4R$이기 때문입니다.

Solution (달팽이팽이)

- 서브태스크 4

- 따라서 답이 되는 $P(a,b) = (K_{R-1}, R-1)$이며, K_{R-1}은 이분탐색으로 구하거나 $\sqrt{2R}$ 이하라는 점을 이용해 반복문으로 구할 수 있습니다. 또는 일반식을 이용해 sqrt() 함수를 사용하면 조금 더 빠른 시간에 구할 수도 있습니다.

- 시간 복잡도는 $O(logR)$ 또는 $O(\sqrt{R})$입니다.

타이어 끌기

- 알고리즘 분류: DP
- First Solve: 날강두(171분)
- 해결한 팀의 수: 1팀
- 정답률: 0.862%
- 출제자: 장래오(leo020630)

Solution (타이어 끌기)

- 서브태스크 1

- 각 타이어별로 질 것이라면 0명을, 비길 것이라면 P_i명을, 이길 것이라면 $P_i + 1$명을 보내는 것이 최적임은 자명합니다.

- 1번 서브태스크의 경우 1개의 타이어만 승리하고 다른 타이어들에 대해 비기는 전략을 사용하면 무조건 승리할 수 있음을 알 수 있습니다.

Solution (타이어 끌기)

- 서브태스크 2

- 각 경우에 대해 얻는 상대적인 점수를 구하기 위해, 1학년 2반의 점수를 $\sum S_i = K$로 고정시켜 생각합시다.
- 이 때, 각 타이어별로 지는 경우 0의 점수를, 비길 경우 S_i의 점수를, 이길 경우 $2S_i$의 점수를 얻는 것으로 해석할 수 있습니다.
- 이를 정리하면 다음과 같습니다. 이를 모든 타이어에 대해 일일이 계산하면 2번 서브태스크를 해결할 수 있으며, 시간 복잡도는 $O(3^N)$입니다.

경우	투자 인원	획득 점수
짐	0	0
비김	P_i	S_i
이김	$P_i + 1$	$2S_i$

Solution (타이어 끌기)

- 서브태스크 3

- 이러한 상황은 타이어의 수와 투자한 총 인원 수를 상태 값으로 하는 DP (Knapsack DP)로 해결할 수 있음이 잘 알려져 있습니다.

- $DP[i][j]$를 i번째 타이어까지 존재할 때 j명을 투자해 얻을 수 있는 최대 점수로 정의합시다.

- $DP[i][j] = \max(DP[i-1][j], DP[i-1][j-P_i] + S_i, DP[i-1][j-P_i-1] + 2S_i)$
와 같은 점화식으로 DP 테이블을 채울 수 있습니다..

Solution (타이어 끌기)

- 서브태스크 3

- 얻을 수 있는 최대 점수는 $\max_{0 \le j \le M} DP[n][j]$ 이며, 이 값이 K보다 작다면 L, 크다면 W, K와 같다면 D를 출력하면 됩니다.

- 시간 복잡도는 $O(NM)$, 공간 복잡도는 구현에 따라 $O(NM)$ 혹은 $O(M)$입니다.

바코드 찢기

- 알고리즘 분류: Math, String
- First Solve: py (197분)
- 해결한 팀의 수: 1팀
- 정답률: 5.882%
- 출제자: 장래오(leo020630)

Solution (바코드 찢기)

- 주어진 바코드를 찢어 만들 수 있는 올바른 바코드의 개수는 바코드 안에 있는 ()의 개수와 같습니다.

- 만약 올바른 바코드의 형태가 S_1S_2 형태라면 S_1과 S_2로 나눠 올바른 바코드를 한 개 더 만들 수 있고, (S) 형태라면 $(, S,)$로 나누어도 올바른 바코드의 개수는 같기 때문입니다.

- 이는 반복 횟수가 1번이라면 $O(|S|)$에 구할 수 있습니다.

- 만약 반복 횟수 P가 2 이상이라면 기존 문자열의 끝과 시작을 이었을 때에 괄호가 나타나는지 판단한 후, 나타난다면 $P-1$번 만큼을 더해주면 됩니다.

Solution (바코드 찢기)

- 가지고 있는 바코드의 () 개수를 Y, 음료수 바코드의 () 개수를 X라 합시다.
- 만약 $Y < K$라면 음료수를 구매하지 못하며, 답은 0입니다.
- 만약 $Y \geq K, X \geq K$ 이라면 음료수를 무한히 구매할 수 있으며, 답은 N입니다.

Solution (바코드 찍기)

- 서브태스크 1

- 1번 서브태스크의 경우에는 $K = 1$으로, $X \geq K$ 혹은 $X = 0$임을 알 수 있습니다.

- $X \geq K$인 경우의 답은 위에 따라 $Y < K$라면 0, $Y \geq K$라면 N입니다.

- $X = 0$인 경우 바코드 개수는 Y개에서 K개씩 감소를 반복하므로 답은 $\frac{Y}{K}$ 입니다.

Solution (바코드 찍기)

- 서브태스크 2, 3

- 이제 $Y \geq K > X$인 경우만 생각할 수 있습니다.

- 이 경우, 올바른 바코드 수는 음료수를 더 이상 구매하지 못할 때 까지 $-K, +X$를 반복합니다.

- 구매한 음료수의 개수를 T라고 합시다. 이때 T가 만족해야 하는 조건은 $0 \leq Y - T(K - X) < K$입니다.

- 즉, 음료수를 T개 구매한 후 바코드가 음수가 되지 않으면서 더 이상 음료수를 살 수 없는 상태입니다.

Solution (바코드 찢기)

- 서브태스크 2, 3

- 부등식을 정리하면 $\frac{Y-K}{K-X} < T \leq \frac{Y}{K-X} = \frac{Y-K}{K-X} + \frac{K}{K-X}$입니다.

- 이때 T는 정수이므로 T의 최솟값 $= \mathrm{T}_{\min} = \frac{Y-K}{K-X} + 1$임을 알 수 있습니다.

- 음료수를 N개보다 많이 살 수는 없으므로 답은 $\min(N, \mathrm{T}_{\min})$ 이 됩니다.

- 서브태스크 2는 $\mathrm{T}_{\min} \leq 10^6$, 서브태스크 3은 $\mathrm{N} \leq 10^6$ 을 만족하므로 반복문을 이용한 $O(\min(N, \mathrm{T}_{\min}))$해법으로 답을 직접 구할 수 있습니다.

Solution (바코드 찢기)

- 서브태스크 4

- 만점을 맞기 위해서는 위처럼 답의 일반식을 계산하거나 $0 \leq Y - T(K - X) < K$를 만족하는 최소의 T를 이분탐색으로 찾는 방법을 구현하면 됩니다.

- 이 경우 시간 복잡도는 X, Y를 구하는 작업에 지배적이므로 $O(|S|)$입니다.

알프스의 힘

- 알고리즘 분류: Math
- First Solve: -
- 해결한 팀의 수: 0팀
- 정답률: 0%
- 출제자: 이채준(juney)

Solution (알프스의 힘)

- 서브태스크 1

- 오버플로우에 유의하며 각 쌍에 대해 주어진 조건을 만족하는지 확인하면 해결할 수 있습니다.

Solution (알프스의 힘)

- 서브태스크 2

  - 모든 원소 쌍 (A_i, A_j)에 대해 $A_i \neq A_j$입니다.

  - 따라서 인수분해 공식을 통해 $A_i^2 + A_iA_j + A_j^2 = \frac{A_i^3 - A_j^3}{A_i - A_j}$로 나타낼 수 있으며, 이를 합동식으로 나타내면 $A_i^3 - A_j^3 \equiv K(A_i - A_j)$를 만족해야 합니다.

  - 이를 정리하면 $A_i^3 - KA_i \equiv A_j^3 - KA_j$입니다.

Solution (알프스의 힘)

- 서브태스크 2

  - 따라서 $A_i^3 - KA_i \equiv f(A_i)$라 하면 문제는 $f(A_i) = f(A_j)$를 만족하는 (A_i, A_j)쌍을 찾는 것으로 바뀌게 됩니다.
  - 만약 같은 함수값을 가지는 원소가 x개라면 $\binom{x}{2}$개의 쌍을 만들 수 있고, 각각의 함수값에 대해 이를 모두 더해주면 답을 구할 수 있습니다.

Solution (알프스의 힘)

- 서브태스크 2

  - 이는 map등의 자료구조를 사용하거나 원소 각각에 대해 A_i를 $f(A_i)$로 변환한 후 정렬하는 방법을 사용할 수 있으며, 시간 복잡도는 모두 $O(NlogN)$입니다.

  - 다른 풀이로는, 주어진 식을 이차방정식 형태로 바꾸어 합동식에서 이차방정식을 푸는 알고리즘을 사용하면 답을 구할 수 있습니다. 이 경우 시간복잡도는 $O(Nlog^2P)$입니다.

합성함수와 쿼리 2

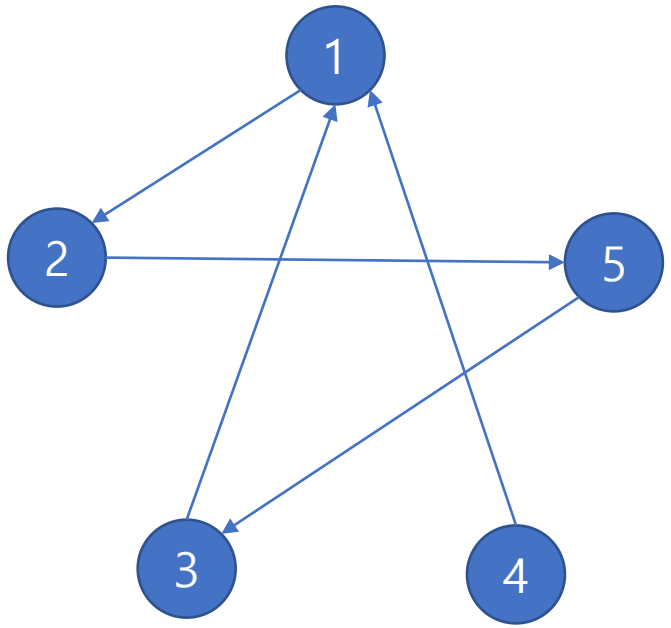
- 알고리즘 분류: Sparse Table, DFS, Case Work
- First Solve: -
- 해결한 팀의 수: 0팀
- 정답률: 0%
- 출제자: 박수빈(psb0623)

Solution (합성함수와 쿼리 2)

- 주어진 함수를 그래프로 모델링할 수 있습니다.

$$f(1) = 2$$
$$f(2) = 5$$
$$f(3) = 1$$
$$f(4) = 1$$
$$f(5) = 3$$

↔



Solution (합성함수와 쿼리 2)

- 이 경우, 합성함수의 값을 구하는 과정은 그래프의 간선을 따라가는 과정으로 볼 수 있습니다.

$f(1) = 2$

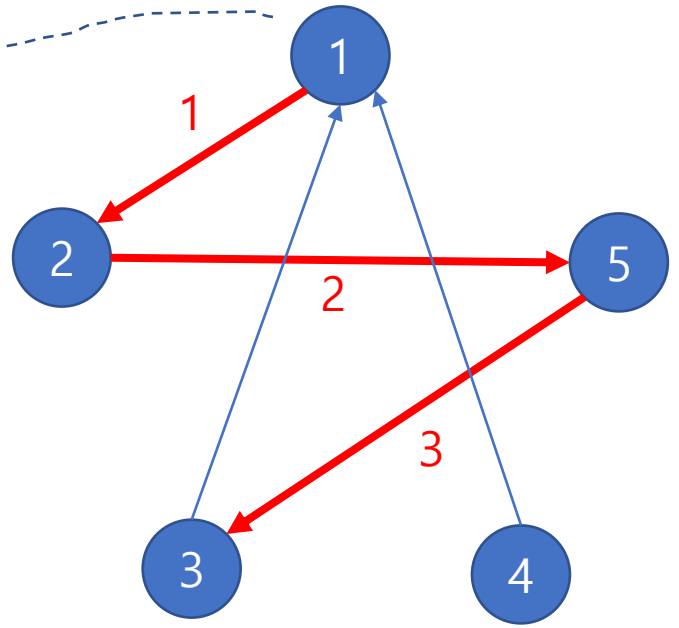
$f(2) = 5$

$f(3) = 1$

$f(4) = 1$

$f(5) = 3$

↔



$f^3(1) =$

$f(f(f(1)) = f(f(2)) = f(5) = 3$

Good!

Solution (합성함수와 쿼리 2)

- 보통의 경우 희소 테이블로 풀 수 있지만…
- $f(1)$값이 계속 변하기 때문에 처리가 필요합니다.

(* 희소 테이블은 update를 효율적으로 수행하지 못합니다)

Solution (합성함수와 쿼리 2)

- 서브태스크 1
- 희소 테이블을 활용하지 않고도 함숫값(혹은 노드)를 따라가는 과정을 일일이 진행해 주면 $O(Qm)$으로 서브태스크 1을 통과할 수 있습니다.

Solution (합성함수와 쿼리 2)

- 서브태스크 2
- $f(1)$ 값이 바뀔 때마다 희소 테이블을 새로 만들 수 있습니다.
- 희소 테이블을 새로 만드는데 걸리는 시간은 $O(N \log 10^9)$이고, 업데이트 쿼리는 최대 Q개이므로 $O(QN \log 10^9)$ 로 서브태스크 2를 통과할 수 있습니다.

Solution (합성함수와 쿼리 2)

- 서브태스크 3
- $f(1)$ 이 가질 수 있는 값이 최대 N개이므로, $f(1)$값이 $1, 2, \ldots, N$ 인 모든 경우에 대해 희소 테이블을 미리 만들어둘 수 있습니다.
- N 개의 희소 테이블을 사용하므로 $O(N^2 \log 10^9)$ 의 공간을 사용하고, 1번 쿼리는 $O(1)$, 2번 쿼리는 $O(\log 10^9)$에 해결 가능합니다.
- 따라서 $O(N^2 \log 10^9 + Q \log 10^9)$ 로 서브태스크 3을 통과할 수 있습니다.

Solution (합성함수와 쿼리 2)

- 서브태스크 4
- 모든 희소 테이블을 계산하거나 저장할 수 없습니다.
- 단 하나의 희소 테이블로 문제를 풀어야 하겠지만…
- $f(1)$의 값이 계속 변하는게 문제입니다.

→ $f(1)$의 영향력을 줄여봅시다.

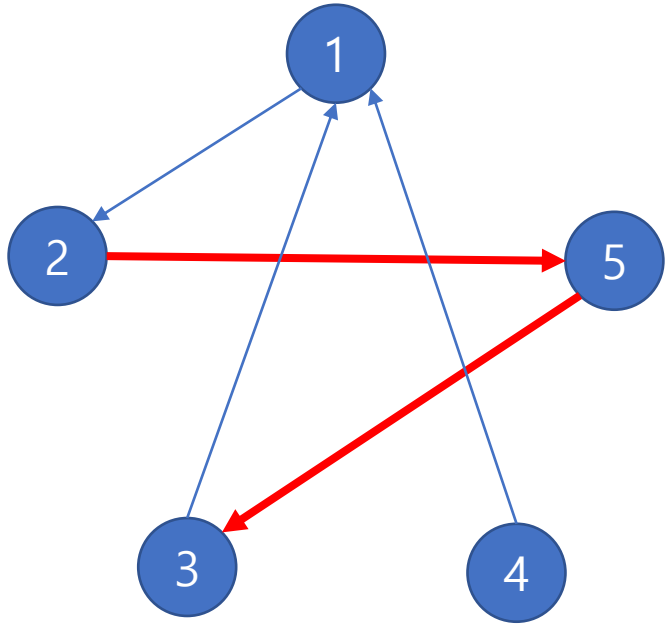
Solution (합성함수와 쿼리 2)

- 서브태스크 4
- 우선, $f(1)$의 값을 제외한 모든 값이 제대로 들어간 희소 테이블을 하나 만듭시다.

- 계산 경로에 1이 등장하지만 않는다면, 이 희소 테이블은 올바른 결과를 낼 것입니다.

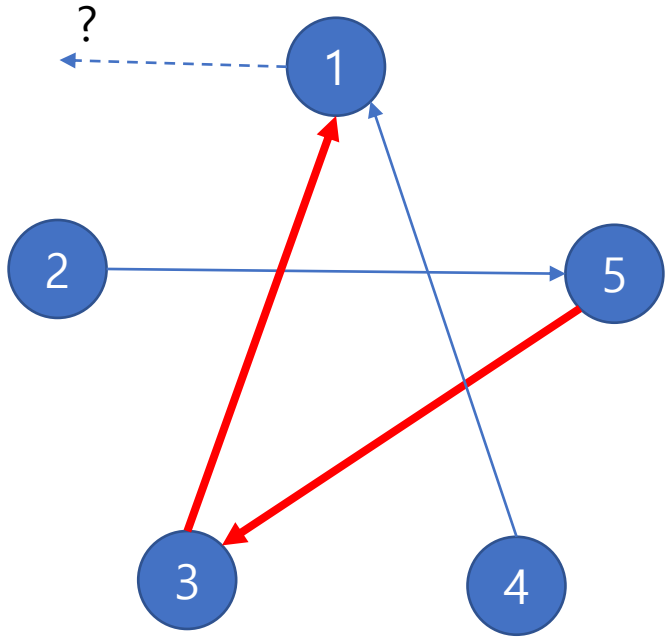
Solution (합성함수와 쿼리 2)

- 서브태스크 4
- 계산 경로에 1이 등장하지 않으면, 그냥 희소 테이블을 사용해도 올바른 결과가 보장됩니다.



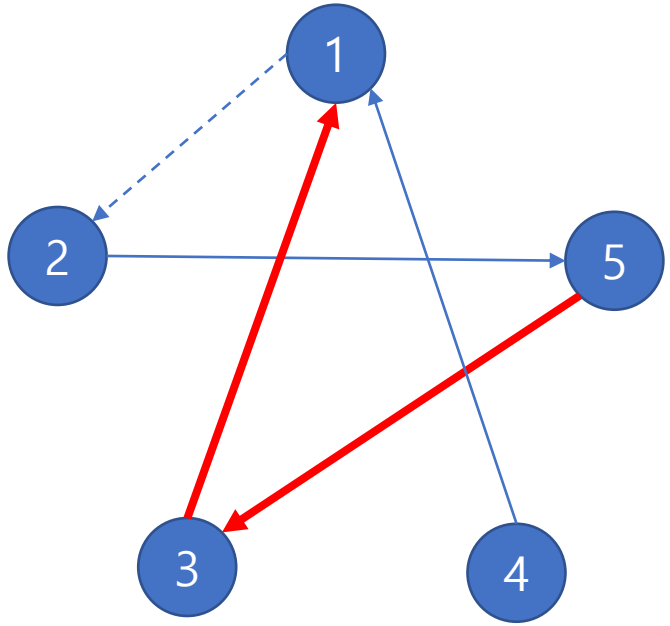
Solution (합성함수와 쿼리 2)

- 서브태스크 4
- 계산 경로에 1이 등장하면, 1이 나오는 시점에 끊어줍시다.
- 우리의 희소 테이블은 이 다음에 어디로 갈지 모릅니다.



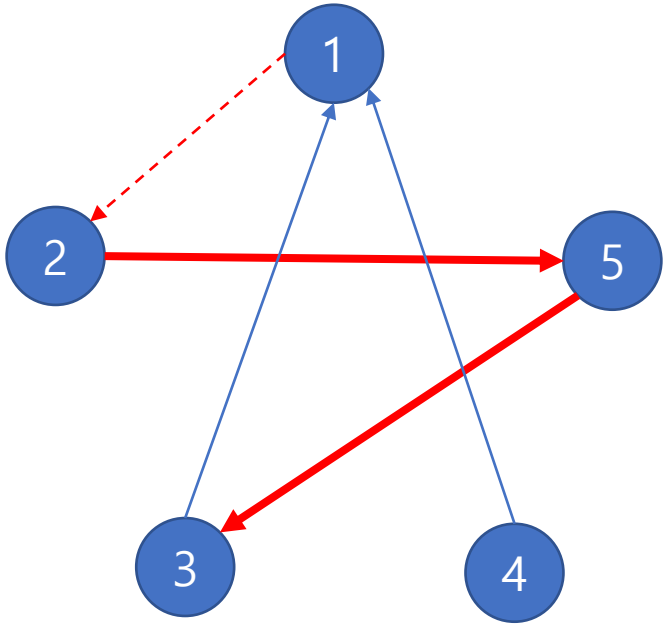
Solution (합성함수와 쿼리 2)

- 서브태스크 4
- 현재의 $f(1)$값으로 화살표를 보낸 후, 남은 거리만큼 다시 이동해봅시다.



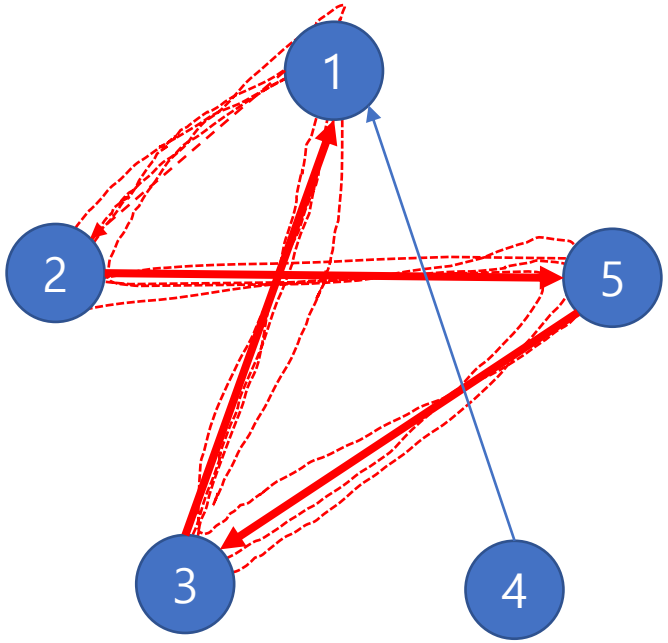
Solution (합성함수와 쿼리 2)

- 서브태스크 4
- $f(1)$으로 보낸 후 다시 1로 돌아오지 않는다면…
- 역시 그냥 희소 테이블을 사용해도 올바른 결과가 보장됩니다.



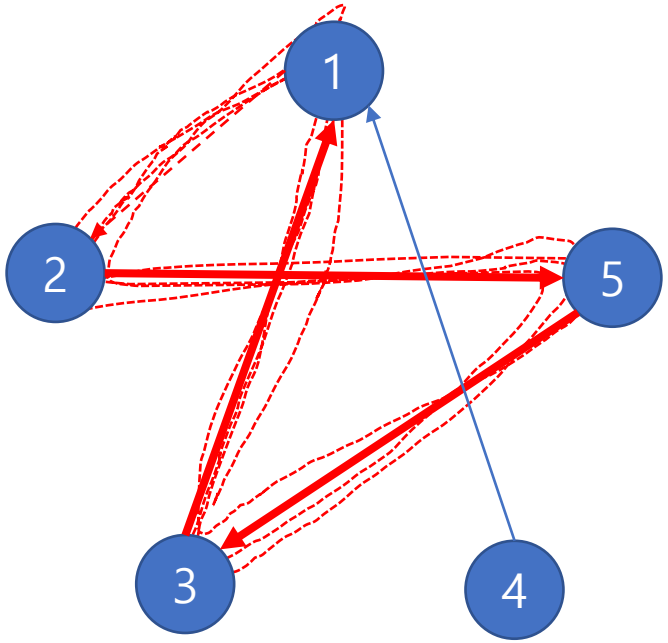
Solution (합성함수와 쿼리 2)

- 서브태스크 4
- $f(1)$으로 보낸 후 다시 1로 돌아온다면…
- 1은 다시 $f(1)$으로 보낼 것이고, 똑같은 경로가 계속 반복됩니다!



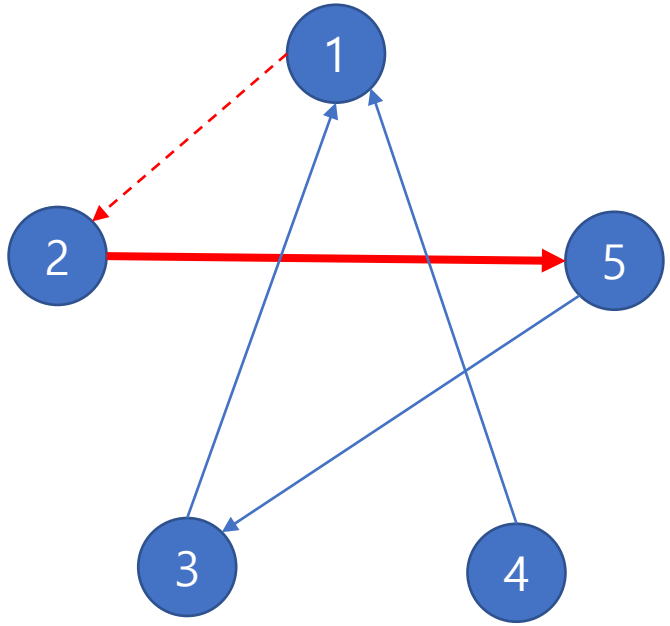
Solution (합성함수와 쿼리 2)

- 서브태스크 4
- 아래같은 경우, 1에서 4번 이동할 때마다 다시 1로 돌아옵니다.
- 따라서 남은 거리를 4로 나눈 나머지를 취해도 결과는 같습니다.



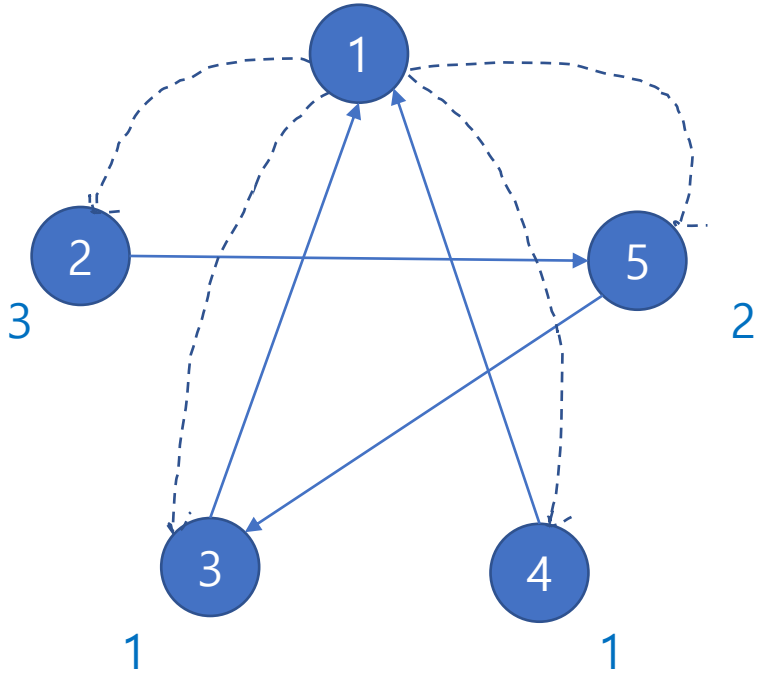
Solution (합성함수와 쿼리 2)

- 서브태스크 4
- 나머지를 취하면, 나머지의 특성에 따라 결국에는 1로 돌아오지 않는 경로를 계산하게 됩니다.



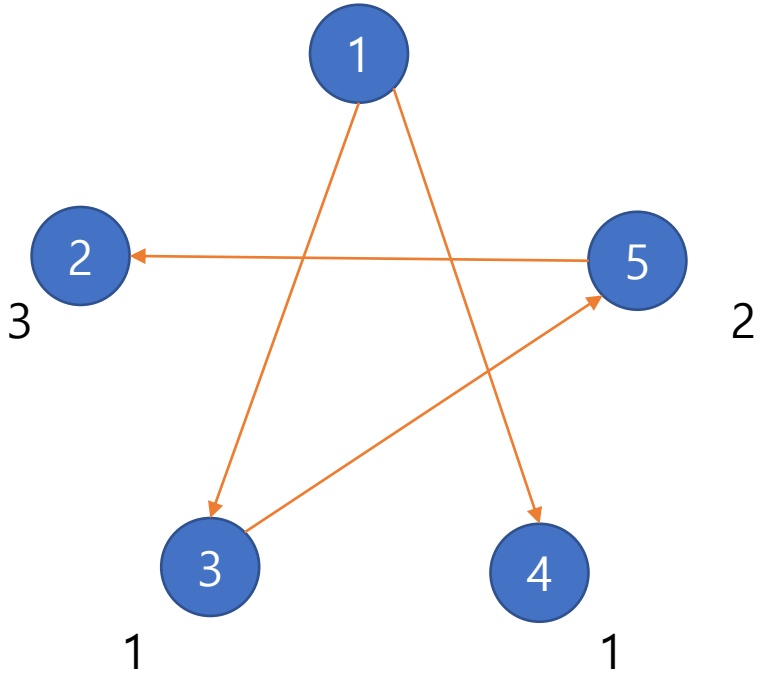
Solution (합성함수와 쿼리 2)

- 서브태스크 4
- 이를 효율적으로 구현하려면, 1이 2, 3, …, N으로 보냈을 때 각각 얼마 후에 1로 돌아오는지를 알아야 합니다.



Solution (합성함수와 쿼리 2)

- 서브태스크 4
- 이는 1에서 역방향 DFS를 해주면 $O(N)$에 구할 수 있습니다.
- 역방향 DFS에서 방문하지 않은 점들은 절대로 1로 오지 않음을 알 수 있습니다.



Solution (합성함수와 쿼리 2)

- 서브태스크 4
- 각 점에서 1까지 오는 거리를 모두 알았습니다.
- 이 정보를 이용해서 다음과 같은 알고리즘을 진행할 수 있습니다.

Solution (합성함수와 쿼리 2)

- 서브태스크 4
- 현재 점에서 m만큼 가면 1을 만나는지? (1까지의 거리와 비교)
- 1을 만나지 않으면 바로 희소 테이블 적용
- 1을 만나면...?

Solution (합성함수와 쿼리 2)

- 서브태스크 4
- 1을 만나는 경우 1까지만 이동한다고 생각합니다.
- m에서 이동한 거리를 빼서 남은 거리를 계산합니다.
- 1에서 f(1)로 보낸 후, 남은 거리만큼 이동하면 1을 만나는지?
- (역시 f(1)에서 1까지 가는 거리와 비교합니다)
- 1을 만나지 않으면 바로 희소 테이블 적용
- 1을 만나면...?

Solution (합성함수와 쿼리 2)

- 서브태스크 4
- 1을 만나는 경우 사이클이 형성됩니다. (경로가 반복됨)
- 남은 거리를 사이클의 길이로 나눈 나머지를 취합시다.
- 구한 나머지만큼 이동한 결과를 희소 테이블로 구해줍니다.

- 이 과정을 모두 구현하면 서브태스크 4를 통과할 수 있습니다.
- 시간복잡도는 $O(N \log 10^9 + Q \log 10^9)$ 입니다.