

제 1회 과수원컵 해설

Official Solutions

by

제 1회 과수원컵 운영진

2026년 10월 4일

Staff

운영

- cywohoy
- kyia
- saturnalorbit

출제

- _junick
- cywohoy
- kyia
- carrot
- saturnalorbit

검수

- 0000000000
- DonutMaster
- isthis_dejavu
- magic_spirit
- rlatjwls3333
- toycartoon
- yyyy7089

문제	의도한 난이도	출제자
2A 맛있는 사과	Prelude	saturnalorbit
2B 더블사과	Opening	_junick
2C 사과 수확	Opening	saturnalorbit
2D 사과 포장	Opening	saturnalorbit
2E 사과 상자 분배	Opening	saturnalorbit
2F 사과 지우기	Opening	_junick

문제	의도한 난이도	출제자
1A/2G 당도 측정	Middle	saturnalorbit
2H 리부트 과수원	Middle	saturnalorbit
1E 마법의 사과 탈무드 이야기	Middle	saturnalorbit
1C/2I 사과 피라미드 게임	Middle	kyia
1J 역시청사과보다는황금사과지	Finale	saturnalorbit
1I 과수원을 운영하는 나	Finale	saturnalorbit
1B 사과의 감시자	Finale	carrot
2J 사과나무 심기	Finale	saturnalorbit

문제	의도한 난이도	출제자
1H 사과나무 삼합점	Finale	carrot
1G RAM (Rotating Auto-Mobiles)	Encore	saturnalorbit
1F 노크한 과수원은 똑똑한 과수원이다	Encore	saturnalorbit
1D 반복을 좋아하는 떼랴 어린이	Encore	cywohoy

2A. 맛있는 사과

implementation

출제진 의도 – **Prelude**

- **Div. 2** — 처음 푼 참가자: **AdTerram**, 42초
- 출제자: saturnalorbit

2A. 맛있는 사과

- 크기와 당도를 입력받은 뒤, $S \geq A$ 이고 $W \geq B$ 이면 YES를, 그렇지 않으면 NO를 출력하면 됩니다.

2B. 더블사과

ad_hoc

출제진 의도 – **Opening**

- **Div. 2** — 처음 푼 참가자: **AdTerram**, 4분 19초
- 출제자: _junick

2B. 더블사과

- **서브태스크 1 (7점):** $N \leq 4$ — 가능한 교환을 모두 시도하는 완전 탐색으로 해결할 수 있습니다.
- **서브태스크 2 (8점): A, B만 등장**
- 잘못 포장된 상자는 모두 AB 형태입니다.
- 이런 상자의 개수는 짝수이며, AB 상자 두 개를 골라 한 번 교환하면 AA, BB로 만들 수 있습니다.
- 따라서 잘못 포장된 상자의 수를 M 이라 하면 답은 $M/2$ 입니다.
- 서브태스크 3에서도 같은 방식으로 가능한 교환을 직접 찾아가며 해결할 수 있습니다.

2B. 더블사과

- 이미 같은 품종 두 그루가 들어 있는 상자는 건드릴 필요가 없습니다.
- 잘못 포장된 상자는 순서를 무시하면 AB, AC, BC 세 종류뿐입니다.
- 각각의 개수를 x, y, z 라 합시다.
- 전체 A의 개수가 짝수이므로 $x + y$ 는 짝수입니다.
- 같은 이유로 $x + z, y + z$ 도 짝수이므로 x, y, z 의 홀짝은 모두 같습니다.

2B. 더블사과

- x, y, z 가 모두 짝수라면 같은 종류의 잘못된 상자를 두 개씩 짝지으면 됩니다.
- 예를 들어 AB 두 상자는 한 번의 교환으로 AA, BB로 만들 수 있습니다.
- 따라서 이 경우 필요한 작업 횟수는 $(x + y + z)/2$ 입니다.
- x, y, z 가 모두 홀수라면 각 종류에서 하나씩 남기고 나머지를 같은 방법으로 처리합니다.
- 마지막에 AB, AC, BC 하나씩 남는데, 이 세 상자는 두 번의 교환으로 모두 올바르게 만들 수 있습니다.
- 따라서 이 경우 필요한 작업 횟수는 $(x + y + z + 1)/2$ 입니다.

2B. 더블사과

- 잘못 포장된 상자의 총개수를 $M = x + y + z$ 라 합시다.
- 잘못 포장된 상자는 최종적으로 내용이 바뀌어야 하므로 적어도 한 번의 작업에 포함되어야 합니다.
- 한 번의 작업은 최대 두 상자에만 영향을 줄 수 있으므로 최소 $\lceil M/2 \rceil$ 번의 작업이 필요합니다.
- 앞의 구축은 항상 정확히 $\lceil M/2 \rceil$ 번의 작업으로 모든 상자를 고칩니다.
- 따라서 답은 단순히 $\lceil M/2 \rceil$ 입니다.
- 문자열을 한 번 순회하여 잘못 포장된 상자의 개수만 세면 되므로 전체 시간 복잡도는 $\mathcal{O}(N)$ 입니다.

2C. 사과 수확

bitmask, ad_hoc

출제진 의도 – **Opening**

- **Div. 2** — 처음 푼 참가자: **AdTerram**, 9분 22초
- 출제자: saturnalorbit

2C. 사과 수확

- **서브태스크 1 (10점)** — 어떤 품종도 두 나무에 동시에 존재하지 않으므로 작업을 할 수 없습니다. 따라서 처음부터 가장 가치가 큰 나무가 답입니다.
- **서브태스크 2 (30점)** — 한 나무에만 있는 품종은 절대로 수확되지 않고, 두 나무에 있는 품종은 마지막에 남을 수 없습니다.
- **서브태스크 3 (60점)** — 각 품종의 사과 개수의 홀짝을 생각하면 같은 관찰을 일반화할 수 있습니다.

2C. 사과 수확

- 각 품종을 독립적으로 생각합시다.
- 한 번의 작업으로 해당 품종의 사과는 정확히 2개씩 사라지므로, 사과 개수의 홀짝은 변하지 않습니다.
- 더 이상 작업할 수 없다면 같은 품종의 사과가 두 나무 이상에 남아 있을 수 없습니다.
- 따라서 처음에 홀수 개 존재한 품종은 정확히 하나, 짝수 개 존재한 품종은 하나도 남지 않습니다.
- 비트마스크로 나타내면 마지막에 남는 모든 사과의 집합은 $X = A_1 \oplus A_2 \oplus \cdots \oplus A_N$ 으로 고정됩니다.

2C. 사과 수확

- i 번 나무에 마지막까지 남을 수 있는 품종은 처음부터 i 번 나무에 있었고, X 에도 포함된 품종뿐입니다.
- 따라서 그 가치의 합은 A_i & X 를 넘을 수 없습니다.
- 이 상한은 항상 달성할 수 있습니다. 우선 i 번 나무를 선택하지 않고 나머지 나무들끼리 가능한 작업을 끝까지 수행합니다.
- A_i & X 에 포함된 각 품종은 i 번 나무를 제외하면 처음에 짝수 개 존재하므로, 이 과정이 끝나면 나머지 나무에서는 모두 사라집니다.
- 이후 작업에서 i 번 나무를 선택하더라도 이 품종들과 공통된 사과를 가진 다른 나무가 없으므로, 이 사과들은 끝까지 남길 수 있습니다.

2C. 사과 수확

- 따라서 i 번 나무에 남길 수 있는 최대 가치는 정확히 A_i & X 입니다.
- 모든 나무 중 하나를 마지막에 남길 수 있으므로 답은 $\max_i(A_i \text{ \& } X)$ 입니다.
- X 를 한 번 계산한 뒤 모든 A_i 를 확인하면 됩니다.
- 전체 시간 복잡도는 $\mathcal{O}(N)$, 메모리 복잡도는 $\mathcal{O}(1)$ 입니다.

2D. 사과 포장

math, ad_hoc

출제진 의도 – **Opening**

- **Div. 2** — 처음 푼 참가자: **AdTerram**, 15분 16초
- 출제자: saturnalorbit

2D. 사과 포장

- **서브태스크 1 (10점):** $R + G \leq 20$
- 빨간 사과가 놓일 위치의 집합을 모두 완전 탐색하고, 각 배치가 두 조건을 만족하는지 직접 확인할 수 있습니다.
- **서브태스크 2 (30점):** $K \geq R + G - 1$ — 색 변화 횟수 제한은 항상 만족하므로, 같은 색이 세 번 연속하지 않도록 배치할 수 있는지만 확인하면 됩니다.
- 이 경우 더 많은 색의 사과를 두 개씩 묶어 다른 색의 사과 사이에 넣으면 됩니다.
- 따라서 예를 들어 $R \geq G$ 라면 $R \leq 2(G + 1)$ 인지 확인하면 됩니다.

2D. 사과 포장

- 같은 색이 연속한 부분을 하나의 **묶음**으로 생각합시다.
- 빨간 사과 묶음의 수를 x , 청사과 묶음의 수를 y 라 하겠습니다.
- 한 묶음에는 같은 색의 사과가 1개 또는 2개만 들어갈 수 있으므로 $\lceil R/2 \rceil \leq x \leq R$, $\lceil G/2 \rceil \leq y \leq G$ 입니다.
- 두 색의 묶음은 번갈아 나타나야 하므로 $|x - y| \leq 1$ 이어야 합니다.
- 묶음이 총 $x + y$ 개라면 색이 바뀌는 횟수는 정확히 $x + y - 1$ 입니다.
- 따라서 조건을 만족하는 x, y 중 $x + y$ 를 최소화하면 됩니다.

2D. 사과 포장

- 처음에는 각각 가능한 최소 묶음 수인 $x = \lceil R/2 \rceil, y = \lceil G/2 \rceil$ 로 둡니다.
- 이미 $|x - y| \leq 1$ 이면 이것이 최솟값입니다.
- 그렇지 않다면 작은 쪽의 묶음 수를 $|x - y| \leq 1$ 이 될 때까지 늘리는 것이 최선입니다.
- 이 과정에서 $x > R$ 또는 $y > G$ 가 되면 가능한 포장이 존재하지 않습니다.
- 그렇지 않다면 얻은 $x + y - 1$ 이 가능한 색 변화 횟수의 최솟값이므로, 이 값이 K 이하인지 확인하면 됩니다.
- 전체 시간 복잡도는 $\mathcal{O}(1)$ 입니다.

2E. 사과 상자 분배

ad_hoc, parity

출제진 의도 – **Opening**

- **Div. 2** — 처음 푼 참가자: **AdTerram**, 27분 50초
- 출제자: saturnalorbit

2E. 사과 상자 분배

- **서브태스크 1 (10점):** $N \leq 11$ — 남아 있는 상자들의 집합을 상태로 두고 가능한 제거를 모두 시도할 수 있습니다.
- **서브태스크 2 (30점):** $N \leq 3\,000$ — 아래에서 얻는 조건을 각 i 마다 직접 확인하면 $\mathcal{O}(N^2)$ 에 해결할 수 있습니다.
- **서브태스크 3 (60점): 추가 제한이 없음** — 같은 조건을 prefix minimum과 suffix minimum으로 확인하면 $\mathcal{O}(N)$ 에 해결할 수 있습니다.

2E. 사과 상자 분배

- 서로 이웃한 두 상자는 항상 함께 제거할 수 있습니다. 둘 중 사과가 많은 상자에서 적은 상자 방향을 고르면 됩니다.
- i 가 홀수라면 i 의 왼쪽과 오른쪽에 각각 짝수 개의 상자가 있습니다.
- 따라서 양쪽에서 이웃한 상자끼리 계속 제거하면 항상 i 번 상자를 마지막까지 남길 수 있습니다.
- 반대로 i 가 짝수라면 양쪽에 홀수 개의 상자가 있으므로, 어떤 순간에는 반드시 i 를 사이에 둔 두 상자를 함께 제거해야 합니다.

2E. 사과 상자 분배

- 실제로 i 를 가로질러 두 상자를 제거한다고 합시다.
- 선택한 상자의 사과 수가 A_i 보다 크다면, i 번 상자는 선택한 상자보다 사과가 적으므로 i 를 지나칠 수 없습니다.
- 따라서 i 를 제거하지 않고 가로지르는 쌍을 만들려면 i 의 양쪽에 모두 A_i 보다 작은 값이 존재해야 합니다.
- 이 조건은 충분하기도 합니다. 왼쪽과 오른쪽의 최솟값을 각각 L, R 이라 합시다.
- $L, R < A_i$ 이고, 예를 들어 $L > R$ 이라면 L 이 들어 있는 상자에서 오른쪽을 선택하면 제거할 수 있습니다.
- 이 한 번의 제거 이후 양쪽에 짝수 개의 상자가 남으므로, 나머지는 이웃한 상자끼리 제거하면 됩니다.

2E. 사과 상자 분배

- 따라서 i 번 상자가 마지막까지 남을 수 있는 필요충분조건은 다음과 같습니다.
- i 가 홀수이면 항상 가능합니다.
- i 가 짝수이면 왼쪽과 오른쪽에 각각 A_i 보다 작은 값이 하나 이상 존재해야 합니다.
- 즉 짝수 i 에서는 $\min(A_1, \dots, A_{i-1}) < A_i$ 와 $\min(A_{i+1}, \dots, A_N) < A_i$ 를 확인하면 됩니다.
- prefix minimum과 suffix minimum을 전처리하면 모든 i 를 $\mathcal{O}(N)$ 에 판별할 수 있습니다.
- 전체 시간 복잡도는 $\mathcal{O}(N)$, 메모리 복잡도는 $\mathcal{O}(N)$ 입니다.

2F. 사과 지우기

tree_diameter, greedy

출제진 의도 - Opening

- **Div. 2** — 처음 푼 참가자: **js7777**, 32분 18초
- 출제자: _junick

2F. 사과 지우기

- **서브태스크 1 (3점):** $N \leq 8$ — 남아 있는 정점의 집합을 상태로 두고 가능한 행동을 모두 완전 탐색할 수 있습니다.
- **서브태스크 2 (2점):** $M = 0$ — 모든 컴포넌트가 정점 하나이므로 정점마다 한 번의 행동이 필요합니다. 답은 N 입니다.
- **서브태스크 3 (4점): 모든 정점의 차수가 2 이하**
- 주어진 그래프는 포레스트이고 모든 정점의 차수가 2 이하이므로, 각 컴포넌트는 하나의 단순 경로입니다.
- 따라서 각 컴포넌트를 한 번의 행동으로 전부 지울 수 있습니다.

2F. 사과 지우기

- 하나의 컴포넌트만 생각하고, 정점 수를 S , 처음 지름을 d 라 합시다.
- 정점을 지운 뒤 생기는 모든 컴포넌트는 원래 트리의 부분트리입니다.
- 따라서 이후 어떤 컴포넌트의 지름도 처음의 d 보다 커질 수 없습니다.
- 한 번의 행동으로 지울 수 있는 정점은 항상 최대 d 개이므로, 모든 정점을 지우려면 적어도 $\lceil S/d \rceil$ 번의 행동이 필요합니다.
- 이제 이 하한을 항상 달성할 수 있음을 보이겠습니다.

2F. 사과 지우기

- 처음 트리의 지름을 이루는 경로 하나를 P 라 합시다. P 에는 정확히 d 개의 정점이 있습니다.
- P 밖에 정점이 남아 있다면, P 의 정점을 건드리지 않고 바깥쪽의 리프부터 하나씩 지울 수 있습니다.
- 이런 순서로 정점들을 지우면 매 순간 남은 그래프는 하나의 트리이고, P 도 그대로 남아 있습니다.
- 따라서 지우는 순서를 d 개씩 묶어 한 번의 행동으로 처리할 수 있습니다.
- 이 동안 P 가 계속 남아 있으므로 현재 컴포넌트의 지름도 계속 d 입니다.

2F. 사과 지우기

- P 바깥의 $S - d$ 개 정점을 모두 지운 뒤에는 P 만 남습니다.
- 마지막 행동으로 P 의 d 개 정점을 한꺼번에 모두 지웁니다.
- 따라서 필요한 행동 수는 $\lceil (S - d)/d \rceil + 1 = \lceil S/d \rceil$ 입니다.
- 앞에서 얻은 하한과 같으므로, 정점 수가 S 이고 지름이 d 인 트리의 답은 정확히 $\lceil S/d \rceil$ 입니다.

2F. 사과 지우기

- **서브태스크 4 (6점): 모든 컴포넌트의 지름이 3 이하**
- 각 컴포넌트의 정점 수와 지름만 알면 앞의 공식으로 바로 답을 계산할 수 있습니다.
- 지름이 1 이면 정점 하나, 2이면 간선 하나, 3이면 중심 하나와 여러 리프로 이루어진 형태입니다.
- **서브태스크 5 (12점): $N \leq 2000$**
- 각 정점에서 DFS를 수행하여 모든 정점 사이의 거리를 구하면 각 컴포넌트의 지름을 충분히 빠르게 구할 수 있습니다.
- 이후 각 컴포넌트마다 $\lceil S/d \rceil$ 을 계산합니다.

2F. 사과 지우기

- **서브태스크 6 (18점):** $M = N - 1$
- 그래프 전체가 하나의 트리이므로 정점 수는 N 입니다.
- 임의의 정점에서 가장 먼 정점 u 를 찾고, 다시 u 에서 가장 먼 정점까지의 거리를 구하면 트리의 지름을 구할 수 있습니다.
- 지름의 정점 수를 d 라 하면 답은 $\lceil N/d \rceil$ 입니다.
- 두 번의 DFS 또는 BFS로 $\mathcal{O}(N)$ 에 해결할 수 있습니다.

2F. 사과 지우기

– 서브태스크 7 (55점): 추가 제한이 없음

- 서로 다른 초기 컴포넌트 사이에는 어떤 행동으로도 영향을 줄 수 없으므로 각각 독립적으로 처리합니다.
- 각 컴포넌트에서 정점 수 s 와 지름의 정점 수 d 를 구하고 $\lceil s/d \rceil$ 을 답에 더합니다.
- 포레스트 전체에서도 각 정점과 간선을 상수 번만 방문하면 됩니다.
- 전체 시간 복잡도는 $\mathcal{O}(N + M)$ 입니다.

1A/2G. 당도 측정

ad_hoc, constructive

출제진 의도 - Middle

- **Div. 1** — 처음 푼 참가자: **middle_man**, 134분 11초
- **Div. 1** — 처음 푼 팀: **trickcal**, 25분 28초
- **Div. 2** — 처음 푼 참가자: **hjroh0315**, 197분 24초
- 출제자: saturnalorbit

1A/2G. 당도 측정

- i 번째 길이 K 구간에서 최댓값의 위치를 p_i 라 하고, $s_i = p_i - i$ 라 합시다.
- $b_i = 1$ 이면 $s_i = 0$, $b_i = 2$ 이면 $s_i = K - 1$, $b_i = 0$ 이면 $1 \leq s_i \leq K - 2$ 입니다.
- $s_i > 0$ 이면 현재 최댓값은 다음 구간에도 남아 있습니다.
- 이 경우 다음 최댓값은 그대로 유지되거나, 새로 들어오는 오른쪽 끝 원소로 바뀌는 두 경우뿐입니다.
- 따라서 $s_i > 0$ 이면 $s_{i+1} = s_i - 1$ 또는 $s_{i+1} = K - 1$ 입니다.
- 반대로 $s_i = 0$ 이면 현재 최댓값이 빠지므로 s_{i+1} 은 자유롭게 정할 수 있습니다.

1A/2G. 당도 측정

- **서브태스크 1 (5점):** $K = N$ — 구간이 하나뿐이므로 b_1 에 맞는 위치에 최댓값을 놓으면 됩니다.
- **서브태스크 2 (10점):** 모든 $b_i = 0$
- 최댓값을 나타내는 s_i 는 매번 1씩 감소해야 하므로, 길이 $N - K + 1$ 의 0 구간이 $K - 2$ 보다 길면 불가능합니다.
- 그렇지 않다면 하나의 위치를 모든 구간의 최댓값으로 사용할 수 있습니다.
- **서브태스크 3 (15점):** 모든 $b_i \neq 0$
- $K > 2$ 에서 한 번 2가 나타난 뒤 바로 1로 바뀔 수는 없습니다. 그 사이에는 $K - 2$ 개의 0이 필요합니다.
- 따라서 이 경우에는 2 뒤에 다시 1이 나타나지만 확인하면 됩니다.

1A/2G. 당도 측정

- **서브태스크 4 (25점):** $K = 3$
- 이때 가능한 s_i 는 0, 1, 2 뿐이고, 각각 $b_i = 1, 0, 2$ 와 정확히 대응합니다.
- 따라서 b 를 보면 s 가 자동으로 결정됩니다.
- 남은 것은 인접한 두 상태가 앞에서 얻은 전이 규칙을 만족하는지 확인하는 것뿐입니다.
- 즉 일반 문제에서도 결국 b 와 맞는 $s_1, s_2, \dots$ 를 하나 만드는 것이 핵심입니다.

1A/2G. 당도 측정

- 일반적인 경우에는 b 에서 연속한 0들의 묶음을 하나씩 생각합니다.
- 하나의 0 묶음 안에서는 새로운 오른쪽 끝이 최댓값이 될 수 없으므로, s 는 매번 정확히 1씩 감소합니다.
- 따라서 모든 0 묶음의 길이는 $K-2$ 이하여야 합니다.
- 묶음 바로 왼쪽이 2라면 첫 s 는 $K-2$ 로 강제됩니다.
- 반면 왼쪽이 1이거나 묶음이 맨 앞에 있다면, 길이가 z 인 묶음에 $z, z-1, \dots, 1$ 을 배정할 수 있습니다.
- 특히 $2, 0, \dots, 0, 1$ 에서 두 0이 아닌 원소 사이의 0 개수는 정확히 $K-2$ 여야 합니다.

1A/2G. 당도 측정

- 위 조건을 모두 만족하면 각 0 묶음의 s_i 를 바로 정할 수 있고, 1 과 2 인 위치의 s_i 도 이미 정해져 있습니다.
- 따라서 모든 $p_i = i + s_i$ 를 선형 시간에 구축할 수 있습니다.
- 이제 실제 순열 a 를 만들어야 합니다.
- p_i 는 감소하지 않으며, 값이 변할 때마다 더 오른쪽 위치로 이동합니다.
- 따라서 최댓값이 되는 위치들을 등장 순서대로 보면 항상 위치가 증가합니다.

1A/2G. 당도 측정

- $p_i \neq p_{i+1}$ 인 순간만 보면 필요한 대소관계를 알 수 있습니다.
- $p_i > i$ 라면 이전 최댓값이 다음 구간에도 남아 있는데 새로운 오른쪽 끝으로 최댓값이 바뀐 것이므로 $a_{p_i} < a_{p_{i+1}}$ 입니다.
- $p_i = i$ 이고 $p_{i+1} < i + K$ 라면 p_{i+1} 도 이전 구간에 있었으므로 $a_{p_{i+1}} < a_{p_i}$ 입니다.
- $p_i = i$ 이고 $p_{i+1} = i + K$ 라면 두 위치가 같은 구간에 등장한 적이 없으므로 별도의 대소관계가 필요하지 않습니다.
- 이렇게 얻는 관계는 등장 순서상 이웃한 최댓값 위치 사이에만 존재합니다.

1A/2G. 당도 측정

- 어떤 구간의 최댓값도 아닌 위치에는 가장 작은 값들을 임의로 배정합니다.
- 최댓값이 되는 위치들 사이의 대소관계 그래프는 하나의 경로의 일부이므로 사이클이 없습니다.
- 따라서 간단한 위상 정렬을 수행하고, 이 위치들에는 나머지 큰 값들을 위상 순서대로 배정합니다.
- 그러면 각 구간에서 p_i 가 실제 최댓값이 되고, 주어진 b 와 정확히 같은 결과를 얻습니다.
- 모든 과정은 선형 시간에 수행할 수 있으므로 전체 시간 복잡도는 $\mathcal{O}(N)$ 입니다.

2H. 리부트 과수원

trees, dp_tree, rerooting

출제진 의도 - Middle

- **Div. 2** — 처음 푼 참가자: **ThatOnePythonUser**, 149분 20초
- 출제자: saturnalorbit

2H. 리부트 과수원

- **서브태스크 1 (10점):** $N \leq 20$
- 마지막에 남길 정점의 집합을 완전 탐색하고, 하나의 경로를 이루는 경우에 얻는 값을 직접 계산할 수 있습니다.
- **서브태스크 2 (15점): 주어진 트리가 하나의 단순 경로**
- 마지막에 남는 집하장들은 원래 경로의 연속한 구간입니다.
- 왼쪽에서 제거하는 비용과 오른쪽에서 제거하는 비용의 누적합을 구하면, 모든 구간의 값을 계산할 수 있습니다.
- 이를 적절히 스위핑하면 $\mathcal{O}(N)$ 에 해결할 수 있습니다.

2H. 리부트 과수원

- **서브태스크 3 (15점):** $b_i = 0$, 모든 $c_{u,v} \geq 0$
- 남기는 경로에 간선이 하나 더 포함되면, 그 간선을 철거해서 얻을 수 있었던 음이 아닌 값을 포기하게 됩니다.
- 따라서 최적해 중 하나는 정점 하나만 남깁니다.
- 각 정점을 마지막 정점으로 정했을 때의 값을 rerooting으로 계산하면 $\mathcal{O}(N)$ 에 해결할 수 있습니다.
- **서브태스크 4 (20점):** $c_{u,v} = c_{v,u}$
- 간선을 어느 방향으로 철거해도 얻는 값이 같으므로 방향을 신경 쓸 필요가 없습니다.

2H. 리부트 과수원

- 서브태스크 4에서 간선 uv 의 공통 정비 손익을 c_{uv} 라 합시다.
- 모든 간선을 철거한다고 생각하면 $\sum c_{uv}$ 를 얻습니다.
- 최종 경로에 정점 u 를 남기면 b_u 를 얻고, 간선 uv 를 남기면 그 간선의 c_{uv} 는 얻지 못합니다.
- 따라서 최종 값은 상수 $\sum c_{uv}$ 에, 경로 위 정점의 가중치 b_u 와 간선의 가중치 $-c_{uv}$ 를 더한 값입니다.
- 즉, 트리에서 최대 가중치 단순 경로를 구하는 문제이며 트리 DP로 $\mathcal{O}(N)$ 에 해결할 수 있습니다.

2H. 리부트 과수원

- 이제 방향에 따라 정비 손익이 다른 일반적인 경우를 생각합시다.
- 방향 간선 $u \rightarrow v$ 에 대해, uv 를 끊었을 때 u 쪽 부분을 전부 철거하여 마지막에 v 쪽으로 합치는 값을 $f(u, v)$ 라 합시다.
- u 의 다른 이웃을 w 라 하면, 먼저 w 쪽 부분을 모두 u 쪽으로 철거한 뒤 마지막에 u 를 v 쪽으로 철거합니다.
- 따라서 $f(u, v) = c_{u,v} + \sum_{w \neq v} f(w, u)$ 입니다.
- 한 방향의 값들을 먼저 구한 뒤 rerooting하면 모든 방향 간선의 f 를 $\mathcal{O}(N)$ 에 계산할 수 있습니다.

2H. 리부트 과수원

- 정점 u 의 새로운 가중치를 $b_u + \sum_v f(v, u)$ 로 둡니다.
- 이는 u 만 최종 경로에 남기고, u 에 붙은 모든 방향의 부분트리를 철거한다고 생각한 값입니다.
- 그런데 최종 경로에 간선 uv 를 남기면 u 쪽을 v 로 철거하지도 않고, v 쪽을 u 로 철거하지도 않습니다.
- 따라서 경로에 간선 uv 를 사용할 때 $f(u, v) + f(v, u)$ 를 빼 주면 됩니다.
- 즉 간선 uv 의 새로운 가중치를 $-f(u, v) - f(v, u)$ 로 둡니다.

2H. 리부트 과수원

- 그러면 임의의 최종 경로에서 실제로 얻는 값은, 그 경로의 새로운 정점 가중치와 간선 가중치의 합과 정확히 같습니다.
- 따라서 원래 문제는 다시 **트리에서 최대 가중치 단순 경로를 구하는 문제**가 됩니다.
- 정점 u 에서 아래로 이어지는 최적 경로를 DP로 관리하고, 가장 좋은 두 방향을 이어 붙이면 u 를 지나는 최적 경로를 얻을 수 있습니다.
- 정점과 간선 가중치가 음수일 수도 있으므로, 각 방향을 사용하는 것이 손해라면 사용하지 않습니다.
- rerooting과 최대 경로 DP를 모두 합쳐 전체 시간 복잡도는 $\mathcal{O}(N)$, 메모리 복잡도는 $\mathcal{O}(N)$ 입니다.

1E. 마법의 사과 탈무드 이야기

ad_hoc, math, graphs

출제진 의도 - Middle

- **Div. 1** — 처음 폰 참가자: **mjkim112358**, 90분 6초
- **Div. 1** — 처음 폰 팀: **choisteam**, 40분 5초
- 출제자: saturnalorbit

1E. 마법의 사과 탈무드 이야기

- 서로 다른 연결 요소 사이에는 영향을 줄 수 없으므로, 각 연결 요소를 독립적으로 해결합니다.
- 성장도가 이미 K 인 나무가 하나라도 있다면, 그 나무에서 시작하여 K 를 주변으로 전파할 수 있습니다.
- 성장도가 K 인 나무와 연결된 나무는 한 번의 마법으로 바로 K 가 됩니다.
- 따라서 이 경우 필요한 횟수는 처음에 성장도가 K 가 아닌 나무의 개수입니다.
- **서브태스크 2 (2점):** $A_i \in \{0, K\}$ 는 각 연결 요소에 K 인 나무가 하나라도 있는지만 확인하면 됩니다.

1E. 마법의 사과 탈무드 이야기

- 이제 연결 요소에 성장도가 K 인 나무가 하나도 없다고 합시다.
- 처음으로 어떤 나무를 K 로 만드는 데 필요한 최소 횟수를 t 라 하겠습니다.
- 최초의 K 가 만들어진 순간 다른 모든 나무의 성장도는 아직 K 보다 작습니다.
- 이후에는 K 를 연결 요소 전체로 전파하여 나머지 $S - 1$ 개의 나무를 각각 한 번씩 갱신하면 됩니다.
- 따라서 연결 요소의 답은 $t + S - 1$ 입니다.
- 모든 성장도가 0이거나, 정점이 하나뿐인데 성장도가 K 보다 작다면 불가능합니다.

1E. 마법의 사과 탈무드 이야기

- 하나의 간선 양 끝의 초기 성장도를 $x \geq y$ 라 합시다.
- 두 나무를 번갈아 갱신하면 성장도는 피보나치 수열과 같은 방식으로 증가합니다.
- t 번의 마법을 사용한 뒤 만들 수 있는 값은 $F_{t+1}x + F_t y$ 입니다. 여기서 $F_0 = 0, F_1 = 1$ 입니다.
- **서브태스크 1 (7점)**에서는 각 연결 요소에 양수인 나무가 하나뿐입니다.
- 그 성장도를 x 라 하면 처음에는 인접한 0인 나무에 x 를 복사하고, 이후 두 나무를 번갈아 갱신하면 됩니다.
- 따라서 $F_{t+1}x \geq K$ 가 되는 최소 t 를 찾으면 됩니다.

1E. 마법의 사과 탈무드 이야기

- 서브태스크 3 (12점): 각 연결 요소가 완전 그래프
- 초기 성장도가 가장 큰 두 나무를 $x \geq y$ 라 합시다.
- 두 나무는 반드시 연결되어 있으므로, 이 둘만 번갈아 갱신할 수 있습니다.
- 따라서 $F_{t+1}x + F_t y \geq K$ 가 되는 최소 t 를 사용하면 됩니다.
- 일반 그래프에서는 가장 큰 두 나무가 서로 연결되어 있다는 보장이 없습니다.
- 그렇다면 여러 간선을 돌아다니며 값을 합치는 것이 한 간선만 사용하는 것보다 더 빠를 수 있는지를 알아야 합니다.

1E. 마법의 사과 탈무드 이야기

- 현재 성장도 배열에 대해 다음 값을 생각합니다.
- 간선 (u, v) 의 양 끝 값을 $x \geq y$ 라 할 때, 남은 마법 횟수가 t 라면 이 간선의 잠재력을 $F_{t+1}x + F_t y$ 로 정의합니다.
- 모든 간선 중 잠재력의 최댓값을 P_t 라 합니다.
- 한 번의 마법을 사용한 뒤의 P_{t-1} 은 사용하기 전의 P_t 를 넘을 수 없습니다.
- 이는 갱신된 값이 두 끝점의 합이라는 것과 $F_{t+1} = F_t + F_{t-1}$ 을 대입하여 확인할 수 있습니다.
- 따라서 t 번의 마법으로 어떤 나무를 K 까지 성장시킬 수 있다면, 처음부터 $P_t \geq K$ 여야 합니다.

1E. 마법의 사과 탈무드 이야기

- 반대로 어떤 간선 (u, v) 에서 $F_{t+1}x + F_t y \geq K$ 라면, 그 두 나무를 번갈아 갱신하여 실제로 t 번 안에 K 를 만들 수 있습니다.
- 따라서 최초의 K 를 만드는 최소 횟수는 모든 간선 (u, v) 에 대해 $x = \max(A_u, A_v)$, $y = \min(A_u, A_v)$ 로 두고, $F_{t+1}x + F_t y \geq K$ 가 되는 최소 t 의 최솟값입니다.
- 즉 여러 나무를 복잡하게 사용하는 것이 더 빠를 수 없으며, 항상 하나의 간선만 사용한 최적해가 존재합니다.
- 피보나치 수는 지수적으로 증가하므로 필요한 t 의 개수는 $\mathcal{O}(\log K)$ 입니다.

1E. 마법의 사과 탈무드 이야기

– 서브태스크 4 (79점): 추가 제한이 없음

- 각 연결 요소에서 이미 K 인 나무가 있으면, K 가 아닌 나무의 개수만 답에 더합니다.
- 그렇지 않으면 모든 간선을 확인하여 최초의 K 를 만드는 최소 횟수 t 를 구하고, 연결 요소의 크기가 S 라면 $t + S - 1$ 을 더합니다.
- 사용할 수 있는 간선이 없거나 모든 초기 성장도가 0이면 불가능합니다.
- 연결 요소별 결과를 모두 더하면 전체 답을 얻습니다.
- 전체 시간 복잡도는 $\mathcal{O}((N + M) \log K)$ 입니다.

1C/2I. 사과 피라미드 게임

ad_hoc, game_theory

출제진 의도 - Middle

- **Div. 1** — 처음 푼 참가자: **jwpassion1**, 9분 57초
- **Div. 1** — 처음 푼 팀: **notflake**, 20분 38초
- **Div. 2** — 처음 푼 참가자: **cirno**, 85분 29초
- 출제자: kyia

1C/2I. 사과 피라미드 게임

- 서브태스크 1 (3점): $N = 3$ 입니다.
- 피라미드에는 첫 번째 층의 인접한 두 사과 위에 있는 두 자리와 꼭대기 한 자리가 존재합니다.
- 각 위치에 놓을 수 있는 사과의 맛은 바로 아래 두 사과의 최대공약수의 약수 중 하나입니다.
- N 이 작으므로 가능한 모든 위치와 모든 약수를 직접 시도할 수 있습니다.
- 게임이 끝난 뒤 가능한 모든 경로의 점수 중 최댓값을 계산하면 됩니다.

1C/2I. 사과 피라미드 게임

- 서브태스크 2 (1점): 모든 $A_i = 1$ 입니다.
- 두 사과의 양의 공약수는 항상 1 뿐입니다.
- 따라서 어떤 순서로 게임을 진행하더라도 새로 놓이는 모든 사과의 맛은 1 입니다.
- 완성된 피라미드의 모든 사과의 맛이 1 이므로, 어떤 경로의 점수도

N

입니다.

- 따라서 답은 N 입니다.

1C/2I. 사과 피라미드 게임

- 서브태스크 3 (9점): 모든 $A_i = 2^i$ 입니다.
- 인접한 두 사과의 최대공약수는

$$\gcd(A_i, A_{i+1}) = 2^i$$

입니다.

- 따라서 해당 두 사과 위에 놓을 수 있는 가장 큰 사과의 맛은 2^i 입니다.
- 인접한 두 사과를 이용했을 때 얻을 수 있는 값을 계산하여 비교하면 됩니다.
- 이 경우

$$\max(A_i, A_{i+1}) + \gcd(A_i, A_{i+1}) = 3 \cdot 2^i$$

이므로 가장 좋은 위치를 쉽게 찾을 수 있습니다.

1C/2I. 사과 피라미드 게임

- 서브태스크 4 (20점): $N \leq 5$ 입니다.
- 각 단계에서 사과를 놓을 수 있는 모든 위치를 확인하고, 해당 위치에 놓을 수 있는 모든 약수를 시도할 수 있습니다.
- 현재 플레이어가 루루라면 가능한 결과 중 최댓값을, 파노라면 최솟값을 선택합니다.
- 따라서 게임 상태를 저장하면서 미니맥스를 계산하면 해결할 수 있습니다.
- 하지만 N 이 커지면 가능한 상태의 수가 급격하게 증가하므로, 전체 제한에서는 게임의 구조를 이용해야 합니다.

1C/2I. 사과 피라미드 게임

- 서브태스크 5 (7점): $A_i \neq 1$ 인 i 가 두 개 이하이며, $A_1, A_2, \dots, A_N$ 에 등장하는 서로 다른 값의 개수도 두 개 이하입니다.
- 1이 아닌 사과가 적기 때문에 1보다 큰 내부 사과를 만들 수 있는 위치가 제한됩니다.
- 인접한 두 사과 A_i, A_{i+1} 에 대해

$$B_i = \max(A_i, A_{i+1}) + \gcd(A_i, A_{i+1})$$

를 계산하면 각 위치에서 얻을 수 있는 가장 좋은 값을 비교할 수 있습니다.

- 이 관찰은 추가 제한이 없는 경우에도 그대로 사용할 수 있습니다.

1C/2I. 사과 피라미드 게임

- 서브태스크 6 (60점): 추가 제한이 없습니다.
- 먼저 새로 놓이는 사과의 맛이 1 보다 클 수 있는 조건을 생각합시다.
- 두 아래 사과의 맛을 x, y 라고 하면 새 사과의 맛은 $\gcd(x, y)$ 의 약수입니다.
- 따라서 새 사과의 맛이 1 보다 크다면 반드시

$$x > 1, \quad y > 1$$

이어야 합니다.

- 반대로 두 사과 중 하나라도 1 이면

$$\gcd(x, 1) = 1$$

이므로 새 사과의 맛은 반드시 1 입니다.

1C/2I. 사과 피라미드 게임

- 즉, 1인 사과 하나가 생기면 그 사과를 바로 아래에 가지는 모든 위쪽 사과는 1이 될 수밖에 없습니다.
- 따라서 파노가 어떤 위치에 1을 놓으면, 그 위치를 지나는 위쪽 경로에서 1보다 큰 사과가 계속 이어지는 것을 막을 수 있습니다.
- 파노는 자신의 차례마다 아직 비어 있고 아래 두 사과가 놓인 위치를 선택할 수 있으므로, 루루가 만든 큰 사과의 위쪽 전파를 차단할 수 있습니다.

1C/2I. 사과 피라미드 게임

차단 전략

파노는 루루가 만든 1보다 큰 사과가 같은 경로에서 연속해서 위쪽으로 이어지는 것을 막을 수 있습니다.

- 어떤 사과가 1보다 크려면 바로 아래의 두 사과가 모두 1보다 커야 합니다.
- 루루가 두 자식을 모두 1보다 크게 만든 경우, 그 위의 부모 위치가 다음으로 큰 사과를 만들 수 있는 위치가 됩니다.
- 파노는 그 부모 위치에 1을 놓아 이후의 전파를 차단할 수 있습니다.
- 이를 반복하면 루루가 만든 큰 사과가 여러 층에 걸쳐 하나의 경로에 연속해서 등장하는 것을 막을 수 있습니다.

1C/2I. 사과 피라미드 게임

- 따라서 어떤 경로에서 추가적으로 1보다 큰 내부 사과가 점수에 기여하더라도, 가장 아래에 있는 큰 내부 사과 하나만 고려하면 충분합니다.
- 가장 아래의 내부 사과는 반드시 첫 번째 층의 인접한 두 사과 A_i, A_{i+1} 위에 있습니다.
- 이 위치에 놓을 수 있는 사과의 최대 맛은

$$\gcd(A_i, A_{i+1})$$

입니다.

- 루루는 점수를 최대화하므로 이 값의 가장 큰 약수인 $\gcd(A_i, A_{i+1})$ 를 선택합니다.

1C/2I. 사과 피라미드 게임

- 해당 내부 사과를 포함하는 경로에서는 아래의 두 사과 중 하나를 선택하여 내려갈 수 있습니다.
- 따라서 두 사과 중 더 큰 값을 선택하는 것이 최적입니다.
- 그러므로 해당 위치에서 얻을 수 있는 가장 큰 기여를

$$B_i = \max(A_i, A_{i+1}) + \gcd(A_i, A_{i+1})$$

로 정의합시다.

- 이 내부 사과를 제외한 나머지 $N - 2$ 개의 내부 사과가 모두 1 이라면 해당 경로의 점수는

$$B_i + N - 2$$

입니다.

1C/2I. 사과 피라미드 게임

- 한편 모든 사과의 맛은 양의 정수이므로, 내부 사과 하나에서는 항상 적어도 1의 점수를 얻습니다.
- 첫 번째 층에서 가장 큰 최댓값을

$$M = \max_i A_i$$

라고 합시다.

- 첫 번째 층에서 M 인 사과를 포함하는 경로를 선택하면, 내부의 $N - 1$ 개 사과에서 적어도 1씩 얻을 수 있습니다.
- 따라서 어떤 경우에도

$$M + N - 1$$

이상의 점수를 얻을 수 있습니다.

1C/2I. 사과 피라미드 게임

- 루루가 선공인 경우를 생각합시다.
- 루루는 첫 번째 수에서 원하는 인접한 두 사과를 골라 그 위의 사과를 만들 수 있습니다.
- 따라서

$$B_{\max} = \max_i B_i$$

라고 하면 루루는 $B_{\max}$ 에 해당하는 위치를 선택할 수 있습니다.

- 이 위치를 포함하는 경로에서는

$$B_{\max} + N - 2$$

이상의 점수를 확보할 수 있습니다.

1C/2I. 사과 피라미드 게임

- 반대로 파노는 차단 전략을 사용하여 루루가 만든 큰 내부 사과가 위쪽으로 계속 이어지는 것을 막을 수 있습니다.
- 따라서 하나의 경로에서 1 보다 큰 추가 기여를 하는 내부 사과는 가장 아래의 하나만 고려하면 됩니다.
- 그 사과가 첫 번째 내부층의 i 번째 위치라면 그 기여는 B_i 이하입니다.
- 따라서 추가 기여는 $B_{\max}$ 를 넘을 수 없습니다.

1C/2I. 사과 피라미드 게임

- 따라서 루루 선공에서는

$$\max(M + N - 1, B_{\max} + N - 2)$$

가 정확한 답입니다.

- 즉 모든 B_i 를 확인하여 최댓값만 구하면 됩니다.

1C/2I. 사과 피라미드 게임

- 이제 파노가 선공인 경우를 생각합시다.
- B_i 들을 내림차순으로

$$B_1 \geq B_2 \geq \cdots \geq B_{N-1}$$

라고 합시다.

- 파노는 첫 번째 수에서 B_1 에 해당하는 위치를 선택하여 그 사과를 1로 만들 수 있습니다.
- 따라서 루루는 B_1 에 해당하는 위치를 사용할 수 없습니다.

1C/2I. 사과 피라미드 게임

- 루루가 사용할 수 있는 가장 좋은 위치는 B_2 에 해당하는 위치입니다.
- 루루는 그 위치에서

$$\gcd(A_i, A_{i+1})$$

를 선택하여 $B_2 + N - 2$ 의 점수를 확보할 수 있습니다.

- 파노는 이후에도 차단 전략을 사용하여 루루의 추가적인 큰 사과가 위쪽으로 이어지는 것을 막을 수 있습니다.
- 따라서 루루가 얻을 수 있는 추가 기여는 B_2 이하입니다.

1C/2I. 사과 피라미드 게임

- 따라서 파노 선공의 답은

$$\max(M + N - 1, B_2 + N - 2)$$

입니다.

- 결국 B_i 중 가장 큰 두 값만 알면 충분합니다.

1C/2I. 사과 피라미드 게임

- 각 인접한 두 사과에 대해

$$B_i = \max(A_i, A_{i+1}) + \gcd(A_i, A_{i+1})$$

를 계산합니다.

- 동시에

$$M = \max_i A_i$$

를 구합니다.

- 루루가 선공이면 B_i 의 최댓값을, 파노가 선공이면 B_i 의 두 번째 최댓값을 사용합니다.

1C/2I. 사과 피라미드 게임

– 따라서

$$\text{Lulu 선평} = \max(M + N - 1, B_1 + N - 2)$$

$$\text{Kyia 선평} = \max(M + N - 1, B_2 + N - 2)$$

입니다.

– B_i 의 최댓값과 두 번째 최댓값만 필요하므로 정렬할 필요가 없습니다.

1C/2I. 사과 피라미드 게임

- 모든 i 에 대해 $\gcd(A_i, A_{i+1})$ 를 한 번씩 계산합니다.
- 최대공약수 계산에 $O(\log(\max A_i))$ 가 걸리므로, 전체 시간 복잡도는

$$O(N \log(\max A_i))$$

입니다.

- 모든 테스트 케이스에 대한 N 의 합이 200 000 이하이므로 충분히 빠릅니다.
- 배열 A 를 저장하는 데 $O(N)$ 의 공간을 사용하며, 추가적인 변수는 $O(1)$ 개만 필요합니다.

1J. 역시청사과보다는황금사과지

dp, prefix_sum, math

출제진 의도 - **Finale**

- **Div. 1** — 처음 풀 참가자: **middle_man**, 93분 35초
- **Div. 1** — 처음 풀 팀: **trickcal**, 80분 47초
- 출제자: saturnalorbit

1J. 역시청사과보다는황금사과지

- **서브태스크 1 (10점):** $A = 1$
- 키트 하나당 황금사과를 정확히 하나 얻으므로, 총 N 개의 키트가 반드시 필요합니다.
- 배송비는 주문할 때마다 발생하므로 처음에 N 개를 한 번에 주문하는 것이 최적입니다.
- **서브태스크 2 (15점):** $B = 0$
- 배송비가 없다면 여러 개의 키트를 미리 주문할 이유가 없습니다.
- 한 번에 k 개를 주문하는 대신 1개씩 k 번 주문해도, 모두 사용한다면 비용은 같습니다.
- 또한 중간에 목표를 달성하면 이후의 키트는 주문하지 않아도 되므로, 항상 하나씩 주문하는 것이 최적입니다.
- 따라서 항상 키트를 하나씩 주문하는 것이 최적입니다.

1J. 역시청사과보다는황금사과지

- 이제 일반적인 경우를 생각합시다.
- $dp[x]$ 를 앞으로 황금사과를 x 개 더 모으는 데 필요한 최소 기대 비용이라 합시다. $dp[0] = 0$ 입니다.
- $f_k(x)$ 를 지금부터 키트 k 개를 사용한 뒤 남은 필요량에 대한 dp 의 기댓값이라 합시다.
- $f_0(x) = dp[x]$ 이고, $f_k(x) = \frac{1}{A} \sum_{d=1}^A f_{k-1}(\max(0, x-d))$ 입니다.
- 지금 키트를 k 개 주문한다면 기대 비용은 $B + kC + f_k(x)$ 입니다.
- 따라서 $dp[x] = \min_{1 \leq k \leq x} (B + kC + f_k(x))$ 입니다.

1J. 역시청사과보다는황금사과지

- **서브태스크 3 (25점):** $N \leq 300$
- $x = 1, 2, \dots, N$ 순서로 $dp[x]$ 를 계산합니다.
- $f_k(x)$ 는 $f_{k-1}(x-1), f_{k-1}(x-2), \dots$ 중 최대 A 개 값의 평균입니다.
- 따라서 가능한 결과를 직접 순회하여 모든 $f_k(x)$ 를 계산할 수 있습니다.
- x 마다 $1 \leq k \leq x$ 를 모두 확인하면 $\mathcal{O}(N^2 \min(N, A))$ 에 해결할 수 있으며, 이는 이 서브태스크에서 충분합니다.

1J. 역시청사과보다는황금사과지

– 서브태스크 4 (50점): 추가 제한이 없음

- $f_k(x)$ 의 전이에 필요한 값들은 f_{k-1} 의 연속한 구간에 들어 있습니다.
- 각 k 에 대해 $f_k(0), f_k(1), \dots$ 의 누적합을 관리합시다.
- 그러면 $f_k(x)$ 는 이전 단계 누적합의 구간합 하나로 $\mathcal{O}(1)$ 에 계산할 수 있습니다.
- x 가 증가하는 순서로 계산하면 필요한 f_{k-1} 의 값은 모두 이미 계산되어 있습니다.
- 따라서 각 (x, k) 를 한 번씩 계산하여 전체 시간 복잡도를 $\mathcal{O}(N^2)$ 으로 줄일 수 있습니다.

1J. 역시청사과보다는황금사과지

- 구현에서는 $dp[0] = f_k(0) = 0$ 으로 두면 $\max(0, x - d)$ 를 자연스럽게 처리할 수 있습니다.
- $k > x$ 이면 키트를 사용하기도 전에 목표량보다 많은 키트를 구매하게 되므로 확인할 필요가 없습니다.
- 각 x 에서 모든 $1 \leq k \leq x$ 에 대해 $B + kC + f_k(x)$ 의 최솟값을 구합니다.
- 전체 시간 복잡도는 $\mathcal{O}(N^2)$ 입니다.

11. 과수원을 운영하는 나

graphs, dfs, constructive

출제진 의도 – **Finale**

- **Div. 1** — 처음 푼 참가자: **middle_man**, 71분 2초
- **Div. 1** — 처음 푼 팀: **notflake**, 40분 52초
- 출제자: saturnalorbit

11. 과수원을 운영하는 나

- 모든 간선에는 양의 유량이 흐르고, 모든 정점에서는 유량 보존이 성립해야 합니다.
- 단절선 e 가 있다고 합시다. e 를 제거했을 때 한쪽 컴포넌트의 모든 정점에 대한 유량 보존식을 더해 봅시다.
- 컴포넌트와 바깥을 잇는 간선은 e 하나뿐이므로, e 를 지나는 순유량은 0이어야 합니다.
- 하지만 e 에는 양의 유량이 흘러야 하므로 모순입니다.
- 따라서 **단절선이 하나라도 존재하면 불가능합니다.**

11. 과수원을 운영하는 나

- **서브태스크 1 (6점):** $M = N - 1$
- 그래프는 트리이므로 모든 간선이 단절선입니다. 따라서 답이 존재하지 않습니다.
- **서브태스크 2 (7점):** $M = N$
- 연결 그래프에는 사이클이 정확히 하나 존재합니다.
- 사이클 밖의 간선은 단절선이므로, 가능한 경우 그래프 전체가 하나의 사이클이어야 합니다.
- 사이클을 한 방향으로 향하게 하고 모든 간선의 유량을 1로 둡니다.
- 성장 순서도 사이클 방향을 따라 정하면 모든 조건을 만족합니다.

11. 과수원을 운영하는 나

– 서브태스크 3 (9점): 완전 그래프

- 성장 순서를 $1, 2, \dots, N$ 으로 정합시다.
- $(i, i+1)$ 은 $i \rightarrow i+1$ 로, 나머지 $i < j$ 인 간선은 $j \rightarrow i$ 로 방향을 정합니다.
- 그러면 $v > 1$ 에서 자신보다 먼저 성장한 정점으로부터 들어오는 간선은 $(v-1) \rightarrow v$ 하나뿐입니다.
- 각 비연속 간선 $j \rightarrow i$ 는 $i \rightarrow i+1 \rightarrow \dots \rightarrow j$ 와 함께 하나의 유향 사이클을 이룹니다.
- 모든 이러한 사이클의 간선에 유량 1을 더하면 됩니다.

11. 과수원을 운영하는 나

- 하나의 유향 사이클에 같은 양의 유량을 추가하면, 사이클 위 모든 정점에서 유입량과 유출량이 똑같이 증가합니다.
- 따라서 여러 사이클의 유량을 더해도 항상 유량 보존이 성립합니다.
- 완전 그래프에서는 긴 경로 $1 \rightarrow 2 \rightarrow \dots \rightarrow N$ 을 뼈대로 사용했습니다.
- 일반 그래프에서는 이 경로 대신 **DFS 트리**를 사용하면 같은 구축을 할 수 있습니다.

11. 과수원을 운영하는 나

- 임의의 정점에서 DFS를 수행하고, 성장 순서를 DFS 방문 순서로 정합니다.
- 트리 간선은 부모에서 자식으로 향하게 합니다.
- 무방향 그래프의 DFS에서 모든 비트리 간선은 조상과 자손을 연결하므로, 비트리 간선은 자손에서 조상으로 향하게 합니다.
- 따라서 루트를 제외한 모든 정점에서, 자신보다 먼저 성장한 정점으로부터 들어오는 간선은 부모 간선 하나뿐입니다.
- 이제 방향에 관한 조건은 모두 만족했습니다.

11. 과수원을 운영하는 나

- 비트리 간선 $v \rightarrow u$ 에서 u 는 v 의 조상입니다.
- 이 간선과 u 에서 v 까지의 트리 경로는 하나의 유향 사이클을 이룹니다.
- 각 비트리 간선이 만드는 사이클에 유량 1을 추가합니다.
- 비트리 간선은 자신의 사이클에서 유량 1을 얻습니다.
- 단절선이 없다면 모든 트리 간선도 적어도 하나의 사이클에 포함되므로 유량이 양수가 됩니다.
- 비트리 간선의 개수는 $M - N + 1$ 이므로, 어떤 간선의 유량도 $M - N + 1$ 을 넘지 않습니다.

11. 과수원을 운영하는 나

- **서브태스크 4 (23점):** $N, M \leq 2000$
- 각 비트리 간선 $v \rightarrow u$ 마다 부모 포인터를 따라 v 에서 u 까지 직접 올라가며 모든 트리 간선의 유량을 1씩 증가시킬 수 있습니다.
- 모든 비트리 간선에 대해 이를 반복한 뒤, 유량이 0인 트리 간선이 있다면 단절선이므로 답이 없습니다.
- 한 경로의 길이가 최대 N 이고 비트리 간선이 $O(M)$ 개이므로 $\mathcal{O}(NM)$ 에 해결할 수 있습니다.
- 이제 이 경로 덧셈을 한 번에 처리하면 됩니다.

11. 과수원을 운영하는 나

- 각 비트리 간선 $v \rightarrow u$ 에 대해 v 에 $+1$, u 에 -1 을 기록합니다.
- 이후 DFS 트리에서 각 정점의 subtree sum을 계산합니다.
- 정점 v 의 subtree sum은 부모와 v 를 잇는 트리 간선을 사용하는 유향 사이클의 개수와 정확히 같습니다.
- 따라서 이 subtree sum을 해당 트리 간선의 유량으로 사용하면 됩니다.
- 비트리 간선의 유량은 모두 1입니다.
- subtree sum이 0인 트리 간선이 존재하면 그 간선은 어떤 사이클에도 포함되지 않으므로 단절선이고, 답이 존재하지 않습니다.

11. 과수원을 운영하는 나

- 서브태스크 5 (55점): 추가 제한이 없음

- DFS 방문 순서를 성장 순서로 사용하고, 트리 간선은 부모에서 자식으로, 비트리 간선은 자손에서 조상으로 향하게 합니다.
- 비트리 간선마다 자손에 $+1$, 조상에 -1 을 더한 뒤 subtree sum을 계산하여 모든 트리 간선의 유량을 얻습니다.
- 모든 트리 간선의 유량이 양수이면 구축이 완성되고, 하나라도 0이면 답이 없습니다.
- 모든 과정은 DFS 몇 번으로 수행할 수 있으므로 전체 시간 복잡도는 $\mathcal{O}(N + M)$ 입니다.

2J. 사과나무 심기

constructive

출제진 의도 – **Finale**

- **Div. 2** — 처음 푼 참가자: -, -
- 출제자: saturnalorbit

2J. 사과나무 심기

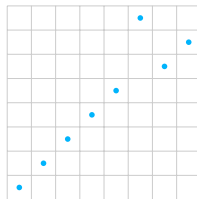
- i 번째 행의 p_i 번째 열에 나무를 심었다고 하면, p 는 1 부터 N 까지의 순열입니다.
- 첫 번째 행과 마지막 행의 나무는 서로 다른 열에 있으므로 거리가 적어도 $(N - 1) + 1 = N$ 입니다. 따라서 $D \geq N$ 이어야 합니다.
- $N \geq 3$ 에서 거리 $2N - 2$ 가 나타나려면 서로 마주 보는 두 꼭짓점에 나무가 있어야 합니다.
- 이 경우 거리 $2N - 3$ 인 쌍을 만들 수 없으므로, $D = 2N - 2$ 는 불가능합니다.
- 따라서 일반적으로 가능한 범위는 $N \leq D \leq 2N - 3$ 입니다. 작은 N 에는 몇 가지 예외가 있습니다.

2J. 사과나무 심기

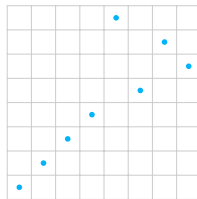
- **서브태스크 1 (2점):** $D = 2N - 2$ — $N = 2$ 일 때만 가능합니다.
- **서브태스크 2 (8점):** $N \leq 8$ — 모든 순열을 완전 탐색할 수 있습니다.
- **서브태스크 3 (10점):** $D = 2N - 3$ — 순열 $1, 2, \dots, N - 3, N, N - 2, N - 1$ 을 사용합니다.
- **서브태스크 4 (20점):** $D = N$ — 두 개의 긴 역대각선을 접은 형태의 구축을 사용합니다.
- 실제 구현에서는 $N \leq 6$ 을 완전 탐색하여 작은 예외를 모두 처리합니다.

2J. 사과나무 심기

- 최대 거리가 홀수라면 $1, 2, \dots, m-3, m, m-2, m-1$ 을 사용합니다.
- 이 배치의 거리 집합은 정확히 $\{2, 3, \dots, 2m-3\}$ 입니다.
이를 **3-gadget**이라 하겠습니다.
- 최대 거리가 짝수라면
 $1, 2, \dots, m-4, m, m-3, m-1, m-2$ 를 사용합니다.
- 이 배치의 거리 집합은 정확히 $\{2, 3, \dots, 2m-4\}$ 입니다.
이를 **4-gadget**이라 하겠습니다.



3-gadget



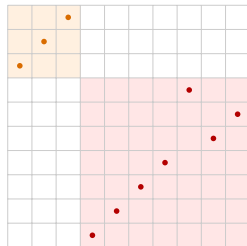
4-gadget

2J. 사과나무 심기

- 두 gadget 모두 앞부분은 (i, i) 꼴의 대각선입니다.
- 3-gadget에서는 마지막 세 점과 대각선 위의 점 사이에서 홀수 거리와 짝수 거리가 번갈아 나타나며, 마지막 세 점 사이에서 거리 2, 3을 얻습니다.
- 같은 방법으로 4-gadget에서는 마지막 네 점을 이용하여 2부터 $2m - 4$ 까지의 모든 거리를 얻습니다.
- 따라서 $q = 3$ 또는 4를 각각 D 의 홀짝에 맞게 고르고 $m = (D + q)/2$ 로 두면, 크기 m 의 gadget 내부에서 이미 $2, 3, \dots, D$ 가 모두 나타납니다.

2J. 사과나무 심기

- 이제 gadget 앞에 $m + 1, m + 2, \dots, N$ 을 붙입니다.
- 즉, 남은 $N - m$ 개의 점은 하나의 대각선 위에 놓입니다.
- 이 부분을 **대각선 꼬리**라 하겠습니다.
- gadget 내부에서 필요한 거리는 이미 모두 만들어졌으므로, 새로 생기는 거리가 D 를 넘지 않는지만 확인하면 됩니다.



$N = 10, D = 11, m = 7$

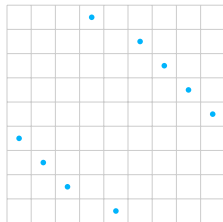
주황색: 꼬리, 빨간색: gadget

2J. 사과나무 심기

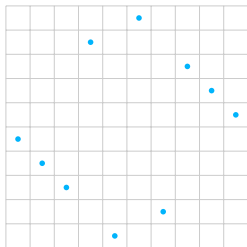
- 꼬리의 길이를 $t = N - m$ 이라 하고, gadget의 순열을 $g_1, \dots, g_m$ 이라 합시다.
- 꼬리의 i 번째 점은 $(i, m + i)$ 이고, gadget의 s 번째 점은 $(t + s, g_s)$ 입니다.
- 두 점 사이의 거리는 $(t + s - i) + (m + i - g_s) = N + s - g_s$ 입니다.
- 중요한 점은 **꼬리에서 어느 점을 선택했는지가 사라진다는** 것입니다.
- 3-gadget과 4-gadget에서 각각 $\max(s - g_s) = 1, 2$ 이므로, $D \geq N + q - 2$ 이면 꼬리를 붙여도 D 보다 큰 거리가 생기지 않습니다.

2J. 사과나무 심기

- $D \geq N + q - 2$ 가 성립하지 않는 경우에는 gadget을 사용할 수 없습니다.
- 가능한 경우 중 남는 것은 $D = N$ 과, 홀수 N 에서의 $D = N + 1$ 뿐입니다.
- $N = 2h + 1$ 이고 $D = N$ 이면 $h, h - 1, \dots, 2, N, 1, N - 1, \dots, h + 1$ 을 사용합니다.
- $N = 2h$ 이고 $D = N$ 이면 $h, h - 1, \dots, 3, N - 1, 1, N, 2, N - 2, \dots, h + 1$ 을 사용합니다.
- 홀수 N 에서 $D = N + 1$ 이면 첫 번째 구축의 첫 두 원소만 교환합니다.



$N = 9, D = 9$



$N = 10, D = 10$

2J. 사과나무 심기

- $N \leq 6$ 이면 모든 순열을 완전 탐색하여 답을 미리 구합니다.
- $N \geq 7$ 에서 $D < N$ 또는 $D > 2N - 3$ 이면 답이 없습니다.
- D 가 홀수이면 $q = 3$, 짝수이면 $q = 4$ 로 둡니다.
- $D \geq N + q - 2$ 이면 $m = (D + q)/2$ 인 q -gadget 앞에 대각선 꼬리를 붙입니다.
- 그렇지 않으면 앞에서 설명한 구축을 사용합니다.
- 시간 복잡도는 테스트 케이스마다 $\mathcal{O}(N)$ 입니다.

1B. 사과의 감시자

greedy, cartesian_tree, data_structures

출제진 의도 – **Finale**

- **Div. 1** — 처음 푼 참가자: **physics0523**, 30분 22초
- **Div. 1** — 처음 푼 팀: **reboot_ssal_game**, 32분 33초
- 출제자: carrot

1B. 사과와 감시자

- **서브태스크 1 (7점):** $N \leq 20$
- 수확할 나무의 집합을 모두 완전 탐색할 수 있습니다.
- 정확히 K 그루를 고른 경우, 각 나무가 감시하는 범위에서 선택된 나무의 수가 c_i 이하인지 확인합니다.
- 조건을 만족하는 경우 중 가치 합을 구하면 됩니다.
- 전체 시간 복잡도는 $\mathcal{O}(N2^N)$ 입니다.
- **서브태스크 2 (1점):** 모든 $c_i = N$
- 모든 감시 조건이 자동으로 만족됩니다.
- 따라서 가치가 큰 나무 K 그루를 고르면 됩니다.

1B. 사과와 감시자

- i 번 나무보다 높은 나무 중 왼쪽에서 가장 가까운 위치를 L_i , 오른쪽에서 가장 가까운 위치를 R_i 라고 하겠습니다.
- 그런 나무가 없다면 각각 $L_i = 0, R_i = N + 1$ 로 둡니다.
- i 번 나무가 감시하는 범위는 $[L_i + 1, R_i - 1]$ 입니다.
- L_i 와 R_i 는 단조 스택을 이용해 전체 $\mathcal{O}(N)$ 에 구할 수 있습니다.

1B. 사과의 감시자

- **서브태스크 3 (10점):** $K = 1$
- 한 그루만 고르므로 $c_i \geq 1$ 인 제한은 항상 만족합니다.
- 따라서 $c_i = 0$ 인 나무의 감시 범위에 속하는 위치만 고를 수 없습니다.
- 이 구간들을 차분 배열에 표시하면 각 나무가 선택 가능한지 선형 시간에 알 수 있습니다.
- 선택 가능한 나무 중 가치가 가장 큰 것을 고릅니다.
- 그런 나무가 없다면 답은 -1 입니다.
- 전체 시간 복잡도는 $\mathcal{O}(N)$ 입니다.

1B. 사과의 감시자

- 서브태스크 4 (9점): $h_1, h_2, \dots, h_N$ 이 단조
- 높이가 증가한다면 i 번 나무의 감시 범위는 $[1, i]$ 입니다.
- 왼쪽부터 나무를 하나씩 후보에 추가합니다.
- 후보가 c_i 개보다 많아질 때마다 가치가 가장 작은 나무를 제거합니다.
- 이미 만족한 제한에서는 나무를 더 제거해도 문제가 생기지 않습니다.
- 높이가 감소하는 경우에는 오른쪽부터 같은 방법으로 처리합니다.
- 최소 힙을 사용하면 전체 시간 복잡도는 $\mathcal{O}(N \log N)$ 입니다.

1B. 사과의 감시자

- 일반적인 높이 배열에서는 최대 카르테시안 트리를 생각할 수 있습니다.
- 이 트리의 중위 순회는 $1, 2, \dots, N$ 이고, 부모의 높이는 항상 자식보다 큼니다.
- i 번 정점의 서브트리는 정확히 $L_i + 1$ 번부터 $R_i - 1$ 번까지의 정점으로 이루어집니다.
- 즉, i 번 나무의 감시 범위는 카르테시안 트리에서 i 의 서브트리와 같습니다.
- 따라서 i 의 서브트리에서는 최대 c_i 개의 정점만 선택할 수 있습니다.

1B. 사과의 감시자

- **서브태스크 5 (25점):** $N \leq 3000$
- $dp_u[k]$ 를 u 의 서브트리에서 정확히 k 개의 정점을 골랐을 때의 최대 가치 합이라 하겠습니다.
- 두 자식의 DP를 knapsack처럼 합치고, u 를 고르는 경우와 고르지 않는 경우를 확인합니다.
- $k > c_u$ 인 상태는 사용할 수 없으므로 버립니다.
- 각 정점에서 두 자식의 DP를 직접 합치면 전체 $\mathcal{O}(N^2)$ 에 해결할 수 있습니다.
- 루트에서 정확히 K 개를 고르는 상태가 없다면 답은 -1 입니다.

1B. 사과의 감시자

- **서브태스크 6 (48점): 추가 제한이 없음**
- 서브태스크 4에서 사용한 그리드를 카르테시안 트리에서도 그대로 사용할 수 있습니다.
- 두 자식의 제한을 모두 처리한 뒤 남아 있는 후보들과 현재 정점을 합칩니다.
- 후보가 c_u 개보다 많다면 가치가 가장 작은 후보를 제거합니다.
- 필요한 만큼 반복하여 후보를 c_u 개 이하로 유지합니다.

1B. 사과의 감시자

- 이 그리디가 올바른 이유를 보겠습니다.
- 정점 u 를 처리한 뒤 후보가 m 개 남았다고 합시다.
- $0 \leq k \leq m$ 에 대해, 후보 중 가치가 큰 k 개를 고르면 u 의 서브트리에서 정확히 k 개를 고르는 최적해가 됩니다.
- 리프에서는 바로 성립합니다.
- 두 자식에서 성립한다고 하면, 두 후보 집합과 a_u 를 합친 뒤에도 가치가 큰 k 개를 고르는 것이 최적입니다.
- 여기서 새로 생기는 조건은 선택 개수가 c_u 이하여야 한다는 것뿐이므로, 후보가 많을 때 가장 작은 값을 제거해도 c_u 개 이하를 고르는 최적값은 바뀌지 않습니다.

1B. 사과의 감시자

- 각 정점에서는 두 자식의 후보를 합친 뒤 최솟값을 여러 번 제거해야 합니다.
- 따라서 두 힙을 빠르게 합칠 수 있는 자료구조를 사용하면 됩니다.
- leftist heap이나 skew heap을 사용하면 두 최소 힙을 $\mathcal{O}(\log N)$ 에 합칠 수 있습니다.
- 각 나무는 힙에 한 번 들어가고, 제거되는 경우에도 한 번만 제거됩니다.
- 따라서 전체 시간 복잡도는 $\mathcal{O}(N \log N)$, 메모리 복잡도는 $\mathcal{O}(N)$ 입니다.

1B. 사과의 감시자

- 모든 제한을 처리한 뒤 루트의 힙에 M 개의 후보가 남았다고 합시다.
- M 은 모든 감시 조건을 만족하면서 선택할 수 있는 나무 수의 최댓값입니다.
- $M < K$ 이면 정확히 K 그루를 고를 수 없으므로 -1 입니다.
- $M \geq K$ 이면 가치가 작은 후보를 $M - K$ 개 더 제거합니다.
- 남은 K 그루의 가치 합이 답입니다.
- 가치의 합은 64비트 정수로 저장해야 합니다.

1B. 사과의 감시자

– 별해: 단조 스택 + 세그먼트 트리

- 카르테시안 트리를 직접 만들지 않고도 같은 그리드를 구현할 수 있습니다.
- 단조 스택으로 각 나무의 감시 범위를 구한 뒤, 모든 나무를 처음에는 후보로 남겨 둡니다.
- 나무를 높이가 낮은 순서대로 처리합니다.
- i 번 나무를 처리할 때 그 감시 범위 안의 더 작은 범위들은 이미 모두 처리되어 있습니다.
- 따라서 범위 안에 후보가 c_i 개보다 많이 남아 있다면, 그 범위에서 가치가 가장 작은 후보를 제거하면 됩니다.

1B. 사과의 감시자

- 세그먼트 트리에는 각 구간의 후보 수와 최소 가치, 그 위치를 저장합니다.
- 감시 범위의 후보 수를 확인하고, 제한을 넘는 동안 구간 최소값을 찾아 삭제합니다.
- 한 나무는 최대 한 번만 삭제되므로 전체 삭제 횟수는 $O(N)$ 입니다.
- 각 질의와 삭제가 $\mathcal{O}(\log N)$ 에 처리되므로 이 풀이도 전체 시간 복잡도는 $\mathcal{O}(N \log N)$ 입니다.

1H. 사과나무 삼합점

ad_hoc, number_theory, gcd

출제진 의도 - **Finale**

- **Div. 1** — 처음 푼 참가자: -, -
- **Div. 1** — 처음 푼 팀: **trickcal**, 125분 31초
- 출제자: carrot

1H. 사과나무 삼합점

- 처음 사과나무의 위치를 $P_1, P_2, \dots, P_K$ 라 합시다.
- 모든 위치에서 P_1 을 빼도 삼합점 연산의 결과에는 영향을 주지 않습니다. 따라서 $P_1 = (0, 0)$ 이라 생각하고 $v_i = P_i - P_1$ 만 보겠습니다.
- $0, u, v$ 를 만들 수 있다면 삼합점으로 $u + v = u - 0 + v$ 와 $-u = 0 - u + 0$ 을 만들 수 있습니다.
- 따라서 최종적으로 만들 수 있는 위치는 $v_2, \dots, v_K$ 를 더하고 빼서 만들 수 있는 위치와 정확히 같습니다.
- 좌표는 각각 H, W 를 주기로 가지므로 $(H, 0)$ 과 $(0, W)$ 역시 0 과 같은 이동으로 생각할 수 있습니다.

1H. 사과나무 삼합점

- **서브태스크 1 (2점):** $K = 1$
- 새로운 위치를 만들 수 없으므로 답은 1입니다.
- **서브태스크 2 (13점):** $\min(H, W) = 1$
- $H = 1$ 이라 합시다. $d_i = y_i - y_1$ 이라 하면 만들 수 있는 y 좌표의 차이는 $d_2, \dots, d_K$ 의 정수 선형결합입니다.
- $g = \gcd(W, d_2, \dots, d_K)$ 라 하면 만들 수 있는 좌표는 g 의 배수들이므로 답은 W/g 입니다. $W = 1$ 인 경우도 같습니다.

1H. 사과나무 삼합점

- **서브태스크 3 (16점):** $K \leq 2$
- $K = 2$ 라면 하나의 차이 벡터 $v = (x, y)$ 만 반복해서 더하면 됩니다.
- x 좌표가 다시 0이 되는 최소 횟수는 $H / \gcd(H, x)$ 이고, y 좌표는 $W / \gcd(W, y)$ 입니다.
- 두 좌표가 동시에 처음 위치로 돌아와야 하므로 답은 $\text{lcm}(H / \gcd(H, x), W / \gcd(W, y))$ 입니다.

1H. 사과나무 삼합점

- **서브태스크 4 (22점):** $K \leq 4000$
- $(H, 0), (0, W), v_2, \dots, v_K$ 가 만드는 정수 격자를 생각합시다.
- 두 정수 벡터 a, b 가 만드는 평행사변형의 넓이는 $|\det(a, b)|$ 입니다.
- 여러 정수 벡터가 만드는 격자의 기본 넓이는 모든 두 벡터의 행렬식 절댓값의 최대공약수와 같습니다.
- $v_i = (x_i, y_i)$ 라 하면 필요한 값은 HW, Hy_i, Wx_i , 그리고 모든 $i < j$ 에 대한 $|x_i y_j - y_i x_j|$ 입니다.

1H. 사과나무 삼합점

- 이 값들의 최대공약수를 D 라 합시다.
- 기본 직사각형 $(H, 0), (0, W)$ 의 넓이는 HW 이고, 우리가 만든 격자의 기본 넓이는 D 입니다.
- 따라서 하나의 기본 직사각형 안에는 우리가 만든 격자의 점이 정확히 HW/D 개 존재하고, 이것이 답입니다.
- 모든 v_i, v_j 의 쌍을 직접 확인하면 전체 시간 복잡도는 $\mathcal{O}(K^2 \log(HW))$ 입니다.
- 이제 K 가 큰 경우에는 모든 쌍을 확인하지 않고 같은 값을 구해야 합니다.

1H. 사과나무 삼합점

– 서브태스크 5 (47점): 추가 제한이 없음

- 먼저 x 좌표만 생각합시다. $v_i = (x_i, y_i)$ 라 두고 $g = \gcd(H, x_2, \dots, x_K)$ 라 합시다.
- 만들 수 있는 모든 벡터의 x 좌표는 g 의 배수이고, 반대로 베주 항등식에 의해 x 좌표가 g 인 벡터도 만들 수 있습니다.
- 단순히 g 만 구하는 것으로는 부족합니다. 실제로 만들 수 있는 벡터 (g, t) 하나를 함께 구해 두겠습니다.
- 처음에는 $(g, t) = (H, 0)$ 입니다.

1H. 사과나무 삼합점

- 현재 (g, t) 를 만들 수 있고 새 벡터가 (x, y) 라고 합시다.
- 확장 유클리드 알고리즘으로 $ag + bx = g'$ 인 a, b 를 구하면 $g' = \gcd(g, x)$ 입니다.
- (g, t) 와 (x, y) 를 같은 계수로 더하면 $(g', at + by)$ 도 만들 수 있습니다.
- 따라서 $g \leftarrow g', t \leftarrow at + by$ 로 갱신하면 됩니다. t 는 매번 W 로 나눈 나머지만 저장해도 됩니다.
- 모든 벡터를 처리한 뒤에는 (g, t) 가 실제로 만들 수 있는 벡터이고, 모든 x_i 는 g 의 배수입니다.

1H. 사과나무 삼합점

- 이제 x 좌표가 0인 이동만 모아 봅시다.
- 각 $v_i = (x_i, y_i)$ 에서 (x_i/g) 번의 (g, t) 를 빼면 $(0, y_i - (x_i/g)t)$ 를 만들 수 있습니다.
- 또한 (g, t) 를 H/g 번 더한 뒤 $(H, 0)$ 을 빼면 $(0, (H/g)t)$ 를 만들 수 있습니다.
- 원래부터 $(0, W)$ 도 같은 위치로 돌아오는 이동입니다.
- 따라서 x 좌표를 바꾸지 않고 만들 수 있는 모든 y 이동의 최대공약수를 q 라 하면 $q = \gcd(W, (H/g)t, y_i - (x_i/g)t)$ 입니다.

1H. 사과나무 삼합점

- 반대로 원래의 모든 생성 벡터는 (g, t) 와 방금 만든 $x = 0$ 인 벡터들의 정수 선형결합으로 다시 만들 수 있습니다.
- 따라서 가능한 x 좌표는 정확히 H/g 개이고, 하나의 x 좌표를 고정했을 때 가능한 y 좌표는 정확히 W/q 개입니다.
- 답은 $(H/g)(W/q)$ 입니다.
- 첫 번째 순회에서는 (g, t) 를 구하고, 두 번째 순회에서는 q 를 구하면 됩니다.
- 전체 시간 복잡도는 $\mathcal{O}(K \log \max(H, W))$, 메모리 복잡도는 $\mathcal{O}(K)$ 입니다.

1G. RAM (Rotating Auto-Mobiles)

geometry, binary_search

출제진 의도 – **Encore**

- **Div. 1** — 처음 푼 참가자: **mjkim112358**, 49분 9초
- **Div. 1** — 처음 푼 팀: **reboot_ssal_game**, 180분 29초
- 출제자: saturnalorbit

1G. RAM (Rotating Auto-Mobiles)

- **서브태스크 1 (2점): 모든 나무가 원점에 있음**
- 반지름이 0인 원형 경로를 선택하면 운반기와 모든 로봇의 이동 거리가 0입니다.
- 따라서 답은 0입니다.
- 이후에는 최대 이동 거리가 $2\pi r$ 이하가 가능한지를 생각해 봅시다.
- 실제 원의 반지름을 ρ 라 하면 운반기의 이동 거리 때문에 $\rho \leq r$ 이어야 합니다.
- i 번 로봇은 a_i 번 왕복하므로, 나무와 원 사이의 거리는 $\pi r / a_i$ 이하여야 합니다.

1G. RAM (Rotating Auto-Mobiles)

- $a_i \leq 3 < \pi$ 이므로 $\pi r / a_i \geq r$ 입니다.
- 따라서 나무가 원 내부에 있다면 원까지의 거리가 $\rho \leq r$ 이하이므로 자동으로 조건을 만족합니다.
- 반지름이 $\rho < r$ 인 가능한 원이 있다면, 중심의 방향을 유지하면서 중심과 반지름을 함께 늘려 반지름을 r 로 만들 수 있습니다.
- 확대된 원판은 기존 원판을 포함하며, 원 밖의 나무와 원 사이의 거리도 증가하지 않습니다.
- 따라서 가능한 원이 존재한다면 반지름이 정확히 r 인 가능한 원도 존재합니다.
- 이제 중심은 원점에서 거리 r 인 점이므로, 중심의 방향만 정하면 됩니다.

1G. RAM (Rotating Auto-Mobiles)

- **서브태스크 2 (8점): 모든 나무가 하나의 반직선 위에 있음**
- 나무의 방향으로 원의 중심을 잡는 것이 항상 최적입니다.
- 원점에서 거리가 d_i 인 나무 하나만 생각하면, 최적의 r 은 $a_i d_i / (\pi + 2a_i)$ 입니다.
- 모든 나무가 같은 방향에 있으므로 이 선택을 동시에 사용할 수 있고, 필요한 r 은 이 값들의 최댓값입니다.
- **서브태스크 3 (11점): 하나의 직선 위, $a_i = 1$**
- 양의 방향과 음의 방향에서 가장 먼 나무만 남겨도 됩니다. 그 거리를 각각 d_+ , d_- 라 합시다.
- 두 방향에서 요구하는 중심 각도의 조건을 함께 정리하면 $r^2 \geq d_+ d_- / (\pi(\pi + 2))$ 을 얻습니다.
- 각 나무 하나만 존재할 때의 조건까지 함께 확인하면 답을 바로 계산할 수 있습니다.

1G. RAM (Rotating Auto-Mobiles)

- 이제 중심의 방향을 θ , i 번 나무의 편각을 ϕ_i , 원점과의 거리를 d_i 라 합시다. d_i 가 0인 나무들은 제약을 만들지 않으니 무시할 수 있습니다.
- 반지름이 정확히 r 이므로, i 번 나무가 만족해야 하는 조건은 $|P_i - C| \leq (1 + \pi/a_i)r$ 입니다.
- 코사인 법칙을 적용하면 $\cos(\theta - \phi_i) \geq v_i$ 인 꼴이 됩니다.
- 여기서 $v_i = (d_i^2 + r^2 - (1 + \pi/a_i)^2 r^2) / (2d_i r)$ 입니다.
- $v_i \leq -1$ 이면 모든 방향이 가능하고, $v_i > 1$ 이면 어떤 방향도 불가능합니다.
- 그 외에는 $\alpha_i = \arccos v_i$ 라 하면 허용되는 방향은 $\phi_i - \alpha_i$ 부터 $\phi_i + \alpha_i$ 까지의 원형 구간입니다.

1G. RAM (Rotating Auto-Mobiles)

- **서브태스크 4 (14점):** $x_i y_i = 0, a_i = 1$
- 나무가 존재할 수 있는 방향은 네 방향뿐입니다.
- 같은 방향의 나무 중에서는 원점에서 가장 먼 나무가 가장 강한 조건을 만들므로, 방향마다 하나의 나무만 남겨도 됩니다.
- 따라서 최대 네 개의 원형 구간만 생각하면 됩니다.
- r 에 대해 이분 탐색하면서 네 구간의 공통 부분이 존재하는지 직접 확인하면 충분합니다.

1G. RAM (Rotating Auto-Mobiles)

- **서브태스크 5 (15점):** 모든 d_i 가 같고 $a_i = 1$
- 같은 r 에서는 모든 나무가 만드는 허용 구간의 길이가 같습니다.
- 따라서 필요한 것은 모든 나무의 방향을 포함하는 가장 짧은 원형 구간입니다.
- 나무들의 편각을 정렬하고, 인접한 두 편각 사이의 가장 큰 공백을 찾습니다.
- 그 공백을 제외한 나머지 원의 길이가 모든 방향을 덮는 최소 각도 범위가 됩니다.
- 이 범위의 절반만큼 허용각 α 가 넓어지는 최소 r 을 계산하면 답을 얻을 수 있습니다.

1G. RAM (Rotating Auto-Mobiles)

- **서브태스크 6 (20점):** $N \leq 150$
- 이분 탐색 없이 직접 최적의 원 중심을 찾아 봅시다.
- $\beta_i = 1 + \pi / a_i$, $\gamma_i = \beta_i^2 - 1$ 이라 합시다.
- 중심을 C 라 하면 필요한 조건은 $|P_i - C| \leq \beta_i |C|$ 입니다.
- 식을 정리하면 C 는 중심이 $-P_i / \gamma_i$ 이고 반지름이 $d_i \beta_i / \gamma_i$ 인 원의 **바깥쪽**에 있어야 합니다.
- 따라서 각 나무는 중심 C 가 들어갈 수 없는 하나의 원판을 만듭니다.
- 문제는 모든 금지 원판의 바깥에 있으면서 원점에 가장 가까운 점을 찾는 문제로 바뀝니다.

1G. RAM (Rotating Auto-Mobiles)

- 최적의 중심은 금지 원판들의 합집합 경계 위에 있습니다.
- 따라서 후보는 어떤 한 원의 경계에서 원점에 가장 가까운 점이거나, 두 원의 경계의 교점입니다.
- 모든 원에 대해 첫 번째 종류의 후보를 만들고, 모든 원 쌍에 대해 교점을 구합니다.
- 각 후보가 모든 금지 원판의 바깥에 있는지 $\mathcal{O}(N)$ 에 확인합니다.
- 후보가 $\mathcal{O}(N^2)$ 개이므로 전체 시간 복잡도는 $\mathcal{O}(N^3)$ 입니다.
- 이 풀이에는 실수 이분 탐색이 필요하지 않습니다.

1G. RAM (Rotating Auto-Mobiles)

- **서브태스크 7 (30점): 추가 제한이 없음**
- 다시 최대 이동 거리가 $2\pi r$ 이하인지 판정하는 문제를 생각합니다.
- 각 나무는 중심 방향 θ 에 대한 하나의 원형 구간을 만듭니다.
- 모든 나무의 구간에 동시에 포함되는 방향이 존재하면 현재 r 은 가능합니다.
- 각 구간의 시작점에 $+1$, 끝점에 -1 이벤트를 만들고 각도순으로 스위핑합니다.
- 0을 가로지르는 구간은 처음부터 활성화되어 있다고 처리하면 됩니다.
- 활성 구간의 수가 필요한 구간의 수와 같아지는 순간이 존재하는지 확인합니다.

1G. RAM (Rotating Auto-Mobiles)

- 한 번의 판정에는 최대 $2N$ 개의 이벤트를 정렬하므로 $\mathcal{O}(N \log N)$ 이 필요합니다.
- 가능 여부는 r 에 대해 단조이므로 실수 이분 탐색을 적용할 수 있습니다.
- 충분한 횟수만큼 이분 탐색한 뒤 최소 r 에 대해 $2\pi r$ 을 출력합니다.
- 전체 시간 복잡도는 이분 탐색 횟수를 상수로 보면 $\mathcal{O}(N \log N)$ 입니다.

1F. 노크한 과수원은 똑똑한 과수원이다

sweeping, segment_tree, math

출제진 의도 - **Encore**

- **Div. 1** — 처음 푼 참가자: -, -
- **Div. 1** — 처음 푼 팀: -, -
- 출제자: saturnalorbit

1F. 노크한 과수원은 똑똑한 과수원이다

- **서브태스크 1 (10점): 투입 기간이 서로 겹치지 않음**
- 동시에 투입되는 수확반이 최대 하나이므로, 식은 $C_i + \lfloor T/D_i \rfloor = T$ 가 됩니다.
- $T - \lfloor T/D_i \rfloor$ 는 단조 증가하므로 각 구간에서 이분 탐색으로 답을 찾을 수 있습니다.
- **서브태스크 2 (15점): 모든 $C_i = 0$**
- 투입된 수확반 중 하나라도 $D_i > 1$ 이면 총수확량은 T 보다 작습니다.
- 따라서 모든 수확반의 $D_i = 1$ 인 구간에서 T 가 투입된 수확반 수의 배수인지만 확인하면 됩니다.

1F. 노크한 과수원은 똑똑한 과수원이다

- **서브태스크 3 (15점):** 모든 $D_i = 1$
- 현재 투입된 수확반의 수를 k , $C = \sum C_i$ 라 하고 $T = kq + r$ ($0 \leq r < k$) 로 나타냅니다.
- 총수확량은 $C + kq$ 이므로, 정확히 T 가 되려면 $r = C$ 여야 합니다.
- 따라서 $C < k$ 라면 현재 구간에서 $T \equiv C \pmod{k}$ 인 값들을 세면 되고, $C \geq k$ 라면 답이 없습니다.
- 이제부터는 $D_i > 1$ 인 수확반도 존재하는 일반적인 경우를 생각합니다.

1F. 노크한 과수원은 똑똑한 과수원이다

- L_i 와 $R_i + 1$ 에서 이벤트가 발생하므로, 투입되는 수확반의 집합이 일정한 구간별로 스위핑합니다.
- 현재 투입된 수확반의 수를 k 라 하고 $T = kq + r$ ($0 \leq r < k$)로 나타냅니다.
- 그러면 $\lfloor T/(kD_i) \rfloor = \lfloor q/D_i \rfloor$ 이므로, 같은 q 에서는 총수확량이 일정합니다.
- $D_i > 1$ 인 수확반의 수를 p , 이들의 합을 $F(q) = \sum_{D_i > 1} \lfloor q/D_i \rfloor$ 라 합니다.
- 총수확량은 $H(q) = C + (k - p)q + F(q)$ 입니다.

1F. 노크한 과수원은 똑똑한 과수원이다

- $H(q) = T = kq + r$ 이어야 하므로 $r = C - (pq - F(q))$ 입니다.
- $G(q) = pq - F(q)$ 라 정의하면, 어떤 q 에서 답이 존재할 필요충분조건은 $C - k + 1 \leq G(q) \leq C$ 입니다.
- 각 항 $q - \lfloor q/D_i \rfloor$ 은 q 에 대해 단조 증가하므로 $G(q)$ 도 단조 증가합니다.
- 따라서 가능한 q 들은 하나의 연속한 구간이며, $G(q)$ 의 lower bound를 두 번 구하면 됩니다.
- **서브태스크 4 (25점): 서로 다른 D_i 의 개수가 20 이하**에서는 $G(q)$ 를 $\mathcal{O}(20)$ 에 계산하여 평범하게 이분 탐색하면 충분합니다.

1F. 노크한 과수원은 똑똑한 과수원이다

- 일반 제한에서는 $F(q) = \sum \lfloor q/D_i \rfloor$ 의 계산도 빨라야 합니다.
- $\lfloor q/d \rfloor$ 의 값이 같은 d 들은 연속한 구간을 이루며, 이러한 구간의 개수는 $\mathcal{O}(\sqrt{q})$ 입니다.
- 각 D 의 등장 횟수를 Fenwick Tree에 저장하면 각 구간에 포함된 수확반의 수를 빠르게 구할 수 있습니다.
- 한편 현재 서로 다른 D 의 개수가 적다면, 등장하는 D 들만 직접 순회하는 편이 더 빠릅니다.
- 따라서 한 번의 $F(q)$ 계산에는 두 방법 중 작은 쪽을 사용합니다.

1F. 노크한 과수원은 똑똑한 과수원이다

- 이 두 방법을 함께 사용하는 것이 중요한 이유를 살펴봅시다.
- 현재 구간에서 의미 있는 q 는 $q \leq T_{\max}/k$ 이고, 서로 다른 D 의 개수는 k 를 넘지 않습니다.
- 따라서 한 번의 계산에는 대략 $\min(k, \sqrt{T_{\max}/k})$ 개의 항 또는 구간만 확인하면 됩니다.
- 이 값은 $k \approx T_{\max}^{1/3}$ 일 때 가장 크므로, 항상 $\mathcal{O}(T_{\max}^{1/3})$ 수준으로 제한됩니다.
- Fenwick Tree의 구간합까지 포함하면 한 번의 $F(q)$ 계산은 $\mathcal{O}(T_{\max}^{1/3} \log D_{\max})$ 에 할 수 있습니다.

1F. 노크한 과수원은 똑똑한 과수원이다

- 이제 lower bound에서 이분 탐색까지 없애 봅시다.
- $p > 0$ 일 때 $\alpha = \sum_{D_i > 1} (1 - 1/D_i)$ 라 두면 $\alpha q \leq G(q) < \alpha q + p$ 입니다.
- 또한 모든 $D_i > 1$ 에 대해 $1 - 1/D_i \geq 1/2$ 이므로 $\alpha \geq p/2$ 입니다.
- $G(q) \geq x$ 인 최소 q 를 찾고 싶다면 $q_0 = \lceil x/\alpha \rceil$ 라 둘 수 있습니다. q_0 가 음수가 되는 경우에 주의해야 합니다.
- 위 부등식으로 실제 답은 $q_0 - 2, q_0 - 1, q_0$ 중 하나임을 알 수 있으므로, G 를 상수 번만 계산하면 lower bound를 얻습니다.
- $p = 0$ 이면 모든 $D_i = 1$ 인 경우이므로 앞에서 설명한 방법으로 별도 처리합니다.

1F. 노크한 과수원은 똑똑한 과수원이다

- 마지막으로 현재 스위핑 구간을 $[S, E]$ 라 합시다.
- $T = kq + r$ ($0 \leq r < k$) 이므로, 하나의 q 에 대응하는 T 는 항상 $[kq, kq + k - 1]$ 안에 있습니다.
- 따라서 S 와 E 를 포함하는 두 블록을 제외하면, q 의 블록 전체가 현재 스위핑 구간 안에 들어옵니다.
- 가능한 q 의 구간과 $\lfloor S/k \rfloor, \dots, \lfloor E/k \rfloor$ 을 교차시킨 뒤, 양 끝의 q 에서만 $T = H(q)$ 가 실제로 $[S, E]$ 에 들어오는지 확인하면 됩니다.
- 모든 이벤트를 스위핑하면서 이를 반복하여 답의 개수와 최댓값을 구합니다.
- 전체 시간 복잡도는 $\mathcal{O}(NT_{\max}^{1/3} \log D_{\max})$ 수준입니다.

1D. 반복을 좋아하는 떼려어린이

ad_hoc, number_theory, graph, constructive

출제진 의도 - **Encore**

- **Div. 1** — 처음 푼 참가자: -, -
- **Div. 1** — 처음 푼 팀: -, -
- 출제자: cywohoy

1D. 반복을 좋아하는 떼라 어린이

- 모든 인덱스와 시행 번호를 0부터 세겠습니다.
- $d = \gcd(N, M)$, $N' = N/d$, $M' = M/d$, $L = \text{lcm}(N, M)$ 라 합시다.
- $L = M'N = N'M$ 번 시행하면 칸과 카드에서 선택되는 인덱스가 모두 처음으로 돌아옵니다.
- $S_s = \sum_{y=0}^{M'-1} c_{s+dy}$ 라 하면 한 주기 동안 b_i 에는 $S_{i \bmod d}$ 가 더해집니다.
- 따라서 $k = uL + v$ ($0 \leq v < L$)로 두면

$$b_i^k = uS_{i \bmod d} + b_i^v.$$

- 이제 v 를 고정하고 가능한 u 와 카드 값을 찾는 문제로 바꿉니다.

1D. 반복을 좋아하는 떼라 어린이

- **서브태스크 1 (6점):** $M = 1$
- $k = qN + r$ 라 하면 앞의 r 칸은 $(q + 1)c_0$, 나머지는 qc_0 입니다.
- 따라서 목표 배열이 두 상수 구간으로 나뉘는지 확인하면 됩니다.
- **서브태스크 2 (9점):** $M \mid N$
- 같은 칸은 항상 같은 카드를 사용하므로 $a_i = (q + [i < r])c_{i \bmod M}$ 입니다.
- r 을 열거하며 모든 카드에서 같은 q 가 가능한지 확인하면 됩니다.

1D. 반복을 좋아하는 떼라 어린이

- 이제 하나의 v 를 고정합시다.
- 문제는 $i \bmod d$ 가 같은 원소끼리 d 개의 블록으로 나눕니다.
- 블록 s 에서 $A_x = a_{s+dx}$, $C_y = c_{s+dy}$ 라 합시다.
- 처음 v 번 중 이 블록에 속하는 시행의 수는 $R_s = \max(0, \lceil (v-s)/d \rceil)$ 입니다.
- 처음 R_s 번의 기여를

$$F_x = \sum_{\substack{0 \leq t < R_s \\ t \equiv x \pmod{N'}}} C_{t \bmod M'}$$

라 하면 필요한 조건은 $A_x = uS_s + F_x$ 입니다.

1D. 반복을 좋아하는 떼라 어린이

- 한 블록에서 F_x 의 다음 시작점을 따라가면 반드시 사이클 $\mathcal{C}$ 가 생깁니다.
- 그 크기를 m 이라 하면, 사이클 위의 F_x 들을 모두 더했을 때 모든 카드가 같은 횟수 q 만큼 등장합니다.
- 따라서

$$\sum_{x \in \mathcal{C}} A_x = (mu + q)S_s.$$

- $S_s \neq 0$ 이면 $|S_s| \geq 1$ 이므로 $u \leq a_{\max}$ 입니다.
- 모든 $S_s = 0$ 이면 완전한 주기의 효과가 없으므로 $u = 0$ 으로 바꿀 수 있습니다.
- 즉 항상 $0 \leq u \leq a_{\max}$ 인 해만 확인하면 충분합니다.

1D. 반복을 좋아하는 떼라 어린이

- **서브태스크 3 (11점):** $N \leq 10, \max |a_i| \leq 10$
- $0 \leq u \leq a_{\max}, 0 \leq v < L$ 을 모두 열거합니다.
- 고정된 $k = uL + v$ 에서는 각 a_i 가 $c_0, \dots, c_{M-1}$ 의 선형식입니다.
- 크기가 작으므로 정수 연립방정식의 해 존재 여부를 직접 판정할 수 있습니다.

1D. 반복을 좋아하는 때라 어린이

- 이제 N 이 큰 경우를 위해 고정된 u, v 를 더 구조적으로 처리하겠습니다.
- $T_s = uS_s$ 라 두면 $A_x = T_s + F_x$ 입니다.
- T_s 를 없애기 위해 $\Delta_x = A_x - A_{(x-M') \bmod N'}$ 를 생각합시다.
- 두 F 의 공통 부분이 모두 소거되어

$$\Delta_x = \sum_{\substack{0 \leq t < M' \\ t \equiv x \pmod{N'}}} C_t - \sum_{\substack{R_s \leq t < R_s + M' \\ t \equiv x \pmod{N'}}} C_{t \bmod M'}.$$

- $M' < N'$ 이므로 두 합에는 각각 항이 최대 하나만 존재합니다.
- 따라서 $C_p - C_q = \Delta_x, C_p = \Delta_x, C_q = -\Delta_x, \Delta_x = 0$ 중 하나만 남습니다.

1D. 반복을 좋아하는 떼라 어린이

- $C_0, \dots, C_{M'-1}$ 을 정점으로 하는 그래프를 만듭니다.
- $C_p - C_q = \Delta$ 이면 두 정점 사이의 차이를 간선으로 기록하고, $C_p = \Delta$ 는 정점의 값을 고정합니다.
- 각 C_y 는 두 경계에 각각 한 번씩만 등장하므로 모든 연결 성분은 단일 경로이거나 사이클입니다.
- 경로에서는 고정된 값에서 시작해 모든 값을 유일하게 정할 수 있습니다.
- 사이클에서는 상대적인 값만 정해지고, 사이클 전체에 같은 정수를 더하는 자유도 하나가 남습니다.
- 순회 중 모순이 생기면 현재의 v 는 불가능합니다.

1D. 반복을 좋아하는 때라 어린이

- 경로의 값은 정하고, 각 사이클의 기준값을 0으로 둔 해를 C^0 라 합시다.
- $S_s^0 = \sum C_y^0$, $T_s^0 = A_0 - F_0^0$ 를 계산합니다.
- 자유 사이클 j 의 크기를 m_j , 전체에 1을 더했을 때 F_x 의 증가량을 q_j 라 합시다.
- 사이클에 z_j 를 더하면 S_s 는 $m_j z_j$ 증가하고 T_s 는 $q_j z_j$ 감소합니다.
- 따라서 고정된 u 에서 남은 조건은 $\sum_j (q_j + u m_j) z_j = T_s^0 - u S_s^0$ 이고,
- 정수해가 존재할 필요충분조건은

$$\gcd_j(q_j + u m_j) \mid T_s^0 - u S_s^0.$$

1D. 반복을 좋아하는 떼라 어린이

- 여기까지로 서브태스크 4, 5를 해결할 수 있습니다.
- 서브태스크 4 (14점): $N \leq 30$
- 모든 u, v 에 대해 위 그래프를 처음부터 만들고 gcd 조건을 검사해도 충분합니다.
- 서브태스크 5 (17점): $\max |a_i| \leq 1$
- $u = 0, 1$ 뿐이므로 같은 느린 풀이가 충분합니다.
- 전체 조건에서는 같은 v 에서 많은 u 를 검사해야 하므로, 사이클들을 더 압축해야 합니다.

1D. 반복을 좋아하는 떼라 어린이

– 서브태스크 6 (43점): 추가 제한이 없음

- 같은 블록의 모든 자유 사이클은 사실 같은 (m, q) 를 가집니다.
- 사이클 i 에서 $H_i = \sum_{x \in \mathcal{C}_i} A_x = m_i T_s + q_i S_s$ 라 합시다.
- 다른 사이클 j 를 움직여도 H_i 는 변하지 않으므로 $m_i q_j = m_j q_i$ 입니다.
- 또한 사이클의 원형 순서가 보존되어 $\gcd(m_i, q_i) = 1$ 이므로 모든 (m_i, q_i) 가 같습니다.
- 따라서 모든 자유도를 하나의 정수 w 로 합칠 수 있습니다.

1D. 반복을 좋아하는 떼라 어린이

- 자유 사이클 하나에만 w 를 더하면

$$S_s = S_s^0 + mw, \quad T_s = T_s^0 - qw.$$

- $T_s = uS_s$ 를 대입하면

$$(mu + q)w = T_s^0 - uS_s^0.$$

- 따라서 블록마다 S_s^0, T_s^0, m, q 만 저장하면 됩니다.
- $D = mu + q, E = T_s^0 - uS_s^0$ 라 하면 $D > 0$ 에서는 $D \mid E, D = 0$ 에서는 $E = 0$ 인지 확인합니다.
- 하나의 u 는 모든 d 개 블록에서 $\mathcal{O}(d)$ 에 검사할 수 있습니다.

1D. 반복을 좋아하는 떼라 어린이

- 전체 알고리즘을 정리합니다.
- $v = 0, 1, \dots, L-1$ 을 순회하며 각 블록의 그래프를 한 번 분석합니다.
- 블록마다 S_s^0, T_s^0, m, q 를 저장합니다.
- $u = 0, 1, \dots, a_{\max}$ 를 순회하며 모든 블록의 나눗셈 조건을 확인합니다.
- 가능한 u 를 찾으면 $w = E/D$ 를 구해 자유 사이클 하나에 더합니다.
- 모든 블록의 카드 값을 복원한 뒤 $k = uL + v$ 를 출력합니다.

1D. 반복을 좋아하는 떼라 어린이

- 경계 차분은 T_s 만 제거하며, $\gcd(N', M') = 1$ 이므로 정보가 손실되지 않습니다.
- 그래프의 모든 해는 기준해에서 자유 사이클들을 평행이동한 형태입니다.
- 따라서 위 나눗셈 조건은 정확히 $T_s = uS_s$ 를 만족시킬 필요충분조건입니다.
- 또한 $|\Delta_x| \leq 2a_{\max}$ 이고 연결 성분의 크기는 최대 M' 이므로

$$|c_i| \leq 5(M')^2 a_{\max} < 10^9$$

인 해를 항상 선택할 수 있습니다.

1D. 반복을 좋아하는 떼라 어린이

- 고정된 v 에서 모든 블록의 그래프를 만드는 비용은 $\mathcal{O}(N + M)$ 입니다.
- 하나의 u 를 검사하는 비용은 $\mathcal{O}(d)$ 입니다.
- 따라서 전체 시간 복잡도는

$$\mathcal{O}(L(N + M) + Ld(a_{\max} + 1)) = \mathcal{O}(N^3 + N^2 a_{\max}).$$

- $a_{\max} \leq 1000$ 이므로 이 문제에서는 $\mathcal{O}(N^3)$ 입니다.
- 공간 복잡도는 $\mathcal{O}(N)$ 입니다.