UCPC 2026

전국 대학생 프로그래밍 대회 동아리 연합
여름 대회 2026

Finals

Official Solutions

본선 해설

2026년 7월 12일, 11:00 – 17:00

주최

전국 대학생 프로그래밍 대회 동아리 연합

후원

Jane Street®
HRT hudson river trading
HYUNDAI MOBIS
CITADEL | CITADEL Securities
MOLOCO
ASTEROMORPH
NEXON
STARTLINK

김동현(kdh9949) 김영현(kipa00) 나정휘(jhnah917) 정재호(cubic)

UCPC 2026

Finals - 본선대회

예상 난이도

Final – 본선대회

문제		의도한 난이도	출제자
A	!New Game Start!	**Challenging**	dadas08
B	11 싫어 수	**Easy**	bubbler
C	C4ESAR	**Easy**	hibye1217
D	비상 물자 보급	**Easy**	ncy09
E	Shift	**Hard**	pk661
F	배수	**Hard**	pk661
G	Vector Rails	**Medium**	pauljjang410
H	판다스틱	**Hard**	mujigae
I	Yet Another Binary Problem	**Challenging**	pyb1031
J	무한 문자열 정렬	**Hard**	azberjibiou
K	갑천 자전거길	**Medium**	queued_q
L	아침약	**Easy**	psb0623
M	악령 퇴치	**Medium**	functionx
N	지그재그	**Medium**	79brue

문제 A. !!New Game Start!!

출제자 양승민 (dadas08)

먼저, 쿠폰을 처리해봅시다.

쿠폰을 2개 이상 사용한다면, 마지막에 실질적으로 적용되는 쿠폰을 제외한 나머지 쿠폰을 모두 삭제해도 최종 층수는 동일함을 증명할 수 있습니다. 그러므로, 쿠폰은 아예 사용하지 않거나 최대 하나만 사용합니다.

쿠폰 효과가 발동되기 전에 체크포인트를 사용하는 것도, 동일한 이유로 고려하지 않아도 됩니다.

그렇다면, 각 쿠폰을 쓰는 경우를 처리하려면 적절한 $t \in [L_i, R_i]$ 를 골라, t에서 높이 X_i로 시작해서 이후 체크포인트를 최대 $\lfloor \frac{W-X_i}{D} \rfloor$번 사용해 얻을 수 있는 최대 층수를 구하면 됩니다.

$f(i,j)$를 i번 스테이지 이후 0층에서 시작해서, 최대 j번의 체크포인트 선언을 통해 도달할 수 있는 최대 층수로 정의하면, 각 쿠폰에 대한 답은

$$\max_{t \in [L_i, R_i]} f(t, \lfloor \frac{W - X_i}{D} \rfloor) + X_i$$

가 됩니다.

이제, $f(i,j)$ 를 빠르게 관리하는 문제로 환원됩니다. 간단한 식 정리를 통해, 다음과 같은 식이 성립함을 알 수 있습니다.

$$f(i,j) = \min(f(i+1,j) + A_i,\ \min_{i<t} f(t, j-1))$$

위 식 그대로 최적화하긴 어렵지만, 위 식을 바탕으로 $f(i,j)$를 평면에 그려볼 수는 있습니다. 이때 알 수 있는 것은, $f(i,j+1) - f(i,j)$ $(1 \le i \le N)$ 의 값이 거의 변하지 않는다는 것입니다. 엄밀히는,

$$f(i,j+1) - f(i,j) \neq f(i-1,j+1) - f(i-1,j)$$

인 (i,j) 쌍의 개수는 최대 N개 존재합니다. 증명은 생략합니다.

이제 다음과 같은 알고리즘을 생각해볼 수 있습니다.

초기에는 $S_N = 0, S_i = S_{i+1} + A_{i+1}$ 로 정의되는 길이 N의 수열을 만듭니다. (각각 $f(0,0), f(1,0), f(2,0) \dots$ 를 의미합니다.) 이후, $f(i,1) - f(i,0) = d_i$라 할 때, $d_i \neq d_{i+1}$ 가 되는 지점을 모두 찾아, 수열의 prefix에 적당한 수를 더하는 연산을 반복하면 수열의 각 값이 $f(0,1), f(1,1), f(2,1) \dots$ 가 되게 됩니다. 이를 반복하면서 각 쿠폰마다 range maximum 쿼리 한 번으로 원하는 값을 구할 수 있습니다.

d_i가 변화하는 지점을 찾으려면, $S_i < \max(S_{i+1}, S_{i+2}, S_{i+3} \dots S_N)$ 가 되는 모든 i를 찾으면 됩니다. 이 또한 segment tree를 이용해 처리 가능합니다.

세그먼트 트리와 이분탐색을 이용하면 $O(N \log^2 N + Q \log N)$, segment tree walk를 이용하면 $O((N+Q) \log N)$ 에 해결할 수 있습니다.

문제 B. 11 싫어 수

출제자 곽우석 (bubbler)

먼저, 어떤 수가 11 싫어 수일 조건을 구체적으로 알아봅시다.

1자리와 2자리 수는 11 싫어 수가 될 수 없음을 어렵지 않게 알 수 있으므로, n의 길이 $\ell \geq 3$입니다.

10^i의 자리의 숫자가 d일 때, 이 숫자를 바꿔서 얻을 수 있는 수는 $n - 10^i d, n - 10^i d + 10^i, \cdots, n - 10^i d + 9 \cdot 10^i$의 10개($n$ 포함)이며, 이들은 11로 나눈 나머지가 모두 서로 다릅니다.

또한, 10^i의 자리의 숫자 d_i와 10^{i+2k}의 자리의 숫자 d_{i+2k}가 서로 다르다면, $10^i d_i$와 $10^{i+2k} d_{i+2k}$를 11로 나눈 나머지가 서로 다르게 되므로, 둘 중 하나를 적절히 바꾸는 것으로 11로 나눈 나머지 11가지를 모두 만들 수 있게 됩니다.

따라서, 11 싫어 수는 $ababab\cdots aba$ 또는 $bababa\cdots baba$의 형태여야 함을 알 수 있습니다. 편의상 1의 자리의 숫자를 a, 10의 자리의 숫자를 b로 통일합니다.

길이가 홀수 $2k+1$일 경우, $n \equiv (k+1)a - kb \bmod 11$입니다. 한 자리를 바꿔서 $\bmod 11$로 0이 나오지 않게 하려면, 어떤 a를 0으로 바꿨을 때 $\bmod 11$로 1이 나오고, 어떤 b를 0으로 바꿨을 때는 $\bmod 11$로 10이 나오면 됩니다. 이를 수식으로 나타내면 다음과 같습니다.

$$ka - kb \equiv 1 \bmod 11 \tag{1}$$
$$(k+1)a - (k-1)b \equiv 10 \bmod 11 \tag{2}$$

이제 (2)에서 (1)을 빼면 $a + b \equiv 9 \bmod 11$을 얻고, 따라서 $2ka \equiv 1 + 9k \bmod 11$, 즉 $a \equiv (1+9k)(2k)^{-1} \bmod 11$이 됩니다.

예를 들어 $k = 1$을 대입하면 $a = 5, b = 4$가 되어 545를 얻을 수 있습니다.

$k \equiv 0 \bmod 11$이면 답이 존재하지 않고, 또한 $a \equiv 0 \bmod 11$, 즉 $k \equiv 6 \bmod 11$이면 짝수 길이의 수와 중복됩니다.

길이가 짝수 $2k$일 경우, $n \equiv ka - kb \bmod 11$입니다. 같은 방식으로 식을 세우면 다음을 얻습니다.

$$(k-1)a - kb \equiv 1 \bmod 11 \tag{3}$$
$$ka - (k-1)b \equiv 10 \bmod 11 \tag{4}$$

이 경우 역시 (4)에서 (3)을 빼서 $a + b \equiv 9 \bmod 11$을 얻을 수 있고, 추가로 (4)와 (3)을 더해서 $(2k-1)(a-b) \equiv 0 \bmod 11$을 얻을 수 있습니다.

$2k - 1 \not\equiv 0 \bmod 11$이면 $a \equiv b \bmod 11$이어야 하는데, 그러면 $a = b = 10$이 되어 불가능합니다. 따라서 $2k - 1 \equiv 0 \bmod 11$, 즉 $k \equiv 6 \bmod 11$을 얻습니다. 이제 a와 b는 합이 9인 아무 쌍이면 모든 조건을 충족합니다.

이제 범위 내의 11 싫어 수를 구하려면 다음과 같이 할 수 있습니다.

- A보다 길고 B보다 짧은 11 싫어 수의 개수를 구합니다. 이는 각 길이에 대해 상수 시간에 구할 수 있습니다.
- A나 B와 길이가 같은 11 싫어 수들을 모두 만들어 본 다음, 문자열 비교로 범위 내에 들어오는 것의 개수를 셀 수 있습니다. 이는 $\mathcal{O}(\log A + \log B)$ 시간에 구할 수 있습니다.
- A와 B의 길이가 서로 같은 경우는 따로 처리합니다.

문제 C. C4ESAR

출제자 이성현 (hibye1217)

우선 가능한지 여부부터 고민해봅시다. 카이사르 암호를 적용한다고 숫자가 알파벳이 되거나 알파벳이 숫자가 되지는 않으니, 알파벳과 숫자가 괄호 쌍처럼 매치가 되어야 합니다.

이거면 충분할까요? 알파벳은 26종류이고 숫자는 10종류이므로 매치되는 알파벳과 숫자의 홀짝성 역시 고려해야 합니다. $\gcd(26, 10) = 2$이고 카이사르 암호는 1칸씩 이동하므로 이외에는 고려해야 할 부분이 없습니다.

이제 가능할 때의 최소 연산 횟수를 구해봅시다. C4를 지우는 연산은 항상 $N/2$번 해야 하기 때문에, 카이사르 암호를 적용한 횟수만 구해도 됩니다.

$f(l, r)$를 l번째 글자랑 r번째 글자를 각각 C랑 4로 만들기 위해 필요한 카이사르 암호의 연산 횟수라고 하고, d_p를 글자 쌍 $p = (l, r)$을 없애기 위해 필요한 카이사르 암호의 연산 횟수라고 합시다. 그러면 d_p는 p 내부에 있는 모든 쌍 q에 대해 $\max d_q$ 이상이면서 $d_p \equiv f(l, r) \pmod{130}$여야 합니다. 이 값은 쌍마다 $O(1)$에 구해줄 수 있습니다.

글자 쌍은 $O(N)$개 있고 이들끼리 트리 구조를 이루기 때문에 $O(N)$ 시간에 답을 구할 수 있습니다.

문제 D. 비상 물자 보급

출제자 서유완 (ncy09)

지문에서 설명된 보급 계획은 1을 루트로 하는 트리에서의 깊이 우선 탐색(dfs)과 같습니다.

i번 도시가 수령하게 되는 물자 가치를 A_i라 합시다.

[보조 정리] 임의의 두 도시 $i \neq j$에 대해 $A_i < A_j$일 확률 $P(A_i < A_j)$는 다음과 같다.

$$P(A_i < A_j) = \begin{cases} 1 & i\text{가 } j\text{의 조상일 때} \\ 0 & j\text{가 } i\text{의 조상일 때} \\ \frac{1}{2} & \text{나머지 경우} \end{cases}$$

[증명]. 1을 루트로 하는 트리에서 i가 j의 조상이면, 어떤 dfs 순서 상에서도 항상 i를 먼저 방문하므로 $A_i < A_j$입니다. 반대로 j가 i의 조상이라면 항상 $A_i > A_j$입니다.

서로 조상 관계가 아닌 경우, 두 도시의 최소 공통 조상을 x라고 합시다. A_i와 A_j의 대소 관계는, x에서 i가 속한 서브트리와 j가 속한 서브트리 중 어느 쪽을 먼저 방문하는지와 동치입니다. 각 점에서 자식 노드를 선택하는 순서는 균등한 확률로 정해지므로, 각 서브트리를 먼저 방문할 확률은 $\frac{1}{2}$로 동일합니다. ■

이제 특정 도시 k를 고정하고, k번 도시가 수령하게 되는 물자 가치의 기댓값을 구해봅시다.

$1 \le j \le N (j \neq k)$에 대해 X_j라는 확률변수를 다음과 같이 정의합니다.

$$X_j = \begin{cases} 1 & A_j < A_k \\ 0 & A_j > A_k \end{cases}$$

그러면 다음과 같은 식이 성립합니다.

$$A_k = 1 + \sum_{j \neq k} X_j$$

양변에 기댓값을 취하면 다음과 같습니다.

$$E[A_k] = 1 + \sum_{j \neq k} E[X_j] = 1 + \sum_{j \neq k} P(A_j < A_k)$$

보조 정리의 결과를 이용하면 이를 계산할 수 있습니다.

구체적으로 k의 조상의 수를 a_k, k를 루트로 하는 서브트리의 크기를 s_k라 하면 $E[A_k] = 1 + a_k + \frac{1}{2}(N - a_k - s_k)$입니다.

각각의 k에 대한 a_k와 s_k는 트리 dp로 $O(N)$에 계산할 수 있습니다.

문제 E. Shift

출제자 손재원 (pk661)

RowShift(i)와 ColShift(P_i)를 합쳐 Step(i)라고 하자.

어떤 원소가 Step(r) 직전에 (r, c)에 있다고 하자. 만약 $c+1 = P_r$이면 1번의 Step 이후에 $(r+1, c+1)$로 이동한다. 반대로 $c+1 \neq P_r$이면 $N+1$번의 Step 이후에 $(r+1, c+1)$로 이동한다.

$d = c-r$이라고 하자. (r, c)에 있는 원소는 $(r, c) \to (r+1, c+1) \to \cdots \to (r-1, c-1) \to (r, c)$의 과정을 따라 다시 제자리로 돌아온다. 이때 $P_i - i - 1 \equiv d \pmod N$인 i의 개수를 X_d라고 하면, X_d번의 이동은 1번의 Step, 나머지 $N - X_d$번의 이동은 $N+1$번의 Step이 필요하다. 그러므로 필요한 Step의 수는

$$1 \cdot X_d + (N+1) \cdot (N - X_d) = N(N+1-X_d)$$

이다. 필요한 **조작**의 횟수는 $N+1-X_d$이다.

그러므로 전체 격자판이 원래 상태로 돌아오기 위한 최소 조작 횟수는 $N+1-X_0, N+1-X_1, \cdots, N+1-X_{N-1}$의 최소공배수이다.

문제 F. 배수

출제자 손재원 (pk661)

A_i의 배수가 되는 구간을 I_i라고 하자. 이때 **아름다움**은 $\sum_{i=1}^{N} |I_i|$이다. 주어진 입력에 의해 각 I_i의 왼쪽 끝점은 어떤 값 B_i 이상이 되어야 한다.

서로 교차하는 두 구간은 포함 관계가 되도록 바꾸어도 문제 조건이 여전히 성립함을 관찰하자. 따라서 $\{I_1, I_2, \cdots, I_N\}$이 Laminar Set Family인 것이 최적임을 알 수 있다.

이제 동적 계획법을 이용하여 문제를 해결한다. $D[l][r]$을 $l \le i \le r$에 대해 $I_i \subseteq [l, r]$이 되도록 할 때 $\sum_{i=l}^{r} |I_i|$의 최댓값으로 정의하자. $E[l][r]$은 그중에서 $I_i = [l, r]$인 원소가 적어도 하나 있을 때의 최댓값으로 정의하자.

$E[l][r]$을 계산할 때, $I_i = [l, r]$인 i를 $i_1 < i_2 < \cdots < i_k$라고 하자. 단, $B_{i_j} \le l$이어야 한다. 점화식을 세워보면 다음과 같다. (단, $i_0 = l - 1, i_{k+1} = r + 1$)

$$E[l][r] = \max_{l \le i_1 < i_2 < \cdots < i_k \le r} \left(k(r - l + 1) + \sum_{j=0}^{k} D[i_j + 1][i_{j+1} - 1] \right)$$

$D[l][r]$의 점화식은 다음과 같다.

$$D[l][r] = \max_{l \le i \le r} (E[l][i] + D[i + 1][r])$$

정답은 $D[1][N]$이며, 전체 시간복잡도는 $O(N^4)$이다.

문제 G. Vector Rails

출제자 정민규 (pauljjang410)

기점 $(0,0)$에서 시작해, i번 건설 모드를 선택하고 비용 $t_i \geq 0$을 투입하면 벡터 $\vec{v}_i = (a_i, b_i)$의 t_i배만큼 선로가 연장된다. 목표 도시 $\vec{P} = (p_x, p_y)$에 도달한다는 것은

$$t_1\vec{v}_1 + t_2\vec{v}_2 + \cdots + t_N\vec{v}_N = \vec{P}, \qquad t_1, \ldots, t_N \geq 0$$

을 만족시키는 계수들이 존재한다는 뜻이고, 총 비용 $T = t_1 + \cdots + t_N$을 최소화하는 것이 목표이다. 두 벡터 $\vec{u}, \vec{v}$의 외적은 $\vec{u} \times \vec{v} := u_x v_y - u_y v_x$로 쓴다.

핵심 아이디어는 $\vec{P}$ 방향 반직선이 벡터들의 볼록 껍질 경계를 뚫고 나가는 가장 먼 지점을 찾는 것이다. 위 식의 양변을 T로 나누면 $\sum_i \frac{t_i}{T}\vec{v}_i = \frac{\vec{P}}{T}$이고 계수 $\frac{t_i}{T}$는 음이 아니며 합이 1이므로, $\frac{\vec{P}}{T}$는 $\mathrm{conv}(\vec{v}_1, \ldots, \vec{v}_N)$ 위의 점이다. T 최소화는 $\mu := \frac{1}{T}$ 최대화와 같고, 이 최댓값에서 $\mu\vec{P}$는 볼록 껍질의 경계 위에 있어야 하며 경계 위의 점은 인접한 두 꼭짓점만의 결합이므로 최적해는 많아야 벡터 두 개를 쓴다.

반직선이 껍질을 관통하면 경계와 두 번(진입점, 탈출점) 만나는데, 원점에서 먼 탈출점 쪽이 항상 유리하므로 원점에서 가려진 능선만 후보가 된다. 이를 골라내는 방법은 원점 O를 벡터들과 함께 점으로 추가해 볼록 껍질을 다시 구하는 것이다. O가 새 꼭짓점으로 편입되면 가려졌던 능선이 자동으로 떨어져 나가고, 이 껍질에서 O를 다시 빼면 후보 벡터 $M(\leq N)$개가 남는다.

이 M개를 원점 기준 극각(polar angle) 순으로 정렬한다(양의 x축으로 반평면을 나누고 같은 반평면 안에서는 정수 외적의 부호로 비교하면 부동소수 없이 정렬 가능). 인접한 두 벡터 $\vec{A}, \vec{B}$마다 부채꼴이 하나씩 생기고, $\vec{P}$가 속한 부채꼴에 따라 다음 세 경우로 나뉜다.

- $\vec{A} \times \vec{B} > 0$ (사잇각 $< 180°$): $\vec{A}, \vec{B}$ 두 벡터만으로 도달 가능.
- $\vec{A} \times \vec{B} = 0$ (사잇각 $= 180°$, 정반대 방향이거나 $M = 1$): 이 직선 위의 점만 도달 가능.
- $\vec{A} \times \vec{B} < 0$ (사잇각 $> 180°$): O가 껍질 경계에 있어 생기는 구멍으로, 이 방향은 어떤 조합으로도 도달 불가능 — 무조건 -1.

$\vec{A} \times \vec{B} > 0$이면 $t_A\vec{A} + t_B\vec{B} = \vec{P}$의 양변에 $\vec{B}$를 외적해

$$t_A = \frac{\vec{P} \times \vec{B}}{\vec{A} \times \vec{B}}, \qquad t_B = \frac{\vec{A} \times \vec{P}}{\vec{A} \times \vec{B}}, \qquad T = t_A + t_B = \frac{\vec{P} \times \vec{B} + \vec{A} \times \vec{P}}{\vec{A} \times \vec{B}}$$

를 얻는다. $t_A, t_B \geq 0$은 $\vec{A} \times \vec{P} \geq 0$, $\vec{P} \times \vec{B} \geq 0$으로 확인한다. $\vec{A} \times \vec{B} = 0$이면 $\vec{A} \times \vec{P} = 0$을 확인한 뒤, $\vec{A} \cdot \vec{P} \geq 0$이면 $\sqrt{\frac{|\vec{P}|^2}{|\vec{A}|^2}}$, $\vec{B} \cdot \vec{P} \geq 0$이면 $\sqrt{\frac{|\vec{P}|^2}{|\vec{B}|^2}}$ 중 가능한 값의 최솟값을 답으로 한다.

$\vec{P} = O$이면 비용 0으로 따로 처리한다. 그 외에는 극각 정렬된 M개에 이분 탐색을 하면 $O(\log M)$에 후보 구간을 찾는데, 경계 사례를 위해 찾은 위치 바로 앞뒤 두세 구간을 함께 확인한다.

좌표가 10^9 이하라 외적은 최대 2×10^{18}, T의 분자는 최대 4×10^{18}로 `long long` 안에서 오차 없이 계산되고, 극각 비교를 `atan2` 같은 부동소수로 하면 이 정밀도에서 위험하므로 피한다. 볼록 껍질과 정렬에 $O(N \log N)$, 쿼리당 이분 탐색에 $O(\log M)$이 걸려 전체 $O((N+Q)\log N)$에 해결된다.

문제 H. 판다스틱

출제자 서종환 (mujigae)

tag: math, hall's theorem, ad_hoc, constructive, greedy

판다가 좋아하는 스틱의 집합마다 명칭과 해당하는 판다의 수를 다음과 같이 정하자.

$$\begin{aligned} &\{a\} : S_a && \{b\} : S_b && \{c\} : S_c \\ &\{a,b\} : X && \{b,c\} : Y && \{c,a\} : Z \\ &\{a,b,c\} : S_{abc} \end{aligned}$$

$$s_a = |S_a|, \quad s_b = |S_b|, \quad s_c = |S_c|, \quad x = |X|, \quad y = |Y|, \quad z = |Z|, \quad s_{abc} = |S_{abc}|$$

홀의 결혼 정리를 이용하면, 순서와 관계 없이 모든 판다에게 좋아하는 스틱을 하나씩 배정할 수 있는지를 먼저 판별할 수 있다. 확인해야 하는 비자명한 조건은 다음 여섯 가지이다.

$$\begin{aligned} &s_a \le A, && s_b \le B, && s_c \le C, \\ &s_a + s_b + x \le A + B, && s_b + s_c + y \le B + C, && s_c + s_a + z \le C + A. \end{aligned}$$

이제 하나의 줄로 모든 스틱 순서에 대응할 수 있는지 판별해야 한다. 먼저 다음 두 가지 관찰을 할 수 있다.

- S_a, S_b, S_c에 속한 판다들은 모두 줄의 맨 앞에 몰아 둘 수 있다.
- S_{abc}에 속한 판다들은 모두 줄의 맨 뒤에 몰아 둘 수 있다.

문제는 남은 X, Y, Z를 가운데에 어떻게 배치할 것인지로 줄어든다. 이를 해결할 수 있는 두 가지 방법을 살펴보자. 문제를 더 단순화하여 s_a, s_b, s_c, s_{abc}는 모두 0이라 생각하고 $N = x + y + z$라고 가정한다.

풀이 1.

어떤 해가 존재한다면, 그 해에서 각 타입의 마지막 등장 순서를 일반성을 잃지 않고 X, Y, Z라 하자. 홀의 정리를 이용하여, 이러한 해가 존재한다면 다음 세 부등식이 성립해야 한다는 것을 증명할 수 있다.

$$x \le A + B, \quad y \le C + \max(0, B - x), \quad z \le \max(0, A - x) + \max(0, C - y)$$

반대로 위 세 부등식이 성립한다면, 다음 형태의 배치를 갖는 해가 존재함도 귀납법을 이용해 증명할 수 있다.

$$\underbrace{X X \cdots X}_{x} \underbrace{Y Y \cdots Y}_{y} \underbrace{Z Z \cdots Z}_{z}$$

따라서 해의 존재성과 세 부등식의 성립은 동치이며, 해가 존재한다면 항상 위 형태의 해를 구성할 수 있다. 등장 순서 6가지에 대해 부등식을 확인하고 성립하는 순서 중 아무거나 하나를 이용해 해를 구성하면 된다. 성립하는 순서가 없다면 해가 존재하지 않는 것이다.

풀이 2.

맨 뒤에 배치할 수 있는 판다의 조건을 확인해보자. X가 맨 뒤에 들어가기 위해서는 모든 Y가 b를 먹고, 모든 Z가 a를 먹더라도 a 또는 b가 남아야 한다. 홀의 정리로 조건을 확인하면 되고, 맨 뒤에 배치할 수 있는 타입이 2가지 이상일 때는 어느 것을 먼저 배치하더라도 이후 앞의 배치에 영향을 주지 않는다는 것도 식을 통해 확인할 수 있다. 그러므로 뒤에서부터 조건을 만족하는 타입을 아무거나 그리디하게 배치하면 되고, 중간에 막히면 해가 존재하지 않는 것이다.

문제 I. Yet Another Binary Problem

출제자 박영빈 (pyb1031)

일단 어떤 매칭이 존재한다고 가정해보자. 다음 조건을 만족하는 2차원 배열 B를 생각해보자:

$a_{i,j} = 0$이고 (i, j)가 가로 매칭에 포함될 시 $B_{i,j} = 1$

$a_{i,j} = 1$이고 (i, j)가 세로 매칭에 포함될 시 $B_{i,j} = 1$

그 외의 위치에서 $B_{i,j} = 0$이다.

잘 생각해보면 B의 i행에 있는 1의 개수는 a의 i행에 있는 1의 개수 이하이고, B의 i열에 있는 1의 개수는 a의 i열에 있는 0의 개수 이하이다.

반대로, 위의 개수 조건을 만족하는 임의의 B에서 B의 총 1의 개수와 같은 개수의 매칭을 구성할 수 있다.

따라서 남은 문제는 위의 조건을 만족하며 1의 개수가 최대가 되는 B를 구성하는 것이며, 이는 적절한 그리디로 해결할 수 있다.

문제 J. 무한 문자열 정렬

출제자 장근영 (azberjibiou)

문자열 A를 무한히 이어 붙인 문자열을 A^∞라고 하자. 다음 사실이 성립한다.

$$A^\infty < B^\infty \iff AB < BA.$$

증명은 뒤에서 다룬다. 따라서 두 구간에 대응하는 문자열을 각각 A, B라고 하면, A^∞와 B^∞를 비교하는 대신 유한 문자열 AB와 BA를 비교하면 된다.

먼저 Suffix Array와 LCP 배열을 이용해 S의 임의의 두 부분문자열을 $O(1)$에 비교할 수 있도록 전처리한다. 구체적인 방법과 증명은 뒤에서 설명한다.

이제 $a = |A|$, $b = |B|$라고 하자. AB에서는 a번째 문자 뒤에서 A가 끝나고, BA에서는 b번째 문자 뒤에서 B가 끝난다. 따라서 두 문자열을

$$0, \quad \min(a,b), \quad \max(a,b), \quad a+b$$

에서 자르면 최대 세 쌍의 문자열이 만들어진다. 각 문자열은 모두 원래 문자열 S의 연속한 부분문자열이므로, 한 쌍을 $O(1)$에 비교할 수 있다.

앞에서부터 각 쌍을 비교하여 처음으로 다른 쌍의 대소 관계를 사용하면 AB와 BA의 대소 관계를 알 수 있다. 비교해야 하는 쌍은 최대 세 개이므로, 두 무한문자열의 비교에는 $O(1)$의 시간이 걸린다.

모든 쌍이 같다면 $AB = BA$이고, 따라서 $A^\infty = B^\infty$이다. 이 경우에는 문제의 조건에 따라 구간의 번호가 작은 것을 앞에 둔다.

비교 한 번이 $O(1)$이므로, Q개의 구간을 정렬하는 데 $O(Q \log Q)$의 시간이 걸린다.

보조정리의 증명 각 알파벳을 1 이상 26 이하의 정수에 대응시키고, 문자열 T를 27진수로 보았을 때의 값을 $\mathrm{val}(T)$라고 하자. T를 무한히 반복한 27진 순환소수의 값은

$$\frac{\mathrm{val}(T)}{27^{|T|}-1}$$

이다.

27진 순환소수의 대소 관계는 대응하는 무한 문자열의 사전순 관계와 같다. 실제로 처음으로 다른 자리에서 발생하는 차이가 그 뒤의 모든 자리에서 발생할 수 있는 차이의 합보다 크기 때문이다.

$a = |A|$, $b = |B|$라고 하면 다음과 같다.

$$\begin{aligned}
A^\infty < B^\infty &\iff \frac{\mathrm{val}(A)}{27^a-1} < \frac{\mathrm{val}(B)}{27^b-1} \\
&\iff \mathrm{val}(A)(27^b-1) < \mathrm{val}(B)(27^a-1) \\
&\iff \mathrm{val}(A)27^b + \mathrm{val}(B) < \mathrm{val}(B)27^a + \mathrm{val}(A) \\
&\iff \mathrm{val}(AB) < \mathrm{val}(BA) \\
&\iff AB < BA.
\end{aligned}$$

같은 계산에서 등호도 보존되므로 $A^\infty = B^\infty$와 $AB = BA$ 역시 동치이다.

보조정리의 두 번째 증명 이번에는 문자열을 직접 비교하여 증명하자. 일반성을 잃지 않고 $|A| \le |B|$라고 가정하고, $|B| = p|A| + r$로 나타내자. 여기서 $p \ge 1$이고 $0 \le r < |A|$이다.

먼저 B의 길이 $p|A|$인 접두사가 A^p가 아닌 경우를 생각하자. A^x가 B의 접두사가 되도록 하는 가장 큰 음이 아닌 정수 x를 잡으면 $x < p$이다.

A^∞의 길이 $(x+1)|A|$인 접두사는 A^{x+1}이다. 한편 B의 접두사는 A^x이지만 A^{x+1}은 아니므로, A^∞와 B^∞는 길이 $(x+1)|A|$ 이내에서 처음으로 달라진다.

또한 B가 A^x로 시작하므로 AB는 A^{x+1}로 시작하고, BA는 B로 시작한다. 따라서 AB와 BA도 같은 위치에서 처음으로 달라지며, 그 대소 관계는 A^∞와 B^∞의 대소 관계와 같다.

이제 B의 길이 $p|A|$인 접두사가 A^p인 경우를 생각하자. 이때 $B = A^pC$로 쓸 수 있으며, $|C| = r < |A|$이다. C가 빈 문자열이라면 $B = A^p$이므로 $A^\infty = B^\infty$이고 $AB = BA$이다.

C가 비어 있지 않다면

$$AB = A^{p+1}C, \qquad BA = A^pCA$$

이므로, 공통 접두사 A^p를 제거하면 AB와 BA의 비교는 AC와 CA의 비교와 같다.

한편 $A^\infty = A^pA^\infty$이고 $B^\infty = A^pCB^\infty$이므로, A^∞와 B^∞의 비교는 A^∞와 CB^∞의 비교와 같다.

C가 A의 접두사가 아니라면 두 문자열은 처음 $|C|$자리 안에서 달라진다. 따라서 A^∞와 CB^∞의 비교는 A와 C의 비교와 같고, AC와 CA의 비교 역시 같은 위치에서 결정된다.

반대로 C가 A의 접두사라면, A^∞의 길이 $|A|+|C|$인 접두사는 AC이고, CB^∞의 길이 $|A|+|C|$인 접두사는 CA이다. 따라서 $AC \ne CA$라면 두 비교는 같은 위치에서 처음으로 달라지고, 대소 관계도 같다.

$AC = CA$라면 A와 C는 어떤 문자열을 각각 여러 번 반복한 형태이므로 $A^\infty = C^\infty$이다. 따라서 $B = A^pC$ 역시 같은 문자열을 반복한 형태이고, $A^\infty = B^\infty$와 $AB = BA$가 모두 성립한다.

모든 경우에 대해 A^∞와 B^∞의 비교 결과가 AB와 BA의 비교 결과와 같으므로 보조정리가 증명된다.

두 부분문자열을 $O(1)$에 비교하는 방법 S의 Suffix Array를 SA라고 하고, 접미사 $S[i,|S|]$의 Suffix Array에서의 위치를 $\mathrm{rank}[i]$라고 하자.

또한 인접한 두 접미사의 최장 공통 접두사 길이를 저장하는 LCP 배열을 만든다. 즉,

$$\mathrm{LCP}[k] = \mathrm{LCP}\big(S[\mathrm{SA}[k-1],|S|], S[\mathrm{SA}[k],|S|]\big)$$

로 정의한다.

두 접미사 $S[x,|S|]$, $S[y,|S|]$에 대해 $p = \mathrm{rank}[x] < \mathrm{rank}[y] = q$라고 하면,

$$\mathrm{LCP}\big(S[x,|S|], S[y,|S|]\big) = \min_{p<k\le q} \mathrm{LCP}[k]$$

가 성립한다. Suffix Array에서 두 접미사 사이에 있는 모든 인접한 접미사 쌍이 길이 t 이상의 공통 접두사를 가질 때, 그리고 그럴 때에만 양 끝의 두 접미사도 길이 t 이상의 공통 접두사를 가지기 때문이다.

따라서 두 접미사의 LCP를 구하는 문제는 LCP 배열 위의 구간 최솟값 질의가 된다. LCP 배열에 Sparse Table을 만들면 전처리에 $O(|S|\log|S|)$, 각 구간 최솟값 질의에 $O(1)$의 시간이 걸린다.

두 부분문자열의 길이가 L이고 시작 위치가 각각 x, y라면, 두 접미사의 LCP를 구해 L과의 최솟값을 취한다. 그 값이 L이라면 두 부분문자열은 같고, 그렇지 않다면 바로 다음 문자를 비교하여 대소 관계를 알 수 있다. 따라서 원래 문자열 S의 임의의 두 같은 길이 부분문자열을 $O(1)$에 비교할 수 있다.

복잡도 $n = |S|$라고 하자. Suffix Array, LCP 배열, Sparse Table을 만드는 데 $O(n\log n)$의 시간과 공간이 필요하다. 이후 비교 한 번은 $O(1)$이고, 전체 정렬에는 $O(Q\log Q)$의 시간이 걸린다.

따라서 전체 시간 복잡도는 $O(n\log n + Q\log Q)$이고, 공간 복잡도는 $O(n\log n + Q)$이다.

Suffix Array를 $O(n\log^2 n)$에 만드는 구현을 사용해도 제한 내에 통과할 수 있다.

문제 K. 갑천 자전거길

출제자 한동규 (queued_q)

1 풀이

자전거를 초기 위치 s_i 기준으로 정렬한 배열을 A라고 하자. A에서 인덱스가 $2K$만큼 떨어진 두 자전거를 생각하자. 먼저 귀류법을 통해 두 자전거가 어떤 시점에도 만나지 않음을 보일 것이다.

두 자전거가 만난다고 가정하자. 그러면 둘 사이에서 출발한 자전거는 두 자전거 중 하나와 반드시 만나야 한다. 사이에 $2K-1$개의 자전거가 존재하므로, 비둘기집의 원리에 따라 두 자전거 중 하나는 적어도 K개의 자전거와 만난다. 두 자전거가 서로 만나는 것까지 고려하면, 하나의 자전거는 적어도 $K+1$개의 자전거와 만나게 되므로 조건에 위배된다. 따라서 두 자전거는 만나지 않는다.

이제 자전거를 $2K$개마다 하나씩 뽑아서 배열 B를 만들자. 즉, $B_i = A_{2Ki}$이다. B 안의 자전거가 서로 만나지 않으므로 어떤 시점에서도 항상 정렬된 상태를 유지한다.

쿼리 (t, x)를 처리하는 방법은 다음과 같다. B에서 이분 탐색을 수행해서, 시각 t에 좌표 x의 바로 왼쪽과 오른쪽에 있는 자전거 B_i와 B_{i+1}을 찾자. 배열 A에서 B_{i-1} 및 그 왼쪽에 있는 자전거는 B_i보다 왼쪽에 있음이 보장되며, B_{i+2} 및 그 오른쪽에 있는 자전거는 B_{i+1}보다 오른쪽에 있음이 보장된다. 따라서 가장 가까운 자전거를 찾으려면 배열 A에서 B_{i-1}과 B_{i+2} 사이의 자전거만 확인하면 된다. 즉, $A_{2K(i-1)+1}$부터 $A_{2K(i+2)-1}$까지의 자전거 중에서, 시각 t에 좌표 x와 가장 가까운 것을 완전 탐색하여 찾으면 답이 된다.

시간복잡도는 초기 정렬에 $O(N \log N)$, 쿼리당 $O(\log N + K)$이다.

조금 더 빠른 풀이

사실은 배열 B를 만들지 않고 배열 A에서 바로 이분 탐색을 수행하는 풀이가 가능하다. 배열 A는 임의의 시각에 대해서 정렬된 상태가 아닌데 어떻게 가능한 것일까?

이분 탐색 알고리즘을 다시 복습해 보자. 우선 A의 양 끝에 $\pm\infty$ 위치에 있는 가상의 자전거가 있다고 생각하여 포인터 l, r을 초기화한다. 중간에 있는 포인터 m을 확인하여 시각 t에 자전거가 x보다 왼쪽에 있으면 포인터 l을 업데이트하고, 오른쪽에 있으면 r을 업데이트한다. 이를 반복하면 l과 r의 거리는 점점 줄어들어 결국 1이 된다.

그런데 이 과정에서 포인터 l은 x보다 왼쪽에 있는 자전거를, r은 x보다 오른쪽에 있는 자전거를 가리킨다는 사실은 항상 변하지 않는다! 즉, 이분 탐색으로 가장 가까운 자전거를 찾지는 못하더라도, 왼쪽에 있는 자전거와 오른쪽에 있는 자전거가 배열에서 인접한 위치를 하나 찾을 수 있다. l과 r 사이의 거리를 1로 좁히고 나면 앞선 풀이와 같은 원리로 A_{l-2K+1}부터 A_{r+2K-1}까지의 자전거만 확인하면 답을 구할 수 있다. 확인해야 하는 자전거의 수가 약 $6K$개에서 $4K$개로 줄어들었다.

참고로, 이 풀이에서 자전거를 $4K$개보다 하나라도 적게 확인하는 구현을 틀리게 하는 입력 데이터가 존재한다. 그러한 입력이 무엇일지는 독자들에게 연습문제로 남긴다.

문제 L. 아침약

출제자 박수빈 (psb0623)

아침약, 점심약, 저녁약을 각각 0, 1, 2로 두자.

남아 있는 약봉지는 항상 처음 배열의 연속 구간이고, 배열은 $0,1,2,0,1,2,\cdots$로 주기가 3이다. 따라서 맨 앞 약봉지의 종류가 x, 맨 뒤 약봉지의 종류가 y인 상태를 (x,y)로 표현할 수 있다. 초기 상태는 $(0,2)$이며, 상태 (x,y)에서 가능한 행동은 다음 두 가지이다.

- 맨 앞의 x를 먹으면 상태는 $((x+1) \bmod 3,\ y)$로 전이된다.
- 맨 뒤의 y를 먹으면 상태는 $(x,\ (y+2) \bmod 3)$으로 전이된다.

따라서 문제는 초기 상태 $(0,2)$에서 시작하여, 문자열 S가 요구하는 숫자를 매 단계 하나씩 먹으면서 이 상태 전이를 끝까지 수행할 수 있는지 판별하는 문제로 볼 수 있다.

이번 단계에 먹어야 하는 숫자를 c라 하고, 두 가지 경우로 나누어 생각하자.

$x \neq y$인 경우, 행동은 유일하게 결정된다.

- $c = x$라면 맨 앞을 먹고 $((x+1) \bmod 3,\ y)$로 전이한다.
- $c = y$라면 맨 뒤를 먹고 $(x,\ (y+2) \bmod 3)$으로 전이한다.
- 둘 다 아니라면 어떤 선택을 해도 불가능하다.

$x = y$인 경우, 먼저 $c \neq x$라면 불가능하다. $c = x$라면 가능한 선택은 다음 두 가지이다.

- 선택 A: 맨 앞의 x를 먹고 $((x+1) \bmod 3,\ x)$로 전이한다.
- 선택 B: 맨 뒤의 x를 먹고 $(x,\ (x+2) \bmod 3)$으로 전이한다.

그러나 바로 다음에 먹어야 하는 숫자 c'까지 고려하면 선택은 다시 유일하게 결정된다.

- $c' = x$인 경우: 어느 쪽을 선택해도 그다음 행동이 강제되어(선택 A 이후에는 맨 뒤만, 선택 B 이후에는 맨 앞만 x이다), 두 단계 뒤의 상태는 $((x+1) \bmod 3,\ (x+2) \bmod 3)$으로 같아진다. 따라서 아무 쪽이나 선택해도 된다.
- $c' = (x+1) \bmod 3$인 경우: 선택 B의 상태 $(x,\ (x+2) \bmod 3)$에서는 c'를 먹을 수 없으므로 반드시 선택 A를 택해야 한다.
- $c' = (x+2) \bmod 3$인 경우: 마찬가지로 선택 A의 상태 $((x+1) \bmod 3,\ x)$에서는 c'를 먹을 수 없으므로 반드시 선택 B를 택해야 한다.
- 마지막 약봉지라서 c'가 존재하지 않는 경우: 남은 약봉지가 하나뿐이므로 두 선택은 같은 봉지를 먹는 같은 행동이다.

따라서 어떤 상태에서도 행동은 항상 유일하게 결정된다. 초기 상태 $(0,2)$에서 시작하여 매 단계 유일하게 정해지는 전이를 따라가며 상태를 갱신하고, 어느 순간 필요한 숫자를 먹을 수 없다면 `No`를, $3N$개의 약을 끝까지 모두 먹을 수 있다면 `Yes`를 출력하면 된다.

각 단계의 처리가 $O(1)$이므로 전체 $O(N)$의 시간복잡도로 문제를 해결할 수 있으며, 이것이 의도한 풀이이다.

문제 M. 악령 퇴치

출제자 배근우 (functionx)

핵심 아이디어

핵심 위치의 좌표가 x일 때 (빨간 퇴마 영역 길이) - (파란 퇴마 영역 길이)를 $f(x)$라고 하자.

이 함수는 매 저주마다, $x \in [X_a, X_b]$에서 정의된다. 해당 함수는 정의역의 모든 범위에서 연속하다.

- $a \le i < b$를 만족하는 정수 i에 대해서,
- i번 주술사와 $i+1$번 주술사가 같은 색이라면 상수 c가 존재하여 $f(x) = c\ (X_i \le x \le X_{i+1})$를 만족한다.
- i번 주술사가 빨간색, $i+1$번 주술사가 파란색이라면 상수 c가 존재하여 $f(x) = 2x + c\ (X_i \le x \le X_{i+1})$를 만족한다.
- i번 주술사가 파란색, $i+1$번 주술사가 빨간색이라면 상수 c가 존재하여 $f(x) = -2x + c\ (X_i \le x \le X_{i+1})$를 만족한다.

따라서 $f(x)$의 최솟값이 m이고 최댓값이 M이라면, 임의의 실수 $m \le y \le M$에 대해서 $f(x') = y$를 만족하는 x'가 존재한다.

즉, m과 M를 계산하면 문제의 정답 $\min(|f(x)|)$를 구할 수 있다.

또한, 이 함수의 극값은 $x = X_i$일 때에만 존재하므로 $\min_{a \le i \le b} f(X_i)$와 $\max_{a \le i \le b} f(X_i)$만 구해도 충분하다.

풀이 - 부분합 세그먼트 트리

부분합 세그먼트 트리(금광 세그)를 사용한다면, 세그먼트 트리의 각 노드 $[l, r]$에 아래 정보를 넣으면 된다.

- $f(l)$
- $f(r)$
- $\min f(x)$
- $\max f(x)$

풀이 - 나이브 세그먼트 트리

$[X_1, X_N]$ 범위에서 정의되는 함수 g, h를 생각하자.

악령이 1번과 N번 주술사에 저주를 건 상태에서, 핵심 위치의 좌표가 x일 때 x 왼쪽의 색깔 차이는 $g(x)$, x 오른쪽의 색깔 차이를 $h(x)$라 하자. $g(X_i)$와 $h(X_i)$는 미리 계산해둔다.

그러면 $f(x) = g(x) + h(x) - g(X_a) - h(X_b)\ (X_a \le x \le X_b)$를 만족한다.

매 저주마다 $g(X_a)$, $h(X_b)$는 상수이므로, $g(X_i) + h(X_i)$를 가지고 RMQ를 처리하면 된다.

두 풀이 모두 시간복잡도는 전처리 $O(N)$, 저주 당 $O(\log N)$이다.

문제 N. 지그재그

출제자 반딧불 (79brue)

이진 문자열 A에 대해, $z(A)$를 다음과 같은 수열로 정의하자.

- $A_i = 1$이면, $z(A)_i = 1$이다.
- $A_i = 0$이면, $z(A)_i = -1$이다.

길이 N의 이진 문자열 A에 대해, $B := z(A)$라 하고, S를 B의 누적 합으로 정의한다. 즉,

$$S_i = \begin{cases} 0 & (i = 0) \\ S_{i-1} + B_i & (1 \le i \le N) \end{cases}$$

이다. 이때 다음이 성립함을 보일 수 있다:

$$f(A) = \max_{0 \le i \le N} S_i - \min_{0 \le i \le N} S_i.$$

증명은 다음과 같다.

(1) $f(A) \ge \max S - \min S$. S에서 $\min S$가 $\max S$보다 먼저 나타난다고 가정하자. $d := \max S - \min S$로 정의한다.

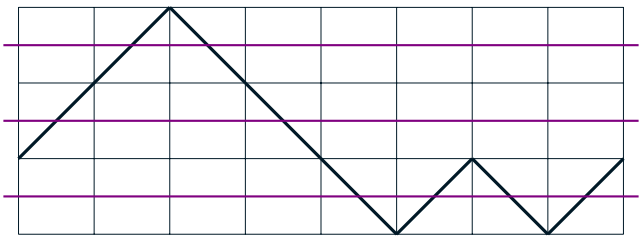
$l := \arg\min S + 1$, $r = \arg\max S$라 하자. 이때 $B[l : r]$에서 1의 개수는 -1의 개수보다 d개 많다. 한편, 하나의 지그재그 수열로는 이 구간에서 이 차이를 최대 1까지만 벌릴 수 있다. 따라서 이 구간을 덮기 위해서 최소 d개의 지그재그 수열이 필요하다.

$\max S$가 $\min S$보다 먼저 등장하는 경우도 같은 방식으로 증명할 수 있다.

(2) $f(A) \le \max S - \min S$. A를 정확히 $d := \max S - \min S$개의 지그재그 수열로 덮을 수 있음을 보인다.

A를 d개의 지그재그 문자열 $S_1, S_2, \ldots, S_d$로 덮을 것이다. 이때, $1 \le i \le n$인 i에 대해, A_i는 $x_i := \max(S_{i-1}, S_i) - \min S$에 대해, S_{x_i}에 포함시킬 것이다.

예를 들어, A가 `11000101`일 때, B는 $[1, 1, -1, -1, -1, 1, -1, 1]$이고, 이때 S를 그림으로 나타내면 다음과 같다.



이때 각 가로선을 보면, 올라가는 대각선 ($A_i = 1$)과 내려가는 대각선 ($A_i = 0$)을 교대로 만날 수밖에 없음을 알 수 있다. 또한 가로선의 개수는 정확히 $\max S - \min S$개임을 알 수 있다. 따라서 위와 같은 방식을 사용하면 항상 A를 $\max S - \min S$개의 지그재그 문자열로 덮을 수 있고, $f(A) \le \max S - \min S$이다.

위의 두 증명에 의해 $f(A) = \max S - \min S$이다. 따라서 문제는 다음을 구하는 것이 된다:

$$\sum_{l=0}^{n-1} \sum_{r=l+1}^{n} \max_{l \le i \le r} S_i - \min_{l \le i \le r} S_i$$

max의 합과 min의 합을 구하는 부분은 대칭적이다. 따라서 max의 합을 구하는 방법을 알아보자.

구간 $[l, r]$에 대해, $[l, r]$의 모든 부분구간 $[s, e]$에 대해 $\max_{s-1 \le i \le e} S_i$의 합을 $g(l, r)$이라 하자. 이때, $[l-1, r]$ 구간에서 S_i의 최댓값이 나타나는 지점 중 하나를 S_x라 하자. 그렇다면, $x \in [s-1, e]$인 모든 구간 $[s, e]$에 대해서 더해지는 합은 x가 될 것이고, 그렇지 않은 구간은 x의 왼쪽과 오른쪽으로 쪼개질 것이다. 따라서, 최댓값이 포함되는 구간의 개수와 최댓값을 곱해 주고, 최댓값을 포함하지 않는 구간들은 왼쪽과 오른쪽으로 나눠 분할 정복을 하면 된다. 각 단계에서 최댓값과 위치를 구하는 것은 Segment Tree를 이용하면 된다.

위와 같은 방식으로 $O(N \log N)$에 문제를 해결할 수 있다.