UCPC 2025

전국 대학생 프로그래밍 대회 동아리 연합
여름 대회 2025

Preliminaries

Official Solutions

예선 해설

전국 대학생 프로그래밍 대회 동아리 연합 · UCPC 2025 출제진

LG전자 · HYUNDAI MOBIS · Jump Trading · FURIOSA · KREW CAPITAL · NEXON · HRT hudson river trading

채널톡 · Jane Street · MOLOCO · codetree · HYUNDAI AutoEver · Samsung Software Membership · STARTLINK · 나정휘 (jhnah917) · 정재헌(gravekper)

문제		의도한 난이도	출제자
A	체육은 수학과목 입니다 2	**Easy**	man_of_learning
B	도미노 게임	**Easy**	wapas
C	빔	**Hard**	kdh9949
D	연산 추가하기	**Medium**	ncy09
E	콘서트	**Medium**	akim9905
F	시장조성하기	**Hard**	golazcc83
G	스마트 창고	**Hard**	79brue
H	해안선	**Hard**	isekaijoucho
I	로봇 청소기	**Medium**	bubbler
J	소어그래프	**Hard**	_junick
K	Distance Multiplication Maximization	**Challenging**	molamola

A. 체육은 수학과목 입니다 2

UCPC 2025

`arithmetic`

출제진 의도 – **Easy**

- 제출 216번, 정답 178팀 (정답률 82.407%)
- 처음 푼 팀: **우주최강엄종노** (뚱뚱한엄종, 광교산호랑이, 나쁜말은노노), 0분
- 가장 짧은 정해: 207B^{C++}, 69B^{Python}
- 출제자: `man_of_learning`

A. 체육은 수학과목 입니다 2

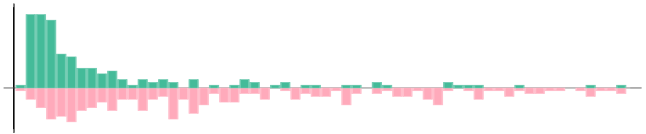
- 운동장 4바퀴를 달린 시간이 주어지고, 남은 한 바퀴를 달린 시간을 300초라고 할 때, 총 5바퀴를 달린 시간의 총합이 1800초 이하인지를 판별하는 문제입니다.
- $t_1 + t_2 + t_3 + t_4 \leq 1500$이라면 "Yes", 그렇지 않다면 "No"를 출력합니다.

B. 도미노 게임

UCPC 2025

`greedy`, `ad_hoc`
출제진 의도 – **Easy**

- 제출 384번, 정답 167팀 (정답률 43.490%)
- 처음 푼 팀: **결속 밴드** (고토 후타리, 히로이 키쿠리, 이지치 세이카), 2분
- 가장 짧은 정해: $348B^{C++}$, $171B^{Python}$
- 출제자: `wapas`



B. 도미노 게임

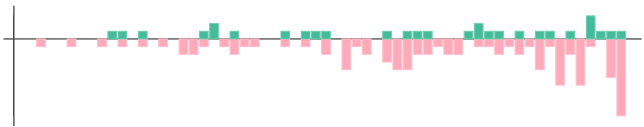
- 차례의 수를 최대한 길게 하려면 도미노 범위 중 한 칸은 0이 되게 행동하는 것을 최대화하는 게 최선의 전략입니다.
- 따라서 0인 칸이 존재한다면, 0인 칸을 포함하여 1개의 수를 계속 빼는 형식으로 최대한 오래 게임을 할 수 있습니다. 이 경우 답은 (보드에 적힌 모든 수의 합)이 됩니다.
- 만약 0인 칸이 존재하지 않는다면 강제로 0인 칸을 만들어야 하므로, 가장 작은 값을 찾아 0으로 만듭니다.
- 그러므로 최종 답은 (보드에 적힌 모든 수의 합) – (보드에 적힌 수 중 최솟값)이 됩니다.

C. 빔

`sweeping`

출제진 의도 – **Hard**

- 제출 120번, 정답 30팀 (정답률 25.000%)
- 처음 푼 팀: **팀명못정함** (imeimi2000, numbering, blackking26), 28분
- 가장 짧은 정해: 1,589B^{C++}, 1,653B^{Python}
- 출제자: kdh9949



- 구간이 $[l, r]$ 이고 빔이 $[s, e]$ 일 때 구간을 치우는 비용에 대해 살펴봅시다.
- 왼쪽으로 치우는 비용: $(r - l) \cdot \max(0, r - s)$
- 오른쪽으로 치우는 비용: $(r - l) \cdot \max(0, e - l)$
- 두 식을 비교하면 "$l + r \leq s + e$면 왼쪽, 아니면 오른쪽으로 치우면 된다" 라는 결론을 얻습니다.

- 쿼리마다, 왼쪽으로 치워야 하는 비용의 합을 우선 구해 봅시다.
- $l + r \le s + e$인 구간들에 대해 $(r - l) \cdot \max(0, r - s)$ 의 합을 구하면 됩니다.
- 2차원 평면이 있고, 각 구간 $[l, r]$ 이 $x \ge l + r$, $y \le r$ 범위에 $(r - l) \cdot (r - y)$ 만큼 가중치를 더한다고 합시다.
- $[s, e]$ 빔에 대해서 $(s + e, s)$ 위치의 가중치 합을 구하면 됩니다.

- $x \geq l + r$, $y \leq r$ 범위에 $(r - l) \cdot (r - y)$, 즉 y에 대한 일차식을 더하는 꼴입니다.
- 구간에 일차식을 더하고, 점의 값을 구하는 것을 빠르게 하는 자료구조가 있다고 합시다.
- 그렇다면, Plane sweeping을 사용할 수 있습니다.
  - $l + r$ 순서대로 구간을 정렬하여 하나씩 자료구조에 추가하고 ($[1, r]$ 구간에 $-(r - l)y + (r - l)r$ 을 더하기),
  - 각 빔마다 적절한 시점에 s 에 적힌 값을 구하면 됩니다.
- 왼쪽으로 치우는 케이스 기준으로 식을 설명하였는데, 오른쪽도 비슷하게 하면 됩니다.

- 이제 일차식 자료구조를 구현하는 것만 남았습니다.
- 그런 자료구조는 단순히 "구간에 값 더하기 + 점 쿼리" 를 지원하는 자료 구조 (Segment Tree 또는 Fenwick Tree) 2개로 구현할 수 있습니다.
- $Ax + B$에서 A와 B의 합을 각각 관리하면 됩니다.
- 최대 좌표가 10^6으로 그렇게 크지 않으므로, 좌표 압축 등을 추가로 수행하지 않고 $\mathcal{O}\left((N+Q)\log 10^6\right)$ 정도에 구현이 됩니다.

– 자료구조의 구현 예시입니다.

```
const int sz = 1000005;
using ll = long long;
struct Fenwick {
    ll d[sz];
    void u(int x, ll v) { for(; x < sz; x += x & -x) d[x] += v; }
    ll g(int x) { ll r = 0; for(; x; x &= x - 1) r += d[x]; return r; }
};
struct LinearFenwick {
    Fenwick A, B;
    // [s, e] 구간에 ax+b 더하기
    void u(int s, int e, ll a, ll b) {
        A.u(s, a); B.u(s, b);
        A.u(e + 1, -a); B.u(e + 1, -b);
    }
    // x 위치에 적힌 값 구하기
    ll g(int x) {
        return A.g(x) * x + B.g(x);
    }
} L, R;
```

D. 연산 추가하기

UCPC 2025

`sorting`, `sweeping`, `prefix_sum`

출제진 의도 – **Medium**

- 제출 451번, 정답 112팀 (정답률 24.834%)
- 처음 푼 팀: **공치군** (공대, 치대, 군대), 13분
- 가장 짧은 정해: 1,015B^{C++}, 880B^{Python}
- 출제자: `ncy09`

- 가장 간단한 경우인, 기존 연산이 주어지지 않은 경우($N = 0$)를 생각해봅시다.
- 이 경우, 단순히 $1 \le S \le S + T - 1 \le H$를 만족하는 정수 S의 개수를 구하는 문제가 됩니다.
- 이는 정확히 $H - T + 1$개입니다.

- 본 문제를 살펴봅시다. 각각의 기존 연산이 실행되는 명령 사이클은 구간의 형태로 주어집니다.
- 이들의 합집합을 생각해보면, 마찬가지로 서로 교차하지 않는 여러 개의 구간들로 표현될 것입니다.
- 추가 연산은 기존 연산들과 겹치지 않아야 하므로, 결국 합집합을 제외한 영역에 삽입되어야 합니다.

- 이러한 영역들은 여러 개의 구간들로 표현됩니다.
- 각각의 구간에 추가 연산을 삽입하는 경우의 수는 $N = 0$인 경우와 동일하게 계산할 수 있습니다.
- 구체적으로, 구간 $[a, b]$에 추가 연산을 삽입하는 경우의 수는 $(b - a + 1) - T + 1$입니다.
- 이때, 구간의 길이가 추가 연산보다 짧을 경우 추가 연산을 삽입할 수 없습니다.
- 각 구간 별 경우의 수를 전부 더한 값이 문제에서 요구하는 정답이 됩니다.

D. 연산 추가하기

- 구현 방법은 다음과 같습니다.
- 먼저 기존 연산들에 대한 정보를 정렬합니다. 간단한 스위핑을 통해, 이들의 합집합을 구할 수 있습니다.
- 이로부터, 합집합을 제외한 영역들을 구성하는 구간들 또한 계산할 수 있습니다.
- 이렇게 계산된 구간들을 $[a_i, b_i]\,(1 \le i \le m)$ 이라 할 때, 정답은

$$\sum_{i=1}^{m} \max\left\{(b_i - a_i + 1) - T + 1, 0\right\}$$

  입니다.
- 그러나 이를 단순히 더하면 $\mathcal{O}(QN)$ 으로 시간초과를 받습니다.

- 결국 중요한 것은 구간의 길이입니다. 구간의 길이 $l_i = b_i - a_i + 1$을 정의하면, 정답은

$$\sum_{l_i \geq T-1} (l_i - T + 1)$$

  입니다.
- 구간들의 길이를 저장하고 있는 배열을 만들고 이를 정렬합니다. 해당 배열의 누적합을 이용하면 식을 빠르게 계산할 수 있습니다.
- 시간복잡도는 $\mathcal{O}\left((N+Q)\log N\right)$입니다.

E. 콘서트

`simulation`, `ad_hoc`

출제진 의도 – **Medium**

- 제출 583번, 정답 106팀 (정답률 18.182%)
- 처음 푼 팀: **도박중독** (포커, 마작, 코인), 5분
- 가장 짧은 정해: 729B^{C++}, 610B^{Python}
- 출제자: `akim9905`

- 임의의 위치 $c-1$와 c 사이에서 소음 x의 콘서트가 발생했다고 가정합시다.
- 양방향에 대해서 같은 작업을 하므로, 소음이 오른쪽으로만 퍼지는 상황만 생각합시다.
- 현재 c번 방음벽에 대하여, 다음의 두 가지 경우가 존재합니다.

1. $D_c < x$
   콘서트가 종료된 뒤 D_c의 값은 2배가 됩니다. 흡수되지 못한 여분의 소음은 $c+1$번으로 넘어갑니다.
2. $D_c \geq x$
   모든 소음이 c번 방음벽을 통해 흡수됩니다.

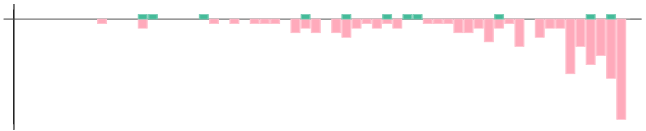
- $x \le 10^9$ 이므로, 각 방음벽이 2 배씩 커지는 상황이 최대 $\log 10^9$ 번 일어남을 알 수 있습니다.
- 따라서 모든 방음벽이 보강을 하게 되는 횟수는 $\mathcal{O}\left(N \log 10^9\right)$ 입니다.
- 그 이후로 발생하는 소음은 오직 양 옆의 방음벽에서만 처리됩니다.
- 따라서 각 쿼리를 Naïve하게 처리하면 됩니다.
- 시간 복잡도는 $\mathcal{O}\left(Q + N \log 10^9\right)$ 입니다.

F. 시장조성하기

`dp`, `sorting`, `data_structure`

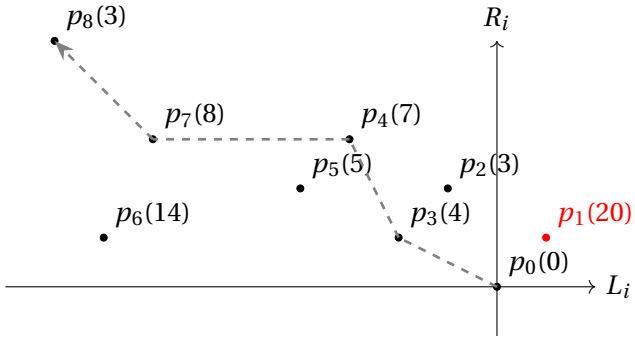
출제진 의도 – **Hard**

- 제출 146번, 정답 11팀 (정답률 7.534%)
- 처음 푼 팀: **Fox is cute** (Fox, is, cute), 37분
- 가장 짧은 정해: 991B^{C++}, 1,358B^{Python}
- 출제자: `golazcc83`

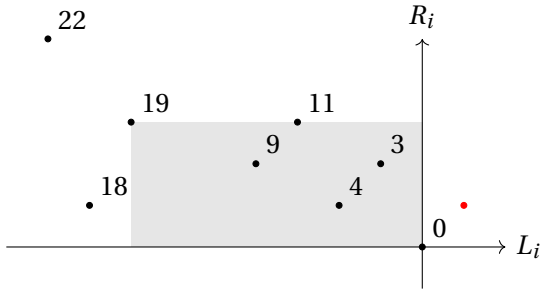


- 첫 번째부터 i 번째 매매 시점까지 a_i, b_i 의 누적 합을 각각 L_i, R_i 라고 했을 때, i 번째 매매 시 보유할 수 있는 주식의 수는 L_i 이상 R_i 이하의 정수 중 하나입니다.
- 이 때 $L_i \le 0 \le R_i$ 라면 i 번째 매매 이후 보유 주식의 수를 0 으로 만들 수 있습니다.
- i 번째 매매 이후 보유 주식이 0 주라면, $i \le j$ 인 j 번째 매매 시 보유할 수 있는 주식의 수는 $L_j - L_i$ 이상 $R_j - R_i$ 이하의 정수 중 하나입니다.
- 이 때 $L_j - L_i \le 0 \le R_j - R_i$ 라면 i, j 번째 매매 시점에 보유 주식의 수를 0 으로 만들 수 있습니다.
- 즉 L_i 가 감소하면서 R_i 가 증가하도록 매매 시점들을 선택하여 그 안정성의 합을 최대화시키는 것이 목표입니다.

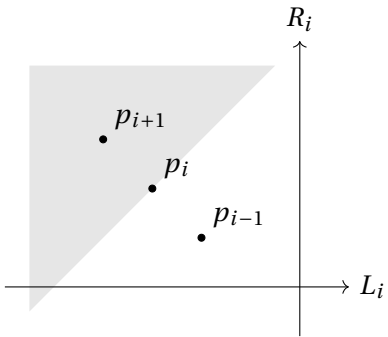
- L_i 를 X 축으로, R_i 를 Y 축으로 하는 2차원 좌표가 있다고 가정합시다. i 번째 매매 후 시점과 이 때 얻을 수 있는 안정성을 $p_i(x_i)$ 라고 할 때, 아래의 그림처럼 L_i 가 감소, R_i 가 증가하는 순으로 점을 선택하는 문제로 바꿀 수 있습니다.



- $D[i]$: i 번째 매매 시점에 주식의 보유 수가 0일 때 얻을 수 있는 안정성의 최댓값일 때,
- $D[i] = \max(D[k] + x_i \mid k < i, L_i \le L_k \le 0, 0 \le R_k \le R_i)$, $L_i > 0$ 또는 $R_i < 0$일 때는 계산하지 않습니다.
- 2D Segment Tree를 이용하면 $\mathcal{O}\left(N \log^2 N\right)$의 시간 복잡도로 해결할 수 있어 보이지만, N의 최댓값이 1 000 000이므로 쉽지 않아보입니다.



– 문제의 조건에 따라 $a_i \le b_i$ 이므로 $i < j$ 일 때 $R_i - L_i \le R_j - L_j$ 를 만족합니다. 따라서 점을 선택할 때 L_i 가 감소, R_i 가 증가하는 순으로 점을 선택하면 매매 시점을 고려하지 않아도 됨을 관찰할 수 있습니다.

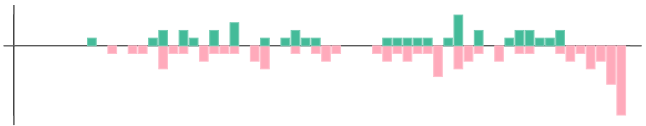


- 앞에서 관찰한대로 L_i 가 감소, R_i 가 증가하는 순으로 매매 시점을 정렬합니다. 정렬된 매매 시점의 R, x를 각각 R', x'라고 하면 L을 고려하지 않아도 되므로 DP 식을 아래와 같이 바꿀 수 있습니다.
- $D[i] = \max\left(D[k] + x'_i \,\middle|\, k < i, 0 \le R_k \le R_i\right)$
- 위 식은 R'의 감소하지 않는 부분수열의 가중치의 최댓값을 구하는 문제와 동일합니다. 이는 Set 또는 Segment Tree를 활용하여 $\mathcal{O}\left(N \log N\right)$에 해결할 수 있습니다.

G. 스마트 창고

`dp`, `prefix_sum`
출제진 의도 – **Hard**

- 제출 108번, 정답 39팀 (정답률 36.111%)
- 처음 푼 팀: **도박중독** (포커, 마작, 코인), 23분
- 가장 짧은 정해: 1,233B^{C++}, 1,012B^{Python}
- 출제자: `79brue`



- 문제에서 가능한 직사각형 영역은 총 $\mathcal{O}\left(N^2M^2\right)$ 개가 있습니다.
- $N, M \leq 500$ 이기 때문에, 가능한 모든 직사각형 영역을 순회하는 것만으로도 시간 초과를 받을 수 있습니다.
- 그러나, 이 문제에서 실제로 유의미한 직사각형 영역의 개수를 더 줄일 수 있습니다.

- 직사각형의 좌우 범위 $[c_1, c_2]$를 고정합니다.
- 이때 가능한 직사각형의 개수는, 위쪽 끝변과 아래쪽 끝변을 자유롭게 정할 수 있으므로, 총 $\mathcal{O}\left(N^2\right)$가 존재합니다.
- 이 수를 $\mathcal{O}\left(N\right)$으로 줄여 봅시다.

- 모든 행 r에 대해, $[r, r] \times [c_1, c_2]$ 범위의 구간 합을 구해 B_r이라고 합시다.
- 이 값은 사전에 2차원 누적 합을 구해 두면 하나의 r 값에 대해 $\mathcal{O}(1)$에 구할 수 있습니다.
- 이제 직사각형 $[r_1, r_2] \times [c_1, c_2]$ 범위의 구간 합은 $\sum_{r=r_1}^{r_2} B_r$과 같습니다.

- 문제에서 요구하는 것은 특정 칸 (r, c)에 대해, 해당 칸을 포함하는 직사각형의 구간 합의 최댓값입니다.
- $c_1 \le c \le c_2$인 경우, 직사각형의 좌우 끝점이 $[c_1, c_2]$인 경우만 생각한다면, 답이 될 수 있는 후보는 B에서 원소 B_r을 포함하는 구간 합의 최댓값뿐입니다.
- 따라서 배열 B의 각 원소 B_i에 대해, 해당 원소를 포함하는 구간 합의 최댓값을 구해야 합니다.
- 누적 합 알고리즘과 DP를 응용하면 $\mathcal{O}(N)$에 간단히 구할 수 있습니다.
- 이것을 모든 (c_1, c_2) 쌍에 대해 구하려면 $\mathcal{O}(NM^2)$의 시간이 소요됩니다.

- 지금까지 알아낸 정보는 다음과 같습니다:
  - 모든 $1 \le r \le N, 1 \le c_1 \le c_2 \le M$에 대해,
  - 좌우 범위가 $[c_1, c_2]$이고, 상하 범위에 r 행이 포함되는 직사각형의 구간 합의 최댓값
  - 이 값을 $V[r][c_1][c_2]$라고 정의합니다.
- 이제 이 정보를 통해 원래 문제를 풀어 봅시다.

- 이제 행별로 생각합니다.
- (r, c)를 포함하는 직사각형의 구간 합의 최댓값은 무엇일까요?
- (r, c)를 포함하면서, 구간 합의 최대가 되는 직사각형이 $[r_1, r_2] \times [c_1, c_2]$라고 합시다.
- 이때, $c_1 \le c \le c_2$가 성립해야 하며, 해당 직사각형의 구간 합은 V의 정의에 의해 $V[r][c_1][c_2]$와 같아야 합니다.
- 따라서, 칸 (r, c)에 대한 답은, $1 \le c_1 \le c \le c_2 \le M$인 모든 (c_1, c_2)에 대해 $V[r][c_1][c_2]$의 최댓값과 같습니다.

- 다음과 같은 DP를 생각해 봅시다. (r 행에 해당하는 V 값만 생각해도 충분하므로, 잠시 r 을 생략합니다.)
- $D[c_1][c_2]$: $[c_1, c_2] \subset [c_1', c_2']$ 인 모든 열 구간 $[c_1', c_2']$ 에 대해, $V[c_1'][c_2']$ 의 최댓값으로 정의합니다.
- 이때 칸 (r, c) 에 대한 답은 $D[c][c]$ 가 됩니다.

$$D[c_1][c_2] = \begin{cases} -\infty & (c_1 < 1 \text{ 또는 } c_2 > M) \\ \max(V[c_1][c_2], D[c_1-1][c_2], D[c_1][c_2+1]) & (1 \le c_1 \le c_2 \le M) \end{cases}$$

- 위와 같은 점화식으로 계산하면 모든 $D[c_1][c_2]$를 행별로 $O(M^2)$의 시간에 구할 수 있습니다.
- 따라서 전체 문제를 $\mathcal{O}\left(NM^2\right)$에 해결할 수 있습니다.

- 이밖에도 행별로 덱을 이용해 구간 최댓값을 빠르게 업데이트하는 방법이나, 루트 버킷을 이용해 $\mathcal{O}\left(NM^{2.5}\right)$에 해결하는 풀이가 있습니다.
- $\mathcal{O}\left(N^2M^2\right)$ 시간복잡도의 풀이와 정해를 구분하기 위해, 시간 제한이 작게 설정되어 있습니다.
- 시간 복잡도가 충분히 빠르더라도, 덱을 이용하는 등 상수가 큰 경우 시간 초과를 받을 수 있습니다.

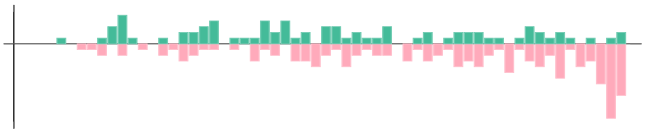
H. 해안선

UCPC 2025

`combinatorics`, `ad_hoc`

출제진 의도 – **Hard**

- 제출 209번, 정답 79팀 (정답률 37.799%)
- 처음 푼 팀: **인공지능 사용 적발 시 실격 처리 됩니다** (Yann LeCun, Yoshua Bengio, Geoffrey Hinton), 14분
- 가장 짧은 정해: 835B$^{\text{C++}}$, 662B$^{\text{Python}}$
- 출제자: `isekaijoucho`



- 문제의 정의에 의해, 해안선 위의 도시와 도로는 완전 그래프 K_N을 이루고 있습니다.
- 완전 그래프 K_N 위에서 정의된 해밀턴 경로 중, 어떤 두 간선도 교차하지 않는 경로를 비교차 해밀턴 경로라고 부르겠습니다.
- 문제에서 찾아야 하는 경로는 비교차 해밀턴 경로 중, 간선 (a, b)를 지나는 경로입니다.
- 이제, 문제를 해결하기 위해 몇 가지 관찰이 필요합니다.

- 비교차 해밀턴 경로는 시작 정점이 1로 고정되어 있으며, 종료 정점은 $2, 3, \ldots, N$ 중 하나가 될 수 있습니다.
- 따라서 전체 비교차 해밀턴 경로의 수는, 시작 정점이 1이고 종료 정점이 $x\,(2 \le x \le N)$인 비교차 해밀턴 경로의 수들의 합으로 정의할 수 있습니다.
- 반드시 포함해야 하는 간선 (a, b)의 두 정점 중 하나를 종료 정점으로 하고 1번 정점을 시작 정점으로 하는 비교차 해밀턴 경로의 수와, 간선 (a, b)의 나머지 정점을 시작 정점으로 하는 비교차 해밀턴 경로의 수를 각각 구할 수 있다면, 문제의 답을 계산할 수 있습니다.

- 먼저, K_N의 시작 정점에서 시작하여 x번 정점 $(2 \le x \le N)$에서 종료되는 비교차 해밀턴 경로의 수를 구해보겠습니다.
- 처음 시작 정점은 1번 정점이고, 1번 정점과 x번 정점 사이에 끼어 있는 나머지 정점들을 두 개의 호로 영역을 나누면, 각 호에 있는 정점의 개수는 $i = x - 2$와 $j = N - x$로 나눌 수 있습니다.
- 비교차 해밀턴 경로를 구성하기 위해 1번 정점과 인접한 두 정점 중 하나를 선택하여, 1번 정점과 간선으로 잇습니다.
- 다시, 선택한 정점을 시작 정점으로 보고, 시작 정점이 x가 될 때까지 이 과정을 재귀적으로 반복합니다.

- f를 **두 개의 호로 나누어진 영역의 정점의 개수가 i, j일 때, 비교차 해밀턴 경로의 수**로 정의하면 다음과 같은 점화식을 얻을 수 있습니다.

$$f(i,j) = f(i-1,j) + f(i,j-1), \quad f(0,j) = f(i,0) = 1$$

- 이 점화식의 해는 다음과 같습니다.

$$f(i,j) = \binom{i+j}{i}$$

- 따라서 K_N의 시작 정점에서 시작하여 x번 정점에서 종료되는 비교차 해밀턴 경로의 수는

$$f(x-2, N-x) = \binom{N-2}{x-2}$$

- 이제, K_N의 전체 비교차 해밀턴 경로의 수를 구해보도록 하겠습니다.
- 종료 정점이 2부터 N까지인 경우를 모두 더하면 다음과 같습니다.

$$\sum_{x=2}^{N} \binom{N-2}{x-2} = 2^{N-2}$$

- 지금까지 계산한 식을 이용하여, $a < b$인 간선 (a, b)를 포함하는 비교차 해밀턴 경로의 수를 구해보겠습니다.
- 이때, $|a - b| \neq 1$인 경우와 $|a - b| = 1$인 경우로 나누어 계산하겠습니다.

- 먼저, $|a-b| \neq 1$ 인 경우에 대해 계산하겠습니다.
- a번 정점과 b번 정점 사이에 끼어 있는 나머지 정점들을 두 개의 호로 영역을 나눌 수 있습니다.
- 두 영역 중, 1번 정점을 포함하는 영역을 생각해보겠습니다.
- 이 영역에서는 시작 정점이 1로, 종료 정점이 a 또는 b로 정해집니다.
- 또한, 종료 정점이 a인 경우와 b인 경우의 경로는 서로 대칭적 이라는 것을 알 수 있습니다.

- 따라서 정점의 개수가 $N-b+a$인 완전 그래프의 비교차 해밀턴 경로가 시작 정점과 종료 정점이 결정 되어있는 상태라고 할 수 있습니다.
- 대칭인 경로까지 고려했을 때, 경로의 수는

$$2 \cdot \binom{N-b+a-2}{a-2}$$

- 1 번 정점을 포함하지 않는 나머지 영역에 대하여, 정점의 개수가 $b-a$인 완전 그래프의 전체 비교차 해밀턴 경로의 수를 구하면 됩니다.

- 이때, 경로의 수는

$$2^{b-a-2}$$

- 따라서 $|a-b| \neq 1$인 경우, 간선 (a, b)를 지나는 비교차 해밀턴 경로의 수는 다음과 같습니다.

$$\binom{N-b+a-2}{a-2} \cdot 2^{b-a-1}, \quad \text{if } |a-b| \neq 1$$

- 이제, $|a-b|=1$ 인 경우에 대해 계산하겠습니다.
- 두 정점 a와 b를 하나의 정점으로 간주했을 때, K_{N-1} 위에서의 비교차 해밀턴 경로의 수는

$$2^{N-3}$$

- 이 경우, 간선 (a,b)를 반드시 지나는 것이 보장되지만, 한 가지 방향만 계산에 포함되고 있습니다.

- 이때, 시작 정점이 1이고 종료 정점이 a 또는 b인 경로의 수는

$$\binom{N-b+a-2}{a-2}$$

- 따라서 $|a-b|=1$인 경우, 간선 (a,b)를 지나는 비교차 해밀턴 경로의 수는 다음과 같습니다.

$$\binom{N-b+a-2}{a-2}+2^{N-3}$$

- 경로의 수는 다음과 같습니다:

$$\text{경로의 수} = \begin{cases} \binom{N-b+a-2}{a-2} \cdot 2^{b-a-1} & (|a-b| \neq 1) \\ \binom{N-b+a-2}{a-2} + 2^{N-3} & (|a-b| = 1) \end{cases}$$

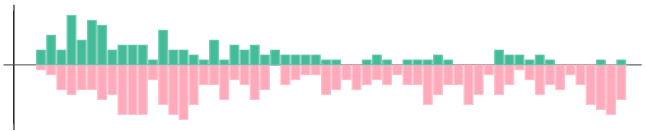
- 모듈러 역원을 이용하여 팩토리얼 전처리를 $\mathcal{O}\left(N + \log \text{MOD}\right)$에 하고, 문제에서 요구하는 답을 $\mathcal{O}(1)$에 처리할 수 있습니다.
- $|a-b| = 1$인 경우, $\mathcal{O}(N)$에 처리하더라도 제한 시간 내에 통과할 수 있습니다.

I. 로봇 청소기

`greedy`, `flood_fill`

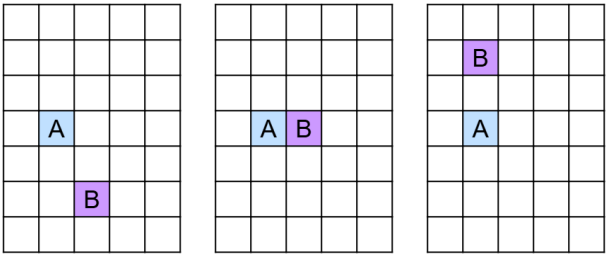
출제진 의도 – **Medium**

- 제출 432번, 정답 131팀 (정답률 30.324%)
- 처음 푼 팀: **팀명못정함** (imeimi2000, numbering, blackking26), 7분
- 가장 짧은 정해: 742B^{C++}, 578B^{Python}
- 출제자: `bubbler`

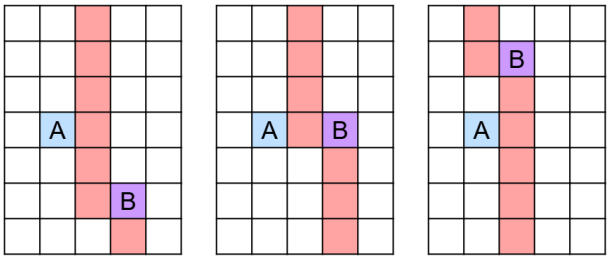


- 먼저, 모든 오염된 칸을 서로 떼어 놓으면 오염 영역의 개수가 N이 됩니다.
- 가능한 x좌표의 범위가 충분히 넓으므로, 이는 모든 칸의 x좌표를 적어도 2씩 떨어뜨려서 배정하는 것으로 달성할 수 있습니다.
- 이보다 많은 오염 영역을 만드는 것은 불가능함을 쉽게 알 수 있으며, 따라서 문제에서 요구하는 답 중 최댓값은 N입니다.

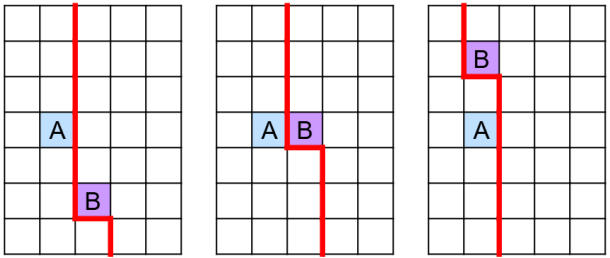
- 이제 최솟값을 구해 봅시다.
- 입력 상에서 이웃한 두 오염된 칸 중 첫 번째 칸을 A, 두 번째 칸을 B라고 합시다.
- 이때, y좌표의 대소 관계에 따라 B가 위치할 수 있는 x좌표가 가장 작은 칸은 아래와 같습니다.



– 여기서 B를 x좌표가 최소가 아닌 칸으로 움직이면 빨간색으로 칠한 영역에는 오염된 칸이 존재하지 않습니다.



– 이는 아래와 같이 최소인 칸에 배치한 다음 빨간 선을 따라 평면을 잘라버린 것과 동치가 됩니다.



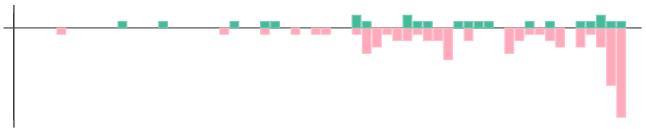
- 따라서, 오염 영역의 개수를 최소화하기 위해서는 모든 칸의 x좌표를 가능한 최소로 배정하는 것이 최적입니다.
- 최악의 경우 좌표의 범위가 매우 넓으므로 일반적인 2D 격자 상의 플러드필 구현은 사용하기 어렵고, 좌표들을 해시맵에 저장하여 그래프를 구성한 후에 그래프 탐색을 하거나, 분리 집합 자료구조 등을 사용하여 답을 구할 수 있습니다.

J. 소어그래프

UCPC 2025

`case_work`, `graph_theory`, `bfs`
출제진 의도 – **Hard**

- 제출 98번, 정답 24팀 (정답률 24.490%)
- 처음 푼 팀: **와샌즈아시는구나정말어렵습니다혹시모르시는분들에대해설명해드** (와, 샌즈, 아시는구나), 31분
- 가장 짧은 정해: 1,287B^{C++}, 1,153B^{Python}
- 출제자: `_junick`



- $N \le 10^{18}$ 이기 때문에 정점 x에서 y로 직접 경로를 찾는 $\mathcal{O}(N)$ 풀이는 불가능합니다.
- 문제에서 가장 눈에 띄는 건 t의 범위입니다.
- $t < 2^{20}$ 라는 작은 범위를 이용해 봅시다.

- 우선 문제를 다시 해석해 봅시다.
- 한 번의 연산에서는 $x \oplus t$ 또는 $(x \oplus t) + 1$을 적용할 수 있습니다.
- 이러한 연산을 반복하여 x를 y로 만들 수 있는지 판단하고, 가능하다면 최소 몇 번의 연산이 필요한지를 구해야 합니다.
- 이때 x는 어느 순간에서도 $N - 1$을 초과하면 안됩니다.

- 먼저 $t = 0$인 경우부터 살펴봅시다.
- 만약 $t = 0$이라면 $i \oplus 0 = i$이므로, 유일하게 i를 변화시킬 수 있는 연산은 $(i \oplus 0) + 1 = i + 1$ 뿐입니다. 즉, 한 번에 i를 1씩 증가시킬 수 있습니다.
- $y > x$라면 단순히 $y - x$번 연산을 반복하면 되고, $y < x$라면 도달할 수 없어 -1을 출력하면 됩니다.
- $\mathcal{O}(1)$에 답을 구할 수 있습니다.

- 이제 $t > 0$인 일반적인 경우를 살펴봅시다.
- t는 2^{20} 미만의 정수이므로 $i \oplus t$는 많아야 하위 20비트에만 영향을 줍니다.
- 편의상 $t < 2^m$을 만족하는 가장 작은 음이 아닌 정수를 m이라고 하겠습니다.
- 그리고 어떤 정수 i에 대해 $i = A_i \times 2^m + B_i$로 표현해 보겠습니다.
- 여기서 A_i는 상위 비트를 나타내고, B_i는 하위 m개의 비트를 나타냅니다.

- 각 연산이 어떤 역할을 하는지 알아봅시다.
- $i \oplus t$는 하위 m비트에만 영향을 주기 때문에, A_i는 변하지 않고 B_i만 $B \oplus t$로 바뀝니다.
- $(i \oplus t) + 1$은 $i \oplus t$와 유사하지만 한 가지 중요한 차이점이 있습니다.
- 대부분의 경우 A_i는 유지되며, B_i는 $(B_i \oplus t) + 1$로 바뀝니다.
- 단, $B_i \oplus t = 2^m - 1$일 경우 $(B_i \oplus t) + 1 = 2^m$이 되어 받아올림이 발생합니다. 이때 A_i는 1 증가하고 B_i는 0이 됩니다.
- 이 경우만이 A를 증가시킬 수 있는 **유일한 방법**입니다. A를 줄이는 것은 불가능합니다.

- $i \oplus t$는 A_i를 건드리지 않고, $(i \oplus t) + 1$는 A_i를 증가시키기만 할 수 있습니다.
- 따라서 x에서 y로 이동하려면 $A_x \le A_y$를 만족해야 합니다.
- 만약 $A_x > A_y$라면 어떤 연산을 해도 y에 도달할 수 없으므로 -1을 출력해주면 됩니다.

- $A_x \le A_y$ 라면 받아올림은 $A_y - A_x$ 번 발생합니다. 이 수를 D 라고 정의하겠습니다.
- 받아올림 이외의 연산은 모두 하위 비트 B 들만 놓고 생각할 수 있습니다.
- 정점이 0 부터 $2^m - 1$ 까지 있는 유향 그래프 G 를 만들어 봅시다. 이 그래프는 하위 비트 전이만 고려하는 그래프입니다.
- 각 정점 u 에서는 항상 $u \oplus t$ 로 가는 간선을 추가합니다.
- 만약 $u \oplus t \neq 2^m - 1$ 이라면 추가로 $(u \oplus t) + 1$ 로 가는 간선도 넣어줍니다.
- 이렇게 하면 하위 비트만으로 가능한 모든 이동을 표현할 수 있습니다.

- 이 그래프 G에서 정점 0, 출발점 B_x, 도착점 B_y, 그리고 받아올림이 일어나는 정점 $S = (2^m - 1) \oplus t$ 사이의 최단 경로를 구해줍니다.
- 편의상 정점 p부터 q까지의 최단 거리를 $\mathrm{dist}(p, q)$라고 정의하겠습니다.

- 일반적인 경우, 즉 $A_x < A_y$ 일 때는 총 $D = A_y - A_x$ 번의 받아올림이 발생합니다.
- 받아올림은 $S \to 0$ 으로 이동할 때 발생하므로, 매번 S에서 0으로 넘어가면서 A의 값을 하나씩 증가시키게 됩니다.
- 따라서 전체 경로는 다음과 같이 구성됩니다.
  - 먼저 B_x에서 출발해 S까지 이동합니다. 이때는 $\mathrm{dist}(B_x, S)$ 만큼의 연산이 필요하고, 받아올림을 위해 추가로 1 번의 연산이 더 필요합니다.
  - 이후 $D-1$ 번의 받아올림을 반복합니다. 각각 $0 \to S$까지 이동한 후 다시 받아올림을 수행해야 하므로, 총 $(D-1)(\mathrm{dist}(0, S)+1)$ 번의 연산이 발생합니다.
  - 마지막으로 받아올림을 마친 후 $0 \to B_y$ 로 도착하면 됩니다. 이는 $\mathrm{dist}(0, B_y)$ 번의 연산을 반복하면 됩니다.

- 따라서 $A_x < A_y$ 인 경우에 대해 전체 연산 횟수는 다음과 같습니다.

$$\mathrm{dist}(B_x, S) + 1 + (D-1)(\mathrm{dist}(0, S) + 1) + \mathrm{dist}(0, B_y)$$

- 다만, $A_y = A_N$ 의 경우 조금 다르게 처리해주어야 합니다.
- 이 경우 마지막 받아올림 이후 도착점 B_y 로 이동 가능한지를 확인해야 하므로, 전체 정점 수를 B_N 으로 제한한 새로운 그래프 G' 를 구성하여 이 위에서 $\mathrm{dist}'(0, B_y)$ 를 다시 계산해주어야 합니다.
- 이때 G' 상에서 $0 \to B_y$ 경로가 존재하지 않는다면, 문제 조건을 만족하는 경로가 없다는 의미이므로 -1 을 출력해주면 됩니다.

- $A_x = A_y$ 인 경우에는 받아올림이 일어나지 않으므로 $\mathrm{dist}(B_x, B_y)$ 가 답이 됩니다.
- 단, 이때도 $A_y = A_N$ 인 경우 제한된 그래프 G' 에서 $\mathrm{dist}'(B_x, B_y)$ 를 계산해야 합니다.
- 만약 해당 경로가 존재하지 않는다면 답은 -1 입니다.
- $A_x \le A_y \ne A_N$ 의 경우 항상 경로가 존재한다는 것을 증명할 수 있습니다.

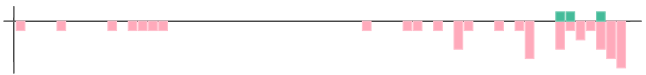
- 각 정점 간 최단 경로는 BFS를 사용하여 $\mathcal{O}(t)$의 시간 복잡도로 구할 수 있습니다.
- 필요한 거리 정보는 많아야 세 번의 BFS로 모두 계산할 수 있으므로, 전체 알고리즘의 시간 복잡도는 $\mathcal{O}(t)$입니다.
- $t \le 2^{20}$이므로, 실제 실행 시간에서도 매우 빠르게 문제를 해결할 수 있습니다.

K. Distance Multiplication Maximization

`tree`

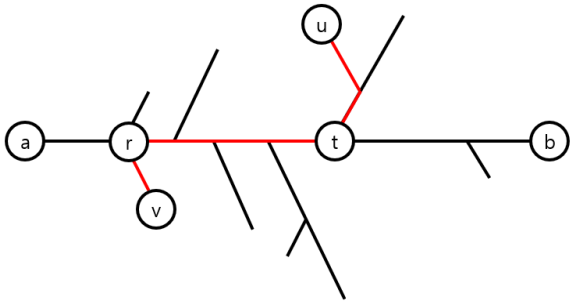
출제진 의도 – **Challenging**

- 제출 44번, 정답 3팀 (정답률 6.818%)
- 처음 푼 팀: **뚝심햄구이** (옥토끼, 레올레, 페페), 159분
- 가장 짧은 정해: 2,173B^{C++}
- 출제자: `molamola`

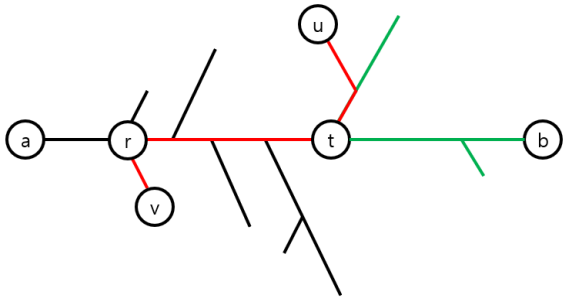


- 쿼리로 u, v가 주어질 때 $\mathrm{dist}(u, x) \times \mathrm{dist}(v, x)$의 최댓값을 구하는 문제입니다.
- 트리의 지름을 하나 찾아서 양 끝점을 a, b라 합시다.
- 꽤 많은 경우 $x = a$ 혹은 $x = b$이 답입니다.
- 이를 더 자세히 분석해 봅시다.

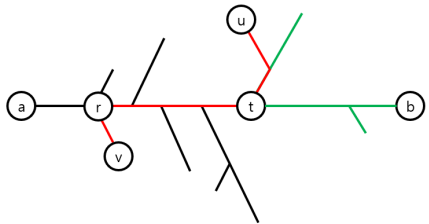
- 아래와 같이 u, v 쿼리가 들어왔다고 가정합니다.
- 지름과 만나는 점을 t, r 이라고 합시다.



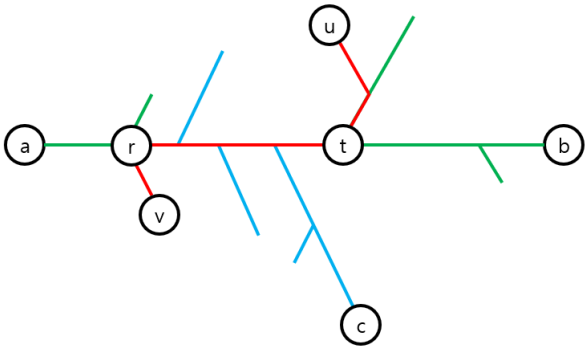
– 초록색 영역의 경우, $x = b$만 고려하면 해결됩니다.



- 이 영역에서 임의의 정점 x에 대해
- $\mathrm{dist}(a,x) \le \mathrm{dist}(a,b)$ 에서 $\mathrm{dist}(t,x) \le \mathrm{dist}(t,b)$ 입니다.
- $\mathrm{dist}(v,x) = \mathrm{dist}(v,t) + \mathrm{dist}(t,x) \le \mathrm{dist}(v,t) + \mathrm{dist}(t,b) = \mathrm{dist}(v,b)$
- 같은 이유로 $\mathrm{dist}(u,x) \le \mathrm{dist}(u,b)$ 또한 성립합니다.



- 양쪽 초록색 영역은 $x = a, x = b$로 고려할 수 있습니다.
- 파란색 부분, 즉 r과 t 사이에서 분기하는 정점들만 잘 고려하면 됩니다.



- $\mathrm{dist}(a, b) = C$라 할 때, 다음과 같이 $X_0, X_1, \cdots, X_C$를 정의합니다.
- X_i: $\mathrm{dist}(a, z) = i$ 인 $a \sim b$ 경로상의 정점 z에 대해, z에서 $a \sim b$ 경로의 간선을 하나도 거치지 않고 갈 수 있는 가장 멀리 떨어진 정점까지 거리
- $\mathrm{dist}(a, r) = l$, $\mathrm{dist}(a, t) = r$ 이라 하면
- $\max_{l<m<r} ((\mathrm{dist}(v, r) + (m - l) + X_m) \times (\mathrm{dist}(u, t) + (r - m) + X_m))$ 을 구하면 됩니다.

- X_m 이 고정이라면 m 은 위로 볼록한 이차함수입니다.
- 이 이차함수는 m 이 $\frac{\mathrm{dist}(u,t)+l+r-\mathrm{dist}(v,r)}{2}$ 일 때 최대입니다.
- 이 지점에 최대한 가까운 m 을 고르면 됩니다.
- 참고로 이 지점은 u 와 v 사이 경로의 중점에 해당합니다.

- X_i 들의 합은 N 보다 작기 때문에, 서로 다른 X_i 들의 개수는 $(2N)^{0.5}$ 이하입니다.
- 가능한 값마다 위 지점에 제일 가까운 m 을 찾으면 됩니다.
- 여기서는 이분 탐색을 사용해도 전체 시간복잡도는 $\mathcal{O}\left(N^{1.5}\log N\right)$ 이 아니라 $\mathcal{O}\left(N^{1.5}\right)$ 입니다.