BCF 2025 에디토리얼

Official Editorial

by

79brue(반딧불)

2025년 8월 16일

문제		출제자	검수진	실제 티어
1	타임위버	R5	D1	R5
I	빛의 길	P3	P4	P3
II	마을 짓기	G3	G3	G2
III	의식의 광장	S4	S1	S2
IV	안정적인 구조	D3	D4	D3
V	기록의 판	P5	P1	P3
VI	소용돌이 문양	P1	P3	P2
VII	힘의 결합	D1	R5	R5
VIII	크로스링크	D5	D4	D5
IX	조화로운 사각형	P4	P4	P4
X	잃어버린 섬	B1	S1	S5
2	빛의 유적	UR	UR	UR

- 난이도 표는 통상적인 에디토리얼과는 다르게 출제자가 의도한 난이도, 검수진이 예측한 난이도, 그리고 2026년 1월 12일 (에디토리얼 업로드 날짜) 기준 실제 solved.ac 티어를 모두 표기하였습니다.
- 이를 통해 출제자, 검수진, 그리고 참가자의 입장에서 대회를 바라보는 다양한 관점에 대해 이해할 수 있을 것으로 기대됩니다.

- 문제의 풀이에 대한 이해를 돕고자 일반적인 에디토리얼보다 훨씬 긴 분량으로 작성하였습니다. (하지만 뒤로 갈 수록 귀찮아졌기 때문에 일부 문제들의 풀이를 충분히 자세히 작성하지 못했습니다. 죄송합니다.)
- 각 서브태스크 섹션 옆에 적힌 점수는 현재까지의 누적 점수입니다.
- 정해와 관련이 없는 일부 서브태스크의 풀이는 생략되어 있습니다.

I. 빛의 길

`set, sweeping`
출제자의 의도 – **P3**
검수진 예측 – **P4**
실제 티어 – **P3**

- ✓ 제출 193번, 정답 67명 (정답률 35.233%)
- ✓ First Solve: **rubikun**, 19분

풀이 요약

- ✓ 구조물이 없다고 생각한 뒤 각 색상의 칸이 얼마나 있는지를 구하고, 이후 구조물을 위부터 하나씩 차례로 적용하며 답을 업데이트해 나갑니다.
- ✓ 인접한 같은 색을 가진 열을 하나의 색상 그룹으로 묶어서 `std::set`을 통해 관리하면 시간 복잡도를 $\mathcal{O}\left((M+K)\log M\right)$으로 줄일 수 있습니다.

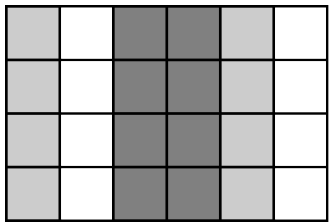
Subtask 1: $K = 0$ (3점)

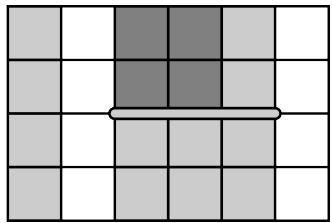
- ✓ 구조물이 하나도 존재하지 않습니다.
- ✓ 즉 각 열의 빛의 색은 항상 일정합니다.
- ✓ 색 x가 격자에서 등장하는 횟수는 색 x가 배열 a에서 등장하는 횟수에 행의 수 N을 곱한 것과 같습니다.
- ✓ Counting Array를 이용해 선형 시간에 해결할 수 있습니다.

Subtask 2: $K = 1$ (11점)

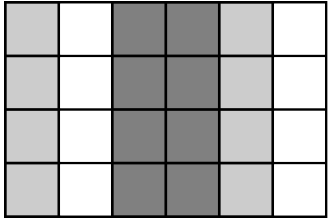
- ✓ 구조물이 단 하나뿐입니다.
- ✓ 접근 방식은 다음과 같습니다.
- ✓ 먼저 구조물이 하나도 없을 때의 답을 **Subtask 1**과 같이 구합니다.
- ✓ 다음으로, 구조물이 생겼을 때, 답이 얼마나 변화하는지를 구합니다.

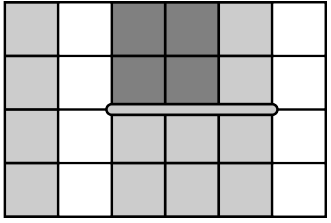
- 아래 그림과 같은 상황에서, 구조물로 인해 생긴 변화를 계산해 봅시다.
- 구조물은 총 세 개의 열을 덮고 있고, 아래에는 행이 두 개 있습니다.
- 따라서 구조물은 총 여섯 칸에 영향을 미칩니다.





- 기존에 구조물로 내려오는 세 열의 빛은 진한 회색이 2개, 연한 회색이 1개입니다.
- 즉 원래대로라면 진한 회색의 칸은 4개, 연한 회색의 칸은 2개여야 합니다.
- 하지만 구조물로 인해, 이제는 모든 칸이 연한 회색으로 변하고, 연한 회색의 칸이 총 6개가 됩니다.





- 이런 방식으로 계산하면, 결국 중요한 것은 구조물의 위치와, 구조물을 향해 각 색의 빛이 들어오는 열의 개수입니다.
- 이 값들은 모두 선형 시간에 구할 수 있습니다.
- 따라서 문제를 선형 시간 복잡도에 해결할 수 있습니다.

Subtask 4: $N \leq 2000, M \leq 2000, K \leq 2000$ (55점)

- ✓ **Subtask 2**의 방법을 사용해 문제를 해결해 봅시다.
- ✓ 구조물이 K 개가 되었지만, 상황이 크게 달라지지 않았습니다.
- ✓ 구조물을 위에 있는 것부터 아래에 있는 것 순서대로 정렬합니다.
- ✓ 구조물이 하나도 없는 상황부터 시작해, 구조물을 하나씩 추가한다고 생각하고, 각 색상의 칸이 얼마나 변하는지를 계산합니다.
- ✓ 구현에 따라, 구조물 하나당 $\mathcal{O}(NM)$ 시간 또는 $\mathcal{O}(M)$ 시간에 계산할 수 있고, 각각 **Subtask 3**과 **Subtask 4**의 풀이가 됩니다.

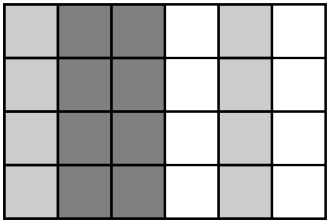
Subtask 5: $N \leq 3000, M \leq 3000$ (61점)

- 구조물의 개수가 2×10^5 개로 늘었습니다.
- 앞에서 본 **Subtask 4**의 풀이는 총 $\mathcal{O}(MK)$ 의 시간이 걸리므로, **Subtask 5**를 해결하기 힘듭니다.
- 그러나 문제의 조건에 의해, 구조물들은 서로 겹치지 않습니다.
- 따라서 구조물 길이의 총합은 $\mathcal{O}(NM)$ 이 됩니다.
- 각 구조물에 따라 일어나는 변화를 구조물의 길이에 비례하는 시간에 계산하면, 전체 문제를 $\mathcal{O}(NM)$ 에 해결할 수 있습니다. ($K \leq NM$ 임이 보장되므로 시간 복잡도에 넣지 않아도 됩니다.)

Full Task (100점)

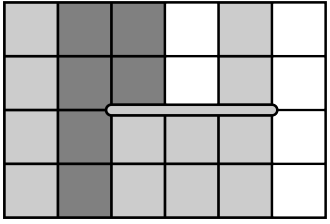
- 풀이의 방향은 앞과 같습니다.
- 하지만 N 과 M 이 커졌기 때문에, 단순히 구조물의 모든 열을 순회하며 변화량을 처리하는 방식으로는 해결할 수 없습니다.
- 핵심 아이디어는, 구조물이 많아질수록 인접한 열들이 서로 같은 색을 가지는 경우가 많다는 것입니다.
- 이렇게 인접한 열이 같은 색을 가지는 경우, 이를 묶어서 처리하면 시간을 단축할 수 있을 것 같습니다.

- 아이디어를 구체화해 봅시다.
- 어떤 연속한 열의 색이 같은 경우, 해당 열들을 하나로 묶어 **색상 그룹**이라고 합니다.
- 예를 들어, 아래 그림에서 색상 그룹은 총 5개 존재합니다.

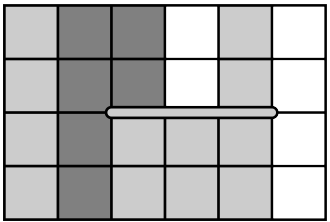


- 맨 처음, 빛이 1 행으로 내려올 때는 최대 M 개의 색상 그룹이 존재합니다.
- 빛이 아래로 내려가고, 구조물을 만남에 따라 색상 그룹의 수와 형태가 변화할 수 있습니다.
- 색상 그룹 중 구조물에 닿은 것이 있는 경우, 해당 색상 그룹들을 모두 제거하고, 다시 만들어 준다고 생각합니다.

- 예를 들어, 아래 그림과 같이 구조물이 놓인 경우를 생각해 봅시다.
- 초기 색상 그룹을 왼쪽부터 1 번부터 5 번까지 번호를 붙여 봅시다.
- 이때 구조물은 기존의 $2, 3, 4$ 번 색상 그룹에 닿아 있습니다.
- 따라서 해당 세 개의 색상 그룹들을 삭제해 줍니다.



- 구조물 아래에서 새로 생기는 색상 그룹은 어떤 것이 있을까요?
- 먼저 3 열부터 5 열까지는 구조물의 색으로 변하므로, 이 세 열을 하나의 색상 그룹으로 묶어줄 수 있습니다.
- 또한, 2 열은 기존에 2 번 색상 그룹에 포함되어 있었지만, 구조물에는 닿아있지 않으므로, 이 부분을 따로 색상 그룹으로 만들어 줘야 합니다.



- 결론적으로 구조물이 새로 추가될 때 다음과 같은 색상 그룹을 새로 만들어 줘야 합니다.
  - 구조물의 열 범위에 해당하는 색상 그룹
  - 구조물의 왼쪽 경계가 기존에 존재하던 색상 그룹을 둘로 나눌 때, 해당 색상 그룹의 왼쪽 부분에 해당하는 색상 그룹
  - 구조물의 오른쪽 경계가 기존에 존재하던 색상 그룹을 둘로 나눌 때, 해당 색상 그룹의 오른쪽 부분에 해당하는 색상 그룹
- 따라서 전 과정을 통틀어 생성되는 색상 그룹의 총 개수가 $M + 3K$ 개 이하임을 알 수 있습니다.

- 이제 색상 그룹을 빠르게 관리하며 답을 구하는 방법을 살펴봅시다.
- 색상 그룹을 저장하기 위해 필요한 정보는 왼쪽/오른쪽 경계와 그 색입니다.
- 색상 그룹을 `std::set`과 같은 자료구조에 관리합니다. 정렬 순서는 열의 순서로 설정합니다.
- 이분 탐색을 통해 각 구조물이 영향을 주는 색상 그룹을 모두 찾을 수 있고, **Subtask 2**와 같은 방법으로 답을 갱신해 줍니다.

- 시간 복잡도 증명은 지금까지의 과정을 따라왔다면 간단합니다.
- 답 갱신을 위해 셋에서 어떤 색상 그룹을 보게 되면, 해당 색상 그룹은 그 즉시 셋에서 제거됩니다.
- 생성되는 색상 그룹의 총 개수가 $\mathcal{O}(M+K)$ 이므로, 최대 제거 횟수도 이와 같습니다.
- 따라서 셋에서 색상 그룹을 추가하거나 제거하는 횟수가 총 $\mathcal{O}(M+K)$ 이하이고, 문제를 총 $\mathcal{O}((M+K)\log M)$ 에 해결할 수 있습니다.

Additional Note: 구현 팁

- ✓ 구현을 편리하게 하려면, 처음 배열 a의 값과 상관 없이, 초기에 각 열 i마다 하나의 색상 그룹이 있다고 생각하는 것이 편리합니다. 즉, 초기에 색상 그룹이 M개 있다고 생각하는 것이 편리합니다.
- ✓ 물론, 처음 존재하는 색상 그룹을 미리 계산한다고 해도 구현이 크게 복잡해지지는 않습니다.

II. 마을 짓기

`two_pointer`
출제자의 의도 – **G3**
검수진 예측 – **G3**
실제 티어 – **G2**

- ✓ 제출 257번, 정답 94명 (정답률 38.911%)
- ✓ First Solve: **playsworld16**, 7분

풀이 요약

- ✓ 마을 주대각선의 $r - c$ 값을 고정합니다.
- ✓ 해당 주대각선을 가지는 마을의 개수를 찾기 위해, 마을의 윗 변과 아랫 변을 투 포인터로 관리합니다.
- ✓ $\mathcal{O}(NM)$ 에 해결할 수 있습니다.

Subtask 2: $N = 2$ (11점)

- ✓ 모든 정사각형 영역에 대해 마을을 지을 수 있는지 확인합시다.
- ✓ $N \times M$ 격자에서 정사각형 영역이 몇 개가 있을까요?
- ✓ 왼쪽 위 끝점으로 가능한 점이 NM 개이고, 정사각형의 크기를 정하는 방법이 $\min(N, M)$ 개가 있습니다.
- ✓ 따라서 가능한 정사각형의 총 개수가 $NM \min(N, M)$ 개임을 알 수 있습니다.

- 이 서브태스크에서는 $N = 2$ 이므로 정사각형의 총 개수가 $\mathcal{O}(M)$ 개뿐입니다.
- 또한 각 정사각형의 크기도 최대 4 입니다.
- 따라서 모든 정사각형을 시도해 보며 답이 되는 것의 개수를 $\mathcal{O}(M)$ 에 구할 수 있습니다.

Subtask 3: $N \leq 80, M \leq 80$ (38점)

- ✓ **Subtask 2**의 풀이는 시간 복잡도가 $\mathcal{O}\left(N^4 M\right)$ 이었습니다. (편의상 $N \leq M$ 이라고 생각합니다. 시간 복잡도 분석에만 해당하며, 풀이에는 영향을 주지 않습니다.)
- ✓ 시간 복잡도를 네제곱 정도로 줄여야 합니다.
- ✓ 정사각형을 그 왼쪽 끝점 (r, c) 와 한 변의 길이 s 에 대해 $\mathrm{Rect}[r, c, s]$ 라고 표기합니다.
- ✓ 현재는 모든 가능한 $\mathcal{O}\left(N^2 M\right)$ 개의 $\mathrm{Rect}[r, c, s]$ 에 대해 해당 정사각형에 마을을 지을 수 있는지 $\mathcal{O}\left(N^2\right)$ 의 시간 동안 판단합니다.

Observation 1: 만약 $\mathrm{Rect}[r, c, s]$에 마을을 지을 수 없다면, $\mathrm{Rect}[r, c, s+1]$에도 마을을 지을 수 없습니다.

Proof:

- ✓ $\mathrm{Rect}[r, c, s]$에 마을을 지을 수 없는 경우, 다음 둘 중 하나가 성립합니다.
  - 해당 정사각형의 주대각선 칸 중 하나는 빛이 흐르지 못합니다.
  - 해당 정사각형의 주대각선이 아닌 칸 중 하나는 집을 짓지 못합니다.
- ✓ $\mathrm{Rect}[r, c, s+1]$의 주대각선 칸은 $\mathrm{Rect}[r, c, s]$의 주대각선 칸을 포함하므로, 위 둘 중 어느 경우에 해당하더라도 $\mathrm{Rect}[r, c, s+1]$에 마을을 지을 수 없습니다.

□

- 따라서 다음과 같이 풀이의 윤곽을 잡을 수 있습니다.
- 각 칸 (r, c)에 대해, s를 차례로 늘려가며 $\text{Rect}[r, c, s]$에 마을을 지을 수 있는지 검사합니다.
- 모든 s^2개의 칸을 전부 다 보지 말고, $\text{Rect}[r, c, s-1]$에서 새로 추가되는 $2s-1$개의 칸만 확인해도 충분합니다.
- 만약 $\text{Rect}[r, c, s]$에 마을을 지을 수 없다면, 더 이상 더 큰 s 값들은 보지 않아도 됩니다.
- 이렇게 하면 각 칸 (r, c)당 모든 $\text{Rect}[r, c, s]$ 판별을 총합 $\mathcal{O}(NM)$에 할 수 있게 됩니다.
- 따라서 전체 문제를 $\mathcal{O}\left(N^2M^2\right)$에 해결할 수 있습니다.

Subtask 4: $N \leq 400, M \leq 400$ (63점)

- ✓ $\mathrm{Rect}[r, c, s]$에 마을을 지을 수 있는지 더 빠르게 판별하는 방법을 생각해 봅시다.
- ✓ 다음과 같은 두 가지 조건을 만족해야 합니다.
  - 주대각선 칸은 모두 빛 칸이다.
  - 주대각선이 아닌 칸은 모두 집 칸이다.
- ✓ 아래와 같이 조건을 고쳐쓰는 것이 가능합니다.
  - 주대각선 칸은 모두 빛 칸이다.
  - **전체 s^2 개의 칸 중, 빛 칸은 s 개뿐이다.**

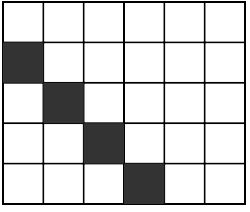
- 이제 두 조건을 누적합을 이용해 빠르게 처리해 봅시다.
- $A[r][c]$를 칸 $(r, c), (r-1, c-1), (r-2, c-2), \cdots$ 중 빛 칸의 개수로 정의합니다.
- $B[r][c]$를 $[1, r] \times [1, c]$ 영역의 빛 칸의 개수로 정의합니다. (B는 빛 칸 개수의 2차원 누적합과 같습니다.)
- A와 B는 모두 $\mathcal{O}(NM)$ 시간에 전처리 가능합니다.
- 위 두 배열을 이용하면 $\mathrm{Rect}[r, c, s]$가 유효한지 $\mathcal{O}(1)$에 확인이 가능합니다.
- 가능한 정사각형의 수가 $\mathcal{O}\left(N^2 M\right)$ 개이므로, 서브태스크를 해결할 수 있습니다.

Full Task (100점)

- ✓ 이제 임의의 정사각형이 유효한지 상수 시간에 확인할 수 있습니다.
- ✓ 그러나 문제는 가능한 정사각형의 개수가 너무 많다는 것입니다.
- ✓ 문제를 해결하려면 판별하는 정사각형의 개수를 줄여야 합니다.

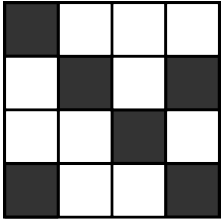
- 가장 쉬운 방법은 이분 탐색을 이용하는 것입니다.
- 목표는 각 (r, c)에 대해 $\mathrm{Rect}[r, c, s]$가 유효한 최대 s의 값을 찾는 것입니다.
- 모든 칸에 대해 이분탐색을 수행하므로, 전체 시간 복잡도 $\mathcal{O}(NM \log N)$에 문제를 해결할 수 있습니다.
- 이 풀이는 구현에 따라 시간 초과를 받을 수 있습니다.

- 더 빠른 풀이를 살펴봅시다.
- 아래 그림과 같이 정사각형의 주대각선 위치를 고정해 봅시다. (주대각선 칸들은 $r - c$ 값이 일정한데, 이 값을 고정합니다.)
- 총 $N + M - 1$ 가지가 가능합니다.

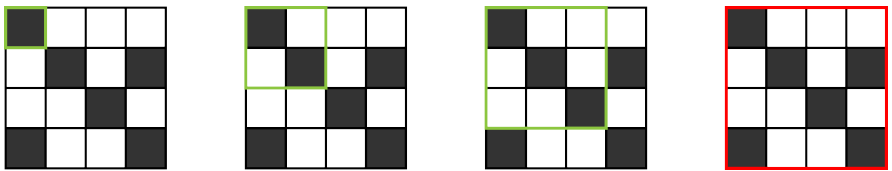


- 이제 저 대각선 위에서 마을이 될 정사각형의 윗변과 아랫변 위치를 투 포인터 형식으로 관리하며, 가능한 모든 영역을 찾습니다.
- 대각선 위 칸들은 모두 빛이 흐를 수 있어야 합니다. 빛이 흐를 수 있는 연속된 대각선 칸을 묶어서 생각합니다.
- 마을을 이룰 수 있는 경우 아랫변 위치를 다음에 1 만큼 내리고, 마을을 이룰 수 없는 경우 윗변 위치를 1 만큼 올립니다.

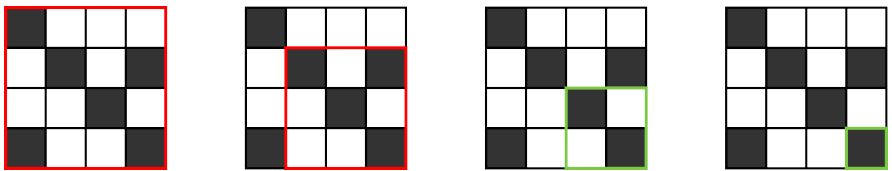
- 다음은 투 포인터 알고리즘의 작동 예시입니다.
- 검은 칸은 빛이 흐를 수 있는 칸, 하얀 칸은 집을 지을 수 있는 칸을 나타냅니다.
- 마을의 주대각선은 $r - c = 0$으로 고정합니다.



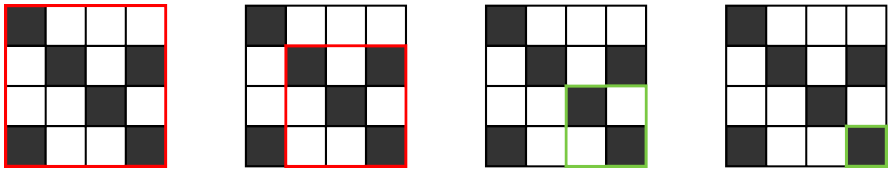
- 투 포인터 알고리즘을 진행합니다.
- 영역이 올바른 마을을 이룬다면 계속 확장해 나갑니다.
- 올바르지 않은 마을을 이룬다면, 윗변이 고정되었을 때 만들 수 있던 가장 큰 올바른 마을의 크기를 기록합니다.
- 아래 그림의 경우, 윗변을 1 행으로 고정했을 때 가능한 마을의 크기는 3 이 최대였습니다.



- 계속 진행하며, 윗변이 고정되었을 때 마을의 최대 크기를 구합니다.
- 윗변을 2 행으로 진행했을 때, 마을 크기가 3 인 것은 불가능합니다.
- 마을 크기가 2 인 경우 가능함을 알 수 있습니다. (1 행에서 마을 최대 크기가 3 이었기 때문에, 해당 마을에서 첫 행, 첫 열만을 제외한 경우도 가능합니다.)
- 이런 식으로 모든 가능한 윗변에 대해 투 포인터 알고리즘으로 최대 크기를 구합니다.



- 결과는 다음과 같습니다.
  - 윗변이 1 행인 경우 최대 크기는 3
  - 윗변이 2 행인 경우 최대 크기는 2
  - 윗변이 3 행인 경우 최대 크기는 2
  - 윗변이 4 행인 경우 최대 크기는 1



- 이제 이 결과를 토대로 해당 주대각선을 가지는 모든 크기의 영역 개수를 세어 봅시다.
- 결과를 바탕으로 Counting Array $[1, 2, 1, 0, \cdots]$ 를 만들 수 있습니다.
- 윗변이 1 행인 경우 최대 크기가 3 이라는 것은, 윗변이 1 인 경우 크기가 $1, 2$ 인 마을도 가능하다는 뜻입니다.
- 따라서 역방향 누적합을 구해 주면, $[4, 3, 1, 0, \cdots]$ 과 같은 결과를 얻을 수 있습니다.
- 결과적으로, 전체 시간 복잡도 $\mathcal{O}(NM)$ 에 문제를 해결할 수 있습니다.

III. 의식의 광장

`ad_hoc, constructive`
출제자의 의도 – **S4**
검수진 예측 – **S1**
실제 티어 – **S2**

- ✓ 제출 298번, 정답 141명 (정답률 48.658%)
- ✓ First Solve: **kyo20111**, 4분

풀이 요약

- ✓ N 개의 판을 c_i 순으로 정렬합니다.
- ✓ c_i 값이 작은 것부터 차례대로 1 부터 N 까지의 이동 거리를 배정합니다.

Subtask 4: $H = 1$ (43점)

- 행이 단 하나뿐입니다.
- 어떻게 이동 거리를 배정해야 판끼리 겹치지 않을까요?
- 초기에 두 판이 열 c_1, c_2에 있었고, 이 두 판에 이동거리 d_1, d_2를 각각 배정한다고 합시다.
- 만약에 초기에 $c_1 < c_2$였다면, 이후에 $c_1 + d_1 < c_2 + d_2$가 성립해야 합니다.

- 한 가지 간단한 방법은 c_i 가 더 큰 쪽에 더 큰 d_i 를 배정하는 것입니다.
- 이렇게 하면 $c_i < c_j$ 일 때 항상 $d_i < d_j$ 이므로, 결과적으로 $c_i + d_i < c_j + d_j$ 도 항상 성립합니다.
- 따라서 간단한 정렬 한 번으로 문제를 해결할 수 있습니다.
- 시간 복잡도는 $\mathcal{O}(N \log N)$ 입니다.

Full Task (100점)

- ✓ 이제 판이 여러 행에 놓일 수 있습니다.
- ✓ 하지만 문제는 여전히 같은 방법으로 해결할 수 있습니다.

- 판의 위치를 정렬해, N 개의 위치 $(r_1, c_1), \cdots, (r_N, c_N)$ 이 $c_1 \leq c_2 \leq \cdots \leq c_N$ 을 만족한다고 가정합시다.
- 이제 i 번 판에 이동 거리 i 를 배정하면, $i < j$ 일 때 $c_i < c_j$ 가 성립합니다.
- 따라서, N 개의 판은 모두 다른 열에서 이동을 종료하며, **Subtask 1**과 같은 이유로 두 판이 겹치는 일도 일어나지 않습니다.
- 시간 복잡도는 $\mathcal{O}(N \log N)$ 입니다.

Ⅳ. 안정적인 구조

`set, combinatorics`
출제자의 의도 – **D3**
검수진 예측 – **D4**
실제 티어 – **D3**

- ✓ 제출 46번, 정답 16명 (정답률 34.783%)
- ✓ First Solve: **gs888484**, 120분

풀이 요약

- ✓ 문제의 상황을 순열 문제로 변형하고, 적당한 관찰을 통해 격자 위에서 특정한 점을 지나고 대각선 아래를 지나지 않는 경로를 세는 문제로 바꿉니다.
- ✓ 이제 지나야 하는 점을 `std::set` 등의 자료구조에 넣거나 빼면서 경우의 수를 유동적으로 관리합니다.
- ✓ 대각선 아래를 지나지 않는 경로의 개수는 대각선을 기준으로 격자를 대칭시키는 트릭을 통해 구할 수 있습니다.

Subtask 3: $Q = 1$ (10점)

- 먼저 아무런 조건이 없을 때 안정적인 구조의 개수를 살펴봅시다.
- 규칙성을 찾으면 풀이의 방향에 도움이 될 것입니다.
- 브루트 포스를 이용하면, 가능한 경우의 수가 카탈란 수와 같다는 것을 관찰할 수 있습니다.
- 카탈란 수는 공식이 존재하므로, 해당 공식을 이용해 서브태스크를 해결할 수 있습니다.

Subtask 6: $Q \leq 2$ (23점)

- 경우의 수가 카탈란 수라는 것을 아는 것도 중요하지만, 왜 같은지를 알아야 진전을 이룰 수 있습니다.
- 경우의 수가 카탈란 수와 같다는 것을 보이기 위해, 일대일 대응을 생각해 봅시다.
- 카탈란 수와 관련된 상황은 많이 알려져 있습니다. 그 중 어떤 것과 일대일 대응을 시도해야 할까요?

- 먼저 문제의 상황을 격자가 아닌 순열로 생각해야 합니다.
- c 행에 있는 빛이 P_c 행에 위치해 있다고 합시다.
- 이때 $i < j < k$ 이고 $P_i > P_j > P_k$ 인 (i, j, k) 가 존재하면 안 됩니다.
- 이러한 순열의 특징을 찾아 봅시다.

- 예를 들어 $[3, 1, 5, 2, 4]$와 같은 순열을 생각해 봅시다.
- 이러한 순열은 최댓값이 등장하는 원소와 그렇지 않은 원소로 나눌 수 있습니다.
- 예를 들어, 위의 순열에서 3과 5는 전에 등장한 모든 수보다 크지만, $1, 2, 4$는 그렇지 않습니다.
- 이렇게 전에 등장한 모든 원소보다 큰 수를 **중요한 원소**, 나머지 수들은 **중요하지 않은 원소**라고 합시다.

Observation 1: 중요하지 않은 원소들은 크기 순으로 등장합니다.

Proof:

- ✓ 만약 어떤 두 중요하지 않은 원소 P_i, P_j 가 존재해, $i < j$ 이고 $P_i > P_j$ 라고 해 봅시다.
- ✓ P_i 는 중요하지 않은 원소이므로, $x < i$ 이고 $P_x > P_i$ 인 x 가 존재합니다.
- ✓ 따라서 $x < i < j$ 이고, $P_x > P_i > P_j$ 이므로, 문제의 조건을 성립하지 않게 됩니다.
- ✓ 따라서, 중요하지 않은 원소들은 모두 오름차순으로 등장합니다.

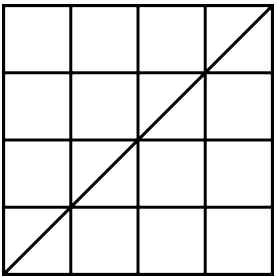
□

- 이는 곧, 중요한 원소와 그 위치가 모두 확정되면 나머지 원소들도 확정된다는 뜻입니다.
- 즉, 중요한 것은 중요한 원소를 확정하는 방법의 수입니다.
- 중요한 원소가 $P_{x_1}, P_{x_2}, \cdots, P_{x_k}$ 라고 해 봅시다.
- 이들은 어떤 조건을 만족해야 할까요?

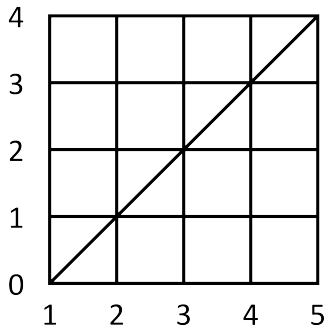
- 먼저 $x_1 < x_2 < \cdots < x_k$ 를 고정해 봅시다.
- 이때 중요한 원소들은 앞에 있는 수들보다 모두 커야 하므로, 당연히 $P_{x_1} < P_{x_2} < \cdots < P_{x_k}$ 를 만족해야 합니다.
- 또한, 중요한 조건이 하나 더 있습니다.
- P_x 가 $P_1, \cdots, P_x$ 중 최대가 되려면, $P_x \geq x$ 여야 합니다.
- 중요한 원소들이 충족해야 하는 조건은 위 두 가지로 충분하고, 위 두 가지에 해당하는 경우 조건을 만족하는 순열을 정확히 한 개씩 만들 수 있습니다.

- 결국 $P_{x_i} \geq x_i$ 이고 x_i 와 P_{x_i} 가 모두 증가하는 점들을 결정하는 것이 중요합니다.
- 카탈란 수와 이 상황을 어떻게 연결할 수 있을까요?
- 이 부분이 이 문제에서 가장 어려운 부분입니다.

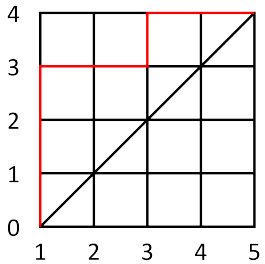
- 다음과 같은 $N \times N$ 격자를 생각해 봅시다.
- 해당 격자에서 대각선 아래를 지나지 않고 맨 왼쪽 아래 점에서 맨 오른쪽 위 점까지 (오른쪽이나 위로만 이동해) 가는 경로는 C_N 개임이 잘 알려져 있습니다.



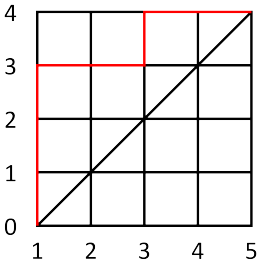
- 격자의 각 가로줄과 세로줄에 다음과 같이 번호를 붙여 봅시다.
- 아래는 $N = 4$인 경우의 예시입니다.



- 다음과 같은 경로를 생각해 봅시다.
- 이 경로에서 위 방향으로 향하다가 오른쪽 방향으로 꺾는 꼭짓점들을 생각해 봅시다.



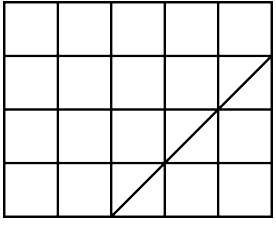
- 이러한 꼭짓점들의 좌표를 순열의 중요한 원소들과 일대일 대응시킬 수 있습니다.
- 예를 들어, 아래의 격자는 순열 $[3, 1, 4, 2]$에 대응됩니다.



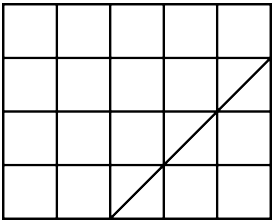
- 이제 격자와 순열을 일대일 대응시키는 방법을 찾았습니다.
- 그렇다면 문제에서 등장하는 조건을 살펴봅시다.
- 문제의 조건은 $\max(P_1, \cdots, P_x) = v$ 꼴입니다.
- 이를 격자에서 생각하면, 경로가 (x, v) 와 $(x+1, v)$ 를 모두 지난다는 것과 같습니다.
- 만약 P_x 가 중요한 원소라면 (x, v) 에서 경로가 꺾여야 하고, P_x 가 중요한 원소가 아니라면 P_x 전의 가장 뒤에 있는 중요한 원소 $P_{prv} = v$ 라는 뜻이므로, 경로는 (prv, v) 에서부터 $(x+1, v)$ 까지 오른쪽으로 향해야 합니다.

- 결론적으로 문제의 조건을 격자 위로 옮기면, 지나야 하는 정점 두 개가 추가되는 것과 같습니다.
- **Subtask 6**에서는 $Q \leq 2$ 이므로, 조건이 최대 하나인 경우만 생각해도 됩니다.
- 조건이 하나인 경우, 문제는 격자의 $(1, 0)$ 에서 두 점을 거쳐 $(N + 1, N)$ 까지 대각선 위로 가는 경우의 수를 세는 것과 같습니다.
- 이를 위해서, 임의의 칸 (x_1, y_1) 에서 (x_2, y_2) 로 대각선 위로만 가는 방법의 수를 구할 수 있어야 합니다.

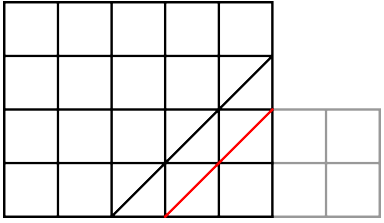
- ✓ (x_1, y_1)과 (x_2, y_2)를 두 꼭짓점으로 가지는 직사각형 영역 내부만 생각해도 충분합니다.
- ✓ 만약 격자의 대각선이 이 영역 밖에 있다면 단순히 $\begin{pmatrix} x_2 - x_1 + y_2 - y_1 \\ x_2 - x_1 \end{pmatrix}$을 계산하면 답이 됩니다.
- ✓ 그러나, 아래 그림처럼 대각선이 내부에 있다면, 상황이 조금 복잡해집니다.



- 이 경우, 먼저 대각선 조건을 무시하고 전체 경우의 수를 구해 줍니다.
- 다음으로, 대각선을 지나가는 경우의 수를 빼 주면 됩니다.



- 대각선을 지나는 모든 경로는 빨간 대각선 위의 점을 최소 하나는 지납니다.
- 빨간 대각선을 지난 이후의 경로에서 위 방향 이동과 오른쪽 방향 이동을 서로 바꿔 준다고 생각합시다.
- 그러면 최종적으로 도달하는 위치는 회색 격자의 오른쪽 위 끝점이 됩니다.



- 이렇게 대칭을 이용한 트릭을 쓰면 대각선 조건이 있는 경우에도 경로의 수를 구할 수 있습니다.
- 여기까지 왔다면, 격자 위에서 지나야 할 점들 목록을 알고 있으니, 각각의 구간에 대해 경우의 수를 구한 뒤 곱해 주면 답이 됩니다.
- 이렇게 많은 관찰 끝에 **Subtask 6**을 해결할 수 있었습니다.

Subtask 5: $N \leq 2000, Q \leq 2000$ (82점)

- 앞의 관찰을 모두 했다면 제곱 서브태스크는 쉽습니다.
- 마찬가지로 각 조건을 격자 조건으로 변환한 뒤, 지나가야 하는 점의 목록을 구합니다.
- 이제 점을 순서대로 정렬하고 경우의 수를 구하면 됩니다.
- 점의 배치에 따라 불가능한 경우가 발생할 수도 있습니다.
- 시간 복잡도 $\mathcal{O}(NQ \log Q)$에 문제를 해결할 수 있습니다.

Full Task (100점)

- 전체 문제도 비슷합니다.
- 조건이 추가되거나 제거될 때마다 지나야 하는 격자점 목록을 `std::set`에 관리합니다.
- 점을 추가하거나 제거할 때마다 셋에서 인접한 점 사이의 경로 수를 관리하고, 답에 곱하거나 나눕니다.
- 경우의 수가 0 (또는 mod 값의 배수)가 되는 경우 나눗셈이 불가능하므로, 그 개수를 따로 관리합니다.
- 이렇게 하면 시간 복잡도 $\mathcal{O}\left(N + Q \log Q\right)$에 문제를 해결할 수 있습니다.

V. 기록의 판

`greedy`, `dp`
출제자의 의도 – **P5**
검수진 예측 – **P1**
실제 티어 – **P3**

- ✓ 제출 103번, 정답 34명 (정답률 33.981%)
- ✓ First Solve: **black_yuki**, 70분

풀이 요약

- ✓ 아래쪽 행부터 시작하여, 다음 행보다 작은 최대의 수를 만드는 그리디한 전략이 옳음을 관찰해야 합니다.
- ✓ 이후 다음 행보다 작은 d자리를 만들기 위해 필요한 최소 공간을 DP로 관리하며, 왼쪽 자리부터 확정시켜 나갑니다.
- ✓ $\mathcal{O}\left(9N^2\right)$에 문제를 해결할 수 있습니다.

Subtask 5: $N \leq 500$ (69점)

- ✓ 먼저 가장 간단한 부분부터 생각해 봅시다.
- ✓ 맨 아래 행의 경우 어떻게 해야 할까요?
- ✓ 생각해 보면, 맨 아래 행에서는 숫자를 지울 이유가 없습니다.
- ✓ 맨 아래에 있기 때문에, 가장 큰 수여야 하고, 숫자를 지우면 수의 크기가 작아지기 때문입니다.

- 그렇다면 그 위의 행은 어떻게 해야 할까요?
- 만약 수를 지우지 않아도 된다면, 그렇게 하는 것이 최적일 것입니다.
- 하지만 경우에 따라, 수를 지워야 할 수도 있습니다.
- 그렇다면 수를 하나만 지우는 것이 최적일 것이고, 수를 하나만 지우는 경우의 수 중에서도 해당 행의 수를 최대화하는 것이 최적일 것입니다.

- 결론적으로, 맨 아래 행부터 시작하며 각 행의 수를 그 아래 행보다 작으면서 최대로 만드는 그리디한 전략을 생각할 수 있습니다.
- 이제 이 전략을 어떻게 구현해야 할지 생각해 봅시다.

- 목표는 $i+1$ 행에 적힌 수를 알 때, i 행에 적힌 숫자들 중 몇 개를 지워서 $i+1$ 행보다 작은 수 중 최대가 되게 하는 것입니다.
- $i+1$ 행의 수가 d 자리 수라고 해 봅시다.
- 먼저 i 행의 수가 d 자리가 될 수 있는지 확인해 봅시다.
- i 행에 있는 숫자들로 만들 수 있는 최소의 d 자리 수가 $i+1$ 행의 수보다 작은지 확인하면 됩니다.

- 결론적으로 i 행에 있는 숫자들로 만들 수 있는 가장 작은 d 자리 수를 구해야 합니다.
- 그리디한 전략을 이용해, 앞 자리부터 보며 가능한 최소한의 숫자를 넣는 것을 시도해 볼 것입니다.
- 첫째 자리에 1 을 넣을 수 있으려면, 해당 행의 첫 번째 1 뒤로 최소한 $d-1$ 개의 수들이 더 있어야 할 것입니다.
- 이러한 조건들을 확인하며 앞에서부터 수를 채워넣는 것은 $\mathcal{O}\left(N^2\right)$ 에 가능합니다.
- 따라서 i 행에 있는 숫자들로 만들 수 있는 가장 작은 d 자리 수를 $\mathcal{O}\left(N^2\right)$ 에 구할 수 있습니다.
- 이 수를 $\mathcal{O}\left(N\right)$ 에 $i+1$ 행의 수와 비교하여, 최종적으로 i 행의 수가 $d-1$ 자리여야 하는지, d 자리여야 하는지를 결정할 수 있습니다.

- 이제 경우를 둘로 나눠 봅시다.
- 만약 i 행의 수가 $d-1$ 자리라면, 가장 큰 $d-1$ 자리 수를 만들어야 합니다.
- 이것은 앞과 비슷한 방식으로 $\mathcal{O}\left(N^2\right)$ 에 가능합니다.
- 그렇지 않다면, $i+1$ 행보다는 작은 가장 큰 d 자리 수를 만들어야 합니다.
- 이 문제에서 가장 어려운 부분입니다.

- 비슷한 아이디어를 사용해 봅시다.
- 맨 앞 자리에 올 수 있는 가장 큰 수를 구해 봅시다.
- $i+1$ 행을 따라 읽은 수의 d 개 자릿수가 $p_d p_{d-1} \cdots p_1$ 이라고 해 봅시다. (높은 자리부터 낮은 자리 순입니다.)
- 일단 첫 번째 수는 당연히 p_d 이하여야 합니다.
- p_d 를 놓을 수 있다면 가장 좋을 텐데, 어떻게 알 수 있을까요?
- 뒤 $d-1$ 자리를 최소로 잡았을 때 그 결과가 $i+1$ 행의 끝 $d-1$ 자리 미만이라면 가능할 것입니다.

- 여기서 문제가 재귀적으로 반복된다는 것을 알 수 있습니다.
- p_d를 놓을 수 있는지 확인하려면 뒷부분에서 $d-1$자리를 놓아야 하는데, 그 다음 자리가 p_{d-1}이 될 수 있는지를 또 같은 방식으로 확인해야 합니다.
- 한편 이러한 숫자가 존재하지 않는 순간이 언젠가 발생할 것이고, 이 경우 더 작은 대안들을 찾아보아야 합니다.
- 이 경우 숫자가 한 번이라도 대응되는 p_i에서 작아졌다면 그 뒤의 수는 아무거나 써도 상관없으므로 단순히 인덱스를 비교하는 것으로 가능성을 판별할 수 있습니다.
- 이 전체 과정은 행별로 $\mathcal{O}\left(N^2\right)$에 구현 가능합니다.
- 따라서 문제를 $\mathcal{O}\left(N^3\right)$에 해결할 수 있습니다.

Full Task (100점)

- 만점을 받으려면 각 행별로 최적의 수를 $\mathcal{O}\left(O(N)\right)$에 찾아야 합니다.
- 앞과 같이, 한 행의 상황만 생각합니다.
- 전 행의 수의 마지막 d자리 이하인 d자리 수를 만들기 위해 이번 행의 $[c, N]$번 열 구간으로 충분할 때, 이러한 c의 최댓값을 $DP[c]$라고 정의합시다.
- 여기서 $DP[c]$가 c에 따라 감소한다고 착각하기 쉽지만, 그렇지 않습니다. d번째 자리가 9 등으로 매우 크다면 그냥 아무 d자리만 잡아도 성공하기 쉬운 반면, d번째 자리가 1이라면 선택의 여지가 매우 제한됩니다. 이런 점들을 고려하면 DP가 c에 따라 감소할 이유가 없습니다.

- DP를 시행하기에 앞서 $prv[x][d]$ 배열을 전처리해 둡니다. 이 배열의 정의는 현재 행의 x번 열 바로 이전에 존재하는 숫자 d의 위치입니다. 이는 $\mathcal{O}(9N)$에 쉽게 구할 수 있습니다.
- 비슷한 방식으로 $nxt[x][d]$ 배열도 전처리해 둡니다. 이 배열의 정의는 prv와 비슷하지만 방향이 반대입니다.

- 두 가지 경우로 나누어 봅시다.
- 다음 행의 d번째 자리보다 이번 행의 d번째 자리를 작게 잡을 경우, 단순히 $N - d + 1$열 또는 그 이전의 가장 오른쪽 수를 고르면 됩니다. 이러한 경우의 수는 앞서 구해 둔 prv 배열을 통해 d당 $\mathcal{O}(9)$ 시간에 쉽게 확인할 수 있습니다.
- 다음 행의 d번째 자리와 이번 행의 d번째 자리를 같게 잡을 경우, 다음 $d - 1$번째 자리부터는 이전 행보다 작아야 합니다. 이 조건은 $DP[d-1]$과 유사함을 알 수 있습니다. 따라서 $DP[d-1]$ 위치에서 prv 배열을 확인하면 됩니다.

- 결과적으로 DP 배열을 $\mathcal{O}(9N)$ 시간에 구할 수 있습니다.
- 이제 이를 통해 전 행보다 작은 최대의 수를 구하는 방법을 알아봅시다.
- 일단 $DP[1]$의 값이 1 이상, 즉 전 행과 같은 자릿수를 가지며 더 작은 수를 만드는 게 가능한지 확인합니다.
- 불가능하다면 자릿수가 하나 더 작은 가장 큰 수를 만들어야 합니다. 이것은 앞 자리부터 최대한 크게 정하려고 시도하는 그리디를 사용하면 되는데, 9부터 시도해 보며 nxt 배열을 이용해 최대한 왼쪽에 있는 수를 찾으면 됩니다.
- 오른쪽에서 d번째 자리를 정할 때 고르는 수는 최대 $N-d+1$열이어야 합니다. 그 오른쪽이라면 그 수는 고를 수 없고, 더 작은 후보를 시도해 봐야 합니다.
- 결과적으로 이 경우 $\mathcal{O}(9N)$ 시간에 최적의 답을 찾을 수 있습니다.

- 이제 전 행과 같은 자릿수의 수를 만드는 것이 가능하다고 해 봅시다.
- 이 경우 DP 배열을 이용해 답을 구할 것입니다.
- 먼저 가장 높은 자리를 이전 수와 같게 만들 수 있는지 봐야 하는데, 이것은 현재 열에서 이전 행의 가장 높은 자리가 존재하는 가장 왼쪽 위치 (nxt를 이용해 빠르게 찾습니다), 그리고 $DP[2]$의 값을 비교해 판단할 수 있습니다.
- 이것이 가능하다면, 다음 자리에 대해 똑같은 과정을 반복합니다.
- 이것이 한 번이라도 불가능해진다면, 차선의 후보를 확인해야 합니다. 여기부터는 앞에서 같은 자릿수를 만드는 것이 불가능했던 상황과 마찬가지로, 더 작은 후보들을 nxt를 이용해 확인해 보며 오른쪽에 자리가 충분한 최대의 수를 뽑는 과정을 끝까지 반복하면 됩니다.
- 결과적으로 문제를 $\mathcal{O}\left(9N^2\right)$에 해결할 수 있습니다.

VI. 소용돌이 문양

`prefix_sum`
출제자의 의도 – **P1**
검수진 예측 – **P3**
실제 티어 – **P2**

- ✓ 제출 71번, 정답 21명 (정답률 29.577%)
- ✓ First Solve: **hos_lyric**, 37분

풀이 요약

- ✓ 문제에서 주어진 모양을 살짝 변형해 고리 모양 여러 개로 만듭니다.
- ✓ 고리를 상하좌우 부분으로 나누어 생각합니다.
- ✓ 해당 부분을 만들기 위해서는 누적합을 사용하는 것이 좋습니다. 누적합을 사용하기 전의 형태를 생각해 보고, 그 형태를 누적합을 이용해 만들어야 합니다.
- ✓ 최종적으로 상수 개의 칸을 업데이트하고, 누적합을 두 번 사용해 최종 모양이 만들어집니다.

Subtask 2: $K = 1$ (7점)

- ✓ 문제에 주어진 규칙대로 소용돌이 문양을 직접 만들면 됩니다.
- ✓ 소용돌이 문양 하나당 $\mathcal{O}(NM)$에 만들 수 있습니다.
- ✓ 같은 방식으로 하면 **Subtask 4**도 어렵지 않게 풀 수 있습니다.

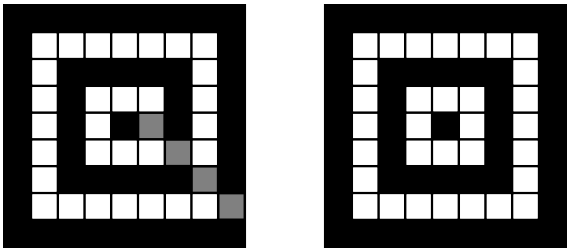
Subtask 5: $K \leq 2000$ (40점)

- ✓ 소용돌이 문양을 만드는 규칙을 잘 살펴보면, 하나의 소용돌이 문양은 최대 $\mathcal{O}(N)$ 개 정도의 직사각형들로 나눌 수 있음을 알 수 있습니다.
- ✓ 각각의 직사각형에 대해 2 차원 누적합을 사용하면 각 칸이 속한 소용돌이 문양의 개수를 $\mathcal{O}(NK + NM)$ 시간에 구할 수 있습니다.

Full Task (100점)

- ✓ 전체 문제를 풀기 위해서는 소용돌이 문양의 모양에 대한 관찰이 필요합니다.
- ✓ 모양 자체는 매우 복잡하게 생겼지만, 소용돌이 문양과 비슷하게 생겼으면서도 더 단순한 모양을 찾는다면 문제에 더 쉽게 접근할 수 있을 것입니다.

- 아래 그림과 같이, 오른쪽 격자에 표시된 칸들을 반전시키면 소용돌이 문양을 반지름 $0, 2, 4, \cdots$ 의 고리들로 바꿀 수 있습니다.

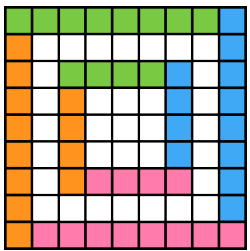


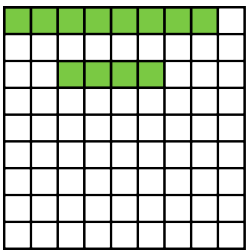
- 따라서 각 소용돌이 문양을 만들기 위해서는 다음 두 가지를 하면 됩니다.
  - 고리 모양을 업데이트하기
  - 반전된 대각선 부분을 업데이트하기

- 상대적으로 쉬운 대각선 부분부터 생각해 봅시다.
- 만약 특정 대각선 부분에 $+1$을 하는 거라면, difference array를 만들고 누적 합을 구해 해결할 수 있습니다.
- 예를 들어, $A[r][c] = 1$, $A[r+d][c+d] = -1$로 업데이트해 놓고, 모든 (r, c)에 대해 $A[r][c] = A[r][c] + A[r-1][c-1]$ 꼴의 업데이트를 시행한다면, 대각선 방향으로 누적 합을 구해 원하는 형태를 얻을 수 있을 것입니다.
- 그러나 현재는 홀수 번째 칸에 -1, 짝수 번째 칸에 $+1$을 해야 하는 형태입니다.
- 이 경우 $A[r][c] = -1$, $A[r+d][c+d] = 1$로 시작하고, 모든 (r, c)에 대해 $A[r][c] = A[r][c] - A[r-1][c-1]$의 형태로 업데이트해 문제점을 해결해줄 수 있습니다.

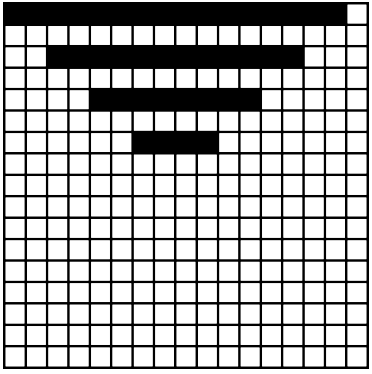
- 이제 고리 부분을 살펴봅시다.
- 가운데의 반지름 0 고리 부분은 소용돌이 문양의 중심에 있는 한 칸을 업데이트하는 것인데, 이 부분은 쉽기 때문에 제외하고 생각합니다.
- 이제 업데이트해야 하는 모양은 반지름 $2, 4, \cdots$ 인 고리 부분입니다.

- 아래 그림과 같이, 고리를 위쪽 부분, 오른쪽 부분, 아래쪽 부분, 왼쪽 부분으로 나눠서 생각합시다.
- 그러면 네 부분은 위치/방향만 다른 대칭적인 모양이 되므로, 서로 비슷한 방식으로 업데이트가 가능합니다.
- 위쪽 부분을 업데이트하는 방법을 알아봅시다.





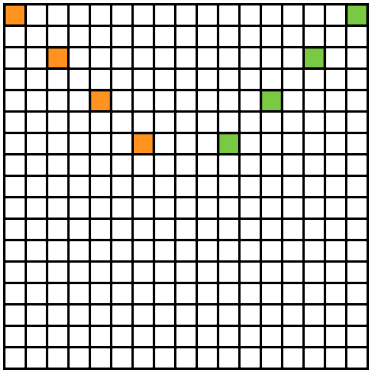
- 다음은 이해를 돕기 위해, 소용돌이 문양의 크기가 더 커졌을 때 ($d = 8$) 위쪽 부분을 표시해 놓은 그림입니다.



- 형태를 관찰하면, 가로 길이가 $4, 8, 12, \cdots$, 세로 길이가 1 인 형태의 직사각형들을 그려야 함을 알 수 있습니다.
- 이러한 형태의 그래프를 그릴 때는, 행렬의 처음 몇 칸을 적당히 설정해준 뒤, 가로 방향으로 누적합을 진행하면 빠르게 처리할 수 있습니다.

- 형태를 관찰하면, 가로 길이가 $4, 8, 12, \cdots$, 세로 길이가 1 인 형태의 직사각형들을 그려야 함을 알 수 있습니다.
- 이러한 형태의 그래프를 그릴 때는, 격자의 처음 몇 칸을 적당히 설정해준 뒤, 가로 방향으로 누적합을 진행하면 빠르게 처리할 수 있습니다.

- 이를 위해서는, 누적합을 돌리기 전 격자의 아래 주황색 칸들 (우하단 방향의 대각선) 위에는 $+1$ 이, 초록색 칸들 (우상단 방향의 대각선) 위에는 -1 이 적혀 있어야 합니다.



- 그렇다면 저 상태를 만들기 위해서는 어떻게 해야 할까요?
- 주황색 부분과 초록색 부분에 대해 따로 생각합시다. 대칭적인 상황이므로 주황색 부분만 살펴봅니다.
- 주황색 부분 역시 일정한 간격으로 업데이트하는 것이므로 누적 합을 사용할 수 있습니다.
- 처음 적당한 두 칸에 $+1$과 -1을 적고, $A[r][c] = A[r][c] + A[r-2][c-2]$ 꼴로 누적합을 계산하면 됩니다.
- 같은 방식으로 초록색 부분도 계산할 수 있습니다.

- 따라서 상수 개의 칸을 업데이트하고, 누적합을 두 번 사용해 반지름이 $2, 4, \cdots$ 인 고리 모양을 만들 수 있습니다.
- 여기서 앞에서 설명한 대로 대각선 반전 부분을 누적합을 통해 적당히 업데이트하면 소용돌이 문양이 만들어집니다.
- 이 모든 과정을 K 개의 소용돌이 문양에 대해 한꺼번에 처리하면 $\mathcal{O}(NM + K)$ 에 문제를 해결할 수 있습니다.

Additional Note: 대각선 반전을 사용하지 않아도 됩니다

- ✓ 사실 위의 풀이를 모두 이해했고, 소용돌이 문양을 잘 살펴본다면, 굳이 대각선 부분을 반전해 고리로 만들지 않고도 비슷한 방식으로 해결할 수 있습니다.
- ✓ 그러나 이 경우 네 방향에 대해 업데이트해야 하는 영역의 길이 등이 살짝 달라지게 됩니다.
- ✓ 에디토리얼에서는 설명의 편의를 위해 이렇게 설명했습니다.

Additional Note: 이중 누적합을 사용하지 않는 다른 풀이

- 이중 누적합을 사용하지 않는 다른 풀이 방식도 존재합니다.
- 고리의 위쪽 부분을 다음과 같이 업데이트할 수 있습니다.
- 고리의 각 가로줄에 대해, 각 칸에 '씨앗'을 심는다고 생각해 보겠습니다.
- 가로줄마다, 가장 왼쪽 칸에 씨앗 A, 양옆 두 칸을 제외한 가운데 칸에는 씨앗 B, 가장 오른쪽 칸에 씨앗 C를 심는다고 생각합니다.
- 이 경우, DP와 비슷한 방식으로 문제를 해결할 수 있습니다.

Additional Note: 이중 누적합을 사용하지 않는 다른 풀이

- 먼저 가장 아래쪽 가로줄에 씨앗을 심습니다. 길이가 4 이므로, 네 칸에 각각 씨앗 A, B, B, C를 심습니다.
- 그 위의 줄들은 위 방향으로 스위핑하면서 다음과 같이 처리합니다.
  - 씨앗 A는 $(-2, -2)$ 떨어진 칸에 씨앗 A를, $(-2, -1)$ 과 $(-2, 0)$ 떨어진 칸에는 씨앗 B를 심습니다.
  - 씨앗 B는 $(-2, 0)$ 떨어진 칸에 씨앗 B를 심습니다.
  - 씨앗 C는 $(-2, 0)$ 과 $(-2, 1)$ 떨어진 칸에 씨앗 B를, $(-2, 2)$ 떨어진 칸에는 씨앗 C를 심습니다.

Additional Note: 이중 누적합을 사용하지 않는 다른 풀이

- ✓ 위 과정을 반복하면 반지름이 $2, 4, \cdots, \infty$ 인 고리들이 만들어집니다.
- ✓ 반지름을 적당한 위치에서 제한하려면, 적당한 가로줄에 씨앗 A, B, C를 음수 개수만큼 심어서 고리가 더 생기는 것을 제한할 수 있습니다. (이 과정에서 누적합이 필요합니다.)
- ✓ 원래는 이 풀이가 정해였지만, 출제 및 검수 과정에서 더 쉬운 풀이가 발견되었습니다.

VII. 힘의 결합

`segtree`, `sweeping`
출제자의 의도 – **D1**
검수진 예측 – **R5**
실제 티어 – **R5**

- ✓ 제출 142번, 정답 7명 (정답률 4.930%)
- ✓ First Solve: **hos_lyric**, 257분

풀이 요약

- ✓ A의 누적합 S를 생각하고, S를 2차원으로 plotting해 $O(N+Q)$개의 직사각형 구간으로 생각합니다.
- ✓ 인덱스 축으로 스위핑하며 Segment Tree Beats에 historic sum을 추가하면 문제를 해결할 수 있습니다.

Subtask 3: $s_i = e_i, l_i = r_i$ (34점)

- ✓ 이 문제를 시도할 실력이라면 어느 정도 배경 지식이 있을 테니 **Subtask 3**부터 설명합니다.
- ✓ 이 서브태스크는 쿼리로 특정 (t, x)에 대해 P_{tx}의 값을 물어봅니다.
- ✓ 금광 세그와 오프라인 쿼리를 이용한 정직한 수열과 쿼리 문제로, $\mathcal{O}(Q \log N)$에 해결하는 것이 쉽습니다.

Subtask 4: $s_i = e_i$ (53점)

- ✓ 여기서부터는 구간 합을 있는 그대로 생각하면 어렵습니다.
- ✓ A의 누적 합 S를 생각해 봅시다.
- ✓ x번 집을 포함하는 최대 구간 합은 $\max_{y \geq x} S_y - \min_{y < x} S_y$와 같습니다.
- ✓ 따라서 문제를 두 부분 ($\max S_y$ 구하기와 $\min S_y$ 구하기)로 나눌 수 있습니다.
- ✓ **Subtask 4**부터는 고려하는 (t, x) 쌍이 많아지지만, 이때도 저렇게 두 부분으로 나눠서 생각하는 것이 더 쉽습니다.

- ✓ 문제의 업데이트를 S_x에 적용하면 점 업데이트라는 사실을 알 수 있습니다.
- ✓ 따라서 처리해야 하는 업데이트와 쿼리는 다음과 같은 형태입니다.
  - S_x를 v로 변경한다.
  - $x \in [l, r]$에 대해, $\max_{y \geq x} S_y$의 합을 구한다.

- 초기 S의 값은 $N+1$개가 있고, 쿼리로 인해 최대 Q번 값이 변경됩니다.
- S의 값들의 역사를 시간 축을 세로로, 인덱스 축을 가로로 하여 나타내어 보면 $O(N+Q)$개의 직사각형으로 나타낼 수 있다는 것을 알 수 있습니다. 여기서 하나의 직사각형은 한 위치에 같은 값이 보관되어 있는 연속된 기간에 대응됩니다.
- 문제의 상황은 이 직사각형들을 자기 왼쪽까지 최대로 확장해 2d range max update를 한 뒤, 구간 합을 구하는 것과 같습니다.
- 이 문제는 Offline Dynamic Connectivity와 비슷한 형태의 분할 정복 알고리즘으로 해결할 수 있습니다. 정해와 큰 관련이 없으므로 자세한 설명은 생략합니다.

Subtask 5: $l_i = r_i$ (70점)

- ✓ 앞의 $O(N+Q)$ 개 직사각형 접근을 떠올려 봅시다.
- ✓ 인덱스 축으로 스위핑을 한다면, 세그먼트 트리 위에서 구간 max 업데이트를 하고, 구간 합을 구하는 형태의 문제처럼 생각할 수 있습니다.
- ✓ 문제를 이렇게 환원하고 나면 가장 기본적인 형태의 Segment Tree Beats를 이용해 $\mathcal{O}\left((N+Q)\log N\right)$에 해결할 수 있습니다.

Full Task (100점)

- ✓ 여기까지 왔다면 전체 문제는 **Subtask 5**와 같이 해결하되, 세그먼트 트리에 historic sum을 담당하는 부분을 추가하는 것으로 해결할 수 있습니다.
- ✓ historic sum은 세그먼트 트리에 누적되는 합을 관리하는 기법으로, 기본적으로는 현재 값을 a라 할 때 historic sum이 $at + b$ 꼴의 함수로 표현되는 것을 이용합니다.
- ✓ 다소 구현이 복잡하지만 historic sum을 Segment Tree Beats에 추가하면 $\mathcal{O}\left((N + Q) \log N\right)$에 해결할 수 있습니다.

Additional Note: 여담

- ✓ 기존에 MITIT 대회에 제안했다 떨어진 문제입니다.
- ✓ 에디토리얼의 풀이 말고도, BOJ 24506 blobpopcorn 문제에서 사용되는 것과 같은 세그먼트 트리를 이용해 $\mathcal{O}\left((N+Q)\log^2 N\right)$ 정도에 해결하는 풀이가 있다는 제보를 받았습니다. 흥미가 있다면 찾아보시기 바랍니다.

VIII. 크로스링크

`dial`
출제자의 의도 – **D5**
검수진 예측 – **D4**
실제 티어 – **D5**

- ✓ 제출 169번, 정답 17명 (정답률 15.385%)
- ✓ First Solve: **hos_lyric**, 117분

풀이 요약

- ✓ 네 변에서 한 점으로 도달하는 데 필요한 최단 거리를 구하는 것과 비슷한 느낌으로 Dial 알고리즘을 사용합니다.
- ✓ 여러 변에서 같은 점을 여러 번 사용할 때 이를 한 번만 세어야 합니다.
- ✓ 크로스링크의 형태를 관찰해 적은 횟수의 Dial 알고리즘만으로 답을 구할 수 있도록 알고리즘을 설계할 수 있습니다.

- 앞 세 서브태스크는 정해와는 크게 관련이 없으며, Subtask를 만들기 위해 Ad-hoc하게 설정하였습니다.
- 보통 서브태스크가 이렇게 만들어지는 이유는 적당한 부분 문제를 주기 어려워서, 즉 풀이에 대한 단서를 얻으면 곧바로 100점 풀이로 이어져서인 경우가 많고, 이 문제도 비슷한 유형에 해당합니다.
- **Subtask 1**은 모든 2^{NM} 개 경우를 시도해 보면 됩니다.
- **Subtask 2**는 답이 항상 $N + M - 1$입니다.
- **Subtask 3**은 홀수 행에서 가장 작은 비용들만 골라 더하면 됩니다.

Subtask 4: $N \leq 100, M \leq 100$ (64점)

- 문제의 목표는 네 변에 모두 맞닿는 연결성분을 최소 비용으로 만드는 것입니다.
- 변이 네 개 뿐이므로, 맞닿았는지의 여부를 비트마스킹으로 표현할 수 있을 것 같습니다.
- 하지만 이러한 상태들을 어떤 방식으로 효율적으로 합쳐야 할까요?

- $D[r][c][b]$를 칸 (r, c)를 포함하고 비트마스크 b에 해당하는 변이 모두 맞닿는 연결성분을 만드는 최소 비용이라고 정의합시다.
- 이때 $D[r][c][b]$를 구하는 과정은 먼저 점 b를 고르고, 인접한 4개의 칸에서 b의 부분집합들을 적당히 최소 비용으로 조합하는 것입니다.
- DP의 상태 전이가 굉장히 복잡해 보이지만, 작은 값부터 먼저 계산한다고 생각하면 최단 거리 알고리즘과 비슷한 방식으로 처리하면 됨을 알 수 있습니다. (엄밀히 말하면 최단 경로 문제는 아니지만, 거의 그대로 적용할 수 있습니다.)

- 핵심은 Dijkstra 알고리즘을 적용할 때, 매번 새로운 칸의 거리를 구할 때마다 인접한 네 칸을 모두 보며, 해당 칸들의 거리를 더 개선할 수 있는지 확인하는 것입니다.
- 기존 Dijkstra 알고리즘은 현재 점에서 가는 것만 고려해도 되지만, 이 문제에서는 최단 거리 문제가 아니기 때문에 모든 이웃을 통해 거리를 최적화하는 것을 고려해야 합니다.
- 이를 통해 $\mathcal{O}(NM \log NM)$에 문제를 해결할 수 있습니다. 구현에 따라 2^4 혹은 그 이상의 상수가 붙게 됩니다.

Subtask 5: $N \leq 700, M \leq 700$ (82점)

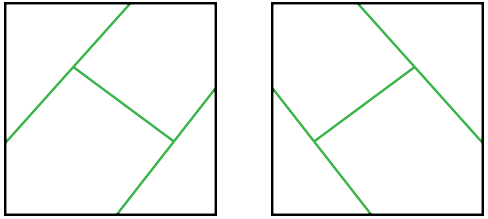
- ✓ 앞의 알고리즘은 시간 복잡도 자체만 본다면 충분히 동작할 것 같지만, 실제로는 상수가 꽤 크기 때문에 느립니다.
- ✓ 이 문제의 상황을 잘 생각해 보면, 각 칸의 가중치는 최대 35 입니다.
- ✓ 또한 문제의 최대 정답도 $35(N + M - 1)$ 임을 알 수 있습니다.
- ✓ 따라서 굳이 Dijkstra 알고리즘을 이용할 필요가 없고, 더 빠른 알고리즘을 이용할 수 있습니다.

- Dial 알고리즘은 0-1 BFS와 유사하지만, 사용하는 큐의 개수를 최대 가중치만큼 늘려 최대 가중치가 적을 때 최단거리를 빠르게 구할 수 있는 알고리즘입니다.
- 이 알고리즘을 사용하면 시간 복잡도의 로그 항이 제거되고, $\mathcal{O}\left(O(NM)\right.$에 문제를 해결할 수 있습니다.
- 단, 2^4 이상의 상수가 여전히 붙기 때문에 100점을 받기에는 충분하지 못할 가능성이 큽니다.

Full Task (100점)

- ✓ 여기서 더 이상 시간 복잡도를 줄이는 것은 불가능합니다.
- ✓ 따라서 100점을 받기 위해서는 상수를 더 줄여야 합니다.
- ✓ 여기서 상수를 2배 이상 줄이기 위해서는 조금 다른 접근법을 생각해 보아야 합니다.

- 잠시 현재의 풀이에서 벗어나, 크로스링크의 가능한 형태가 어떤 모양일지 생각해 봅시다.
- 정확히는, 네 가지 변이 어떤 모양으로 연결되어 있을지에 집중해 생각해 봅시다.
- 여러 가지 형태를 관찰하다 보면, 크로스링크가 아래 두 가지 형태 중 하나라는 추측을 하게 됩니다.



- 즉, 크로스링크의 형태는 먼저 인접한 두 변씩을 이은 뒤, 그 두 그룹을 다시 이은 형태여야 합니다.
- 증명은 크로스링크의 스패닝 트리를 생각하면 쉽습니다.
- 크로스링크를 격자 그래프로 나타냅시다. 즉, 크로스링크의 각 격자칸이 정점이 되고, 인접한 격자칸들이 간선으로 이어집니다.
- 이 그래프의 스패닝 트리는 네 변에 정점이 하나씩 있어야 합니다.
- 그렇다면, 스패닝 트리 상에서 이 네 정점을 둘씩 분할하는 간선이 적어도 하나는 있어야 합니다. 이 간선을 기준으로 나누어 생각하면 크로스링크는 항상 위 둘 중 하나의 형태여야 합니다.

- 두 형태는 대칭이므로, 첫 번째 형태를 만드는 최소 비용을 구해 봅시다.
- 먼저 네 변에서 Dial 알고리즘을 돌려 모든 칸으로 접근하는 최소 비용을 구합니다. 각각 $U[r][c], L[r][c], D[r][c], R[r][c]$ 라고 합시다.
- 다음으로, 윗 변과 왼쪽 변의 결과를 합쳐 $UL[r][c] := U[r][c] + L[r][c] - C[r][c]$ 로 정의합니다. $C[r][c]$ 를 빼는 이유는 이 칸이 두 번 세어졌기 때문입니다. 마찬가지로 $DR[r][c]$ 를 정의합니다.
- $UL[r][c]$ 를 시작 가중치로 정하고 다시 Dial을 돌려 UL 을 재계산합니다. 이는 앞에서 UL 을 단순히 U 와 L 의 합으로 더했을 때 두 번 더해진 점들을 제거하기 위해서입니다. 즉, U 와 L 이 합쳐진 후 추가된 점들은 한 번만 더하기 위해서입니다.
- 이후 $UL[r][c] + DR[r][c] - A[r][c]$ 의 최솟값을 구하면 됩니다.

- 이러한 방식으로 진행하면 Dial 알고리즘을 여섯 번만 돌려도 문제를 해결할 수 있게 됩니다.
- 상수가 크게 줄게 되어 $\mathcal{O}(NM)$에 전체 문제를 해결할 수 있습니다.

Additional Note: 시간 제한이 너무 작아요!

- ✓ 이 문제는 $O(NM)$ 또는 유사한 시간 복잡도의 다양한 풀이가 많아, 정답에 근접했더라도 조금 더 효율적인 풀이를 생각해볼 것을 위해 시간 제한을 의도적으로 작게 잡았습니다.
- ✓ 정해는 424 ms에 작동합니다만, 이 코드는 출제자가 문제의 특징을 완전히 이해하고 난 뒤 짠 코드로 실제 대회에서 참가자가 이 정도로 빠르게 작동하는 코드를 짜기는 어려웠을 것으로 추정됩니다.

- 정해보다 조금 비효율적인 코드도 적당히 좋은 점수를 주기 위해 사소한 N, M 의 차이에 서브태스크를 여러 개 두었으나, 참가자들 간의 코딩 스타일 차이로 같은 풀이를 구현해도 일부 참가자가 점수를 더 받는 경우가 발생했습니다.
- 결과적으로 작은 TL 로 인해 많은 참가자들이 대회 도중 불편을 겪으셨고, 이에 대해 사과의 말씀을 드립니다. 앞으로는 비슷한 방향의 문제 출제를 지양하도록 하겠습니다.

IX. 조화로운 사각형

ad_hoc
출제자의 의도 – **P4**
검수진 예측 – **P4**
실제 티어 – **P4**

- ✓ 제출 129번, 정답 26명 (정답률 20.155%)
- ✓ First Solve: **hos_lyric**, 62분

Subtask 1: $Q = 0$ (6점)

- ✓ 주어진 격자에서 조화로운 사각형의 수를 세면 됩니다.
- ✓ 반복문을 이용해 간단하게 해결할 수 있습니다.
- ✓ 다만, 어떤 2×2 칸이 조화로운 사각형인지 판별하는 부분의 구현이 까다로울 수 있습니다.
- ✓ 한 가지 간단한 방법은, 네 칸의 문자를 모두 `std::set`에 넣고, 그 크기가 4가 되는지를 확인할 수 있습니다.
- ✓ 조화로운 사각형의 수를 세는 데 걸리는 시간은 $\mathcal{O}(NM)$ 입니다.

Subtask 2: $N \leq 100, M \leq 100, Q \leq 100$ (13점)

- ✓ 업데이트의 수가 적습니다.
- ✓ 따라서, 각 업데이트를 $\mathcal{O}(NM)$에 처리하고, 답도 매번 $\mathcal{O}(NM)$에 구할 수 있습니다.
- ✓ 총 시간 복잡도는 $\mathcal{O}(QNM)$입니다.

Subtask 2: $N \leq 100, M \leq 100, Q \leq 100$ (13점)

- ✓ 업데이트의 수가 적습니다.
- ✓ 따라서, 각 업데이트를 $\mathcal{O}(NM)$에 처리하고, 답도 매번 $\mathcal{O}(NM)$에 구할 수 있습니다.
- ✓ 총 시간 복잡도는 $\mathcal{O}(QNM)$입니다.

Subtask 4: $r_{1i} = c_{1i} = 1, r_{2i} = N, c_{2i} = M$ (21점)

- ✓ 이 서브태스크에서는 모든 업데이트가 모든 격자 칸의 원소를 변경합니다.
- ✓ 이 서브태스크를 효율적으로 해결하려면 이 문제에서 가장 중요한 관찰이 필요합니다.

- 어떤 2×2 사각형의 네 칸 모두가 업데이트에 포함되었다고 가정해 봅시다.
- 이 경우, 이 업데이트는 이 사각형이 조화로운 사각형인지의 여부를 바꾸지 않습니다.
- 증명을 위해, 네 가지 원소를 두 자리 이진수로 생각합시다. 각 원소는 `00`, `01`, `10`, `11` 중 하나로 생각할 수 있고, 각 업데이트는 첫째 자리 혹은 둘째 자리를 반전시킵니다.

- 어떤 비트를 반전시키더라도, 초기에 네 가지 (00, 01, 10, 11) 원소가 모두 있었다면 반전 이후에도 네 가지 원소가 모두 존재하게 됩니다.
- 또한, 업데이트는 역방향으로 시행해도 똑같으므로, 이것이 필요충분임을 알 수 있습니다.
- 따라서 업데이트는 답을 전혀 바꾸지 않으며, 초기의 답이 모든 업데이트 이후의 답이 됩니다.

Subtask 3: $r_{1i} = r_{2i}$ (52점)

- ✓ 이 서브태스크의 경우 각 업데이트가 한 행 위의 격자칸만을 변경합니다.
- ✓ 따라서 답이 변경될 가능성이 있습니다.
- ✓ 하지만, 영향을 받는 2×2 사각형의 개수가 $\mathcal{O}(M)$ 개로 매우 적습니다.
- ✓ 해당 범위 안에서만 원소 종류를 바꾸고 조화로운 사각형 개수를 변경하면 됩니다.
- ✓ $\mathcal{O}(NM + MQ)$ 시간 복잡도에 해결할 수 있습니다.

Full Task (100점)

- ✓ **Subtask 3**, **Subtask 4**와 비슷한 방식을 적용해 봅시다.
- ✓ 앞에서 업데이트는 완전히 포함되는 2×2 사각형의 조화로움 여부를 바꾸지 않음을 관찰했습니다.
- ✓ 따라서 각 업데이트가 변경하는 부분은 경계에 걸치는 2×2 사각형의 조화로움 여부뿐임을 알 수 있습니다.
- ✓ 이 부분에 걸친 사각형은 업데이트당 $\mathcal{O}(N + M)$ 개뿐이므로, 잘 업데이트하면 $\mathcal{O}(NM + Q(N + M))$에 문제를 해결할 수 있을 것처럼 보입니다.

- 하지만 한 가지 문제점이 있습니다.
- 겉으로 보았을 때는 업데이트가 내부에 포함되는 2×2 사각형의 조화로움 여부에 영향을 미치지 않으므로, 내부 원소를 무시해도 될 것 같습니다.
- 하지만 경계는 업데이트하고 그 안을 업데이트하지 않으면, 이 부분은 동기화가 더 이상 유지되지 않게 됩니다.
- 따라서 향후 쿼리에서 이 격자칸들이 경계 부분에 위치할 때, 답이 부정확해질 가능성이 있습니다.
- 따라서 직사각형의 업데이트 영역 내부의 변화를 무시할 수는 없습니다.
- 그렇다면 이 부분을 어떻게 효율적으로 고려해야 할까요?

- 답은 주변 칸과의 상대적인 차이를 고려하는 것입니다.
- 모든 격자칸에 대해, 그 칸의 원소를 저장하지 말고, 그 왼쪽 칸과 오른쪽 칸의 원소 코드 XOR 값을 저장해 둡니다.
- 예를 들어, 현재 위치의 원소 코드가 01 이고, 현재의 왼쪽/위쪽 위치 원소 코드가 각각 11, 10 이면, 그 XOR 값인 10과 11을 저장합니다.
- 이렇게 하면, 한 2×2 사각형이 조화로울 조건은 그 왼쪽 위 칸과 다른 칸들의 XOR 값에 00, 01, 10, 11 이 모두 한 번씩 등장하는 것으로 바꾸어 생각할 수 있습니다.

- 중요한 점은, 직사각형 업데이트 시 그 내부의 XOR 차이 값들이 영향을 받지 않는다는 것입니다.
- 따라서 이 XOR 값은 경계 부분만 업데이트를 해 주어도 충분합니다.
- 이렇게 하면 문제를 $\mathcal{O}\left(NM + Q(N+M)\right)$ 시간 복잡도에 해결할 수 있습니다.

Additional Note: 시간 제한이 너무 작아요!

- ✓ 정해는 132 ms에 작동합니다.
- ✓ XOR 등 비트 연산을 적극적으로 활용하면 실행 시간을 크게 줄일 수 있습니다.
- ✓ TL을 정해의 10배 이상으로 주었기 때문에 전혀 걱정을 하지 않았지만, 대회 중 TLE로 고생하시는 분들이 생각보다 많았습니다.
- ✓ 앞의 크로스링크 문제와는 다르게 세팅이 잘못되었다고 생각하지는 않습니다. 대회 시간이 길고 최적화 방법도 많기에 적은 노력으로도 대다수의 참가자가 시간 제한 안에 작동하는 AC 코드를 작성할 수 있었을 것이라고 생각합니다.

X. 잃어버린 섬

`parity`
출제자의 의도 – **B1**
검수진 예측 – **S1**
실제 티어 – **S5**

- ✓ 제출 400번, 정답 251명 (정답률 64.750%)
- ✓ First Solve: **coxld**, 2분

풀이 요약

- ✓ M 이 짝수인 경우, 모든 수가 짝수 번 등장해야 가능합니다.
- ✓ M 이 홀수인 경우, 홀수 번 등장하는 수의 종류가 N 가지 이하여야 가능합니다.
- ✓ 격자에 각 수가 등장하는 횟수를 counting array를 통해 $\mathcal{O}(NM)$ 에 세면 문제를 해결할 수 있습니다.

- 이 문제에서는 모든 A_{rc}를 조건을 만족하도록 재배열할 수 있는지를 묻고 있습니다.
- 따라서 초기 배치에서 A_{rc} 값들이 어디에 위치해 있는지는 중요하지 않습니다.
- 중요한 것은 각각의 A_{rc} 값이 총 몇 개씩 있는지뿐입니다.
- 각 값의 개수에 중점을 두고 문제를 해결해 봅시다.

Subtask 4: M 은 짝수 (30점)

- M 이 짝수인 경우를 살펴봅시다.
- 이 경우, 모든 칸 (r, c) 에 대해, 좌우대칭인 다른 칸 $(r, M + 1 - c)$ 가 존재함을 알 수 있습니다.
- 이 두 칸에 적힌 수는 항상 같아야 합니다.
- 따라서, NM 개의 칸을 두 칸씩 묶을 수 있고, 묶인 두 칸에 있는 수는 같아야 합니다.
- 이는 각각의 수가 등장한 횟수가 항상 2 의 배수, 즉 짝수여야 함을 나타냅니다.

- 모든 수가 짝수 번 등장한다면 항상 재배열이 가능할까요?
- 격자를 좌우대칭으로 만들어야 하므로, 왼쪽 반절을 결정하면 오른쪽 반절이 자동결정됩니다.
- 왼쪽 반절에 각각의 수를 정확히 반만큼 임의로 배치합시다.
- 오른쪽 반절을 왼쪽 반절과 대칭이 되도록 채우면, 재배열이 항상 가능함을 알 수 있습니다.
- 따라서, M 이 짝수일 경우, 모든 수가 짝수 번 등장하는 것이 좌우대칭으로 재배열이 가능할 필요충분조건임을 알 수 있습니다.
- 각 수가 몇 번 등장하는지를 counting array를 이용해 세면, $\mathcal{O}(NM)$ 에 문제를 해결할 수 있습니다.

Subtask 3: $N = 1$ (50점)

- ✓ 이번에는 $N = 1$인 경우를 살펴봅시다.
- ✓ 앞에서 이미 M이 짝수인 경우를 풀었으니, M이 홀수인 경우만 생각해 봅시다.
- ✓ M이 홀수인 경우에도, 대부분의 칸 $(1, c)$는 좌우대칭인 다른 칸 $(1, M + 1 - c)$와 묶을 수 있습니다.
- ✓ 그러나 가운데 칸 $\left(1, \frac{M+1}{2}\right)$는 자기 자신과 좌우대칭이므로, 묶이지 않습니다.

- 가운데 칸 $\left(1, \frac{M+1}{2}\right)$ 에 등장하는 수는 다른 칸에서 등장하지 않는다면 정확히 1 번 등장합니다.
- 다른 칸에서 등장한다면, 두 칸씩 묶인 그룹에 동시에 등장해야 하므로, 등장 횟수가 항상 홀수여야 함을 알 수 있습니다.
- 반면, 가운데 칸 $\left(1, \frac{M+1}{2}\right)$ 에 등장하지 않는 수는 **Subtask 4**와 같은 이유로 짝수 번 등장해야 합니다.
- 이를 통해, 정확히 한 가지 수는 홀수 번 등장하고, 나머지 수가 짝수 번 등장해야 함을 알 수 있습니다.

- 이 조건을 만족하면 항상 좌우대칭으로 배치가 가능할까요?
- 홀수 번 등장하는 유일한 수를 가운데 칸에 배치했다고 생각합시다.
- 여기서 가운데 열을 격자에서 제거한다면, 배치해야 할 수들은 모두 짝수 번 등장하고, 열의 개수는 짝수가 됩니다.
- 이것은 **Subtask 4**와 정확히 같은 상황입니다.
- 따라서, 항상 배치가 가능합니다.
- 이 경우 역시 $\mathcal{O}(NM)$에 해결할 수 있습니다.

Full Task (100점)

- 이번에는 가운데 열에 칸이 N 개 존재합니다.
- 홀수 번 등장하는 수의 종류가 o 개라고 할 때, 좌우대칭으로 재배열이 가능할 조건이 무엇일까요?

- 먼저 $o \leq N$ 이어야 함을 알 수 있습니다.
- 어떤 수가 홀수 번 등장할 수 있으려면, 가운데 열에 적어도 한 번 등장해야 합니다.
- 그렇지 않으면 좌우에 같은 횟수만큼 등장하므로, 짝수 번 등장할 수밖에 없습니다.
- 가운데 열에 등장할 수 있는 수는 최대 N 가지이므로, $o \leq N$ 이어야 합니다.

- 이제 이 조건을 만족하면 항상 좌우대칭이도록 재배치가 가능함을 증명합니다.
- 먼저 홀수 번 등장하는 o개의 수를 가운데 열에 한 개씩 넣습니다.
- 가운데 열에 남은 칸에는 등장 횟수가 충분히 많이 남은 임의의 수를 하나 고른 뒤, 2회 넣는 과정을 반복합니다.
- **Subtask 3**과 마찬가지로 가운데 열을 제거하고 생각하면, M은 짝수가 되었고 남은 수들은 모두 짝수 번 등장해야 하므로 **Subtask 4**의 상황으로 변했습니다.
- 이렇게 전체 문제를 $\mathcal{O}(NM)$에 해결할 수 있습니다.

Additional Note: 가운데 열을 항상 채울 수 있는 이유

- 전체 수의 등장 횟수의 합은 NM 입니다.
- M 이 홀수이므로, NM 의 홀짝성은 N 의 홀짝성과 같습니다.
- 한편, 전체 수의 등장 횟수의 합은 각각의 수의 등장 횟수의 합과 같은데, 이들 중 홀수는 정확히 o 개이므로, NM 의 홀짝성은 o 의 홀짝성과도 같습니다.
- 따라서 홀수 번 등장하는 수들을 가운데 열에 넣고 나면, 가운데 열에는 $N - o$ 개의 칸이 남는데, 이 수는 짝수입니다.
- 즉 남은 칸들은 등장 횟수가 충분한 수들을 두 개씩 넣는 것으로 채울 수 있습니다.

1. 타임위버

ad_hoc
출제자의 의도 – **R5**
검수진 예측 – **D1**
실제 티어 – **R5**

- ✓ 제출 156번, 정답 4명 (정답률 2.564%)
- ✓ First Solve: **parkky**, 271분

풀이 요약

- 기본 격자는 1과 0을 번갈아 채웁니다.
- 앞 89개 칸을 인접한 두 칸 중 하나는 색칠한다는 조건 하에 자유롭게 색칠하고, 90번째 칸을 색칠한 뒤, 마지막 10행은 행 복원을 위해 사용합니다.
- 이 조건만으로도 대부분의 경우 열 복원이 가능합니다.
- 열 복원이 되지 않는 케이스는 DP를 통해 제거하거나, 제켄도르프 분할 아이디어를 통해 복원합니다.

- ✓ $L = 1$ 부터 62 까지의 모든 L 값에 대해 점수가 부여되어 있습니다.
- ✓ 대회가 끝난 직후 $L = 63$ 인 풀이도 발견되었으며, 이 이상 최적화하기는 매우 어려울 것으로 전망됩니다.

Subtask: $L = 55$ (70점)

- ✓ 먼저 행 삭제나 열 삭제가 일어나지 않는다고 해 봅시다.
- ✓ 어떤 기본 격자가 가장 효율적일까요?

- 행이나 열 삭제가 일어나지 않으므로, 색칠한 모든 수가 보존됩니다.
- 따라서, 색칠한 수들을 순서대로 모두 알 수 있을 것이고, 결과적으로 격자의 모든 100개 수를 순서대로 나열한 문자열의 모든 부분 수열을 전달할 수 있습니다.
- 이 경우, $101010\cdots$ 와 같이 1과 0을 번갈아 나열하는 것이 가장 효율적이고, 만들 수 있는 결과의 수가 피보나치 수와 비슷합니다.

- ✓ 즉, 기본 격자는 다음과 같은 형태일 때 최적입니다. 행 삭제나 열 삭제가 있더라도, 비슷한 형태가 최적일 것이라고 추측할 수 있습니다.

1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0

대전제

- ✓ 문자열 $1010\cdots1010$의 모든 부분수열이 가능한 격자의 한 상태에만 대응될 수 있도록 색칠해야 합니다.
- ✓ 따라서 다음과 같은 조건을 추가합니다: 100개의 칸 중 색칠되지 않은 연속된 두 칸이 존재하면 안 됩니다.
  - 이때 100칸의 순서는 행 순서대로, 행이 같으면 열 순서대로 생각합니다.
  - 마지막에 2개 이상의 칸이 색칠되지 않은 것은 괜찮습니다.
- ✓ 이를 통해, 100개 칸을 순서대로 읽은 문자열이 주어졌을 때 원래 배치를 알 수 있습니다.
- ✓ 구체적으로는, 현재 색칠한 칸 이후에 등장하는 두 개 칸 중, 다음 문자(0 또는 1)과 일치하는 칸을 색칠하면 됩니다.

- 100개 칸을 위 조건대로 색칠하는 방법의 수를 구하는 DP를 생각해 봅시다. 이때 점화식의 꼴은 다음과 같습니다.

$$DP[i] = DP[i-1] + DP[i-2]$$

- 위 DP를 조금 변형하면 사전순으로 k번째로 작은 색칠 방법이나, 특정 색칠 방법이 사전순으로 몇 번째로 작은지 등을 알아낼 수 있습니다.
- 이를 통해 행이나 열 삭제가 없는 경우 문제를 해결할 수 있으며, 이 경우의 L은 약 70 정도가 될 것입니다. (약간의 오차가 있을 수 있습니다.)

- 이제부터 행 삭제와 열 삭제가 있는 경우에 어떻게 대처해야 하는지를 생각해 보겠습니다.
- 구체적으로는, 행 삭제를 대비해 10 행을 이용하고, 열 삭제를 대비해 1 행을 이용합니다.
- 따라서 정보를 자유롭게 배치할 수 있는 칸은 10 행 중 8 행뿐이 됩니다.

열 삭제

- ✓ 1 열의 모든 10 개 칸을 칠해 둡니다.
- ✓ 만약 열 삭제가 일어났다면, 첫 10 개 문자 중 정확히 하나가 ? 로 바뀝니다.
- ✓ 이때 ? 의 위치를 통해 어떤 열이 삭제되었는지 알 수 있습니다.

행 삭제

- 한 행을 색칠하는 경우의 수는 $F_{12} = 144$ 가지가 있습니다.
- 2 행부터 9 행까지를 자유롭게 색칠한다고 생각합시다. 단, 모종의 이유로 9 행 10 열의 칸은 항상 색칠하는 것으로 고정합니다.
- 한 행을 색칠하는 144 가지 방법에 대해 $[0, 143]$ 구간 내의 정수 하나씩을 대응시킵시다.
- $r \in [2, 9]$ 에 대해 x_r 을 r 행을 칠한 방법에 대응되는 정수라고 정의합니다.
- 10 행의 경우 $r_2 + r_3 + \cdots + r_{10} \equiv 0 \pmod{144}$ 가 되는 r_{10} 을 골라, 대응되는 방법대로 칠합니다.

행 삭제

- 한 행을 색칠하는 경우의 수는 $F_{12} = 144$ 가지가 있습니다.
- 2 행부터 9 행까지를 자유롭게 색칠한다고 생각합시다. 단, 모종의 이유로 9 행 10 열의 칸은 항상 색칠하는 것으로 고정합니다.
- 한 행을 색칠하는 144 가지 방법에 대해 $[0, 143]$ 구간 내의 정수 하나씩을 대응시킵시다.
- $r \in [2, 9]$ 에 대해 x_r 을 r 행을 칠한 방법에 대응되는 정수라고 정의합니다.
- 10 행의 경우 $r_2 + r_3 + \cdots + r_{10} \equiv 0 \pmod{144}$ 가 되는 r_{10} 을 골라, 대응되는 방법대로 칠합니다.

Full Task: $L = 62$ (100점)

- ✓ 두 가지 풀이가 있습니다.
- ✓ 쉬운 풀이와 어려운 풀이가 있는데, 양쪽 다 보겠습니다.

직관

- ✓ 70점 풀이에서 격자칸을 조금 낭비하고 있다는 생각이 듭니다.
- ✓ 구체적으로, 행 삭제 복원을 위한 10행은 그렇다 쳐도, 열 삭제 복원을 위해 1행을 전부 쓰는 건 너무 비효율적으로 보입니다.
- ✓ 다른 방법을 찾아봅시다.

- 1 행 부분을 사소하게 최적화하는 여러 가지 방법들이 있습니다.
- 이 방법들은 대체로 100 점 풀이에 필요한 일부 관찰을 했지만, 필요한 모든 관찰을 하지 못했을 때 떠올릴 만한 것들입니다.
- 이런 관찰들을 이용해 $L = 57$, 또는 그보다 조금 더 점수를 올릴 수 있습니다.
- 그러나 여기서 100 점 풀이로 바로 향하려면 다른 발상이 필요합니다.

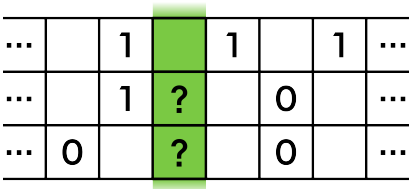
- 핵심은 1 행의 열 복원 메커니즘을 완전히 제거하는 것입니다.
- 하지만, 1행이 없다면 열 복원을 하는 것이 불가능해 보입니다.
- 이게 어떻게 가능할지 생각해 봅시다.

1행 없이 열 복원을 하는 방법

- 놀랍게도 대부분의 경우, 1 행이 없이도 열 복원이 가능합니다.
- 만약 지워진 숫자가 없다면, 굳이 열 복원을 하지 않아도 됩니다.
- 지워진 숫자가 있다면, 그중 첫 번째 숫자를 생각해 봅시다.
- 우리의 색칠 규칙에 의해, x 번 칸 $(1 \leq x \leq 100)$이 색칠되어 있다면, $x+1$ 번 또는 $x+2$ 번 칸이 색칠되어 있어야 합니다.
- 따라서, 지워진 숫자가 0 이냐 1 이냐에 따라, 실제 격자에서 지워진 열이 어떤 열인지도 알 수 있게 됩니다.

- 그리고, 기본 격자의 같은 열에는 항상 같은 수가 적혀 있게 되므로, 실질적으로 나올 수 있는 격자의 최종 상태는 둘 중 하나입니다.
- 하지만 여전히 두 개의 후보가 있다는 점이 마음에 걸립니다.
- 이 둘 중 원래 상태가 무엇이었을지 어떻게 결정할 수 있을까요?
- 어떻게 결정할 수 있을지를 생각하기보다, 어떤 경우에 구분이 불가능할지를 생각해 봅시다.

- 지워진 열을 c열이라 하고 $(1 \leq c \leq 10)$, 각 행의 c열의 상태에 집중해서 살펴봅시다.
- 특정 행의 c열이 색칠되지 않으면 `'.'`, 색칠이 되어 있고 양옆의 인접한 숫자가 같으면 `'='`, 색칠이 되어 있고 양옆의 인접한 숫자가 다르면 `'!'`로 표기한 길이 10의 문자열 S를 생각합시다.



- 예를 들어 위 세 행의 경우 $S =$ `".!="` 입니다.

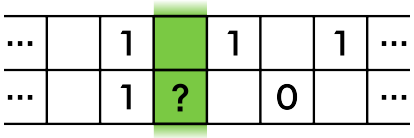
- 이제 S를 통해 케이스를 나누며, 어떤 경우 열 복원이 가능한지 생각해 봅시다.

모든 문자가 '.' 인 경우

- 모든 문자가 '.' 인 경우 후술할 일부 경우에 한해 어떤 열이 지워졌는지 확정지을 수 없는 경우가 있을 수 있습니다.
- 하지만 모든 문자가 '.' 란 뜻은 모든 숫자가 지워지지 않고 복원되었다는 뜻입니다.
- 따라서 굳이 복원을 하지 않아도 모든 정보를 가지고 있고, 고려하지 않아도 됩니다.

'.'가 존재하는 경우

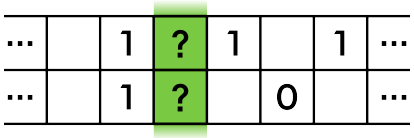
- 풀이의 인코딩에서는 항상 인접한 두 칸 중 한 칸은 색칠합니다.
- 따라서 '.'가 S에 존재한다면, 그 왼쪽과 오른쪽 두 칸이 모두 사용되어야 함을 알 수 있습니다.
- '.'가 아닌 이전/이후 행과 비교해 공백 간격을 알아낸다면, '.' 바로 이전/이후 문자가 제거되지 않음을 이용해 지워진 열을 특정할 수 있습니다.
- 글로만 설명하면 이해가 어려우니 예시를 보겠습니다.



- 위 그림의 두 번째 행에서 '0' 인지 '1' 인지 알 수 없는 지워진 숫자 '?' 가 등장합니다.
- 그러나, 만약 이 숫자가 '1' 이였을 경우, 간격을 통해 윗 줄의 '1' 역시 지워졌어야 함을 추론할 수 있습니다.
- 실제로는 지워지지 않았으므로 모순입니다. 따라서 지워진 숫자가 '0' 임을 특정할 수 있습니다.
- 결론적으로 S에 '.' 가 한 글자라도 존재하면 위 두 케이스 중 하나에 해당해, 지워진 문자를 복원할 필요가 없거나, 복원하는 전략이 존재합니다.

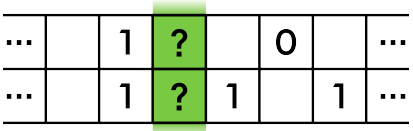
'='와 '!'가 둘 다 존재하는 경우

- ✓ 이제 남은 경우는 S의 모든 문자가 '=' 또는 '!'인 경우입니다.
- ✓ 두 가지 문자 모두 존재하는 경우를 보겠습니다. 이 경우 반드시 어떤 두 인접한 행이 존재하여 한쪽은 '=', 한쪽은 '!'입니다.



- 윅 그림과 같이 S에 "=!"이 존재하는 경우를 봅시다.
- 이 경우 '?' 사이의 간격이 핵심 단서가 됩니다.
- '='의 경우, '?'의 값이 '0'이냐 '1'이냐에 따라 간격이 달라집니다. 양옆 문자와 같은 경우 양옆 문자의 거리가 네 칸이 되고, 다른 경우 두 칸이 되기 때문입니다.
- 따라서 인접한 두 '?' 문자의 간격을 이용하면 지워진 숫자가 무엇인지 알아낼 수 있습니다.

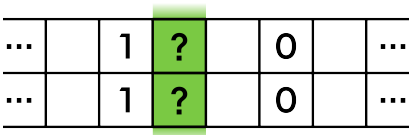
- 같은 방식으로 S에 "=="가 존재하는 경우에도 지워진 숫자를 알아낼 수 있습니다.
- 이 시점에서 남은 경우의 수는 "!!…!=" 또는 "!!…!" 꼴 뿐입니다.



- 위 그림과 같이 S에 "!="가 존재하는 경우를 봅시다.
- 이 경우에는 지워진 문자에 따라 윗 줄 '?'와 아랫줄 '?'의 간격이 변하지는 않습니다.
- 하지만 마지막 '?' 이후의 간격이 변하게 되므로, 끝나는 시점의 마지막 두 칸이 넘치는지 부족한지의 여부를 통해 복원이 가능합니다.

'!'만 존재하는 경우

- ✓ 이 경우 구분이 굉장히 어렵습니다.
- ✓ 구체적으로, $S =$ "!!…!"라는 것은 모든 행이 아래 그림과 같은 상황이라는 뜻입니다.



이 경우 '?'가 있는 열이 한 칸 더 오른쪽이라도 아무 문제가 생기지 않습니다.

- 이 문제를 해결하는 방법은 두 가지가 있습니다.
- 아름다운 풀이와 복잡한 풀이가 있습니다.
- 복잡한 풀이 먼저 설명합니다.

Solution

- 모든 가능한 인코딩 방법을 DP로 세려고 시도해 봅시다.
- 단, 위와 같이 $S = "!!\cdots!"$ 인 경우는 제외하고 셉니다.
- 이렇게 하면 세어진 모든 경우는 복원이 항상 가능함을 보장할 수 있습니다.
- 비트마스크 DP와 비슷한 방식으로 진행이 가능하며, 제거되는 경우가 0.1% 정도뿐임을 알 수 있게 됩니다.
- DP를 이용해 K 번째 인코딩을 찾기와 특정 인코딩이 몇 번째인지 알기를 빠르게 할 수 있으면, 문제를 해결할 수 있습니다. 자세한 과정은 생략합니다.

- 이제 조금 더 깔끔한 풀이를 보겠습니다.
- 열 복원의 문제점을 해결하기 위해 아직 쓰지 않은 정보가 있습니다.
- 그것은 바로 10번 행의 복원 조건입니다. 즉, $r_1 + r_2 + \cdots + r_{10} \equiv 0 \pmod{144}$라는 정보를 아직 쓰지 않았습니다.
- 문제의 케이스에서 지워진 문자가 `'0'`, `'1'`인 경우 두 개 모두 이 조건을 뚫을 확률은 많이 낮아 보입니다. (기존의 0.1% 풀이에서 약 $1/144$ 확률로 성립되는 조건이 하나 추가되었으므로, 대략 10^{-5} 정도로 추측할 수 있습니다.) 따라서 이 조건까지 맞는 경우를 믿어 봅시다.

- 위와 같이 제출하면 놀랍게도 정답을 받습니다.
- 틀릴 확률이 작으니 데이터가 약한 걸까요?
- 출제자도 이렇게 생각했지만, 놀랍게도 이 풀이는 정당한 풀이가 됩니다. 지금부터 그 증명을 살펴보겠습니다.

Solution Proof

- 결국 문제가 되는 케이스는 모든 행에서 지워진 숫자 주변이 "0?1" (또는 대칭)인 경우뿐입니다.
- 지워진 숫자가 '0' 인지 '1' 인지에 따라 $r_1, \ldots, r_{10}$ 이 어떻게 변하는지를 살펴봅니다.
- Zeckendorf의 정리에 의해, 지워진 숫자가 '0' 에서 '1' 로 변할 경우, 모든 r_i 값이 특정한 F_i 만큼 변화합니다. 여기서 i는 지워진 열의 번호와 관련이 있는 값이 됩니다.
- 따라서, '0' 이 지워진 경우와 '1' 이 지워진 경우 모두 말이 되려면, $10F_i \equiv 0 \pmod{144}$ 인 i가 존재해야 합니다. 이런 i는 작은 범위 내에서 존재하지 않습니다.
- 따라서, 단순히 10 행이 조건을 만족하는지 검사하는 것만으로도 두 경우를 구분할 수 있습니다.

$L = 63$의 풀이

- 대회가 끝나고 $L = 63$의 풀이가 발견되었습니다.
- 핵심 아이디어는 위의 복잡한 풀이를 그대로 사용하되, r_i의 정의를 사소하게 바꾸는 것입니다.
- 생각해 보면, 각 행에서 몇 칸이 색칠되었는지를 아는 것은 그리 어렵지 않습니다.
- 따라서 r_i를 가능한 모든 144개 배치에 대해 인덱싱하지 말고, 색칠된 칸이 같은 경우 내에서만 인덱싱합니다.
- 이렇게 하면 모듈로를 144에서 56으로 줄일 수 있고, 가능한 인코딩 개수를 조금 더 늘릴 수 있습니다. 비슷한 DP를 통해 2^{63}가지 이상의 경우를 셀 수 있습니다.

2. 빛의 유적

`metapuzzle`
출제진 의도 – **Unratable**

- ✓ 제출 45번, 정답 8명 (정답률 20.000%)
- ✓ First Solve: **playsworld16**, 386분

- ✓ 메타퍼즐인 만큼 당연히 풀이 과정에 대한 힌트를 올리진 않습니다.
- ✓ 대신 이 공간에는 대회 개최 후기를 간략히 적어볼 예정입니다.

- ✓ BCF 2025의 개최 아이디어는 2024년 1월에 처음 떠올렸습니다.
- ✓ 개인 대회를 처음 열게 된다면 꼭 메타퍼즐을 넣고 싶다는 생각이 있었고, 거기에 맞게 대회 테마를 고안하는 데 꽤나 오랜 시간이 걸렸습니다.
- ✓ 2024년 3월경 문제 수를 12개로 확정지었고, 2024년 7월경까지 약 5개 정도의 문제 아이디어가 나왔습니다.

- ✓ 하지만 다른 대회 출제 기회들이 들어왔고, 그 과정에서 BCF를 위해 아껴 둔 일부 문제 아이디어들이 다른 대회를 위해 사용되었습니다.
- ✓ LGCPC에 출제된 BOJ 32126 돌고래 사진, UCPC에 출제된 BOJ 34091 ChannelTalk이 여기에 해당합니다.

- 다른 대회에 문제를 낼 때마다 부족한 수를 충원하기 위해 문제를 더 만들어야 했습니다.
- 문제를 만들다 보니 격자 컨셉 문제들이 많았습니다.
- 이렇게 된 김에 아예 대회의 전체 컨셉을 격자로 잡으면 좋겠다는 생각이 들었습니다.
- 또한 메타퍼즐도 격자를 이용해 만들자는 생각을 하게 되었습니다.

- 처음에는 메타퍼즐을 조금 더 클래식한 방향으로, 격자 위에 다양한 그림을 그려 놓고 푸는 퍼즐로 계획했습니다.
- 그러나 제 지능의 한계로 복잡한 퍼즐을 만드는 데 한계가 있었고, 그다지 재밌다고 느끼지도 못했습니다.
- 2025년 겨울쯤에 메타퍼즐을 제외하고 대회에 사용할 모든 문제의 아이디어가 완성되었습니다.
- 5월 말쯤 한국에 돌아온 뒤 문제 세팅을 시작했고, 일주일 정도 뒤 메타퍼즐을 제외한 문제들의 세팅이 끝났습니다.

- 이맘때쯤 대회 제목에 대한 고민도 있었습니다.
- 제가 이름 짓기를 정말 못하는데, 대회 이름을 결정하는 것도 꽤나 힘들었습니다.
- 주변에서 많은 추천을 받은 제목은 당연히 '브루컵'이었지만, 저는 -컵으로 끝나는 대회명을 피하고 싶었습니다. 예전에 오렌지컵을 연 적도 있긴 한데, 뭔가 최근에는 양산형 대회명이라는 느낌이 들었습니다.
- 결국 BCF로 제목을 정했습니다. 공식적으로 무엇의 약자인지 정해지진 않았지만, 여러분이 다양한 추측을 하시는 것을 봤습니다.

- 비슷한 시기 메타퍼즐의 형식에 대해 많은 고민이 있었습니다.
- 앞서 메타퍼즐이 등장한 다른 대회들도 많이 참고를 해 보았습니다. 대표적으로 전시관과 키파컵 등을 참고하였습니다.
- 전시관의 경우 앞 문제들을 해결해야지만 메타퍼즐을 풀 수 있다는 점에서 진입장벽이 조금 높아 보였습니다. 특히 대회 문제들의 전반적인 난이도가 높게 예정된 상황이라 이런 형식으로 가고 싶지 않았습니다.
- 키파컵의 경우 반대로 난이도가 너무 쉽다고 느꼈고, 그것보다는 조금 도전적이고 재미있게 생각할 만한 컨셉을 원했습니다.
- 대회 문제들의 컨셉이 격자라는 데 착안하여, 고전적인 격자 퍼즐을 만들어 보기로 했습니다.

- 처음에는 앞 열한 문제와 비슷한 컨셉을 가진 격자 퍼즐 기믹을 찾아보려고 시도했습니다.
- 하지만 연결이 나름 쉽지 않았습니다. 그러던 중 현재의 메타퍼즐 기믹을 생각하게 되었습니다.
- 또한 지금까지 생각해오던 대회 문제들의 스토리가 있었는데, 이것을 부각하기 위해 메타퍼즐을 pygame 형식으로 제작하기로 했습니다.
- 퍼즐 기믹과 해법을 제작하는 데는 3일 정도가 걸렸습니다.

- ✓ 이제 남은 것은 메타퍼즐을 실제로 pygame으로 구현하는 것뿐이었습니다.
- ✓ 저는 대략 1달 정도면 모든 작업이 끝날 거라고 예상했지만, 실제로는 대회 직전까지 약 2달 반 동안 진행되었습니다.
- ✓ 메타퍼즐의 코딩, 텍스트 작업, 그림까지 많은 부분을 ChatGPT에 의존했습니다.

- 텍스트의 경우 ChatGPT의 초안을 제가 다듬었고, mjhmjh1104가 더 풍성하게 작업해주어 현재의 형태가 되었습니다.
- 그림의 경우 전부 DALL-E가 그렸습니다. 제가 그림을 잘 그리지 못해 유일한 방안이었고, 매우 아쉽습니다.
- 코딩은 ChatGPT를 이용한 바이브 코딩으로 70% 정도, 제가 30% 정도 짠 것 같습니다. ChatGPT 버전이 업그레이드되면서 바이브 코딩이 한결 편해지긴 했는데, 스레드가 길어지면서 오류가 많아진 부분들이 있었습니다. 이런 경우를 대비해 ChatGPT의 코드를 그대로 복사 붙여넣기 하는 것이 아니라 대부분 제가 일일히 타이핑하면서 코드의 전체적인 개요를 이해하려고 노력했고, ChatGPT 사용이 불가능할 때에는 제가 직접 코딩한 후 나중에 GPT를 이용해 코드를 재정리했습니다.

- 메타퍼즐을 만들면서 들었던 한 가지 생각은 음악을 직접 작곡해 넣고 싶다는 생각이었습니다.
- 다만 시간이 조금 촉박해 cermix의 도움을 받아 음악 작업을 끝냈습니다.
- pygame 완성본에는 음소거 기능을 넣지 않았는데, 대회 중 다른 음악을 듣고 싶어 mute 기능에 대해 질문하신 분들이 상당히 많았습니다. 작곡가로써 조금 슬프지만 이해는 됩니다.
- 만약 나중에 비슷한 형식의 메타퍼즐을 만들게 된다면 이런 부분까지 고려해야겠다고 느꼈습니다.

- ✓ 대회가 끝나고 생각보다 많은 분들이 메타퍼즐을 풀어서 놀랐습니다.
- ✓ 역시 CP 평균 지능은 꽤나 높은 것 같습니다.
- ✓ 저는 약 3~5분 정도가 풀어주실 거라고 예상했는데 총 8분이 푸셨습니다.
- ✓

- 대회를 성공적으로 마칠 수 있게 해 주신 훌륭한 검수진 분들, 그리고 참가자 분들께 감사드립니다.
- 대회를 개최하기 위해 정말 많은 시간과 노력이 필요했습니다. 특히 80% 이상의 시간은 마지막 메타퍼즐을 만들기 위해 사용한 것 같습니다.
- 2026년에는 BCF가 열리지 않을 가능성이 있으며, 열리더라도 시간이 부족해서 훨씬 더 작은 규모로 열어야 할 것 같습니다.
- 향후 열리는 BCF 대회에도 많은 참가 부탁드립니다.
- 끝까지 읽어주셔서 감사합니다.