한양대학교 ERICA 프로그래밍 경시대회 HEPC 2025

Official Solutions

by

한양대학교 ERICA 캠퍼스 알고리즘 학회 0&1

✓ 후원



COSS	MOSS		의도한 난이도	출제자
A	**A**	아깝게 놓친 COSS 장학금	**Easy**	김재민 jaemkim01
	B	헤키레키잇센	**Easy**	김동우 gildong2
E	**C**	OR이 아니면? XOR	**Medium**	이민구 qsccsq22
	D	HEPC1	**Easy**	김재민 jaemkim01
I	**E**	괄호 붙이기	**Hard**	이준호 rhdqor213
B	**F**	딸깍	**Medium**	권혁준 khj20006
	G	캠핑장 부지 평탄화	**Easy**	오익준 putdata
	H	게임의 신	**Medium**	권혁준 khj20006
C		불필요한 for문	**Medium**	이준호 rhdqor213
D		영일 마을에 살고 있는 엄은 친구의 집에 가고 싶다	**Medium**	김동우 gildong2
F		투명 스프레이	**Hard**	길정민 kjmin622
G		점심 나가서 먹기	**Hard**	이준호 rhdqor213
H		지형 평탄화 탐색기	**Challenging**	오익준 putdata

1A/2A. 아깝게 놓친 COSS 장학금

`implementation`, `math`

출제진 의도 – **Easy**

@ Coss Division

- 제출 97번, 정답 -팀 (정답률 17.52%)
- 처음 푼 팀: **segfault**, 4분

@ Moss Division

- 제출 54번, 정답 3팀 (정답률 5.56%)
- 처음 푼 팀: **onnu**, 18분

✓ 출제자: 김재민 `jaemkim01`

1A/2A. 아깝게 놓친 COSS 장학금



- 다음의 공식을 이용해 각 학생의 효율성을 계산한다:
- $efficiency = \dfrac{score^3}{cost \cdot (risk + 1)}$
- 효율성, cost, 이름 총 3개의 요소를 기준으로 정렬해야 하므로 `tuple`을 사용한다.
- 정렬 기준은 효율성의 내림차순 → cost의 오름차순 → 이름 사전순이다.
- 정렬한 후, 2등 학생의 이름을 출력한다.

2B. 헤키레키잇센

`implementation`
출제진 의도 – **Easy**

@ Moss Division

- ✓ 제출 224번, 정답 15팀 (정답률 6.69%)
- ✓ 처음 푼 팀: **모스코스**, 23분
- ✓ 출제자: 김동우`gi1dong2`



- ✓ 단순 구현 문제입니다.
- ✓ 수열을 첫 원소부터 마지막 원소까지 확인하며, 다음 원소가 현재 원소보다 작은 경우를 확인하여 구간을 나누면 됩니다.
- ✓ 구간의 개수가 벽력일섬을 사용한 횟수이고, 가장 긴 구간의 길이가 한 번의 벽력일섬으로 베어낸 원소의 개수 중 최댓값입니다.

1E2C. OR이 아니면 XOR

`math, sliding_window`
출제진 의도 – **Medium**

@ Coss Division

- 제출 15번, 정답 1팀 (정답률 6.67%)
- 처음 푼 팀: **segfault**, 25분

@ Moss Division

- 제출 20번, 정답 0팀 (정답률 0.0%)
- 처음 푼 팀: -, -분

✓ 출제자: 이민구[qsccsq22]



✓ 문제에서 주어진 조건은 2가지 입니다.

1. $A_i \oplus A_j$ = K
2. $j - i \leq M\ (i < j)$

첫 번째 방법

- ✓ 이중 For문을 사용하여 가능한 모든 (i, j) 쌍을 탐색합니다.
- ✓ 이 경우 시간 복잡도는 $\mathcal{O}\left(N^2\right)$ 이므로, N 이 클 경우 시간 초과를 유발할 수 있습니다.
- ✓ 따라서, 이 방법은 비효율적이므로 사용할 수 없습니다.



두 번째 방법

- ✓ $A_i \oplus A_j = K \Rightarrow A_i = A_j \oplus K$
- ✓ 즉, 현재 A_j 에 대해, 이전에 나온 수 중 $A_j \oplus K$ 인 수가 몇 번 등장했는지를 확인하면 됩니다.

1E2C. OR이 아니면 XOR

슬라이딩 윈도우 + 해시맵

- ✓ 슬라이딩 윈도우를 유지하면서 각 값의 빈도를 해시맵으로 관리합니다.
- ✓ j 번째 원소에 대해 다음을 수행합니다:
  1. $target = A[j] \oplus K$
  2. 해시맵에서 $target$의 등장 횟수를 정답에 더합니다.
  3. 해시맵에 $A[j]$의 등장 횟수를 1 증가시킵니다
  4. 만약 윈도우 크기가 M을 초과하면, 해시맵에서 $A[j-M]$의 등장 횟수를 1 감소시킵니다
- ✓ 위 과정을 거치면 j번 인덱스에 대해서 이전 M개의 인덱스 중 $A_i \oplus A_j = K$를 만족하는 인덱스 i의 개수를 구할 수 있습니다.

시간 복잡도 분석

- ✓ 각 원소는 한 번씩만 처리되고, 해시맵 연산은 평균 $\mathcal{O}(1)$ 입니다.
- ✓ 따라서, 전체 시간 복잡도는 $\mathcal{O}(N)$ 으로 매우 효율적으로 처리할 수 있습니다.

2D. HEPC1

`implementation, string`

출제진 의도 – **Easy**

@ Moss Division

- ✓ 제출 57번, 정답 20팀 (정답률 35.09%)
- ✓ 처음 푼 팀: **모스코스**, 30분
- ✓ 출제자: 김재민`jaemkim01`

2D. HEPC1



- 두 개의 문자열을 입력받은 후, 각각을 뒤집어 이어 붙여 새로운 문자열을 만든다.
- 새로운 문자열을 두 번 이어붙여 회전된 형태를 모두 검사할 수 있도록 한다.
- 앞에서부터 4 글자씩 잘라 HEPC 또는 CPEH가 포함되어 있는지 검사한다.
- 둘 중 하나라도 발견되면 YES를 출력하고 종료한다.
- 끝까지 발견되지 않으면 NO를 출력한다.

1I/2E. 괄호 붙이기

`ad_hoc, stack`
출제진 의도 – **Hard**

@ Coss Division

- 제출 38번, 정답 0팀 (정답률 0.0%)
- 처음 푼 팀: -, -분

✓ 출제자: 이준호`rhdqor213`

@ Moss Division

- 제출 9번, 정답 0팀 (정답률 0.0%)
- 처음 푼 팀: -, -분



- ✓ 먼저 올바른 괄호 문자열의 조건에 대해 분석해 봅니다.
  - 괄호 문자열에서 (,)의 개수가 같아야 합니다.
  - 괄호 문자열의 모든 접두사에서 (가)보다 많거나 같아야 합니다.
- ✓ $A1 + B + A2$ 문자열은 $A1$, $A1 + B$, $A1 + B + A2$ 구간으로 나누어 생각해볼 수 있습니다.
- ✓ 즉, 구간별로 나누어 (가)보다 많거나 같은지를 파악하고 전체 (,)의 개수가 같은지를 보면 됩니다.

1I/2E. 괄호 붙이기



- 괄호의 개수를 셀 변수 $x_b = 0$ 을 생각합니다.
- 문자열 B 를 순회하며 (가 나올 때 x_b 값을 $+1$,)이 나올 때 x_b 값을 -1 합니다.
- 이 과정에서 x_b 의 최솟값 min_b 과, 최종 x_b 값을 알 수 있습니다.
- A 에 대해서 같은 작업을 반복하며 k 번째 문자까지 봤을 때 x_a 의 값을 c_k 라 합시다.
- A 의 $[0, i]$ 구간에서 c_k 의 최솟값을 구합니다. 이 값을 l_i 라 합니다.
- A 의 $[i, |A| - 1]$ 구간에서 c_k 의 최솟값을 구합니다. 이 값을 r_i 라 합니다.
- 이상의 값들은 $\mathcal{O}(|A| + |B|)$ 의 시간복잡도에 전부 구할 수 있습니다.

1I/2E. 괄호 붙이기



- ✓ 이후 A를 나누는 모든 경우에 대해 $A_1 + B + A_2$가 올바른지 확인합니다.
- ✓ $A_1 + B + A_2$의 모든 c_k의 값이 0 이상이고 (,)의 개수가 같아야 합니다.
- ✓ 앞서 구간에 대해 c_k의 최솟값을 구했습니다. 이를 통해 올바름을 판별합시다.
- ✓ A를 구간 $[0, i]$, $[i + 1, |A| - 1]$로 나눌 때
  - A_1 구간에 대해 모든 c_k 값이 0 이상인지 판별: $l_i \geq 0$
  - B 구간에 대해 모든 c_k 값이 0 이상인지 판별: $c_i + min_b \geq 0$
  - A_2 구간에 대해 모든 c_k 값이 0 이상인지 판별: $x_b + r_{i+1} \geq 0$
  - (,)의 개수가 같은지 확인: $x_a + x_b = 0$

  위 네 조건을 만족하는 경우 $A_1 + B + A_2$가 올바릅니다.

1B/2F. 딸깍

`implementation`, `graph_traversal`
출제진 의도 – **Medium**

@ Coss Division

- 제출 9번, 정답 3팀 (정답률 33.33%)
- 처음 푼 팀: **Sudo rm -rf /**, 46분

@ Moss Division

- 제출 2번, 정답 0팀 (정답률 0.0%)
- 처음 푼 팀: -, -분

✓ 출제자: 권혁준 `khj20006`



- 하나의 세그먼트를 7개의 비트로 나타내어 처리할 수도 있고,
- 미리 각 숫자가 상하좌우로 연결될 수 있는 숫자들을 찾아내도 됩니다.
- 이후에는 그래프 탐색을 통해 해결할 수 있고, 시간복잡도는 $\mathcal{O}(NM)$ 입니다.
- 분리 집합으로도 해결이 가능합니다.

2G. 캠핑장 부지 평탄화

`bruteforcing`
출제진 의도 – **Easy**

@ Moss Division

- ✓ 제출 18번, 정답 4팀 (정답률 22.22%)
- ✓ 처음 푼 팀: 한비, 96분
- ✓ 출제자: 오익준`putdata`

2G. 캠핑장 부지 평탄화



- 지도에서 주어진 작업 계획도와 크기가 동일한 모든 영역에 대해 평탄화가 가능한지 확인하면 됩니다.
- 가능한 영역은 2중 for문을 이용하여 간단하게 선택할 수 있습니다.
- 해당 영역에 대해 다시 2중 for문을 이용하여 평탄화 가능한지 확인합니다.
- 해당 영역이 모두 동일한 높이가 되는지 확인하는 간단한 방법은,
  $[$지도에서 선택한 영역 $-$ 작업 계획도$]$의 최솟값과 최댓값이 동일한지 확인하는 것입니다.
- 이렇게 총 4중 for문을 이용하여 코드를 작성하면 평탄화 가능한 모든 경우의 수를 구할 수 있습니다.
- 시간 복잡도는 $\mathcal{O}\left(N * M * r * c\right)$입니다.

2H. 게임의 신

`dp`, `gcd`

출제진 의도 – **Medium**

@ Moss Division

- ✓ 제출 54번, 정답 0팀 (정답률 0.0%)
- ✓ 처음 푼 팀: -, -분
- ✓ 출제자: 권혁준 `khj20006`



- 다음과 같은 dp를 정의해 봅시다.
- $dp[i] =$ (현재 수가 i인 경우에 수를 바꾸는 사람이 이길 수 있다면 1, 아니면 0)
- 현재 수가 Y인 상태에서는 더이상 수를 바꿀 수 없기 때문에 $dp[Y] = 0$입니다.
- Y와 서로소이며 Y보다 작은 어떤 수 k에 대해, $dp[k] = 1$입니다.
- k에서 반드시 Y로 바꿀 수 있기 때문입니다.



- 일반화를 해 봅시다.
- 현재 수가 a 인 상태에서 문제의 조건에 따라 바꿀 수 있는 수들을 $b_1, b_2, \cdots$ 라고 하면,
- 이 수들 중에서 만약 $dp[b_j] = 0$ 인 b_j 가 하나라도 존재한다면,
  a 를 b_j 로 바꿨을 때 게임을 이길 수 있습니다.
- 따라서 $dp[a] = 1$ 이 됩니다.
- 반대로, 만약 그러한 b_j 가 하나도 없다면 $dp[a] = 0$ 입니다.



- ✓ 위 dp 식은 Top-down으로 구하거나, Y 에서부터 내려가며 Bottom-up으로 구할 수 있습니다.
- ✓ 두 수가 서로소인지 여부는 최대공약수가 1 인지 확인해보면 됩니다.
- ✓ 시간복잡도는 $\mathcal{O}\left(N^2 \log N\right)$ 입니다.

1C. 불필요한 for 문

`implementation`
출제진 의도 – **Medium**

@ Coss Division

- ✓ 제출 24번, 정답 6팀 (정답률 25.00%)
- ✓ 처음 푼 팀: **순대국밥**, 51분
- ✓ 출제자: 이준호`rhdqor213`

1C. 불필요한 for문



- 어려워보이지만 쉬운 문제가 출제 의도였습니다.
- 먼저 배열을 사용하여 각각의 변수가 어느 for문에서 마지막으로 선언되는지 저장합니다.
- 연산식은 조건에 의해 fo로 시작하지 않습니다. 이를 통해 연산식을 구별합니다.
- 이후 연산식에 사용된 변수에 해당하는 for문만을 남깁니다.
- 시간복잡도는 $\mathcal{O}(N)$ 입니다.



1D. 영일 마을에 살고 있는 엄은 친구의 집에 가고 싶다

`dfs`, `bfs`

출제진 의도 – **Medium**

@ Coss Division

- ✓ 제출 49번, 정답 17팀 (정답률 34.69%)
- ✓ 처음 푼 팀: **segfault**, 17분
- ✓ 출제자: 김동우`gi1dong2`

1D. 영일 마을에 살고 있는 엄은 친구의 집에 가고 싶다



- ✓ 문이 잠긴 집과 연결된 도로를 모두 사용할 수 없다는 것은 그 집으로 들어갈 방법이 없다는 것을 의미합니다.
- ✓ 입력으로 주어진 그래프에서 여행을 떠난 K 명의 친구들의 집 번호에 해당하는 정점을 방문할 수 없게 처리합니다.
- ✓ 1 번 정점에서 dfs 혹은 bfs로 그래프를 탐색하여, 1 번 정점을 제외하고 방문할 수 있는 정점의 수를 구하면 됩니다.

1F. 투명 스프레이

`binary_search`, `0-1_bfs`
출제진 의도 – **Hard**

@ Coss Division

- ✓ 제출 24번, 정답 1팀 (정답률 4.17%)
- ✓ 처음 푼 팀: **segfault**, 71분
- ✓ 출제자: 길정민 `kjmin622`

1F. 투명 스프레이



- $N \times M$ 격자의 $(1, 1)$ 에서 (N, M) 으로 이동해야 합니다.
- 각 칸 (i, j) 에는 위험도 F_{ij} 가 존재하며, $F_{ij} > X$ 인 경우 스프레이가 필요합니다.
- 스프레이는 최대 K 개까지 사용할 수 있으며, 한 칸당 1개가 소모되고, 재방문 시에는 소모되지 않습니다.
- 위험도를 무시하고 이동하되, 스프레이 사용 횟수가 K 이하여야 할 때, 가능한 최소의 X 를 구해야 합니다.
- 어떠한 X 에 대해서도 $(1, 1)$ 에서 (N, M) 까지 도달할 수 없다면 -1 을 출력합니다.

1F. 투명 스프레이



- ✓ 위험도의 기준값 X 를 이분탐색하여 가능한 최소값을 찾는 방식으로 접근합니다.
- ✓ 각 X 에 대해서는 다음과 같은 방식으로 $(1,1) \to (N,M)$ 경로가 존재하는지 확인합니다.
  - 위험도가 X 이하인 칸은 스프레이 없이 이동 가능합니다.
  - 위험도가 X 초과인 칸은 스프레이를 1개 사용해야 이동 가능합니다.
  - 단, 동일한 칸을 여러 번 방문하더라도 스프레이는 최초 1회만 사용됩니다.
- ✓ 위 조건을 만족하며 도착점에 도달할 수 있는지를 판별하기 위해 0-1 BFS를 사용합니다.

1F. 투명 스프레이



- 0-1 BFS는 다익스트라에서 PQ($\mathcal{O}\left(NM \log NM\right)$) 대신 덱을 사용하는 알고리즘($\mathcal{O}\left(NM\right)$)입니다.
  - 현재 위치에서 상하좌우로 이동할 수 있으며, 이동하는 칸이 X 이하이면 덱 앞에 삽입 (가중치 0),
  - X 초과이면 덱 뒤에 삽입 (가중치 1)하여 전체 스프레이 사용량을 최소화합니다.



- ✓ 이분탐색의 범위는 $[0, 10^9]$ 로, 최대 약 30회의 반복이 발생합니다.
- ✓ 각 이분탐색 단계에서 수행되는 0-1 BFS는 $\mathcal{O}(NM)$ 의 시간복잡도를 가집니다.
- ✓ 따라서 전체 시간복잡도는 $\mathcal{O}(NM \log 10^9)$ 로, 제한 시간 내에 충분히 실행 가능합니다.

1G. 점심 나가서 먹기

`graph_traversal`, `dijkstra`
출제진 의도 – **Hard**

@ Coss Division

- ✓ 제출 14번, 정답 2팀 (정답률 14.28%)
- ✓ 처음 푼 팀: **퉁퉁퉁퉁퉁퉁퉁퉁퉁**, 38분
- ✓ 출제자: 이준호`rhdqor213`

1G. 점심 나가서 먹기



- 준호의 집을 시작점으로 각 건물까지의 최소거리를 구합니다.
- 다익스트라 후 방문할 수 있었던 건물의 식당, 카페 가격을 비교해 준호가 방문할 곳을 고릅니다.
- 그 중 한 곳을 시작점으로 다른 한 곳까지의 최소거리를 구합니다.
- (준호의 집)-(식당), (식당)-(카페), (카페)-(준호의 집)의 최소거리를 더하면 됩니다.
- 즉, 힙을 사용한 다익스트라 2번으로 문제를 풀 수 있으며 시간복잡도는 $\mathcal{O}\left(E\log(V)\right)$ 입니다.

1H. 지형 평탄화 탐색기

`implementation, hashing`
출제진 의도 – **Challenging**

@ Coss Division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: -, -분
- ✓ 출제자: 오익준putdata

문제 요약

- 지도 : $N \times M$ 크기의 격자판. 각 칸에는 1 이상 5 이하의 정수가 적혀있다. $1 \leq N, M \leq 500$
- 작업 계획도 : $r \times c$ 크기의 격자판. 각 칸에는 0 이상 5 이하의 정수가 적혀있다. 단, $r * c \leq 9$
- 평탄화 가능 : 작업 계획도를 지도의 특정 위치에 올리고 겹친 부분을 뺐을 때 모두 동일한 값이 되는 것을 의미





- ✓ 쿼리 종류
  - $1\ x\ y\ h$
    지도에서 위치 (x, y)의 값을 수정
  - $2\ r\ c\ a_{1,1}\ a_{1,2}\ \ldots\ a_{r,c}$
    입력으로 주어지는 작업 계획도를 지도 위의 모든 위치에 적용해보며, 평탄화 가능한 개수를 구한다.
- ✓ 지도의 값과 작업 계획도가 계속 달라지는 상황에서 최대 10만개의 쿼리를 처리해야 한다.

문제 분석

- ✓ 먼저, 문제를 단순하게 해결할 때의 시간 복잡도를 살펴봅시다.
- ✓ 1번 쿼리 – 값 수정
  - 특정 좌표의 값을 수정하기만 하면 되므로 $\mathcal{O}(1)$입니다.
- ✓ 2번 쿼리 – 평탄화 가능 지역 탐색
  - 모든 (i, j) 위치에서 크기가 $r \times c$인 작업 계획도를 적용하여 평탄화가 가능한지 확인해야 합니다.
  - 이는 4중 for문으로 해결할 수 있으며, 시간복잡도는 $\mathcal{O}(N * M * r * c)$입니다.



- ✓ Q 개의 쿼리가 있고, 작업 계획도의 크기 $r * c$ 는 최대 9 이므로 무시한다고 치면, 시간 복잡도는 $\mathcal{O}\left(N * M * Q\right)$ 가 됩니다.
- ✓ N 과 M 이 500, 쿼리의 개수 Q 가 $100,000$ 이라고 하면 아무리 제한 시간이 5 초여도 시간 초과가 발생합니다.
- ✓ 그렇다면 2 번 쿼리를 최적화하는 데에 초점을 맞추어 봅시다. 모든 서브 매트릭스를 탐색하는 것은 너무 비효율적입니다.

아이디어 도출 - 관찰

지도

1	2	3	2	4
2	3	4	3	4
3	4	5	4	5

작업 계획도

1	2
2	3

예제 1의 첫 번째 쿼리를 직접 수행해 봅시다. 그러면 지도에서 총 5개의 평탄화 가능 영역을 찾을 수 있습니다.

1	2
2	3

2	3
3	4

2	3
3	4

3	4
4	5

3	4
4	5

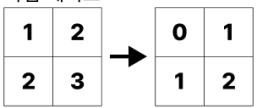
1H. 지형 평탄화 탐색기



도출된 각 영역에 대해 각각 가장 작은 값에 대한 차이를 구해보면 위와 같이 됩니다.

마찬가지로 작업 계획도에도 적용해보면, 같은 결과를 얻을 수 있습니다.

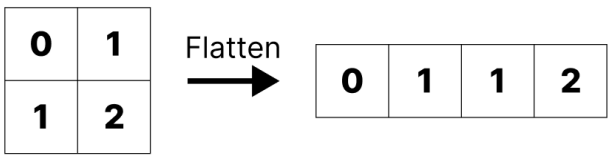
작업 계획도





- ✓ 값의 절대적 크기가 아닌, 값들의 상대적 차이만 동일하면 결과는 동일함을 발견했습니다.
- ✓ 그렇다면 지도에서 모든 모양에 대한 패턴(상대적 차이)을 저장해두고, 2번 쿼리를 수행할 때 해당 모양에 대한 패턴을 찾아보는 것은 어떨까요?
- ✓ 다시 말해, $r \times c$인 격자판에 대한 패턴의 개수를 모두 구해놓고, 2번 쿼리를 상수 시간 내에 처리하는 것입니다.
- ✓ 이때, 가능한 $r \times c$인 격자판의 모양은 총 23가지 입니다. $(\because 1 \leq r * c \leq 9)$
- ✓ $1 \times 1, ..., 1 \times 9$ / $2 \times 1, ..., 2 \times 4$ / $3 \times 1, ..., 3 \times 3$
  $4 \times 1, 4 \times 2$ / 5×1 / 6×1
  7×1 / 8×1 / 9×1

- 어떻게 각 모양에 대한 패턴을 저장할 수 있을까요?
- 가장 작은 수에 대한 상대적 차이는 0부터 4까지 발생합니다. 따라서 5진법을 이용해 패턴을 압축할 수 있습니다.
- 즉, 상대적 차이로 변환한 매트릭스를 평탄화(flatten)하여 5진법으로 해싱합니다.



- 위의 예시를 해싱하면, $0 * 5^0 + 1 * 5^1 + 1 * 5^2 + 2 * 5^3 = 280_{(10)}$ 이 됩니다.



- ✓ 실제로 지도에 존재하는 모든 모양에 대한 패턴을 저장할 수 있는지 확인해 봅시다.
- ✓ 먼저 메모리 사용량을 예측해 봅시다.
- ✓ 다음과 같은 2차원 배열을 정의합니다.
- ✓ $cnt[$직사각형 모양 번호$][$해싱된 값$]$ = 지도에서 해당 모양이면서 상대적 차이를 해싱한 값의 등장 개수
- ✓ 해싱될 수 있는 최대 값은 $4 * 5^0 + 4 * 5^1 + 4 * 5^2 + \cdots + 4 * 5^8 = 1,953,124$
- ✓ 배열의 크기는 $23 \times 1,953,125 = 44,921,875$이고, int 변수는 4바이트이므로, 전체 메모리 사용량은 $179,687,500\ Bytes \approx 172\ MB$로 메모리 제한 $1024\ MB$ 이내로 들어옵니다.

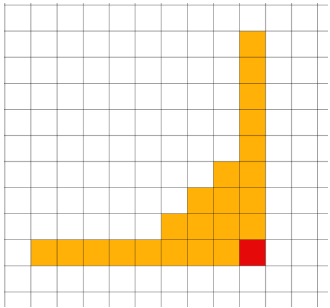


- 다음으로 시간 복잡도를 계산해 봅시다.
- 모든 모양에 대해 지도를 탐색하고 평탄화(Flatten)하여 해싱하는 과정은 간단?하게 6중 for 문을 통해 수행할 수 있습니다.
- 먼저 모양을 정하고, 지도에서 가능한 영역을 탐색하며 이 영역을 해싱하는 과정의 시간 복잡도는 $\mathcal{O}\left((r*c)^2*N*M\right)$ 이 됩니다.
- 단순하게 $r=9, c=9, N=500, M=500$ 으로 계산하면 약 16 억이라는 수치가 나오지만, 실제로는 $r \times c = 9$ 이므로 훨씬 적은 수치가 나오고, 시간 제한 5 초 내에 충분히 돌아갑니다.
- 이제 2 번 쿼리가 들어오면, 작업 계획도의 패턴을 해싱한 후, cnt 배열을 통해 상수 시간 내로 정답을 구할 수 있습니다.



- 마지막으로 1 번 쿼리에 대해 이야기해 봅시다.
- 2 번 쿼리를 빠르게 계산하기 위해 cnt 배열을 구축했기 때문에, 이전처럼 단순하게 지도의 값을 변경하는 것은 불가능합니다.
- 이제 지도의 한 위치의 값을 변경한다는 것은 해당 값을 사용하여 해싱한 부분들을 모두 찾아 변경해줘야 함을 의미합니다.

1H. 지형 평탄화 탐색기



- 빨간색 : 변경 지점, 주황색 : 해당 인덱스 기준으로 해싱할 지점
- 값을 변경하기 전에, 빨간색과 주황색 지점에서 모든 모양에 대해 수행한 해싱값을 모두 cnt 에서 제거합니다.
- 값을 변경하고, 다시 빨간색과 주황색 지점에서 모든 모양에 대해 수행한 해싱값을 모두 cnt 에 추가합니다.



- 이 두 번의 과정을 거치면,
  $2 * 23 * (\sum_{k=1}^{9} k + \sum_{k=1}^{4} 2k + \sum_{k=1}^{3} 3k + \sum_{k=1}^{2} 4k + 5 + 6 + 7 + 8 + 9) = 5980$의 배수 연산이 소요됩니다. (중간에 빨간색을 업데이트하지 않는 모양에 대해선 패싱하면 이보다 더 줄일 수 있습니다. - 상수 커팅)
- 너무 비효율적으로 코드를 작성하는 것이 아니라면, 충분히 10만개에 대한 쿼리를 처리할 수 있을 것입니다.