제1회 코더즈 코딩페어 해설

Official Solutions

by

운영진 일동

2025년 12월 25일

H. 오랑

#math #combinatorics #ad_hoc #dp
출제진 의도 – **Hard**

- **Onsite**
  - 제출 20번, 정답 0명 (정답률 0.000%)
- **Open**
  - 제출 185번, 정답 33명 (정답률 17.838%)
  - 처음 푼 사람: **dohyonne**, 27분
- 출제자: **mon_de2738**

H. 오량

오량으로 인해 노트를 뺏긴 사람은 0개의 노트를 가지게 됩니다.

노트를 0개 가진 학생의 수는 줄어들지 않습니다. 또한 첫 오량을 시행하면 노트를 0개 가진 사람이 생깁니다.

- ✓ $A_1 = A_2 = \cdots = A_N = 1$이기 위해선 오량이 일어나지 않아야 합니다.
- ✓ $A_1 = A_2 = \cdots = A_N = 1$일때 가능한 경우의 수는 1입니다.

H. 오량

앞의 경우를 제외하면, 노트를 0개 가진 학생이 적어도 한 명 존재합니다. 이 학생을 통해 서로 다른 두 학생의 노트를 교환할 수 있습니다.

학생 a, b가 노트 0개를 가진 학생 c를 통해 서로 가진 노트를 교환하려면

- ✓ 학생 i가 학생 a의 노트를 오량하고,
- ✓ 학생 a가 학생 b의 노트를 오량하고,
- ✓ 학생 b가 학생 i의 노트를 오량하면 됩니다.

결과적으로 각 학생에게 $A_1, A_2, \ldots, A_N$ 개씩 나누어 주는 **모든 경우**가 가능해집니다.

H. 오량

각 경우에 대해, 아래의 과정과 같이 $\mathcal{O}(N)$ 번의 오량을 통해 원하는 상태를 만들 수 있음을 증명할 수 있습니다.

1. 오량을 통해 각 학생에게 갈 노트를 모읍니다.
2. $A_i = 0$인 학생 i에게 $A_j \neq 0$이고 노트를 들고 있지 않는 학생 j가 오량을 하여 모든 $A_i = 0$인 학생이 0개의 노트를 들도록 합니다.
3. 0개의 노트를 가지고 있는 학생을 통해 노트를 적절히 교환합니다.

각 과정에서 $\mathcal{O}(N)$ 번의 오량이 필요하므로 필요한 오량의 총 횟수는 $\mathcal{O}(N)$ 번입니다.

H. 오량

S_i 를 $\sum_{j=1}^{i} A_j$ 라고 합시다. (단, $S_0 = 0$)

이때 각 학생에게 $A_1, A_2, \ldots, A_N$ 개씩 나누어 주는 경우의 수는 다음과 같습니다.

$$\text{Ans} = \prod_{1 \le i \le N, A_i \neq 0} \binom{N - S_{i-1}}{A_i} \mod 10^9 + 7$$

$A_1, A_2, \cdots, A_N$ 이 모두 1 인지 체크하는 것을 잊지 맙시다.

H. 오량

$n \leq 2000$에 대해, 모든 $\binom{n}{r}$을 총 $O(2000^2)$의 시간에 미리 계산해 두면 각 테스트 케이스당 $O(N)$에 문제를 해결 가능해 넉넉히 AC를 받습니다.

$\binom{n}{r} = (\binom{n-1}{r-1} + \binom{n-1}{r}) \mod 10^9+7$을 점화식으로 하고, 초깃값을 $\binom{n}{0} = \binom{n}{n} = 1$로 설정해두면 쉽게 구현할 수 있습니다.

python의 경우 `math.comb` 함수를 이용해 전처리 없이 각 테스트 케이스당 $O(N)$에 문제를 해결할 수 있습니다.

$\binom{n}{r}$을 $\frac{n!}{r!(n-r)!}$으로 계산하거나 다항정리를 사용해도 문제를 해결할 수 있습니다. 이때는 페르마의 소정리를 사용해 $n!$의 역원을 계산해야 하며, 전처리 및 구현 방식에 따라 테스트 케이스당 $\mathcal{O}(N \log(10^9+7))$ 혹은 $\mathcal{O}(N)$이 됩니다.

I. 룩 vs 폰

#ad_hoc #case_work #game_theory
출제진 의도 – **Hard?**

- **Onsite**
  - 제출 12번, 정답 0명 (정답률 0.000%)
- **Open**
  - 제출 145번, 정답 22명 (정답률 15.172%)
  - 처음 푼 사람: **tjrn3712**, 29분
- 출제자: **js7777**

- * 해설의 편의상 P_R을 R열에 위치한 폰으로 정의합니다.
- * 룩이 P_R을 공격한다는 것은, 룩이 R열에 위치함을 의미합니다.
- $M \leq N$ 이라면 P_M을 잡으며 1번의 이동으로 승리할 수 있습니다.
- $M > N$인 경우를 생각해 봅시다.

Case: $N = 1$

- 룩이 1열로 이동한 뒤 다음 차례에 폰을 잡으며 승리합니다.
- **따라서 최소 이동 횟수는 2입니다.**
- 이후 어떠한 경우에도 최소 이동 횟수가 2보다 크다는 것도 알 수 있습니다.

I. 룩 vs 폰

Case: $N = 2, K = 3$

- ✓ 룩을 1열과 2열 중 하나로 움직여야 합니다. 폰의 배치가 좌우 대칭이므로 오량이가 룩을 1열로 움직인다고 가정합니다.
- ✓ 대신이는 P_1 을 보호하기 위해 무조건 P_1 을 1칸 움직여야 합니다. 이때 오량이는 2가지 선택을 할 수 있습니다.
  1. **룩을 2열로 움직였을 때:** 대신이는 P_2 를 보호하기 위해 무조건 P_2 를 2칸 움직여야 합니다. 이후 오량이가 룩을 다시 1열로 움직이면, 대신이가 어떠한 선택을 해도 다음 차례에 P_1 을 잡을 수 있습니다. (**이동 횟수 4**)
  2. **룩을 3열로 움직였을 때:** 대신이가 P_2 를 1칸 움직이면, P_1 과 P_2 가 앞으로 1칸 전진한 걸 제외하고 초기 상태와 똑같아지게 됩니다.
- ✓ **따라서 최소 이동 횟수는 4입니다.**

Case: $N = 2, K > 3$

- ✓ $K = 3$일 때와 다르게, 룩에게는 하나의 선택지가 더 있습니다. 바로, **폰이 없는 열로 이동하는 것**입니다.
- ✓ 폰의 배치가 좌우 대칭이므로 대신이가 P_1을 움직인다고 가정합니다.
  1. P_1**을 1칸 움직였을 때:** 룩이 1열로 이동합니다. 대신이는 어떠한 선택을 해도 다음 차례에 P_1이 잡히는 것을 막을 수 없습니다. (**이동 횟수 3**)
  2. P_1**을 2칸 움직였을 때:** 룩이 2열로 이동합니다. 대신이는 어떠한 선택을 해도 다음 차례에 P_2가 잡히는 것을 막을 수 없습니다. (**이동 횟수 3**)
- ✓ 룩이 폰이 있는 열로 이동하게 되면 $N = 2, K = 3$일 때와 같은 양상이 됩니다.
- ✓ **따라서 최소 이동 횟수는 3입니다.**

Case: $N \geq 3$

$N \geq 3$일 때 오량이가 4번 이하의 이동으로 승리할 수 있는 방법은 없다.

- 최종적으로 잡는 폰이 P_X 라고 가정합시다.
- $1 < X < N$**인 경우:**
- P_X 를 그냥 공격하면, 대신이는 P_X 를 1칸 움직여 P_{X-1}, P_{X+1} 이 P_X 를 보호하게 할 수 있습니다. 따라서 P_{X-1}, P_{X+1} 을 먼저 공격해야 합니다.
- 하지만 대신이가 P_{X-1}, P_{X+1} 을 1칸씩만 움직인다면, 이후 룩이 X 열에 위치했을 때 P_X 를 2칸 움직여 P_{X-1}, P_{X+1} 에게 보호받을 수 있게 됩니다.
- **따라서 4번 이하의 이동으로 승리할 수 없습니다.**

- $X = 1$ **인 경우 (좌우 대칭이므로** $X = N$ **과 동일):**
- P_1 을 보호하고 있는 유일한 폰은 P_2 입니다. 따라서 P_2 를 먼저 공격해야 합니다.
- $1 < X < N$ 일 때와 같은 이유로, P_2 가 1칸만 이동한다면 P_1 이 2칸 움직이며 보호받을 수 있습니다.
- 즉, P_1 을 잡기 위해선 P_2 **가 2칸 이동한 상태에서** P_1 **을 공격해야 합니다.**
- 그런데 P_2 가 2칸 움직이도록 강제하려면, P_1 과 P_3 가 1칸씩 이동해야 합니다.
- P_3 만 공격한 뒤 P_2 를 공격하면, P_2 는 1칸만 움직여도 P_1 에게 보호받을 수 있기 때문입니다.
- 결국 P_1, P_3 를 모두 공격한 뒤 P_2 를 공격하고, 이후에 1열로 움직여야 P_1 을 잡을 수 있습니다. 이는 이동 횟수가 5번인 방법입니다.
- **따라서 4번 이하의 이동으로 승리할 수 없습니다.**

- ✓ $N \geq 3$**일 때 최소 이동 횟수는 5입니다.** 아래는 가능한 방법 중 하나입니다.
  1. 룩이 **1열**로 이동합니다. 대신이는 이를 방어하기 위해 P_1 을 1칸 이동해야 합니다.
  2. 룩이 **3열**로 이동합니다. 대신이는 이를 방어하기 위해 P_3 를 1칸 이동해야 합니다.
  3. 룩이 **2열**로 이동합니다. P_1 과 P_3 가 모두 1칸 움직인 상태이기 때문에, 대신이는 이를 방어하기 위해 P_2 를 2칸 이동해야 합니다.
  4. 룩이 **1열**로 이동합니다. 대신이는 어떠한 선택을 해도 다음 차례에 P_1 이 잡히는 것을 막을 수 없습니다.
  5. P_1 을 잡으며 오량이가 승리합니다.

- ✓ 위 내용을 종합해보면 오량이는 어떠한 경우에도 항상 승리하며, 최악의 경우에도 5번 이하의 이동으로 승리한다는걸 알 수 있습니다.
- ✓ 시간복잡도는 $O(1)$ 입니다.

J. 균형의 창조자

#knapsack

출제진 의도 – **Hard**

- **Onsite**
  - 제출 13번, 정답 0명 (정답률 0.000%)
- **Open**
  - 제출 52번, 정답 28명 (정답률 53.848%)
  - 처음 푼 사람: **leinad2**, 41분
- 출제자: **nflight11**

J. 균형의 창조자

- 모든 경우의 수를 테스트하면서 뱃지를 취득할 수 있는지 여부를 체크하는 것은 2^{200} 가지의 경우 전부를 탐색해야 하므로 당연히 불가능합니다.
- 문제를 더 쉬운 문제로 바꿔 봅시다. 대신이가 뱃지를 얻기 위해서 필요한 구현 레이팅이 없으며, 모든 문제가 기하 레이팅 G_i 만을 올려주게 된다면 어떻게 될까요?
- 이 경우 $dp[i][j]$ 를 앞에서부터 i 개의 문제까지만 간주하였을 때 기하 레이팅을 j 만큼 올리는 데 필요한 문제의 최솟값으로 정의하면 됩니다.
- DP 전이식은 $dp[i][j+G_i] = \min(dp[i][j+G_i], dp[i-1][j]+1)$ 입니다. 이러한 문제를 배낭 문제(Knapsack Problem)이라고 합니다.

J. 균형의 창조자

- 이제 원래 문제로 돌아가 봅시다. 문제를 풀기 위해서는 구현 레이팅까지 생각해 주어야 합니다.
- $dp[i][a][b]$ 를 앞에서부터 i 개의 문제까지만 간주하였을 때 구현 레이팅을 a, 기하 레이팅을 b 만큼 올리는 데 필요한 문제의 최솟값으로 정의하여야 합니다.
- $dp[0][0][0] = 0$, 그렇지 않을 경우 매우 큰 수 $\inf$ 로 초기화해주고, 전이식 $dp[i][a + I_i][b + G_i] = \min(dp[i][a + I_i][b + G_i], dp[i - 1][a][b] + 1)$ 을 적용해 주면 $dp[N][r_I][r_G]$ 의 값이 문제의 정답이 됩니다.
- 만일 $dp[N][r_I][r_G] = \inf$ 인 경우 모든 문제를 풀어도 목표를 달성하지 못했다는 뜻입니다.
- 이 과정에서 $a + I_i > r_I$ 나 $b + G_i > r_G$, 즉 목표를 초과달성하게 되는 경우 $a + I_i$ 대신 r_I, $b + G_I$ 대신 r_G 를 사용하면 메모리 낭비를 방지할 수 있습니다.
- 시간복잡도와 공간복잡도 모두 $\mathcal{O}(Nr_Ir_G)$ 입니다. 최악의 경우라도 $N = r_I = r_G = 200$ 이므로 시간 제한 내에 충분히 동작합니다.

Challenge.

- 시간제한이 이 문제와 동일하나 메모리 제한이 극단적으로 적은 경우(e.g. 32KB) 이 문제를 어떻게 풀 수 있을까요?
- $(N+1) \times (r_I+1) \times (r_G+1)$ 크기의 3차원 배열을 선언하는 경우 즉각적으로 메모리 초과로 이어지는 메모리 제한에서는 이 풀이를 그대로 이용할 수는 없습니다.
- $dp[i][-][-]$의 값은 $dp[i-1][-][-]$만이 결정한다는 것에 주목하여야 합니다.
- 이러한 기법을 **토글링**이라고 부릅니다.

K. 급식 뭐 먹지

`#greedy #sort #priority_queue`
출제진 의도 – **Hard**

- **Onsite**
  - 제출 0번, 정답 0명 (정답률 0.000%)
- **Open**
  - 제출 50번, 정답 30명 (정답률 60.000%)
  - 처음 푼 사람: **0xchaser**, 35분
- 출제자: **young_out**

K. 급식 뭐 먹지

- 학생들의 정보를 a_i에 대해 오름차순으로 정렬한 후 새로 인덱스를 부여한다고 해봅시다.
- 앞에서부터 순회하여 K번째 학생까지 관찰했을 때, 맛 수치를 a_i 만큼, 영양 수치를 $M - a_i$ 만큼 증가시킨다고 해봅시다.
- 그렇다면 K번째 이하의 학생들은 맛 수치를 만족하고, 영양 수치를 만족하는지만 관찰하면 됩니다. 이를 어떻게 효율적으로 구할 수 있을까요?
- a_i가 증가하는 방향이므로, 영양 수치는 감소하는 방향이 됩니다.
- 즉 어떤 학생이 앞에서 영양 수치를 이미 만족하지 못했다면, 앞으로도 만족하지 못할 수밖에 없습니다.

K. 급식 뭐 먹지

- K 번째 학생까지가 급식을 먹게 하는 영양 수치들을 최솟값 우선순위 큐에 저장해둡니다.
- 맛 수치는 모두 만족할 수밖에 없고, 영양 수치를 만족시키는지만 보면 되므로 우선순위 큐의 최솟값이 $M - a_i$ 보다 작지 않을 때까지 pop합니다.
- K 번째 학생까지를 봤을 때 급식을 먹는 학생 수는 우선순위 큐의 크기가 되며, 모든 K 에 대해 검사하면 답을 구할 수 있습니다.
- 정렬, 우선순위 큐를 이용하므로 $\mathcal{O}(N \log N)$ 에 해결 가능합니다.

L. 해안 경비

`#geometry #prefix_sum`
출제진 의도 – **Hard**

- **Onsite**
  - 제출 0번, 정답 0명 (정답률 0.000%)
- **Open**
  - 제출 32번, 정답 29팀 (정답률 90.625%)
  - 처음 푼 팀: **sadtreap**, 41분
- 출제자: **lunarlity**

L. 해안 경비

- 문제를 요약하면 $(0,0)=A, (W,0)=B, (x_i,H)=X_i$ 라 할 때, 다음의 쿼리를 처리하는 것입니다.
- $l\ r$: $\triangle ABX_l, \triangle ABX_{l+1}, \cdots, \triangle ABX_r$ 의 합집합의 넓이를 구하여라.
- X_i 에는 중요한 특징이 있는데, 항상 $x_i < x_{i+1}$ 이며 X_i 의 y 좌표가 같다는 것입니다.
- $\overline{AX_{i+1}}$ 와 $\overline{BX_i}$ 의 교점을 P_i 라 해봅시다.
- 그러면 우리가 구해야할 쿼리는 $[\square ABX_rX_l] - \sum_{i=l}^{r-1}[\triangle X_iX_{i+1}P_i]$ 를 구하는 문제가 됩니다.
- $[\square ABX_rX_l]$ 의 경우 사다리꼴의 넓이이므로 $\mathcal{O}(1)$ 에 구할 수 있습니다.
- $[\triangle X_iX_{i+1}P_i]$ 의 경우 $\triangle X_iX_{i+1}P_i \sim \triangle BAP_i$ 임을 이용하면 넓이를 쉽게 구할 수 있고, 쿼리에 필요한 구간 넓이 합은 누적합을 사용하면 $\mathcal{O}(1)$ 에 구할 수 있습니다.
- 전체 시간복잡도는 $\mathcal{O}(N+Q)$ 입니다.

M. gmb 수열

#dp

출제진 의도 – **Hard**

- **Onsite**
  - 제출 0번, 정답 0명 (정답률 0.000%)
- **Open**
  - 제출 43번, 정답 22명 (정답률 51.163%)
  - 처음 푼 사람: **sadtreap**, 34분
- 출제자: **choyj416**

M. gmb 수열

- 수열 A 가 gmb 수열이라는 것은 길이가 3 이상이고 $A_i = A_{i-2} - A_{i-1}$ 를 만족한다는 것입니다.
- 수열 $B_{1 \dots N}$ 에서 길이가 2 인 부분 수열은 ${}_NC_2$ 개 있습니다.
- 정답을 구할 때 gmb 수열의 조건 중 길이가 3 이상이라는 것을 길이가 2 이상이라는 것으로 생각하고 ${}_NC_2$ 를 빼도 됩니다.

M. gmb 수열

- 수열이 gmb 수열이 될 조건을 다시 생각해봅시다.
- 길이가 $L(L > 3)$ 인 수열 A 가 gmb 수열이라는 것은 $A_{1 \dots L-1}$ 이 gmb 수열이고 $A_L = A_{L-2} - A_{L-1}$ 이라는 것입니다.
- 즉, 수열 A 가 gmb 수열인지를 판단할 때 $A_{1 \dots L-1}$ 이 gmb 수열인지를 판단하고 A_L 을 추가하는 방식을 사용할 수 있습니다.
- 이를 통해 $B_{1 \dots i}$ 의 정답을 구할 때 $B_{1 \dots i-1}$ 에 B_i 을 추가하는 방식으로 구할 수 있음을 예상할 수 있습니다.

M. gmb 수열

- $B_{1 \dots i-1}$ 의 정답을 이용해 $B_{1 \dots i}$ 의 정답을 찾을 수 있을까요?
- $B_{1 \dots i-1}$ 의 정답에서 B_i 가 추가되어 생기는 부분 수열 중 gmb 수열의 개수를 더하면 $B_{1 \dots i}$ 의 정답을 찾을 수 있습니다.
- 이를 다이나믹 프로그래밍(dp)를 사용해 구현할 수 있습니다.

M. gmb 수열

- $dp[i][j][k]$ 를 $B_{1...i}$ 에서 길이가 2 이거나 gmb 수열인 부분 수열 중 j 로 끝나고 마지막에서 두 번째 수가 k 인 것의 개수라고 합시다.
- $dp[i][j][k]$ $(j \neq B_i)$ 는 추가되는 부분 수열이 없으므로 $dp[i-1][j][k]$ 입니다.
- $dp[i][B_i][k]$ 는 $[..., B_i + k, k, B_i]$ 꼴의 부분 수열이 추가되므로 $dp[i-1][k][B_i + k]$ 를 더해야합니다.
- 또한, 길이가 2 인 수열 $[k, B_i]$ 가 추가 될 수 있으므로 $B_{1...i-1}$ 의 k 의 개수를 더해야합니다.
- $B_{1...x}$ 에서 y 의 개수를 $C[x][y]$ 라고 하면 점화식은 아래와 같습니다.
  - $dp[i][j][k] = dp[i-1][j][k]$ $(j \neq B_i)$
  - $dp[i][j][k] = dp[i-1][j][k] + dp[i-1][k][B_i + k] + C[i-1][k]$ $(j = B_i)$
- 이때 정답은 $dp[N][i][j]$ 를 모두 더한 것에서 ${}_NC_2$ 를 빼면 됩니다.

- 그렇다면 $C[i][j]$ 는 어떻게 구할까요?
- $B_{1...i}$ 는 $B_{1...i-1}$ 에서 B_i 만 추가된 것이므로 $j = B_i$ 일때 개수가 추가됩니다.
- 즉, $C[i][j] = C[i-1][j]$ $(j \neq B_i)$ 이고 $C[i][j] = C[i-1][j] + 1$ $(j = B_i)$ 입니다.

M. gmb 수열

- $dp[i][j][k]$ 는 $dp[i-1][j][k]$ 에서 $j = B_i$ 일때만 갱신하면 되므로 갱신은 최대 300 번 일어납니다.
- $C[i][j]$ 도 $j = B_i$ 일때만 갱신하면 되므로 갱신은 각 i 마다 한 번씩 일어납니다.
- 즉, 서로 다른 수의 개수를 $M (M \leq 300)$ 개라고 하면 시간복잡도는 $O(NM)$ 이 됩니다.

N. 그래프 복원

#ad_hoc #bitmask
출제진 의도 – **Challenging**

- **Onsite**
  - 제출 0번, 정답 0명 (정답률 0.000%)
- **Open**
  - 제출 21번, 정답 12명 (정답률 57.143%)
  - 처음 푼 사람: **sadtreap**, 17분
- 출제자: **realpsdoingdamyoo**

N. 그래프 복원

- 정점 개수가 40인 그래프를 유일하게 결정하기 위해서는 얼마큼의 정보가 필요할까요?
- 정점 1과 정점 2, 정점 1과 정점 3, $\cdots$ 정점 $N-1$과 정점 N 사이에 간선이 있는지의 여부를 1비트로 저장합시다.
- 길이 ${}_{40}C_2 = 780$ 비트에 가능합니다.
- 한 번의 메시지로는, 거리 정보를 무시하고 $40 \times 40 = 1600 > 2^{10}$ 이므로 10비트의 정보를 보낼 수 있습니다.
- 따라서 78개의 메시지를 사용하여 필요한 정보를 모두 보낼 수 있습니다.