DSPC 2026 에디토리얼

Official Solutions

by
캘리퍼스

문제	의도한 난이도	출제자
A 샷건 금지	**Easy**	류나현
B 무트코인	**Easy**	유지원
C 동아리 부원 모집하기	**Easy**	류나현
D 울트라 369	**Medium**	황태균
E 호박농장	**Medium**	김태경
F 사이언키피아 보고서 복구	**Medium**	황태균
G 유전자(펑!)	**Medium**	류나현
H welcome to maimai	**Hard**	김태경
I 닮은 수 쌍	**Hard**	유지원
J 요동치는 무트코인	**Hard**	유지원
K 슬라임 공연장 대청소	**Challenging**	유지원
L 가짜 박홍이 어디에 있을까??	**Challenging**	유지원

A. 샷건 금지 (류나현)

Math, Implementation

출제자 의도 - **Easy**

- 샷건을 1회 칠 때마다 X점 감소, 2회 칠 때마다 Y점 증가합니다.
- 최종 점수는 $100 - X * T + Y * (T//2)$로 $O(1)$에 구할 수 있습니다.

B. 무트코인 (유지원)

Brute Force

출제자 의도 - **Easy**

- i번째 이후의 날들의 가격을 전부 확인하여 무 가격이 가장 높은 날을 찾으면 됩니다.
- $O(NQ)$에 해결할 수 있습니다.

C. 동아리 부원 모집하기 (류나현)

Sorting, Implementation

출제자 의도 - **Easy**

- 동아리 이름을 키로 map(c++), unordered_map(c++), dictionary(python3), defaultdict(python3) 등을 사용하면 간단하게 풀이가 가능합니다.
- 동아리 이름 3개가 입력될 때마다 각각의 value에 4, 2, 1을 더합니다.
- 내림차순으로 정렬하면 $O(N \log N)$에 해결할 수 있습니다.

D. 울트라 369 (황태균)

Implementation, String

출제자 의도 - **Medium**

- 1부터 100까지의 진행은 항상 같습니다.
- 한 주기 동안 말하는 음절들을 하나의 문자열로 만들어 둡니다.
- 게임은 이 문자열이 무한히 반복되는 형태입니다.
- N번째 턴은 주기 길이로 나눈 나머지를 출력하면 됩니다.
- 전체 시간복잡도는 $O(1)$입니다.

E. 호박농장 (김태경)

Dynamic Programming

출제자 의도 - **Medium**

- dp[*i*][*j*]를 *i* 행 *j* 열을 끝점으로 하는 가장 큰 정사각형의 변의 길이로 정의합니다.
- 0 인 땅에서는 dp[*i*][*j*] = 0으로 계산합니다.
- 1 인땅에서는 dp[*i*][*j*] = *min*(dp[*i*-1][*j*], dp[*i*][*j*-1], dp[*i*-1][*j*-1]) + 1으로 계산합니다.
- 최대의 dp값을 찾으면 $O(N^2)$으로 해결할 수 있습니다.

F. 사이언키피아 보고서 복구 (황태균)

Greedy

출제자 의도 - **Medium**

- 복구 키에서 앞쪽의 0은 현재 위치를 바꾸지 못합니다.
- 따라서 앞쪽 0들을 제거하고, 첫 번째 1부터 생각합니다.

F. 사이언키피아 보고서 복구 (황태균)

Greedy

출제자 의도 - **Medium**

- 이후에는 왼쪽부터 그리디하게 처리합니다.
- 현재 위치에서 X와 Y가 다르면, 이 위치를 바꿀 수 있는 방법은 현재 위치에 복구 연산을 적용하는 것뿐입니다.
- 반대로 이미 같다면 연산을 적용할 필요가 없습니다.

F. 사이언키피아 보고서 복구 (황태균)

Greedy

출제자 의도 - **Medium**

- 왼쪽부터 결정하면 이전 위치는 다시 변하지 않습니다.
- 따라서 각 위치의 선택이 강제되고, 그 결과가 최소 연산 횟수가 됩니다.
- 모든 연산을 적용한 뒤 X가 Y와 같으면 답을 출력합니다.
- 끝까지 처리해도 다르다면 복구가 불가능하므로 -1을 출력합니다.
- 전체 시간복잡도는 $O(NM)$입니다.

G. 유전자(펑!) (류나현)

Graph, String

출제자 의도 - **Medium**

- 안전한 N개의 서열들과 T를 set에 저장합니다.
- 현재 서열에서 한 글자씩 바꾸며 각 문자열을 노드로 두어 BFS를 진행합니다.
- 최대 N개의 서열을 방문할 수 있습니다.

G. 유전자(펑!) (류나현)

Graph, String

출제자 의도 - **Medium**

- 하나의 서열에서 다음 서열을 확인할 때 L개의 위치에 대해 4가지 문자를 시도할 수 있습니다.
- 또한 문자열 탐색(hash)에 $O(L)$이 소요되므로 $O(NL^2)$에 해결할 수 있습니다.

H. welcome to maimai (김태경)

Dynamic Programming

출제자 의도 - **Hard**

- 2차원 DP를 활용합니다.
- dp[*i*][*j*]를 첫 번째 문자열의 *i*번째까지, 두 번째 문자열의 *j*번째까지 처리했을 때의 최소 횟수로 정의합니다.

H. welcome to maimai (김태경)

Dynamic Programming

출제자 의도 - **Hard**

- 각 상태에서는 한쪽 문자열만 전진하는 경우를 먼저 고려합니다.
- 첫 번째 문자열에서 1개 또는 2개를 추가할 수 있습니다.
- 두 번째 문자열에서도 1개 또는 2개를 추가할 수 있습니다.
- dp[*i*-1][*j*], dp[*i*-2][*j*], dp[*i*][*j*-1], dp[*i*][*j*-2]중에서 최소가 되는 값으로 전이하며 값을 갱신합니다.

H. welcome to maimai (김태경)

Dynamic Programming

출제자 의도 - **Hard**

- 두 문자열이 같은 센서를 누르는 경우도 추가로 고려합니다.
- 이 경우 두 문자열을 동시에 전진시킬 수 있습니다.
- 첫 번째 문자열은 i - 1 또는 i - 2에서 올 수 있습니다.
- 두 번째 문자열은 j - 1 또는 j - 2에서 올 수 있습니다.
- 이때 선택한 문자열이 서로 같다면 같은 센서를 한 번 누르는 것으로 처리합니다.

H. welcome to maimai (김태경)

Dynamic Programming

출제자 의도 - **Hard**

- 초기 상태는 추가적인 처리가 필요합니다.
- i 또는 j가 0인 경우가 존재합니다.
- 이는 두 문자열 중 하나에서 아직 아무 문자도 선택하지 않은 상태입니다.
- 이 경우를 제외하면 한쪽 문자열만 먼저 처리하는 경우를 계산할 수 없습니다.

H. welcome to maimai (김태경)

Dynamic Programming

출제자 의도 - **Hard**

- 모든 상태를 순서대로 계산하면 최종적으로 dp[N][M]이 답이 됩니다.
- 전체 시간복잡도는 $O(NM)$입니다.

I. 닮은 수 쌍 (유지원)

Math

출제자 의도 - **Hard**

- N의 제한이 10^{13}이므로 가능한 경우를 탐색하는 방식은 TLE가 남을 알 수 있습니다.
- 따라서 더 빠른 방법이 필요합니다.

I. 닮은 수 쌍 (유지원)

Math

출제자 의도 - **Hard**

- $g = gcd(A, B)$라고 둡니다.
- $A = gx$, $B = gy$, $gcd(x, y) = 1$이 됩니다.
- 따라서 원래 식을 $gx + gy = g^2$에서 $x + y = g$로 변형할 수 있습니다.

I. 닮은 수 쌍 (유지원)

Math

출제자 의도 - **Hard**

- 따라서 본 문제는 어떤 g에 대해서 $x + y = g$, $gcd(x, y) = 1$을 만족하는 자연수 순서쌍 (x, y)의 개수를 세는 것으로 환원할 수 있습니다.
- 이때 g^2는 N 이하이어야 합니다.
- $x + y = g$일 때, $gcd(x, y) = 1$로부터 $gcd(x, g) = 1$임을 알 수 있습니다.

I. 닮은 수 쌍 (유지원)

Math

출제자 의도 - **Hard**

- 따라서 가능한 x의 수는 g = 2부터 $sqrt(N)$까지의 오일러 피 함수의 합과 같습니다.
- 오일러 피 함수의 값은 Sieve 방식으로 구할 수 있습니다.
- 전체 시간복잡도는 $O(sqrt(N)\ log\ log\ N)$가 됩니다.

J. 요동치는 무트코인 (유지원)

Segment Tree

출제자 의도 - **Hard**

- 질의 2 *s K* 는 구간 [*s*+1, *N*]에서 값이 *P*[*s*] + *K*이상인 가장 오른쪽 위치와 같습니다.
- 또한 1 *i x* 연산으로 특정 날의 가격이 바뀝니다.
- 이러한 연산은 Segment Tree로 빠르게 처리할 수 있습니다.

J. 요동치는 무트코인 (유지원)

Segment Tree

출제자 의도 - **Hard**

- Segment Tree의 각 노드에 해당 구간의 최댓값을 저장합니다.
- 어떤 구간의 최댓값이 $P[s] + K$보다 작다면 해당 구간을 빠르게 버릴 수 있습니다.
- Query 구현에서 오른쪽 자식을 먼저 탐색하도록 구현을 하면 됩니다.
- 전체 시간복잡도는 $O((N + Q)\ log\ N)$이 됩니다.

K. 슬라임 공연장 대청소 (유지원)

Dynamic Programming

출제자 의도 - **Challenging**

- 이 문제는 전형적인 Removing Boxes 형태의 문제입니다.
- 다만 P_x가 임의이기 때문에 많이 합치면 이득이라는 보장이 없습니다.

K. 슬라임 공연장 대청소 (유지원)

Dynamic Programming

출제자 의도 - **Challenging**

- 길이가 x인 같은 색 연속 구간을 어떻게 처리해야 최선이 되는지 계산을 해야합니다.
- 이는 Rod Cutting Problem과 일치합니다.
- $Q_0 = 0$ 와 같은 점화식을 풀어 해결할 수 있습니다.

$$Q_x = \max\left(P_x,\ \max_{1 \le i < x}(Q_i + Q_{x-i})\right)$$

- 전처리 후에 길이 x 짜리 구간의 가치는 Q_x로 둘 수 있게 됩니다.

K. 슬라임 공연장 대청소 (유지원)

Dynamic Programming

출제자 의도 - **Challenging**

- DP 상태를 정의합니다.
- dp(l, r, k): 구간 [l, r]과, C_r과 같은 색의 슬라임 k개가 구간의 오른쪽에 붙어 있는 구간에 대해 이를 최적으로 제거했을 때 얻을 수 있는 최대 점수
- 즉, r번째 슬라임은 오른쪽의 k개와 합쳐져서 $k + 1$ 짜리 블록의 일부가 될 수 있습니다.

K. 슬라임 공연장 대청소 (유지원)

Dynamic Programming

출제자 의도 - **Challenging**

- DP 상태 전이를 정의합니다.
- r번째 슬라임과 k개의 슬라임을 같이 처리하는 경우부터 살펴봅시다.
- 이는 구간 $[l, r-1]$과 $k+1$ 개의 같은 색 연속 구간을 처리하는 것으로 볼 수 있습니다.
- 이때 dp 전이식은 $\mathrm{dp}(l, r, k) = \mathrm{dp}(l, r-1, 0) + Q_{k+1}$과 같이 표현할 수 있습니다.

K. 슬라임 공연장 대청소 (유지원)

Dynamic Programming

출제자 의도 - **Challenging**

- 다음으로 r번째 슬라임을 구간 [l, r-1]과 같이 처리하는 경우를 살펴봅시다.
- 어떤 i ($l \le i < r$)에 대해 $C_i = C_r$이라면, [i+1, r-1]을 없애 i와 r을 붙일 수 있습니다.
- 남은 r을 오른쪽의 k 짜리 구간과 합쳐 i 쪽에 넘겨서 더 큰 블록을 만들 수 있습니다.
- 이때 dp 전이식은 dp(l, r, k) = max(dp(l, i, k+1) + dp(i+1, r-1, 0)) (for all i with $C_i = C_r$)와 같이 표현할 수 있습니다.

K. 슬라임 공연장 대청소 (유지원)

Dynamic Programming

출제자 의도 - **Challenging**

- dp 기저 조건은 dp(l, r, k) = 0 (if $l > r$)로 둘 수 있습니다.
- dp(0, N-1, 0)을 계산하면 해결이 가능합니다.
- 전체 시간복잡도는 $O(N^4)$입니다.

L. 가짜 박홍이 어디에 있을까?? (유지원)

Bitmasking, Dynamic Programming

출제자 의도 - **Challenging**

- Bitmask DP를 활용합니다.
- 각 행에서 복도인 칸들의 위치를 비트마스크로 관리합니다.

L. 가짜 박홍이 어디에 있을까?? (유지원)

Bitmasking, Dynamic Programming

출제자 의도 - **Challenging**

- dp[*mask*]를 이전 행까지 처리했을 때의 최소 비용으로 정의합니다.
- 이때 *mask*는 현재 이어지고 있는 세로 복도 구간의 감시 여부를 나타냅니다.
- *j*번째 비트가 켜져 있으면, *j*열의 세로 복도 구간은 이미 감시된 상태입니다.

L. 가짜 박홍이 어디에 있을까?? (유지원)

Bitmasking, Dynamic Programming

출제자 의도 - **Challenging**

- 현재 행의 복도 칸들을 *cur*이라고 합니다.
- 이전 행의 복도 칸들을 *prev*라고 합니다.

L. 가짜 박홍이 어디에 있을까?? (유지원)

Bitmasking, Dynamic Programming

출제자 의도 - **Challenging**

- 먼저 현재 행에서 가짜 박홍을 배치할 칸의 집합을 정합니다.
- 이 집합을 *install*이라고 하면, *install*은 *cur*의 부분집합입니다.
- 현재 행의 모든 가로 복도 구간마다 적어도 하나의 칸이 선택되어야 합니다.
- 그래야 모든 가로 복도 구간이 감시됩니다.

L. 가짜 박홍이 어디에 있을까?? (유지원)

Bitmasking, Dynamic Programming

출제자 의도 - **Challenging**

- 각 *install*에 대해 배치 비용을 미리 계산해 둡니다.
- *install*에 포함된 칸들의 비용 합이 해당 경우의 추가 비용입니다.

L. 가짜 박홍이 어디에 있을까?? (유지원)

Bitmasking, Dynamic Programming

출제자 의도 - **Challenging**

- 세로 복도 구간도 함께 처리합니다.
- *prev*에는 있었지만 *cur*에는 없는 열은 세로 복도 구간이 끝난 경우입니다.
- 따라서 이 구간은 이전까지 반드시 감시되어 있어야 합니다.
- 즉, 해당 비트가 *mask*에 켜져 있지 않으면 그 상태는 사용할 수 없습니다.

L. 가짜 박홍이 어디에 있을까?? (유지원)

Bitmasking, Dynamic Programming

출제자 의도 - **Challenging**

- *prev*와 *cur*에 모두 있는 열은 세로 복도 구간이 계속 이어지는 경우입니다.
- 이 구간의 감시 여부는 다음 행으로 그대로 넘겨야 합니다.
- 따라서 기존 *mask*에서 이어지는 열의 비트만 남겨 *carry*로 사용합니다.

L. 가짜 박홍이 어디에 있을까?? (유지원)

Bitmasking, Dynamic Programming

출제자 의도 - **Challenging**

- 현재 행에 가짜 박홍을 배치한 열은 세로 복도 구간이 감시됩니다.
- 따라서 다음 상태는 *carry* | *install*이 됩니다.
- 이 값으로 DP를 갱신합니다.

L. 가짜 박홍이 어디에 있을까?? (유지원)

Bitmasking, Dynamic Programming

출제자 의도 - **Challenging**

- 모든 행을 처리한 뒤에는 빈 행을 하나 더 처리합니다.
- 이를 통해 마지막 행에서 끝나는 세로 복도 구간들도 검사할 수 있습니다.
- 마지막에는 이어지는 세로 복도 구간이 없어야 하므로 dp[0]이 답이 됩니다.
- 전 시간복잡도는 $O(N4^M)$입니다.