2024 HICON
홍익대학교 프로그래밍 경진대회 풀이

Official Solutions

by

HICON

2024년 11월 18일

문제		의도한 난이도	출제자
A	와우와 쿼리	**Easy**	swoon
B	드랍 더 비트	**Easy**	xhdtlsid2
C	유물 발굴	**Medium**	swoon
D	혼각대학교	**Medium**	mastershim
E	시큼한 표정	**Advanced**	dosawas
F	시간을 돌리고 싶어	**Advanced**	swoon
G	멘토 매칭하기	**Hard**	swoon
F	구슬 옮기기	**Hard**	smjun0̸4

A. 와우와 쿼리

`implementation`

출제진 의도 – **Easy**

- ✓ 제출 97번, 정답 40명 (정답률 41.237%)
- ✓ 처음 푼 참가자: **보석함**, 1분
- ✓ 출제자: swoon

A. 와우와 쿼리

- ✓ 연속된 문자열의 WOW 등장 횟수를 세어 주는 문제입니다.
- ✓ 반복문으로 $i, i+1, i+2$ 인덱스를 검사해주면 시간복잡도 $\mathcal{O}(N)$ 에 해결할 수 있습니다.

B. 드랍 더 비트

`bruteforcing, math`

출제진 의도 – **Easy**

- ✓ 제출 75번, 정답 38명 (정답률 50.667%)
- ✓ 처음 푼 참가자: **hssong**, 6분
- ✓ 출제자: `xhdtlsid2`

B. 드랍 더 비트



- ✓ N의 제한이 작기 때문에, $2^N \times 2^N$ 가지의 모든 경우를 탐색하여 답을 구할 수 있습니다.
- ✓ 이는 이중 반복문을 이용하여 간단하게 구현할 수 있습니다.

B. 드랍 더 비트

- ✓ 모든 경우를 탐색하지 않고 푸는 것도 가능합니다.

- ✓ $a+b \leq N$ 일 경우, $x \oplus y$ 는 가장 높은 자리부터 $a+b$ 개의 자리가 1 일 때 최대가 됩니다.
- ✓ $a+b > N$ 일 경우, x 와 y 에서 공통으로 1 이 되는 자리가 최소 $a+b-N$ 개 존재합니다.
  이러한 자리들을 $x \oplus y$ 의 가장 낮은 자리부터 채우면 최대가 됩니다.

C. 유물 발굴

`implementation, math`

출제진 의도 – **Medium**

- ✓ 제출 109번, 정답 27명 (정답률 24.771%)
- ✓ 처음 푼 참가자: **hssong**, 16분
- ✓ 출제자: swoon

C. 유물 발굴

- 각 유물의 유물 조각들을 모두 포함하는 가장 작은 직사각형의 크기를 구하는 문제입니다.
- R, C의 제한이 크기 때문에 직접 직사각형을 그려서 구하기는 어렵습니다.
- 효율적인 방법을 생각해봅시다.
  - 직사각형의 윗변은 가장 윗쪽에 있는 유물의 조각보다 높거나 같아야 할 것입니다.
  - 직사각형의 왼쪽 변은 가장 왼쪽에 있는 유물의 조각보다 왼쪽에 있거나 같아야 할 것입니다.
  - ...
- 이 방식으로 생각해보면 해당 유물의 조각에서 가장 위, 아래, 왼쪽, 오른쪽인 유물의 조각의 좌표로 직사각형을 구해낼 수 있습니다.

C. 유물 발굴

- ✓ $arr[i][pos]$ 에 i 번 유물의 조각의 정보를 모아둡시다.
- ✓ $(arr[i][down] - arr[i][up] + 1) \times (arr[i][right] - arr[i][left] + 1)$ 로 i 번 유물의 직사각형을 구할 수 있습니다.
- ✓ 시간복잡도 및 공간복잡도 $\mathcal{O}(N)$ 에 해결할 수 있습니다.



D. 혼긱대학교

`combinatorics`

출제진 의도 – **Easy**

- ✓ 제출 75번, 정답 42명 (정답률 26.667%)
- ✓ 처음 푼 참가자: **This too shall pass**, 36분
- ✓ 출제자: `mastershim`

- ✓ 발음의 구성단위는 음이고, 음의 구성단위는 여러 개의 자음과 한 개의 모음입니다.
- ✓ 즉, 어떤 단어의 발음에 대해 경우의 수를 구하기 위해서는 음의 경우의 수를 먼저 생각해 보아야 합니다.
- ✓ 음은 한 개의 모음과 여러 개의 자음이 앞뒤로 붙어야 하고, 그것이 어떻게 붙느냐에 따라 경우의 수가 정해집니다.

D. 혼각대학교

- 단어에서 모음이 M 개 있을 때, 모음을 각각 $m_1, m_2 \dots m_M$ 이라고 칭할 수 있습니다.
- 이때 당연히 m_1 앞에 있는 자음은 m_1 과 붙는 경우밖에 없습니다.
- 마찬가지로 m_M 뒤에 있는 자음 역시 m_M 과 붙는 경우밖에 없습니다.

- ✓ 그렇다면 모음과 모음 사이에 있는 자음은 어떻게 될까요?
- ✓ m_1 과 m_2 사이의 자음이 J 개 있을 때, $m_1\ j_1 j_2 j_3 ... j_J\ m_2$ 같은 형태일 것입니다.
- ✓ 이때의 경우의 수를 생각해봅시다.
- ✓ m_1 이 아무 자음도 선택하지 않으면, 남은 $j_1 j_2 j_3 ... j_J$ 는 모두 m_2 와 붙게 됩니다.
- ✓ m_1 이 j_1 을 선택하면, 남은 $j_2 j_3 ... j_J$ 는 모두 m_2 와 붙게 됩니다.

$$J+1 = \begin{cases} m_1/j_1j_2j_3...j_Jm_2, \\ m_1j_1/j_2j_3...j_Jm_2, \\ m_1j_1j_2/j_3...j_Jm_2, \\ ... \\ ... \\ m_1j_1j_2j_3...j_J/m_2, \end{cases}$$

이런 방식으로 모든 자음이 m_1 에 붙는 경우부터, 모든 자음이 m_2 에 붙는 경우까지, 총 $J+1$ 가지의 경우의 수가 나온다는 것을 확인해 볼 수 있습니다.

- ✓ m_1 과 m_2 사이의 자음 개수를 J_1
- ✓ m_2 와 m_3 사이의 자음 개수를 J_2
- ✓ ...
- ✓ m_{M-1} 과 m_M 사이의 자음 개수를 J_{M-1} 이라고 했을 때,
- ✓ 각 사건은 서로 영향을 주지 않으므로 발음의 경우의 수는 $(J_1+1)(J_2+1)...(J_{M-1}+1)$ $mod\, 10^9+7$ 이 됩니다.

- ✓ 이 때 경우의 수가 너무 커져서 overflow가 생기거나 시간초과를 받을 수 있습니다.
- ✓ 곱셈을 할 때 꼬박꼬박 mod 연산을 해주시면 이를 방지할 수 있습니다.
- ✓ 모든 단어 길이의 합을 S라고 할 때, 총 시간 복잡도는 $O(S)$ 입니다.
- ✓ 테스트케이스의 수가 많으므로 빠른 입출력을 사용해야지만 시간초과를 방지할 수 있습니다.

E. 시큼한 표정

`implementation`

출제진 의도 – **Advanced**

- ✓ 제출 13번, 정답 4명 (정답률 30.769%)
- ✓ 처음 푼 참가자: **dicohy27**, 66분
- ✓ 출제자: dosawas

E. 시큼한 표정



- 먼저, 문자열에 있는 모든 시큼한 표정의 위치를 찾아봅시다.
- 시큼한 표정의 정의에 따라서 시큼한 표정 최대 substring의 개수는 $O(N)$개 입니다.
- 따라서, 어떤 시작점을 정해두고 시큼한 표정이 아닐 때까지 탐색을 하면 $O(N)$만에 모든 시큼한 표정을 찾을 수 있습니다.
- 한번 탐색한 부분은 다시 탐색할 필요가 없지만, 이 문제는 제한이 작아서 $O(N^2)$으로 시큼한 표정을 모두 찾아도 됩니다.

- 시큼한 표정을 찾았다면 그 표정의 양 끝 index를 구해서 문자열의 앞 뒤로 없애야할 문자의 개수를 쉽게 찾을 수 있습니다.

E. 시큼한 표정

- ✓ 위의 방법으로 구한 각각의 시큼한 표정을 만드는 동전 던지는 경우의 수를 구해서 합치면 됩니다.
- ✓ 어떤 시큼한 표정이 나오려면 문자열의 앞에서 문자가 x 개, 뒤에서 y 개가 없어져야한다는 것은 고정입니다.

- ✓ 이제 그 앞면과 뒷면이 나오는 순서만 고려하면 됩니다.
- ✓ 그 순서의 경우의 수는 $x+y$ 개의 자리에서 앞면의 자리 x 개를 골라주는 경우의 수와 같습니다.
- ✓ 각 시큼한 표정마다 $\binom{x+y}{x}$ 를 계산하여 더해주면 정답입니다.

- ✓ $O(N)$ 만에 시큼한 표정을 구했다면 $\binom{x+y}{x}$ 는 $O(N)$ 에 계산해도 됩니다.
- ✓ $O(N^2)$ 에 시큼한 표정을 구했다면 $\binom{x+y}{x}$ 는 dp 테이블로 이항계수를 전처리해두고 $O(1)$ 만에 계산해야 합니다.
- ✓ 계산 중간중간에 모듈로 연산을 해서 오버플로우가 나지 않도록 주의해야 합니다.

F. 시간을 돌리고 싶어

parametric-search, binary-search, sweeping

출제진 의도 – **Advanced**

- ✓ 제출 16번, 정답 3명 (정답률 18.750%)
- ✓ 처음 푼 참가자: **dicohy27**, 81분
- ✓ 출제자: swoon

F. 시간을 돌리고 싶어

- ✓ 문제를 풀기 위해 함수를 정의해봅시다.
- ✓ $f(i, j) =$ 타임머신을 j 번 사용하여 i 일차에 도착했을 때, 최소 T
- ✓ 일단 N, K 제한이 크므로 저 두가지 인자를 모두 사용하여 상태를 정의하긴 어렵습니다.
- ✓ 함수를 다른 방식으로 정의해봅시다.

F. 시간을 돌리고 싶어

- ✓ $f(i) = T$ 가 i 일때 1 일까지 갈 수 있는가?
- ✓ $f(i)$ 에 1 부터 $N-1$ 을 모두 넣어보고 가능한 가장 큰 값이 답이 될 것입니다.
- ✓ 그렇게 진행하면 시간복잡도는 $\mathcal{O}(NK)$ 가 됩니다.
- ✓ 만약 $T = i$ 일때 1 일로 돌아가는 것을 성공했는데 $T = i+1$ 일때 성공하지 못할 수는 없습니다.
- ✓ $f(i)$ 의 단조성을 이용하여 이분탐색을 진행해주면 $\mathcal{O}(KlogN)$ 으로 해결할 수 있습니다.

F. 시간을 돌리고 싶어

- ✓ $f(i)$를 구현할 때 중요한 점이 있는데, 타임머신을 타고 이동한 날과 가장 가까운 타임머신이 켜진 날을 빠르게 구해야합니다.
- ✓ 이는 이분탐색이나 스위핑 전처리로 해결할 수 있습니다.

G. 멘토 매칭하기

`combination, sorting`

출제진 의도 – **Hard**

- ✓ 제출 0번, 정답 0명 (정답률 0%)
- ✓ 처음 푼 참가자: -, -분
- ✓ 출제자: swoon

G. 멘토 매칭하기

- 먼저, 매칭을 완료하였을때의 학생의 실력의 최솟값을 최대화하는 방법은 다음과 같습니다.
- 학생의 실력과 멘토의 지도력을 모두 정렬해두고, 학생은 앞부터 멘토는 뒤부터 매칭을 시켜줍니다.
- 이렇게 구한 학생의 최소실력을 이용하여 경우의 수를 구해봅시다.

G. 멘토 매칭하기

- ✓ i번 학생의 실력이 A_i, 학생 실력의 최솟값을 x라고 두겠습니다.
- ✓ $A_i < x$인 경우
  - 학생의 실력이 x이상이 되어야 하므로 지도력이 $x - A_i$이상인 멘토와 매칭시켜줘야 합니다.
  - 그러면 매칭시키는 방법의 수는 남은 멘토 중 $x - A_i$이상인 멘토의 수만큼 곱해집니다.
  - $A_x < A_y$인 학생이 있다고 했을 때, A_x를 먼저 매칭시켜주면 A_y인 학생 역시 매칭시킬때 있어 문제가 생기지 않습니다.

G. 멘토 매칭하기

- ✓ i번 학생의 실력이 A_i, 학생 실력의 최솟값을 x라고 두겠습니다.
- ✓ $A_i >= x$인 경우
  - 이런 학생은 매칭을 시켜도 되고 시키지 않아도 됩니다.
  - 이런 학생의 수가 X 명, 남은 멘토의 수가 Y 명이라고 둬 봅시다.
  - 서로 다른 상자 X 개에 서로 다른 공 Y 개를 최대 하나까지 넣어 배분하는 문제가 됩니다.
  - $$\sum_{k=0}^{min(X,Y)} \binom{X}{k}\binom{Y}{k}k!$$
  - 해당 식을 계산하기 위해 모듈러 역원을 구해야 합니다.

- ✓ 위 두 케이스로 얻은 방법의 수를 곱해주면 전체 경우의 수가 됩니다.
- ✓ 총 시간복잡도는 $\mathcal{O}((N+M)logN + log(MOD))$ 입니다.

H. 구슬 옮기기

`bipartite_matching`

출제진 의도 – **Hard**

- ✓ 제출 7번, 정답 0명 (정답률 0%)
- ✓ 처음 푼 참가자: -, -분
- ✓ 출제자: `smjun04`

H. 구슬 옮기기

- ✓ 구슬 보드판의 목표 상태에 도달하기 위한 조건을 생각해 봅시다.
- ✓ 구슬을 추가하거나 제거할 수 없으므로 초기 상태와 목표 상태의 구슬 갯수는 동일해야 합니다.
- ✓ 만약 구슬의 갯수가 다르다면 불가능한 경우이므로 -1을 출력합니다.
- ✓ 이외의 경우에는 무조건 목표 상태에 도달할 수 있습니다.

H. 구슬 옮기기

- 구슬을 상하좌우 인접한 빈 칸으로 옮기는 동작은 1회의 이동 횟수를 소모합니다.
- 이 행동을 1번 동작이라고 하겠습니다.
- 원하는 칸의 구슬을 제거하고 저장하는 동작은 반드시 저장한 구슬을 원하는 빈 칸에 넣는 동작으로 이어져야 합니다.
- 이 일련의 과정은 2회의 이동 횟수를 소모합니다.
- 이 행동을 2번 동작이라고 하겠습니다.
- 만약 두 동작이 같은 결과를 불러온다면, 1번 동작을 실행하는 것이 이득입니다.

H. 구슬 옮기기

- 구슬 보드판의 초기 상태에서 값이 x인 칸이 목표 상태에서도 값이 x라면, 그 칸은 처리할 필요가 없습니다.
- 초기 상태에서 1인 칸이 목표 상태에서 0이라면, 그 칸은 구슬을 없애야 하는 칸입니다.
- 이런 종류의 칸을 0+ 칸이라고 하겠습니다.
- 초기 상태에서 0인 칸이 목표 상태에서 1이라면, 그 칸은 구슬을 채워야 하는 칸입니다.
- 이런 종류의 칸을 1- 칸이라고 하겠습니다.
- 초기 상태와 목표 상태의 0/1 칸 갯수가 같으므로, 0+ 칸과 1- 칸의 갯수도 같습니다.

H. 구슬 옮기기

- ✓ 인접한 0+ 칸과 1- 칸은 1번 동작으로 처리할 수 있습니다.
- ✓ 최대한 많은 1번 동작을 사용해야 하므로, 이 문제는 최대 매칭 문제로 볼 수 있습니다.
- ✓ 각 0+ 칸에서 상하좌우 4방향에 있는 1- 칸으로 향하는 간선을 설정하면 이분 그래프가 됩니다.
- ✓ 이분 매칭 알고리즘을 사용하면 최대로 사용할 수 있는 1번 동작의 수를 구할 수 있습니다.
- ✓ 매칭되지 않은 0+ 칸과 1- 칸은 2번 동작으로 해결할 수 있습니다.
- ✓ 따라서 최소 이동 횟수는 (1번 동작의 수) + (2번 동작의 수) * 2가 됩니다.