보드게임컵
해설

2023년 1월 14일

문제 일람

Easy / Medium / Hard / Challenging

A 노 땡스!
cozyyg

B 할리갈리
havana723

C 크레이지 타임
havana723

D Yacht Dice
shiftpsh

E 벚꽃 내리는 시대에 결투를
havana723 shiftpsh

F 도미니언
stonejjun

G 모든 곳을 안전하게
cozyyg

H 테라포밍 마스
pichulia

I KPvK 엔드게임
cologne

J 브루마블
79brue

K 100% Orange Juice!
pichulia

L 보드게임컵 파티!
cozyyg

M 더블 아웃
cozyyg

N 수 나누기 게임
havana723

A

노 땡스!

Easy #implementation

- 제출 685번, 정답 557명 (정답률 82.482%)
- 처음 푼 사람: **tlsdydaud1**, 1분 7초
- 출제자: cozyyg

A 노 땡스!

- 카드에 적힌 정수가 오름차순으로 주어집니다.
- 첫 번째 수는 항상 정답에 더해집니다.
- x_i는 x_{i-1}과의 차이가 1이 아닐 때만 정답에 더해집니다.
- 이전 수를 `prv`라는 변수에 저장하고, x_i와의 비교가 끝나면 `prv`를 x_i로 갱신하는 방식으로 풀 수 있습니다.
  - `prv` 변수의 초기화에 유의합시다. 2보다 작은 값으로 초기화해야 합니다.

B

할리갈리

Easy #implementation

- 제출 700번, 정답 530명 (정답률 78.571%)
- 처음 푼 사람: **puppy**, 2분 2초
- 출제자: havana723

B 할리갈리

- 각 과일이 몇 개 나왔는지를 저장하고 하나 이상의 과일이 정확히 5개 있을 경우 YES를 출력하면 됩니다.
- 각 과일의 개수를 세는 방법은 여러 가지가 있는데,
  - 입력으로 들어오는 과일의 종류가 4가지이므로 4개의 변수를 만든 위 `if`문을 통해 구해주거나
  - `map<string, int>`(Python의 `dict`)를 통해 구해줄 수 있습니다.

C

크레이지 타임

Easy #implementation

- 제출 599번, 정답 461명 (정답률 77.796%)
- 처음 푼 사람: **bnb2011**, 6분 39초
- 출제자: havana723

C 크레이지 타임

- 현재 시각과 일치하는 시각의 카드가 들어오면,
  - 들어온 문자열이 HOURGLASS와 같은지 확인합니다. 같다면 NO, 다르다면 YES를 출력하면 됩니다.
- 그렇지 않은 경우,
  - 들어온 문자열이 HOURGLASS라면 시간 진행의 방향을 바꿔야 합니다. `rev` 또는 비슷한 `bool` 변수를 만들고, 이 경우마다 `rev = ! rev`를 해 줍시다.
  - 이 경우 출력은 항상 NO입니다.
- `rev`가 `false`라면 현재 시각에 1을 더해 주고, 아니라면 현재 시각에서 1을 빼 줍시다.
  - 1시에서 1시간 전은 0시가 아닌 12시이고, 12시에서 1시간 후는 13시가 아닌 1시임에 주의합시다.

D

Yacht Dice

Medium #bruteforcing #implementation

- 제출 1,089번, 정답 309명 (정답률 28.834%)
- 처음 푼 사람: **golazcc83**, 9분 18초
- 출제자: shiftpsh

D Yacht Dice

- 나머지 두 개의 주사위의 눈으로 될 수 있는 모든 경우에 대한 점수를 계산합니다. 총 6×6개의 경우가 있습니다.
- 다음 정보들을 미리 계산해 놓으면 점수를 계산하기 수월합니다.
  - $1, 2, 3, 4, 5, 6$의 등장 횟수
  - 정렬된 주사위 눈

등장 횟수를 통해 **Ones**부터 **Sixes**까지와 **Four of a Kind**, **Yacht** 조건을 체크할 수 있습니다. **Full House**도 체크할 수 있지만 정렬된 배열을 이용한 조금 더 간단한 방법이 있습니다.

D Yacht Dice

정렬된 배열을 S라고 합시다.

- **Full House**: 다음 중 하나를 만족하면 됩니다.
  - $S_1 = S_2$ and $S_2 \neq S_3$ and $S_3 = S_5$
  - $S_1 = S_3$ and $S_3 \neq S_4$ and $S_4 = S_5$
- **Little Straight**: $S = [1, 2, 3, 4, 5]$.
- **Big Straight**: $S = [2, 3, 4, 5, 6]$.

D Yacht Dice

문제를 디버깅하는 데에 도움이 될 수 있는 반례 몇 개입니다.

- $2, 2, 2, 2, 2$: 이 경우는 Full House로 취급할 수 없습니다.
- $2, 2, 2, 3, 3$과 $2, 2, 3, 3, 3$: 두 경우 모두가 Full House입니다.
- $5, 5, 5, 5, 5$: Four of a Kind로 취급할 경우 25점이 아닌 20점입니다.

E

벚꽃 내리는 시대에 결투를

Medium #dp

- 제출 528번, 정답 91명 (정답률 17.803%)
- 처음 푼 사람: **jjang36524**, 19분
- 출제자: shiftpsh

- i 번째 공격까지 받았고, 라이프가 j 남았을 때 최대한으로 남길 수 있는 오라에 대해 생각해 봅시다.
- 우선 dp 를 $-\infty$ 로 초기화하고, $dp\,(0, L) = A$ 로 둡니다.

현재 처리해야 하는 공격이 「X/Y」일 때,

- $i+1$ 번째 공격을 오라로 받을 수 있다면, 즉 $dp\,(i,j) \geq X$ 라면:

$$dp\,(i+1,j) \xleftarrow[\max]{} dp\,(i,j) - X.$$

- $i+1$ 번째 공격을 라이프로 받을 수 있다면, 즉 $j > Y$ 라면:

$$dp\,(i+1,j-Y) \xleftarrow[\max]{} dp\,(i,j)\,.$$

현재 처리해야 하는 공격이 「$X/-$」라면:

$$dp\,(i+1,j) \xleftarrow[\max]{} \max\left\{dp\,(i,j)-X,0\right\}.$$

현재 처리해야 하는 공격이 「$-/Y$」일 때, $j > Y$ 라면:

$$dp\,(i+1,j-Y) \xleftarrow[\max]{} dp\,(i,j).$$

- 모든 $1 \leq x \leq L$에 대해 $dp\,(N, x) \geq 0$인 것이 하나라도 있다면 이번 턴에서 살아남을 수 있습니다.
- 역추적을 위해 $\max$ 연산 시마다 이전 상태를 저장해 둬야 함에 유의합시다.

F

도미니언

Hard #Greedy #Simulation

- 제출 82번, 정답 3명 (정답률 3.659%)
- 처음 푼 사람: **hyperbolic**, 112분
- 출제자: stonejjun03

F 도미니언

정수 A, B 가 쓰여진 행동카드는 (A, B) 로 표기하겠습니다.

먼저 최종 턴에 대한 생각을 해봅시다.

- (a, b) 에서 $a > 0, b > 0$ 인 모든 카드들은 사용해도 손해를 보지 않습니다. 따라서 다 쓰고 시작을 한다고 가정합니다.
- $(0, 1), (1, 0)$ 는 사용하면 무조건 손해를 봅니다. 따라서 덱에서 제외를 하고 시작한다고 가정합니다.
- 점수는 들고 손에 들고 있을 수 있는 카드 장수의 최댓값에 의해 결정됩니다. 목표는 행동을 통하여 가장 많은 카드를 손에 들고 있는 것 입니다.

F 도미니언

- $(0, b_1), (0, b_2)$ 의 두 가지 카드가 있을 때 $b_1 > b_2$ 라면 $(0, b_1)$ 을 일단 쓰는 것이 항상 더 좋습니다.
- 따라서 행동 카드들을 $(a, 0)$ 그룹과 $(0, b)$ 그룹으로 나눈 뒤 각 그룹의 카드들을 내림차순 정렬해둡시다.
- 가장 많은 카드를 들고 있기 위해서는 남은 행동 횟수가 1 일 때는 $(a, 0)$ 카드를, 그렇지 않을 때는 $(0, b)$ 카드를 정렬해둔 순서대로 사용하는 것이 가장 좋습니다.
- 하지만 (a, b) 에서 $a > 0, b > 0$ 인 카드를 모두 사용했을 때, 행동 횟수가 1 이며 손에 있는 카드가 1장인 경우는 행동 카드 한 장만을 사용할 수 있기 때문에 예외 처리를 해야 합니다.

F 도미니언

시작할 때 가지고 있는 행동 횟수 i 에 대해서 최대로 들고 있을 수 있는 카드의 수 $card[i]$ 를 구해봅시다.

- 맨 처음에는 $(a, 0)$ 카드는 한 장도 사용하지 않고 $(0, b)$ 카드는 모두 쓴다고 가정하고 시작합니다.
- 행동이 부족할 때마다, 가장 좋은 $(a, 0)$ 카드를 한 장씩 더 쓴다고 가정하며, $(a, 0)$ 카드가 남아있지 않다면 가장 좋지 않은 $(0, b)$ 카드부터 쓰지 않는다고 가정합니다.
- 위 과정을 i 를 덱에 있는 점수 카드의 수 $+ \alpha$ 에서 1까지 줄여가면서 반복하면 $card[i]$ 테이블을 채우는 과정을 $O(N)$ 에 할 수 있습니다.

F 도미니언

- 아무 카드도 사지 않았을 때와, 각 카드를 구매했을 경우에 대해서 $card$ 배열을 이용하여 손에 들 수 있는 최대 카드 수를 구할 수 있으며, 이를 이용하여 얻을 수 있는 최대 점수를 구할 수 있습니다.
- 아무 카드도 사지 않았을 때의 경우는 구매할 수 있는 점수 카드 중 P가 가장 큰 카드를 구매했다고 한 후 점수를 구하면 됩니다.
- 정렬 과정을 포함하여 전체 시간 복잡도 $\mathcal{O}\left(N \log N + M\right)$에 문제를 해결할 수 있습니다.

G

모든 곳을 안전하게

Medium #bruteforcing #implementation #case_work

- 제출 890번, 정답 212명 (정답률 24.382%)
- 처음 푼 사람: **dlalswp25**, 32분
- 출제자: cozyyg

G 모든 곳을 안전하게

풀이 1: 브루트포스 알고리즘

- $0 \le i \le n - x$이고 $a_i \ge 1$인 i마다, i번 칸의 말을 $i + x$번 칸으로 옮겼을 때 안전한 상태가 되는지 판정하면 됩니다.
- **말이 1개 있는 칸의 개수**를 K라는 변수로 정의하면, $K = 0$일 때만 안전한 상태입니다.
- i번 칸의 말을 $i + x$번 칸으로 옮길 때 K가 변하는 경우는 다음과 같습니다.
  - $a_i = 1$인 경우 K가 1 감소
  - $a_i = 2$인 경우 K가 1 증가
  - $a_{i+x} = 0$인 경우 K가 1 증가
  - $a_{i+x} = 1$인 경우 K가 1 감소
- 따라서 초기 상태에서의 K 값을 미리 계산해두면 각 이동 방법마다 $K = 0$인지를 $\mathcal{O}(1)$에 판정할 수 있습니다.

풀이 2: 케이스 워크

- 초기 상태에서 **말이 1개 있는 칸의 개수**를 K라 합시다.

1. $K > 2$: 항상 불가능합니다.
2. $K = 2$: $a_i = 1$인 두 i의 차이가 정확히 x일 때만 가능합니다.
3. $K = 1$: $a_i = 1$인 i에 대해, $a_{i-x} \geq 3$이거나 $a_{i+x} \geq 2$일 때만 가능합니다.
4. $K = 0$: $a_i \geq 3$이고 $a_{i+x} \geq 2$인 i가 존재할 때만 가능합니다.

Challenge: 만약 실버가 성질이 급하지 않아서 주사위 눈을 모두 남겨놨다면?

- 사실 이게 원본 문제였습니다.
- 일단 브루트포스 알고리즘으로는 잘 안 될 것 같습니다.
- 케이스 워크 풀이를 확장해야 합니다.
- **케이스가 그렇게 많을 것이라고는 생각하지 못했습니다...**
  - 심지어 두 주사위 눈이 같은 경우는 따로 처리해야 합니다...
- 풀이를 완성하긴 했지만, **모두의 행복**을 위해 대회에는 쉬운 버전의 문제를 출제했습니다.
- 어려운 버전은 추후 공개될 예정입니다. 많은 관심 부탁드립니다.

H

테라포밍 마스

Hard #greedy #simulation

- 제출 73번, 정답 11명 (정답률 15.068%)
- 처음 푼 사람: **yooyou7**, 23분
- 출제자: pichulia

가장 먼저 다음과 같은 그리디적인 관찰이 요구됩니다.

- 점수는 항상 마지막 Z 일에 연속적으로 획득하는 것이 좋다.
- 생상력은 항상 초반에 연속적으로 올리는 것이 좋다.

점수를 먼저 먹는다고 해서 얻는 이득이 따로 없기 때문에, 가능하면 항상 후반에 연속해서 Z 일에 점수를 획득해도 아무런 손해가 발생하지 않습니다.

비슷한 논리로, 생산력도 중간에 올리지 않고 나중에 생산력을 올린다고 해서 얻는 이득이 없기 때문에, 생산력은 가능하면 항상 초반에 얻도록 해도 아무런 손해가 발생하지 않습니다.

이 논리를 통해, 우리는 “최종적으로 도달할 생산력 값” C 를 고정시켜놓고, 생산력이 C 이상이 될 때까지는 항상 생산력을 올리고, 그 이후로는 항상 점수를 얻도록 하는 방식을 떠올릴 수 있습니다. 가능한 $B \leq C \leq Y$ 에 대해서 필요한 최소 턴 수를 계산해본 뒤, 이 중 최소값을 출력해보면 됩니다.

예외로 $Y < B$ 인 경우에는 기본 생산력만으로 항상 점수를 얻을 수 있으므로, 이 경우에는 답이 Z 가 됩니다.

H 테라포밍 마스

초기 자원 A, 초기 생산력 B 인 상태로 시작해서 다음과 같은 시뮬레이션을 돌려봅니다.

- 턴 수++
- 자원 += 생산력
- **if** $(\text{생산력} < C)$ **and** $(\text{자원} \geq X)$: 자원 -= X, 생산력++
- **if** $(\text{턴 수} \geq \text{“최소 턴 수”} - Z)$: 자원 -= Y

여기서 “최소 턴 수” 값은, 생산력이 C 가 되는 순간의 턴 수 이상, 이 턴 수 $+ Z$ 이하의 값을 가질 것입니다.
해당 범위 내에서 “최소 턴 수” 값을 이분탐색으로 결정한 뒤, 자원 값이 항상 음이 아닌 정수가 될 수 있는지를 체크하면 최소 턴 수를 결정할 수 있습니다.

I

KPvK 엔드게임

Challenging #game_theory #implementation

- 제출 5번, 정답 0명 (정답률 0.000%)
- 처음 푼 사람: -
- 출제자: cologne

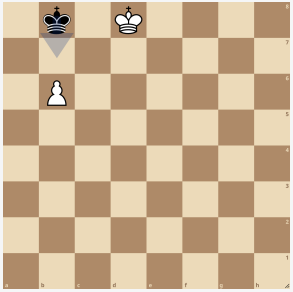
- 두 킹과 기물 하나가 있는 경우의 수는 $64^3 \times 5$ 보다 작거나 같습니다.
- 모든 상태를 전부다 탐색해보는 전략을 사용할 수 있습니다.

- DP를 바로 사용하는 경우, 순환 참조가 생깁니다.
- 사이클을 해결하기 위해 다른 방법이 필요합니다.
- 모든 체크메이트인 상태부터 시작해서, 이기는 경우만 방문합니다.
- 어떤 위치에서 흰색 측이 이긴다는 의미는, 흰색 측의 움직임이고 이기는 상태로 가는 방법이 존재하거나, 검은색 측의 움직임이고 다음 모든 가능한 움직임이 이기는 상태라는 의미입니다.
- 현재 어떤 측의 차례인지에 따라 바로 탐색을 할지, 해당 노드에서 만들 수 있는 모든 상태를 전부 탐색한 이후 탐색을 할지 정해주면 됩니다.

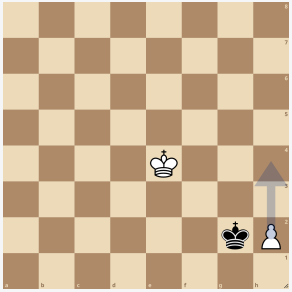
- 다음과 같은 디테일들이 있습니다.
  - 비숍 혹은 나이트로 승진하는 경우는 항상 비깁니다.
  - 단, 룩으로 승진해야하는 경우는 존재합니다.
  - 답의 최댓값은 26 입니다(항상 50 수 이내에 끝납니다.)
  - 폰이 두 칸 가는 경우를 잘 처리합니다.
  - 퀸 혹은 룩과 검은색 킹이 일직선으로 있어도, 가운데 하얀색 킹이 있어서 체크가 아닌 경우를 생각합니다.
  - 폰은 앞으로만 이동가능하고, 대각선으로는 이동이 불가능하다는 것을 잘 구현합니다.

I KPvK 엔드게임

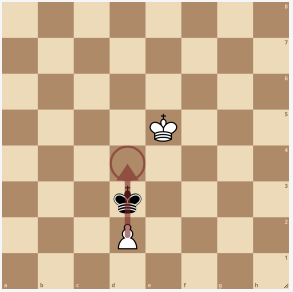
- 다음은 디버깅에 도움이 되는 몇 가지 예제입니다.



Black to move, Kb7, Draw

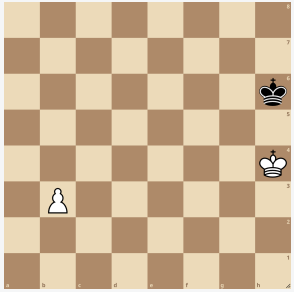


White to move, h4, #10

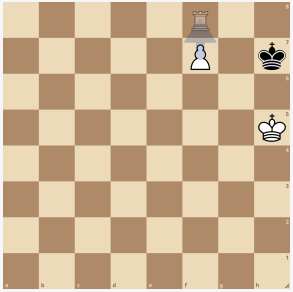


White to move, Draw

- 다음은 디버깅에 도움이 되는 몇 가지 예제입니다. (Cont'd)



Black to move, #26; Opposition



White to move, f8=R!, #6

J

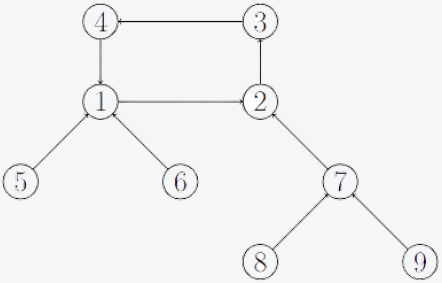
브루마블

Challenging #data_structures #divide_and_conquer

- 제출 5번, 정답 0명 (정답률 0.000%)
- 처음 푼 사람: -
- 출제자: 79brue

- 문제의 상황을 그래프로 해석해 봅시다.
- 각 정점에서 나가는 간선이 하나씩 있는 방향 그래프가 됩니다.
- 이러한 그래프를 **Functional Graph**라고 합니다.

- Functional Graph의 한 연결성분은 중심 사이클 하나와, 그 사이클로 향하는 트리 여러 개로 이루어져 있습니다.
- 이때 사이클에 속한 행사 쿼리를 **사이클 행사**, 사이클 밖에 속한 행사 쿼리를 **트리 행사**라고 합시다.



전처리

- 먼저 전처리를 통해 그래프의 구조를 분석합니다.
- 각 연결성분에 대해 임의의 점에서 이미 방문한 점이 나올 때까지 계속 진행하다 보면 사이클에 속한 정점 목록을 순서대로 알아낼 수 있습니다.
- 이후 사이클 밖에 있는 트리 정점들이 어떤 사이클의 어떤 정점으로 연결되어 있는지도 계산해 줍니다.
- 이렇게 하면 전처리 과정은 $\mathcal{O}(NM)$에 끝이 납니다.
- 앞으로 질문 쿼리에 답할 때는 사이클 정점 위의 행사로 얻는 점수와 트리 정점 위의 행사로 얻는 점수를 따로 계산해 더해 줄 것입니다.

사이클 행사 계산하기

- 만약 모든 행사 쿼리가 먼저 주어진 후 질문 쿼리가 주어졌다면 어떻게 될까요?
- 이 경우 다음과 같은 방법을 통해서 총 $\mathcal{O}(Q \log Q)$ 에 풀 수 있습니다.

- 어떤 출발점 s에서 출발했을 때, 충분한 턴 횟수 이후 어떤 사이클 C로 진입하며, 이때 사이클의 정점 중 가장 먼저 방문하게 되는 점이 e라고 합시다.
- 또한 e가 사이클 C의 idx_e번째 정점이고, 이 점까지 진입하는 데 d_s번의 턴이 걸렸다고 합시다.
- 마지막으로 사이클의 길이를 L이라고 합시다.
- 이때, T번째 턴에 사이클의 x번째 점에서 열리는 행사에 참여할 수 있을 조건은 $T - d_s + idx_e \equiv x \mod L$ 이고 $T \geq d_s$ 입니다.

- 위의 합동식을 다시 쓰면 $T - x \equiv d_s - idx_e \mod L$ 이 됩니다.
- 좌변은 모두 행사에 관한 항이고, 우변은 모두 질문 쿼리에 대한 항입니다.
- 이 값이 일치하는 경우에만 두 번째 조건이 판별하는지 보아도 되기 때문에, 이 값이 일치하는 행사 쿼리, 질문 쿼리들끼리 모아줍니다.
- 이 과정은 위 값을 가중치로 둔 쿼리 객체를 만들고 이 가중치에 따라 정렬하는 방법으로 가능합니다.
- 이제 T 또는 d_s 의 내림차순으로 남은 쿼리를 정렬하면 질문 쿼리들에 대해 쉽게 답을 구할 수 있습니다.

- 실제로는 행사 쿼리와 질문 쿼리가 섞여 있기 때문에 위 방법을 사용할 수 없습니다.
- 그러나 쿼리에 대한 분할 정복을 하면 위 방법을 사용할 수 있게 됩니다.
- 쿼리를 순서에 따라 반으로 쪼갠 뒤, 앞쪽 반의 행사 추가 쿼리와 뒤쪽 반의 질문 쿼리에 대해서 계산합니다.
- 이 과정을 앞쪽 반과 뒤쪽 반에 대해서 재귀적으로 반복하면 답을 구할 수 있게 되고, 이때 시간 복잡도는 $\mathcal{O}\left(Q \log^2 Q\right)$ 가 됩니다.

트리 행사 계산하기

- 시작점이 s이고, 트리 상에서 그 깊이가 d_s일 때, T턴에 정점 p에서 열리는 행사에 참여할 조건을 구해 봅시다. 이때 p의 깊이는 d_p입니다.
- 이 조건은 $T = d_s - d_p$, 그리고 정점 p가 정점 s의 조상이어야 합니다.
- 앞의 조건은 사이클 쿼리를 계산할 때와 같은 방식으로 처리할 수 있습니다.
- 뒤의 조건까지 보기 위해서는 질문 쿼리에 대해 자신의 조상 정점에 쓰인 수의 합을 계산하는 HLD를 구현할 수 있습니다.
- 더 쉬운 방법은 행사 쿼리에 대해 서브트리 덧셈 쿼리를 사용하는 것인데, 이 경우 dfs ordering과 Fenwick Tree를 이용하면 구현이 간편해집니다.
- 양쪽 모두 시간 복잡도는 $\mathcal{O}\left(Q \log NM + Q \log Q\right)$입니다.

- 위 세 가지 과정을 모두 구현하면 $\mathcal{O}\left(NM + Q\log^2 Q + Q\log NM\right)$에 문제를 해결할 수 있습니다.
- 사이클 쿼리 처리를 $\mathcal{O}\left(Q\log Q\right)$에 할 수도 있으나, 이 경우 구현이 더 복잡해집니다.

K

100% Orange Juice!

Hard #dp #simulation

- 제출 40번, 정답 16명 (정답률 40.000%)
- 처음 푼 사람: **moonrabbit2**, 43분
- 출제자: pichulia

다음과 같은 DP를 구성해 봅시다.

$$dp\,(i, j, k) = \text{공격수의 체력이 } i\text{, 수비수의 체력이 } j\text{일 때 공격수가 승리할 확률.}$$
$$(k = 0\text{이라면 선공의 공격 상황, } k = 1\text{이라면 후공의 공격 상황})$$

위와 같이 정의한 경우 상태 전이 그래프는 공격과 수비 모두가 회피에 성공하는 경우를 제외하고 DAG의 형태로 잘 정의됩니다.

이런 경우가 없다면 공격수의 주사위와 수비수의 주사위에 따라 나올 수 있는 모든 6×6가지 경우에 대해 시뮬레이션하고, 그 결과에 따라 i 혹은 j에서 데미지를 차감한 뒤 모든 경우의 확률을 합해 6×6으로 나누면 됩니다.

공격과 수비 모두가 회피에 성공하는 경우에는 식이 자기 자신을 간접적으로 참조하는 경우가 생기므로 주의해야 합니다.

예를 들어 $dp\,(i, j, 0)$ 상황에서 회피에 성공하는 경우는 $dp\,(j, i, 1)$ 을 참조하며, $dp\,(j, i, 1)$ 에서도 회피를 성공하면 다시 $dp\,(i, j, 0)$ 를 참조하게 됩니다.

이를 해결하기 위해 $dp\,(i, j, 0)$ 에서 $dp\,(j, i, 1)$ 을 특별하게 고려하여 점화식을 바꿔 줍시다.

$dp\,(i,j,k)$에서 가능한 $6\times 6=36$가지 경우 중 시뮬레이션 결과 회피에 성공한 경우의 수를 $E_{i,j,k}$이라고 합시다. k에 대해 일반성을 잃지 않고,

$$\begin{aligned} dp\,(i,j,0) &= (\text{회피 성공 확률})\times dp\,(j,i,1) + (\text{회피 실패 확률})\times \frac{\text{회피 실패하는 경우의 } dp \text{ 합}}{\text{회피 실패하는 경우의 수}} \\ &= \frac{E_{i,j,0}}{36}\times dp\,(j,i,1) + \frac{36-E_{i,j,0}}{36}\times \frac{\text{회피 실패하는 경우의 } dp \text{ 합}}{36-E_{i,j,0}} \\ &= \frac{E_{i,j,0}}{36}\times dp\,(j,i,1) + \frac{\text{회피 실패하는 경우의 } dp \text{ 합}}{36} \end{aligned}$$

$dp\,(i,j,k)$에서 회피 실패하는 경우로 상태전이를 했을 때의 dp 합은 자기 자신을 참조하지 않고 계산할 수 있습니다. 식 정리의 편의를 위해 $P_{i,j,k}$로 나타냅시다.

이제 이 연립방정식을 해결하면 됩니다.

$$dp\,(i,j,0) = \frac{E_{i,j,0}}{36} \times dp\,(j,i,1) + \frac{P_{i,j,0}}{36}$$

$$dp\,(j,i,1) = \frac{E_{j,i,1}}{36} \times dp\,(i,j,0) + \frac{P_{j,i,1}}{36}$$

$$\therefore dp\,(i,j,0) = \frac{E_{i,j,0}}{36} \times \left[\frac{E_{j,i,1}}{36} \times dp\,(i,j,0) + \frac{P_{j,i,1}}{36}\right] + \frac{P_{i,j,0}}{36}$$

$$\Rightarrow \left(1 - \frac{E_{i,j,0}}{36} \times \frac{E_{j,i,1}}{36}\right) dp\,(i,j,0) = \frac{E_{i,j,0}}{36} \times \frac{P_{j,i,1}}{36} + \frac{P_{i,j,0}}{36} \qquad \text{(cont'd)}$$

$$\Rightarrow dp\,(i,j,0) = \left(\frac{E_{i,j,0}}{36} \times \frac{P_{j,i,1}}{36} + \frac{P_{i,j,0}}{36}\right) / \left(1 - \frac{E_{i,j,0}}{36} \times \frac{E_{j,i,1}}{36}\right)$$

$$\therefore dp\,(i,j,0) = \frac{E_{i,j,0} \times P_{j,i,1} + 36 \times P_{i,j,0}}{36^2 - E_{i,j,0} \times E_{j,i,1}}$$

정리된 식은 자기 자신을 참조하지 않으므로 이제 모든 DP 테이블을 채울 수 있습니다.
정의에 따라 초기값 $dp\,(i,0,k) = 1$ 이며 $dp\,(0,j,k) = 0$ 입니다.

L

보드게임컵 파티!

Hard #segtree #lazyprop #sqrt_decomposition

- 제출 60번, 정답 13명 (정답률 21.667%)
- 처음 푼 사람: **greedev**, 68분
- 출제자: cozyyg

- 어떤 플레이어가 들어온 순간 x인으로 게임하는 것을 선호하는 플레이어의 수가 x 이상인 x가 존재하게 되었다고 합시다.
- 만약 그 값이 $x+1$ 이상이라면, 이 플레이어가 들어오기 전에도 x 이상이었을 것입니다.
- 따라서 x인을 선호하는 플레이어의 수는 정확히 x고, 그 x명이 모두 매칭됩니다.
- 또한 매칭 이후에는 조건을 만족하는 x가 없는 상태로 돌아오게 됩니다.
- 다음의 두 가지를 구현해야 합니다.
  1. x인으로 게임하는 것을 선호하는 플레이어의 수가 x인 가장 큰 x 확인하기 (Check)
  2. 그 x에 대해 x인으로 게임하는 것을 선호하는 플레이어 모두 구하기 (Find)

Check : 느리게 갱신되는 세그먼트 트리

- **구간의 최솟값을 저장하는** 세그먼트 트리의 리프 노드를 $0, 1, 2, \cdots, N$ 으로 초기화힙니다.
- 범위가 $[a_i, b_i]$ 인 사람이 들어왔다면 구간 $[a_i, b_i]$ 에 1을 뺍니다.
- 세그먼트 트리에서의 이분 탐색을 이용하여 **값이 0인 가장 큰 인덱스**를 구할 수 있습니다.
  - 0번 노드는 언제나 값이 0이기 때문에 현재 루트 노드에 저장된 값은 0입니다.
  - 루트 노드에서 시작해서, 만약 오른쪽 자식 노드에 저장된 값이 0이면 그 노드로 이동합니다.
  - 그렇지 않다면 왼쪽 자식 노드로 이동합니다.
  - 이 과정을 반복하여 마지막에 도달한 리프 노드의 인덱스가 구하려는 값입니다.
- 쿼리당 시간복잡도는 $\mathcal{O}(\log N)$ 입니다.
- 세그먼트 트리 대신 제곱근 분할법을 사용하면 쿼리당 시간복잡도는 $\mathcal{O}(\sqrt{N})$ 입니다.

Find : 세그먼트 트리

- 세그먼트 트리의 각 노드가, $[a_i, b_i]$ 가 해당 노드 구간을 완전히 포함하는 플레이어들의 리스트를 관리하도록 합니다.
- $[a_i, b_i]$ 를 업데이트할 때는 세그먼트 트리의 구간 업데이트/쿼리와 비슷하게 합니다.
- 매칭된 인원이 x 명인 경우 x 번 인덱스의 리프 노드부터 부모 노드로 올라가면서 매칭된 적이 없는 플레이어를 가져오면 됩니다.
- 매칭된 플레이어는 Check에서 다시 구간 $[a_i, b_i]$ 에 1을 더해줍니다.
- **주의: 노드를 확인할 때마다 그 노드의 리스트를 비워줘야 합니다!**
- 시간복잡도와 공간복잡도 모두 $\mathcal{O}(N \log N)$ 입니다.

Find : 제곱근 분할법

- $i \leq C$ 일 때만 i 명으로 플레이하는 것을 선호하는 사람을 리스트로 관리합니다.
- 매칭된 인원이 C 이하인 경우 리스트의 플레이어들 중 아직 매칭되지 않은 플레이어를 가져옵니다.
- 매칭된 인원이 C 초과인 경우 지금까지 들어온 플레이어를 모두 확인하여 답을 구합니다.
- 첫 번째 케이스의 총 시간복잡도는 $\mathcal{O}(C \cdot N)$ 입니다.
- 두 번째 케이스의 총 시간복잡도는 $\mathcal{O}(N/C \cdot N)$ 입니다.
- $C = \sqrt{N}$ 으로 두면 전체 시간복잡도는 $\mathcal{O}(N\sqrt{N})$ 입니다.
- 공간복잡도는 $\mathcal{O}(C \cdot N)$ 입니다.
- 실제로는 C 를 $\sqrt{N}$ 보다 조금 작게 잡는 것이 좋습니다.

더블 아웃

Hard #dp #implementation

- 제출 18번, 정답 10명 (정답률 55.556%)
- 처음 푼 사람: **xiaowuc1**, 46분
- 출제자: cozyyg

- 남은 점수가 x, 남은 다트 수가 y일 때 조준해야 하는 영역과 성공 확률을 $dp[x][y]$로 정의합니다.
- 조준 가능한 각 영역마다, 실제로 맞히게 되는 영역의 점수 c와 확률 p에 대해 $p \cdot dp[x-c][y-1]$을 합하면 그 영역을 조준했을 때 성공 확률이 나옵니다.
- 이 중 최댓값을 구하고 사전순 조건에 따라 조준해야 하는 영역을 구하면 됩니다.
- dp 테이블을 계산할 때 영역별 확률과 Double Out 조건을 정확하게 구현해야 합니다.
- **실수 자료형을 사용하지 않고 정수 연산만으로 문제를 푸는 것이 좋습니다.**
  - 이론상으로는 같아야 하는 값이 실수 연산의 오차로 인해 달라질 수 있습니다.
  - 이로 인해 최댓값에 해당하는 영역이 누락될 수 있습니다.
  - 모든 확률이 백분율로 주어지기 때문에, 확률에 100^y를 곱한 값을 구하도록 하면 모든 값이 정수가 됩니다.

수 나누기 게임

Medium #sieve #number_theory

- 제출 589번, 정답 198명 (정답률 34.295%)
- 처음 푼 사람: **djs100201**, 9분 13초
- 출제자: havana723

- 어떤 수가 등장했는지 여부를 크기 $1\,000\,000$의 배열에 전처리할 수 있습니다.
- 이제 주어진 배열의 수 x에 대해, x의 배수 $y > x$를 모두 확인합니다. y가 등장한 적이 있는지는 전처리 배열을 사용해 $\mathcal{O}(1)$에 확인 가능합니다.
- 위의 경우에서 y가 등장한 적 있다면 x에 1을 더해 주고 y에서 1을 빼 주면 됩니다.

$\sum_{n=1}^{1\,000\,000} \frac{1}{n} \approx 14.4$으로 그렇게 크지 않아서 이 풀이는 시간 안에 동작합니다.