2023 ICPC Sinchon SUAPC Summer Solution

Official Solutions

신촌지역 대학생 프로그래밍 동아리 연합

2023년 8월 26일

240

문제		의도한 난이도	출제자
A	A→B	Hard	이도훈 dhlee1031
B	기초적인 문제	Hard	이명헌 nflight11
C	2048	Hard	최우현 starwh03
D	디지털 트윈	Medium	이협 dlguq0107
E	성게밭	Medium	이협 dlguq0107
F	작곡가 A의 시창 평가	Hard	길수민 2093ab
G	개발자 지망생 구름이의 취업 뽀개기	Beginner	길수민 2093ab
H	탭 UI	Medium	신정환 shjohw12
I	폭탄 피하기	Hard	이지훈 maker29
J	큰 수 만들기 게임	Medium	이지훈 maker29
K	케이크 두 개	Easy	이명헌 nflight11
L	나의 FIFA 팀 가치는?	Easy	이지훈 maker29
M	트리와 케이	Challenging	최우현 starwh03
N	북극여우는 괄호를 뒤집어	Challenging	이협 dlguq0107

240

A. A→B

`constructive`

출제진 의도 – **Hard**

- 제출 18번, 정답 0팀 (정답률 0.000%)
- 처음 푼 팀: —[**대학교], 0분
- 출제자: 이도훈[dhlee1031]

- 우선 집합 A에 들어가는 수에 1, 집합 B에 들어가는 수에 0을 부여해서 두 집합의 상태를 bool string으로 표현해봅시다.
- 예를 들어, 집합 A에 $1, 3, 4, 5, 6$이 있고, 집합 B에 2가 있다면 이를 101111로 표현할 수 있습니다.
- 그러면 이제 집합에서 연속한 수들을 옮기는 작업이 string에서는 $110 \rightarrow 001$ 또는 $011 \rightarrow 100$으로 바꾸는 연산과 동일합니다.

- 이러한 연산의 형태는 Peg Solitaire 퍼즐로 잘 알려져 있고, 일반적인 이 퍼즐은 2차원의 작은 십자가 보드에서 진행됩니다.
- 이 문제는 1차원 보드에서 진행되는 Peg Solitaire 퍼즐로 해석할 수 있고, 이 퍼즐에서 해결 가능한 (1을 하나만 남길 수 있는) 초기 형태는 대략 8가지의 Regular Expression으로 표현할 수 있습니다. (https://arxiv.org/pdf/math/0008172.pdf)
- 이는 역연산 ($100 \to 011$ 또는 $001 \to 110$)을 통해 유도가 가능하다는 정도로 간략히 말씀드리겠습니다.

- 해결 가능한 1차원 Peg Solitaire 퍼즐의 초기 형태 Regular Expression 중 연속한 0이 없는 형태는 다음과 같습니다.
- $11(01)^*(11)^*1011(10)^*11$
- $11(01)^*1101(11)^*(10)^*11$
- $11(01)^*(11)^*01$
- $10(11)^*(10)^*11$

- 문제를 다시 보면 1차원 Peg Solitaire 퍼즐에서 길이 N과 0의 개수 M이 주어졌을 때, 해결 가능한 초기 형태 중 0이 연속하지 않는 것을 출력하는 문제로 볼 수 있습니다.
- 다시 Regular Expression으로 돌아가서 우선 길이 N이 5 이상일 때, “양 끝이 1이고” 해결 가능한 string은 N이 짝수일 때만 가능함을 알 수 있습니다.
- 그렇기 때문에 N이 홀수라면 무조건 첫 자리 또는 마지막 자리를 0으로 둬야 합니다.
- 편의상 첫 자리를 0으로 두겠습니다.

- 그래서 Regular Expression 중 $10(11)^*(10)^*11$ 를 사용하여, N 이 홀수일 때는 $010(11)^*(10)^*11$, 짝수일 때는 $10(11)^*(10)^*11$ 가 정답이 됩니다.
- (Regular Expression에서 $()^*$ 는 괄호 안의 문자들을 0번 이상 반복하겠다는 의미입니다. 예시로, $1(10)^*$ 는 $1, 110, 11010, \ldots$등을 포함하는 string 집합의 표현입니다.)
- 단, N 이 홀수일 때 M 이 1이라면 위 Regular Expression에선 0 이 최소 두 개 필요했기 때문에 정답이 될 수 없습니다.
- 그래서 이 경우는 -1 을 출력하고, 나머지는 각 Regular Expression에서 0 의 index를 전부 출력해주면 AC를 받을 수 있습니다.

240

B. 기초적인 문제

`linear_algebra, modular_multiplicative_inverse`

출제진 의도 – **Hard**

- 제출 16번, 정답 1팀 (정답률 6.250%)
- 처음 푼 팀: **SCC_Sinchon Coding Champions** [연세대학교], 82분
- 출제자: 이명헌 [nflight11]

- 행렬식을 계산하는 방법 중 가장 구현이 간단하고 잘 알려진 방법은 Gaussian Elimination 입니다.
- 다만 이렇게 $N \times N$ 행렬의 행렬식을 계산하는 Gaussian Elimination의 시간복잡도는 $\mathcal{O}(N^3)$ 입니다. 총 Q 개의 행렬식을 계산하여야 하므로, 총 시간복잡도는 $\mathcal{O}(QN^3)$ 이 됩니다.
- $QN^3 = 2.5 \times 10^{10}$ 이 되므로, 이 방법으로는 시간초과를 피할 수 없게 됩니다.
- 훨씬 더 빠른 방법이 필요합니다.
- 두 가지 방법이 있습니다.

방법 1

✓ 문제에서 주어진 행렬식을 조금 일반화하여,

$$M_k[a_1, a_2, \cdots, a_N] = \det \begin{bmatrix} \binom{a_1}{k} & \binom{a_1}{k+1} & \cdots & \binom{a_1}{k+N-1} \\ \binom{a_2}{k} & \binom{a_2}{k+1} & \cdots & \binom{a_2}{k+N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \binom{a_N}{k} & \binom{a_N}{k+1} & \cdots & \binom{a_N}{k+N-1} \end{bmatrix}$$

라 합시다.

- 다음이 성립합니다.

$$M_0[a_1+1, a_2+1, \cdots, a_N+1] = M_0[a_1, a_2, \cdots, a_N]$$

- 증명은 다음 식에서,

$$\begin{bmatrix} \binom{a_1+1}{0} & \binom{a_1+1}{1} & \cdots & \binom{a_1+1}{N-1} \\ \binom{a_2+1}{0} & \binom{a_2+1}{1} & \cdots & \binom{a_2+1}{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \binom{a_N+1}{0} & \binom{a_N+1}{1} & \cdots & \binom{a_N+1}{N-1} \end{bmatrix} = \begin{bmatrix} \binom{a_1}{0} & \binom{a_1}{0}+\binom{a_1}{1} & \cdots & \binom{a_1}{N-2}+\binom{a_1}{N-1} \\ \binom{a_2}{0} & \binom{a_2}{0}+\binom{a_2}{1} & \cdots & \binom{a_2}{N-2}+\binom{a_2}{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \binom{a_N}{0} & \binom{a_N}{0}+\binom{a_N}{1} & \cdots & \binom{a_N}{N-2}+\binom{a_N}{N-1} \end{bmatrix}$$

왼쪽에서부터 인접한 두 column을 빼는 것을 반복하면 됩니다.

✓ 그렇다면, 수열 중 가장 작은 값이 a_1 이 되도록 순서를 바꾸었을 때, 다음이 성립합니다.

$$\begin{aligned} M_0[a_1, a_2, \cdots, a_N] &= M_0[0, a_2 - a_1, a_3 - a_1, \cdots, a_N - a_1] \\ &= M_{\color{red}1}[a_2 - a_1, a_3 - a_1, \cdots, a_N - a_1] \qquad \left(\because \binom{0}{0} = 1\right) \end{aligned}$$

✓ $k = 1$ 인 경우는 어떻게 계산할 수 있을까요?

- 다음이 성립합니다.

$$M_1[a_1+1, a_2+1, \cdots, a_N+1] = \frac{1}{N!}\prod_{i=1}^{N}(a_i+1)\cdot M_0[a_1, a_2, \cdots, a_N]$$

- 증명은 다음 식에서,

$$\begin{bmatrix} \binom{a_1+1}{1} & \binom{a_1+1}{2} & \cdots & \binom{a_1+1}{N} \\ \binom{a_2+1}{1} & \binom{a_2+1}{2} & \cdots & \binom{a_2+1}{N} \\ \vdots & \vdots & \ddots & \vdots \\ \binom{a_N+1}{1} & \binom{a_N+1}{2} & \cdots & \binom{a_N+1}{N} \end{bmatrix} = \begin{bmatrix} \frac{a_1+1}{1}\binom{a_1}{0} & \frac{a_1+1}{2}\binom{a_1}{1} & \cdots & \frac{a_1+1}{N}\binom{a_1}{N-1} \\ \frac{a_2+1}{1}\binom{a_2}{0} & \frac{a_2+1}{2}\binom{a_1}{1} & \cdots & \frac{a_2+1}{N}\binom{a_2}{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{a_N+1}{1}\binom{a_N}{0} & \frac{a_N+1}{2}\binom{a_1}{1} & \cdots & \frac{a_N+1}{N}\binom{a_N}{N-1} \end{bmatrix}$$

i 번째 row의 공통 인수 (a_i+1), j 번째 column의 공통 인수 j 를 빼 주면 됩니다.

- 따라서, $a_1, a_2, \cdots, a_N$ 중 같은 것이 있다면 답은 0 이고, 같은 것이 없다면,
  - 수열에서 가장 작은 수 m 을 찾고, 수열의 모든 수에서 m 을 뺍니다. (첫 번째 식 사용)
  - 값이 0 인 인덱스와 a_1 을 맞바꿉니다. 0 인 인덱스가 **첫 번째가 아닌 경우에만** 행렬식의 부호를 뒤집어 줘야 합니다.
  - 그 값을 중심으로 expand 한 뒤, 두 번째 식을 적용하여 대응하는 인수를 곱합니다.
- 위 과정을 많이 반복하여 N 이 1 이 되면 이때 행렬식의 값은 그냥 $\begin{pmatrix} a_1 \\ 0 \end{pmatrix} = 1$ 이 됩니다.
- 한 번의 과정에 $\mathcal{O}(N)$ 번의 계산이 들어가고, 이 과정을 N 번 반복하므로 시간 복잡도는 쿼리당 $\mathcal{O}(N^2)$ 이 됩니다.

방법 2

- 이항계수를 자세히 관찰하여 봅시다.
- $\binom{N}{K} = \dfrac{N(N-1)(N-2)\cdots(N-K+1)}{K!}$ 이므로, 이는 변수를 N으로 가지고 최고차항의 계수가 $\dfrac{1}{K!}$ 인 K차 다항식이 됩니다.
- 이제 $\binom{N}{K} = \sum_{i=0}^{K} \lambda_{i,K} N^i$ 으로 표기합시다. K차 다항식 $\binom{N}{K}$ 의 i차항 계수가 $\lambda_{i,K}$ 라는 뜻입니다.
- 그러면 주어진 행렬 $A(a_1, a_2, \cdots, a_N)$ 의 (i, j)-component $\binom{a_i}{j-1}$ 는 $\sum_{k=0}^{j-1} a_i^k \lambda_{k,j-1}$ 이 됩니다.

- 그러므로 $A(a_1, \cdots, a_N)$ 는 (i,j)-component가 a_i^{j-1} 인 행렬 V 와 (i,j)-component가 $\lambda_{i-1,j-1}$ 인 행렬 B 의 곱 VB 로 표현이 가능합니다.
- V 의 행렬식은 간단한 귀납법과 Gaussian Elimination을 이용하여 $\prod_{1 \le i < j \le N} (a_j - a_i)$ 임을 증명할 수 있습니다(Vandermonde 행렬식).
- 따라서 $\det V$ 는 $\mathcal{O}(N^2)$ 으로 구할 수 있습니다. $\det B$ 는 어떻게 계산할 수 있을까요?
- 정의에 의하여, $i > j$ 일 때, $\lambda_{i-1,j-1}$ 의 값은 0 입니다. 따라서 B 는 upper triangular matrix가 됩니다!

240

- Triangular matrix의 행렬식을 계산하는 방법은 아주 간단합니다. 주대각원소를 모조리 곱해버리면 됩니다.
- 정의에 따라서 B의 (i,i)-component인 $\lambda_{i-1,i-1}$의 값은 $\dfrac{1}{(i-1)!}$입니다.
- 그러므로 $\det B = \prod_{1}^{N} \dfrac{1}{(i-1)!} = \prod_{0}^{N-1} \dfrac{1}{i!}$입니다.

- 2 부터 500 까지의 N 에 대해서, 대응하는 크기를 가지는 B 의 행렬식은 $\mathcal{O}(N^2 \log \text{mod})$ 으로 계산할 수 있습니다.
- 따라서 전체 시간복잡도는 $\mathcal{O}(N^2 \log \text{mod} + QN^2)$ 으로, 시간초과를 피할 수 있습니다.
- 고급 알고리즘을 사용한다면 수행시간을 $\mathcal{O}(QN \log^3 N)$ 까지 줄일 수 있다고 합니다. 어떻게 해야 가능할지는 나중의 즐거움으로 남겨놓겠습니다.

C. 2048

dp
출제진 의도 – **Hard**

- 제출 3번, 정답 0팀 (정답률 0.000%)
- 처음 푼 팀: — OO대학교, 0분
- 출제자: 최우현 starwh03

- 단순하게 생각해 봐도 $12^{H\times W}$ 이상 존재하기 때문에 하나씩 계산하는 것은 불가능합니다.
- 세로 길이가 최대 6 이기 때문에 이를 활용하면 시간 안에 문제를 해결할 수 있습니다.
- 다음 세로줄의 상태에 영향을 주는 것은 직전 세로줄 밖에 없습니다.
- 이를 활용하면 W 에 대해 선형의 시간에 문제를 해결할 수 있습니다.

- 하나의 세로줄만을 봤을 때, 가능한 경우의 수는 $16 \times 15^{H-1}$ 입니다.
- 여전히 너무 많습니다.
- $1, 2, 3, 4$ 인 세로줄과 $5, 6, 7, 8$ 두 가지 세로줄 다음으로 올 수 있는 경우의 수는 동일합니다.
- 이와 같이 수를 결정하는 것이 아니라, 수들의 패턴과 해당 패턴이 몇 번 등장하는지를 이용하면 경우의 수를 크게 줄일 수 있습니다.
- 길이가 6 인 패턴은 52 가지 존재합니다. 패턴의 종류의 숫자를 P 라고 하겠습니다.
- 각 패턴 간 조합하는 경우를 계산하는 데에 H 만큼의 시간이 걸리므로, $\mathcal{O}(P^2H \times W + QP)$ 의 시간에 문제를 해결할 수 있습니다.

C. 2048

- 여전히 너무 많습니다. 전처리를 통해 시간복잡도를 크게 줄일 수 있습니다.
- 패턴 간 경우의 수를 전처리해 두면 $P \times P$ 행렬의 $W - 1$ 제곱을 구하는 문제가 됩니다.
- 행렬의 2^i 제곱을 미리 구해 놓고, $P \times P$ 행렬 $\log W$ 개의 곱셈을 진행하는 것으로 답을 구할 수 있습니다. 이 경우 $\mathcal{O}(P^3 \times Q \times \log W)$ 의 시간에 문제를 해결할 수 있습니다.
- 행렬 거듭제곱 이후에 벡터와 곱셈을 진행하므로, 계산 순서를 바꾸는 것으로 $\mathcal{O}(P^2 \times \log W)$ 시간에 각 쿼리를 처리할 수 있습니다.
- 최종 시간복잡도는 $\mathcal{O}(P^2H + P^3 \times \log W + P^2 \times Q \times \log W)$ 로, 시간 안에 문제를 해결할 수 있습니다.

D. 디지털 트윈

dp

출제진 의도 – **Medium**

- 제출 21번, 정답 7팀 (정답률 33.333%)
- 처음 푼 팀: **병공병**[홍익대학교], 47분
- 출제자: 이협[dlguq0107]

- 우선 컨테이너 벨트의 특징을 한번 살펴봅시다.
- 컨테이너 벨트는 위에서 아래로밖에 가지 못하기 때문에 한 가로줄에 있는 생산기계는 모두 지나고 다음 가로줄로 내려가야 합니다.
- 그렇다면 각 가로줄에서 지나가야 하는 가장 왼쪽 점과 오른쪽 점을 저장해 둘 수 있고 이를 이용하여 dp 식을 세울 수 있습니다.
- 공장이 시작하는 가로줄을 1 번 가로줄, 끝나는 가로 줄을 n 번 가로줄로 두고 i 번째 가로줄의 가장 왼쪽점과 오른쪽점을 각각 l_i, r_i 라 둡시다. 또한 빈 줄의 여부를 c_i 라고 하고 빈 줄이면 1 아니면 0 이라 둡시다.

- dp 식을 정의 해봅시다.
- i 번째 가로줄에서 l_i 에 도착하고 최소인 경우를 $dp[0][i]$, r_i 에 도착하고 최소인 경우를 $dp[1][i]$ 라고 둡시다.
- dp 식은 l_i, r_i 에 따라 바뀌게 되는데 l_i, r_i 값이 만약 정의가 되어있지 않다면 dp 업데이트 이후 l_{i-1}, r_{i-1} 의 값을 l_i, r_i 에 각각 저장합니다.

✓ 그렇다면 dp 식은 다음과 같이 정의됩니다.

$$\mathrm{dp}[0][i] = \begin{cases} \mathrm{dp}[0][i-1]+1 & \text{if } c_i = 1 \\ r_i - l_i + 1 + \min(\mathrm{dp}[1][i-1] + |r_i - r_{i-1}|, \\ \qquad \mathrm{dp}[0][i-1] + |r_i - l_{i-1}|) & \text{if } r_i \le l_{i-1} \text{ or } c_i = 0, c_{i-1} = 1 \\ r_i - l_i + 1 + \mathrm{dp}[1][i-1] + |r_i - r_{i-1}| & \text{otherwise} \end{cases}$$

$$\mathrm{dp}[1][i] = \begin{cases} \mathrm{dp}[1][i-1]+1 & \text{if } c_i = 1 \\ r_i - l_i + 1 + \min(\mathrm{dp}[0][i-1] + |l_i - l_{i-1}|, \\ \qquad \mathrm{dp}[1][i-1] + |l_i - r_{i-1}|) & \text{if } r_{i-1} \le l_i \text{ or } c_i = 0, c_{i-1} = 1 \\ r_i - l_i + 1 + \mathrm{dp}[0][i-1] + |l_i - l_{i-1}| & \text{otherwise} \end{cases}$$

- $\mathrm{dp}[1][n]$ 의 값이 최소 컨테이너 벨트의 길이가 됩니다.
- 만약 이 값이 정의되지 않는다면 도달할 수 없는 경우이므로 -1 을 출력하면 됩니다.

240

E. 성게밭

`implement, dfs, bipartite_matching`

출제진 의도 – **Medium**

- 제출 4번, 정답 3팀 (정답률 75.000%)
- 처음 푼 팀: **월파출신 코치가 가르치고 있어요**[서강대학교], 48분
- 출제자: 이협[dlguq0107]

- 우선 성게밭의 성질을 한번 알아봅시다.
- 주어진 성게밭의 그래프를 $G = (V, E)$ 라고 한다면 $v \in V$ 중 $\deg(v) \geq 3$ 이라면 항상 성게의 중심이 된다는 사실을 알 수 있습니다.
- 따라서 dfs를 통해 성게를 하나의 컴포넌트로 갖고 맞닿는 성게끼리 변을 구성하여 새로운 그래프를 만들 수 있습니다.
- 그렇게 형성된 그래프에서 성게는 암수끼리만 맞닿아 있으므로 항상 이분그래프로 나타나짐을 알 수 있습니다.

- 그렇다면 문제는 이제 주어진 그래프에서 인접하지 않는 최대 정점 집합을 찾는 문제로 바뀌게 됩니다.
- 최대 독립 집합은 전체 정점에서 최소 정점 커버의 크기만큼을 빼준 것과 같고 Kőnig’s Theorem에 따르면 이분 그래프에서 최소 정점 커버는 최대 매칭의 개수와 같기 때문에 주어진 그래프에서 최대 매칭을 구한다면 문제를 해결할 수 있습니다.
- 따라서 dfs를 통해 최대 매칭을 구하거나 flow로 디자인하여 매칭을 구한다면 문제를 해결할 수 있습니다.

F. 작곡가 A의 시창 평가

`kmp, sprague_grundy`

출제진 의도 – **Hard**

- 제출 22번, 정답 8팀 (정답률 36.364%)
- 처음 푼 팀: **1등하러왔습니다**[연세대학교], 15분
- 출제자: 길수민[2093ab]

- 먼저 악보에서 빨간색으로 색칠된 부분을 찾아봅시다.
- 악보에 포함된 멜로디의 접미사들을 모두 찾아주어 빨간색으로 색칠해주면 됩니다.
- 악보와 멜로디를 전부 뒤집어 KMP를 수행하면 악보에서 일치하는 멜로디의 부분을 $\mathcal{O}(N+M)$에 구할 수 있습니다.
- 만약 해당 인덱스의 값이 양수라면 그 부분은 색칠되어 있다는 의미입니다.
- 빨간색으로 색칠된 연속된 부분의 길이를 모두 저장합니다.

- 길이가 n 인 하나의 연속된 빨간색 구간에 대하여 생각해봅시다.
- 초록색으로 색칠할 연속된 구간을 선택하면 a 와 b 길이의 빨간색 구간으로 나뉘게 됩니다.
- 스프라그-그런디 정리에 의하여 그런디 수 $g(n) = \mathrm{mex}(\{g(a) \oplus g(b)\})$ 임을 알 수 있습니다.
- 이 때 a 와 b 는 모두 0 이상 $n-1$ 이하이고 $a+b \le n-1$ 를 만족합니다.
- a 가 0 이라면 $0 \le b \le n-1$ 이므로 0 이상 $n-1$ 이하인 구간은 만들 수 있습니다.
- $g(0) = 0$ 이고 $g(a) \oplus g(b) \le n-1$ 이기 때문에 $g(n) = n$ 입니다.
- 따라서 NIM 게임과 같은 원리로 연속된 빨간 구간의 길이들을 모두 xor 해 준 결과로 판단할 수 있습니다.
- 만약 0 인 경우 두 번째 순서, 아닌 경우 첫 번째 순서를 선택해주면 됩니다.

G. 개발자 지망생 구름이의 취업 뽀개기

`sorting, greedy`

출제진 의도 – **Beginner**

- 제출 46번, 정답 29팀 (정답률 65.217%)
- 처음 푼 팀: **Redshift**[서강대학교], 4분
- 출제자: 길수민[2093ab]

- 우선 하나의 난이도에 대해, 풀 문제를 선택했을 때 어떤 순서로 배치해야 최소의 시간이 걸리는지부터 알아보겠습니다.
- 선택된 문제 중 예상 시간이 가장 큰 값을 a, 가장 작은 값을 b라 정의합니다.
- a의 문제와 b의 문제 사이에는 항상 $a - b$ 이상의 휴식 시간이 필요하게 됩니다.
- 이는 오름차순 정렬을 한 경우 휴식 시간은 $a - b$로 최솟값이 되어 최적입니다.
- 그러므로 선택한 문제에 대해 걸리는 시간은 예상 시간의 합과 $a - b$를 더한 값입니다.

- 한 난이도에서 걸리는 시간은 예상 시간의 합에서 a를 더하고 b를 뺀 값입니다.
- 그러므로 예상 시간이 가장 작은 값부터 선택하는 경우 최솟값을 구할 수 있습니다.
- 각 난이도에 대하여 해당 값을 구하여 더하고, 난이도 증가시킬 때의 휴식 시간 $60 \times 4 = 240$분을 더하여 구할 수 있습니다.

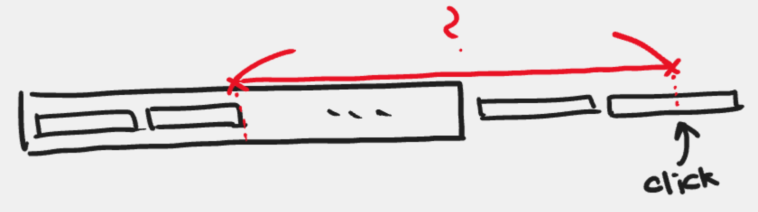
240

H. 탭 UI

`prefix sum`, `implementation`
출제진 의도 – **Easy**

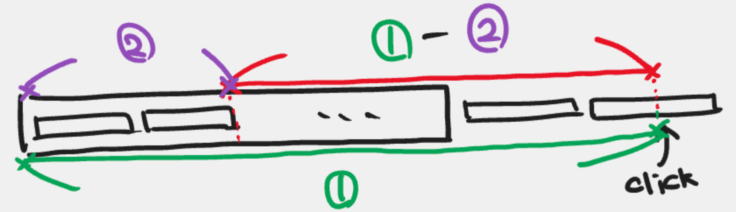
- 제출 56번, 정답 18팀 (정답률 32.143%)
- 처음 푼 팀: **Redshift**[서강대학교], 12분
- 출제자: 신정환[shjohw12]

- 위치 0을 기준으로, 탭을 클릭했을 때 양의 방향으로 얼마만큼 이동하는지 생각해봅시다.



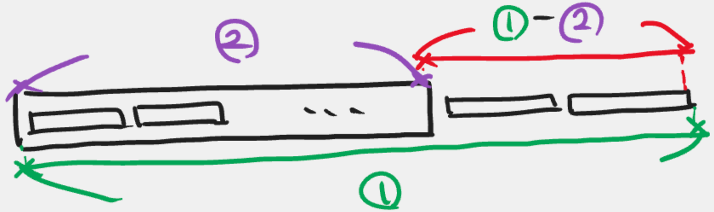
- 화면의 중앙과 클릭한 탭의 중앙 사이의 거리만큼 이동하면 됩니다.

- 거리를 어떻게 구할 수 있을까요?



- 그림에서 알 수 있듯이, 클릭한 탭의 왼쪽에 있는 탭들의 길이와 클릭한 탭의 길이의 절반의 합에서 화면의 길이의 절반을 뺀 값이 됩니다.

- 1 번 탭부터 i 번 탭까지의 길이의 합을 sum_i 로 정의합니다.($sum_0 = 0$)
- i 번 탭을 클릭했을 때 구하고자 하는 값은 $sum_{i-1} + \frac{l_i}{2} - \frac{L}{2}$ 입니다.
- 이제 이동 제한을 처리해주어야 합니다.
- 1번 제한은 위치의 최솟값을 나타내는 제한입니다. 위치 0 일 때 음의 방향으로 이동할 수 없으니 위치의 최솟값이 0 임을 의미합니다.
- 2번 제한은 위치의 최댓값을 나타내는 제한입니다. 가장 오른쪽에 있는 탭의 오른쪽 끝과 화면의 오른쪽 끝이 일치하는 순간의 위치를 구해봅시다.



- 그림에서 알 수 있듯이, 전체 탭의 길이의 합에서 화면의 길이를 뺀 값입니다.
- 위치의 최댓값을 limit이라 할 때, $\mathrm{limit} = \mathrm{sum}_N - L$ 입니다.
- 3번 제한은 위치의 최댓값이 음수가 될 수 없음을 의미합니다. 즉, $\mathrm{limit} = \max(0, sum_N - L)$ 입니다.
- 따라서 i 번 탭을 클릭했을 때 위치는 $\max(0, \min(\mathrm{sum}_{i-1} + \frac{l_i}{2} - \frac{L}{2}, \mathrm{limit}))$ 입니다.

- Q 번의 쿼리에 대해 답을 구해야하므로, prefix sum을 전처리하면 $\mathcal{O}(N+Q)$ 에 문제를 해결할 수 있습니다.
- 모든 길이를 2 배 해서 처리하고 출력할 때만 2 로 나누면 실수 자료형을 쓰지 않아도 문제를 해결할 수 있습니다.

I. 폭탄 피하기

`inclusion-exclusion_principle, bitmasking`

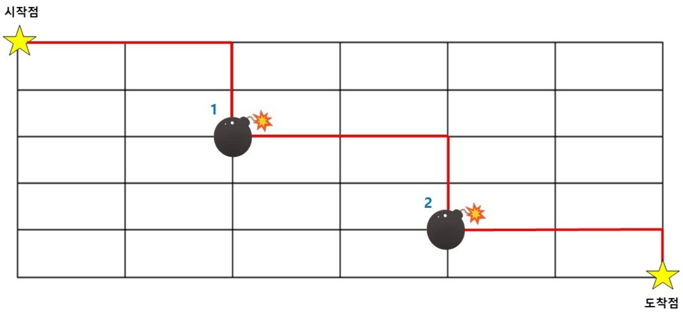
출제진 의도 – **Hard**

- 제출 62번, 정답 10팀 (정답률 16.129%)
- 처음 푼 팀: **1등하러왔습니다**[연세대학교], 24분
- 출제자: 이지훈[maker29]

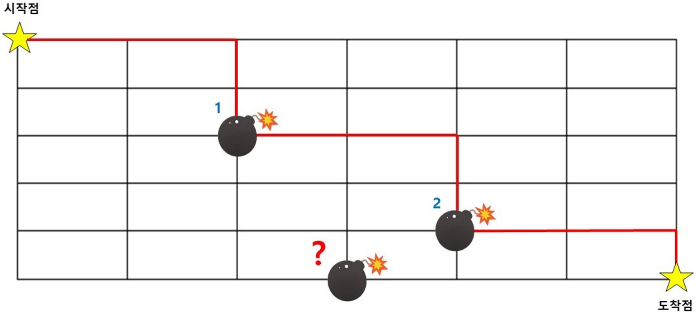
- $A \times B$ 크기의 격자판에서 이동할 수 있는 경우의 수는 $\dfrac{(A+B)!}{A!\,B!}$ 입니다. $(A, B$ 는 양의 정수$)$
- 1 에서 2×10^6 의 Factorial과 inverse를 배열에 저장해 둡니다.
- 저장하는 시간 $\mathcal{O}(2 \times N)$ 이후에는 이동하는 경우의 수를 $\mathcal{O}(1)$ 에 계산 가능합니다.
- 출발점부터 도착점까지 폭탄이 없다고 가정하고 경우의 수를 구합니다.
- 이후 폭탄을 선택하여 선택한 폭탄을 모두 지나는 경로를 포함-배제의 원리로 더하거나 뺍니다.
- 폭탄을 선택하는 가짓수는 2^K 개로, 비트마스킹으로 구현할 수 있습니다.

- 폭탄들을 각각 $B_1, B_2 \cdots B_K$ 라고 합시다.
- 모든 i, j에 대해서 $B_i.X \le B_j.X, B_i.Y \le B_j.Y$가 되게 정렬합니다. $(1 \le i, j \le K,\ i < j)$
- 주의해야 할 점은 정렬이 되었다고 모든 폭탄을 지날 수 있는 것은 아닙니다.
- 오른쪽과 아래로만 이동할 수 있으므로, 2개의 폭탄을 일직선으로 연결한 직선이 좌측 하단이나 우측 상단으로 가면 해당 2개의 폭탄을 지나는 경로는 존재하지 않습니다.

- 시작점부터 시작하여 비트마스킹으로 선택한 폭탄을 넣습니다.
- 마지막으로 넣은 폭탄과 넣으려는 폭탄을 모두 지나는 경로가 존재하면 계속 진행합니다.
- 선택한 모든 폭탄을 지나갈 수 있으면 폭탄을 지나가는 순서는 유일합니다. (정렬 순서 그대로)



✓ 선택한 모든 폭탄을 지나는 경로가 존재하지 않으면 넣는 작업을 그만둡니다(경우의 수 0).



- 폭탄을 다 넣으면 도착점을 포함하여 포함-배제의 원리로 경우의 수를 계산합니다.
- 폭탄(or 시작점) → 폭탄(or 도착점)의 경우의 수를 모두 곱합니다.
- 폭탄의 갯수가 짝수인 경우 더하고, 홀수인 경우 뺍니다.
- 총 시간 복잡도는 $\mathcal{O}(N + M + K \times 2^K)$가 됩니다.
- 비트마스킹 대신 DFS로 탐색하면 시간 복잡도를 $\mathcal{O}(N + M + 2^K)$로 줄일 수 있습니다.

240

J. 큰 수 만들기 게임

`prime_factorization, greedy, ad-hoc`

출제진 의도 – **Medium**

- 제출 36번, 정답 11팀 (정답률 30.556%)
- 처음 푼 팀: **기령손의 그녀들**[서강대학교], 18분
- 출제자: 이지훈[maker29]

- 동작 1을 사용하여 $\frac{D}{K}, K$ 순으로 이어 붙여봅시다.
- 이 수는 $\frac{D}{K} \times 10^{\lfloor \log_{10} K \rfloor + 1} + K$ 가 됩니다.
- $\frac{D}{K} \times K = D < \frac{D}{K} \times 10^{\lfloor \log_{10} K \rfloor + 1}$ 이므로, 동작 1을 할수록 큰 수를 만들 수 있습니다.
- 입력된 수를 소인수분해하면 됩니다.
- 소인수분해는 일반적인 방법을 사용하면 $\mathcal{O}(\sqrt{N})$ 에 가능합니다.

- 동작 2는 동작 1과 반대되는 동작입니다.
- 동작 1을 사용하면 수가 커지므로, 동작 2를 사용하면 수가 작아짐을 알 수 있습니다.
- 큰 수를 만들기 위해 동작 2를 사용할 일은 없습니다.

- 가장 큰 수를 만들 수 있는 M 을 그리디하게 구할 수 있습니다.
- M 의 인수가 많을수록 얻을 수 있는 수는 커집니다.
- 따라서 M 은 2 와 3 의 곱들로 표현되어야 합니다.
- N 부터 시작하여 4 이상인 경우 2 로 나누는 작업을 반복합니다.
- 남은 숫자가 4 미만이 되면 남은 수와 나눴던 수들로 수를 만듭니다.

- N 의 인수를 $a_1, a_2 \cdots a_s$ 라고 합시다.
- 모든 i, j 에 대해서 a_i 와 a_j 를 잇는 방법은 2 가지 뿐입니다. $(1 \leq i, j \leq s,\ i < j)$
- 이어 붙인 수가 $a_i a_j > a_j a_i$ 가 되도록 정렬하면 됩니다.
- 이 정렬의 시간복잡도는 $\mathcal{O}(\log_2 N \times \log(\log_2 N))$ 입니다.
- M 도 동일한 방법으로 가장 큰 수를 만들 수 있습니다.

- M의 경우 $N = 10^{12}$이면 만들 수 있는 수의 최댓값이 약 40자리가 됩니다.
- 수들이 `long long` 범위를 넘어갈 수 있으므로 문자열로 바꾸어 계산하는 등의 방법이 필요합니다.

K. 케이크 두 개

`geometry, number_theory`

출제진 의도 – **Easy**

- 제출 61번, 정답 23팀 (정답률 37.705%)
- 처음 푼 팀: **월파출신 코치가 가르치고 있어요**[서강대학교], 22분
- 출제자: 이명헌[nflight11]

- 간단한 기하학을 이용하여 직사각형의 중심(두 대각선의 교점)을 지나는 직선은 반드시 직사각형을 이등분함을 증명할 수 있습니다.
- 따라서 두 직사각형의 중심을 지나는 직선이 문제의 답이 됩니다.
- 직사각형의 꼭짓점이 무작위로 주어졌으므로, 대각선을 이용하여 중심을 구하는 것보다 네 점의 좌표를 더하고 4로 나누어 중심의 좌표를 구하는 것이 효율적입니다.

240

L. 나의 FIFA 팀 가치는?

`priority_queue`

출제진 의도 – **Easy**

- 제출 67번, 정답 23팀 (정답률 34.328%)
- 처음 푼 팀: **SCC_Sinchon Coding Champions** 연세대학교, 4분
- 출제자: 이지훈 maker29

- 선발 선수는 매년 가치가 1 이 줄지만, 후보 선수의 가치는 차이가 없습니다.
- 선발 선수를 우선순위 큐의 top 으로 생각할 수 있습니다.
- K 번 동안 각 포지션마다 선발 선수의 능력치를 바꾼 뒤 다시 넣어주면 됩니다.
- 선발 선수의 능력치가 음수가 되지 않게 $\max(0, W_{top} - 1)$ 를 해야 하는 것을 유의합시다.

240

- 우선순위 큐의 삽입 및 삭제의 시간복잡도는 $\mathcal{O}(\log N)$ 입니다.
- 삽입 및 삭제를 K 년 동안 11 포지션을 계산해야 합니다.
- 총 시간 복잡도는 $\mathcal{O}(N + 2 \times 11 \times KlogN)$ 입니다.

M. 트리와 케이

`centroid decomposition, binary search`

출제진 의도 – **Challenging**

- 제출 0번, 정답 0팀 (정답률 0.000%)
- 처음 푼 팀: — OO대학교, 0분
- 출제자: 최우현 `starwh03`

- 나이브하게 생각하면 $\mathcal{O}(Q \times N)$ 의 시간에 문제를 해결할 수 있습니다.
- 쿼리 하나를 쪼개서 구하는 것으로 더 빠른 시간 안에 해결해야 합니다.

- 각 정점에 해당 정점의 서브트리에 속하는 정점들의 거리와 번호를 모두 정렬한 채로 저장해두고, 쿼리가 들어오면 이분탐색 하는 것으로 한 정점의 서브트리에 속하는 거리가 정확히 K인 모든 정점들을 $\log$ 시간에 처리할 수 있습니다.
- 해당 정점의 서브트리에 속하는 정점 중, 쿼리로 들어온 정점과 같은 서브트리에 속하는 정점의 경우 따로 계산해줘야 합니다.
- 이런 이유 때문에 쿼리로 들어온 정점과 루트를 잇는 경로상에 존재하는 모든 정점에 대해 탐색을 진행해야 합니다.
- 나이브에 비해 많이 줄었지만, 최악의 경우 $\mathcal{O}(Q \times N \times \log N)$의 시간복잡도를 가집니다.
- Centroid Deconposition을 이용하는 것으로 트리를 균형 잡히게 만들어 시간복잡도를 줄일 수 있습니다.

- 트리에서 Centroid를 잡고, 해당 정점을 루트로 설정합니다.
- 새롭게 생겨난 서브트리들에 대해서도, Centroid를 잡아 부모 정점과 연결하고, 해당 정점을 기준으로 다시 서브트리를 분할합니다. 이를 계속 반복합니다.
- Centroid의 서브트리의 크기가 절반 이하로 줄어든다는 특징 덕분에 새롭게 만들어진 트리의 깊이는 $\log N$을 넘지 않습니다.
- 이런 방식으로 최적화를 진행하면 $\mathcal{O}(Q \times \log^2 N)$의 시간에 문제를 해결할 수 있습니다.

240

N. 북극여우는 괄호를 뒤집어

`sqrt_decomposition`

출제진 의도 – **Challenging**

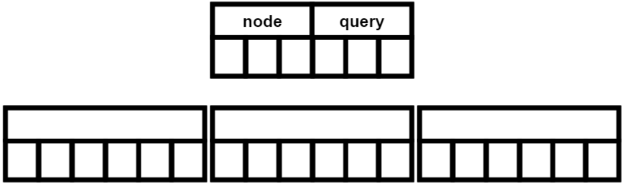
- 제출 2번, 정답 0팀 (정답률 0.000%)
- 처음 푼 팀: — [OO대학교], 0분
- 출제자: 이협[dlguq0107]

- 우선 나이브하게 계산을 하게 된다면 시간복잡도가 $\mathcal{O}(QN)$ 이 되어 제한시간 내에 해결할 수 없게 됩니다. 어떻게 하면 시간복잡도를 줄일 수 있을까요.
- 우선 괄호 문자열에서 효과적으로 올바른 괄호쌍의 개수를 구하는 방법을 생각해봅시다.
- 어떤 괄호 문자열에서 올바른 괄호 쌍을 제외하고 생각한다면 왼쪽에는 ’)’, 오른쪽에는 ’(’ 이 붙어있는 형태로 나타나집니다.
- 여기서 이 개수들이 각각 r, l 이고 원래 문자열의 길이가 n 이라면 올바른 괄호쌍의 개수는 $\frac{n-l-r}{2}$ 로 계산할 수 있습니다.
- 이런 형태로 한 문자열을 나타내는 노드를 (n, l, r) 로 정의한다면 올바른 쌍의 개수를 $\mathcal{O}(1)$ 에 구할 수 있습니다.

- 두 문자열을 합치는 과정도 생각해볼까요. 예를 들어 ")))(((", "))((" 라는 문자열이 있고 이를 나타내는 $(5, 3, 2), (4, 2, 2)$ 라는 노드가 있다고 생각해봅시다.
- 두 문자열을 합쳤을 때 올바른 괄호쌍이 생기는 경우는 왼쪽 문자열의 '(' 와 오른쪽 문자열의 ')' 가 만나는 경우 말고는 존재하지 않습니다. 이를 제외하고 생각한다면 합친 문자열 ")))((())((" 는 $(9, 3, 2)$ 의 노드로 표현이 가능합니다.
- 같은 방식으로 $(n_1, l_1, r_1), (n_2, l_2, r_2)$ 를 합친 노드는 $(n_1 + n_2, l_2 + \max(0, l_1 - r_2), r_1 + \max(0, r_2 - l_1))$ 로 나타낼 수 있고 이 또한 $O(1)$ 에 가능합니다.

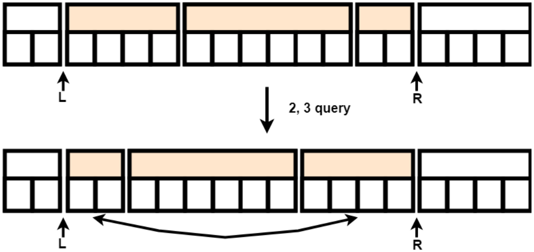
- 다음으로는 쿼리를 적용시킨 문자열을 한번 생각해봅시다. 기존 문자열에서 쿼리를 적용시키면 올바른 괄호쌍이었던 것이 올바른 괄호쌍이 아니게 될 수 있습니다.
- 이는 노드에 정보를 추가함으로써 해결이 가능합니다.
- 이번에는 역으로 생각하여 ")("가 올바른 괄호쌍이라고 생각해봅시다. 그럼 올바른 괄호쌍을 제외하면 왼쪽에는 '(' 오른쪽에는 ')'이 붙어있는 형태로 나타나게 됩니다.
- 각각의 개수를 rl, rr 이라고 둔다면 이 문자열을 반전했을 때 올바른 괄호쌍의 개수는 $\frac{n-rl-rr}{2}$ 로 구할 수 있습니다.

- 노드가 (n, l, r, rl, rr)로 정의되어있을 때 아래와 같이 $O(1)$에 쿼리를 적용할 수 있습니다.
  - 1번 쿼리 : (n, rr, rl, r, l)
  - 2번 쿼리 : (n, rl, rr, l, r)
  - 3번 쿼리 : (n, r, l, rr, rl)
- 노드가 잘 정의되기 때문에 겉보기에 세그먼트 트리로 관리할 수 있어 보이지만 2, 3번 쿼리를 적용할 때 자식 노드의 위치를 바꾸는 것이 쉽지 않아 보입니다.
- 대신에 square root decomposition을 이용하여 해결할 수 있습니다.

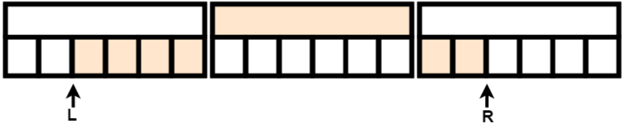


- 우선 전체 문자열을 $\sqrt{N}$ 구간으로 나눈 것을 block이라고 하겠습니다.
- block안에는 구간 내 문자 하나를 담고 있는 노드들의 정보, 전체 구간을 나타내는 노드 하나와 적용되는 쿼리의 정보가 들어있습니다.

- 또한 저희는 block에 대해 split이라는 연산을 정의할 수 있습니다.
- 이 split 연산은 위치 k가 주어지면 block을 k번째에서 잘라 두 개의 block으로 만드는 연산입니다.
- 블록 하나의 크기가 $\sqrt{N}$ 이하이므로 split의 시간복잡도는 $\mathcal{O}(\sqrt{N})$ 입니다.



- (l, r) 범위의 4가 아닌 쿼리가 들어왔다고 생각합시다. $l, r+1$ 인 위치에서 split을 하고 각 블럭의 쿼리 정보를 업데이트 합니다. 이 과정은 $\mathcal{O}(\sqrt{N})$ 에 가능합니다.
- 2,3번의 쿼리의 경우 구간에 속하는 block를 reverse 해줍니다. 이 때 block의 개수를 T 개라고 하면 $\mathcal{O}(T)$ 만큼의 시간이 걸립니다.



- 4번 쿼리가 들어오는 경우를 생각해 봅시다.
  - 쿼리가 block에 걸치는 경우 걸치는 노드들은 모두 합쳐줍니다.
  - 쿼리가 block을 포함하는 경우 block 전체 구간을 나타내는 노드를 합쳐줍니다.
- 쿼리가 block에 걸치는 경우가 최대 두 번 있을 수 있고 block의 길이가 최대 $\sqrt{N}$ 이므로 이를 계산하는 것은 $\mathcal{O}(\sqrt{N})$ 만큼 걸립니다.
- 쿼리가 block을 포함하는 경우는 최대 block의 개수 T 번 있을 수 있으므로 $\mathcal{O}(T)$ 만큼 걸립니다.

- 처음에는 block의 개수가 $\sqrt{N}$ 개이지만 쿼리를 여러 번 진행할수록 block의 개수가 늘어 비효율적이게 됩니다.
- 이를 해결하기 위해서 쿼리를 $\sqrt{N}$ 번 진행할 때마다 기존의 block들을 없애버리고 새로 $\sqrt{N}$ 개의 block을 만듭니다.
- 우선 한 쿼리당 최대 2개의 block이 생겨날 수 있고 $\sqrt{N}$ 번의 쿼리 후 최대 $3\sqrt{N}$ 개의 block이 존재할 수 있기 때문에 모든 쿼리가 $\mathcal{O}(\sqrt{N})$ 안에 해결됩니다.
- 또한 새로 block을 만드는 작업은 $O(N)$ 이 걸리는데 $\sqrt{N}$ 번마다 반복하므로 amortized $\mathcal{O}(N/\sqrt{N}) = \mathcal{O}(\sqrt{N})$ 에 해결됩니다.
- 따라서 시간복잡도 $\mathcal{O}(Q\sqrt{N})$ 으로 문제를 해결할 수 있습니다.