SNUPC 2023 풀이

Official Solutions

by

SNUPC 2023

2A. 경우의 수의 합

`implementation, math`

- 제출 65번, 정답 47명 (정답률 72.308%)
- 처음 푼 사람: **김상훈**, 57초
- 출제자: `16silver`

2A. 경우의 수의 합

- $f(a, b, c)$: $0 \le x \le a, 0 \le y \le b, x + y = c$를 모두 만족하는 정수 순서쌍 (x, y)의 개수
- 구해야 하는 값은

$$\sum_{i=0}^{n+m} f(n, m, i) = f(n, m, 0) + f(n, m, 1) + \cdots + f(n, m, n + m)$$

- $1 \le n, m \le$ **100**

2A. 경우의 수의 합

- 풀이 1
  - n, m의 제한이 크지 않으므로, $0 \le x \le n, 0 \le y \le m$인 모든 순서쌍을 확인하여 $f(n, m, i)$를 모두 구할 수 있습니다.
  - 시간복잡도는 $\mathcal{O}(nm(n+m))$ 또는 $\mathcal{O}(nm)$입니다.

– 풀이 2

- $f(n, m, c)$를 직접 계산해봅시다.
- 일반성을 잃지 않고 $n \leq m$라 하면, $x + y = c$가 되기 위해서 x는 $0 \leq x \leq n$와 $0 \leq c - x \leq m$, 즉 $c - m \leq x \leq c$를 모두 만족해야 합니다.
- 따라서 $f(n, m, c) = \min(n, c) - \max(0, c - m) + 1$입니다.

$$f(n,m,c) = \begin{cases} c+1 & (0 \leq c \leq n) \\ n+1 & (n < c \leq m) \\ n+m+1-c & (m < c \leq n+m) \end{cases}$$

- 이 값을 모두 합하면 $\mathcal{O}(n + m)$의 시간복잡도로 답을 구할 수 있습니다.

– 풀이 3

- $f(n, m, c)$ 의 값은 3개의 등차수열을 이루게 됩니다.
- 각 등차수열에 대해 등차수열의 합 공식을 사용하여 $\mathcal{O}(1)$ 에 합을 계산할 수 있습니다.
- 모두 계산하여 합한 다음 식을 정리해보면 답은 $(n+1)(m+1)$ 입니다.

$$\begin{aligned}\sum_{i=0}^{n+m} f(n,m,i) &= \sum_{i=1}^{n+1} i + (n+1)\cdot(m-n) + \sum_{i=1}^{n} i \\ &= \frac{(n+1)(n+2)}{2} + (n+1)(m-n) + \frac{n(n+1)}{2} \\ &= (n+1)(m+1)\end{aligned}$$

– 풀이 3

- 복잡한 과정에 비해 답은 매우 간단합니다.
- 사실 문제에서 구해야 하는 값은, $0 \le x \le n, 0 \le y \le m$ 인 정수 순서쌍 (x, y) 의 총 개수를 다른 방법으로 나타낸 것에 불과합니다.
- 따라서 구하려는 답은 $0 \le x \le n, 0 \le y \le m$ 인 정수 순서쌍 (x, y) 의 총 개수인 $(n+1)(m+1)$ 입니다.
- 이렇게 같은 경우의 수를 여러 가지 방법으로 구하여 결론을 얻어내는 기법을 **Double Counting** 이라고 합니다.

2B. 여우의 꿈

`recursion`

- 제출 96번, 정답 27명 (정답률 28.125%)
- 처음 푼 사람: **김상훈**, 10분
- 출제자: `sharaelong`

2B. 여우의 꿈

- 반지름 N 인 원판이 도달해야 하는 기둥이 $L(\neq K)$ 인 경우를 생각해봅시다.
- 최종 상태에 도달하기 전의 어떤 상태에서는 반지름 N 인 원판이 기둥 L 로 이동해야 합니다.
- 반지름 N 인 원판을 기둥 A 에서 B 로 이동하기 위한 조건은 다음과 같습니다.
- 1. 기둥 A 에는 반지름 N 인 원판 이외의 원판이 존재해서는 안 된다.
- 2. 기둥 B 에는 원판이 존재해서는 안 된다.
- 따라서 반지름 1 부터 $N-1$ 까지의 모든 원판은 A 와 B 가 아닌 다른 기둥에 있어야 합니다.

2B. 여우의 꿈

- 반지름 N 인 원판이 L 에 도달하기 전, 다른 기둥을 경유하는 것은 항상 손해입니다.
- 반지름 N 인 원판을 이동하기 위해서는 다른 모든 원판을 한 기둥에 몰아넣어야 하기 때문입니다.
- 따라서 최종 상태에 도달하기 위해서는 다음 세 과정을 거쳐야 합니다.
- 1. 반지름 1 부터 $N-1$ 까지의 원판을 K 와 L 이 아닌 다른 기둥$(=6-K-L)$으로 모두 이동한다.
- 2. 반지름 N 인 원판을 기둥 K 에서 L 로 이동한다.
- 3. 반지름 1 부터 $N-1$ 까지의 원판을 아름다운 배열로 배치한다.

2B. 여우의 꿈

- 기둥 K에 있는 반지름 1부터 N까지의 원판을 아름다운 배열로 이동하는 최소 이동 횟수를 $f(N, K)$라고 하겠습니다.
- 반지름 N인 원판이 기둥 L에 있다면, 3번 과정의 최소 이동 횟수는 $f(N-1, 6-K-L)$로 표현할 수 있습니다.
- 따라서 1번과 2번 과정을 해결한다면 재귀적으로 문제를 해결할 수 있습니다.
- 2번 과정은 한 번의 이동이 필요합니다.

2B. 여우의 꿈

- 1번 과정은 반지름 1 부터 $N-1$ 까지의 원판에 대해 일반적인 하노이 문제를 수행하는 것과 같습니다.
- 반지름 1 부터 N 까지의 원판을 기둥 1 에서 3 으로 이동하는 일반적인 하노이 문제는 다음과 같은 재귀적 과정으로 풀 수 있습니다.
- a. 반지름 1 부터 $N-1$ 까지의 원판을 기둥 1 에서 2 로 모두 이동한다.
- b. 반지름 N 인 원판을 기둥 1 에서 3 으로 이동한다.
- c. 반지름 1 부터 $N-1$ 까지의 원판을 기둥 2 에서 3 으로 모두 이동한다.
- a번과 c번 과정은 모두 반지름 1 부터 $N-1$ 까지의 원판에 대해 일반적인 하노이 문제를 수행하는 것과 같습니다.

2B. 여우의 꿈

- N 개의 원판에 대한 하노이 문제의 최소 이동 횟수를 $h(N)$ 이라 할 때, 다음과 같은 점화식이 성립합니다.
- $h(N) = 2h(N-1) + 1, h(1) = 1$
- 일반해는 다음과 같습니다.
- $h(N) = 2^N - 1$

2B. 여우의 꿈

- 따라서 $L \neq K$ 인 경우의 최종 점화식은 다음과 같습니다.
- $f(N, K) = f(N-1, 6-L-K) + 2^N$

2B. 여우의 꿈

- 이제 반지름 N 인 원판이 초기에 있던 기둥이 K 인 경우를 생각합시다.
- 이때는 이 원판을 움직이지 않는 것이 이득입니다.
- 다른 원판들이 반지름 N 인 원판에 의해 움직임을 제약받을 일이 없고,
- 원판을 다른 곳으로 움직이더라도 최종 상태에서는 기둥 K 에 있어야 하기 때문입니다.

2B. 여우의 꿈

- $L = K$ 인 경우의 점화식은 다음과 같습니다.
- $f(N, K) = f(N-1, K)$

2B. 여우의 꿈

- f 값은 첫 번째 인자가 N 인 경우부터 1 인 경우까지 N 개만 계산하면 됩니다.
- 2^1 부터 2^N 을 $1,000,000,007$ 로 나눈 나머지는 $O(N)$ 에 전처리할 수 있습니다.
- 따라서 $O(N)$ 시간에 문제를 해결할 수 있습니다.

2C. 문자열 만들기 1

`constructive, adhoc`

- 제출 124번, 정답 18명 (정답률 14.516%)
- 처음 푼 사람: **신지환**, 20분
- 출제자: Karuna

2C. 문자열 만들기 1

- S, U로만 구성돼 있고, S, U의 개수가 같은 문자열이 하나 주어진다.
- 주어진 시행을 사용하여 문자열을 만드는 방법을 하나 구하자.

2C. 문자열 만들기 1

- 주어진 문자열은 올바른 괄호 문자열이 여러 개 이어져 있는 것과 같이 생각할 수 있다.
- 올바른 괄호 문자열이란, 다음과 같은 규칙을 정의된다.
  - 빈 문자열은 올바른 괄호 문자열이다.
  - X가 올바른 문자열이면, (X)도 올바른 문자열이다.
  - X, Y가 올바른 문자열이면, XY도 올바른 문자열이다.
  - 위 규칙으로 만들 수 없는 문자열은 올바른 괄호 문자열이 아니다.
- 예를 들어서, 예제 1에서 주어진 문자열 SUSU은 ()() 처럼, 예제 2에서 주어진 문자열 SUSUUS는 ()()() 처럼 생각할 수 있다.

2C. 문자열 만들기 1

- 주어진 문자열을 올바른 괄호 문자열에 대응시킨 후 생각한다.
- 문자열을 뒤에서 부터 보면서, 다음을 반복한다.
  - 문자열을 뒤에서부터 순회한다.
  - 현재 보고 있는 위치의 문자가 대응되는 괄호 문자열에서)에 대응되면, 현 위치의 문자가 U이면 1번 시행, S이면 3번 시행을 한다. 그 후 2번 시행을 한번 한다.
  - 현재 보고 있는 위치의 문자가 대응되는 괄호 문자열에서 (에 대응되면, 2번 시행을 한번 한다.

2C. 문자열 만들기 1

- 앞의 방법으로, 주어진 문자열을 생성할 수 있다.
- 총 문자가 N 개 있고, 각 위치마다 최대 2 번의 시행을 하므로, 최대 $2N$ 번의 시행안에 주어진 문자열을 생성할 수 있다.

2D. 소설

- 제출 93번, 정답 7명 (정답률 7.527
- 처음 푼 사람: **신지환**, 70분
- 출제자: `sharaelong`

2D. 소설

– 서현이가 작성한 소설은 아쉽게도 출판 직전에 치명적인 오류가 발견되어, SNUPC 2024에서 공개될 예정입니다.

2D. 소설

- 길이 N의 문자열 S이 주어집니다.
- 이 문자열을 한 줄에 L개의 문자가 들어가도록 표시할 때, 같은 문자가 K번 이상 연속되는 경우가 없도록 하고 싶습니다.
- 가능한 L의 최댓값을 구해야 합니다.

2D. 소설

- L의 최댓값을 구하는 문제이니, 이분 탐색을 기반으로 한 **파라메트릭 서치**를 이용해봅시다.
- 폭을 L로 고정하고 나면, 문자열을 앞에서부터 L개씩 끊어보면서, 각 부분 문자열 안에 연속된 문자가 K번 나타나는 지 확인하면 됩니다.
- 결정 문제를 $\mathcal{O}(N)$ 시간에 확인할 수 있으니, 전체 문제가 $\mathcal{O}(N \log N)$ 시간에 해결되었습니다...?

틀렸습니다

틀렸습니다

틀렸습니다

틀렸습니다

틀렸습니다

틀렸습니다

틀렸습니다

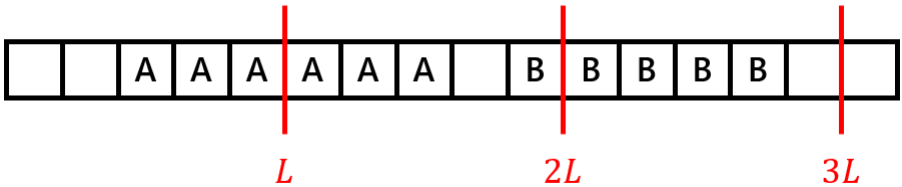
틀렸습니다

어라?

2D. 소설

- 이 문제에서는 파라메트릭 서치를 **사용할 수 없습니다**!
- $N = 6, K = 2, S =$ "abccab" 인 경우를 봅시다.
  - $L = 2$인 경우, ab | cc | ab이므로 NO
  - $L = 3$인 경우, abc | cab 이므로 OK
  - $L = 4$인 경우, abcc | ab 이므로 NO
- 파라메트릭 서치가 가능하려면, L이 증가함에 따라 답이 OK, OK, ..., OK, NO, NO, ..., NO와 같이 변하는 경우여야 합니다.

- 다른 접근으로 해결해야 합니다.
- 원래 문자열에서, 같은 문자가 K 개 이상 연속해 있는 구간들을 전부 모아봅시다.



- 그런 구간들은 $L, 2L, \cdots$ 지점에 의해 길이 K 미만의 구간들로 쪼개져야 합니다.

2D. 소설

- 각 구간 $[S, E]$에 대해서, S와 E를 L로 나눈 몫을 계산하고, 두 값을 비교합니다.
- 두 값이 같다는 것은, 해당 구간이 $L, 2L, \cdots$ 중 어떤 것에 의해서도 잘리지 않았다는 것을 의미합니다.
- 두 값이 다르다면, 잘린 구간들의 길이가 K 미만인 지 확인해주어야 합니다.
- 모든 구간이 길이 K 미만의 작은 구간들로 잘 쪼개졌다면 L은 답의 후보가 됩니다.

- 이런 방식으로 1 부터 N 까지의 모든 L 이 답의 후보가 되는 지 확인해줍시다.
- 하나의 L 에 대해서, 잘릴 수 있는 구간의 개수는 많아봤자 $\lfloor \frac{N}{L} \rfloor$ 개입니다.
- 따라서, 모든 L 에 대해서 답이 될 수 있는 지 확인하는 데에는 $\lfloor \frac{N}{1} \rfloor + \lfloor \frac{N}{2} \rfloor + \cdots + \lfloor \frac{N}{N} \rfloor$ 만큼의 시간이 걸립니다.
- 이 값은 $\int_1^N \frac{1}{x} dx$ 정도의 값을 가지므로, $\mathcal{O}(N \log N)$ 시간에 문제가 해결됩니다.

2E. 마술 도구

`math`, `constructive`, `ad_hoc`

- 제출 57번, 정답 12명 (정답률 22.807%)
- 처음 푼 사람: **최성준**, 47분
- 출제자: `sharaelong`

2E. 마술 도구

- 마술이 항상 성공한다는 말을 명확하게 쓸 필요가 있습니다.
- 이를 수학적으로 표현해봅시다.
- 마술에 실패하는 경우를 먼저 생각해봅시다.
- 만약 K 장의 카드가 모두 똑같다면 어떤 일이 일어날까요?
- 그러면 실버가 어떤 수를 생각하더라도 같은 답변이 돌아옵니다.
- 샤레롱의 입장에서 알 수 있는 정보는 실버의 답변뿐이므로, 일반화하면 실버의 답변이 같은 두 수가 있다면 두 수를 구분할 수 없다는 뜻이 됩니다.

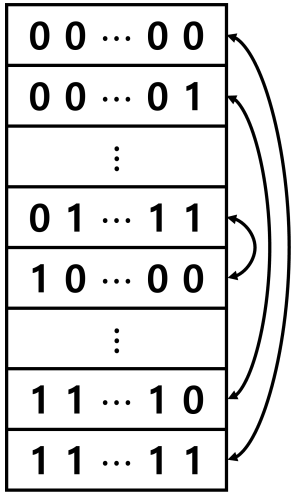
- 반대로 실버가 어떤 두 수를 생각해도 카드에 수가 써 있는 패턴이 다르다면, 최종적으로 샤레롱은 모든 수를 구분할 수 있습니다.
- 실버가 x를 생각했을 때, K장의 카드에 x가 쓰여 있다면 1, 아니라면 0이라고 표기하겠습니다.
- 그러면 x에 대한 실버의 답변은 길이 K의 이진 문자열 $S_x = 0100101$로 표기할 수 있습니다.
- 이제 문제를 다시 서술할 수 있습니다.
- **길이 K의 서로 다른 이진 문자열 N개를 만들되, 각 위치에 써 있는 1의 개수가 모두 같아야 한다.**

2E. 마술 도구

- 조건이 복잡하니, 1의 개수가 같다는 조건은 일단 제외하고 생각합시다.
- 서로 다른 이진 문자열 N 개를 만들 수 있는 최소의 K 부터 구해봅시다.
- 서로 다른 길이 K 의 이진 문자열은 총 2^K 개 존재함이 알려져 있습니다.
- 그렇다면 최소한 $2^K \geq N$ 을 만족해야 하므로, 최소한 $K \geq \lceil log_2 N \rceil$ 를 만족합니다.

- 1의 개수가 같다는 조건을 다시 생각할 차례입니다.
- 그런데 이 조건이 추가되어도 최소의 K가 유지될 수 있지 않을까요?
- 놀랍게도 그러한 구성을 항상 찾을 수 있습니다!
- 실버가 a와 b를 생각했을 때, a에 대응하는 문자열이 0100101라고 합시다.
- 그러면 샤레롱은 b에 0과 1을 뒤집은 문자열인 1011010을 대응시킵니다.
- 이렇게 하면 서로 다른 2개의 문자열을 각 위치에 정확히 1을 1개씩만 사용하면서 만들 수 있습니다.

2E. 마술 도구



- 따라서 N 이 짝수라면, (1) 현재까지 만들어지지 않은 문자열을 하나 뽑고, (2) 이를 뒤집은 문자열을 만드는 과정을 $\frac{N}{2}$ 번 반복할 수 있습니다.
- 한 번 뒤집은 문자열을 또다시 뒤집으면 원래 문자열이 되므로, 아직 사용되지 않은 문자열을 뒤집어도 항상 사용된 적이 없는 문자열이 만들어집니다.
- 그렇지만 N 이 홀수라면 어떨까요?
- $11...1$ 과 같은 문자열은 그 자체로 각 위치에 쓰인 1 의 개수가 같은 문자열입니다.
- 따라서 앞서 말한 시행을 똑같이 $\lfloor \frac{N}{2} \rfloor$ 번 시행하되, $11...1$ 을 뽑지 않기만 하면 됩니다. 여전히 문자열을 새로 뽑지 못하는 일은 벌어지지 않습니다.

2E. 마술 도구

- 이진 문자열을 2진수로 해석한다면 더 직관적으로 이해할 수도 있습니다.
- 일반적인 구현에서는, N 이 정확히 2^K 꼴인 경우 예외적인 사항이 나올 수 있습니다.
- 이러한 구성 말고도 다양하게 마술 도구들을 만들 수 있습니다!

2F. 징검다리 뒤로 건너기

`dp`

- 제출 12번, 정답 4명 (정답률 33.333%)
- 처음 푼 사람: **최민기**, 99분
- 출제자: `16silver`

2F. 징검다리 뒤로 건너기

- 앞으로 갈 때는 최대 K 칸, 뒤로 갈 때는 1 칸 이동 가능합니다.
- 이미 밟은 돌은 다시 밟을 수 없습니다.
- 이때 1 번 돌에서 N 번 돌까지 가는 경우의 수를 구해야 합니다.

2F. 징검다리 뒤로 건너기

- 다이나믹 프로그래밍을 이용하여 문제를 풀어야 할 것 같습니다.
- 상태를 어떻게 정의해야 할까요?
- 단순히 i번 돌에 도달하는 경우의 수를 $dp(i)$로 놓는다면 다음과 같은 문제점이 생깁니다.
  1. 뒤로 몇 칸까지 갈 수 있는지 알 수 없습니다.
  2. $i+1$번 돌에서 i번 돌로 온 경우를 업데이트하기 어렵습니다.
- 이러한 문제점들을 해결하기 위해 다음과 같이 상태를 정의합니다.
- $dp(i, j)$: i번 돌에 도달하면서, **마지막 이동이 뒤로 간 이동이 아니고, 뒤로 j칸까지** 갈 수 있는 경우의 수

2F. 징검다리 뒤로 건너기

- $dp(i, j)$는 $1 \le x \le K$에 대해 $dp(i + x, x - 1)$에 영향을 줍니다.
- 상태 (i, j)에서 상태 $(i + x, x - 1)$로 가는 방법은, 뒤로 p번 연속으로 간 다음 앞으로 $p + x$칸 가는 것입니다.
- 이때 $0 \le p \le j, p + x \le K$를 만족해야 합니다.
- 이것을 만족하는 p의 개수는 $\min(j, K - x) + 1$입니다.
- 모든 업데이트를 마친 뒤 $dp(n, 0)$부터 $dp(n, k - 1)$까지의 합을 $10^9 + 7$로 나눈 나머지가 답이 됩니다.
- 시간복잡도는 $\mathcal{O}(NK^2)$입니다.

2G. 빙글빙글 물대포

`binary_search`

- 제출 0번, 정답 0명 (정답률 0.000%)
- 처음 푼 사람: -
- 출제자: `16silver`

2G. 빙글빙글 물대포

- $0 \le r \le N-2$일 때 $2N \cdot k + r$초에 i번 참가자의 물대포는 $i + r + 1$번 참가자를 향합니다.
- i번째 물대포는 초기 체력이 A_i고, T_i초마다 물대포를 발사합니다.
- 체력이 0이 되면 탈락하며 더 이상 물대포를 발사하지 않습니다.
- 무한한 시간이 흐른 뒤 끝까지 남는 물대포의 위치를 구해야 합니다.

- 현재 상태에서, **처음으로 탈락하는 위치**가 무엇인지가 중요합니다.
- 이때 1 **개 이상의 물대포가 탈락하는 시각**을 알아내는 것이 중요합니다.
- 시간 T 를 정하면, T 초의 시간이 흐른 뒤 1 개 이상의 물대포가 탈락하는지를 $\mathcal{O}(N^2)$ 에 판정할 수 있습니다.
- 이제 **이분 탐색**을 이용하면 1 개 이상의 물대포가 탈락하는 시각 T 을 알 수 있고, 이때 탈락하는 위치가 어디인지도 알 수 있습니다.

2G. 빙글빙글 물대포

- 탈락한 위치는 T 초까지 물대포를 쏜 것으로 생각하고, 그 물대포에 맞은 참가자들의 초기 체력을 맞은 횟수만큼 깎습니다.
- 이 과정을 더 이상 아무도 탈락하지 않을 때까지 반복하면 됩니다.
- 시간복잡도는 $\mathcal{O}(N^3 \log M)$ where $M = n \max(A_i) \max(T_i)$ 입니다.

21. 트리 만들기

`math, constructive`

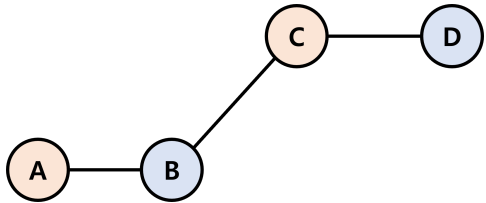
- 제출 0번, 정답 0명 (정답률 0.000%)
- 처음 푼 사람: -
- 출제자: `sharaelong`

21. 트리 만들기

- 정점이 N 개인 트리들을 일일이 확인하기에는 트리가 너무 많습니다.
- 그러므로 거리가 3인 정점 쌍에 특이한 성질이 있는지 관찰해보아야 합니다.

2I. 트리 만들기

- 먼저 트리가 주어졌을 때 거리가 3인 정점 쌍의 개수를 세는 방법부터 생각해봅시다.
- 거리가 3만큼 떨어진 정점 A, D가 있습니다.
- 이 정점 사이에는 B와 C도 존재합니다.



- 트리에서 두 정점 사이의 경로는 유일하므로, (B, C), (A, D)는 $1-1$ 대응입니다.
- 이 때 B와 C는 간선을 이루므로, 모든 간선을 순회하면서 간선에 연결된 정점들을 보면 됩니다.

- 즉 수식으로 쓰면 다음과 같습니다.

$$K = \frac{1}{2} \sum_{(u,v) \in E} (deg(u) - 1)(deg(v) - 1)$$

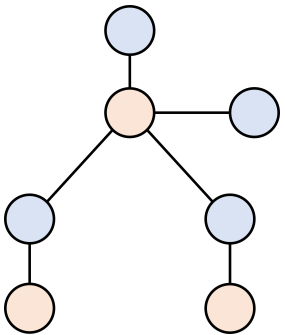
- 이 식을 어떻게 활용할 수 있을까요?

21. 트리 만들기

- 사실 별 방법이 없습니다.
- 트리를 만들기도 전에 트리의 간선에 대한 식을 활용하기는 무리가 있습니다.
- 다른 접근을 해볼 필요가 있습니다.

2I. 트리 만들기

- 이번에는 조건을 만족하는 트리가 존재하는 K 의 최솟값, 최댓값을 생각해봅시다.
- 트리의 좋은 성질 중 하나로, 다음과 같이 정점들에 색깔을 번갈아서 칠할 수 있음이 알려져 있습니다.



2I. 트리 만들기

- 그런데 거리가 홀수만큼 떨어진 정점들은 항상 다른 색으로 칠해져 있어야 합니다.
- 간선을 따라 정점을 이동할 때마다 색이 바뀌기 때문입니다.
- 즉 거리가 3만큼 떨어진 정점들의 색도 서로 다릅니다.
- 그러므로 K는 $(red, blue)$ 꼴인 정점 쌍의 개수와 관련있을 것이라고 생각할 수 있습니다.

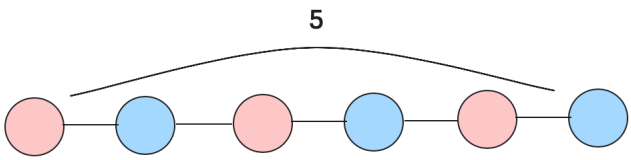
2I. 트리 만들기

- 파란색 정점의 개수를 b라고 두면, 빨간색 정점은 $N - b$개입니다.
- 따라서 거리가 3인 정점 쌍의 개수의 최댓값은 $b(N - b)$개입니다.
- 그러나 거리가 1인 정점 쌍들도 서로 색깔이 다릅니다!
- 이들은 **정확히** 간선의 개수인 $N - 1$개만큼 있습니다.
- 즉 K의 최댓값은 $b(N - b) - (N - 1)$개입니다.
- 이를 b에 대한 이차함수로 보면 최댓값을 구해볼 수 있습니다.

- 최솟값은 얼마일까요?
- b가 0이나 N에 가까울 때 최솟값을 가지게 됩니다.
- 정점이 2개 이상이므로, 파란색 정점이 최소 1개는 있습니다.
- 따라서 $1 * (N-1) - (N-1) = 0$이 가능한 최솟값입니다.

- 그러면 $K > 0$인 경우, 가능한 다음 값은 $K = 1$일까요?
- 일반적으로 아닙니다.
- $b = 2$이면 $2(N-2) - (N-1) = N-3$인데, 이 값은 거리 3인 정점 쌍**으로만** 이루어진 값입니다.
- 즉 거리가 5인 정점 쌍, 거리가 7인 정점 쌍... 들이 트리에 존재할 수 없습니다.

- 왜 그럴까요?
- 거리가 5인 정점 쌍이 트리에 존재한다고 가정합시다.
- 그러면 다음과 같이 트리를 색칠하는 색깔이 최소 3개씩은 사용되므로, $b \geq 3$이어야 합니다.

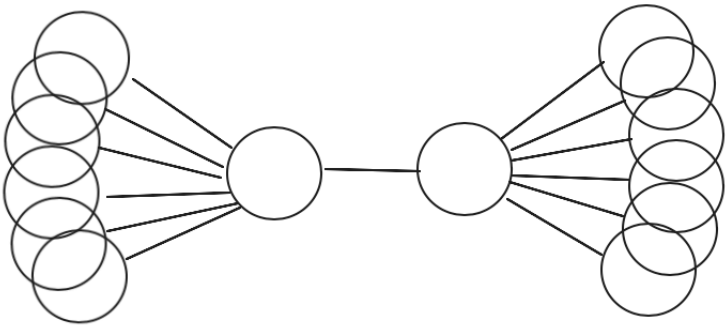


- 이 이후로는 더 이상 비슷한 분석을 진행할 수 없습니다.
- 정리하자면, 가능한 K 의 값은 0 또는 $[N-3, \frac{N^2}{4} - (N-1)]$ 의 범위에 속해야 함을 알 수 있습니다.
- 여기서 한 번 더 가설을 세워봅니다.
- **앞서 구한 범위에서는 항상 트리가 존재한다.**

- 그러나 아직 트리를 만들 수 있는 방법은 모릅니다.
- 일단 K 의 최소나 최대에 대해 트리를 직접 구성해봅시다.
- K 가 최대일 때는 조금 복잡한데, 앞에서 진행한 논의에 따르면 이 트리는 거리가 5 인 정점 쌍을 가지면 **안 됩니다.**
- 그러므로 만약 조건을 만족하는 트리가 존재한다면, 그 트리의 **지름**은 4 이하입니다.

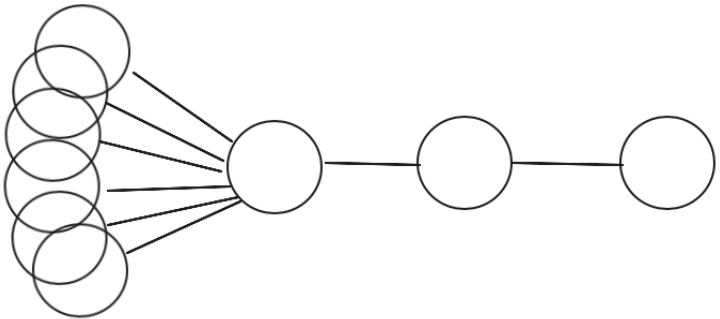
21. 트리 만들기

– 이렇게 생각하면 다음과 같은 경우 최대의 K 를 갖는 트리가 만들어집니다.



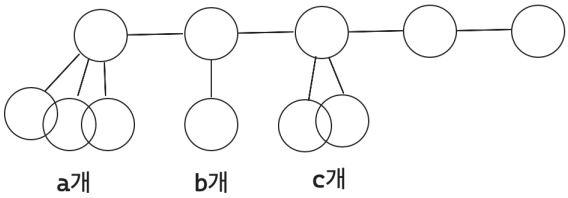
2I. 트리 만들기

– 게다가 최소의 $K = N - 3$에 대해서는 다음과 같은 트리가 존재합니다.



2I. 트리 만들기

- 더 나아가서, 지름이 작은 트리들로 범위 내의 K 를 전부 만들 수 있다고 추정할 수 있습니다.
- 지름을 먼저 그리고, 나머지 정점들을 지름 위에 놓이는 정점들에 다는 형태를 생각합시다.



- 따라서 거리 3 인 정점 쌍 개수에 대한 식을 쓰면 다음과 같습니다.

$$K = ab + bc + a + b + 2c + 2$$

2I. 트리 만들기

- 정리하면, $K - N + 3 = b(N - 5 - b) + c$입니다.
- b를 상수로 생각하면, $0 \le c \le N - 5 - b$입니다.
- 그러므로 고정된 b에 대해 $I_b = [b(N - 5 - b), (b + 1)(N - 5 - b)]$를 만들 수 있습니다.
- 이 때 b도 $0 \le bN - 5$를 만족하는 변수로 다시 봅시다.
- 최종적으로 저러한 트리 형태는

$$I = \bigcup_{b=0}^{N-5} I_b = [N - 3, \frac{N^2}{4} - (N - 1)]$$

를 모두 만들어낼 수 있습니다.

2I. 트리 만들기

- 일반적인 구현에서는, N이 작은 경우 예외 상황들이 나올 수 있습니다. 이 풀이에서 설명한 방식대로라면 $N \leq 4$인 경우를 따로 처리해줘야 합니다.
- 이러한 구성 말고도 다양하게 트리를 만들 수 있습니다!

1A. 재민이의 생일

- 제출 77번, 정답 18명 (정답률 23.377%)
- 처음 푼 사람: **조승현**, 11분
- 출제자: `arnold518`

1A. 재민이의 생일

- $H \times W$ 모양의 격자판이 주어지고, 각 칸의 양의 정수 가중치 S_{ij} 가 주어져 있습니다.
- 넓이가 정확히 N 이 되도록 직사각형을 선택해, 내부에 있는 칸들의 가중치를 봅니다.
- 가중치 중 최댓값과 최솟값의 차이를 최대화해야 합니다.

1A. 재민이의 생일

- 넓이가 N 이 되도록, 직사각형의 가로 길이 R과 세로 길이 C를 고정합시다.
- 모든 $R \times C$ 크기의 직사각형에 대해서, 가중치의 최댓값과 최솟값을 각각 구하는 걸 시도해봅시다.

1A. 재민이의 생일

$R = 2$, $C = 4$인 경우입니다.

2	6	5	8	1	2	5	9	7
1	3	3	1	2	3	8	7	3
2	3	9	9	8	8	8	2	3
5	4	9	6	6	3	5	6	2

1A. 재민이의 생일

– 우선, 각 $1 \times C$ 직사각형에 대한 최댓값을 전부 구합니다.

2	6	5	8	1	2	5	9	7
1	3	3	1	2	3	8	7	3
2	3	9	9	8	8	8	2	3
5	4	9	6	6	3	5	6	2

1A. 재민이의 생일

– 우선, 각 $1 \times C$ 직사각형에 대한 최댓값을 전부 구합니다.

8	8

2	6	5	8	1	2	5	9	7
1	3	3	1	2	3	8	7	3
2	3	9	9	8	8	8	2	3
5	4	9	6	6	3	5	6	2

1A. 재민이의 생일

– 우선, 각 $1 \times C$ 직사각형에 대한 최댓값을 전부 구합니다.

8	8	8	8	9

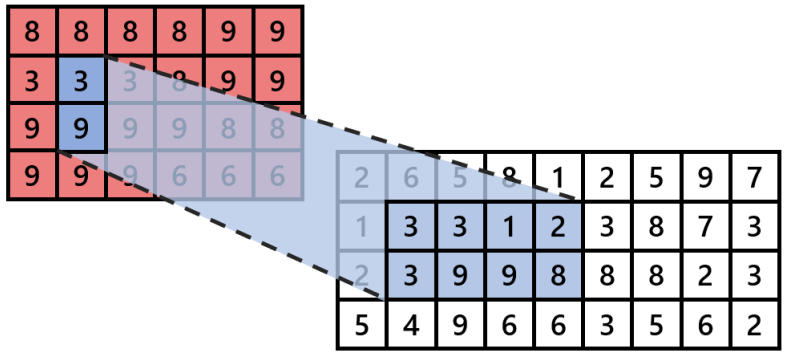
2	6	5	8	1	2	5	9	7
1	3	3	1	2	3	8	7	3
2	3	9	9	8	8	8	2	3
5	4	9	6	6	3	5	6	2

1A. 재민이의 생일

- 다양한 방법으로 모든 $1 \times C$ 직사각형에 대한 최댓값을 구할 수 있습니다.
- 각 가로줄에 대해서, 인접한 C 개의 원소를 슬라이딩 윈도우로 보면서, 모노톤하게 덱을 관리하는 방식으로 $\mathcal{O}(W)$ 시간에 구할 수 있습니다.
- `set<int>`를 사용하거나 세그먼트 트리를 사용해서 $\mathcal{O}(W \log W)$ 시간에 구할 수도 있습니다.

1A. 재민이의 생일

– 어떤 $R \times C$ 직사각형 내부의 가중치 최댓값은, 이렇게 새로 만든 이차원 배열에서 $R \times 1$ 직사각형에 대한 최댓값과 같습니다.



- 이는, 앞서 $1 \times C$ 직사각형의 최댓값을 구하는 것과 같은 방법으로 할 수 있습니다.
- 최솟값에 대해서도 동일한 방법을 적용합시다.
- 이제 모든 $R \times C$ 직사각형에 대해서 최댓값과 최솟값을 각각 구할 수 있습니다.

1A. 재민이의 생일

- 가능한 R과 C의 쌍은 몇 가지나 있을까요?
- R과 C는 N의 약수여야 하므로, 가능한 (R, C) 쌍은 많아야 N의 약수의 개수만큼 존재합니다.
- N의 약수의 개수는 많아야 $2\sqrt{N}$개이므로, 모든 (R, C) 쌍에 대해서 앞서 설명한 작업을 다 해보더라도 $\mathcal{O}(HW\sqrt{N})$ 내지는 $\mathcal{O}(HW\sqrt{N}\log W)$ 시간에 문제를 해결할 수 있습니다.

1B/2H. 조교의 기묘한 시험

`data_structure`, `tree_set`, `prefix_sum`

- 제출 53번, 정답 22명 (정답률 41.509%)
- 처음 푼 사람: **이민제**, 28분
- 출제자: `16silver`

1B/2H. 조교의 기묘한 시험

– 다음과 같은 쿼리를 처리해야 하는 문제입니다.

- $1\ x$: x를 적은 사람이 들어옵니다.
- $2\ k$: k번째로 들어왔던 사람이 나갑니다.
- $3\ s$: 들어온 사람들 중 유일한 수를 적은 사람들이 1점을 얻고, 그 중 가장 큰 수를 적은 사람이 추가로 s점을 얻습니다.

1B/2H. 조교의 기묘한 시험

- k번째로 들어온 사람이 적은 정수는 배열 등에 저장합니다.
- x를 적은 사람이 누구인지를 알아야 하지만, 정확한 값을 알아야 하는 경우는 1명일 때뿐입니다.
- 따라서 x를 적은 **사람의 수**와, x를 적은 사람의 **인덱스의 합**만 알고 있어도 됩니다.
  - C++의 `set` 등으로 관리해도 상관은 없습니다.
  - 합 대신 xor을 사용할 수도 있습니다. 참고로 출제자는 xor을 좋아합니다.
- x를 적은 사람의 수가 1명인지의 여부가 바뀔 때마다 3번 쿼리에서 x를 적은 사람이 점수를 얻는지 여부가 바뀝니다.
- x를 적은 사람이 몇 번째 3번 쿼리부터 점수를 얻는지를 안다면, x를 적은 사람이 점수를 얻는지 여부가 바뀔 때만 점수 업데이트를 하는 것으로 3번 쿼리의 추가 s점을 제외한 부분을 빠르게 처리할 수 있습니다.

1B/2H. 조교의 기묘한 시험

- 추가 s 점을 받을 사람을 구하기 위해서는 유일한 수의 집합에서 가장 큰 수를 빠르게 알아내야 합니다.
- C++의 `set::max`가 정확히 이 동작을 $\mathcal{O}(\log N)$에 할 수 있습니다.
- 값을 우선순위 큐(max heap)에 넣되, 각 값이 빠져야 하는지의 여부를 따로 저장해두고 느리게 갱신하는(lazy update) 방법도 가능합니다.

1B/2H. 조교의 기묘한 시험

- 정리해보면 다음과 같습니다.
  - 각 정수 x에 대해, x를 적은 사람의 수(`cnt`), x를 적은 사람의 인덱스의 합(`idxs`), x를 적은 사람이 몇 번째 3번 쿼리부터 점수를 얻는지(`qnum`)를 저장하고 관리합니다.
  - 현재까지의 3번 쿼리의 개수(`q3_cnt`)를 저장하고, k번째로 들어온 사람이 적은 정수를 배열에 저장합니다.
  - x를 적은 사람이 1명이 아니었다가 1명이 되었다면, `qnum`을 `q3_cnt`로 업데이트합니다.
  - x를 적은 사람이 1명이었다가 1명이 아니게 되었다면, `q3_cnt`$-$`qnum`만큼 `idxs`번 사람의 점수를 더합니다.
  - 추가 s점을 받을 사람은 C++의 `set::max` 또는 우선순위 큐를 느리게 갱신하여 구합니다.
- 시간복잡도는 $\mathcal{O}(Q \log Q)$ 입니다.
- 구현에 따라 코드의 수행 시간은 많이 차이날 수 있습니다.

1E. 사과 바나나 트리

`tree`, `dp`

- 제출 33번, 정답 20명 (정답률 60.606%)
- 처음 푼 사람: **이민제**, 63분
- 출제자: `jhwest2`

1E. 사과 바나나 트리

- 트리의 각 정점에 A 또는 B가 적혀 있습니다.
- 간선으로 직접 연결된 두 정점에 적힌 알파벳을 바꾸는 시행(이동)을 할 수 있습니다.
- 최소 횟수의 이동으로 각 알파벳을 하나의 덩어리(컴포넌트)로 모아야 합니다.

1E. 사과 바나나 트리

- 목표하는 상태가 어떤 상태인지 먼저 생각해 봅시다.
- 둘 중 하나의 알파벳만 써 있다면, 이미 목적 달성입니다. 답은 0입니다.
- 그렇지 않다면, 어떤 간선이 존재하여 그 간선을 잘랐을 때 한 쪽에는 A만, 반대쪽에는 B만 써 있는 상태가 되어야 합니다.
  - 트리를 두 개의 연결 성분으로 나누는 방법은 간선을 하나 자르는 것 뿐이기 때문입니다.

1E. 사과 바나나 트리

- 트리에 A가 총 a개 적혀있다고 합시다.
- 어떤 간선을 기준으로 한쪽은 크기 a, 반대쪽은 크기 $N-a$ 짜리 컴포넌트로 나뉜다면 해당 간선을 자르는 것이 답의 후보가 됩니다. 크기 a인 쪽을 "왼쪽", $N-a$인 쪽을 "오른쪽" 이라 합시다.
- 왼쪽에 있는 B를 모두 오른쪽으로, 오른쪽에 있는 A를 모두 왼쪽으로 보내면 됩니다.
- 이 때, 최소 이동 횟수는 어떻게 구할 수 있을까요?

1E. 사과 바나나 트리

- 간선의 왼쪽에 붙은 노드를 u, 오른쪽에 붙은 노드를 v라고 합시다.
- 왼쪽 컴포넌트에서 B가 적힌 노드들을 $l_1, l_2, \ldots, l_m$, 오른쪽 컴포넌트에서 A가 적힌 노드들을 $r_1, r_2, \ldots, r_m$ 이라고 합시다.
- $d(x, y)$를 두 노드 x와 y를 잇는 경로의 간선 개수라고 하면, 최소 이동 횟수는 다음과 같습니다.

$$m + \sum_{i=1}^{m} d(u, l_i) + \sum_{i=1}^{m} d(v, r_i)$$

- 이 값은 이동 한 번당 많아야 1 줄어듭니다. 즉, 답의 하한입니다.
- 매번 l_i 중 u와 제일 가까운 것, r_i 중 v와 제일 가까운 것을 각각 u와 v까지 끌어다 놓은 후 $u \Leftrightarrow v$ 이동을 해주는 것을 반복하면 실제로 저 횟수만에 목표 달성이 가능합니다.

1E. 사과 바나나 트리

- 이제, 각 간선 및 방향에 대해 앞서 구한 식을 빠르게 계산할 수 있으면 문제가 풀립니다.
- 루트를 임의로 하나 정하면, 간선마다 한쪽 방향(부모 $\Rightarrow$ 자식)에 대한 값은 다음과 같은 트리 DP로 간단하게 구해집니다.
  - $C_A[u]$: u가 루트인 서브트리에 있는 A 개수
  - $D_A[u]$: u가 루트인 서브트리에서 A가 적힌 각 노드들에 대해 $d(u, x)$의 합
  - $C_B[i], D_B[i]$도 유사하게 정의할 수 있습니다.
- 이 값을 가지고 반대 방향에 대한 DP 값도 채워볼 수 있을까요?

1E. 사과 바나나 트리

- u가 루트가 아닐 때, 아래와 같은 "반대방향 DP값"을 정의합시다.
  - 전체 트리에서 u가 루트인 서브트리를 제외한 나머지 부분을 T라고 합시다. u의 부모를 p_u라고 합시다.
  - $UC_A[u]$: T에 있는 A 개수
  - $UD_A[u]$: T에서 A가 적힌 노드들에 대해 $d(p_u, x)$의 합
- u의 UC, UD 값을 구하기 위해서는 p_u에서 u 말고 다른 쪽으로 나가는 DP값들이 필요합니다.
- 즉, p_u의 각 자식들에 대한 C, D 값에 더해 p_u의 UC, UD 값이 필요합니다. (단, p_u가 루트면 필요 없음)
- DFS를 한 번 더 돌면서 순차적으로 UC, UD 값을 채워주면 됩니다.

1E. 사과 바나나 트리

- 반대방향 DP를 구하는 과정을 $O(N)$에 작동하도록 유의하여 구현해야 합니다.
- 부모에서 모든 방향으로 나가는 DP값의 합을 미리 구해 놓으면 간단히 구현 가능합니다.
- 전체 문제 역시 $O(N)$ 시간에 해결할 수 있습니다.

1H. SNUPC 게임

`game_theory`, `sprague_grundy_theorem`, `ad_hoc`

- 제출 27번, 정답 15명 (정답률 55.556%)
- 처음 푼 사람: **이하린**, 106분
- 출제자: `dhyang24`, `blackking26`

– 한별이가 승리했습니다.

- 각 칸에 있는 말의 개수가 크고, 두 부원이 취할 수 있는 행동의 종류가 똑같은 이상(Impartial game), Sprague-Grundy Theorem을 사용하고 싶습니다.
- 문제는, Sprague-Grundy Theorem은 게임에 루프가 없을 경우에만 사용할 수 있다는 점입니다.

- 하지만, 과연 진짜 그럴까요?
- 사실 루프가 있는 게임에도 Sprague-Grundy Theorem을 적용시킬 수 있습니다.
- Generalized Sprague-Grundy Function을 사용하면 (논문을 찾아보실 수 있습니다) 문제를 풀 **수는** 있습니다.
- 그러나 이와는 별개로 게임판의 성질을 관찰하여 문제를 더 쉽게 풀 수 있고, 당연히 이쪽이 정해입니다.

- 당연하게도 C에 놓인 말 개수는 답에 영향을 주지 않습니다.
- 먼저 누군가 확실히 이기는 경우들을 생각해 봅시다.
- S, N, U에 말이 놓여 있지 않고, P에 말이 짝수 개 놓여 있을 경우 후공이 승리합니다.
- 반대로 S, N, U에 말이 놓여 있지 않고, P에 말이 홀수 개 놓여 있을 경우 선공이 승리합니다.
- 한 단계 더 생각해보자면 S, N에 말이 놓여 있지 않고, U에 말이 정확히 하나 놓여 있고, P에 말이 홀수 개 놓여 있을 경우 선공이 승리합니다. 선공의 첫 번째 차례에 U에서 P로 말을 이동하면 되기 때문입니다.
- 이 세 경우를 **승부 결정 상태**라고 합시다.

1H. SNUPC 게임

- 이름에서 유추하셨을지도 모르지만, 사실 승부 결정 상태에서 시작하지 않은 게임은 항상 무승부가 됩니다!
- 증명의 아이디어는 승부 비결정 상태에서 후공 승인 승부 결정 상태로 이동할 수 없음을 보이는 것입니다.
- 후공 승인 승부 결정 상태는 S, N, U에 말이 없고 P에 말이 짝수 개 놓여 있는 경우 뿐인데, 이 상태의 직전 상태는 승부 결정 상태일 수밖에 없습니다.
- 승부 비결정 상태에서 선공 승인 승부 결정 상태로 이동하는 것은 패배를 자초하는 꼴이므로 해서는 안 되는 이동입니다.
- 즉 승부 비결정 상태에서는 승부 비결정 상태로밖에 이동할 수 없는데, 누군가 승리하기 직전의 상태는 명백히 승부 결정 상태이므로 승부 비결정 상태에서 시작하는 게임은 모두 무승부라는 결론을 얻습니다.

1D. 문자열 만들기 2

`constructive`, `adhoc`, `greedy`

- 제출 37번, 정답 10명 (정답률 27.027%)
- 처음 푼 사람: **김세빈**, 118분
- 출제자: `jhwest2`

1D. 문자열 만들기 2

- S, U로만 구성돼 있고, S, U의 개수가 같은 문자열이 하나 주어진다.
- 주어진 시행을 **최소로** 사용하여 문자열을 만드는 방법을 하나 구하자.

1D. 문자열 만들기 2

- 문자열의 문자 개수는 고정돼 있으므로, (1번 시행의 횟수)+(3번 시행의 횟수)는 $N/2$로 정해져 있다.
- 2번 시행의 횟수를 최소로 해야 한다.

1D. 문자열 만들기 2

- 문자열을 앞에서부터 보자
- 현재 보고 있는 위치의 문자가 S라고 했을 때, 이 S가 추가될 때 같이 추가된 U의 후보를 생각해보자.
- 가능한 후보들에 대해, 원래 문자열을 다음과 같이 표현할 수 있다. T = S T1 U T2 이때, T1, T2는 각각 S, U의 개수가 같아야 한다.
- T1, T2에 대한 부분문제를 해결했다고 했을 때, 원래 문자열을 만들려면 다음과 같이 하면 된다.
  - T2를 생성한다.
  - 2번 시행을 반복하여 커서를 T2의 왼쪽 끝으로 옮긴다.
  - 1번 시행을 한다. (현재 보고 있는 문자가 U면 3번 시행)
  - 2번 시행을 한다.
  - T1을 생성한다.

1D. 문자열 만들기 2

- 앞의 알고리즘을 분석하면, T2의 길이가 짧으면 짧을 수록 이득이다.
- 따라서, 가능한 후보 중 T2의 길이가 가장 짧은 것과 매칭 시킨 후, 앞의 알고리즘을 실행하면 시행을 최소로 하여 문자열을 생성할 수 있다.
- 단, 예외적으로, 현재 보고 있는 위치의 문자가 S라고 했을 때, 바로 다음 위치의 문자가 U일 경우, 이 U와 매칭 시켜줘야 한다.

1G. V

`math`, `bipartite_graph`

- 제출 6번, 정답 1명 (정답률 16.667%)
- 처음 푼 사람: **김세빈**, 217분
- 출제자: `sharaelong`, `16silver`

1G. V

- 정점마다 정수 a_i 가 적혀 있습니다.
- (u, v), (u, w) 를 연결하는 간선이 있을 때 a_v, a_w 에 같은 정수를 더할 수 있습니다.
- 위 연산을 0 번 이상 적용하여 모든 정점에 같은 정수가 적히도록 할 수 있는지 판정해야 합니다.

1G. V

- 주어진 그래프 $G(V, E)$에 대해 새로운 그래프 $G'(V, E')$을, (u, v), (u, w)를 연결하는 간선이 있을 때 v와 w를 연결하여 구성합니다.
- 새로운 그래프에서, 인접한 두 정점에 같은 정수를 더할 수 있을 때 모든 정점에 같은 정수가 적히도록 할 수 있는지 판정하면 됩니다.
- 그래프의 각 연결 성분은 독립적입니다.
- 따라서 각 연결 성분마다 **어떤 값으로** 모두 같게 만들 수 있는지를 알아내면 됩니다.
- 그래프의 형태에 따라 가능한 값은 없거나, 1개 있거나, 무한히 많을 수 있습니다.

Case 1. 그래프가 트리인 경우

- 트리는 이분 그래프입니다.
- 홀수점의 개수를 A, 적힌 정수의 합을 S라 합시다.
- 마찬가지로 짝수점의 개수를 B, 적힌 정수의 합을 T라 합시다.
- 인접한 두 정점 중 하나는 홀수점, 하나는 짝수점입니다.
- 따라서 **홀수점에 적힌 정수의 합 - 짝수점에 적힌 정수의 합**은 $S-T$로 변하지 않습니다.
- 모든 정점에 같은 정수 x가 적혔다면 위 값은 $(A-B)x$입니다.
- 따라서 $(A-B)x=S-T$를 만족하는 x만이 같게 만들 수 있는 값입니다.
- 이 값을 실제로 만들 수 있음은 귀납법 등을 이용하여 확인할 수 있습니다.
- 특히 $A=B$인 경우 모든 정수가 가능합니다.

Case 2. 그래프가 이분 그래프인 경우

- **홀수점에 적힌 정수의 합 - 짝수점에 적힌 정수의 합**의 불변성은 이분 그래프에서도 적용됩니다.
- 따라서 트리에서와 같이 $(A-B)x = S-T$를 만족하는 x만이 같게 만들 수 있는 값입니다.
- 실제로 만드는 것도 스패닝 트리를 잡으면 트리에서와 같습니다.

Case 3. 그래프가 이분 그래프가 아닌 경우

- 이분 그래프가 아니라면 길이가 3 이상의 홀수인 사이클이 존재합니다.
- 홀수 사이클에서는 정점 하나에 $2k$를 더할 수 있습니다.
- 스패닝 트리를 잡아서 홀수 사이클에 있는 정점 u을 제외하고 모두 x로 만들었을 때, u에 적힌 수와 x의 홀짝성이 같으면 가능합니다.
- 정점의 개수 A, 적힌 정수의 합 S의 홀짝성에 따라 다음과 같이 됩니다.
  - A가 홀수, B가 홀수: 모든 홀수로 가능
  - A가 홀수, B가 짝수: 모든 짝수로 가능
  - A가 짝수, B가 홀수: 불가능
  - A가 짝수, B가 짝수: 모든 정수로 가능

1G. V

- 이제 원래 문제에 대해서도 판정을 할 수는 있게 되었지만, 시간복잡도가 문제입니다.
- 새로 생성된 그래프에는 간선이 최대 $\mathcal{O}(V^2)$ 개 있을 수 있습니다.
- 원래 그래프의 연결 성분마다 새로운 연결 성분이 어떤 형태인지를 판정해 봅시다.

1G. V

Case 1. 그래프가 이분 그래프인 경우

- 새로운 그래프의 연결 성분은 2 개며, 각각 원래 그래프에서의 홀수점과 짝수점입니다.
- 새로운 그래프의 두 연결 성분을 각각 홀수 파생 그래프, 짝수 파생 그래프라 합시다.
- 홀수점 중 차수(degree)가 3 이상인 것이 있다면, 짝수 파생 그래프에는 길이 3 인 사이클이 존재하여 이분 그래프가 아닙니다.
- 반대로 모든 홀수점의 차수가 2 이하라면, 짝수 파생 그래프의 간선의 개수는 홀수점의 개수 이하입니다.
- 따라서 각 파생 그래프의 이분 그래프 여부를 $\mathcal{O}(V)$ 에 판정할 수 있으며, 이분 그래프인 경우 그 간선의 개수가 V 개 이하입니다.
- 이분 그래프가 아닌 경우 정점의 개수와 적힌 정수의 합만 구하면 되기 때문에 모든 판정은 $\mathcal{O}(V)$ 에 가능합니다.

Case 2. 그래프가 이분 그래프가 아닌 경우

- 새로운 그래프의 연결 성분은 1 개입니다.
- 원래 그래프에서 홀수 사이클을 이루는 정점들은 새로운 그래프에서도 홀수 사이클을 이룹니다.
- 따라서 새로운 그래프도 이분 그래프가 아니므로, 정점의 개수와 적힌 정수의 합만 구하면 됩니다.

1G. V

- 각 연결 성분을 분석하면서 나오는 **같게 만들 수 있는 값의 집합**은 다음 중 하나입니다.
  - 공집합
  - 원소가 1개인 집합
  - 모든 홀수의 집합
  - 모든 짝수의 집합
  - 모든 정수의 집합
- 위 분류 중 하나에 속하는 집합 두 개의 교집합을 구하는 것은 $\mathcal{O}(1)$에 가능합니다.
- 전체 시간복잡도: $\mathcal{O}(V+E)$

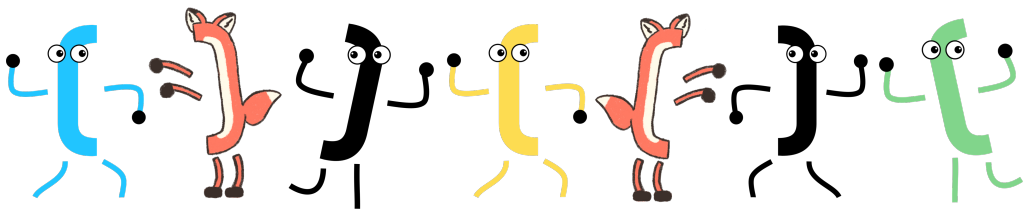
1F. 괄호 댄스

`mcmf`, `segment_tree`

- 제출 8번, 정답 0명 (정답률 0.000%)
- 처음 푼 사람: -
- 출제자: `jhwest2`, `arnold518`, `blackking26`

1F. 괄호 댄스

– 예제 입력 1 의 그림입니다.



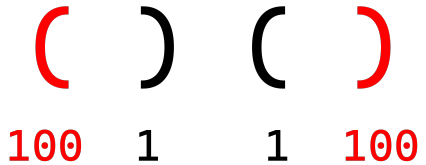
1F. 괄호 댄스

- 문제를 요약하면 다음과 같습니다.
- 여는 괄호와 닫는 괄호로 이루어진 길이 N 인 문자열이 주어집니다.
- 각 괄호에는 정수 가중치가 하나씩 존재합니다.
- 각 $K = 1, 2, \cdots, \lfloor \frac{N}{2} \rfloor$ 에 대해서, 주어진 문자열의 길이 $2K$ 인 subsequence를 선택해서, 다음 조건을 만족하도록 해야 합니다.
  - 올바른 괄호 문자열이어야 합니다.
  - 가중치의 합이 최대가 되어야 합니다.

1F. 괄호 댄스

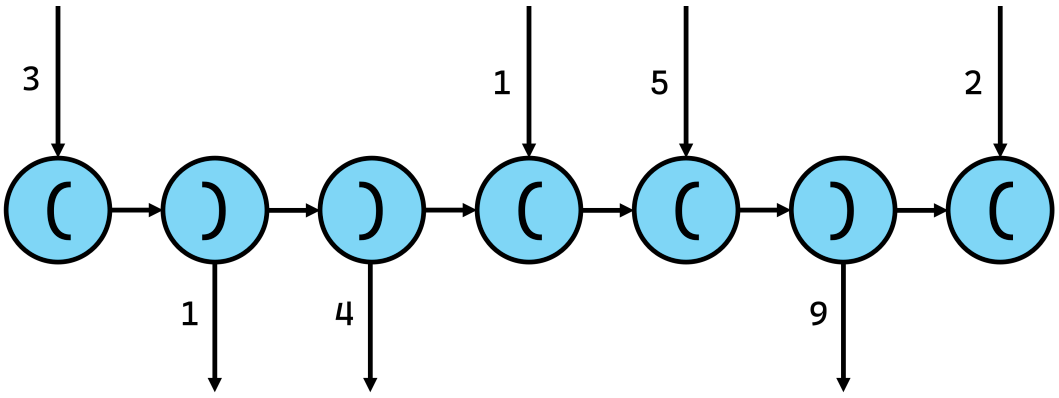
- 우선 다음과 같은 그리디 풀이를 생각해봅시다.
- $K = 1$일 때는 ()의 형태의 괄호 문자열만 가능하기 때문에, 가능한 모든 () 쌍에 대하여 가중치의 합을 최대화하면 됩니다.
- $K = 2, 3, \cdots$ 일 때도 아직 선택되지 않은 가능한 모든 () 쌍에 대하여 가중치의 합이 최대가 되는 쌍을 골라서 답에 더해주면 어떨까요?
- 이렇게 괄호들을 선택하면 항상 올바른 괄호 문자열을 얻을 수 있음은 보장됩니다.

- 하지만 다음과 같은 반례가 있습니다.



- $K = 1$일 때 양쪽 끝의 괄호들을 선택하게 되고, 그리디 알고리즘은 더 이상 괄호들을 선택하지 못합니다.
- 따라서, () 쌍만 선택하지 말고,)(과 같은 쌍들도 선택해야 합니다.
- 그렇다면 올바른 괄호문자열 조건을 지키며 어떻게 괄호들을 선택해야 가중치 합을 최대화할 수 있을까요?

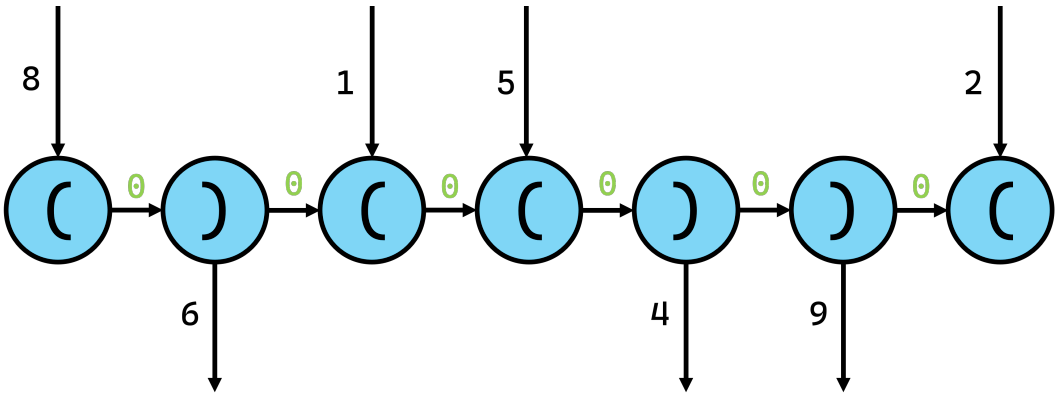
– 다음과 같이 flow network를 모델링합시다.



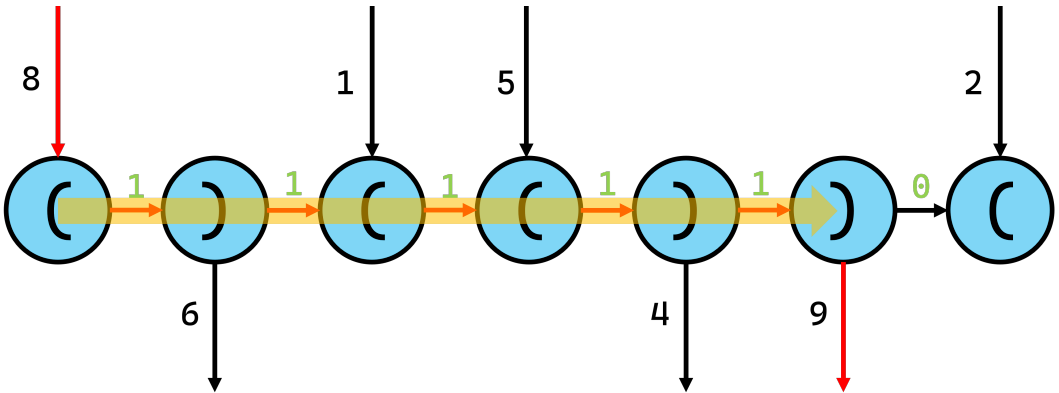
- 구체적으로, 각 간선에는 capacity와 cost가 있습니다.
- Source와 sink를 제외하고 총 N 개의 정점이 있으며 각 정점은 하나의 괄호를 의미합니다.
- 각 괄호들 사이에는 순서대로 i번 정점에서 $i+1$으로 가는 capacity ∞, cost 0의 간선이 있습니다.
- Source에서 여는 괄호 (인 i번 정점으로 가는 capacity 1, cost $A[i]$의 간선이 있습니다.
- 닫는 괄호) 인 i번 정점에서 sink로 가는 capacity 1, cost $A[i]$의 간선이 있습니다.

1F. 괄호 댄스

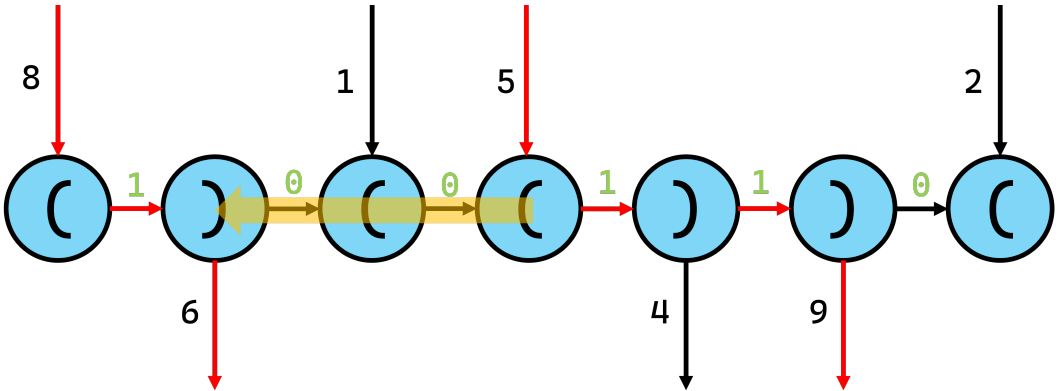
– 이와 같은 flow network에서 cost가 최대가 되도록 flow를 1 씩 흘려봅시다.



– flow를 1 흘린 후에는 그리디 풀이와 같은 괄호들이 선택됩니다.



– flow를 2 흘린 후에는 그리디 풀이와 다르게 괄호)(를 추가할 수 있습니다.



- 괄호들 사이의 반대 방향 flow는 매칭되는 괄호쌍을 변경시킵니다.
- 괄호들 사이의 간선의 flow 양이 항상 음이 아닌 정수여야 하기 때문에, 임의의 시점에도 올바른 괄호 문자열임이 보장됩니다.
- 이와 같은 flow network에서 k의 flow를 흘렸을 때의 최대 cost를 구하면 우리가 원하는 답을 구할 수 있습니다.
- MCMF의 시간복잡도는 $O(Ef)$인데, $E = N$, $f = N$이니 최단경로 알고리즘 등을 활용하여 MCMF를 구하면 $O(N^2)$의 시간복잡도에 문제를 해결할 수 있습니다.
- 이제 자료구조를 활용하여 위 알고리즘을 최적화합시다.

1F. 괄호 댄스

- 우리는 각 단계에서 valid한 경로 중 가장 cost의 합이 큰 경로를 선택하고 싶습니다.
- 경로가 오른쪽으로 향하는, 즉 (...)의 형태라면 따로 고려할 조건 없이, 가운데 지나는 경로 구간의 flow 양만 1 증가시키면 됩니다.
- 하지만 경로가 왼쪽으로 향하는, 즉)...(의 형태라면 고려해야할 조건이 있습니다.
- 사이에 지나는 경로 구간의 flow 양이 0 이라면 cancel할 flow가 없기 때문에, 경로 구간의 flow의 최솟값이 0 초과여야 합니다.
- 조건을 만족하는 최적의 flow를 찾은 후 가운데 지나는 경로 구간의 flow 양을 1 감소시키면 됩니다.
- MCMF의 정당성에 따라, 미래를 생각하지 말고 현재 상태에서의 최적 경로만 고려하면 됩니다.

1F. 괄호 댄스

- Segment tree를 사용하여 필요한 값들을 관리합시다.
- 우선, 반대 방향 간선에서 고려할 조건을 무시한다면 매우 잘 알려진 "금광" 문제에서 사용하는 segment tree를 응용할 수 있습니다.
- "(형태의 가중치 최댓값", ") 형태의 가중치 최댓값", "() 형태의 가중치 최댓값"과 그 위치, 그리고 구간의 flow의 최솟값을 관리하면 lazy segment tree를 이용하여 관리할 수 있습니다.

- 반대 방향 간선에서 업데이트를 하였을 때는 구간의 flow들이 감소할 수 있기 때문에 위 정보로는 충분하지 않습니다.
- "만약 현재 구간의 flow 최솟값이 0이 되도록 구간 전체에 음수 업데이트가 여러번 가해졌을 때"의 값을 저장하면 lazy update가 일어났을 때에도 segment tree의 값들을 $O(1)$에 갱신할 수 있습니다.
- "(형태의 가중치 최댓값", ") 형태의 가중치 최댓값", "() 형태의 가중치 최댓값", ")(형태의 가중치 최댓값"과 그 위치를 각각 "현재 상태"와 "만약 현재 구간의 flow 최솟값이 0이 되도록 음수 업데이트가 가해졌을 때"에 대해서 저장하고, 그리고 구간의 flow의 최솟값을 관리하면 최종적으로 lazy segment tree를 이용하여 관리할 수 있습니다.
- 시간복잡도는 새로운 flow를 흘려줄 때마다 $O(logN)$이니, 전체 $O(NlogN)$입니다.

1C. 두 수열

`greedy`, `string`, `suffix_array`

- 제출 4번, 정답 0명 (정답률 0.000%)
- 처음 푼 사람: -
- 출제자: `jhwest2`, `arnold518`, `blackking26`

1C. 두 수열

- 문제 내용을 간단하게 요약하면 다음과 같습니다.
  - 어떤 두 수열 P와 Q에 대해서, 새로운 수열 $P \star Q$를 정의합니다.
  - P와 Q의 원소들을 각 수열에서의 순서를 보존하면서 합친 결과 중, 사전 순으로 가장 작은 것이 $P \star Q$ 입니다.
  - 문제는 어떤 수열 A와 B에 대해서, $A[1..s] \star B[1..t]$의 k번째 수를 출력하는 쿼리에 답하는 것입니다.
- 설명의 편의를 위해서, **사전 순 최소**가 아닌 **사전 순 최대**를 구하는 것으로 생각합니다.
- $A_i \leftarrow 10^9 - A_i$의 변환을 통해 사전 순 최대를 구하는 문제로 바꿀 수 있습니다.

1C. 두 수열

- 문제가 복잡하게 생겼으니, 간단한 형태의 문제부터 먼저 해결해봅시다.
- 두 수열 A와 B가 주어졌을 때, $A \star B$는 어떻게 계산할 수 있을까요?

1C. 두 수열

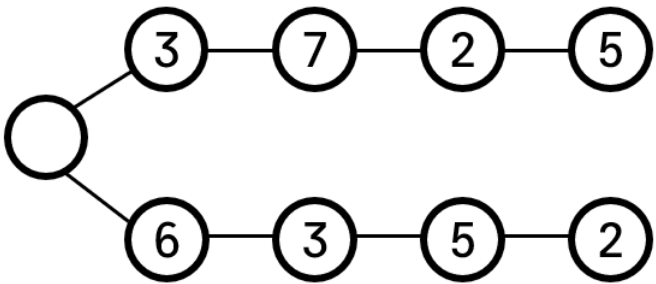
- $A \star B$를 구하는 과정을, 두 수열의 앞에서부터 원소를 하나씩 선택하는 것으로 이해합시다.
- A와 B의 모든 수 중 **가장 큰 수**에 집중합시다.
  - 해당 원소가 수열에서 가장 왼쪽에 위치하는 경우, 그냥 선택해주면 됩니다.
  - 그렇지 않은 경우, **자신의 바로 왼쪽에 위치하는 수**가 선택되는 순간 해당 수를 바로 선택하는 것이 좋습니다.
- 따라서, 가장 큰 수는 반드시 자신의 바로 왼쪽에 위치하는 수와 연달아서 선택되어야 합니다.

1C. 두 수열

- 가장 큰 수와 그 바로 왼쪽에 있는 수는 반드시 연속해서 선택됩니다.
- 따라서, 그냥 **두 수를 합쳐서 길이가 2인 수열처럼 생각**해도 좋습니다!

1C. 두 수열

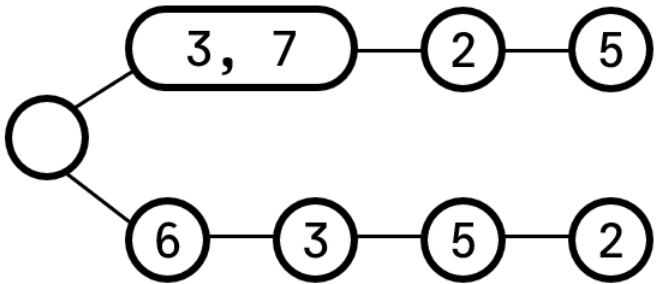
- 가장 큰 수와, 그 왼쪽에 있는 수를 합치는 작업을 반복합니다.



Ans : [ ]

1C. 두 수열

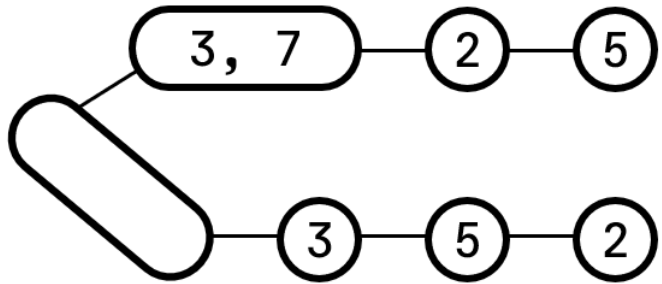
– 가장 큰 수와, 그 왼쪽에 있는 수를 합치는 작업을 반복합니다.



Ans : [ ]

1C. 두 수열

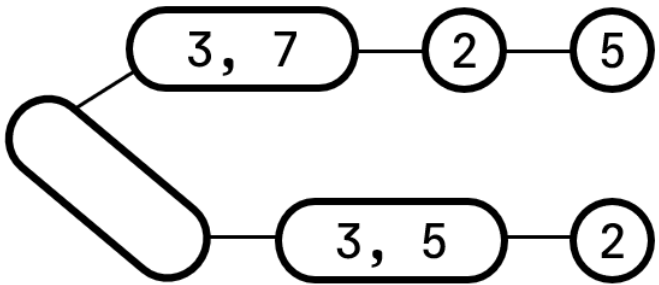
– 가장 큰 수와, 그 왼쪽에 있는 수를 합치는 작업을 반복합니다.




```
Ans : [ 6 ]
```

1C. 두 수열

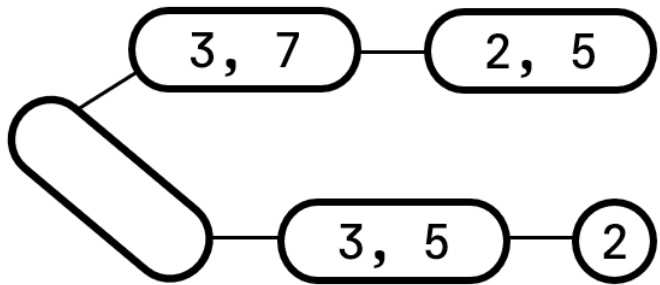
– 가장 큰 수와, 그 왼쪽에 있는 수를 합치는 작업을 반복합니다.



Ans : [6]

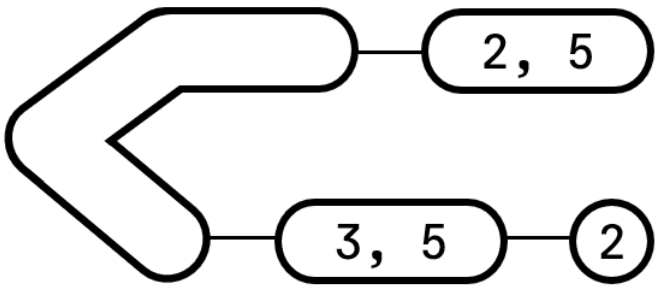
1C. 두 수열

– 가장 큰 수와, 그 왼쪽에 있는 수를 합치는 작업을 반복합니다.




```
Ans : [ 6 ]
```

– **가장 큰 수열**에 대해서, 그 왼쪽에 있는 **수열**과 합치는 작업을 반복합니다.




```
Ans : [ 6, 3, 7 ]
```

- 각 원소가 하나의 수가 아니라 수열이기 때문에, **가장 크다**의 의미가 모호합니다.
- 어떤 두 수열 S와 T에 대해서, $S <_\star T$를, $ST < TS$로 정의합시다. 이때,
  - $<_\star$은 잘 정의된 연산자이며,
  - 어떤 수열들을 $<_\star$를 기준으로 정렬하고 큰 원소부터 이어붙였을 때, 가능한 모든 순열 중 사전 순 최대가 됨

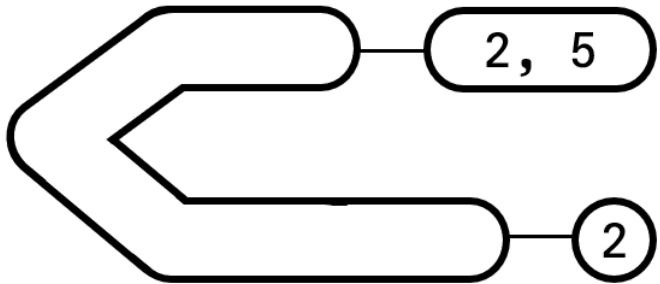
  이 알려져 있습니다.
- 어떤 수열이 **가장 크다**는 것을 $<_\star$ 기준으로 가장 크다는 것으로 정의합니다.

1C. 두 수열

- 아까와 같은 논리로, A와 B에 남은 수열 중 가장 큰 수열을 선택합니다.
- 바로 왼쪽에 위치하는 수열이 선택되는 순간 해당 수열을 무조건 선택하는 것이 좋습니다.
- 따라서, 다음 과정을 통해서 답을 얻을 수 있습니다.
  - 가장 큰 수열 중 가장 앞쪽에 위치한 것을 찾아서, 바로 왼쪽의 수열과 합치는 것을 반복합니다.
  - 전체가 하나의 수열로 합쳐지면, 과정을 종료합니다.
- 마지막에 남는 하나의 수열이 정답이 됩니다.

1C. 두 수열

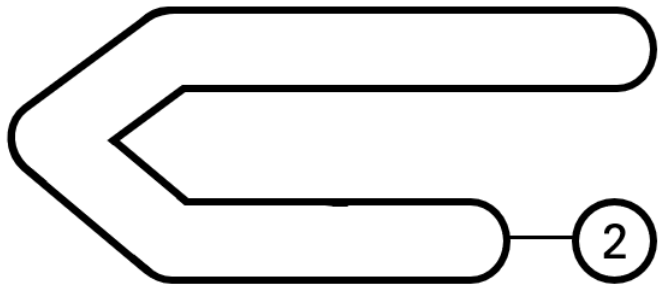
– **가장 큰 수열**에 대해서, 그 왼쪽에 있는 **수열**과 합치는 작업을 반복합니다.



Ans : [6, 3, 7, 3, 5]

1C. 두 수열

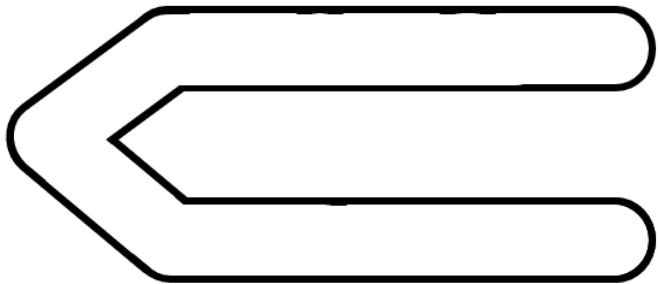
– **가장 큰 수열**에 대해서, 그 왼쪽에 있는 **수열**과 합치는 작업을 반복합니다.




```
Ans : [ 6, 3, 7, 3, 5, 2, 5 ]
```

1C. 두 수열

– **가장 큰 수열**에 대해서, 그 왼쪽에 있는 **수열**과 합치는 작업을 반복합니다.



Ans : [6, 3, 7, 3, 5, 2, 5, 2]

- 여담으로, 이 접근에서 사용한 관찰은 CERC 2013 Escape의 풀이에서 사용하는 아이디어로, 최근 SCPC 본선에 출제되기도 한 아이디어이므로 공부해보는 것을 추천합니다.
- 아쉽게도, 이 과정을 그대로 전체 문제의 풀이로 발전시킬 수는 없습니다. 이 과정을 그대로 구현한다면, 비교 연산에만 $\mathcal{O}(|S| + |T|)$ 시간이 걸려 $A \star B$ 를 시간 내에 구하는 것도 어렵습니다.
- 해싱 등을 이용해 비교 연산이 $\mathcal{O}(\log^2 N)$ 정도에 작동하도록 잘 관리할 수 있지만, 다소 복잡한 관계로 생략합니다.

1C. 두 수열

- 풀이로 발전시킬 수 없는 접근을 소개하는 데에는 이유가 있습니다.
- 방금 소개한 과정에서, 루트 노드랑 합쳐지는 수열들을 관찰합시다.
- `[6]`, `[3, 7]`, `[3, 5]`, `[2, 5]`, `[2]`
- 각 수열들은 **사전 순으로 감소**합니다!

Lyndon Decomposition

- 어떤 문자열 w가 **Lyndon word**라는 것은, 자신을 제외한 임의의 suffix보다 w가 strict하게 작다는 것을 의미합니다.
- 임의의 문자열 S를 다음과 같이 표현하는 방법이 유일하게 존재합니다. 이를 S의 **Lyndon decomposition**이라고 합니다.

$$S = w_1 w_2 \cdots w_k \ (w_1 \geq w_2 \geq \cdots \geq w_k)$$

Each w_i is a Lyndon word

- 다음 사실을 증명하는 것이 목표입니다.

  Theorem. $A \star B$의 Lyndon decomposition은 A와 B 각각의 Lyndon decomposition을 합치고 정렬한 것과 같다.

- 위 사실을 증명하기 위해, 우리는 앞서 얻은 그리디 알고리즘의 정당성을 이용합니다.

  Theorem. 위 알고리즘의 중간 과정에 등장하는 모든 수열들은 Lyndon word이다.

- 그리디 알고리즘에서, 매 단계 우리는 두 개의 인접한 수열을 합치는 것을 반복합니다.
- 중간 과정에 등장하는 모든 수열이 Lyndon word임을 보이기 위해, 다음 성질이 성립했으면 좋겠습니다.

Proposition 1. L_1, L_2가 각각 Lyndon word라고 하자. $L_1 < L_2$이 성립할 때, $L_1 L_2$는 Lyndon word이다.

- **Proposition 1.** L_1, L_2 가 각각 Lyndon word라고 하자. $L_1 < L_2$ 이 성립할 때, L_1L_2 는 Lyndon word이다.
- **Proof.** L_1L_2 가 자신을 제외한 모든 L_1L_2 의 suffix S 보다 작다는 것을 증명합니다.
  - $S = L_1'L_2$ 인 경우, Lyndon word의 조건에 의해 $L_1' > L_1$ 이므로, 첫 $|L_1'|$ 개의 수만 비교하더라도 $L_1L_2 < S$ 입니다.
  - $S = L_2'$ 인 경우, Lyndon word의 조건에 의해 $L_2' > L_2$ 이므로, 첫 $|L_2'|$ 개의 수만 비교하더라도 $L_2 < S$ 입니다. 이제 $L_1 < L_2$ 에서 $L_1L_2 < L_2$ 이므로, $L_1L_2 < S$ 입니다.

(이어서)

- **Proposition 1.** L_1, L_2가 각각 Lyndon word라고 하자. $L_1 < L_2$이 성립할 때, L_1L_2는 Lyndon word이다.
- **Proof.**
  - $S = L_2$인 경우,
    - L_1이 L_2의 prefix가 아닌 경우, L_1과 L_2의 첫 $|L_1|$개의 수 중 다른 것이 존재하므로, $L_1L_2 < L_2$가 성립합니다.
    - L_1이 L_2의 prefix인 경우, $L_2 = L_1L_3$이라고 씁니다. 이때 L_2가 Lyndon word라는 조건에 의해 $L_3 > L_1L_3$이므로, 양변의 prefix에 L_1을 붙였을 때 $L_2 = L_1L_3 > L_1L_1L_3 = L_1L_2$이 성립합니다. □

- **Proposition 2.** L_1, L_2가 각각 Lyndon word일 때, $L_1 < L_2 \Leftrightarrow L_1L_2 < L_2L_1$.
- **Proof.**

  $\Rightarrow$) Proposition 1에 의해 L_1L_2는 Lyndon word이므로, $L_1L_2 < L_2 < L_2L_1$이 성립합니다.

  $\Leftarrow$) $L_1 \geq L_2$를 가정하면, $L_1 = L_2$일 수는 없으므로 $L_1 > L_2$입니다. 이때 Proposition 1에 의해 L_2L_1이 Lyndon word이므로, $L_2L_1 < L_1 < L_1L_2$가 성립하여 모순입니다. $\square$

- **Corollary.** 위 알고리즘의 중간 과정에 등장하는 모든 수열들은 Lyndon word이다.
- **Proof.** 귀납적으로 증명합니다.
  - 초기 상태에서 각 수열은 길이가 1이므로, 정의에 의해 Lyndon word입니다.
  - 어떤 시점에 합쳐지는 두 수열을 U와 V라고 합시다. 귀납 가정에 의해 U와 V 각각은 Lyndon word이며, V를 선택할 때 사용한 정렬 기준에 의해, $UV < VU$가 성립합니다.
  - Proposition 2에 의해 $U < V$이고, Proposition 1에 의해 UV는 Lyndon word입니다. 따라서 합쳐진 이후의 수열도 Lyndon word입니다. □

- **Theorem 1.** 루트 노드와 합쳐지는 수열 A의 부분 수열들을 순서대로 나열하면, A의 Lyndon decomposition을 얻을 수 있다.
- **Proof.**
  - 부분 수열 U가 루트 노드와 합쳐지는 경우를 생각합시다.
  - Corollary에 의해 루트 노드와 합쳐지지 않은 수열들은 모두 Lyndon word이며, Proposition 2에 의해 U는 그 중에서 사전 순으로 가장 큰 수열입니다.
  - 즉, U를 루트 노드와 합친 뒤, 남아 있는 수열들은 전부 U보다 작거나 같습니다.

- **Theorem 1.** 루트 노드와 합쳐지는 수열 A의 부분 수열들을 순서대로 나열하면, A의 Lyndon decomposition을 얻을 수 있다.
- **Proof.**
  - 두 Lyndon word $L_1 < L_2$에 대해 $L_1 < L_1L_2 < L_2$이므로 합친 후의 Lyndon word가 L_2보다 strict하게 작음을 관찰합시다.
  - U 다음으로 루트 노드와 합쳐지는 A의 부분 수열은, 위 논의에 의해 반드시 U 이하입니다.
  - 따라서, 루트 노드와 합쳐지는 A의 부분 수열들을 순서대로 나열하면, 이들은 모두 Lyndon word이고, 사전 순으로 단조 감소하게 됩니다.
  - 즉, A의 유일한 Lyndon decomposition이 됩니다. □

- **Theorem 2.** $A \star B$의 Lyndon decomposition은 A와 B 각각의 Lyndon decomposition을 합치고 정렬한 것과 같다.
- **Proof.** A와 B 각각의 Lyndon decomposition을 합친 수열을 $A \star' B$로 정의합니다.
  - A와 B 각각의 Lyndon decomposition은 정렬되어 있으므로, $A \star' B$ 상에서 각 Lyndon word들의 순서가 유지됩니다. 따라서 $A \star' B$는 A와 B를 순서를 유지하면서 합친 수열 중 하나이고, 그런 수열들 중 사전 순 최대인 것을 $A \star B$로 정의했으므로 $A \star' B \leq A \star B$가 성립합니다.
  - 그리디 알고리즘의 결과가 $A \star B$이며, Theorem 1에 의해 $A \star B$는 A와 B 각각의 Lyndon decomposition을 적당한 순서로 합친 결과입니다. 한편 $A \star' B$는 A와 B의 Lyndon decomposition을 합친 수열 중 사전순으로 가장 큰 결과이므로, $A \star B \leq A \star' B$가 성립합니다. □

1C. 두 수열

- 이제야 전체 문제에 접근할 준비가 되었습니다.
- 각 쿼리에서 요구하는 것은 A와 B 각각의 prefix에 대한 Lyndon decomposition입니다. 즉, 우선은 어떤 수열의 Lyndon decomposition을 구하는 쉬운 방법이 있다면 좋겠습니다.

- **Proposition 3.** S의 Lyndon decomposition은 S의 가장 작은 suffix를 포함한다.
- **Proof.** S의 가장 작은 suffix T에 대해서, $S = S'T$라고 적습니다.
  - 우선, T는 Lyndon word입니다. T의 모든 suffix들을 고려했을 때 이는 S의 suffix이기도 하고, T는 S의 suffix 중 가장 작은 것으로 선택했기 때문입니다.
  - S'의 Lyndon decomposition을 생각했을 때, 가장 마지막으로 오는 Lyndon word를 U라고 합시다. 만약 $U < T$였다면, Proposition 2에 의해 UT는 Lyndon word이므로, $UT < T$가 성립합니다. 이는 T가 S의 가장 작은 suffix임에 모순입니다.
  - 따라서 S'의 Lyndon decomposition 뒤에 T를 추가하더라도 Lyndon decomposition의 성질을 모두 만족합니다. 따라서, Lyndon decomposition의 유일성에 의해 T는 S의 Lyndon decomposition에 포함됩니다. □

- 즉, A와 B에 대해서, 모든 prefix에 대한 가장 작은 suffix의 길이를 계산합니다.
- 이 값을 모두 계산했다면, 어떤 prefix에 대한 Lyndon decomposition은 $x \to x - sl(x)$의 간선들로 구성된 트리에서, 루트까지 올라가는 경로에 존재하는 모든 Lyndon word들을 순서대로 나열한 것이 됩니다.

1C. 두 수열

- Lyndon decomposition을 incremental하게 계산하는 Duval's algorithm을 응용하여 각 prefix에 대한 최소 suffix의 길이를 $\mathcal{O}(N)$ 시간에 계산할 수 있음이 알려져 있습니다.
- 이 값을 계산한 뒤, 총 $N + M$ 개의 최소 suffix들을 적당한 방법을 사용해서 정렬하고 사전순으로 번호를 매깁니다.
- 이제 A와 B에 대해서 각각 트리를 만들면, 정점에 사전순 번호가 붙어 있고, 루트까지 가는 경로 2개에 있는 모든 정점들 중 번호가 큰 것부터 그리디하게 사용했을 때 k번째 문자를 출력하는 문제가 됩니다.
- 적절한 자료구조를 이용해서 해결해줍시다. 시간 복잡도는 전처리에 $\mathcal{O}(N \log N + M \log M)$, 쿼리 당 $\mathcal{O}(\log N \log M)$ 입니다.

1X. 히스토그램 K개 빼기

`segment_tree`

- 제출 ???번, 정답 ???명
- 처음 푼 사람: **???**, ???분
- 출제자: `blackking26`, `azberjibiou`

1X. 히스토그램 K개 빼기

- 주어진 히스토그램에 대해, 기둥을 0개, 1개, $\ldots$, $N-1$개 빼서 만들 수 있는 히스토그램에 포함될 수 있는 가장 넓은 직사각형의 넓이를 각각 구해야 합니다.
- 이 문제는 원래 Div1 D번으로 출제될 예정인 문제였으나, 정해인 $O(N^2 \log N)$ 코드와 거의 비슷한 시간에 작동하는 $O(N^3)$ 코드가 발견되어 Open에만 출제하게 되었음을 알려 드립니다.
- 두 풀이 모두 설명드리겠습니다.

1X. 히스토그램 K개 빼기

- 우선 정해인 $O(N^2 \log N)$ 풀이입니다.
- 고려해야 할 직사각형의 높이는 원래 히스토그램의 기둥 높이에 해당하는 최대 N 종류 뿐입니다.
- 높이 h 를 고정하면, 각 기둥별로 해당 높이를 가지는 직사각형을 포함할 수 있는지 / 없는지에 따라 (즉, 높이가 h 이상인지 미만인지에 따라) 1 또는 0으로 나타낼 수 있습니다.
- 이제, 문제는 0과 1로 이루어진 수열에서 중간에 0을 몇 개 제거했을 때 연속한 1을 최대 몇 개까지 만들 수 있는지를 구하는 것입니다.

1X. 히스토그램 K개 빼기

- Window의 길이 w를 고정하고, 수열에서 연속한 w개씩의 원소들로 이루어진 window들을 봅시다.
- 이 중에서 유의미한 window는 물론 1을 가장 많이 포함하는 window입니다.
- 그 window가 1을 a개 포함한다면, 기둥을 $w-a$개 뺐을 때의 답의 후보에 ah를 추가해 줍니다.
- 모든 높이와 window의 길이에 대해 각각 후보가 하나씩 나올 텐데, 오직 이것들만 고려해도 전체 문제에 대한 답을 구할 수 있습니다!
- 임의의 K에 대해 기둥을 K개 뺐을 때의 정답에 해당하는 직사각형이 후보로 무조건 한 번은 등장하기 때문입니다.

1X. 히스토그램 K개 빼기

- 아까는 설명의 편의를 위해 h를 고정한 뒤 w를 고정했는데, 사실 w를 먼저 고정해도 무방합니다.
- w를 먼저 고정하면, 수열에서 0을 1로 하나씩 업데이트 해나가면서 매번 길이 w의 window 중 1을 가장 많이 포함하는 부분을 빠르게 구하는 문제가 됩니다.
- $C[i]$를 $[i, i+w-1]$ 구간의 1 개수라고 하면, C의 특정 구간에 1 더하기 및 최댓값 구하기를 빠르게 수행하면 되므로 Segment tree (with Lazy Propagation) 를 사용하면 됩니다.
- w 하나당 $O(N \log N)$ 시간이 걸리고, w는 1부터 N까지 가능하므로 전체 문제를 $O(N^2 \log N)$에 해결 가능합니다.

1X. 히스토그램 K개 빼기

- 지금부터는 매우 빠른 $O(N^3)$ 풀이를 설명드리겠습니다.
- 아까와 마찬가지로 h 를 고정하고, 0과 1의 수열로 문제를 바꿉시다.
- 목표는 각 K 에 대해 원소를 K 개 빼서 만들 수 있는 가장 긴 연속한 1의 길이를 구하는 것입니다.
- 인접한 0과 1들을 각각 한 덩어리로 묶어 봅시다.
- 1로 된 덩어리를 순서대로 $A_1, A_2, \ldots, A_m$ 이라 하면, $i \leq j$ 에 대해 A_i 부터 A_j 사이에 있는 0을 모두 지우는 경우만 고려하면 됩니다.

1X. 히스토그램 K개 빼기

- 지금의 경우에는 K 개의 기둥을 지웠을 때 답을 구하려면 $K-1$ 개 이하의 기둥을 지웠을 때 역시 고려해야 함에 유의합니다. 이 때 남길 수 있는 기둥 최대 개수가 $N-K$ 라는 점도 고려해야 합니다.
- 이 과정으로 답을 구하게 되면 h 하나당 $O(N^2)$ 만큼 걸리기 때문에 총 $O(N^3)$ 이고, 뭔가 별로 안 돌 거 같이 생겼습니다.
- 하지만 실제로 짜 보면, $O(N^2)$ 에 해당하는 for문의 구조가 매우 간단하고, 1로 이루어진 덩어리의 개수가 $N/2$ 이하라서 훨씬 빠르게 동작합니다.
- $N=4000$ 인 최악 데이터에 대해, for문이 도는 횟수가 총 $(1\cdot2/2+2\cdot3/3+\cdots+1999\cdot2000/2+2000\cdot2001/2+1999\cdot2000/2+\cdots+2\cdot3/2+1\cdot2/2)=$ 약 26억 이지만 BOJ 기준 3초 정도에 작동합니다.