ICPC SINCHON

2026.02.20

SUAPC

26 WINTER

CONTENTS

유림이와 하람이의
두쫀쿠 대작전

EASY

수열 정렬 수수께끼

HARD

숫자 놀이 3

MEDIUM

Yet Another
Binary Problem

CHALLENGING

가지가지

MEDIUM

트리와 쿼리 24

CHALLENGING

히스토그램에서의 거리와
가장 가까운 점

CHALLENGING

신촌 방수 계획

HARD

별동대

CHALLENGING/
EXTREME

지하철! 지하철!
몇호선? 몇호선?

MEDIUM

MC 히페리온

HARD

Designant.

EXTREME

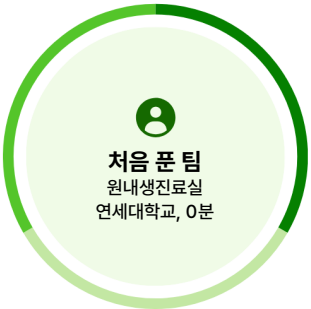
괄호 문자열 카드

EASY

A. 유림이와 하람이의 두쫀쿠 대작전

출제진 의도: Easy

#arithmetic #math



A. 유림이와 하람이의 두쫀쿠 대작전

- 각 재료의 가격을 더하면 두쫀쿠 하나의 원가는 2000원입니다.
- 유림이와 하람이가 가지고 있는 돈을 2000원으로 나누면
  만들 수 있는 두쫀쿠의 개수를 구할 수 있습니다.

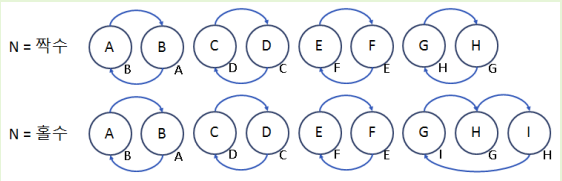
B. 수열 정렬 수수께끼

출제진 의도: Hard

#constructive #permutation cycle decomposition

B. 수열 정렬 수수께끼

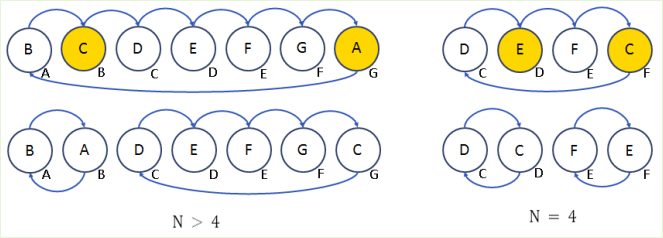
- 문제에 정의된 수열 A를 $(N+1)/2$ 이하의 교환 연산을 사용하여 정렬할 수 있으려면 아래의 조건을 만족해야 합니다.
  - N이 짝수라면, 길이가 2인 순열 사이클 $N/2$개
  - N이 홀수라면, 길이가 2인 순열 사이클 $(N-3)/2$개, 길이가 3인 순열 사이클 1개



- 길이 2인 사이클은 1회의 교환 연산으로, 길이 3인 사이클은 2회의 교환 연산으로 정렬할 수 있으므로 $(N+1)/2$회 조건을 만족합니다.

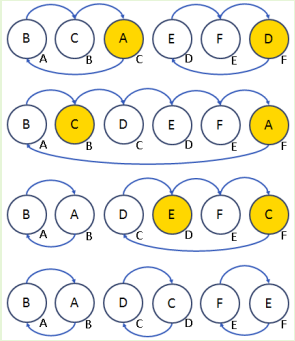
B. 수열 정렬 수수께끼

- 밤 사이에 위의 조건을 만족하도록 수열을 조작해야 합니다. A를 순열 사이클 분할하여 사이클을 만든 다음, 각 사이클의 길이(L)별로 다음과 같이 처리합니다.
- $L > 4$: 1회의 연산으로 길이 2, $L-2$인 사이클로 나눌 수 있습니다.
- $L = 4$: 1회의 연산으로 길이 2인 사이클 2개로 나눌 수 있습니다.



B. 수열 정렬 수수께끼

- $L = 3$인 사이클 2개 : 3회의 연산으로 길이 2인 사이클 3개로 나눌 수 있습니다.
- 위 3가지 모두 길이 2인 사이클 하나를 평균 1회 이내의 연산에 만들 수 있으므로, $N/2$회 이내의 교환 연산으로 수수께끼를 풀 수 있는 형태를 만족하도록 수열을 조작할 수 있습니다.



C. 숫자 놀이 3

출제진 의도: Medium

#math #priority queue

C. 숫자 놀이 3

- 먼저 주어진 연산을 반대로 바꾸어 봅시다.
  - x에 ℓ을 곱한 결과가 b진법에서 길이 ℓ이면, x에 ℓ을 곱한다.
- 그러면 b진법에서 한 자리 수인 수들(= 1 이상 $b-1$ 이하의 모든 정수)에서 시작하여 조건을 충족하는 ℓ을 곱해 나가는 것으로 모든 날씬한 수들을 만들어낼 수 있습니다.

C. 숫자 놀이 3

- 이제 날씬한 수들을 크기 순으로 k개를 뽑아내려면, 최소 힙을 사용하여 다음과 같이 구현할 수 있습니다.
  - 먼저 최소 힙에 1부터 $b-1$까지의 정수를 넣습니다.
  - 힙에서 수를 하나 꺼내서, 각각의 가능한 ℓ에 대해 곱한 결과의 길이가 ℓ이 맞는지 확인하고 힙에 그 곱을 넣는 과정을 k번 반복합니다.
- 10^{16}은 이진수로 54자리이므로, 고려해야 하는 ℓ의 값은 2 이상 54 이하로 잡으면 모든 테스트를 충분히 통과합니다.

C. 숫자 놀이 3

- 이렇게 구현하였을 때 실제로 힙에 들어가는 값의 총 개수는 $2k + b$를 초과하지 않습니다.
- 이는 임의의 x와 b에 대해 ℓx가 b진법으로 ℓ자리인 ℓ이 2개를 초과하지 않기 때문이며, 이는 다음의 사실로부터 확인할 수 있습니다.
  - $\ell x < b^\ell$일 때, $(\ell + 2)x \geq b^{\ell+1}$일 수 없습니다.
  - 이는 $\ell \geq 2$일 때 좌변은 최대 2배 증가하는데, $b \geq 2$이므로 우변 또한 최소 2배 증가하기 때문입니다.
- 실제로 조건을 충족하는 ℓ을 이분 탐색 등으로 빠르게 구하는 것으로 코드를 더 빠르게 만들 수 있지만, 이 문제를 푸는 데에 굳이 필요하지는 않습니다.
- 날씬한 수의 원래의 정의에 의해 특정한 날씬한 수에 도달하는 방법은 유일하므로, 중복 제거를 해줄 필요도 없습니다.

D. Yet Another Binary Problem

출제진 의도: Challenging

#constructive #prefix sum

D. Yet Another Binary Problem

- 1을 +1, 0을 -1로 하는 누적합배열 S를 생각해봅시다.
- 첫번째 시행을 하는 경우 S의 최솟값이 최대 1 증가하고 두번째 시행을 하는 경우 S의 최댓값이 최대 1 감소함을 알 수 있습니다.
- 즉, 초기 상태에서 S의 최솟값은 $-N$ 이상이여야 하고 최댓값은 N 이하여야 합니다.
- 이 경우, 실제로 답을 구성할 수 있음을 보일 수 있습니다.

D. Yet Another Binary Problem

- 앞으로 해야하는 첫번째 시행의 수를 a, 두번째 시행의 수를 b라고 합시다. 초기에는 $a = N, b = N$입니다.
- $a <$-(S의 최솟값), $b <$(S의 최댓값)인 경우, 아무 시행이나 해도 됩니다.
- $a =$-(S의 최솟값)인 경우, 두번째 시행을 더 이상 할 수 없을 때 까지 합니다.
- 그러면 $00\cdots0011\cdots11$꼴의 수열이 남고 이는 첫번째 시행만으로 처리 가능합니다.
- $b =$(S의 최댓값)인 경우에도 똑같이 처리 가능합니다.
- 이는 스택이나 세그먼트 트리 등을 이용해 구현할 수 있습니다.

E. 가지가지

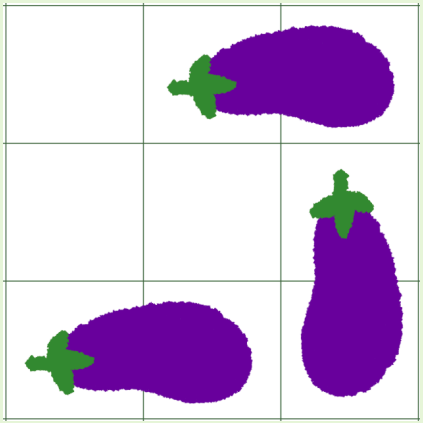
출제진 의도: Medium

#construcive #parity #invariant

E. 가지가지

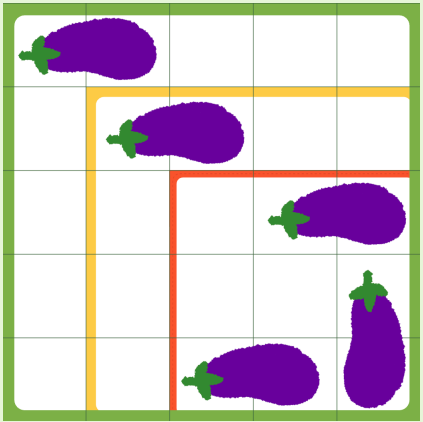
- $N \leq M$ 이라고 가정하겠습니다.
- 최소 1개의 꼭지가 모든 행과 열에 존재해야하기 때문에, 가지의 최소 개수는 M개 입니다.
- 꼭지는 반드시 하나의 행과 하나의 열을 차지합니다. 따라서 조건을 만족시키기 위해선 N과 M의 parity가 같아야합니다.
- 또한 $N = 2$ 이라면 가지의 최소 개수가 M개 이므로 격자판을 가지로 가득 채워야하는데, 2행에 꼭지를 배치하기 위해 1×2 크기의 가지를 사용하게 되면 이후 parity를 유지하면서 가득 채우는게 불가능해집니다.
- 이외의 경우는 모두 해 구성이 가능합니다. 방법은 여러가지가 있지만, 출제자는 3×3 에서 $N \times N$을 거쳐 $N \times M$까지 확장시키는 방법을 소개합니다.

E. 가지가지



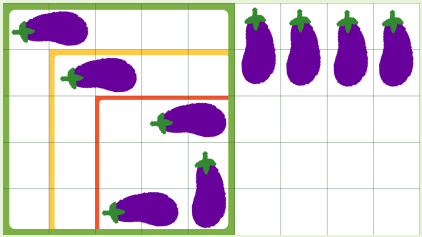
- 3×3을 구성하는 방법입니다. 모든 행과 열에 꼭지가 1개씩 있습니다.

E. 가지가지



- $N \times N$을 구성하는 방법입니다.
- 3×3 격자판에서 1×2 가지를 좌상단에 추가해주면 됩니다.
- 추가되는 가지는 기존 행과 열에 관여하지 않기 때문에, 모든 행과 열에 꼭지가 1개씩 있습니다.

E. 가지가지



- $N \times M$을 구성하는 방법입니다.
- $N \times N$을 구성하는데 가지 N개를 사용했습니다. 남은 가지는 $M - N$개가 됩니다.
- 여기서 N과 M의 parity가 같기 때문에 $M - N$은 짝수입니다.
- 1행은 꼭지가 1개 있기 때문에, 여기에 그림과 같이 일렬로 $M - N$개의 꼭지를 추가해도 여전히 꼭지의 개수는 홀수입니다.
- 1행을 제외한 나머지 행과 열은 꼭지가 1개씩 있게 됩니다.

E. 가지가지

Challenge.

- 격자판이 3차원이고 가지가 $1 \times 1 \times 2$ 크기의 블록이라면 어떨까요?

F. 트리와 쿼리 24

출제진 의도: Challenging

#lca #sqrt decomposition

F. 트리와 쿼리 24

- 루트로부터 더 멀리 떨어진 점을 한 칸씩 올려 주는 방법을 생각해 볼 수 있습니다.
- 하지만 위의 방법은 이동 횟수가 최대 $O(N)$이 될 수 있으므로, 이를 최적화하기 위해 한 번 이동할 때의 이동 거리가 같은 구간을 묶어서 생각해 봅시다.
- 예를 들어, $P = \{0, 1, 2, 2, 3, 4\}$ 라고 하면, 각 점에서의 이동 거리는 $\{0, 1, 1, 2, 2, 2\}$가 됩니다.
- 각 항이 $Q_i = i - P_i$인 새로운 수열 Q_i를 정의합니다.
- 쿼리가 적용된 이후에 Q_i의 각 항은 단조증가함을 쉽게 보일 수 있습니다.
- 또한, $P_p \geq 1$인 점 p에 대해 $Q_a = Q_p$, $Q_p \leq a \leq p$, $a \equiv p \pmod{Q_p}$을 만족하는 a는 p의 조상이 됩니다.

F. 트리와 쿼리 24

- 이제, 두 점 x와 y에 대해 다음의 과정을 반복하여 LCA를 찾을 수 있습니다. 일반성을 잃지 않고 $x > y$라고 가정합니다.
- Q_x와 Q_y가 다른 경우, $Q_a = Q_x$, $Q_x \leq a \leq x$, $a \equiv x \pmod{Q_x}$인 최소의 a를 찾고 x의 값을 $\max(P_a, 1)$로 바꿉니다.
- Q_x와 Q_y가 다른 경우, $Q_x \leq y \leq x$, $a \equiv y \pmod{Q_x}$이면 y가 x의 조상이므로 LCA가 됩니다. 그렇지 않은 경우에는 위의 과정을 진행합니다.
- 이때 위의 과정은 최대 $O(\sqrt{N})$회 반복됩니다. 증명은 아래와 같습니다.

F. 트리와 쿼리 24

- x의 값이 바뀌는 횟수를 K라고 해봅시다.
- x의 값이 바뀔 때마다, Q_x가 감소하므로 i번째 과정에서 $Q_x \leq K - i + 1$이 성립합니다.
- 또한 그 과정에서 x의 값이 Q_x 이상 감소하므로 다음과 같은 식을 세울 수 있습니다.
- $N \geq x - lca \geq \sum_{i=1}^{K} Q_x \geq \sum_{i=1}^{K}(K - i + 1) = \frac{K(K+1)}{2}$
- $K \leq \sqrt{2N}$이 되어 위의 명제가 성립합니다.

F. 트리와 쿼리 24

- 이제, 다음의 값을 빠르게 관리해 줌으로써 해결할 수 있습니다.
  - P_i (또는 Q_i)
  - $Q_a = Q_i$인 최소의 a
- 세그먼트 트리 등을 이용할 수도 있지만, 제곱근 분할법을 이용하면 두 쿼리를 모두 $O(\sqrt{N})$에 처리할 수 있습니다.
- $O(Q\sqrt{N})$만 통과시키는 것을 의도하였으나, 일부 $O(N\sqrt{N\log N})$이나 빠른 $O(N\sqrt{N}\log N)$ 풀이도 통과됩니다.

G. 히스토그램에서의 거리와 가장 가까운 점

출제진 의도: Challenging

#geometry #segment tree

G. 히스토그램에서의 거리와 가장 가까운 점

- 각 쿼리 점에 대해 가장 가까운 점 집합의 점을 구하기 위해서는 두 점 사이의 거리를 알아야 합니다.
- $x_1 \leq x_2$ 인 두 점 (x_1, y_1), (x_2, y_2) 사이의 거리는 $x_2 - x_1 + y_2 - min(y_1, y_2, h_{x_1}, h_{x_1+1}, \ldots, h_{x_2}) + y_1 - min(y_1, y_2, h_{x_1}, h_{x_1+1}, \ldots, h_{x_2})$ 입니다.
- 이는 두 점 사이를 이동하면서 x축 방향으로 이동할 수 있는 최소 거리가 $x_2 - x_1$, y축 방향으로 이동할 수 있는 최소 거리가 $y_2 - min(y_1, y_2, h_{x_1}, h_{x_1+1}, \ldots, h_{x_2}) + y_1 - min(y_1, y_2, h_{x_1}, h_{x_1+1}, \ldots, h_{x_2})$ 임을 확인하는 것으로 증명할 수 있습니다.
- 이 경우, 실제로 해당 이동 거리로 이동할 수 있습니다.

G. 히스토그램에서의 거리와 가장 가까운 점

- 두 점 중 한 점에서 출발하여 다른 점에 도착할 수 있으면 반대로 이동하는 것이 가능하므로 양쪽 점의 거리가 같습니다.
- 따라서 두 점 중 한 점에서 출발하여 다른 점에 도착하는 이동 거리를 구성할 수 있음을 보입시다.
- $(x_1, y_1) \to (x_1, min(y_1, y_2, h_{x_1}, h_{x_1+1}, \ldots, h_{x_2})) \to (x_2, min(y_1, y_2, h_{x_1}, h_{x_1+1}, \ldots, h_{x_2})) \to (x_2, y_2)$와 같은 방식으로 이동할 수 있습니다.
- 따라서, 두 점 간의 거리는 두 점의 좌표와 그 사이의 가장 작은 직사각형 막대 높이에만 의존하는 것을 알 수 있습니다.

G. 히스토그램에서의 거리와 가장 가까운 점

- 이제, 막대가 없는 일반적인 2차원 공간에서 두 점 (x_1, y_1), (x_2, y_2) 간의 거리가 $|x_1 - x_2| + |y_1 - y_2|$일 때 N개의 점 각각에 대해 가장 가까운 점과의 거리를 총 $O(N \log N)$에 구해봅시다.
- 모든 점에 대해 자신보다 x좌표가 작으면서 가장 가까운 점을 구해봅시다.
- 주어진 식에서 (x_1, y_1)이 (x_2, y_2)과 비교될 때는 거리가 $x_2 - x_1 + |y_1 - y_2|$인 것을 알 수 있습니다.
- 해당 식을 $y_1 - y_2$가 음이 아닌 정수거나 음수인 정수인 2개의 케이스로 나눠서 생각할 수 있습니다.
- y좌표 순서대로 미리 좌표압축을 한 후, 두 개의 세그먼트 트리에 각각 $-x_1 + y_1$, $-x_1 - y_1$을 저장합시다.

G. 히스토그램에서의 거리와 가장 가까운 점

- (x_2, y_2)보다 x좌표가 작거나 같은 모든 점에 대해 위 값이 저장되어 있다면 y_2보다 y값이 큰 구간과 작은 구간을 별개로 나누어 각 세그먼트 트리에서 참조하는 것으로 각 케이스에서 거리 값인 $x_2 - x_1 + |y_1 - y_2|$의 최솟값을 구할 수 있음을 알 수 있습니다.
- 두 케이스 중 최솟값이 각 점에 대해 자신보다 x좌표가 작으면서 가장 가까운 점과의 거리이고, 자신보다 x좌표가 크면서 가장 가까운 점과의 거리도 같은 방식으로 구할 수 있습니다.
- 이제 전체 문제를 풀어봅시다.
- 두 점간의 거리 공식을 증명하는 과정에서 해당 경로가 가장 낮은 직사각형의 꼭대기를 지나가게 구성하였습니다. 이에 집중합시다.

G. 히스토그램에서의 거리와 가장 가까운 점

- 세그먼트 트리의 i번째 인덱스에서 지나가는 최저점의 높이가 i인 경우를 관리합시다. 이때, 최저점의 높이가 낮아지더라도 거리가 가장 낮은 막대의 높이인 최저점에 의존하므로 문제가 없습니다.
- 점을 x좌표순으로 정리하여 처리하면서 지나가는 최저점의 높이가 막대의 높이보다 큰 경우를 전부 막대의 높이를 지나가는 경우로 바꾸어 해석하여 처리하면 된다는 것을 알 수 있습니다.
- 맵을 이용하여 관리하면 이 과정을 옮기는 점 하나마다 $O(\log N)$에 처리할 수 있습니다.
- 또한, 이 과정 이후 지나가는 최저점의 높이가 막대의 높이보다 큰 경우가 모두 사라지므로 N개의 막대에 대해 이 과정을 총 처리하는 시간복잡도가 $O(N \log N)$임을 알 수 있습니다.

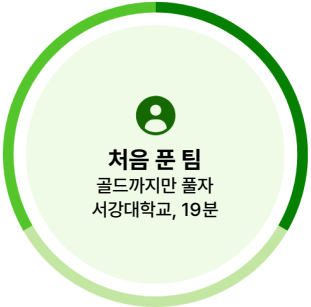
G. 히스토그램에서의 거리와 가장 가까운 점

- 위에서 언급한 부분에만 유의하면서 막대가 없는 일반적인 2차원 공간에서 두 점 (x_1, y_1), (x_2, y_2) 간의 거리가 $|x_1 - x_2| + |y_1 - y_2|$일 때 N개의 점 각각에 대해 가장 가까운 점과의 거리를 구하는 방식을 활용하면 $O((N+K+Q)\log(N+K+Q))$에 문제를 해결할 수 있습니다.
- 추가로, 에디토리얼에서는 다루지 않았지만 간선을 적절히 $O(N)$개 추가하여 멀티 소스 다익스트라 알고리즘을 활용하는 풀이나 카르테시안 트리를 이용한 풀이가 존재합니다.

H. 신촌 방수 계획

출제진 의도: Hard

#graphs #mst



H. 신촌 방수 계획

- 모든 건물로 이동할 수 있어야 한다는 조건과 우산 보관함, 통로의 설치 비용 조건이 MST 문제와 유사합니다.
- 우산 보관함을 설치하면 모든 건물에 갈 수 있으므로, 이를 그래프로 모델링하기 위해 모든 건물과 연결된 가상의 건물 하나를 생각합니다.
- 가상의 건물과 실제 건물 사이 도로 간선의 가중치는 해당하는 실제 건물에 우산 보관함을 짓는 데에 필요한 비용으로 잡겠습니다.
- 이러한 그래프에서 MST를 구하면 우산 보관함을 한 개 이상 설치할 때 필요한 최소 비용을 구할 수 있습니다.

H. 신촌 방수 계획

- 이때 가상의 건물은 다른 건물과 연결되지 않아도 됩니다.
- 즉, 우산 보관함을 전혀 설치하지 않을 때의 비용은 위에서 구한 값보다 더 작을 수 있습니다. (예제 2)
- 따라서 통로만 설치하는 경우로 MST를 한 번 더 구하고, 두 가지 경우 중 더 비용이 작을 때의 값을 정답으로 출력하면 됩니다.
- 시간복잡도는 크루스칼 알고리즘을 사용할 경우 $O(M \log M)$, 프림 알고리즘을 사용할 경우 $O(M \log N)$이 됩니다.

I. 별동대

출제진 의도: Challenging/Extreme

Easy: #cht #mitm / Hard: #convex_hull #greedy

I. 별동대

- Easy ver. ($N \leq 40$)

- 별동대의 전투력을 P라고 할 때, P을 다음과 같이 정리할 수 있습니다.
  $W = \sum S_i$입니다.
- $P = \sum_{i \in X} A_i + \sum_{i \in Y} B_i + \frac{1}{2}(\sum_{i \in X} S_i)^2 + \frac{1}{2}(\sum_{i \in Y} S_i)^2 - \frac{1}{2}\sum S_i^2$
  $= \sum_{i \in X}(A_i - B_i) + \frac{1}{2}\{(\sum_{i \in X} S_i)^2 + (W - \sum_{i \in X} S_i)^2\} - \frac{1}{2}\sum S_i^2 + \sum B_i$
  $= \sum_{i \in X}(A_i - B_i) + (\sum_{i \in X} S_i)^2 - W\sum_{i \in X} S_i + \frac{1}{2}W^2 - \frac{1}{2}\sum S_i^2 + \sum B_i$
- 위 식에서 $C = \frac{1}{2}W^2 - \frac{1}{2}\sum S_i^2 + \sum B_i$은 상수입니다.

I. 별동대

- 별동대원을 절반으로 분할합시다. X를 X_1과 X_2로 분할하였을 때, 다음과 같이 식을 전개할 수 있습니다.
- $a_p = \sum_{i \in X_p} (A_i - B_i)$, $s_p = \sum_{i \in X_p} S_i$
- $P = a_1 + a_2 + (s_1 + s_2)^2 - W(s_1 + s_2) + C$
  $= (a_1 + s_1^2 - Ws_1) + (a_2 + s_2^2 - Ws_2) + 2s_1 s_2 + C$

- 문제에서는 P의 최댓값과 최댓값이 되는 경우의 수를 구해야 합니다. 위 식은 직선 $y = 2s_1 x + (a_1 + s_1^2 - Ws_1)$을 Convex Hull Trick를 이용해 관리하고, $x = s_2$를 대입해 해결할 수 있습니다.

I. 별동대

- 경우의 수를 구하기 위해서는 동일한 직선을 병합해 개수를 관리하고, 직선에 교점의 개수를 저장하는 등의 방식으로 직선 간 교점을 잘 처리하면 됩니다.
- 시간복잡도는 $O(N \cdot 2^{\frac{N}{2}})$ 입니다.

- 경우의 수를 더 쉽게 구할 수 있는 방법도 존재합니다.
- 최댓값을 가질 수 있는 고유한 직선의 개수는 $N+1$개 이하입니다. (증명은 별동대 (Hard) 해설에 있습니다.) 동일한 직선만 병합하고 볼록껍질을 나이브하게 순회해도 됩니다.

I. 별동대

- Hard ver. ($N \leq 200\,000$)

- 위에서 정리한 식을 다시 봅시다.
- $P = \sum_{i \in X}(A_i - B_i) + (\sum_{i \in X} S_i)^2 - W\sum_{i \in X} S_i + \frac{1}{2}W^2 - \frac{1}{2}\sum S_i^2 + \sum B_i$

- 집합 X에 대해 $s_X = \sum_{i \in X} S_i$, $d_X = \sum_{i \in X}(A_i - B_i)$라고 하면,
  $f(s, d) = s^2 - Ws + d + C$가 최대가 되는 $f(s_X, d_X)$의 값과 개수를 구하는 문제가 됩니다.

I. 별동대

- 어떤 X가 f를 최대화할 수 있을까요?

- $f(s_X, d_X)$ 를 최대로 만드는 집합 X의 대응점 (s_X, d_X)은 $\{(s_V, d_V) \mid V \subset \{1, 2, \ldots, N\}\}$이 이루는 Upper hull의 꼭짓점 중 하나입니다.

I. 별동대

- 위 내용을 증명하기 위해 다음 보조정리를 이용합니다.

- Lemma. 서로 다른 세 점 A, B, P에 대해, P가 A와 B의 내분점이라면 $f(P) < \max(f(A), f(B))$이다.
- proof. $A(s_A, d_A)$, $B(s_B, d_B)$에 대해 $P(s_P, d_P)$를 $0 < t < 1$인 t에 대해 $((1-t)s_A + ts_B, (1-t)d_A + td_B)$로 표현할 수 있습니다.
  $(1-t)f(A) + tf(B) - f(P) = t(1-t)(s_A - s_B)^2$인데, 만약 $s_A \neq s_B$라면 위 식에 의해 $(1-t)f(A) + tf(B) > f(P)$이므로 $f(P) < \max(f(A), f(B))$이고, $s_A = s_B$라면 $s_A = s_B = s_P$이고 $d_P < \max(d_A, d_B)$이므로 $f(P) < \max(f(A), f(B))$입니다.

I. 별동대

- 볼록 껍질의 경계 위에 존재하는 꼭짓점이 아닌 모든 점은 Lemma에 의해 최댓값이 될 수 없습니다.
- Upper hull의 경계 위에 존재하지 않는 점 P는 그와 s가 동일하고 d가 큰 볼록 껍질의 경계 위의 점 A를 잡을 수 있습니다. 이 때 $f(P) < f(A)$이고, A는 볼록 껍질의 경계 위의 점이므로 Lemma에 의해 $f(A) \leq f(X)$인 존재하는 점 X를 찾을 수 있기에 최댓값이 될 수 없습니다.

- 그러므로, Upper hull의 꼭짓점만 고려하면 최댓값과 개수를 구할 수 있습니다.

I. 별동대

- 초기에 $x = y = 0$, V를 빈 배열으로 두고 모든 $(S_i, A_i - B_i)$ 쌍을 다음과 같이 전처리합니다.
- $S_i = 0$인 경우: y에 $\max(0, A_i - B_i)$를 더합니다.
- $S_i > 0$인 경우: V에 $(S_i, A_i - B_i)$를 추가합니다.
- $S_i < 0$인 경우: x에 S_i, y에 $A_i - B_i$를 더하고 V에 $(-S_i, B_i - A_i)$를 추가합니다.

- (x, y)는 Upper hull의 가장 좌측에 위치한 점이 되고, V의 원소 (x_i, y_i)를 $\frac{y_i}{x_i}$가 큰 순서대로 누적하면 Upper hull의 모든 꼭짓점을 찾을 수 있습니다.

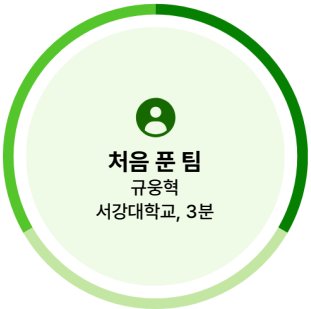
I. 별동대

- 이 때 $S_i = A_i - B_i = 0$인 점 $(S_i, A_i - B_i)$는 포함해도 좌표가 변하지 않기 때문에, $(0, 0)$의 개수 l에 대해 경우의 수는 최댓값이 되는 고유한 점의 개수에 2^l을 곱한 값이 됩니다.
- 역추적은 먼저 최댓값을 구한 후, 대원의 포함 상태를 관리하면서 최댓값과 동일한 전투력일 때를 구해주면 됩니다.
- V를 $O(N \log N)$에 정렬하여 구현하면 시간복잡도는 $O(N \log N)$입니다.

J. 지하철! 지하철! 몇호선? 몇호선?

출제진 의도: Medium

#ad_hoc



J. 지하철! 지하철! 몇호선? 몇호선?

- n이 짝수일 경우, 모든 역을 $(1,2),(3,4),\cdot(2k-1,2k),\cdot(n-1,n)$으로 두 개씩 묶을 수 있습니다.

  선공이 하나의 수를 부르면, 후공은 해당 수와 같은 쌍에 있는 수를 부릅니다.

  이후 선공은 해당 쌍의 좌우에 있는 쌍 중 하나의 수를 부르게 되는데,
  이때 처음 쌍의 좌우에는 무조건 짝수 개의 수가 존재하므로,
  더 작은 수에서의 같은 문제로 귀결되게 됩니다.

  귀납적으로, 수가 두 개 남았을 때 후공이 승리하므로 N이 짝수일 경우 반드시 후공이 승리합니다.

J. 지하철! 지하철! 몇호선? 몇호선?

- n이 홀수일 경우, 선공은 n을 불러 $n-1$이라는 짝수 개의 수를 남겨둘 수 있습니다.

  즉, n이 짝수인 경우에서 선공과 후공만 바뀐 채 게임을 진행할 수 있으므로, 반드시 선공이 승리합니다.

K. MC 히페리온

출제진 의도: Hard

#trie #euler tour technique #segment tree

K. MC 히페리온

- 핵심 관찰: 문자열 T가 S_i의 **접미사**라는 것은, T를 뒤집은 문자열 T^R이 S_i를 뒤집은 문자열 S_i^R의 **접두사**라는 것과 동치입니다.
- 따라서 모든 좋은 단어 $S_1, S_2, \ldots, S_N$을 뒤집은 후 **트라이**를 구성하면, 접미사 관계를 접두사 관계로 변환하여 다룰 수 있습니다.

K. MC 히페리온

- 트라이에서 어떤 노드 v는 특정 접두사 문자열에 대응됩니다. T^R에 대응되는 노드를 v_T라 하면, T를 접미사로 가지는 좋은 단어들은 정확히 v_T의 **서브트리** 내에 끝점이 존재하는 단어들입니다.
- 따라서 2번 쿼리는 v_T의 서브트리에 속하는 좋은 단어들의 가중치 합 $\times |T|$로 귀결됩니다.

K. MC 히페리온

- 트라이에 대해 **오일러 투어**를 수행하면, 각 노드 v의 서브트리가 오일러 투어 배열 위의 **연속 구간** $[\text{in}(v),\ \text{out}(v)]$에 대응됩니다.
- 각 좋은 단어 S_i^R의 끝점 노드에 가중치 W_i를 부여하고, 이 배열 위에 **세그먼트 트리**를 구성합니다.
- **1번 쿼리** (S_i의 가중치를 X로 변경): S_i^R의 끝점 노드의 오일러 투어 인덱스에서 세그먼트 트리의 점 업데이트를 수행합니다.
- **2번 쿼리** (문자열 T의 라임 파워): T^R에 대응되는 트라이 노드 v_T를 찾고, 구간 $[\text{in}(v_T),\ \text{out}(v_T)]$의 합을 세그먼트 트리로 구한 뒤 $|T|$를 곱합니다. 노드가 존재하지 않으면 답은 0입니다.

K. MC 히페리온

- 모든 문자열 길이의 총합을 $|S|$라 하면, 트라이의 노드 수는 최대 $O(|S|)$입니다.
- 트라이 구성 및 오일러 투어: $O(|S|)$
- 세그먼트 트리 구성: $O(|S|)$
- 1번 쿼리: $O(\log |S|)$
- 2번 쿼리: 트라이에서 노드를 찾는 데 $O(|T|)$, 구간 합 질의에 $O(\log |S|)$
- 전체 시간복잡도: $O(|S| + Q \log |S|)$

K. MC 히페리온

- **별해: 문자열 정렬**
- 트라이를 명시적으로 구성하지 않고도 동일한 결과를 얻을 수 있습니다.
- 모든 좋은 단어를 뒤집은 문자열 $S_1^R, S_2^R, \ldots, S_N^R$을 **사전순으로 정렬**합니다.
- T^R이 S_i^R의 접두사인 모든 i는 정렬된 배열에서 **연속 구간**을 이룹니다. 이는 사전순 정렬의 성질에 의해, 동일한 접두사를 공유하는 문자열들이 반드시 인접하기 때문입니다.
- 이분 탐색으로 해당 구간 $[l, r]$을 찾은 뒤, 세그먼트 트리로 구간 합을 구하면 됩니다.

K. MC 히페리온

- 단일 쿼리의 이분 탐색 비용은 $O(|T| \log N)$이지만, **모든 쿼리에 걸친 총 비용**은 문자열 길이 총합 제한에 의해 상한이 존재합니다.
- 이분 탐색 2회 $\times$ 각 $\log N$번 비교 $\times$ 비교당 최대 $|T|$문자이므로, 총 비용은 $2\sum_j |T_j| \cdot \log N$ 이하입니다.
- $\sum |S_i| + \sum |T_j| \le S$이므로 $\sum |T_j| \le S$이고, 따라서 총 비용 $\le 2S \log N$입니다.
- $S = 2\,000\,000$, $\log_2 N \le 17$을 대입하면 총 비용 $\le 6.8 \times 10^7$으로, $|T|$를 키우면 쿼리 수가 줄고, 쿼리 수를 키우면 $|T|$가 줄어 곱이 항상 S 이하로 묶이기에 해당 풀이 또한 성립합니다.

L. Designant.

출제진 의도: Extreme

#euler tour technique #greedy

L. Designant.

- 한글로 ~~공명~~을 읽는 방법은 ~~'디자인앤트'~~ '데지그넌트'. ~~지정자~~라는 뜻을 지닌다.
  - ~~문제명~~
  - ~~디자인개미~~
- 각 쿼리를 다음의 두 작업으로 나눌 수 있습니다.
  - 트리를 잘 쪼갠다.
  - 포레스트를 잘 붙인다.
- 두 번째 작업이 좀 더 쉽고 유명하니까 두 번째 작업을 먼저 풀어봅시다.
  - IOI 기출이나 solved.ac CLASS을 밀다 보면...

L. Designant.

- 트리를 잘 쪼갰다 치고 포레스트를 지름이 최소가 되도록 붙여 봅시다.
- 포레스트에 포함된 가장 긴 트리의 지름 3개를 각각 $a \geq b \geq c$로 두면 답은 $\max(a, \lceil a/2 \rceil + \lceil b/2 \rceil + 1, \lceil b/2 \rceil + \lceil c/2 \rceil + 2)$입니다.
  - 트리의 모든 정점에는 그 트리의 반지름 이상으로 떨어진 정점이 존재합니다.
    반지름보다 멀리 떨어진 정점이 존재하는 정점에 다른 트리를 연결하는 것보다
    딱 반지름만큼 떨어진 정점까지만 있는 정점에 다른 트리를 연결하는 것이 이득입니다.
  - 포레스트의 각 트리를 정점처럼 생각합니다. 한 트리에서 다른 트리로 최대한 빨리 이동할 수 있으려면 트리를 성게 모양으로 잇는 것이 유리합니다.
    즉 지름이 가장 긴 트리를 중심으로 다른 트리들을 잇는 모양이 됩니다.
  - 완성된 트리의 지름은 새로 만든 간선을 0개, 1개, 2개 이용하는 경로 중 가장 긴 것으로 결정되고, 이게 위 수식의 세 길이에 해당합니다.

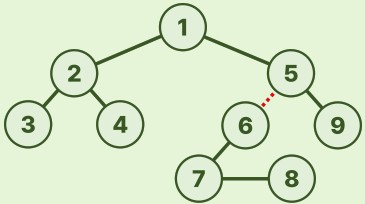
L. Designant.

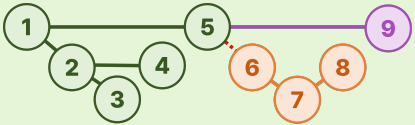
- DFS 등으로 쿼리마다 포레스트를 구성하는 각 트리의 지름을 계산하면 $\mathcal{O}(NQ)$의 시간 복잡도에 문제를 해결할 수 있습니다.
  $N, Q \leq 100\,000$에서 이 시간 복잡도를 통과시키기는 쉽지 않습니다.

- 트리를 더 빨리 쪼개고 각 트리의 지름을 계산할 수 있어야 합니다.

L. Designant.

- 이를 위해 트리의 정점에 DFS 순서대로 번호를 부여합니다.
- 이제 간선을 하나 날려서 트리를 둘로 쪼개버립시다.
- 그러면 DFS 번호를 기준으로 트리가 루트를 포함하지 않는 조각에 해당하는 구간과, 그 조각으로 인해 왼쪽과 오른쪽으로 나뉜 루트를 포함하는 조각에 해당하는 구간이 생겨 최대 3개의 구간이 생깁니다.





L. Designant.

- 이를 일반화하면 간선 하나를 지울 때마다 구간의 개수는 최대 2 늘어나므로, 각 쿼리에서 처리해야 하는 구간의 수는 최대 $2K+1$개임을 알 수 있습니다.

- 전체 쿼리에서 K의 합은 $300\,000$으로 제한되어 있습니다. 이를 활용합시다.

L. Designant.

- 각 구간은 트리의 DFS 순서 상 연속한 정점들의 집합을 표현합니다.

- 우리는 이 집합에 포함된 정점들 사이의 거리 중 최댓값을 관리하는 식으로 트리를 쪼개 얻은 각 서브트리의 지름을 구할 수 있습니다.
  - 우리는 최종적으로 같은 조각 안에 있는 정점들만 합칠 거라서, 당장 구간 안에 있는 정점들이 서로 연결되어 있지 않다는 걱정은 할 필요 없습니다. 그냥 트리의 모든 간선이 살아있는 상태에서의 거리만 계산해 주면 됩니다.

L. Designant.

- 쿼리에서 간선 제거를 서브트리에 해당하는 구간을 열고 닫는 이벤트로 바꿔 DFS 순서대로 정렬한 뒤, 현재 열려있는 구간을 스택으로 관리하는 식으로 간선이 제거됨에 따라 나뉘어진 구간과 해당하는 트리를 찾아줄 수 있습니다.

- 세그먼트 트리를 이용하면 $\mathcal{O}(K \lg N)$ 개의 구간을 합쳐보는 것으로 포레스트의 각 트리에 대한 정보를 얻을 수 있습니다.

L. Designant.

- 두 정점 집합을 빠르게 합치는 법을 생각해 봅시다.
  각 정점 집합 안에서의 최대 거리는 알고 있으니까, 두 정점 집합에서 정점 하나씩을 골랐을 때 얻을 수 있는 최대 거리만 빨리 알 수 있으면 됩니다.

- 다음의 보조정리를 이용합니다.

  트리의 정점 집합 S 안에서 두 정점 $u, v \in S$가 최대 거리를 갖는다고 합시다.
  그러면 **트리 위의** 임의의 정점 x와 S **위의** 임의의 정점 y에 대해
  $\mathrm{dist}(x, y) \leq \max(\mathrm{dist}(x, u), \mathrm{dist}(x, v))$**가 성립합니다.**
  - S가 하나의 연결 요소를 이룰 필요는 없습니다.

L. Designant.

- 증명은 다음과 같습니다.
  - 정점 y와 경로 $u-v$가 만나는 정점 z를 고려합니다.
    z가 S 위의 정점일 필요가 없음에 유의합시다.
  - 여기서 $\mathrm{dist}(u,v)=\mathrm{dist}(u,z)+\mathrm{dist}(z,v)$,
    $\mathrm{dist}(u,y)=\mathrm{dist}(u,z)+\mathrm{dist}(z,y)$,
    $\mathrm{dist}(y,v)=\mathrm{dist}(y,z)+\mathrm{dist}(z,v)$를 얻습니다.
  - 이때 u,v가 S 안에서 최대 거리를 갖는 정점이므로 $\mathrm{dist}(u,v)\geq\mathrm{dist}(u,y)$,
    $\mathrm{dist}(u,v)\geq\mathrm{dist}(y,v)$입니다.
  - 위의 식을 대입해 $\mathrm{dist}(u,z)\geq\mathrm{dist}(y,z)$, $\mathrm{dist}(v,z)\geq\mathrm{dist}(y,z)$을 얻을 수 있습니다.
  - 마찬가지로 정점 x와 경로 $u-v$가 만나는 정점 w를 고려합니다. WLOG $u-w-z-v$ 순으로 배치되어 있다고 하면 $x-z$는 $x-y$와 $x-v$에 모두 포함되어 있으므로,
    $\mathrm{dist}(x,y)\leq\mathrm{dist}(x,v)$임을 위에 보인 바로부터 확인할 수 있습니다.

L. Designant.

- 이제 두 정점 집합에서 최대 거리를 갖는 네 정점 $u_1 \in S_1, v_1 \in S_1, u_2 \in S_2, v_2 \in S_2$를 고려합니다.
- 임의의 두 정점 $x \in S_1$, $y \in S_2$를 잡으면, 보조정리에 의해 $\mathrm{dist}(x, y) \leq \max(\mathrm{dist}(x, u_2), \mathrm{dist}(x, v_2))$이 성립합니다.
- 여기서 u_2와 v_2를 S_1에 잇는 식으로 보조정리를 다시 사용하면 각 집합에서 정점을 하나씩 골랐을 때의 최대 거리는 각 집합에서 최대 거리를 갖는 네 정점을 잇는 경우만 확인해도 충분함을 알 수 있습니다.
- 따라서 각 정점 집합 안에서의 최대 거리까지 모두 고려했을 때, 두 정점 집합을 합치기 위해 확인해야 할 정점 쌍은 각 정점 집합의 지름의 양 끝점에 해당하는 네 정점 중 둘을 고르는 6가지뿐입니다.

L. Designant.

- 희소 배열을 이용해 두 정점 사이의 거리를 $\mathcal{O}(\lg N)$에 계산할 수 있습니다. 세그먼트 트리에 각 구간에 해당하는 정점 집합의 지름의 양 끝점을 저장하고, 구간을 합칠 때마다 6가지 경우에 대한 거리를 확인하는 식으로 $\mathcal{O}(\lg^2 N)$에 트리를 쪼개는 작업을 구현할 수 있습니다.
- 이 풀이의 총 시간 복잡도는 $\mathcal{O}(K \lg^2 N)$입니다. 상수가 꽤 크기 때문에 충분히 섬세하게 구현하지 않으면 시간 초과를 받을 수 있습니다.

- 이러고 풀이가 끝나면 아쉽겠죠??

L. Designant.

- LCA는 ETT 상에서의 RMQ로 환원됩니다. 희소 배열을 이용해 업데이트가 없는 RMQ를 쿼리 당 $\mathcal{O}(1)$에 처리할 수 있습니다. 이를 이용하면 시간 복잡도를 $\mathcal{O}((N+K)\lg N)$으로 줄일 수 있습니다.
- 사실 정점 집합의 지름을 묻는 쿼리 역시 일종의 RMQ로 볼 수 있습니다. 그래서 이쪽도 세그먼트 트리 대신 희소 배열로 전처리하면 $\mathcal{O}(1)$에 구간에 대한 정보를 얻어낼 수 있습니다. 이를 적용해도 역시 $\mathcal{O}((N+K)\lg N)$의 시간 복잡도가 얻어집니다.
  - 간선 제거 이벤트의 ETT 순서 기준 정렬 때문에 위의 두 최적화를 동시에 적용해도 시간 복잡도는 $\mathcal{O}((N+K)\lg N)$로 동일합니다. 단, 실행 속도는 2배 가량 차이를 보입니다. 정해는 300ms 이내에 작동합니다.

L. Designant.

- 빈칸을 복사해 보면 유제가 나올 수도 owo

M. 괄호 문자열 카드

출제진 의도: Easy

#greedy #case work

출제자

유상혁 (golazcc83)

제출 55번, 정답 18팀

정답률: 32.73%

처음 푼 팀

placeholder

연세대학교, 7분

M. 괄호 문자열 카드

- 여섯 종류의 괄호 문자열 카드 '(', '((', ')', '))', '()', ')('로 가장 긴 올바른 괄호 문자열을 만들어야 합니다.
- 올바른 괄호 문자열은 여는 괄호와 닫는 괄호의 개수가 같습니다.
- 괄호 문자열 카드 중 ')(', '()'는 올바른 괄호 문자열에 포함시켜도 여는 괄호와 닫는 괄호의 개수의 차를 바꾸지 않습니다.
- 따라서 '(', '((' 카드는 ')', '))' 카드와 매칭시켜야 합니다.

M. 괄호 문자열 카드

- '(', '((' 카드의 개수가 각각 A, B일 때 만들 수 있는 여는 괄호의 개수 V_X는 아래와 같습니다.

$$\begin{cases} 0 \le V_X \le A + 2B & \text{if } A > 0 \\ 0 \le V_X \le 2B, V_X\text{는 짝수} & \text{if } A = 0 \end{cases}$$

- $A = 0$일 때 V_X는 항상 짝수라는 점에 유의해야 합니다.
- ')', '))' 카드로 만들 수 있는 닫는 괄호의 개수 V_Y도 동일하게 계산하여 V_X, V_Y로 가능한 값 중 최댓값인 V를 찾습니다.

M. 괄호 문자열 카드

- 괄호 문자열 카드 ')('는 $V > 0$일 때 '(', '(('와 ')', '))' 카드 사이에 끼워넣는 방식으로 사용할 수 있습니다.
- 괄호 문자열 카드 '()'는 그 자체로 올바른 괄호 문자열이기 때문에, 기존 올바른 괄호 문자열 옆에 추가하는 방식으로 항상 사용할 수 있습니다.