목차

240

문제		의도한 난이도	출제자
A	튜터-튜티 관계의 수	Easy	박찬솔 chansol
B	선인장이 무럭무럭 자라고 있어요	Challenging	이국렬 lky7674
C	카카오뷰 큐레이팅 효용성 분석	Beginner	홍은기 pichulia
D	Y	Medium	조찬우 myyh1234
E	놀이기구에 진심인 편	Hard	이상원 gumgood
F	mod와 쿼리	Hard	김태영 tae826
G	도로 정보	Medium	박진식 pjshwa
H	신촌방위본부의 부대 배치	Hard	이국렬 lky7674
I	이 멋진 수열에 쿼리를!	Hard	홍은기 pichulia
J	일이 너무 많아...	Medium	홍은기 pichulia
K	올바른 괄호	Easy	박찬솔 chansol
L	팰린드롬 게임	Medium	김태영 tae826
M	불협화음	Challenging	홍은기 pichulia

A. 튜터-튜티 관계의 수

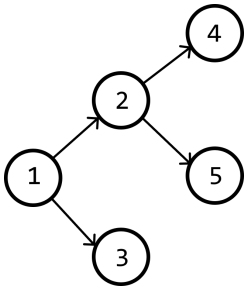
`graph_traversal`
출제진 의도 – **Easy**

- ✓ 제출 153번, 정답 34명 (정답률 22.222%)
- ✓ 처음 푼 팀: **어라랍스타**[연세대학교], 5분
- ✓ 출제자: 박찬솔[chansol]

A. 튜터-튜티 관계의 수

240

- ✓ 그래프에 하나의 트리만 있다고 가정하고, 최초로 교육 자료를 받는 사람을 1 번이라고 합시다.
- ✓ 1 번에서 시작하는 그래프 탐색(DFS, BFS)을 수행합시다. 트리에 속한 모든 사람에게 교육 자료를 전달할 수 있고, 모든 튜터-튜티 관계를 정할 수 있습니다.



A. 튜터-튜티 관계의 수

240

- 하나의 트리에는 최초로 교육 자료를 받는 사람이 한 명이면 충분합니다.
- 최초로 교육 자료를 받는 사람을 정했을 때, 탐색 과정을 따라서 튜터-튜티 관계가 정해지므로 경우의 수는 하나 밖에 없습니다.
- 아무 사람에서 그래프 탐색을 시작하더라도 트리의 모든 사람에게 교육 자료를 전달할 수 있으므로, 하나의 트리에서 나올 수 있는 경우의 수는 트리에 속한 사람의 수와 같습니다.
- 하나의 트리에서 최초로 교육 자료를 받는 사람을 정하는 것은 독립적입니다. 따라서, 각 트리에 속한 사람의 수의 곱이 답입니다.
- DFS, BFS, Union-Find 등을 사용하면 각 트리에 속한 사람의 수를 구할 수 있습니다.

240

B. 선인장이 무럭무럭 자라고 있어요

`cactus`, `bcc`, `pbs`
출제진 의도 – **Challenging**

- ✓ 제출 2번, 정답 0명 (정답률 0%)
- ✓ 처음 푼 사람: **???**, ??분
- ✓ 출제자: 이국렬 `lky7674`

B. 선인장이 무럭무럭 자라고 있어요

240

- 꽃의 개수는 시간이 지나도 줄어들지 않기에 X 일에 특정 색깔의 꽃을 구매할 수 있다면, $X+1$ 일 이후에도 구매할 수 있습니다.
- 이에 따라서 각 색깔의 꽃을 언제부터 구매할 수 있는지를 쿼리에 대한 이분 탐색으로 구할 수 있습니다.
- 구매 가능한 날이 가장 늦은 꽃을 Q 번째 날, 그 다음으로 늦은 꽃을 $(Q-1)$ 번째 날, 이렇게 순차적으로 구매 일을 지정하면 됩니다.

- ✓ 모든 색깔에 대해서 구매 가능한 날을 각각 이분 탐색을 통해서 구하면 시간이 매우 오래 걸립니다.
- ✓ 각 색깔의 꽃에 대한 이분 탐색은 동일한 쿼리를 대상으로 실행합니다.
- ✓ 따라서 동일한 쿼리에 대한 이분 탐색을 한꺼번에 처리하는 Parallel Binary Search로 접근할 수 있습니다.

B. 선인장이 무럭무럭 자라고 있어요

240

- ✓ 이제 $i = 1, \cdots, C$에 대해서 i번째 색깔의 꽃을 m_i번째 날에 구입할 수 있는지를 판별해야 합니다.
- ✓ m_i번째 쿼리까지 처리했을 때, 각각의 i번째 색깔의 정점을 포함하는 BCC에 더해진 값의 총합이 c_i 이상인지를 판별하면 됩니다.
- ✓ 각 i별로 s_i, e_i를 지정하고 중간값인 m_i 값을 넣는 동적 배열(ex. vector)를 준비합니다.
- ✓ 그리고 주어진 쿼리를 순서대로 시행하면서 m_i번째 쿼리가 끝났을 때, 위의 내용을 판별하면 됩니다.

- 현재 쿼리까지 시행했을 때, 해당 BCC에 얼마가 더해졌는지에 대한 배열을 만들어줍니다.
- 이제 $color(j) = i$인 정점 j에 대해서 J가 속한 BCC에 더해진 값을 모두 더해주면 됩니다.
- $\sum_{i=1}^{c} \sum_{color(j)=i}$ (정점 j를 포함하는 BCC의 개수) $= \sum_{i=1}^{n}$ (i번 정점을 포함하는 BCC의 개수) $= O(N+M)$ 입니다. 따라서 위와 같은 방법을 통해서 선형 시간에 판별할 수 있습니다.
- 총 시간 복잡도 : $O((N+M+Q)\log Q)$.
- Lazy Propagation이 있는 Segment Tree를 통해서 $O((N+M+Q)\log^2 Q)$에 답을 구할 수 있지만 시간초과가 발생합니다. (대략 2.3초정도) 참고 부탁드립니다.

C. 카카오뷰 큐레이팅 효용성 분석

`implementation`
출제진 의도 – **Beginner**

- ✓ 제출 67번, 정답 63명 (정답률 95.522%)
- ✓ 처음 푼 팀: **so강은 so 강하다**서강대학교, 2분
- ✓ 출제자: 홍은기`pichulia`

- 첫 번째 정답은 $\sum A_i$ 으로 계산됩니다.
- 두 번째 정답은 $\sum A_i * (1 - B_i)$ 으로 계산됩니다.
- 입력으로 들어온 숫자를 배열에 저장할 수 있는지 여부를 물어본 문제입니다.

D. Y

DFS

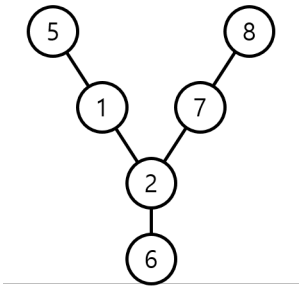
출제진 의도 – **Medium**

- ✓ 제출 30번, 정답 8명 (정답률 26.667%)
- ✓ 처음 푼 팀: **서강베스트**서강대학교, 98분
- ✓ 출제자: 조찬우myyh1234

D. Y

240

- 문제의 조건을 만족하려면 Y-트리는 아래 그림처럼 3개의 정점과 인접한 유일한 정점을 중심으로 정확히 세 개의 가지가 뻗어나가는 형태의 트리여야 합니다.
- 편의상 Y-트리에서 중심이 되어 가지들이 뻗어나가는 정점을 **중심**이라고 부르겠습니다.



D. Y

240

- 주어진 트리에서 정점을 적당히 삭제하여 임의의 정점 v_i 가 중심인 Y-트리를 만드는 것은 v_i 와 인접한 모든 정점 중 가장 길게 뻗어나갈 수 있는 방향을 정확히 세 개 고르는 것과 같습니다.

- 따라서 v_i 에서 가장 길게 뻗어나갈 수 있는 가지 세 개의 길이를 모두 안다면 v_i 가 중심인 가장 큰 Y-트리의 크기를 알 수 있습니다.

- 나아가 트리의 모든 정점마다 해당 정점에서 뻗어나가는 가장 긴 가지 세 개의 길이를 안다면 주어진 트리에서 만들 수 있는 가장 큰 Y-트리의 크기를 구할 수 있습니다.

D. Y

240

- 가장 긴 가지 세 개의 정보를 얻기 위해 임의의 정점을 루트로 설정하고 두 번의 DFS를 수행합니다.

- 첫 번째 DFS에서는 현재 정점의 자식 정점으로 향하는 가지의 길이를 모두 알아냅니다.
- 이후 두 번째 DFS를 통해 부모 정점으로 향하는 가지까지 고려합니다. 부모 정점의 가장 긴 가지가 현재 정점으로 향한다면 두 번째로 긴 가지를, 아니라면 가장 긴 가지를 사용하는 것이 최적입니다.

- 이러한 과정을 통해 모든 정점에 대해 인접한 정점으로 뻗어나가는 모든 가지의 길이를 알 수 있으며, 정답을 구할 수 있습니다.

D. Y

240

- 첫 번째 DFS에서 모든 가지의 길이를 저장한다면 시간복잡도는 $O(N \log N)$, 긴 가지 세 개의 길이만 저장한다면 $O(N)$ 입니다.
- 트리의 지름을 구하고 지름에서 가장 멀리 떨어진 정점과 지름을 이어 Y-트리를 만드는 풀이도 있습니다. 이 풀이의 시간복잡도는 자명히 $O(N)$ 이며, 증명은 여러분의 몫으로 남기겠습니다.

240

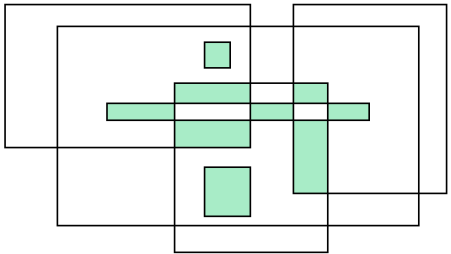
E. 놀이기구에 진심인 편

`sqrt decomposition`, `sweeping`
출제진 의도 – **Hard**

- ✓ 제출 13번, 정답 0명 (정답률 0%)
- ✓ 처음 푼 사람: **???**, ??분
- ✓ 출제자: 이상원`gumgood`

E. 놀이기구에 진심인 편

240



- 키와 몸무게를 2차원 좌표평면 위의 x 축, y 축으로 나타냅시다.
- 각 놀이기구를 탈 수 있는 키와 몸무게는 좌표평면 위 직사각형으로 나타낼 수 있습니다.
- 이제 $x \in [H-D, H+D], y \in [W-D, W+D]$ 인 범위 내에서 직사각형이 K 개 이상 겹쳐있는 격자점을 모두 세는 문제가 되었습니다.

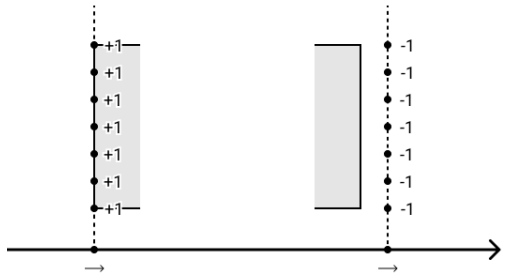
240

먼저, $K = 1$인 경우에 대해서 해결해봅시다.

- ✓ 이 경우는 "BOJ2185 위성지도", "BOJ9318 직사각형의 합집합"과 같이 이미 잘 알려진 형태의 문제입니다.
- ✓ x 축을 1 부터 $100\,000$까지 **Sweeping**하면서, 각 x 좌표마다 직사각형으로 덮힌 점이 몇 개 있는지 세어 봅시다.
- ✓ 직사각형의 분포에 따라 세어야 하는 점들이 변하게 됩니다. 이를 **Segment tree**로 관리할 수 있습니다.

240

- ✓ Segment tree의 각 원소에는 현재 보고있는 x 좌표에서 각 y 좌표마다 몇 개의 직사각형이 겹쳐있는지 저장합니다.
- ✓ Sweeping 중 직사각형의 왼쪽 변을 만난 경우, $[w_{lo}, w_{hi}]$ 구간에 $+1$을 더합니다.
- ✓ 반대로 직사각형의 오른쪽 변을 만난 경우, $[w_{lo}, w_{hi}]$ 구간에 -1을 더합니다.



240

- ✓ 이렇게 각 x 좌표에서 업데이트가 끝날 때마다 1 이상의 값을 가지는 원소의 개수를 세면 됩니다.
- ✓ 정리하면 다음 두 쿼리를 처리할 수 있는 자료구조인 Segment tree를 이용하여 Sweeping 알고리즘으로 해결할 수 있습니다.
  - 구간 $[L, R]$에 값 v를 더한다.
  - 구간 $[L, R]$에 1 이상의 값을 가지는 원소의 개수를 센다.

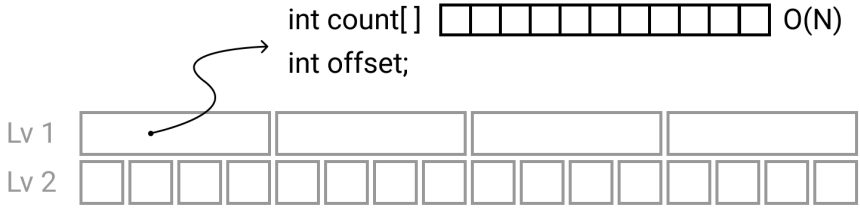
240

이제, $K \geq 1$ 인 경우에 대해서 해결해보겠습니다.

- ✓ 동일하게 접근하면, 다음 쿼리들을 처리할 수 있는 자료구조를 이용하여 Sweeping 알고리즘으로 해결할 수 있습니다.
  - 구간 $[L, R]$ 에 값 v 를 더한다.
  - 구간 $[L, R]$ 에 K 이상의 값을 가지는 원소의 개수를 센다.
- ✓ 이를 Segment tree로 해결하는 것은 쉽지 않습니다.
- ✓ 대신 **Sqrt Decomposition**을 이용하여 해결할 수 있습니다.

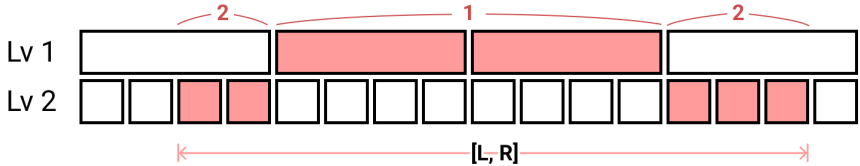
- ✓ 전체 배열을 $O(\sqrt{N})$ 으로 나눈 것을 Lv1 버킷이라 하겠습니다. Lv1 버킷에는 해당 구간에서 값이 K 이상인 원소 개수가 저장됩니다.
- ✓ 전체 배열을 $O(\sqrt{N})$ 으로 나눈 것을 Lv2 버킷이라 하겠습니다. Lv2 버킷은 관리하려는 기존 배열과 동일합니다.

240



- ✓ 쿼리를 위해서 한 가지 자료구조를 추가해야 합니다.
- ✓ Lv1의 각 버킷마다 크기가 $O(N)$ 인 $count$ 배열과 $offset$ 을 할당해줍니다.
- ✓ $count[i]$ 가 x 라는 것은 값이 $i + offset$ 인 원소가 x 개 있다는 뜻입니다.

240

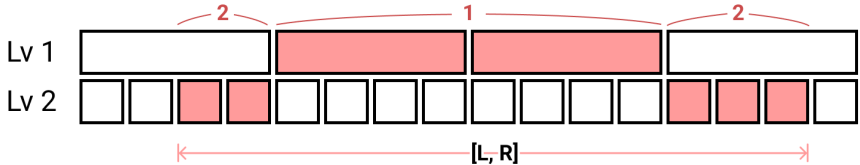


- ✓ 먼저 업데이트 쿼리를 해결해봅시다.
- ✓ 어떤 구간에 -1 또는 1을 더하는 경우, 업데이트할 버킷은 다음 두 경우가 있습니다.
  1. 구간에 완전히 포함되는 Lv1 버킷 $O(\sqrt{N})$ 개
  2. 구간에 걸쳐있는 Lv1 버킷 최대 2개

240

- ✓ 1번의 경우, $offset$을 업데이트하고, $count[k - offset]$을 Lv1 버킷에 더하거나 빼면 됩니다. 여기에 $O(1)$이 걸리고, 그러한 버킷이 $O(\sqrt{N})$개 있으므로 $O(\sqrt{N})$에 해결가능합니다.
- ✓ 2번의 경우, Lv2 버킷을 직접 업데이트 하고, Lv1 버킷을 다시 만들면 됩니다.
  - 이때 Lv1 버킷에 할당된 크기 $O(N)$인 $count$ 배열을 초기화해야 합니다.
  - 이 구간의 Lv2 버킷을 참조하여 $count$ 배열을 cancel하면 $O(\sqrt{N})$에 초기화할 수 있습니다.
  - 그 다음에 Lv1 버킷과 $count$ 배열을 다시 만드는 것까지 $O(\sqrt{N})$에 해결할 수 있습니다.
- ✓ 따라서 업데이트 쿼리를 $O(\sqrt{N})$에 해결할 수 있습니다.

240



- ✓ 이제 어떤 구간에 K 이상 원소의 개수를 묻는 쿼리를 해결해봅시다.
- ✓ 마찬가지로 다음 두 경우를 처리하면 됩니다.
  1. 구간에 완전히 포함되는 Lv1 버킷 $O(\sqrt{N})$ 개
  2. 구간에 걸쳐있는 Lv1 버킷 최대 2개

- ✓ 1번의 경우, Lv1 버킷의 값을 그대로 읽으면 되므로 $O(1)$ 이 걸립니다. 그러한 버킷이 $O(\sqrt{N})$ 개가 있기 때문에 총 $O(\sqrt{N})$ 이 걸립니다.
- ✓ 2번의 경우, 구간에 해당하는 Lv2 버킷을 직접 보면서 세면 되므로 $O(\sqrt{N})$ 이 걸립니다. 그러한 버킷이 최대 2 개 있으므로 총 $O(\sqrt{N})$ 이 걸립니다.
- ✓ 따라서 쿼리당 $O(\sqrt{N})$ 에 해결할 수 있습니다.

240

- ✓ $O(N)$ 개의 x 좌표에 대해 sweeping하면서 $O(N)$ 번의 쿼리를 처리하면 되기 때문에 시간복잡도 $O(N\sqrt{N})$ 에 문제를 해결할 수 있습니다.
- ✓ $O(\sqrt{N})$ 개의 Lv1 버킷마다 크기 $O(N)$ 의 $count$ 배열을 할당해줬으므로 공간복잡도는 O($N\sqrt{N}$) 입니다.

F. mod와 쿼리

`segment tree`, `number theory`
출제진 의도 – **Hard**

- ✓ 제출 44번, 정답 0명 (정답률 0%)
- ✓ 처음 푼 사람: **???**, ??분
- ✓ 출제자: 김태영 `tae826`

240

- ✓ $A \bmod B$ 를 어떻게 처리할 지 우선 생각해봅시다.
- ✓ $A \bmod B$ 는 $A - B \times \lfloor \frac{A}{B} \rfloor$ 로 바꿔서 계산해야합니다.
- ✓ 편의상 $\lfloor \frac{A}{B} \rfloor$ 를 $\frac{A}{B}$ 로 표현하겠습니다

240

- ✓ mod 에 대한 처리를 해결했으니 1번 쿼리부터 해봅시다.
- ✓ $\sum_{i=1}^{N} A_i \bmod X$ 를 $\sum_{i=1}^{N} A_i - X \times \frac{A_i}{X}$ 로 바꿔서 쓸 수 있습니다.
- ✓ $\sum_{i=1}^{N} A_i$ 는 미리 계산해놓을 수 있습니다.
- ✓ 결국 $\sum_{i=1}^{N} X \times \frac{A_i}{X}$ 를 계산하는 문제로 바뀌게 됩니다.

240

- ✓ $\sum_{i=1}^{N} X \times \frac{A_i}{X}$ 식에서 X 또는 $\frac{A_i}{X}$ 중 하나는 항상 $\sqrt{A_i}$ 보다 작거나 같습니다
- ✓ 만약 입력된 X 가 $\sqrt{A_i}$ 보다 작은 경우는 1부터 $\sqrt{A_i}$ 까지 모든 숫자에 대해 1번 쿼리의 결과를 저장해놓을 수 있습니다.
- ✓ 첫 입력때 X 가 1부터 $\sqrt{A_i}$ 일 때의 1번 쿼리 값을 저장해놓고, 3번 쿼리로 A_i 를 갱신할 때도 값을 일일히 바꿔주면 됩니다.

240

- X가 $\sqrt{A_i}$보다 클 경우엔 $\frac{A_i}{X}$가 항상 $\sqrt{A_i}$보다 작습니다.
- $\frac{A_i}{X}$가 1인 경우는 A_i가 X와 크거나 같고 $2X$보다 작은 경우입니다.
- $\frac{A_i}{X}$에 따른 범위를 구해주고, 그 범위에 실제 A_i가 몇 개 있는지 세주면 됩니다.
- 특정 구간에서의 A_i의 개수는 Segment Tree나 Fenwick Tree를 사용하여 구해줄 수 있습니다.

240

- 2번 쿼리도 비슷한 아이디어를 사용합니다.
- $\sum_{i=1}^{N} X \bmod A_i$ 를 $\sum_{i=1}^{N} X - A_i \times \frac{X}{A_i}$ 로 바꿉니다.
- 위 식의 앞 부분은 XN 이므로, 뒷 부분인 $\sum_{i=1}^{N} A_i \times \frac{X}{A_i}$ 만 계산해줍시다.
- 마찬가지로 A_i 가 $\sqrt{X}$ 보다 작거나 같을 경우와 $\frac{X}{A_i}$ 가 $\sqrt{X}$ 보다 작거나 같을 경우로 나눕니다.

240

- ✓ A_i 가 $\sqrt{X}$ 보다 작은 경우는 단순히 개수를 세어 계산해주면 됩니다.
- ✓ $\frac{X}{A_i}$ 가 $\sqrt{X}$ 보다 작은 경우도 1번 쿼리의 경우와 같이 가능한 A_i 의 범위를 나눠 해당 범위의 A_i 들의 합과 개수를 Fenwick Tree나 Segment Tree로 구해주면 됩니다.

- 3번 쿼리는 위의 1, 2번 쿼리에서 썼던 변수나 배열을 update 해주면 됩니다.
- 시간복잡도는 $O(Q\sqrt{X}\log N)$ 이 됩니다.
  - 이는 약 5억 정도의 계산량을 가집니다.
- $\sqrt{100000}$ 정도를 상수로 잡고 나눠서 쿼리를 처리해도 되는데, 그 쿼리에서 한 쪽 연산은 $\log$ 를 사용하고 한 쪽은 $\log$ 를 사용하지 않으므로, 상수를 적당히 조절하여 시간을 줄일 수 있습니다.
- 계산량이 상당히 많으므로 최적화를 진행하지 않은 Segment Tree로 풀이를 진행할 시 TLE를 받을 수 있습니다. Fenwick이나 위에서 언급한 상수 조절을 통해 시간을 단축해야 풀 수 있습니다.

G. 도로 정보

`prefix sum`
출제진 의도 – **Medium**

- ✓ 제출 67번, 정답 16명 (정답률 23.881%)
- ✓ 처음 푼 팀: **so강은 so 강하다**서강대학교, 7분
- ✓ 출제자: 박진식pjshwa

- 길이 N 의 문자열을 순회하면서, 문자열의 처음부터 현재 인덱스 i 까지 각각 T, G, F, P 가 몇 번씩 등장했는지를 기록하고, 이 값을 각각 $t(i), g(i), f(i), p(i)$ 라고 합시다.
- 그리고 문자열 인덱스 i 에 대해 상태값 $state(i)$ 를 정의합니다.

$$state(i) = \begin{cases} 27 \times (t(i)\%3) + 9 \times (g(i)\%3) + 3 \times (f(i)\%3) + (p(i)\%3), & \text{if } 1 \leq i \leq N \\ 0, & \text{if } i = 0 \end{cases}$$

240

- ✓ 그렇다면, 어떠한 도로 구간을 표현하는 부분 문자열의 시작 인덱스가 i, 끝 인덱스가 j 일 경우, $state(i-1) = state(j)$ 라면 해당 도로 구간은 **흥미로운 구간**이 됩니다.
- ✓ 한 번 등장했던 state 값이 다시 등장하려면 그 사이에 T, G, F, P 가 등장하는 횟수가 모두 3의 배수여야 하기 때문입니다.
- ✓ state 가 될 수 있는 값 81가지에 대해서, 문자열을 순회하며 인덱스 i 에 대해 i 이전까지 몇 번 등장했는지를 기록한다면, 위에 언급된 성질을 이용하여 i 로 끝나는 **흥미로운 구간**의 개수를 빠르게 셀 수 있습니다.
- ✓ 시간복잡도는 $O(N)$ 입니다.

H. 신촌방위본부의 부대 배치

`flow`, `bipartite_matching`
출제진 의도 – **Hard**

- ✓ 제출 10번, 정답 0명 (정답률 0%)
- ✓ 처음 푼 사람: **???**, ??분
- ✓ 출제자: 이국렬 `lky7674`

H. 신촌방위본부의 부대 배치

240

홀	짝	홀	짝	홀	짝	홀
짝	홀	짝	홀	짝	홀	짝
홀	짝	홀	짝	홀	짝	홀
짝	홀	짝		짝	홀	짝
홀	짝	홀	짝	홀	짝	홀
짝	홀	짝	홀	짝	홀	짝
홀	짝	홀	짝	홀	짝	홀

- ✓ 각 i 행 j 열에 대해서 $(i+j)$ 가 홀수, 짝수인지를 구분합니다.
- ✓ 그렇다면 코끼리가 홀수 번째 칸에 위치할 때 짝수 번째 칸만 공격하고, 반대로 짝수 번째 칸에 위치할 때 홀수 번째 칸만 공격하게 됩니다.

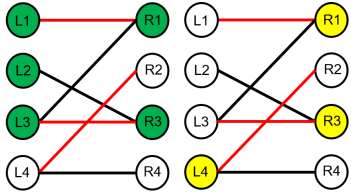
H. 신촌방위본부의 부대 배치

240

- 각 칸을 정점으로 보도록 하겠습니다. 병사가 공격할 수 있는 칸에 코끼리를 배치할 수 없으니 무시합니다.
- 코끼리를 배치했을 때, 서로 공격할 수 있는 칸끼리 간선으로 연결합니다.
- 코끼리가 다른 기우성을 가진 칸만 공격할 수 있기에, 이렇게 만들어진 그래프는 홀수칸과 짝수칸으로 나눠진 이분 그래프로 표현할 수 있습니다.
- 이렇게 만들어진 이분 그래프에서 최대 independent set을 구해야 합니다.
- independent set은 vertex cover의 여집합이기에 vertex cover를 구할 수 있으면 independent set도 구할 수 있습니다.
- 이분 그래프의 vertex cover는 Kőnig's Theorem에 의해서 이분 매칭으로 풀 수 있습니다.

H. 신촌방위본부의 부대 배치

240



- ✓ Kőnig's Theorem : 이분 그래프에서 (maximum matching의 크기) = (minimum vertex cover의 크기)
  - M^* : maximum matching
  - X : 매칭에 속하지 않은 왼쪽 정점과 그 정점으로부터 Alternating Path를 통해서 갈 수 있는 정점의 집합
  - $C^* = (L - X) \cup (R \cap X)$: minimum vertex cover

- ✓ 정점 개수 $O(N^2)$, 간선 개수 $O(N^2)$ 이기에 Hopcroft-Karp 알고리즘을 사용하면 $O(N^3)$ 에 이 문제를 해결할 수 있습니다.
  - Hopcroft-Karp 알고리즘 시간 복잡도 : $O(|E|\sqrt{(|V|)})$
- ✓ 주의할 점 : Ford-Fulkerson 알고리즘은 시간 초과가 발생합니다.
  - Ford-Fulkerson 알고리즘 시간 복잡도 : $O(|V||E|) = O(N^4)$

I. 이 멋진 수열에 쿼리를!

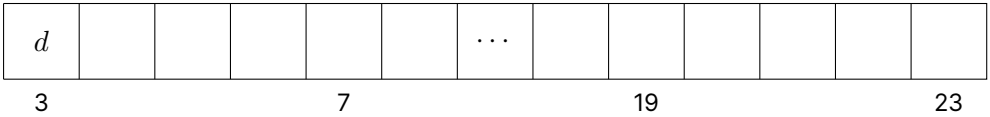
`segment tree`, `linear algebra`
출제진 의도 – **Hard**

- ✓ 제출 1번, 정답 0명 (정답률 0%)
- ✓ 처음 푼 사람: **???**, ??분
- ✓ 출제자: 홍은기`pichulia`

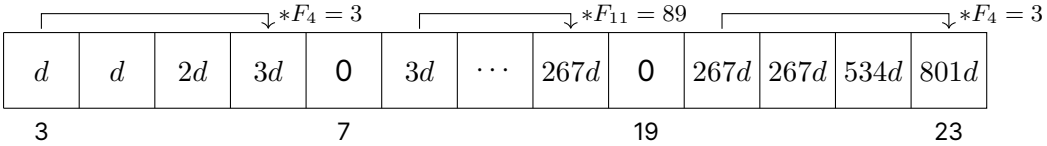
- 우선 아래와 같이 생긴 행렬 F 를 이용해서 n 번째 피보나치 수를 $\mathcal{O}(\log n)$ 만에 계산하는 사전지식이 필요합니다.

- $$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} * \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n * \begin{pmatrix} 1 \\ 0 \end{pmatrix} = F^n * \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

- 쉬운 풀이가 있고 어려운 풀이가 있습니다.
- 우선은 어려운 풀이부터 설명해보겠습니다.



- 위 그림은 설명을 위해 적당히 $N = 23$ 이고 3, 7, 19 번째 수열에 쿼리가 들어온 상황을 나타낸 그림입니다.
- 다른 상황은 전부 무시하고, 3 번 수열 구간에 값이 d 만큼의 변화가 있었다고 가정해봅시다.
- 이 d 만큼의 값의 변화가 최종 N 번째 수열 구간까지 어떻게 전달되는지를 고민해보면 다음과 같은 과정을 거치는 것을 확인할 수 있습니다.



- 6번째 수열 구간까지 F_4 만큼 값이 곱해지고, 6번째 값이 8번째 값으로 그대로 복사됩니다.
- 8번째 값에서 F_{11} 만큼 값이 곱해진 변화량이 18번째 값이 됩니다.
- 20번째 값에서 F_4 만큼 값이 곱해진 변화량이 23번째 수열 구간의 값의 변화량이 됩니다.
- 최종적으로 3번째 수열 구간에 값이 d 만큼 변하면, 그 변화량은 N 번째 수열 구간에 $d * F_{7-3} * F_{19-7-1} * F_{23-19}$ 만큼 영향을 끼칩니다.

240

- 이러한 규칙성을 잘 정리해봅시다.
- 지금까지 쿼리 마법이 날아온 위치 A_i 를 정렬해서 생각했을 때, N 번째 수열 구간의 값은 아래 수식의 합으로 표현이 가능해집니다.
- 수식을 좀 더 깔끔하게 만들기 위해서 더미로 $A_0 = 1$ 인 위치에 $B_0 = 1$ 인 쿼리 마법이 날아왔고, $A_Q = N + 1$ 이라고 가정하겠습니다.
- $$F_N = \sum_{i=0}^{Q-1} \left(B_i * F_{A_{i+1}-A_i} * \prod_{j=i+1}^{Q-1} \left(F_{A_{j+1}-A_j-1} \right) \right)$$

I. 이 멋진 수열에 쿼리를!

$$
\begin{aligned}
F_N = \quad & B_0 * F_{A_1-A_0} * F_{A_2-A_1-1} * F_{A_3-A_2-1} * F_{A_4-A_3-1} * F_{A_5-A_4-1} \\
+ \quad & B_1 * F_{A_2-A_1} * F_{A_3-A_2-1} * F_{A_4-A_3-1} * F_{A_5-A_4-1} \\
+ \quad & B_2 * F_{A_3-A_2} * F_{A_4-A_3-1} * F_{A_5-A_4-1} \\
+ \quad & B_3 * F_{A_4-A_3} * F_{A_5-A_4-1} \\
+ \quad & B_4 * F_{A_5-A_4}
\end{aligned}
$$

- ✓ $Q = 5$ 일 때 수식을 풀어서 쓰면 대략 다음과 같은 모습이 됩니다.
- ✓ 뭔가 중복되는 수식들을 하나로 묶고 싶게 생겼습니다.
- ✓ 우리는 여기서 $Q + 1$ 개의 노드를 가진 Binary Search Tree 구조를 활용해서 중복되는 식들을 깔끔하게 묶을 것입니다.

- ✓ 각 BST 노드는 다음과 같은 정보를 가지고 있도록 합니다.
- ✓ a : 쿼리 마법을 받은 위치 정보입니다. 이 값이 BST 의 key 가 됩니다.
- ✓ b : 쿼리 마법의 기본적인 정보입니다.
- ✓ min_a, max_a : **현재 노드를 root 로 하는 sub tree** 에서 가장 작은 a 값과 가장 큰 a 값을 각각 가지고 있습니다. 이는 나중에 부모 노드의 값을 갱신할 때 사용됩니다.
- ✓ max_b : **현재 노드를 root 로 하는 sub tree** 에서 가장 큰은 a 값을 가진 노드에 대응되는 b 값을 가지고 있습니다 이는 나중에 부모 노드의 값을 갱신할 때 사용됩니다.
- ✓ S : **현재 노드를 root 로 하는 sub tree** 만을 이용해서 계산한, F_N 값에 영향을 주는 수식 값의 합입니다. BST 루트 노드의 S 값이 곧 구하고자 하는 정답이 됩니다.
- ✓ F : **현재 노드를 root 로 하는 sub tree** 만을 이용해서 계산한, 피보나치 수열의 곱을 나타냅니다.

- ✓ 자세한 설명은 생략하고, 결론적으로 각 값의 갱신은 다음과 같은 과정으로 이루어집니다.
- ✓ L 을 left child node, R 을 right child node 라고 하면, 현재 노드 의 S 값과 F 값은 다음과 같은 계산을 통해 갱신됩니다.

$$S = R.S + L.S * R.F * F_{R.min\_a-a-1} * F_{a-L.max\_a-1}$$

- ✓ $$+ b * R.F * F_{R.min\_a-a}$$
$$+ L.max\_b * R.F * F_{R.min\_a-a-1} * F_{a-L.max\_a}$$

- ✓ $$F = L.F * R.F * F_{R.min\_a-a-1} * F_{a-L.max\_a-1}$$

- ✓ 즉, 두 자식 노드 만으로 현재 노드 값을 $\mathcal{O}(\log N)$ 만에 갱신할 수 있습니다.
- ✓ 말단노드 및 자식 노드가 하나만 있는 경우는 **적절히** 예외처리를 해줍시다.

- ✓ 쿼리를 하나 새로 추가하는 것은 곧 노드를 하나 추가하거나, 기존 노드를 찾아서 B_i 값을 변경하는 것과 같습니다.
- ✓ Binary Search Tree 에서 노드를 탐색 및 추가하는데 $\mathcal{O}(\log Q)$ 의 시간복잡도가 보장되도록 하기 위해서 Splay Tree 와 같은 Balanced Binary Search Tree 를 구현해야 합니다.
- ✓ 각 쿼리마다 $\mathcal{O}(\log Q)$ 개의 노드들이 갱신되며, 각 노드 갱신에 $\mathcal{O}(\log N)$ 이 소모될 것입니다.
- ✓ 전체 시간복잡도는 $\mathcal{O}(Q \log Q \log N)$ 이 됩니다.
- ✓ 계산 과정에서 등장하는 피보나치 수의 가지수가 $\mathcal{O}(Q)$ 가지를 넘지 않는다는 성질을 이용해 피보나치 수 계산 결과를 caching 하여 $\mathcal{O}\left(Q(\log^2 Q + \log N)\right)$, 혹은 $\mathcal{O}(Q(\log Q * (Hashing) + \log N))$ 으로 줄일 수 있습니다.
- ✓ BBST 구현 방식에 따라서 caching을 하지 않으면 시간초과를 받을 수도 있습니다.

- 쉬운 풀이는, 피보나치 계산에 사용하는 행렬 F 를 아래와 같이 생긴 3×3 으로 확장시키는 아이디어를 활용합니다.

- $$\begin{pmatrix} F_{n+1} \\ F_n \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} F_n \\ F_{n-1} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}^n * \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = F^n * \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

- 벡터의 3번째 항이 항상 1을 만족시키는 특징을 가지게 됩니다.

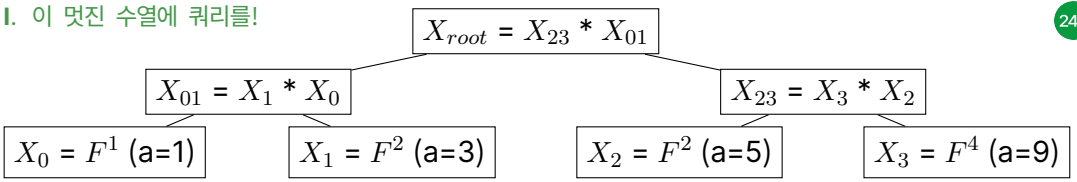
- ✓ 이 경우 쿼리 마법에 의해 a 번 구간의 값이 b 로 고정되었음을 처리를 할 때, 다음과 같이 생긴 행렬 B_b 를 정의해서, 이 행렬을 곱함으로써 쿼리를 처리할 수 있게 됩니다.

- ✓ $$\begin{pmatrix} F_a \\ F_{a-1} \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & b \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} F_{a-1} \\ F_{a-2} \\ 1 \end{pmatrix} = B_b * \begin{pmatrix} F_{a-1} \\ F_{a-2} \\ 1 \end{pmatrix}$$

- ✓ 그래픽스 분야에서 점의 평행이동을 표현할 때 자주 사용하는 방식입니다.
- ✓ 이 아이디어를 활용하면, offline 쿼리와 segment tree 를 통해 쿼리마다 정답을 $\mathcal{O}(\log Q)$ 에 구할 수 있게 됩니다.

I. 이 멋진 수열에 쿼리를!

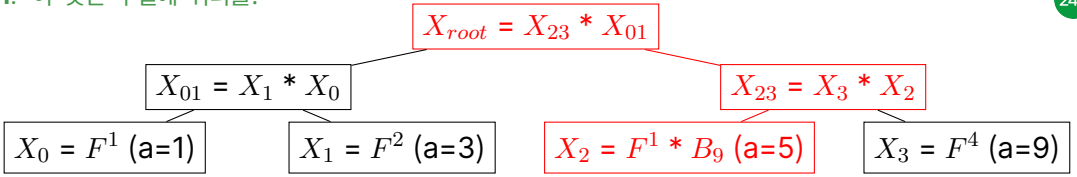
240



- ✓ 각 세그트리의 노드들은 3×3 행렬들을 가지고있고, 두 자식노드의 곱으로 부모노드의 행렬을 계산합니다.
- ✓ 세그트리의 말단노드는 쿼리 마법이 가해지는 수열 구간 위치를 기준으로 나눠줍니다. 예외로 1번째 수열 구간과 N 번째 수열 구간에 해당되는 노드도 말단노드로 추가해줍니다.
- ✓ 초기에 말단노드에는, 노드가 존재하는 이전 수열 구간에서 시작해, 현재 수열 구간에 해당하는 값을 구하기 위해 계산해야할 행렬을 가지고 있습니다. 쿼리 마법이 오기 전 이 값은 3×3 피보나치행렬 F 에다가, 수열 구간 사이의 거리의 거듭제곱을 한 값입니다.

I. 이 멋진 수열에 쿼리를!

240



- ✓ 쿼리가 들어오면, 말단노드의 행렬을 변화시킵니다. 변화된 행렬은, (이전 수열 구간으로부터 현재 위치 직전까지 도달하는데 필요한 행렬) * (쿼리에 대응되는 행렬). 이 될 것입니다.
- ✓ F_n 값은, root node 에 저장된 행렬을 이용해 구할 수 있습니다.
- ✓ $$\begin{pmatrix} F_{n+1} \\ F_n \\ 1 \end{pmatrix} = X_{root} * \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

- ✓ 각 쿼리마다 행렬이 변경되는 노드의 개수는 $\mathcal{O}(\log Q)$ 개입니다. 말단노드를 갱신할 때에는 $\mathcal{O}(\log N)$ 만큼의 계산이 필요합니다.
- ✓ 각 노드 값을 갱신할 때마다 3×3 행렬의 곱셈의 수행됩니다.
- ✓ 거듭제곱된 피보나치 행렬을 전처리로 계산하면 말단노드를 갱신할 때 $\mathcal{O}(\log N)$ 이 사라집니다.
- ✓ 전체적인 시간복잡도는 $\mathcal{O}(27Q(\log Q + \log N))$ (전처리시 $\mathcal{O}(Q \log N + 27Q \log Q)$), 공간복잡도는 $\mathcal{O}(9Q)$ 이 됩니다.

J. 일이 너무 많아...

`number theory`, `inclusion and exclusion`
출제진 의도 – **Medium**

- ✓ 제출 29번, 정답 12명 (정답률 41.379%)
- ✓ 처음 푼 팀: **No Seastar Can't Win**서강대학교, 47분
- ✓ 출제자: 홍은기`pichulia`

J. 일이 너무 많아...

240

- ~~이번 대회에서 4 문제를 출제하고 동시에 회사 일도 진행하니...~~
  ~~일이 너무 너무 많아졌습니다...~~
- N 보다 작거나 같은 자연수 중 A 의 배수인 수의 개수는 $\lfloor \frac{N}{A} \rfloor$ 로 계산할 수 있습니다.
- 따라서 기본적인 아이디어로, $\lfloor \frac{N}{11} \rfloor$, $\lfloor \frac{N}{111} \rfloor$, $\cdots$, $\lfloor \frac{N}{11\,111\,111\,111\,111\,111} \rfloor$ 의 합으로 계산할 수 있습니다.
- 위 공식에서 2 번 이상 중복되서 셈이 되는 숫자들을 찾아서 한번만 세어지도록 수정하는 과정이 필요합니다. (예를 들면 1221 은 11 의 배수이기도 하면서 동시에 111의 배수이기 때문에 $\lfloor \frac{N}{11} \rfloor$ 과 $\lfloor \frac{N}{111} \rfloor$ 에서 모두 계산이 될 것입니다.)
- 이 과정에서 포함 배제의 원리가 활용됩니다.

- 설명을 위해서 간단하게 4, 5, 6 의 배수를 구한다고 생각해봅시다.
- 정답은 $\lfloor \frac{N}{4} \rfloor + \lfloor \frac{N}{5} \rfloor + \lfloor \frac{N}{6} \rfloor - \lfloor \frac{N}{lcm\,(\{4,5\})} \rfloor - \lfloor \frac{N}{lcm\,(\{4,6\})} \rfloor - \lfloor \frac{N}{lcm\,(\{5,6\})} \rfloor + \lfloor \frac{N}{lcm\,(\{4,5,6\})} \rfloor$ 이 됩니다.
- $\lfloor \frac{N}{4} \rfloor$ 과 $\lfloor \frac{N}{5} \rfloor$ 에서 중복되서 계산된 숫자들을 찾아서 빼주는 것이 $\lfloor \frac{N}{lcm\,\{4,5\}} \rfloor$ 의 역할입니다.
- 나머지 식들도 이와 비슷한 역할을 적절히 수행해서 최종적으로는 4 또는 5 또는 6의 배수를 만족하는 수의 개수가 나옵니다.

240

- 이를 일반화해서 포함배제의 원리를 활용한 형태의 식으로 표현하면 다음과같이 됩니다. 여기서 A 는 배수관계를 만족하는 자연수의 개수를 구하려는 요소의 집합, $lcm\,(K)$ 는 집합 K 에 속한 모든 정수들의 최소공배수를 구하는 함수입니다.

- result = $\sum_{K \subseteq A, K \neq \varnothing} \left\{ \lfloor \frac{N}{lcm\,(K)} \rfloor * (-1)^{|K|+1} \right\}$

J. 일이 너무 많아...

240

- 일이 너무 많아... 문제에서 사용한 집합은 $A = \{11, 111, \cdots, 11\,111\,111\,111\,111\,111\}$ 인 총 16개의 원소를 가진 집합입니다.
- 전체 $2^{16} - 1$ 가지 부분집합 K 각각에 대해서 $\left\{ \lfloor \frac{N}{lcm\,(K)} \rfloor * (-1)^{|K|+1} \right\}$ 값을 계산할 수 있고, 이 계산 결과를 전부 더하면 정답이 나오게 됩니다.
- 집합 A 를 서로소인 7개의 수의 집합으로 줄일 수 있습니다. 이 경우 시간복잡도도 $\mathcal{O}\left(7 * 2^7\right)$ 로 줄어들고, 각 원소들이 모두 서로소이기 때문에 lcm 함수를 별도로 구현하지 않아도 되는 장점이 있습니다.
- 구현에 따라서 lcm 을 계산하는 도중에 오버플로우가 발생할 수 있으니 조심해야 합니다.

K. 올바른 괄호

`ad_hoc`, `string`, `stack`
출제진 의도 – **Easy**

- ✓ 제출 134번, 정답 36명 (정답률 26.866%)
- ✓ 처음 푼 팀: **정열맨**홍익대학교, 13분
- ✓ 출제자: 박찬솔`chansol`

240

- 문제의 조건에 따라, (와) 의 개수 차이는 항상 1이며, 더 많은 개수를 가진 문자를 지워야 합니다.
- (의 개수가) 보다 많은 경우와 그 반대의 경우로 나누어 생각해볼 수 있습니다.

240

- 첫번째 경우, 문자열이 ( ) (( ) ( ) ( ) 라고 해봅시다.
- (하나를 제거해서 올바른 괄호열이 되는지 확인해야 합니다.
- 앞의 두 문자 ( ) 는 이미 올바른 괄호열을 구성하고 있으므로 제외합니다. 올바른 괄호열에서 (를 지우면, (와) 의 개수는 같아지지만,) (( ) ( ) ( ) 와 같이 매칭이 안되므로 올바른 괄호열은 제외해서 생각해야 합니다.
- (( ) ( ) ( ) 에서는 어떤 (를 지우더라도 항상 올바른 괄호열이 됩니다.
- 처음으로 **잘못** 등장하는 (의 위치 이후에 있는 모든 (는 답이 됩니다.

240

- 두번째 경우, 문자열이 ( ) ( )) ( ) ( ) 라고 해봅시다.
- 비슷하게) 하나를 제거해서 올바른 괄호열이 되는지 확인해야 합니다.
- 뒤의 네 문자 ( ) ( ) 는 이미 올바른 괄호열을 구성하고 있으므로 제외합니다.
- ( ) ( )) 에서는 어떤) 를 지우더라도 항상 올바른 괄호열이 됩니다.
- 처음으로 **잘못** 등장하는) 의 위치 이전에 있는 모든) 는 답이 됩니다.
- 구현의 편의를 위해, 문자열을 뒤집고, (와) 를 각각 반대로 치환하여 **(의 개수가) 보다 많은 경우**로 문제를 해결할 수도 있습니다.

L. 팰린드롬 게임

`math`, `game theory`
출제진 의도 – **Medium**

- ✓ 제출 56번, 정답 32명 (정답률 58.929%)
- ✓ 처음 푼 팀: **No Seastar Can't Win**서강대학교, 4분
- ✓ 출제자: 김태영`tae826`

L. 팰린드롬 게임

240

- 풀이는 팰린드롬 수의 마지막 숫자가 0이 될 수 없음에 착안합니다.
- 첫 돌의 개수가 10의 배수라면 후공인 승우가 항상 10의 배수인 돌의 개수를 유지할 수 있습니다.
  - 1부터 9까지의 숫자가 모두 팰린드롬 수이기 때문에 항상 가능합니다.
  - A가 돌을 먹고 나서 개수가 항상 10의 배수가 유지된다면 A가 항상 이깁니다.
- 비슷한 논리로 첫 돌의 개수가 10의 배수가 아니라면 선공인 상윤이가 항상 10의 배수인 돌의 개수를 유지할 수 있고 이깁니다.
- 위의 전략을 사용하면, 개수를 입력 받아 10의 배수 여부만 판별하면 해결할 수 있습니다.

M. 불협화음

`rotating calipers`, `geometry`, `calculus`
출제진 의도 – **Challenging**

- ✓ 제출 0번, 정답 0명 (정답률 0%)
- ✓ 처음 푼 사람: **어라랍스타**연세대학교, ??분
- ✓ 출제자: 홍은기`pichulia`

- 행복한 기하문제입니다. :)

- 가장 먼저 파악해야할 것은, 원의 반지름 R 을 고려하지 않아도 된다는 점입니다.
- 작품이 되는 임의의 삼각형의 각 변을 R 만큼 안쪽으로 평행이동시키면, 삼각형의 각 변은 최소 하나의 원의 중심과 접하게 됩니다.
- 이 때 평행이동으로 인해 줄어든 삼각형의 각 변의 길이는 $2\sqrt{3}R$ 이 됩니다.
- 즉, 원의 중심만을 이용해서 구한 삼각형에 대한 정답을 먼저 구한 뒤, 이 결과값에 $2\sqrt{3}R$ 을 더하기만 하면 최종 정답이 됩니다.

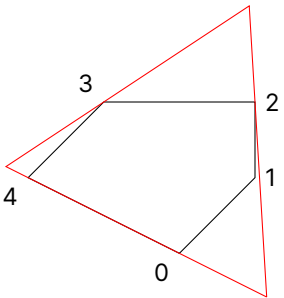
- 이제 '원의 중심 N 개를 모두 포함하는 convex hull 을 구한 뒤, 문제의 조건을 만족하는 삼각형의 변의 길이를 구하는 문제'로 환원되었습니다.

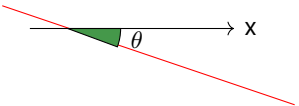
- ✓ 문제 풀이를 위해 삼각형의 상태를 다음과 같이 정의하였습니다.
- ✓ convex hull 위에 있지 않은 점은 정답에 영향을 주지 않으므로, convex hull 위의 M 개의 점들을 반시계방향 순서대로 $0, 1, \dots, M-1$ 차례대로 번호를 부여했습니다. 또한 M 번 점은 0 번 점과 같습니다.

- ✓ (i, j, k, θ) = 반시계방향 순서대로 삼각형의 각 변이 i, j, k 번 점과 접하며, 이 때 i 번 점과 접하는 변은 $+x$ 축 기준으로 반시계방향으로 θ 만큼 회전한 모습이다.

- ✓ 자명하게도, $(i, j, k, \theta) = (j, k, i, \theta + \frac{2\pi}{3}) = (k, i, j, \theta + \frac{4\pi}{3})$. 는 모두 같은 삼각형입니다.

240





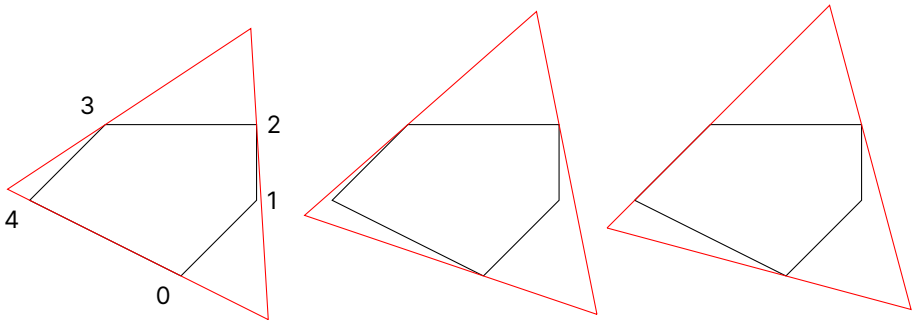
✓ 위 그림은 $(0, 2, 3, \theta \approx -0.463647)$. 가 대략 어떤 모습인지를 나타내고 있습니다.

- 여기서 새로운 변수를 하나 더 정의하고 가겠습니다.

- $\theta_{(i,j,k)}$ = 삼각형의 각 변이 i, j, k 번 정점과 접하게 되는 가장 작은 각도.

- 이 정의를 통해서, 우리는 다음과 같은 상황을 상상할 수 있습니다.

- 삼각형 $(i, j, k, \theta_{(i,j,k)})$ 에서 각도를 조금씩 증가시키다보면 $\theta_{(i+1,j,k)}, \theta_{(i,j+1,k)}, \theta_{(i,j,k+1)}$ 중 가장 작은 값에 맞춰서 삼각형의 상태가 $(i+1, j, k, \theta_{(i+1,j,k)}), (i, j+1, k, \theta_{(i,j+1,k)}), (i, j, k+1, \theta_{(i,j,k+1)})$ 중 하나로 변한다.



✓ 위 그림은 $(0, 2, 3, \theta_{(0,2,3)})$. 에서 삼각형을 조금씩 회전하다가 $(0, 2, 4, \theta_{(0,2,4)})$ 가 되는 모습을 보여주고 있습니다.

- ✓ 자명하게도, $(i, j, k, \theta_{(i,j,k)})$ 인 서로 다른 삼각형의 가지수는 최대 M 가지만 존재합니다. (간력하게 설명하자면, 삼각형의 변 중 하나가 convex hull 의 변과 접하는 순간의 가짓수만큼 삼각형이 존재합니다.)
- ✓ rotating calipers 알고리즘을 사용해서 가능한 모든 $\theta_{(i,j,k)}$ 를 미리 계산해 놓을 수 있습니다.
- ✓ 그럼 이제 $\theta_{(i,j,k)} \leq \theta < min\{\theta_{(i+1,j,k)}, \theta_{(i,j+1,k)}, \theta_{(i,j,k+1)}\}$ 를 만족하는θ 에 대해서 (i, j, k, θ) 일 때 삼각형의 한 변의 길이를 구하는 함수 $d(i, j, k, \theta)$ 를 정의한 뒤, 주어진 범위 내에서 $d(i, j, k, \theta)$ 값의 최대값과 최소값을 각각 구하는 것을 M 가지 삼각형에 대해서 반복하면, 전체 문제의 정답을 구할 수 있습니다.

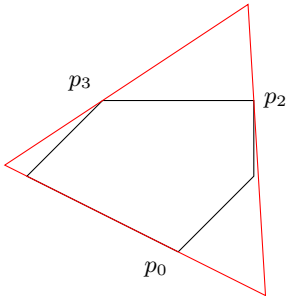
- ✓ $d(i, j, k, \theta)$ 함수가 어떤 형태일지를 구해봅시다.
- ✓ 변수의 자유도가 너무 높으면 헷갈리므로, 우선 변수의 자유도를 낮추는 작업을 진행해보겠습니다.
- ✓ 각 점의 위치를 p_i, p_j, p_k 라고 했을 때, i 번 점을 $(0, 0)$ 으로 평행이동 시킨 후 반시계방향으로 $-\theta_{(i,j,k)}$ 만큼 회전시키는 작업을 수행해둡니다.
- ✓ i, j, k 값이 정해지면 $p_i, p_j, p_k, -\theta_{(i,j,k)}$ 값 모두 θ 와 관련없는 상수값처럼 취급할 수 있습니다.

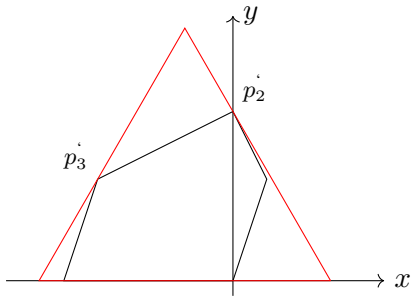
- 즉, 아래의 계산 결과는 θ 와 관계없는 상수 값이 될 것입니다.

- $$p_j^{'}.x = cos(-\theta_{(i,j,k)}) * (p_j - p_i).x - sin(-\theta_{(i,j,k)}) * (p_j - p_i).y$$
  $$p_j^{'}.y = sin(-\theta_{(i,j,k)}) * (p_j - p_i).x + cos(-\theta_{(i,j,k)}) * (p_j - p_i).y$$
  $$p_k^{'}.x = cos(-\theta_{(i,j,k)}) * (p_k - p_i).x - sin(-\theta_{(i,j,k)}) * (p_k - p_i).y$$
  $$p_k^{'}.y = sin(-\theta_{(i,j,k)}) * (p_k - p_i).x + cos(-\theta_{(i,j,k)}) * (p_k - p_i).y$$

- 이 작업이 완료되면, i 번 점과 접하는 변은 x 축과 일치하게 될 것이고, j 번 점과 접하는 오른쪽 변은 $+x$ 축 방향, k 번 점과 접하는 왼쪽 변은 $-x$ 축 방향에 위치하게 됩니다.

240





- ✓ 위 그림은 0 번 점을 원점으로 옮긴 뒤 전체 도형을 $-\theta_{(0,2,3)}$ 만큼 회전시켜서 자유도를 낮춘 뒤의 모습을 나타냅니다.
- ✓ 삼각형의 밑변이 x 축과 일치하게 되고 j 번과 k 번 점과 접하는 변의 위치도 밑변을 기준으로 각각 오른쪽 변 / 왼쪽 변 으로 명확해짐을 알 수 있습니다.

- 이 상태에서 우리는 추가적으로 $\theta - \theta_{(i,j,k)}$ 만큼 삼각형을 더 회전시킬 필요가 있습니다. 수식 전개 편의를 위해 이 값을 $\theta' = \theta - \theta_{(i,j,k)}$ 라는 변수로 표현하겠습니다.
- 삼각형을 θ' 만큼 회전시키는 대신에, 우리는 점을 원점을 중심으로 $-\theta'$ 만큼 회전시킬 것입니다. 이렇게 하면, 회전을 마친 뒤에도 삼각형의 밑변이 x 축이 되도록 유지시킬 수 있기에 계산이 편해지게 됩니다.

- $$p_j''.x = cos(-\theta') * p_j'.x - sin(-\theta') * p_j'.y$$
  $$p_j''.y = sin(-\theta') * p_j'.x + cos(-\theta') * p_j'.y$$
  $$p_k''.x = cos(-\theta') * p_k'.x - sin(-\theta') * p_k'.y$$
  $$p_k''.y = sin(-\theta') * p_k'.x + cos(-\theta') * p_k'.y$$

- ✓ 일련의 과정을 마친 뒤에는, 이제 삼각형의 변의 길이를 θ' 에 대한 함수로 표현할 수 있게 됩니다.
- ✓ 일련의 회전변환 결과, 삼각형의 밑변은 x 축과 동일하고, 오른쪽 변은 p_j'' 와 접하며, 왼쪽 변은 p_k'' 와 접하는 모습일 것입니다.
- ✓ 따라서 p_j'' 에서 $-\frac{2\pi}{3}$ 방향으로 그린 직선과 x 축의 교점의 x 좌표값을 $p_j'''.x$, p_k'' 에서 $-\frac{4\pi}{3}$ 방향으로 그린 직선과 x 축의 교점의 x 좌표값을 $p_k'''.x$ 라고 했을 때,
- ✓ 한 변의 길이는 결국 $p_j'''.x - p_k'''.x$ 가 됩니다.

- $p_j^{'''}.x$ 와 $p_k^{'''}.x$ 두 값을 각각 $\theta^{'}$ 에 대한 함수로 정리해봅시다.
- $$p_j^{'''}.x = p_j^{''}.x + \frac{p_j^{''}.y}{\sqrt{3}} = cos(\theta^{'}) * (p_j^{'}.x + \frac{p_j^{'}.y}{\sqrt{3}}) + sin(\theta^{'}) * (p_j^{'}.y - \frac{p_j^{'}.x}{\sqrt{3}})$$
$$p_k^{'''}.x = p_k^{''}.x - \frac{p_k.y}{\sqrt{3}} = cos(\theta^{'}) * (p_k^{'}.x - \frac{p_k.y}{\sqrt{3}}) + sin(\theta^{'}) * (p_k^{'}.y + \frac{p_k^{'}.x}{\sqrt{3}})$$
- 따라서 최종 삼각형의 한 변의 길이는 다음과 같이 정의됩니다.
- $$d(\theta^{'}) = cos(\theta^{'}) * (p_j^{'}.x + p_k^{'}.x + \frac{p_j^{'}.y - p_k^{'}.y}{\sqrt{3}}) + sin(\theta^{'}) * (p_j^{'}.y + p_k^{'}.y + \frac{-p_j^{'}.x + p_k^{'}.x}{\sqrt{3}})$$

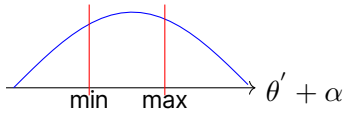
- 여기서 주목해야할 것은, $cos(\theta^{'})$ 와 $sin(\theta^{'})$ 에 곱해진 값들이 모두 $\theta^{'}$ 와 관계없는 상수 값이라는 사실입니다.
- $cos(\theta^{'})$ 에 곱해진 값을 A, $sin(\theta^{'})$ 에 곱해진 값을 B 라고 설정하고, 추가로 $Z = \sqrt{A^2 + B^2}$ 라는 변수와, $\alpha = atan2(B, A)$ (== $sin(\alpha) = A/Z, cos(\alpha) = B/Z$ 를 만족하는 임의의 각도 α 를 의미) 라는 변수도 추가로 설정하겠습니다. 이 네 개의 값 모두 상수입니다.
- 이제 삼각함수의 덧셈 정리를 활용해, $d(\theta^{'})$ 를 간단하게 표현할 수 있게 됩니다.

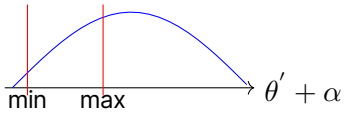
$$
\begin{aligned}
d(\theta^{'}) &= cos(\theta^{'}) * (p_j^{'}.x + p_k^{'}.x + \frac{p_j^{'}.y - p_k^{'}.y}{\sqrt{3}}) + sin(\theta^{'}) * (p_j^{'}.y + p_k^{'}.y + \frac{-p_j^{'}.x + p_k^{'}.x}{\sqrt{3}}) \\
&= cos(\theta^{'}) * A + sin(\theta^{'}) * B \\
&= Z * sin(\theta^{'} + \alpha)
\end{aligned}
$$

- ✓ 기나긴 여정 끝에 여기까지 도달하신 분들 모두 수고 많으셨습니다.

- ✓ 각도에 대한 삼각형의 변의 길이 함수는 sin 함수 꼴이였습니다.
- ✓ $d(\theta')$ 결과값이 삼각형의 변의 길이이므로 이 값이 음수가 될 일은 없을 것입니다.
- ✓ 따라서 물리적으로 $0 \leq \theta' + \alpha \leq \pi$ 를 만족한다고 가정해도 좋을 것입니다. 그리고 이 경우, sin 함수는 위로 볼록한 함수가 됩니다.

240





- $0 \le \theta' < min\{\theta_{(i+1,j,k)}, \theta_{(i,j+1,k)}, \theta_{(i,j,k+1)}\} - \theta_{(i,j,k)}$ 의 범위에서 $Z * sin(\theta' + \alpha)$ 함수 값의 최대값과 최소값을 각각 구해서 정답을 계산하면 됩니다.
- 위로 볼록한 볼록함수이므로, 최소값은 양 끝 경계점 ($\theta' = 0$ 또는 $\theta' = min\{\theta_{(i+1,j,k)}, \theta_{(i,j+1,k)}, \theta_{(i,j,k+1)}\} - \theta_{(i,j,k)}$) 두 경우 중 하나일 것입니다.
- 최대값은 양 끝 경계점과 함께, $\theta' + \alpha = \frac{\pi}{2}$ 를 만족하는 θ' 까지, 세 경우 중 하나일 것입니다.
- 볼록함수이기 때문에 삼분탐색 등을 사용해서 최대값을 구하셔도 정답을 받을 수 있습니다.