SNUPC 2021 풀이

Official Solutions

by

SNUPC 2021

2021년 9월 11일

Div 2 문제		의도한 난이도	출제자
A	5의 수난	**Easy**	kipa00
B	박스 그림 문자	**Easy**	16silver
C	실 전화기	**Medium**	doju
D	누텔라 트리 (Easy)	**Medium**	dlalswp25
E	뛰는 기물	**Medium**	kipa00
F	AND와 OR	**Hard**	kipa00
G	자연수 색칠하기	**Hard**	16silver
H	트리 찾기	**Hard**	dlalswp25
I	큰 수 뒤집기	**Challenging**	16silver

Div 1 문제		의도한 난이도	출제자
A	트리 조각하기	Easy	onjo0127
B	별 보는 교준이	Medium	yclock
C	큰 수 뒤집기	Medium	16silver
D	여우 국수	Hard	doju
E	자연수 색칠하기	Easy	16silver
F	고슴도치 그래프	Hard	kipa00
G	데칼코마니 트리	Hard	yclock
H	RMQ	Challenging	moonrabbit2
I	두 트리	Challenging	yclock
J	문자열 X	Hard	dlalswp25
K	누텔라 트리 (Hard)	Challenging	dlalswp25

2A. 5의 수난

`math, implementation`
출제진 의도 – **Easy**

- 제출 77번, 정답 69명 (정답률 89.61%)
- 처음 푼 사람: **양창석**, 0분
- 출제자: `kipa00`

2A. 5의 수난

- **다섯** 자리 수가 주어졌을 때, 각 자릿수의 **다섯**제곱의 합을 구하는 문제입니다.
- 문제에 설명되어 있는 그대로 구현하면 됩니다.

2A. 5의 수난

- 어떤 수 n의 일의 자릿수가 n을 10으로 나눈 나머지이고, 십의 자릿수 이상이 n을 10으로 나눈 몫이라는 것을 이용하면 정수 입력으로 풀 수 있습니다.
- 문자열로 입력받아서 각 자리를 순회하며 **다섯**제곱을 해서 풀 수도 있습니다.
- C/C++의 경우 `%1d`를 이용하면, 최대 한 글자를 입력받아서 수로 반환한 결과를 돌려주기 때문에 이를 이용해도 됩니다.

2B. 박스 그림 문자

`implementation`
출제진 의도 – **Easy**

- 제출 109번, 정답 60명 (정답률 55.56%)
- 처음 푼 사람: **신원석**, 8분
- 출제자: `16silver`

2B. 박스 그림 문자

3년만에 돌아온 .# 문제입니다. (출제자도 동일합니다.)

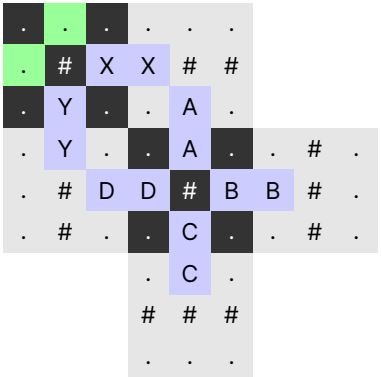
- 일단 채워야 하는 칸의 3×3 중 가운데는 반드시 #입니다.
- 이 #을 기준으로 상하좌우가 .# 중 어떤 것인지 결정하면 됩니다.
- 대각선으로 인접한 칸들은 항상 .입니다.

비어 있는 칸의 상하좌우로 인접한 칸을 확인합니다. 이 칸은

1. 없거나 (모서리),
2. 채워져 있습니다.

2B. 박스 그림 문자

- 모서리 쪽의 경우 문제 조건에 의해 답은 . 입니다.
- 채워져 있는 경우 해당 방향으로 한 칸 더 갔을 때의 문자가 답입니다.



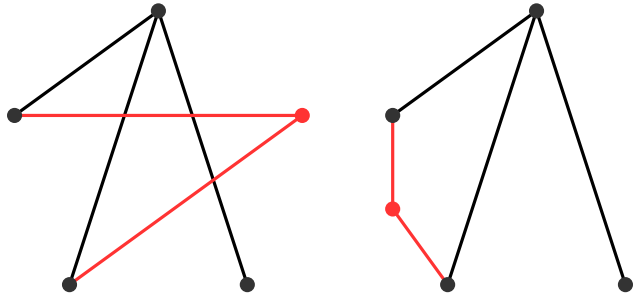
2C. 실 전화기

`case_work`
출제진 의도 – **Medium**

- 제출 130번, 정답 29명 (정답률 28.16%)
- 처음 푼 사람: **박세현**, 25분
- 출제자: `doju`

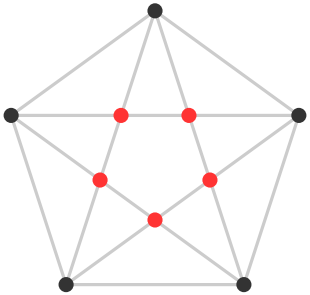
2C. 실 전화기

- 다섯 개의 정점들이 정오각형 모양으로 놓여 있고 그 사이에 간선들이 일직선으로 그어져 있을 때, 점을 옮겨서 간선들이 교차하지 않도록 하는 문제입니다.



2C. 실 전화기

- 처음 상태에서 생길 수 있는 교차점은 총 다섯 개가 있습니다.



2C. 실 전화기

- 다섯 개의 교차점이 하나도 발생하지 않았다면 답은 `0`입니다.
- 교차점이 있는지 확인하려면 각 교차점을 이루는 두 개의 간선이 모두 입력에 포함되었는지 확인하면 됩니다.
  - 예를 들어 2-4 간선과 3-5 간선이 모두 주어지면 가장 아래쪽의 교차점이 만들어집니다.

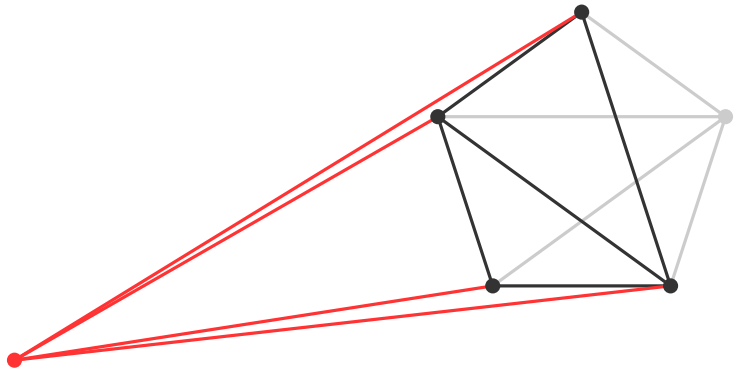
2C. 실 전화기

- 다음으로, 답이 1이 되는 경우를 생각해 봅시다.

2C. 실 전화기

- 다음으로, 답이 1이 되는 경우를 생각해 봅시다.
- 흥미롭게도, 정점을 하나 **없애서** 교차점을 모두 없앨 수 있다면 답은 1이 됩니다.

- 그 이유는 정점을 없애는 것과 충분히 멀리 떨어뜨려 놓는 것이 같은 결과를 내기 때문입니다.

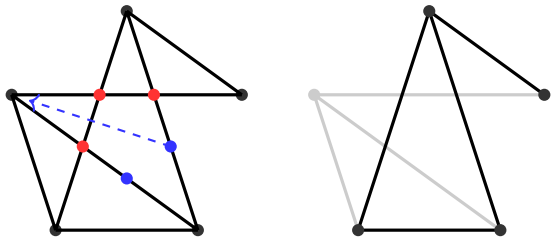


2C. 실 전화기

- 이것만으로도 구현을 할 수는 있지만, 좀 더 간단하게 정리해 봅시다.

2C. 실 전화기

- 이것만으로도 구현을 할 수는 있지만, 좀 더 간단하게 정리해 봅시다.
- 정점 하나를 지우면 정점이 네 개밖에 남지 않으므로, 생길 수 있는 교차점은 단 하나뿐입니다.
- 즉, 다섯 개의 교차점 중 어느 하나가 생기지 않았다면, 그 반대쪽의 정점을 없애면 교차점이 모두 없어집니다.

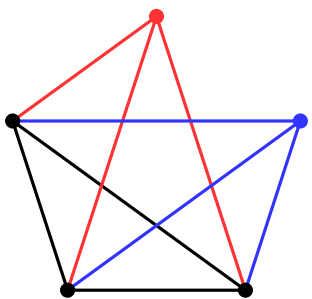


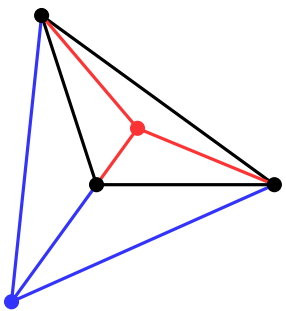
2C. 실 전화기

- 이것만으로도 구현을 할 수는 있지만, 좀더 간단하게 정리해 봅시다.
- 정점 하나를 지우면 정점이 네 개밖에 남지 않으므로, 생길 수 있는 교차점은 단 하나뿐입니다.
- 즉, 다섯 개의 교차점 중 어느 하나가 생기지 않았다면, 그 반대쪽의 정점을 없애면 교차점이 모두 없어집니다.
- 따라서 다섯 개의 교차점 중 하나라도 생기지 않았다면 답은 1이 됩니다.

2C. 실 전화기

- 남은 경우는 다섯 개의 교차점이 모두 주어진 경우입니다.
- 가장자리의 간선 다섯 개가 모두 주어질 경우, 예제로 주어졌듯 답은 -1입니다.
- 그렇지 않은 경우 답이 2임을 어렵지 않게 확인할 수 있습니다.





2C. 실 전화기

- 정리하면 답은 `0`, `1`, `2`, `-1` 중 하나입니다.
- 입력된 실 전화기의 배치를 적절한 조건문을 사용해 네 가지 경우 중 하나로 분류해 주면 문제를 풀 수 있습니다.

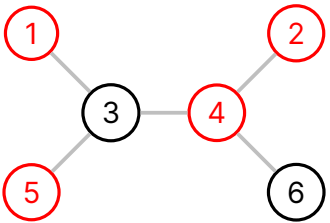
2D. 누텔라 트리 (Easy)

`trees`, `dfs`, `disjoint_set`
출제진 의도 – **Medium**

- 제출 157번, 정답 29명 (정답률 30.85%)
- 처음 푼 사람: **이재찬**, 22분
- 출제자: `dlalswp25`

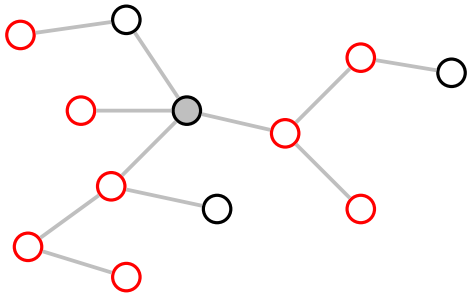
2D. 누텔라 트리 (Easy)

- 각 정점이 빨간색 또는 검은색으로 칠해진 트리가 주어질 때, 이 트리에서 **누텔라 경로**의 개수를 구하는 문제입니다.
  - 누텔라 경로란, 첫 번째 정점이 검은색이고 나머지 정점이 모두 빨간색인 길이 2 이상의 경로를 뜻합니다.



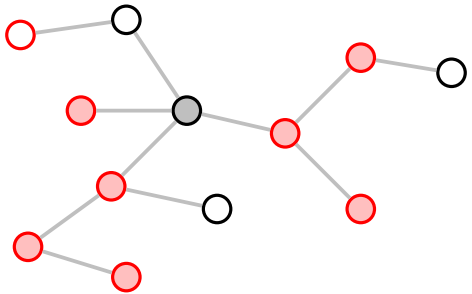
2D. 누텔라 트리 (Easy)

- 먼저 단순한 방법부터 생각해 봅시다.
- 경로의 시작점에 해당하는 검은색 정점을 고정합시다. 이때 경로의 끝점으로 가능한 정점은 몇 개일까요?



2D. 누텔라 트리 (Easy)

- 먼저 단순한 방법부터 생각해 봅시다.
- 경로의 시작점에 해당하는 검은색 정점을 고정합시다. 이때 경로의 끝점으로 가능한 정점은 몇 개일까요?



2D. 누텔라 트리 (Easy)

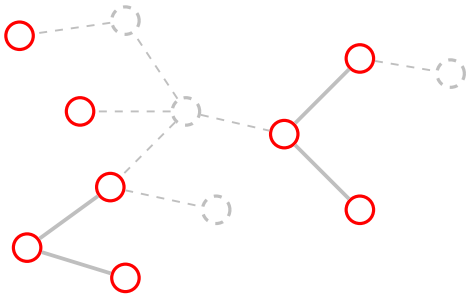
- **빨간색 정점만 따라가면서 도달할 수 있는 정점의 수**임을 알 수 있습니다.
- 각 검은색 정점에 대해 DFS 등으로 이러한 빨간색 정점의 개수를 구할 수 있으며, 이를 모두 더하면 답이 됩니다.
- 그러나 이 풀이의 시간 복잡도는 $\mathcal{O}(N^2)$으로, 너무 느립니다.

2D. 누텔라 트리 (Easy)

- 이 풀이가 느린 이유는 빨간색 정점들을 매번 새로 탐색하기 때문입니다.
- 어떻게 하면 더 효율적으로 해결할 수 있을까요?
- 트리에서 검은색 정점을 전부 무시하고, 각 빨간색 컴포넌트의 크기를 미리 구해 놓읍시다.
  - Union-Find나 DFS 등으로 처리할 수 있습니다.

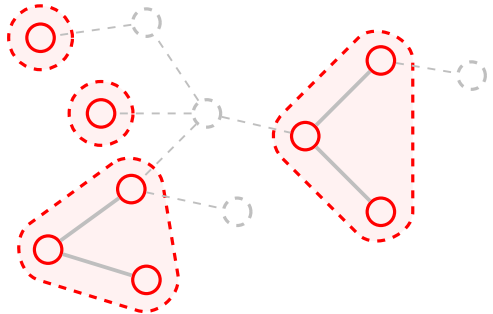
2D. 누텔라 트리 (Easy)

- 트리에서 검은색 정점을 전부 무시하고, 각 빨간색 컴포넌트의 크기를 미리 구해 놓읍시다.



2D. 누텔라 트리 (Easy)

- 트리에서 검은색 정점을 전부 무시하고, 각 빨간색 컴포넌트의 크기를 미리 구해 놓읍시다.



2D. 누텔라 트리 (Easy)

- 이제 각 검은색 정점에 대해 트리 전체를 탐색하는 대신, 인접한 빨간색 컴포넌트의 크기를 더해 주면 됩니다.
- 시간 복잡도는 $\mathcal{O}(N)$ 입니다.

2E. 뛰는 기물

`math`, `case_work`
출제진 의도 – **Medium**

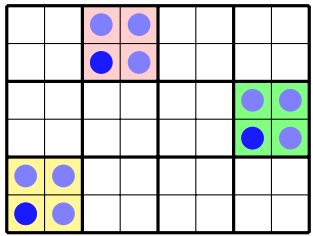
- 제출 134번, 정답 26명 (정답률 34.67%)
- 처음 푼 사람: **김태형**, 27분
- 출제자: `kipa00`

2E. 뛰는 기물

- (n, m)-기물이 좌표평면의 임의의 격자점을 방문하기 위해서 최소 몇 개의 시작점이 필요한지를 구하는 문제입니다.
- 예를 들어 $(1, 2)$-기물은 하나의 지점에서 다른 어떤 지점이든 갈 수 있습니다.

2E. 뛰는 기물

- 만일 $\gcd(n, m) \neq 1$ 이면 어떻게 될까요?
  - 즉, n 과 m 을 동시에 나누는 가장 큰 $g > 1$ 이 있다면 어떻게 될까요?
- 예를 들어 $n = 2, m = 4$ 라고 하면 $g = 2$ 가 n 과 m 을 동시에 나눕니다.

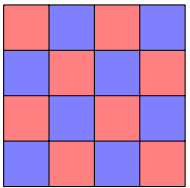


- 그림에서 볼 수 있듯이, $\gcd(n, m)$ 단위로 큰 칸을 새로 나누면, 같은 큰 칸에 속한 작은 칸끼리는 절대로 만날 수 없습니다.

- 따라서, $\gcd(n, m) =: g \neq 1$ 인 경우 $(n/g, m/g)$-기물의 답에 g^2 를 곱하면 됩니다.
  - 문제를 서로소인 경우로 축소했습니다!
- 이제부터 $\gcd(n, m) = 1$ 이라고 가정합니다. 문제를 더 줄일 수 있을까요?

2E. 뛰는 기물

- (n, m) 중 큰 쪽을 줄이려고 해 봅시다. 몇 가지 예외 사항을 제외하고 $n > m \geq 1$ 이라 합시다.
  - 만일 $m > n$ 이면 (m, n)-기물을 대신 생각합니다.
  - 만일 $m = n$ 이면 $\gcd(n, m) = \gcd(n, n) = n = 1$ 이므로 $m = n = 1$ 인데, 이 경우 답은 2 입니다.
    - 체스판 색칠이라는 아주 유명한 테크닉입니다. 아래 그림에서 현재 말이 있는 색의 다른 색으로는 절대 이동할 수 없습니다.



  - 만일 $m = 0$ 이면 $\gcd(n, m) = 1$ 이므로 $n = 1$ 일 것이고 이 경우 답 (1) 은 매우 자명합니다.

2E. 뛰는 기물

- 이제 $n > m \geq 1$ 이라 가정해도 좋습니다.
- 만일 $n > 2m$ 이면, (n, m)-기물이 다음과 같은 과정을 통해 $(n - 2m, m)$-기물이 됩니다:

$$(0, 0) \to (n, m) \to (n - m, m + n) \to (n - 2m, m).$$

- 이때 $(n - 2m, m)$-기물로도 (n, m)-기물을 흉내낼 수 있습니다. 즉 두 기물은 완전히 동등합니다.

$$(0, 0) \to (n-2m, m) \to (n-2m+m, m-(n-2m)) \to (n-2m+m+m, m) = (n, m).$$

- n과 m 중 최댓값은 n인데, $n - 2m$과 m 모두 n보다 작으므로 최댓값 줄이기에 성공했습니다.

2E. 뛰는 기물

- 만일 $m < n < 2m$ 이면, (n, m)-기물이 다음과 같은 과정을 통해 $(2m - n, m)$-기물이 됩니다:

$$(0, 0) \to (-n, m) \to (m - n, m + n) \to (2m - n, m).$$

- 이때 $(2m - n, m)$-기물로도 (n, m)-기물을 흉내낼 수 있습니다. 즉 두 기물은 완전히 동등합니다.

$$\begin{aligned}(0, 0) \to (-(2m - n), m) &\to (m - (2m - n), m + (2m - n)) \\ &\to (2m - (2m - n), m) = (n, m)\end{aligned}$$

- n과 m 중 최댓값은 n인데, $2m - n$과 m 모두 n보다 작으므로 최댓값 줄이기에 성공했습니다.

2E. 뛰는 기물

- 만일 $n = 2m$ 이면, $\gcd(m, n) = \gcd(m, 2m) = m = 1$ 이므로 $(1, 2)$-기물이 됩니다.
- 우리 모두는 이 기물이 한 칸씩 이동할 수 있다는 사실을 알고 있습니다. **예제에 있기 때문이죠.**

$$(0, 0) \to (1, 2) \to (3, 1) \to (1, 0).$$

- 상하좌우로 한 칸씩 이동할 수 있는 "기물"이 $(1, 2)$-기물을 흉내낼 수 있음은 말할 것도 없습니다.

- 즉 우리는 다음과 같은 변환을 찾았습니다.
  - $n > 2m$ 이면 (n, m)-기물은 $(n - 2m, m)$-기물과 동등
  - $m < n < 2m$ 이면 (n, m)-기물은 $(2m - n, m)$-기물과 동등
- 각 과정에서 주의깊게 보아야 할 부분이 있습니다.
  - 모든 과정은 최대공약수를 변화시키지 않습니다. 따라서 $n \neq 2m, n \neq m, m \neq 0$ 인 한 이 과정은 계속해서 적용할 수 있습니다.
  - 모든 과정은 $n + m$ 의 홀짝성을 유지합니다.

2E. 뛰는 기물

세 가지의 끝나는 경우에 대해 살펴봅시다.

- $n = 2m$이 되어 끝나는 경우, 즉 $(1, 2)$-기물로 환원되는 경우
  - 이 경우는 $n + m$이 홀수여야 도달 가능하며, 예제에서 보았듯이 최종적으로는 답이 1이 됩니다.
- $n = m$이 되어 끝나는 경우, 즉 $(1, 1)$-기물로 환원되는 경우
  - 이 경우는 $n + m$이 짝수여야 도달 가능하며, 최종적으로는 답이 2가 됩니다.
- $m = 0$이 되어 끝나는 경우, 즉 $\gcd(n, m) = \gcd(n, 0) = n = 1$이므로 $(1, 0)$-기물로 환원되는 경우
  - 이 경우는 $n + m$이 홀수여야 도달 가능하며, 최종적으로는 답이 1이 됩니다.

...?

2E. 뛰는 기물

- 어떻게 끝나는지에 관계없이 $n + m$ 이 홀수이면 답이 1, 짝수이면 답이 2 가 됩니다!
- 즉, 최종적인 풀이는 다음과 같습니다.
  1. $\gcd(n, m) = g$ 를 계산합니다.
  2. $(n/g + m/g)$ 가 홀수이면 답은 g^2, 짝수이면 답은 $2g^2$ 이 됩니다.

2E. 뛰는 기물

- g를 찾기 위해 가능한 모든 수로 나눠 보는 방법으로는 시간 초과가 납니다.
- n과 m을 $\mathcal{O}(\sqrt{n}+\sqrt{m})$에 각각 소인수분해하는 풀이는 통과할 수 있습니다.
- 위에서 소개드린 테크닉과 비슷한 테크닉, 즉 **둘 중 큰 수 줄이기**를 이용해서 최대공약수를 $\mathcal{O}(\log \min(n, m))$ 시간에 구할 수 있습니다.
  - 사실 이렇게까지 장황하게 설명하지 않아도 되는데 다소 길게 설명한 이유입니다. 부디 insight를 얻어가시기를 바랍니다...
  - 구체적인 알고리즘에 대해서는 "유클리드 호제법"을 찾아보시면 여기에 적는 것보다 훨씬 잘 설명된 자료가 많으므로 따로 설명하지는 않겠습니다.
- C++17의 `numeric` 헤더에는 함수 `gcd`가 내장되어 있습니다!

2F. AND와 OR

`math`, `ad_hoc`, `greedy`
출제진 의도 – **Hard**

- 제출 67번, 정답 35명 (정답률 59.32%)
- 처음 푼 사람: **김태형**, 12분
- 출제자: `kipa00`

2F. AND와 OR

- 음이 아닌 정수 N 개가 주어집니다.
- 두 개의 수 a와 b를 x와 y로 바꾸는 연산을 원하는 만큼 할 수 있습니다.
  - 이때, (a, b)와 (x, y) 쌍의 bitwise AND와 bitwise OR가 모두 같아야 합니다.
- 최종적으로 바뀐 N 개의 수의 곱을 최소화하세요.

2F. AND와 OR

- 관찰을 어떻게 하냐에 따라 천차만별의 방법으로 같은 결론에 도달하셨을 거라 생각합니다.
- 자명한 관찰부터 시작합니다.

Theorem

임의의 음이 아닌 정수 a, b에 대해, a와 b 모두 폐구간 $[a \text{ AND } b, a \text{ OR } b]$ 안에 있다.

- $a \text{ AND } b$는 a나 b의 비트에 포함된 비트 중 일부만 사용한 수이기 때문에 당연히 a와 b보다는 작거나 같습니다.
- $a \text{ OR } b$는 a나 b의 비트를 모두 가져오고 이후 새로 비트를 추가한 수이기 때문에 당연히 a와 b보다는 크거나 같습니다.

- 이 관찰은 그리 자명하지 않은 관찰입니다.

Theorem

임의의 음이 아닌 정수 a, b에 대해, $a + b = (a \text{ AND } b) + (a \text{ OR } b)$.

- 증명은 여러 가지 방법으로 할 수 있습니다.

2F. AND와 OR

- a와 b를 더하는 과정을 생각합시다.
- 받아올림이 없이 더한다면 결과는 $a \text{ XOR } b$입니다.
- 받아올림이 일어나는 자리는 $a \text{ AND } b$ 자리이고, 실제로는 이 값의 2배를 더해 주어야 덧셈이 올바릅니다.
- 즉, $a + b = (a \text{ XOR } b) + 2(a \text{ AND } b)$ 입니다. 그런데

$$\begin{aligned} a + b &= (a \text{ XOR } b) + 2(a \text{ AND } b) \\ &= (a \text{ XOR } b) + (a \text{ AND } b) + (a \text{ AND } b) \\ &= (a \text{ OR } b) + (a \text{ AND } b) \end{aligned}$$

이므로 증명이 완료됩니다.

- 이 사실은 N 개의 수를 어떻게 바꾸더라도 합이 일정하다는 의미입니다.
- 그렇다면 두 수를 최대한 멀리 떨어뜨리는 게 좋지 않을까요?
  - 우리는 a와 b를 $a \text{ AND } b$와 $a \text{ OR } b$로 떨어뜨릴 수 있고, 첫 번째 관찰에 의해 이보다 더 멀리 떨어뜨릴 수는 없습니다.
  - 이렇게 떨어지지 않은 쌍이 있다면 곱이 최소가 되지 않음을 알 수 있습니다: 이렇게 떨어뜨리는 과정을 시행하면 곱이 더 작아질 것이기 때문입니다.

2F. AND와 OR

- 이제 이런 $\mathcal{O}(N^2)$ 풀이를 생각할 수 있습니다.
  1. N 개의 수를 보면서 "떨어뜨리기" 과정을 반복해 모든 수의 OR 값을 구해서 맨 오른쪽에 저장한다.
     - 이 수와 연산을 해서 절대로 더 떨어뜨릴 수 없기 때문에, 이 수를 수열에서 제외하고 생각할 수 있습니다.
  2. 왼쪽에서 $(N-1)$ 개, $(N-2)$ 개, $\cdots$, 2 개의 수로 1.을 반복한다
  3. N 개의 수의 곱을 출력한다

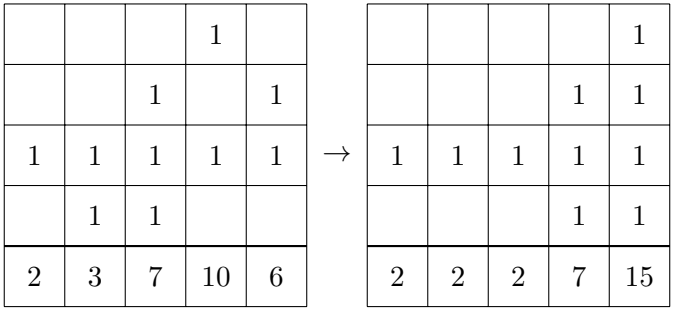
2F. AND와 OR

- 그런데 이 과정을 시행할 때 N이 매우 크면, 대부분의 계산 결과는 변하지 않습니다!
  - 이 연산 결과는 제한에서 30번보다 많이 변할 수는 없습니다.
  - OR을 할 때 1로 설정된 비트는 다시 0으로 설정되는 일이 없고, 가용 비트는 총 30개이기 때문입니다.
- 만일 매 단계마다 30개의 수를 잘 찾을 수만 있다면 $\mathcal{O}(30N)$에 문제를 해결할 수 있습니다.
- 이 방법으로도 문제를 해결할 수는 있으나, 구현이 그렇게 쉽지는 않습니다.

2F. AND와 OR

- 성질을 조금 더 관측해 봅시다.
  - a와 b 중 양쪽 모두 1인 비트는 a AND b에도 속하고, a OR b에도 속할 수밖에 없습니다.
  - a와 b 중 한쪽에만 1이 있는 비트는 a AND b에서는 빠지고, a OR b에는 들어갑니다.
  - a와 b 모두 없는 비트는 a AND b에도 a OR b에도 들어갈 수 없습니다.
- 위 사실을 정리하면, **각 자리의 비트의 개수가 유지됩니다.**
- 제곱 알고리즘에서 OR을 오른쪽으로 보낸다는 것은 **각 자리의 비트를 오른쪽으로 몰아서 계산하는 것**으로 바꾸어 생각할 수 있습니다!

2F. AND와 OR



2F. AND와 OR

- 따라서 다음과 같은 방법으로 문제를 해결할 수 있습니다.
  1. 모든 N 개의 수에 대해, 각 자리의 1 로 설정된 비트 개수를 센다.
  2. 개수가 남은 모든 비트에 대해 그 비트를 1 로 설정한 수를 추가하고, 설정된 각 비트의 개수를 1 뺀다.
  3. 2.를 N 번 반복하여 얻은 수를 모두 곱한다.
- 시간 복잡도는 동일하게 $\mathcal{O}(30N)$ 이나 직전에 소개드린 방법보다 구현이 훨씬 간단합니다.

2G. 자연수 색칠하기

`constructive, number_theory`
출제진 의도 – **Hard**

- 제출 80번, 정답 53명 (정답률 74.65%)
- 처음 푼 사람: **정성재**, 19분
- 출제자: `16silver`

- 가능한 최소 색의 수를 알아야 합니다.
- 반드시 모두 다른 색로 칠해야 하는 K 개의 수를 찾아야 합니다.
- 임의의 서로 다른 두 소수는 서로소입니다.

N 이하의 소수의 개수를 $\pi(N)$ 이라 하자. 최소 $\pi(N)+1$ 개의 색이 필요하다.

Proof. N 이하의 소수들을 $p_1, p_2, \cdots, p_{\pi(N)}$ 이라 하자. 이 때, $1, p_1, p_2, \cdots, p_{\pi(N)}$ 을 모두 다른 색으로 칠해야 하므로 색은 최소 $\pi(N)+1$ 개 필요하다.

2G. 자연수 색칠하기

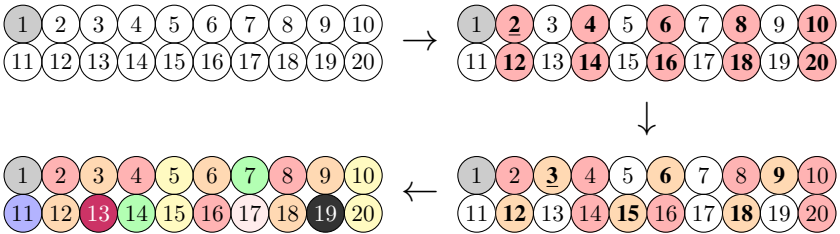
- 이제 소수가 아닌 다른 수들을 색칠해봅시다.
- 모든 합성수 x를 어떤 소수 p와 같은 색으로 칠하고 싶습니다.
- 두 수가 서로소가 아니어야 하기 때문에, x는 p의 배수여야 합니다.
- 색칠에 대한 조건은 이것만으로 충분합니다. (그리고 필요조건이기도 합니다.)

$\pi(N)+1$개의 색으로 N 이하의 모든 자연수를 조건에 맞게 색칠할 수 있다.

Proof. $1, p_1, p_2, \cdots, p_{\pi(N)}$을 모두 다른 색으로 칠한 뒤, 합성수 x는 x의 (임의의) 소인수 p와 같은 색으로 칠한다. 이 때

1. 1은 모든 수와 다른 색
2. x, y가 같은 색 $\Rightarrow$ 그 색의 소수 p에 대해 x, y는 모두 p의 배수 $\Rightarrow x, y$는 서로소가 아님

2G. 자연수 색칠하기



- 에라토스테네스의 체를 이용하여 답을 만들어낼 수 있습니다.

  시간복잡도 : $\mathcal{O}(n \log \log n)$

2H. 트리 찾기

`trees`, `binary_search`
출제진 의도 – **Hard**

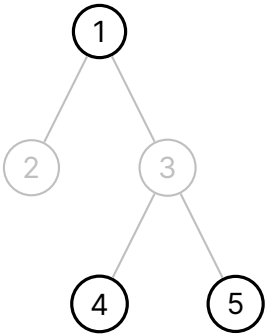
- 제출 27번, 정답 2명 (정답률 66.67%)
- 처음 푼 사람: **신기준**, 144분
- 출제자: `dlalswp25`

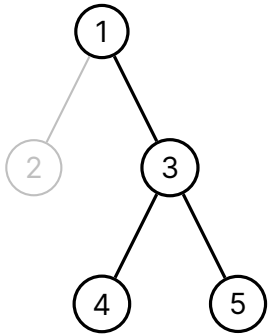
2H. 트리 찾기

- 정점이 N 개인 트리 T 가 있습니다.
- 다음 질의를 최대 $11\,111$ 번 사용하여 이 트리의 간선들을 알아내야 합니다.
  - 서로 다른 K 개의 정점 $u_1, u_2, \cdots, u_K$ 를 선택합니다.
  - 이 질의에 대해, 다음 조건을 만족하는 정점 v 의 개수를 알 수 있습니다.
    - u_i 와 u_j 를 잇는 최단 경로 상에 v 가 있도록 하는 $1 \leq i \leq j \leq K$ 가 존재한다.

2H. 트리 찾기

- "u_i와 u_j를 잇는 최단 경로 상에 v가 있도록 하는 $1 \leq i \leq j \leq K$가 존재한다."





2H. 트리 찾기

- 문제에서 주어진 질의는 "$u_1, u_2, \cdots, u_K$ 를 모두 포함하는 컴포넌트의 최소 크기"를 묻는 것과 동치입니다.
- 질의의 형태가 복잡하니, 가장 단순한 경우인 $K = 2$ 부터 고려해 봅시다.
- 정점 두 개를 골라 질의를 하면 이들 사이의 거리를 알 수 있습니다.
- $N - 1$ 번의 질의를 사용해서 1 번 정점으로부터 나머지 정점까지의 거리를 모두 구해줍시다.
- 이제 트리의 루트를 1 번 정점이라 하면 각 정점의 깊이를 알 수 있습니다.
- 남은 것은 각 정점의 부모를 결정하는 일입니다.

2H. 트리 찾기

- 깊이가 d인 정점 x의 부모는 다음과 같이 결정할 수 있습니다.
  - x의 부모가 될 수 있는 후보는 깊이가 $d-1$인 정점 전부입니다.
  - 다음 과정을 통해 후보를 절반씩 줄여 나갈 수 있습니다.
    - $S =$ (깊이가 $d-2$ 이하인 정점 전부) $\cup$ (후보 정점의 절반) $\cup \{x\}$에 대해 질의를 합니다.
    - 만약 S에 속하는 정점들 중 x의 부모가 있다면, 질의의 답은 $|S|$가 될 것입니다.
    - 그렇지 않다면, 질의의 답은 $|S|+1$이 될 것입니다.
    - 즉, 질의의 결과에 따라 후보의 절반을 제거할 수 있습니다.
  - 따라서 최대 $\lceil \log_2 N \rceil$번의 질의를 통해 후보를 하나만 남길 수 있고, 이 정점이 x의 부모가 됩니다.

2H. 트리 찾기

- 맨 처음 $N-1$ 번의 질의를 사용하였고, 각 정점의 부모를 결정하는 데 최대 $\lceil \log_2 N \rceil$ 번의 질의를 사용하므로 총 질의 횟수는 $N + N\lceil \log_2 N \rceil$ 이하입니다.
- $N = 1000$ 일 때 이 값은 $11\,000$ 으로, 횟수 제한 안에 들어옵니다.

21. 큰 수 뒤집기

`math`, `prefix_sum`
출제진 의도 – **Challenging**

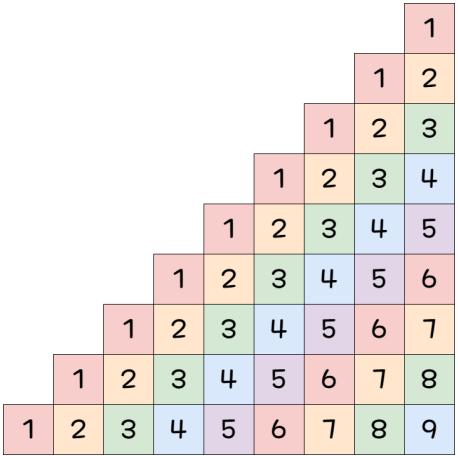
- 제출 32번, 정답 0명 (정답률 0.00%)
- 처음 푼 사람: `-`, `-`분
- 출제자: `16silver`

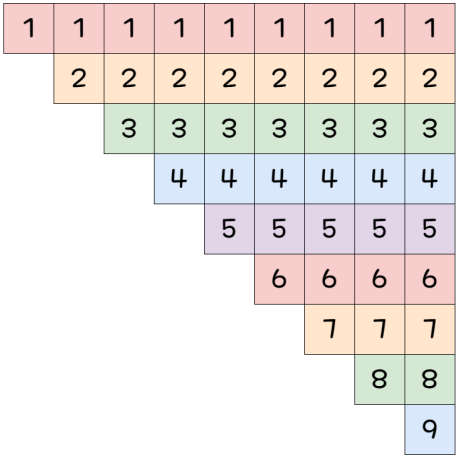
2I. 큰 수 뒤집기

"20658번 : 파이썬은 너무 느려"를 풀다 영감을 받아서 만든 문제 맞습니다.

- 뒤집는 연산이 없을 때의 문제를 풀어봅시다.
- 직접 수를 더해나간다면 $\mathcal{O}(n^2)$ 이 되어 오래 걸립니다.
- 각 숫자가 답에 얼마나 기여하는지를 살펴봅시다.

2I. 큰 수 뒤집기


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


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

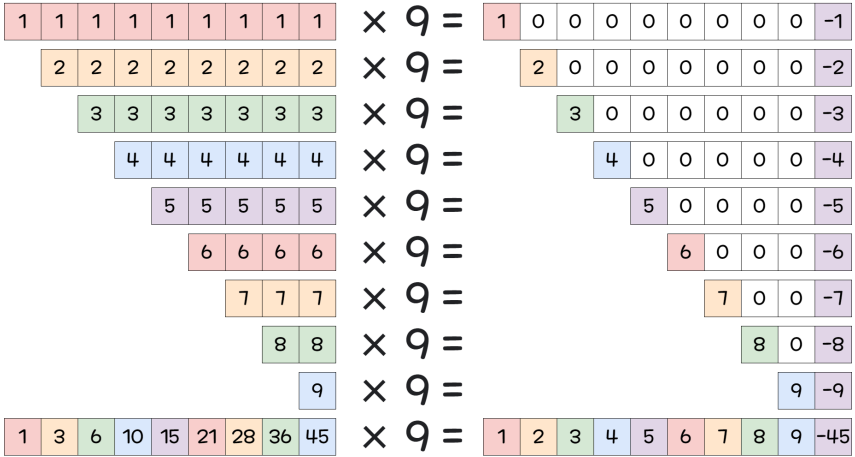
방법 1. 누적 합

1. 왼쪽부터 숫자를 하나씩 채웁니다.
2. 한 칸씩 오른쪽으로 가면서, 이전 값을 더합니다.
3. 오른쪽부터 받아올림을 진행하여 답을 얻어냅니다.

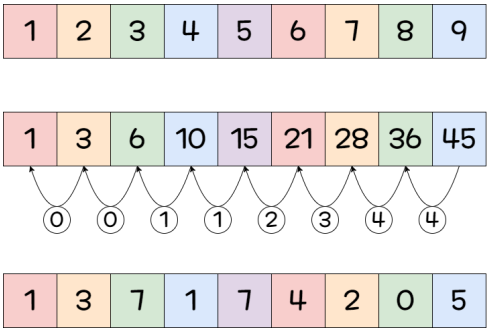
방법 2. 등비수열의 합 활용

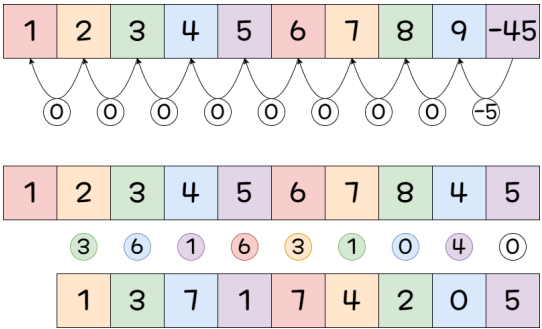
1. 각 숫자마다 곱해야 하는 값은 $\frac{1}{9}(10^n - 1)$ 입니다.
2. 구하려는 답에 9를 곱한 값은 숫자 채워넣기와 받아올림으로 구할 수 있습니다.
3. 이제 이 값을 9로 나누면 됩니다. 직접 나눗셈을 해도 시간복잡도는 $\mathcal{O}(n)$ 입니다.

2I. 큰 수 뒤집기


1 1 1 1 1 1 1 1 1 × 9 = 1 0 0 0 0 0 0 0 0 -1
2 2 2 2 2 2 2 2 × 9 = 2 0 0 0 0 0 0 0 -2
3 3 3 3 3 3 3 × 9 = 3 0 0 0 0 0 0 -3
4 4 4 4 4 4 × 9 = 4 0 0 0 0 0 -4
5 5 5 5 5 × 9 = 5 0 0 0 0 -5
6 6 6 6 × 9 = 6 0 0 0 -6
7 7 7 × 9 = 7 0 0 -7
8 8 × 9 = 8 0 -8
9 × 9 = 9 -9
1 3 6 10 15 21 28 36 45 × 9 = 1 2 3 4 5 6 7 8 9 -45

2I. 큰 수 뒤집기

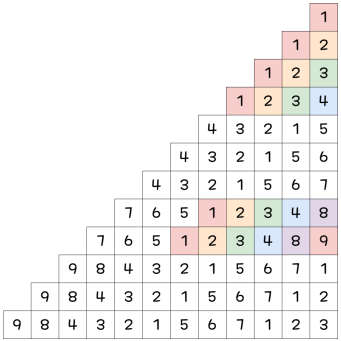

1 2 3 4 5 6 7 8 9
1 3 6 10 15 21 28 36 45
0 0 1 1 2 3 4 4
1 3 7 1 7 4 2 0 5

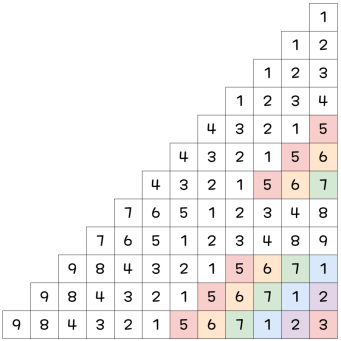

1 2 3 4 5 6 7 8 9 -45
0 0 0 0 0 0 0 0 -5
1 2 3 4 5 6 7 8 4 5
3 6 1 6 3 1 0 4 0
1 3 7 1 7 4 2 0 5

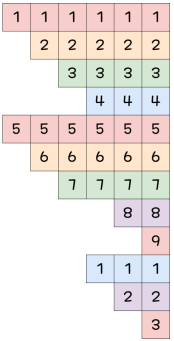
2I. 큰 수 뒤집기

- 뒤집는 연산이 있을 때도 각 숫자가 답에 얼마나 기여하는지 살펴봅니다.
- 뒤집은 횟수가 홀수일 때와 짝수일 때를 나눠서 보면 좋습니다.

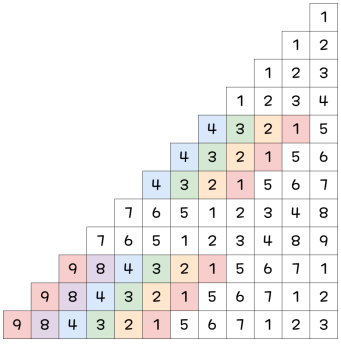
2I. 큰 수 뒤집기

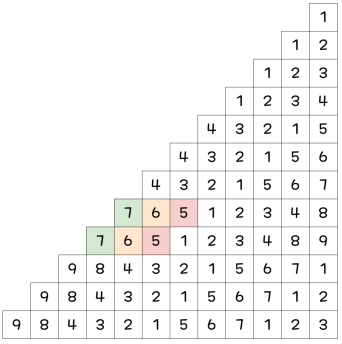


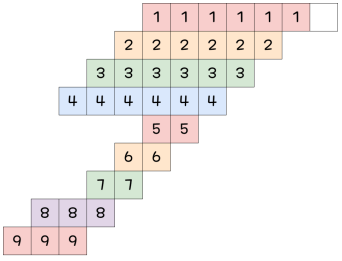




2I. 큰 수 뒤집기


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


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


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

- i번째로 들어간 숫자가 (들어갈 때 기준) 뒤집히지 않은 상태로 더해진 횟수 a_i
- i번째로 들어간 숫자가 (들어갈 때 기준) 뒤집힌 상태로 더해진 횟수 b_i

i번째로 들어간 숫자 x_i가 전체 답에 얼마나 기여할까?

1. 뒤집히지 않은 상태에서 $x_i \times \frac{1}{9} \cdot (10^{a_i} - 1)$
2. 뒤집힌 상태에서 $x_i \times 10^i \cdot \frac{1}{9} \cdot (10^{b_i} - 1)$

- 따라서 **뒤집는 연산이 없었을 때처럼 누적 합이나 등비수열의 합을 이용하여 문제를 풀 수 있습니다.**

2I. 큰 수 뒤집기

- a_i, b_i 는 간단한 dp를 이용해 $\mathcal{O}(n)$ 에 구할 수 있습니다.

$$(a_i, b_i) = \begin{cases} (a_{i+1}+1, b_{i+1}) & \text{(뒤집힌 경우)} \\ (b_{i+1}+1, a_{i+1}) & \text{(뒤집히지 않은 경우)} \end{cases}$$

i	12	11	10	9	8	7	6	5	4	3	2	1
a_i	1	2	3	1	2	4	5	6	3	4	5	6
b_i	0	0	0	3	3	2	2	2	6	6	6	6

전체 시간복잡도 : $\mathcal{O}(n)$

스코어보드 보러 갑시다!

This Page is Intentionally Left Blank

1A. 트리 조각하기

`bfs`, `parametric_search`
출제진 의도 – **Easy**

- 제출 90번, 정답 23명 (정답률 41.82%)
- 처음 푼 사람: **조승현**, 5분
- 출제자: `onjo0127`

1A. 트리 조각하기

- 트리 $T = (V, E)$ 가 주어집니다. T 의 각 정점에 대해 그 정점을 제거해야 하거나 제거하지 않아야 한다는 정보도 주어집니다.
- 원하는 정점들을 골라 폭탄을 설치할 수 있고, 폭탄이 모두 터지고 난 후 폭탄이 설치된 정점과 거리가 p 미만인 정점들은 모두 제거됩니다.
- 폭탄의 세기 p 로 가능한 값 중에서 최댓값을 찾는 문제입니다.

1A. 트리 조각하기

- 세기가 k인 폭탄을 정점 u에 설치한다고 해 봅시다.
- 그렇게 하는 대신에 u, 그리고 u와 이웃한 정점들에 세기가 $k-1$인 폭탄을 설치하면 완전히 같은 효과를 낼 수 있습니다.
- 같은 논리로 세기가 k인 폭탄들로 원하는 정점 집합을 제거하면서 나머지 정점들은 제거하지 않을 수 있다면, 세기가 $k-1$인 폭탄들로도 같은 작업이 가능합니다.
- 귀납적 논리에 의해서 세기가 $k-2, \cdots, 2, 1$인 폭탄들로도 같은 작업을 할 수 있습니다.

1A. 트리 조각하기

- 방금의 논의를 통해서 **파라메트릭 서치**(정답에 대한 이분 탐색)를 사용할 수 있다는 점을 알 수 있습니다.
- 폭탄의 세기가 k일 때 결정 문제를 어떻게 해결할 수 있을까요?

1A. 트리 조각하기

- 제거해야 할 정점들의 집합을 A, 제거하지 않아야 할 정점들을 집합을 B라고 합시다. 문제 조건에 의해 $A \cup B = V$입니다.
- A에 속한 정점 가운데 B로부터 거리가 k 이상 떨어진 정점들의 집합을 R이라고 정의합시다.
- 정의에 의해 R에 속한 정점에는 폭탄을 설치해도 무방합니다. 그렇지 않다면, 폭탄을 설치하면 안 됩니다.
- R에 속한 모든 정점에 폭탄을 설치해도 무방하므로, 그렇게 합니다.
- 폭탄이 터지는 것을 시뮬레이션합니다. 만약 A에 속한 정점 중 제거되지 않은 정점이 있다면 결정 문제의 답은 **불가능**입니다.
- A에 속한 정점들이 모두 제거되었다면 결정 문제의 답은 **가능**입니다.

1A. 트리 조각하기

- B에서부터 Multi Source BFS를 수행하면 $\mathcal{O}(N)$ 시간에 R을 계산할 수 있습니다.
- 마찬가지로 R에서부터 Multi Source BFS를 수행하면 $\mathcal{O}(N)$ 시간에 폭발 과정을 시뮬레이션할 수 있습니다.
- 결정 문제를 $\mathcal{O}(N)$ 시간에 해결할 수 있으므로, 전체 문제를 $\mathcal{O}(N \log N)$ 시간에 해결할 수 있습니다.

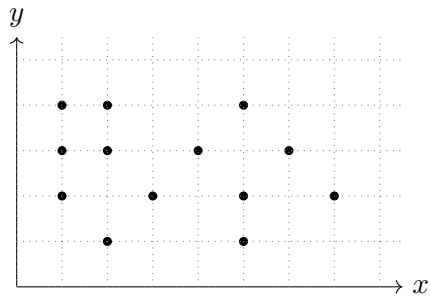
1B. 별 보는 교준이

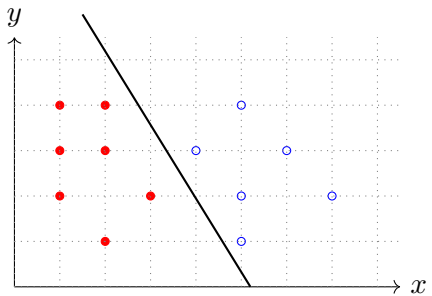
`geometry`
출제진 의도 – **Medium**

- 제출 54번, 정답 17명 (정답률 56.67%)
- 처음 푼 사람: **시제연**, 25분
- 출제자: `yclock`

1B. 별 보는 교준이

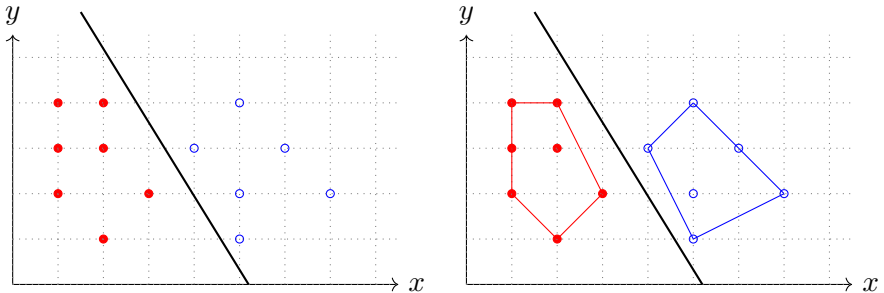
- 이차원 평면에 놓인 N 개의 점을 두 집합 A, B 로 분리하는 경우의 수를 구하는 문제입니다.
- 단, $A \neq \emptyset, B \neq \emptyset$ 이며, 두 집합을 분리하는 직선이 존재해야 합니다.
- A 와 B 에 속하는 점을 각각 속이 찬 빨간 원과 속이 빈 파란 원으로 표현합시다.





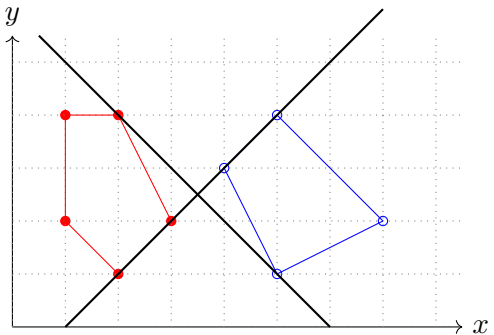
1B. 별 보는 교준이

- 먼저, 두 집합을 분리하는 직선이 존재한다는 것은, 각 집합이 형성하는 볼록 껍질 (Convex hull) 을 분리하는 직선이 있다는 것과 동치입니다.



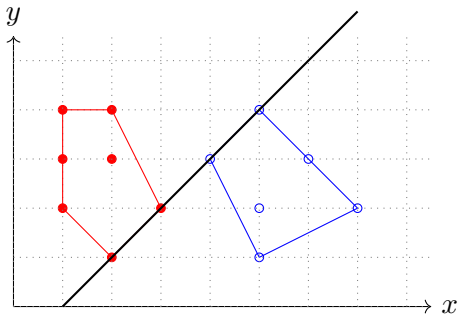
1B. 별 보는 교준이

- 직선으로 분리 가능한 두 볼록 껍질은 두 개의 공통접선을 가집니다.
- 모든 점이 한 직선 위에 있는 경우에는 성립하지 않지만, 나중에 생각합시다.



1B. 별 보는 교준이

- 문제 조건을 만족하는 분할과 하나의 공통접선이 주어졌을 때, "공통접선 상에서 인접한 빨간색 점과 파란색 점의 순서쌍"으로 상태를 나타냅시다.
- 아래의 경우, $((3,2),(4,3))$으로 나타낼 수 있습니다.



1B. 별 보는 교준이

- 어떤 분할과 공통접선의 상태가 (p, q) 라면, 서로 다른 두 점 p, q 를 잇는 선분 위에 다른 어떤 점도 없어야 합니다.
  - 두 점을 지나는 직선 위에서 p 와 q 는 서로 인접하기 때문입니다.
- 문제의 답은, "어떤 분할과 공통접선의 상태" 의 경우의 수의 절반과 같습니다.
  - 각 분할에 대하여 공통접선은 정확하게 두 개 존재하기 때문입니다.
- 서로 다른 두 점 p, q 를 잇는 선분 위에 다른 어떤 점도 없는 순서쌍 (p, q) 의 경우의 수는

$$2 \times \sum_{\text{두 개 이상의 점을 지나는 직선}} (\text{직선 위에 있는 점의 수} - 1)$$

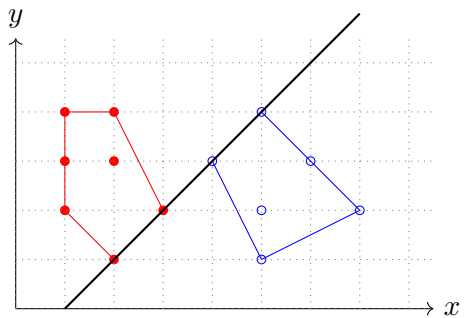
입니다.

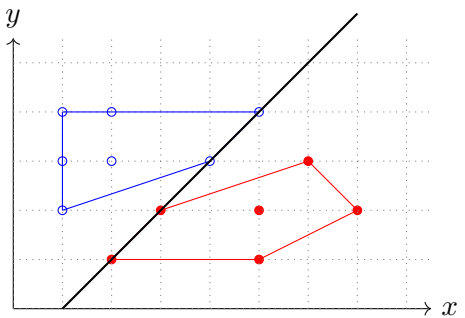
- $d \geq 2$ 개의 점을 지나는 직선에 대하여 총 $2(d-1)$ 개의 (p, q) 순서쌍을 뽑을 수 있습니다.

- 앞에서 서술한 순서쌍 (p, q)에 대하여, (p, q)를 상태로 가지는 분할과 공통접선은 정확하게 두 개 존재합니다.
  - 공통접선은 p, q를 지나는 직선입니다.
  - 공통접선 위의 점의 색을 결정하는 방법은 유일합니다.
  - 공통접선 밖의 점의 색을 결정하는 방법은 두 가지입니다.
  - 이렇게 해서 얻어진 분할은 문제 조건을 만족합니다.

1B. 별 보는 교준이

- 예를 들어, $p = (3, 2), q = (4, 3)$ 라면, (p, q) 를 상태로 가지는 분할과 공통접선은 아래와 같이 두 가지가 있습니다.





- 이 모든 내용을 정리하면 다음과 같습니다:
  - "문제 조건을 만족하는 분할"과 "분할과 공통접선의 상태"의 개수비는 $1:2$
  - "분할과 공통접선의 상태"와 "(p, q) 순서쌍"의 개수비는 $2:1$
  - "(p, q) 순서쌍"의 개수는

$$2 \times \sum_{\text{두 개 이상의 점을 지나는 직선}} (\text{직선 위에 있는 점의 수} - 1)$$

- 따라서 답은 위의 값과 같습니다.
- 위 공식은 모든 점이 한 직선 위에 있어도 성립합니다.

- 서로 다른 두 점을 잡고 이를 지나는 직선을 벡터에 추가합시다.
- $d \geq 2$ 개의 점을 지나는 직선은 벡터에서 총 $d(d-1)$ 번 등장합니다.
- 즉, 벡터에서 n 번 등장하는 직선 위에는 총 $\dfrac{1+\sqrt{1+4n}}{2}$ 개의 점이 있습니다.
- 따라서 벡터를 정렬하면 앞에서 등장한 식을 계산할 수 있습니다.
- (x, y) 와 $(x+\Delta x, y+\Delta y)$ 를 지나는 직선은 최대 공약수 $g = \gcd(\Delta x, \Delta y)$ 에 대하여 튜플 $\left(\dfrac{\Delta x}{g}, \dfrac{\Delta y}{g}, \dfrac{x\Delta y - y\Delta x}{g}\right)$ 로 표현할 수 있습니다.
- 전체 시간 복잡도는 $\mathcal{O}\left(N^2 \lg N\right)$ 입니다.

1C. 큰 수 뒤집기

`math`, `prefix_sum`
출제진 의도 – **Medium**

- 제출 37번, 정답 21명 (정답률 77.78%)
- 처음 푼 사람: **김세빈**, 49분
- 출제자: `16silver`

1D. 여우 국수

`greedy`, `stack`
출제진 의도 – **Hard**

- 제출 23번, 정답 3명 (정답률 42.86%)
- 처음 푼 사람: **시제연**, 101분
- 출제자: `doju`

1D. 여우 국수

- 여우는 사랑입니다.
  - 문제 설정은 일본의 면 요리인 키츠네 우동과 타누키 우동에서 가져왔습니다.
- (대충 $\mathcal{O}(N \log N)$ 때려주고 싶었다는 말)

1D. 여우 국수

- 문제를 좀 더 형식적으로 표현하면
- 알파벳 Y와 N으로 이루어진 문자열이 두 개 주어지고
- 스택 하나를 사용해 첫 번째 문자열을 두 번째 문자열로 바꾸는데
- 첫 번째 문자열의 n 번째 문자가 두 번째 문자열의 첫 번째 문자에 대응되어야 합니다.
  - 즉 처음에는 반드시 n 번 push를 하고 한 번 pop을 해야 합니다.

1D. 여우 국수

- 이 문제는 매개 변수 탐색 (parametric search) 으로 접근하기에 굉장히 좋은 형태입니다.
- 결정 문제로 쉽게 바꿀 수 있고,
  - 처음에 조리만을 n 번 했을 때, 이후 모든 손님을 만족시킬 수 있는가?
- 어떤 n 이 조건을 만족한다면 그보다 작은 모든 값 또한 조건을 만족합니다.
- 이 방향으로 먼저 접근해 봅시다.

1D. 여우 국수

- 장인 여우가 사용할 수 있는 가장 간단한 전략을 생각해 봅시다.
- 만약 지금 진열대에서 꺼낼 수 있는 국수가 맨 앞의 손님과 대응된다면 국수를 내놓습니다.
- 아니면 다음 국수를 조리합니다.

1D. 여우 국수

- 장인 여우가 사용할 수 있는 가장 간단한 전략을 생각해 봅시다.
- 만약 지금 진열대에서 꺼낼 수 있는 국수가 맨 앞의 손님과 대응된다면 국수를 내놓습니다.
- 아니면 다음 국수를 조리합니다.
- 이 전략은 성립할까요?
  - 만약 장인 여우가 적절한 순서로 조리와 판매를 반복해서 모든 손님을 만족시킬 수 있다고 할 때, 이 전략으로도 똑같이 목적을 달성할 수 있을까요?

1D. 여우 국수

- 결론부터 말하면, 이 전략은 성립합니다.
- 따라서 이 전략을 구현하면 $\mathcal{O}(N \log N)$ 풀이를 만들 수 있습니다.
  - (대충 $\mathcal{O}(N \log N)$ 때려주고 싶었다는 말)
- 더 좋은 풀이를 찾기 위해, 이 전략을 더 깊게 이해해 봅시다.

1D. 여우 국수

- 먼저 진열대가 비어 있는 상태에서 시작해서 모든 손님을 만족시키는 것만이 목표인 상황을 생각해 봅시다.
- 이때 이 전략이 성립함을 귀납법으로 보일 수 있습니다.
- 앞으로 편의상 두 문자열을 S와 T라고 부르겠습니다.

1D. 여우 국수

- $N = 1$인 경우는 자명하게 성립합니다.
- $N > 1$일 때, 모든 $1 \leq n \leq N - 1$에 대해 S와 T가 길이 n이고 알파벳 구성이 같다면 전략이 성립한다고 가정합니다.
- 이때 어떤 $1 \leq i < N$에 대해 $S[1 \dots i]$와 $T[1 \dots i]$의 구성이 같다면 이 전략은 성립합니다.
  - $n = i$에서 이 전략이 성립하므로, i번 국수까지 만들면 i번째 손님까지를 모두 만족시키고 진열대는 비게 됩니다.
  - $n = N - i$에서도 마찬가지로 성립하므로, 남은 $N - i$명의 손님들도 모두 만족시킬 수 있습니다.

1D. 여우 국수

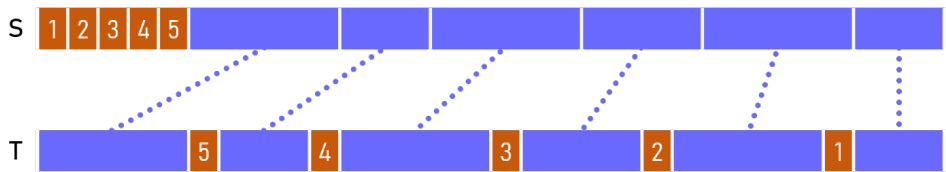
- 그런 i가 없다면, 첫 번째 국수는 중간에 진열대를 빠져나올 수 없습니다.
- 이때 $S[1] = T[N]$임을 보일 수 있습니다.
  - $1 \le i \le N$에 대해 ($S[1 \dots i]$에 포함된 Y의 개수) $-$ ($T[1 \dots i]$에 포함된 Y의 개수)를 생각해 보면 쉽게 관찰할 수 있습니다.
- 따라서 $S[2 \dots N]$과 $T[1 \dots N-1]$은 구성이 같습니다.
- 그러므로 2번부터 N번까지의 국수로 처음 $N-1$명의 손님의 주문이 처리되고, 마지막으로 1번 국수를 마지막 손님에게 내놓게 됩니다.

1D. 여우 국수

- 이제 처음에 n 그릇의 국수가 진열대에 놓여 있는 경우를 생각해 봅시다.
- 여기서는 모든 주문을 만족시키는 어떤 작업 순서가 있을 때, 이 전략이 이 순서에 비해 손해를 보지 않음을 증명합니다.

1D. 여우 국수

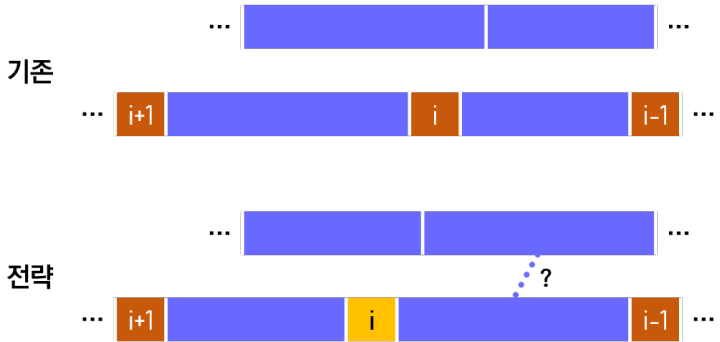
- n 그릇의 국수는 반드시 역순으로 진열대를 빠져나옵니다.
- 아래 그림은 $n = 5$ 의 예시이며, 점선은 두 구간이 서로 대응됨을 의미합니다.



1D. 여우 국수

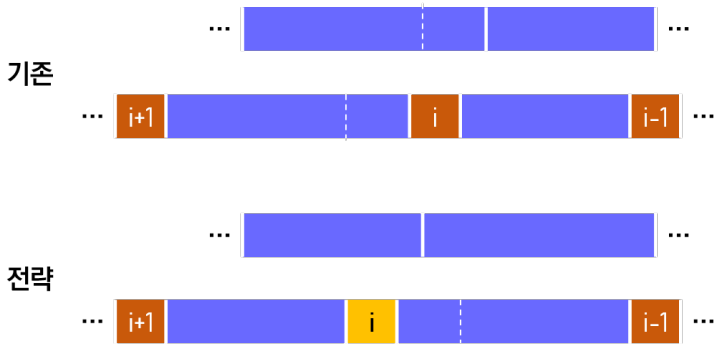
- 이제 이 전략을 사용해서 주문을 처리해 봅시다.
- 만약 어떤 $i \leq n$ 번 국수에서 처음으로 기존의 순서와 차이가 생겼다고 가정합니다.
- 이때 전략상 i 번 국수를 받은 손님은 기존 순서에서 i 번 국수를 받은 손님보다 앞섭니다.
- 이제 기존 순서에서 i 번 국수를 받는 손님을 교체하더라도 이후 순서에 영향이 없음을 보입니다.

1D. 여우 국수



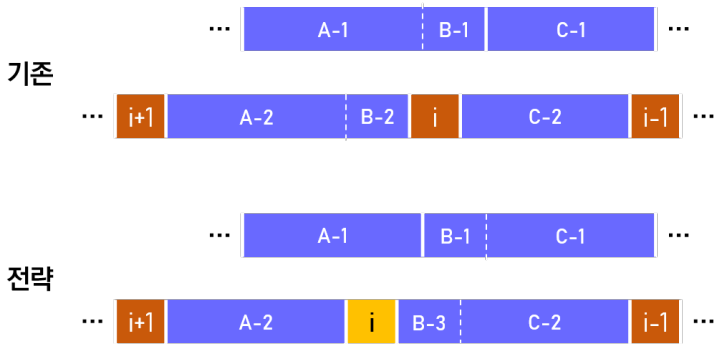
- 위의 점선으로 연결된 두 구역이 대응될 수 있다면, 즉 두 구역의 구성이 같다면 새로운 순서 역시 올바른 순서일 것입니다.

1D. 여우 국수



- 위의 그림에서 점선으로 나뉘어진 세 개의 작은 구역은 모두 구성이 같음을 어렵지 않게 관찰할 수 있습니다.

1D. 여우 국수



- 구역들의 대응 관계를 명시하면 위의 그림과 같습니다.
- 앞에서 제시된 두 구역이 서로 구성이 같음을 확인할 수 있습니다.

1D. 여우 국수

- 이제 증명이 모두 끝났습니다.
- 이 간단한 전략은 어떤 $0 \leq n \leq N$ 이 주어지면, 처음 n 그릇의 국수를 진열대에 쌓아 둔 상태에서 주문을 모두 완료할 수 있는지 판단할 수 있습니다.
- 이제 시간복잡도를 줄이는 일만이 남았습니다.

1D. 여우 국수

- 전략과 증명을 좀더 관찰해 봅시다.
- 이 전략은 진열대에 쌓여 있던 모든 국수를 **가능한 한 앞의** 손님에게 내어 줍니다.
- 따라서 진열대에 n 그릇을 쌓았을 때의 결과와 $n+1$ 그릇을 쌓았을 때의 결과가 완전히 달라집니다.
- 이를 어떻게 극복할 수 있을까요?

1D. 여우 국수

장인 여우는 맨 앞에 있는 손님에게서 눈을 떼고 고개를 들었습니다.
그러자 N 마리의 손님들의 행렬이 눈에 들어왔습니다.

장인 여우는 생각했습니다...

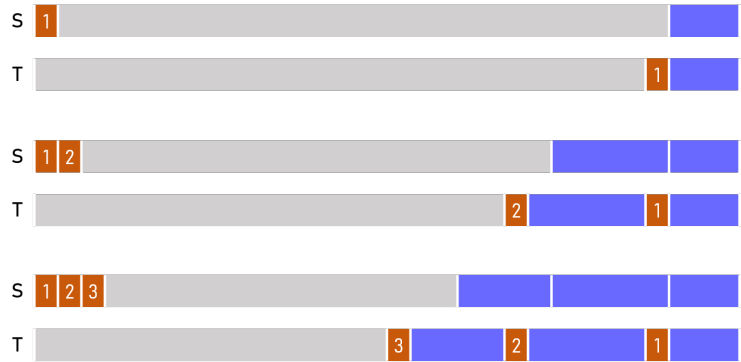
1D. 여우 국수

장인 여우는 맨 앞에 있는 손님에게서 눈을 떼고 고개를 들었습니다.
그러자 N 마리의 손님들의 행렬이 눈에 들어왔습니다.

장인 여우는 생각했습니다…

모든 손님의 순서를 알고 있으므로, 진열대의 국수를 가져갈 손님을 미리 배정할 수 있다!
국수들을 먼저 만든 것부터 **가능한 한 뒤의** 손님에게 배정해 주면 어떨까?

1D. 여우 국수



- 뒤에서부터 배정하면 n을 늘리더라도 기존에 결정된 내용이 변하지 않습니다!

1D. 여우 국수

- 기존 전략의 증명과 완전히 같은 방법으로, 진열대의 국수를 가능한 한 뒤의 손님에게 배정하는 것 역시 타당함을 증명할 수 있습니다.
- n을 하나씩 늘려 나가면, 모든 주문을 처리할 수 있는 가장 큰 n을 $\mathcal{O}(N)$에 구할 수 있습니다.
- 따라서 $\mathcal{O}(N)$에 문제를 해결할 수 있습니다.

1E. 자연수 색칠하기

`constructive`, `number_theory`
출제진 의도 – **Easy**

- 제출 37번, 정답 27명 (정답률 75.00%)
- 처음 푼 사람: **시제연**, 4분
- 출제자: `16silver`

1F. 고슴도치 그래프

`graphs`, `ad_hoc`, `pollard_rho`
출제진 의도 – **Hard**

- 제출 13번, 정답 0명 (정답률 0.00%)
- 처음 푼 사람: -, -분
- 출제자: `kipa00`

1F. 고슴도치 그래프

- 정점이 $V = 10^6$ 개 있는, 연결된 유향 functional graph가 주어집니다.
- 이 functional graph에는 cycle이 유일하게 존재하는데,
  - cycle의 길이는 3 이상이고
  - cycle에 속하지 않은 정점은 cycle과 간선으로 직접 연결되어 있습니다.
- 다음 질의를 $Q = 1\,204$번 할 수 있습니다.
  - 정점 v와 양의 정수 $x \leq 10^{12.4}$를 고르면, v에서 functional graph에 대응하는 함수 f를 x번 적용한 결과 $f^x(v)$를 돌려줍니다.
- cycle의 길이를 알아내야 합니다.
- 위 작업을, $N \leq 10$개의 독립된 functional graph에 대해 수행해야 합니다.

1F. 고슴도치 그래프

- 시간은 아마도 별 문제가 아니니 문제를 해결하는 방법부터 생각해 봅시다.
- 일단 두 가지를 관찰합니다.
  - 서로 다른 정점 번호는 아무 상관이 없습니다. (특정 정점 번호를 다른 정점 번호로 바꿔치기하면 그만입니다.) 다시 말하면, 우리가 정보를 얻어낼 수 있는 방법은 **다른 방법으로 같은 정점에 도달했다**는 정보뿐입니다.
  - 위 사실을 생각하면, 사이클이 아닌 정점은 어느 정점에서부터도 도달이 불가능하므로 아무 쓸모가 없습니다.
- 따라서 아무 정점이나 잡고 V 번 이상을 이동해서 사이클에 있는 정점 하나를 알아내고 나면, 굳이 사이클이 아닐 수 있는 정점을 고르는 모험을 할 이유가 없어집니다. **그 정점에서 도달 가능한 모든 정점은 사이클 안에 있으니까요.**

1F. 고슴도치 그래프

- 질의 **한 번**을 사용해 사이클 안에 있는 정점 v_0를 얻고 나면, 우리가 할 수 있는 일은 뭐가 있을까요?
- 가장 단순한 생각은, v_0부터 아무 길이 x로나 점프를 해서 다시 v_0에 돌아오게 된다면 문제를 해결한 것이나 다름없습니다.
  - 조건을 만족하는 x를 찾은 경우 x의 소인수를 구해서, 각 소인수마다 이분탐색하는 방법으로 소인수의 지수를 모두 결정할 수 있습니다.
- 이렇게 될 확률이 얼마나 될까요? x가 주기의 배수이면 되므로 x를 찾을 확률은 최소 $1/V$이고, 이것을 Q번 반복할 수 있으므로...
  - $Q \approx V/2$라도 $1 - \left(1 - \frac{1}{V}\right)^Q \approx 1 - \frac{1}{\sqrt{e}} \approx 39.3\%$인데, $Q^2 \approx V$라서 가망이 없어 보입니다.

1F. 고슴도치 그래프

- 굳이 v_0 랑 겹쳐야 할 이유는 없죠!
- v_0 에서 x_i 번 이동한 v_i 와 x_j 번 이동한 v_j 에 대해 $v_i = v_j$ 이고 $x_i \neq x_j$ 인 것으로 충분합니다.
- 확률 분석은 상당히 어려운데, **Birthday Paradox**라는 문제로 잘 알려져 있습니다.
  - 생각해 보면 N 명의 사람이 있을 때 쌍의 수가 $\mathcal{O}(N^2)$ 이기 때문에, 생일이 겹치는 것을 독립 시행으로 생각해도 확률 $\approx 1/N^2$ 인 시행이 한 번 성공할 확률은 꽤 높습니다.
- 다음 결과를 사용합니다: $N \gg 1$ 일 때 N 개 중 $c\sqrt{N}$ 번을 (복원 추출로) 뽑았을 때 그중 **하나라도 겹치지 않을 확률**은 $\exp\left(-\frac{c^2}{2}\right)$ 이다.
  - 증명은 다음 장부터 있습니다. 궁금하지 않으신 분은 읽지 않으셔도 좋습니다.

1F. 고슴도치 그래프

- N 개 중 $c\sqrt{N}$ 번을 뽑았을 때, 하나라도 겹치지 않을 확률 $p(N, c)$ 는 다음과 같습니다.

$$p(N,c) = \frac{1}{N^{c\sqrt{N}}} \prod_{1 \le i \le c\sqrt{N}} (N - i + 1).$$

- 위 식의 오른쪽 곱 기호는 $\frac{N!}{(N - c\sqrt{N})!}$ 으로 바꾸어 쓸 수 있습니다.

$$p(N,c) = \frac{N!}{N^{c\sqrt{N}}(N - c\sqrt{N})!}.$$

- $N \gg 1$ 이므로, 스털링 근사 $x! \approx \sqrt{2\pi x} e^{-x} x^x$ 을 $N!$ 과 $(N - c\sqrt{N})!$ 에 모두 적용합니다.

$$
\begin{aligned}
p(N, c) &= \frac{N!}{N^{c\sqrt{N}}(N - c\sqrt{N})!} \\
&\approx \frac{\sqrt{2\pi N}}{\sqrt{2\pi(N - c\sqrt{N})}} \cdot \frac{N^N e^{N - c\sqrt{N}}}{e^N (N - c\sqrt{N})^{N - c\sqrt{N}} N^{c\sqrt{N}}} \\
&= \left(1 - \frac{c}{\sqrt{N}}\right)^{-0.5} \cdot \frac{1}{e^{c\sqrt{N}}} \cdot \left(1 - \frac{c}{\sqrt{N}}\right)^{-(N - c\sqrt{N})} \\
&= \frac{1}{e^{c\sqrt{N}}} \cdot \left(1 - \frac{c}{\sqrt{N}}\right)^{-(N - c\sqrt{N} + 0.5)}.
\end{aligned}
$$

- 양변에 로그를 씌우고 테일러 전개를 이용하면 증명이 완료됩니다.

$$\begin{aligned}
\log p(N,c) &\approx -c\sqrt{N} - \left(N - c\sqrt{N} + \frac{1}{2}\right)\log\left(1 - \frac{c}{\sqrt{N}}\right) \\
&= -c\sqrt{N} + \left(N - c\sqrt{N} + \frac{1}{2}\right)\left(\frac{c}{\sqrt{N}} + \frac{c^2}{2N} + o(N^{-1.5})\right) \\
&= -c\sqrt{N} + c\sqrt{N} - c^2 + \frac{c^2}{2} + o(N^{-0.5}) \\
&\approx -\frac{c^2}{2} \\
\therefore p(N,c) &\approx \exp\left(-\frac{c^2}{2}\right).
\end{aligned}$$

- $Q/\sqrt{V} = 1.204$이므로 **실패할** 확률은 $\exp(1.204^2/2) \approx 48.44\%$ 입니다.
  - **한 그래프에 대한** 성공 확률이 50%를 간신히 넘겼습니다. N번 해서 모두 다 맞기에는 여전히 턱도 없습니다.
- 그래도 이 접근법은 중요한 사실을 일깨워 줍니다: Q개의 질의를 알차게 쓰면, 쌍의 수가 $\mathcal{O}(Q^2)$ 정도이므로 가능성이 있다!

1F. 고슴도치 그래프

- 각 쌍을 모두 잘 쓰는 방법을 생각해 봅시다.
- 무작위로 건너뛰지 않는다면, 몇 칸을 건너뛰든 최악의 경우 별로 상관이 없습니다. 처음에는 무조건 **한 칸씩** 뛰어 봅시다.
  - 주기가 소수인 경우는 세 칸씩 건너뛰는 거나 한 칸씩 건너뛰는 거나 사실상 같습니다.
- 한 칸씩 뛰는 걸 L번 반복해서 $v_0 \to v_1 \to \cdots \to v_L$의 열이 생겼다고 합시다.
- 이런 이후에는 L칸씩 뛰어도 주기가 돌아옴이 보장됩니다!
  - 한 칸씩 뛴 결과를 알고 있는 길이 L의 구간이 있으므로, 아무 데서나 L칸씩 뛰어도 결국에는 이 구간 중 하나와 겹치게 됩니다.

- 이 방법은 deterministic한 방법이므로 질의 제한에만 들어오면 성공합니다.
- 산술기하평균부등식을 이용해 최소의 L을 찾을 수 있습니다.

$$L + \frac{V}{L} \geq 2\sqrt{V} = 2\,000 \quad \text{when} \quad L = \sqrt{V} = 1\,000.$$

- 안타깝게도 질의 제한인 $1\,204$번 안에는 들어오지 못합니다.
- 같은 횟수를 쓸 수 있는 확률적 솔루션의 성공 확률이 $1 - e^{-2} \approx 86.47\%$ 밖에 되지 않는다는 점을 고려하면 큰 발전입니다.

1F. 고슴도치 그래프

- 제한이 $2\,000$이라고 생각하면, 마구잡이로 해 보는 게 쌍의 개수는 훨씬 많습니다.
  - "마구잡이" 솔루션은 $2\,000 \cdot 1\,999/2 = 1\,999\,000$개의 쌍 중 겹치는 것을 찾습니다.
  - deterministic 솔루션은 처음 $L = 1\,000$개끼리와 나중 $L = 1\,000$개끼리의 쌍은 각각 천 개만 고려하므로, $1\,000^2 + 2 \cdot 1\,000 = 1\,002\,000$개의 쌍 중 겹치는 것을 찾습니다.
- 대체 deterministic한 솔루션이 어디서 이점을 가져가기에, 확률 차이가 나는 것일까요?
- 문제는 **길이끼리도** 겹칠 확률이 생각보다 높다는 것입니다.
  - 만일 주기가 7이라면, 2에서 3칸 뛴 것을 해 보았다면 4에서 10칸 뛴 것을 해 볼 필요는 없습니다. 하지만 우리는 주기를 사전에 알지 못하고 마구잡이로 생성하기 때문에, 주기가 V에 가까운 경우 **offset이 겹치는 것이 많이 생성될 확률도 올라갑니다.**
  - deterministic한 솔루션은 주기가 V에 가까워도 겹치는 부분이 아예 없이, 10^6개의 쌍이 1부터 10^6까지의 offset 차를 **고루 생성합니다.**

1F. 고슴도치 그래프

- 이 방법이 가지는 특성을 분석해 봅시다.
- 아래 행렬에서 좌우 차는 1, 상하 차는 $L = \sqrt{V} = 1\,000$ 입니다. 좌우 차가 행에 관계없이 일정하고, 상하 차가 열에 관계없이 일정합니다.

$$\begin{bmatrix} 1 & 2 & \cdots & L \\ L+1 & L+2 & \cdots & 2L \\ \vdots & \vdots & \ddots & \vdots \\ (L-1)L+1 & (L-1)L+2 & \cdots & L^2 \end{bmatrix}$$

- 이 행렬의 좌우 차와 상하 차를 이용해서 원하는 수를 항상 만들어낼 수 있습니다.
- $1 \cdots 1\,1\,L \cdots L$ 은 이 행렬의 역순 좌우 차를 이동하고, 처음 이동 1 을 이동하고, 상하 차를 이동한 것입니다.

1F. 고슴도치 그래프

- 이제 여기에 첫 번째 아이디어를 섞읍시다.
- 우리에게 중요한 것은 **주기 혹은 주기의 배수**를 **겹치지 않게 생성하는 것**입니다.
- 일단 생성해야 할 수의 개수를 줄여 봅시다.
  - 1 부터 $500\,000$ 까지는 두 배 해도 V 보다 작거나 같으므로, $500\,001$ 부터 10^6 까지가 고려된다면 해 보지 않아도 됩니다.
  - 이제 x 제한이 큰 걸 생각합시다. **배수만 생성**하면 되므로, 남은 $500\,000$ 개를 **둘씩 곱해서** $250\,000$ 개의 곱을 아까처럼 배열할 수 있으면 됩니다.
- 수가 $250\,000$ 개로 줄어든다면, 아까처럼 했을 때 수의 개수 걱정은 없습니다. ($2\sqrt{250\,000} = 1\,000$ 이므로.) 근데 그게 정말 가능할까요?

- 이렇게 하려고 노력하다 보면 배치를 찾을 수 있습니다.

$$\begin{bmatrix} \left(\frac{V}{2}+1\right)\left(\frac{V}{2}+L\right) & \left(\frac{V}{2}+2\right)\left(\frac{V}{2}+L-1\right) & \cdots & \left(\frac{V}{2}+\frac{L}{2}\right)\left(\frac{V}{2}+\frac{L}{2}+1\right) \\ \left(\frac{V}{2}+L+1\right)\left(\frac{V}{2}+2L\right) & \left(\frac{V}{2}+L+2\right)\left(\frac{V}{2}+2L-1\right) & \cdots & \left(\frac{V}{2}+\frac{3L}{2}\right)\left(\frac{V}{2}+\frac{3L}{2}+1\right) \\ \vdots & \vdots & \ddots & \vdots \\ \left(\frac{V}{2}+\left(\frac{L}{2}-1\right)L+1\right)\left(\frac{V}{2}+\frac{L^2}{2}\right) & \left(\frac{V}{2}+\left(\frac{L}{2}-1\right)L+2\right)\left(\frac{V}{2}+\frac{L^2}{2}-1\right) & \cdots & \left(\frac{V}{2}+\left(\frac{L}{2}-1\right)L+\frac{L}{2}\right)\left(\frac{V}{2}+\left(\frac{L}{2}-1\right)L+\frac{L}{2}+1\right) \end{bmatrix}$$

- 접근 방식은 사람마다 굉장히 많은 차이가 있을 수 있습니다.

1F. 고슴도치 그래프

$$\begin{bmatrix} \left(\frac{V}{2}+1\right)\left(\frac{V}{2}+L\right) & \left(\frac{V}{2}+2\right)\left(\frac{V}{2}+L-1\right) & \cdots & \left(\frac{V}{2}+\frac{L}{2}\right)\left(\frac{V}{2}+\frac{L}{2}+1\right) \\ \left(\frac{V}{2}+L+1\right)\left(\frac{V}{2}+2L\right) & \left(\frac{V}{2}+L+2\right)\left(\frac{V}{2}+2L-1\right) & \cdots & \left(\frac{V}{2}+\frac{3L}{2}\right)\left(\frac{V}{2}+\frac{3L}{2}+1\right) \\ \vdots & \vdots & \ddots & \vdots \\ \left(\frac{V}{2}+\left(\frac{L}{2}-1\right)L+1\right)\left(\frac{V}{2}+\frac{L^2}{2}\right) & \left(\frac{V}{2}+\left(\frac{L}{2}-1\right)L+2\right)\left(\frac{V}{2}+\frac{L^2}{2}-1\right) & \cdots & \left(\frac{V}{2}+\left(\frac{L}{2}-1\right)L+\frac{L}{2}\right)\left(\frac{V}{2}+\left(\frac{L}{2}-1\right)L+\frac{L}{2}+1\right) \end{bmatrix}$$

- i번째 행과 그 아래 행의 차이는 열 j에 **관계없이** $(LV+2iL^2)$입니다:

$$\left(\frac{V}{2}+iL+j\right)\left(\frac{V}{2}+(i+1)L-j+1\right)-\left(\frac{V}{2}+(i-1)L+j\right)\left(\frac{V}{2}+iL-j+1\right)=LV+2iL^2.$$

- 이 값은 10^9 scale입니다.

1F. 고슴도치 그래프

$$\begin{bmatrix} \left(\frac{V}{2}+1\right)\left(\frac{V}{2}+L\right) & \left(\frac{V}{2}+2\right)\left(\frac{V}{2}+L-1\right) & \cdots & \left(\frac{V}{2}+\frac{L}{2}\right)\left(\frac{V}{2}+\frac{L}{2}+1\right) \\ \left(\frac{V}{2}+L+1\right)\left(\frac{V}{2}+2L\right) & \left(\frac{V}{2}+L+2\right)\left(\frac{V}{2}+2L-1\right) & \cdots & \left(\frac{V}{2}+\frac{3L}{2}\right)\left(\frac{V}{2}+\frac{3L}{2}+1\right) \\ \vdots & \vdots & \ddots & \vdots \\ \left(\frac{V}{2}+\left(\frac{L}{2}-1\right)L+1\right)\left(\frac{V}{2}+\frac{L^2}{2}\right) & \left(\frac{V}{2}+\left(\frac{L}{2}-1\right)L+2\right)\left(\frac{V}{2}+\frac{L^2}{2}-1\right) & \cdots & \left(\frac{V}{2}+\left(\frac{L}{2}-1\right)L+\frac{L}{2}\right)\left(\frac{V}{2}+\left(\frac{L}{2}-1\right)L+\frac{L}{2}+1\right) \end{bmatrix}$$

- j 번째 열과 그 오른쪽 열의 차이는 행 i에 **관계없이** $(L-2j)$ 입니다:

$$\left(\frac{V}{2}+(i-1)L+(j+1)\right)\left(\frac{V}{2}+iL-j\right)-\left(\frac{V}{2}+(i-1)L+j\right)\left(\frac{V}{2}+iL-j+1\right)=L-2j.$$

- 이 값은 10^3 scale입니다.

- 즉, 다음과 같이 하면 $1\,000$ 번에 모든 수를 커버할 수 있습니다!
  1. $(L - 2j)$ 를 j 역순으로 이동한다: 즉 $2,\ 4,\ \cdots,\ 998$.
  2. 첫 수 $\left(\frac{V}{2} + 1\right)\left(\frac{V}{2} + L\right) = 250\,500\,501\,000$ 만큼 이동한다.
  3. $(LV + 2iL^2)$ 를 i 순서대로 이동한다: 즉 $1\,002\,000\,000,\ 1\,004\,000\,000\ \cdots$

1F. 고슴도치 그래프

- 시간에 대해 생각해 봅시다.
- 이동하고 겹치는 것을 확인하는 것은 상수 시간에 할 수 있으므로 큰 문제가 아닙니다.
- 소인수분해를 하는 것은 **큰 문제이지만**, 나중에 생각해 봅시다.
- N 개의 테스트 케이스를 처리하기 위한 초기화를 어떻게 시행해야 할까요?
  - 단순히 초기화하는 것으로는 초기화 시간이 $\mathcal{O}(VN)$ 으로 살짝 아슬아슬합니다...

1F. 고슴도치 그래프

- 메모리 참조가 $\mathcal{O}(Q)$ 번밖에 안 일어나니 참조한 메모리만 초기화하는 전략을 이용해 봅시다.
- vector에 참조한 인덱스를 모두 저장해 둡니다.
- 초기화할 때는, vector에 저장된 인덱스를 모두 초기화하고 vector를 비우면 됩니다.
- 참조한 것만 초기화하므로, 초기화 시간이 amortized $\mathcal{O}(1)$ 이 되었습니다.
- 참고로, 인터랙터 역시 그래프를 전부 받으면 $\mathcal{O}(VN)$ 의 시간이 걸리기 때문에, 그래프를 시드(seed) 의 형태로 표현해 두고 질의가 올 때마다 그래프의 정점을 발견해 나가는 방식으로 구현되어 있습니다.

1F. 고슴도치 그래프

- 소인수분해는 어떻게 하는 게 좋을까요?
- 우리가 구하는 값들은 표에 있는 값이고, 이 값은 굉장히 가까이 있는 두 수의 곱 ab 입니다.
  - a 와 b 는 **주기**이기 때문에 V 보다 작습니다.
  - a 와 b 를 구할 수 있다면, 에라토스테네스의 체 등을 활용한 $\mathcal{O}(V \log \log V)$ 의 전처리로 $\mathcal{O}(\log V)$ 만에 a 와 b 의 소인수분해를 모두 구할 수 있습니다.
- 즉, $4ab = (a+b)^2 - (a-b)^2$ 에서 $(a-b)^2$ 이 굉장히 작습니다.
- 모든 가능한 $(a+b)^2$ 을 작은 것부터 해 보면 거의 상수 시간에 $4ab$ 를 두 개의 짝수 $2a, 2b$ 로 쪼갤 수 있습니다.
  - 이렇게 구한 수를 각각 2 로 나누어 줄 필요가 있습니다.

1F. 고슴도치 그래프

- 질의 횟수는 총 $1\,053$ 회를 넘지 않으므로 상당히 넉넉하게 돌아갑니다.
  - 먼저 사이클에 집어넣는 것 1 회
  - 표에 있는 수들의 차를 출력하는 것이 $1\,000$ 회
  - 소인수분해하는 수는 2^{40} 을 넘지 않고 각 소인수를 binary search 1 회 할 때마다 수의 후보군이 절반으로 줄어드므로 40 회를 넘지 않아야 하나, 각 소인수가 항상 완전히 절반으로 줄일 수 있는 것은 아닐 수 있으므로 소인수 당 1 회씩 추가 (제한을 지키는 최대 소인수의 개수는 12 개) 하여 52 회
- 제한을 넉넉하게 잡은 이유는
  - **12월 4일이 출제자의 생일이기도 하고**
  - 비슷한 수준의 다른 풀이 (각 수의 자승을 곱해서 확인하거나 하는 풀이) 를 빡빡하게나마 통과시켜 주고 싶었기 때문입니다. 문제가 풀리길 바랐어요.

1F. 고슴도치 그래프

- 시간을 고려해 봅시다.
- 아까 논의한 대로 초기화 시간은 고려할 필요가 없습니다.
- 겹치는 걸 확인하는 Q 번의 질의 이후, 상수 시간에 절반으로 쪼개고, 시간 $\mathcal{O}(\log V)$ 의 소인수분해를 두 번 한 다음, $\mathcal{O}(\log V)$ 번의 binary search 시간이 들어갑니다.
- 총 시간은 전처리 $\mathcal{O}(V \log \log V)$ 에 그래프 당 $\mathcal{O}(Q + \log V)$ 이므로, $\mathcal{O}(V \log \log V + N(Q + \log V))$ 입니다.

- 소인수분해하는 수의 성질 분석 없이 pollard's ρ를 이용할 수도 있는데 이 경우는 expected $\mathcal{O}(\sqrt{V})$의 소인수분해 시간이 걸리게 됩니다.
  - 우연의 일치이지만 pollard's ρ 역시 birthday paradox에 기반을 둔 알고리즘입니다.
- 이 경우 총 시간은 전처리 없이 $\mathcal{O}(N(Q+\sqrt{V}))$입니다. 이렇게 해도 무난하게 통과할 수 있습니다.

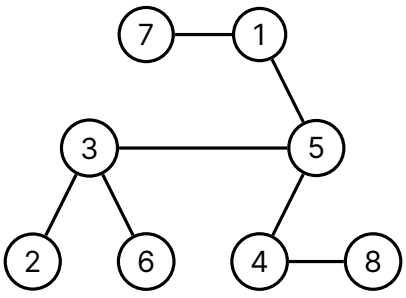
1G. 데칼코마니 트리

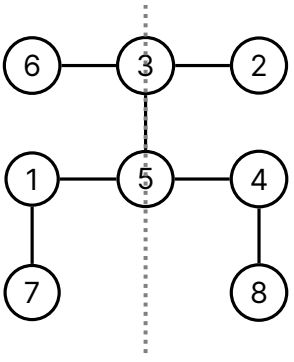
`tree_isomorphism`
출제진 의도 – **Hard**

- 제출 26번, 정답 1명 (정답률 16.67%)
- 처음 푼 사람: **강태규**, 243분
- 출제자: `yclock`

1G. 데칼코마니 트리

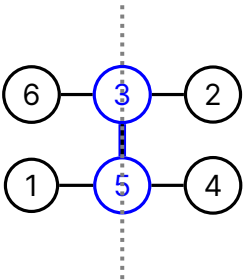
- 트리를 선대칭이 되도록 이차원 평면에 임베딩이 가능한지 판별하는 문제입니다.

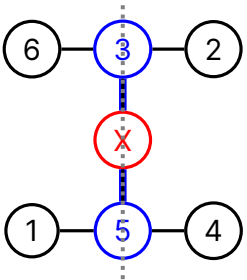




1G. 데칼코마니 트리

- 트리가 두 개의 센트로이드를 가진다면, 둘 사이에 새로운 정점을 하나 추가해도 답에 영향을 주지 않습니다.
- 이때, 새로운 정점은 유일한 센트로이드가 됩니다.



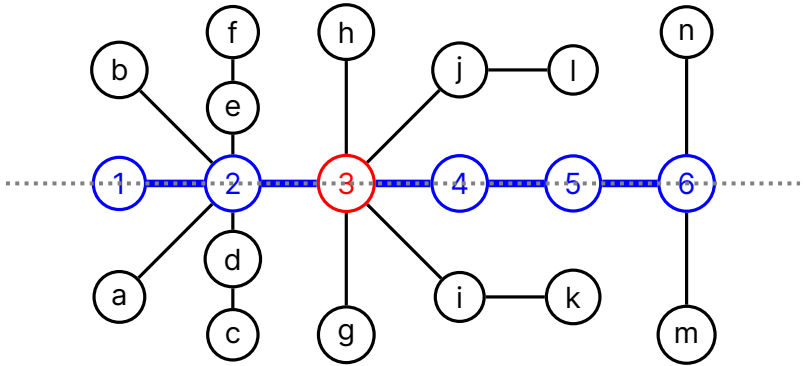


1G. 데칼코마니 트리

- 이제, 트리가 유일한 센트로이드를 가진다고 합시다.
- 데칼코마니 트리의 센트로이드는 항상 대칭축 위에 놓입니다.
  - 그렇지 않다면, 센트로이드와 대칭을 이루는 정점 또한 센트로이드가 되어야 합니다.

1G. 데칼코마니 트리

- 대칭축 위에 올라가는 정점은 하나의 단순 경로를 이룹니다.



1G. 데칼코마니 트리

- 유일한 센트로이드 r 을 루트로 하여, 루트가 있는 트리를 만듭시다.
- 정점 v 를 루트로 가지는 부트리를 T_v 로 부릅시다.
- 두 루트가 있는 트리 T_u, T_v 가 동형 (Isomorphism) 이라면 $T_u \equiv T_v$ 로 나타냅시다.

1G. 데칼코마니 트리

- 정점 v가 대칭축 위에 놓여 있다고 합시다.
- 정점 v의 자식 정점이 $u_1, u_2, \cdots, u_K$라고 합시다.
- 아직 매칭이 안 된 두 자식 정점 u_i, u_j가 있어 $T_{u_i} \equiv T_{u_j}$라고 합시다.
  - 이 경우, T_{u_i}와 T_{u_j}는 대칭축 위에 올라가지 않으며, 서로 대응됩니다.
- 위의 매칭 작업을 최대한 많이 수행합시다.
  - 모든 자식 정점을 대응시켜 줄 수 있다면, 선대칭을 이루도록 T_v를 임베딩할 수 있습니다.
  - 단 하나의 자식 정점 u만 매칭이 안 되었다면, u는 반드시 대칭축 위에 올라가야 하며, T_u를 임베딩할 수 있는지 재귀적으로 확인해야 합니다.
  - 예외적으로, 루트 r은 매칭이 안 된 자식 정점을 두 개까지 허용합니다.
  - 그 외의 경우, T_v는 데칼코마니 트리가 아닙니다.

- 따라서, 두 부트리가 동형인지 빠르게 판별할 수 있어야 합니다.
- 가장 쉬운 방법은 부트리의 해시 (Hash) 를 동형 판별에 이용하는 것입니다.
- 부트리 T_i 의 해시 H_i 를 모두 계산할 수 있다면, 앞에서 자식 정점 $u_1, u_2, \cdots, u_K$ 의 매칭 작업을 단순하게 $H_{u_1}, H_{u_2}, \cdots, H_{u_K}$ 를 정렬함으로써 구현할 수 있습니다.
- 그러면 해시 H_i 를 어떻게 계산할 수 있을까요?

1G. 데칼코마니 트리

- 아주 큰 수 X 에 대하여, 범위 $[1, X)$ 에서 N 개의 정수 $P_1, P_2, \cdots, P_N$ 을 랜덤하게 뽑습니다.
- 정점 하나로 이루어진 부트리의 해시는 1 로 정의합니다.
- 부트리 T_v 의 해시 H_v 는 다음과 같이 계산합니다.
  - 정점 v 의 자식 정점 $u_1, u_2, \cdots, u_K$ 를 $H_{u_1} \leq H_{u_2} \leq \cdots \leq H_{u_K}$ 가 되도록 정렬합니다.
  - $H_v := \left(\sum_{i=1}^{K} H_{u_i} \times P_i \right) \mod X$
- N 개의 부트리의 해시를 $\mathcal{O}(N \lg N)$ 에 계산할 수 있으며, 따라서 전체 문제를 $\mathcal{O}(N \lg N)$ 에 해결할 수 있습니다.

1G. 데칼코마니 트리

- 이 문제를 $\mathcal{O}(N)$에 결정론적으로 해결하는 방법 또한 존재합니다.
- 데칼코마니 트리를 선대칭이 되도록 정수점 위에 임베딩하는 것 또한 가능합니다.
- 이 문제는 선인장 그래프여도 선형에 해결할 수 있습니다.

1H. RMQ

`segtree`
출제진 의도 – **Challenging**

- 제출 1번, 정답 1명 (정답률 100.00%)
- 처음 푼 사람: **신승원**, 148분
- 출제자: `moonrabbit2`

- 편의상 $l > r$ 일 때 $\mathrm{RMQ}(l, r) = 0$ 으로 정의합시다.
- 이제 각 쿼리는 $l \le i \le r, s \le j \le e$ 인 모든 (i, j) 쌍에 대해 $\mathrm{RMQ}(i, j)$ 의 합을 구하는 쿼리입니다.

1H. RMQ

- 2차원 직사각형 영역에 대한 합 쿼리이므로 부분합을 이용합시다.
- $l \le i \le r, 1 \le j \le x$인 모든 (i, j) 쌍에 대해 $\mathrm{RMQ}(i, j)$의 합을 구할 수 있다고 합시다.
- $x = e$일 때의 값에서 $x = s - 1$일 때의 값을 빼면 쿼리에 대한 답을 구할 수 있습니다.
- 이제 각 (l, r, s, e) 쿼리를 (l, r, x) 쿼리 2개로 분해할 수 있습니다.

- 이제 x 를 1 씩 증가시키면서, 해당 x 에 대한 쿼리들을 해결해 봅시다.
- 각 i 마다 $\text{RMQ}(i, 1) + \cdots + \text{RMQ}(i, x)$ 을 잘 관리한다면, 세그먼트 트리 등으로 (l, r) 쿼리를 통해 구간합을 구해 (l, r, x) 쿼리를 해결할 수 있을 것입니다.

1H. RMQ

- 우선 각 i마다 $\mathrm{RMQ}(i, x)$를 구하는 법을 알아봅시다. 물론 그냥 구간 최솟값 쿼리, 구간 최댓값 쿼리를 이용해 구할 수도 있겠지만 이대로는 $\mathrm{RMQ}(i, 1) + \cdots + \mathrm{RMQ}(i, x)$를 구하기는 어려워 보입니다.
- $\mathrm{RMQ}(i, x)$의 값이 $\mathrm{RMQ}(i, x-1)$의 값과 다르다면 A_x가 $[i, x]$ 구간에서 최솟값이거나 최대값이라는 사실을 이용합니다.
- $\mathrm{RMQ}(i, x-1)$과 $\mathrm{RMQ}(i, x)$가 다른 i들을 빠르게 구하고, 값들을 업데이트하면 될 것 같습니다.

1H. RMQ

- 어떤 i에 대해 $\min(A_i, \cdots, A_x) = A_j$인 j들을 생각해보면, j가 증가할수록 A_j 값이 증가합니다. 해당 j들은 스택을 이용해 관리할 수 있습니다.
- 스택의 top j에 대해 $A_j > A_x$일 동안 스택에서 pop을 한 후, x를 push하면 됩니다.
- pop을 했다는 것은 해당 j가 관리하던 i들에 대해, 이제 $\min(A_i, \cdots, A_x) = A_x$라는 뜻입니다. $\mathrm{RMQ}(i, x-1)$에 $\frac{A_x}{A_j}$를 곱해 $\mathrm{RMQ}(i, x)$을 구할 수 있습니다.
- 해당 i들은 스택의 크기가 1이었다면 $[1, j]$, 그렇지 않다면 스택의 다음 top j'에 대해 $[j'+1, j]$로 구간을 이룹니다. 따라서 모듈로 역원을 이용하면 구간에 어떤 값을 곱하는 쿼리로 이 과정을 처리할 수 있습니다.
- 마지막 push는 $\min(A_x, \cdots, A_x) = A_x$라는 뜻입니다.
- $\max$에 대해서도 비슷하게, 스택을 이용해 $\mathrm{RMQ}(i, x)$들을 구할 수 있습니다.

1H. RMQ

- 이제 $i < x$인 i들에 대해 $\mathrm{RMQ}(i, x-1)$에서 $\mathrm{RMQ}(i, x)$를 구할 수 있습니다!
- $\mathrm{RMQ}(x, x)$는 A_x^2입니다. 이 경우만 따로 처리하면 됩니다.
- 구간에 어떤 값을 곱하는 쿼리는 lazy propagation을 이용한 세그먼트 트리를 이용해 $\mathcal{O}(\log N)$에 처리할 수 있습니다.
- 모듈러 역원은 $\mathcal{O}(\log P)$에 구할 수 있습니다. $(P = 10^9 + 7)$
- 각 스택에 대해, 각 x는 한 번 push되고 최대 한 번 pop됩니다. 따라서 쿼리는 $\mathcal{O}(N)$개입니다.
- 따라서 시간복잡도 $\mathcal{O}(N \log N + N \log P)$에 $\mathrm{RMQ}(i, x)$들을 구할 수 있습니다.

1H. RMQ

- 이제 $l \le i \le r$ 인 i 에 대해 $\mathrm{RMQ}(i, 1) + \cdots + \mathrm{RMQ}(i, x)$ 를 구해야 합니다.
- 처음에는 모든 값이 0 인 길이 N 의 수열 v 와 s 에 대해 다음과 같은 쿼리들을 처리할 수 있다면 가능합니다.
  - $[l, r]$ 과 x 에 대해, $l \le i \le r$ 인 모든 i 에 $v_i = v_i \times x$
  - i 와 x 에 대해, $v_i = x$
  - $1 \le i \le N$ 인 모든 i 에 $s_i = s_i + v_i$
  - $[l, r]$ 에 대해, $s_l + \cdots + s_r$ 을 구함
- 기존 lazy propagation 세그먼트 트리에서 추가적으로 3, 4 번 쿼리를 처리할 수 있게 수정하면 될 것 같습니다.

1H. RMQ

- 우선 기존 세그먼트 트리의 구조를 살펴봅시다.
- 세그먼트 트리의 각 노드 i에 대해, 두 값 V_i를 노드 i의 구간의 v_i의 합으로 정의합시다.
- 세그먼트 트리의 각 노드 i에 대해, lazy값 $L_{1,i}$를 정의합니다. 처음에 모든 $L_{1,i} = 1$입니다.
- lazy값을 propagate하는 과정에서, V_i를 다음과 같이 바꿀 것입니다.
  - $V_i = V_i \times L_{1,i}$
- 이후 노드 i의 각 자식 노드 j에 대해, $L_{1,j}$를 다음과 같이 바꿀 것입니다.
  - $L_{1,j} = L_{1,j} \times L_{1,i}$
- 이후 $L_{1,i} = 1$로 초기화합니다.

- 이제 s_i 들의 합 S_i 도 저장한다고 합시다. V_i 의 값이 propagate 과정에서 변하면서 이 값은 어떻게 변할까요?
- $L_{1,i}$ 는 여러 업데이트 쿼리들이 합쳐진 결과이니, V_i 와 $L_{1,i}$ 만을 가지고 변화한 값을 표현하긴 어려워 보입니다.
- 다만, S_i 에 더해지는 값은 쿼리 순서대로 $V_i, \cdots, V_i, aV_i, \cdots, aV_i, abV_i \cdots, abV_i, \cdots$ 꼴이었을 것입니다. 이 사실을 이용합시다. $L_{2,i}$ 를 정의해, propagate 과정에서 S_i 에 $V_i \times L_{2,i}$ 를 더한다고 합시다. $L_{2,i}$ 는 0 으로 초기화해야 합니다.

1H. RMQ

- $L_{1,j} = L_{1,j} \times L_{1,i}$ 과정에서, V_j 에 어떤 값을 곱하는 두 업데이트 쿼리가 합쳐집니다.
- 이때, $L_{1,j}$, $L_{2,j}$ 에 해당하는 쿼리들은 $L_{1,i}$, $L_{2,i}$ 에 해당하는 쿼리들보다 먼저 행해진 쿼리들입니다.
- $L_{1,j}$, $L_{2,j}$ 가 새로 초기화된 값인 경우 이 논리가 틀릴 수도 있지만, 이 경우 그냥 쿼리가 자식으로 내려갔다고 생각하면 됩니다.
- 이 사실을 이용하면 $L_{2,i}$ 값도 자식으로 전파할 수 있습니다. V_j 값이 아니라 $L_{1,j}V_j$ 값에 $L_{2,i}$ 를 적용한다고 생각하면 됩니다. 즉, $L_{2,j} = L_{2,j} + L_{1,j} \times L_{2,i}$ 입니다. 이를 먼저 처리하고 $L_{1,j} = L_{1,j} \times L_{1,i}$ 를 처리해야 함에 유의합시다.

1H. RMQ

- 결국, 다음과 같은 세그먼트 트리를 이용해 쿼리들을 처리할 수 있습니다.
- 세그먼트 트리의 각 노드 i에 대해, 두 값 V_i와 S_i를 각각 노드 i의 구간의 v_i의 합, s_i의 합으로 정의합시다.
- 세그먼트 트리의 각 노드 i에 대해, 두 lazy값 $L_{1,i}$와 $L_{2,i}$를 정의합니다.
- lazy값을 propagate하는 과정에서, V_i와 S_i를 다음과 같이 바꿀 것입니다.
  - $S_i = S_i + V_i \times L_{2,i}$
  - $V_i = V_i \times L_{1,i}$
- 이후 노드 i의 각 자식 노드 j에 대해, $L_{1,j}$와 $L_{2,j}$를 다음과 같이 바꿀 것입니다.
  - $L_{2,j} = L_{2,j} + L_{1,j} \times L_{2,i}$
  - $L_{1,j} = L_{1,j} \times L_{1,i}$
- 이후 $L_{1,i} = 1, L_{2,i} = 0$으로 초기화합니다.

1H. RMQ

- 모든 쿼리들을 해당 세그먼트 트리를 이용해 $\mathcal{O}(\log N)$에 처리할 수 있습니다.
- 따라서 $\mathcal{O}(N \log N + N \log P + Q \log N)$에 문제를 해결할 수 있습니다.
- 각 세그먼트 트리 노드마다, 가장 최근에 방문한 시점 t를 저장하는 식으로 해결하는 방법도 있습니다. 결과적으로는 큰 차이가 없는 방법입니다.
- 사용하는 세그먼트 트리만 놓고 보면 수열과 쿼리 13 등과 굉장히 비슷합니다. 다만 생각하기는 훨씬 더 힘든 것 같습니다.

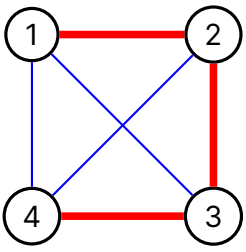
11. 두 트리

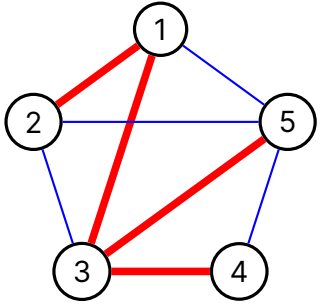
`matroid`
출제진 의도 – **Challenging**

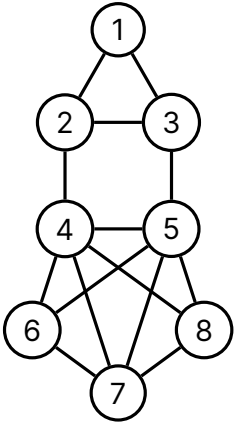
- 제출 8번, 정답 0명 (정답률 0.00%)
- 처음 푼 사람: -, -분
- 출제자: `yclock`

1I. 두 트리

- $2(N-1)$ 개의 간선을 분할하여 각각 트리를 이루는 것이 가능한지 판별하는 문제입니다.







- 간선 집합을 E라고 합시다.
- 부분집합 $F \subseteq E$가 포레스트를 이루면 F를 독립집합으로 정의하는 매트로이드 $\mathcal{M}_1(E, \mathcal{I}_1)$과, $E \setminus F$가 연결되어 있으면 F를 독립집합으로 정의하는 매트로이드 $\mathcal{M}_2(E, \mathcal{I}_2)$를 생각합니다.
- $I \in \mathcal{I}_1 \cap \mathcal{I}_2$를 만족하는 최대 크기의 공통 독립 집합 I가 $|I| = N - 1$을 만족한다면, 분할이 가능합니다.
- 이러한 문제를 매트로이드 교차 문제 (Matroid Intersection Problem) 라고 부릅니다.
  - 여기까지만 관찰해도 $\mathcal{O}\left(N^3\right)$ 풀이가 나옵니다.

1I. 두 트리

- 매트로이드 교차를 명시적으로 구현하지 않는 효율적인 알고리즘은 다음과 같습니다.
- 두 개의 포레스트 F_0, F_1 를 관리합니다.
- 입력으로 주어진 간선 e 를 차례대로 보면서, $F_0 \cup F_1$ 에 e 를 추가합니다.
  - 이때, F_0 과 F_1 가 항상 포레스트가 되도록 잘 관리해야 합니다.
  - 모든 간선을 $F_0 \cup F_1$ 에 추가할 수 있어야만 분할이 가능합니다.

- $F_0 \cup F_1$ 에 간선 e_0 을 추가하는 방법에 대해 알아보기 전에, Augmenting 간선열 $[e_0, e_1, e_2, \cdots, e_k]$ 를 정의합시다.

Definition

서로 다른 간선으로 이루어진 간선열 $[e_0, e_1, e_2, \cdots, e_k]$ 가 다음을 모두 만족하면, 이를 Augmenting 간선열이라 합니다.

- $e_0 \notin F_0 \cup F_1$
- $e_i \in F_{(i \mod 2)}(e_{i-1}) \quad (1 \le i \le k)$
- $F_{(k+1 \mod 2)}(e_k) = \emptyset$
- $e_j \notin F_{(i+1 \mod 2)}(e_i) \quad (i+1 < j)$

- 여기서, $F_i(e)$ 는 간선 $e = \{u, v\}$ 에 대하여, F_i 상에서 u 에서 v 로 가는 경로 상의 간선열을 의미합니다. 그러한 경로가 없다면 $\emptyset$ 으로 정의합니다.

- Augmenting 간선열 $[e_0, e_1, e_2, \cdots, e_k]$ 가 주어지면, 다음과 같은 방법으로 $F_0 \cup F_1$ 에 간선 e_0 을 추가할 수 있습니다.
  - e_0 을 F_1 에 추가합니다.
  - e_i 를 $F_{(i \mod 2)}$ 에서 제거하고 $F_{(i+1 \mod 2)}$ 에 추가합니다. $(1 \leq i \leq k)$
- Augmenting 간선열의 마지막 조건 때문에, 위의 작업을 수행한 후에도 F_0 과 F_1 에는 사이클이 생기지 않습니다.
  - 마지막 조건을 없애면, F_0 과 F_1 이 포레스트가 됨을 보장할 수 없습니다.

- 따라서, 주어진 e_0 에 대하여 Augmenting 간선열을 찾을 수 있다면, 문제를 해결할 수 있습니다.
- 이는 BFS와 유사한 방법으로 효율적으로 찾을 수 있습니다.

1I. 두 트리

- 먼저, Augmenting 간선열의 두 번째 조건만 고려합시다.
- Augmenting 간선열에서 간선 e' 다음에 간선 e가 올 수 있다면, e를 e'으로 라벨링합시다.
- 다음과 같은 방법으로 라벨링 작업을 하면서 Augmenting 간선열을 찾을 수 있습니다.
  - 큐에 간선 e_0을 넣습니다.
  - 큐가 빌 때까지 다음을 반복합니다.
    - 큐에서 간선 e를 뽑습니다. $e \in F_i$라고 합시다. 만약, $e = e_0$이라면 $i = 0$입니다.
    - 만일, $e \notin F_{(i+1 \mod 2)}(e)$라면, e로 끝나는 Augmenting 간선열이 존재합니다. 이는 e에서 시작하여 라벨링 간선을 따라가면 얻을 수 있습니다.
    - 그렇지 않다면, $F_{(i+1 \mod 2)}(e)$에서 아직 라벨링이 되지 않은 간선을 차례대로 보면서 e로 라벨링하고 큐에 넣습니다.

- 만일, Augmenting 간선열을 찾기 전에 앞의 알고리즘이 종료되었다면, Augmenting 간선열은 없으며 $F_0 \cup F_1$ 에 e_0 을 추가할 수 없습니다.
- 앞의 알고리즘에서 찾은 간선열이 Augmenting 간선열의 모든 조건을 만족함은 어렵지 않게 확인할 수 있습니다.
  - 특히, Augmenting 간선열의 마지막 조건을 만족한다는 점에 유의하세요.

- 앞의 알고리즘에서, $F_{(i+1 \mod 2)}(e)$ 에서 라벨링이 되지 않은 간선을 찾을 때, 이미 라벨링이 되어 있는 간선까지 모두 참조하면 안 됩니다.
  - 이 경우, 하나의 Augmenting 간선열을 찾는 데에 $\mathcal{O}\left(N^2\right)$ 의 시간이 소요되어, 전체 시간 복잡도가 $\mathcal{O}\left(N^3\right)$ 가 됩니다.
- $F_{(i+1 \mod 2)}(e)$ 에서 이미 라벨링이 되어 있는 간선은 모두 연속하게 등장한다는 성질을 이용하면, 한 Augmenting 간선열을 $\mathcal{O}\left(N\right)$ 에 찾을 수 있습니다.
- $e_0 = \{u, v\}$ 라면, F_0 과 F_1 의 루트를 u 로 잡으면 편합니다.

- 따라서, 전체 문제를 $\mathcal{O}\left(N^2\right)$에 해결할 수 있습니다.
- 그래프의 절선을 잘 관리하면서 매트로이드 교차 문제를 $\mathcal{O}\left(N^2 \lg N\right)$ 혹은 $\mathcal{O}\left(N^2\right)$에 해결하는 풀이도 있습니다.

1J. 문자열 X

`suffix_array`
출제진 의도 – **Hard**

- 제출 9번, 정답 5명 (정답률 55.56%)
- 처음 푼 사람: **조승현**, 125분
- 출제자: `dlalswp25`

1J. 문자열 X

- N 개의 문자열 단서 $S_1, S_2, \cdots, S_N$ 이 주어집니다.
- 이들 중 정확히 K 개의 단서가 X 를 부분문자열로 갖도록 하는 문자열 X 의 개수를 세는 문제입니다.
- 편의를 위해, "정확히 K 개" 라는 문구를 "K 개 이상" 으로 바꾸겠습니다.
- "K 개 이상" 일 때의 답에서 "$K+1$ 개 이상" 일 때의 답을 빼면 원래 문제의 답을 얻을 수 있습니다.

1J. 문자열 X

- 주어진 모든 단서를 합쳐 하나의 큰 문자열을 만듭시다.
  - 이때 인접한 두 단서 사이에는 의미 없는 문자 ('#' 등) 를 끼워 넣습니다.

b c # b a c # a b c # b c a

- 위 문자열에 대해 접미사 배열 (SA) 과 LCP 배열 (LCP) 을 각각 구해 놓습니다. 또한, 접미사 배열의 각 인덱스가 어느 단서에 포함되는지도 기록해둡니다. 이 배열을 C 라 하겠습니다.

1J. 문자열 X

b c # b a c # a b c # b c a

인덱스	SA	LCP	C	접미사
⋮	⋮	⋮	⋮	⋮
4	14	0	4	a
5	8	1	3	a b c # b c a
6	5	1	2	a c # a b c # b c a
7	4	0	2	b a c # a b c # b c a
8	1	1	1	b c # b a c # a b c # b c a
9	9	4	3	b c # b c a
10	12	2	4	b c a
11	6	0	2	c # a b c # b c a
12	2	2	1	c # b a c # a b c # b c a
13	10	3	3	c # b c a
14	13	1	4	c a

1J. 문자열 X

- SA_i에서 시작하는 문자열의 개수를 구해 봅시다.
- 중복이 발생하면 안 되므로 문자열의 길이는 LCP_i보다 길어야 합니다.
- 또한, 문자열이 #을 포함하면 안 되므로 길이는 특정 값을 넘지 못합니다.
- 이제 남은 것은 **문자열이 K개 이상의 단서에 포함되어야 한다**는 조건입니다.

1J. 문자열 X

- 문자열의 길이가 결정되었다고 하면, 이 문자열을 포함하는 시작 위치들은 접미사 배열 상에서 어떠한 구간 $[i, x]$ 로 나타날 것입니다.
- 이 문자열이 K 개 이상의 단서에 포함되려면 $C_i, C_{i+1}, \cdots, C_x$ 중 서로 다른 수가 K 개 이상이어야 합니다.
- 이를 만족하는 최소의 x 를 r_i 라 합시다. 이때 문자열의 길이는 $\min(LCP_{i+1}, \cdots, LCP_{r_i})$ 를 넘을 수 없습니다.
  - 문자열의 길이가 이보다 길어지면 구간이 짧아지고, 서로 다른 수의 개수가 K 보다 작아지기 때문입니다.
  - i 가 증가할 때마다 r_i 가 단조증가한다는 사실을 이용하면, 투 포인터 기법으로 r_i 를 효율적으로 찾을 수 있습니다.

1J. 문자열 X

- $K = 3, i = 8$인 경우의 예시입니다.
  - 이때 $r_i = 10$이고, 문자열의 최소 길이와 최대 길이는 각각 2입니다.

인덱스	SA	LCP	C	접미사
⋮	⋮	⋮	⋮	⋮
7	4	0	2	b a c # a b c # b c a
8	1	1	1	b c # b a c # a b c # b c a
9	9	4	3	b c # b c a
10	12	2	4	b c a
11	6	0	2	c # a b c # b c a
⋮	⋮	⋮	⋮	⋮

- 이와 같은 방법으로 각 위치에서 시작하는 문자열의 최소 길이와 최대 길이를 구할 수 있습니다.
- 최소 길이와 최대 길이 사이에 있는 문자열은 모두 조건을 만족합니다. 따라서 각 위치에서 시작하는 문자열의 개수 또한 구할 수 있습니다.
- 시간 복잡도는 구현에 따라 $\mathcal{O}(S)$ 또는 $\mathcal{O}(S \log S)$ 입니다. 이때 S 는 모든 단서의 길이의 합입니다.

1K. 누텔라 트리 (Hard)

`centroid_decomposition`
출제진 의도 – **Challenging**

- 제출 0번, 정답 0명 (정답률 0.00%)
- 처음 푼 사람: -, -분
- 출제자: `dlalswp25`

1K. 누텔라 트리 (Hard)

- 각 정점이 빨간색 또는 검은색으로 칠해진 트리가 주어질 때, 이 트리에서 **누텔라 경로**의 개수를 구하는 문제입니다.
  - 누텔라 경로란, 첫 번째 정점이 검은색이고 나머지 정점이 모두 빨간색인 길이 2 이상의 경로를 뜻합니다.
- 단, 정점의 색을 바꾸는 쿼리가 Q 개 주어집니다.

1K. 누텔라 트리 (Hard)

- 매 쿼리마다 빨간색 컴포넌트의 크기가 변하기 때문에 Easy 문제의 풀이는 활용하기 어려워 보입니다.
- 경로의 개수를 다른 방식으로 세어야 합니다.

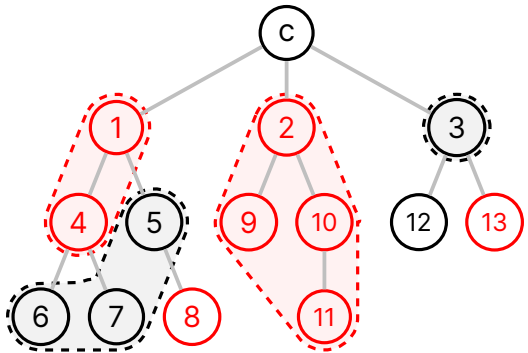
1K. 누텔라 트리 (Hard)

- 센트로이드 분할을 한 후, 각 센트로이드를 지나는 경로의 개수를 세어 봅시다.
- 센트로이드와 인접한 정점 v에 대해,
  - v가 빨간색이라면
    - $R_v =$ 센트로이드와 검은색 정점을 모두 무시할 때, v를 포함하는 컴포넌트의 크기
    - $B_v =$ 위 컴포넌트에 인접한 검은색 정점의 개수 (단, 센트로이드는 제외)
  - v가 검은색이라면
    - $R_v = 0$
    - $B_v = 1$

  이라고 정의합시다.

1K. 누텔라 트리 (Hard)

- 각 서브트리에서 빨간색으로 표시된 정점의 개수가 R_v, 검은색으로 표시된 정점의 개수가 B_v 입니다.



1K. 누텔라 트리 (Hard)

- 센트로이드가 검은색인 경우, 센트로이드를 지나는 경로의 개수는

$$\sum_v R_v$$

  입니다.

- 센트로이드가 빨간색인 경우, 센트로이드를 지나는 경로의 개수는

$$\left(1+\sum_v R_v\right)\left(\sum_v B_v\right)-\sum_v R_v B_v$$

  입니다.

- 따라서 매 쿼리마다 R_v 와 B_v 를 잘 관리하면 문제를 해결할 수 있습니다.

1K. 누텔라 트리 (Hard)

- R_v 를 관리하는 방법은 다음과 같습니다.
  - 모든 정점이 빨간색이라고 가정합시다.
  - 검은색 정점이 생겨날 때마다, 이 정점을 루트로 하는 서브트리 전체에 검은색을 한 겹 칠한다고 생각합시다 (검은색 정점이 사라질 때에는 한 겹을 지운다고 생각합시다).
  - R_v 는 검은색으로 칠해지지 않은 정점의 수입니다.
  - 정점을 DFS 방문 순서대로 나열하면 서브트리 쿼리가 구간 쿼리로 바뀌므로, 세그먼트 트리를 이용하여 이를 처리할 수 있습니다.

- B_v를 관리하는 방법은 다음과 같습니다.
  - R_v에 포함되는 각 정점의 자식 수를 모두 합하면 $R_v + B_v - 1$이 됩니다. 이 값을 관리할 수 있다면 B_v도 관리할 수 있습니다.
  - R_v를 관리할 때와 동일한 세그먼트 트리를 사용하면 됩니다.

- 따라서 하나의 센트로이드를 지나는 경로의 수를 $\mathcal{O}(\log N)$에 업데이트할 수 있습니다.
- 각 쿼리에 대해 $\mathcal{O}(\log N)$개의 센트로이드에서 업데이트가 이루어지므로, 총 시간 복잡도는 쿼리당 $\mathcal{O}(\log^2 N)$입니다.
- 이외에도 Heavy-Light Decomposition을 이용하는 풀이가 있습니다.