SNUPC 2024 풀이

Official Solutions

by

SNUPC 2024 출제진

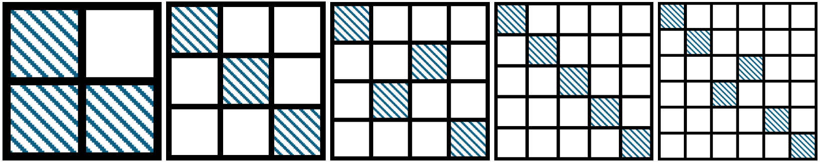
2A. 빙고 막기

`ad_hoc`
출제진 의도 – **Easy** (Div. 2)

- 제출 147번, 정답 56명 (정답률 38.095%)
- 처음 푼 사람: **양창석**, 0분
- 출제자: `whqkrkt04`

2A. 빙고 막기

- 각 행을 모두 막아야 하기 때문에, 답은 N 이상입니다.
- $N = 2$ 인 경우만 답이 3 이고, 나머지 모든 경우는 N 입니다.
- 다음과 같이 2, 홀수, 2 보다 큰 짝수로 나누어 constructive 한 증명이 가능합니다.



1A2B. 이상한 나라의 끈끈이 주걱

`greedy`

출제진 의도 – **Easy** (Div. 2) / **Easy** (Div. 1)

- Div. 1 제출 47번, 정답 22명 (정답률 46.809%)
- Div. 1 처음 푼 사람: **이민제**, 4분
- Div. 2 제출 219번, 정답 41명 (정답률 18.721%)
- Div. 2 처음 푼 사람: **권성안**, 10분
- 출제자: `mujigae`

1A2B. 이상한 나라의 끈끈이 주걱

- 파리는 원하는 만큼 고도를 높일 수는 있지만 낮추는 것은 매우 제한적입니다.
- 그렇기에 아래에서 솟은 끈끈이 주걱을 피하는 것은 항상 가능하므로 위에서 솟은 끈끈이 주걱을 피하는 것과 정확히 출구에 도착하는 것이 가능한지를 알아내면 됩니다.
- 어떤 입력이 안전하게 출구를 통과하는 것이 가능한 경우라면 아래에서 솟은 끈끈이 주걱을 마주치기 전까지 계속 고도를 낮추며 이동하는 것이 항상 해답 중에 존재합니다.
- x가 증가하도록 정렬하는 시간 복잡도인 $O(N \log N)$에 문제를 해결할 수 있습니다.

2C. 더 게임 오브 데스

`functional_graph`, `math`
출제진 의도 – **Easy** (Div. 2)

- 제출 159번, 정답 55명 (정답률 34.591%)
- 처음 푼 사람: **양창석**, 10분
- 출제자: `whqkrkt04`

2C. 더 게임 오브 데스

- $T \le N$ 이면 그냥 시뮬레이션하면 되기 때문에 $T > N$ 인 경우만 생각해주면 됩니다.
- $N + 1$ 번째 사람을 방문할 때는 비둘기집의 원리에 의해 적어도 한 명은 두 번 방문하게 됩니다.
- N 제한이 작기 때문에 처음으로 중복되어 나타난 사람이 누구인지, 언제 방문했는지는 다양한 방법으로 구할 수 있습니다.
- 그 다음부터는 특정한 주기를 가지고 계속 반복되기 때문에 매우 큰 T 에 대해서도 T 번째로 방문될 사람이 누구인지 알 수 있습니다.
- 여담으로, 각 정점에서 나가는 간선이 정확히 하나인 방향 그래프를 함수 $f : S \to S$ 에 대해 $x \to f(x)$ 들을 나타낸 것과 같다고 하여 functional graph라고 합니다.

1B2D. 보드게임

`math, ad_hoc, game_theory`
출제진 의도 – **Medium** (Div. 2) / **Easy** (Div. 1)

- Div. 1 제출 28번, 정답 22명 (정답률 78.571%)
- Div. 1 처음 푼 사람: **이민제**, 10분
- Div. 2 제출 104번, 정답 31명 (정답률 29.808%)
- Div. 2 처음 푼 사람: **조성해**, 40분
- 출제자: `sharaelong`

- 자기 자신의 카드를 뽑으면 다음 턴에도 카드를 뽑을 사람이 변하지 않으므로, 문제의 중요한 요소는 상대 이름의 첫 글자가 적힌 카드일 것으로 추정할 수 있습니다.
- 같은 사람이 카드를 계속 뽑는 연속된 턴을 합쳐서 **기회**라고 정의합니다.
- Alice 앞에 B가 a장 있다고 합시다. Alice는 최대한 빠르게 게임을 종료하려고 할 것이므로, 몇 번의 기회 만에 자신 앞의 카드를 전부 없앨 수 있는지 알아봅시다.
- 우선 Alice의 기회가 최소 a번은 필요합니다. B를 제거하자마자 다시 Bob에게 턴이 넘어가기 때문입니다.
- 또한 $a+1$번의 기회만 있으면 무조건 자신 앞에 있는 모든 카드를 없앨 수 있습니다. 모든 B 카드를 없앤 후에는 남은 카드가 A뿐이므로 더 이상 턴을 넘기지 않고 카드를 제거할 수 있습니다.

- 즉, Alice가 최종적으로 제거하는 카드가 B가 될 수 있으면 a번, 불가능하다면 $a+1$번이 필요합니다. 이는 마지막 행(N행)에 B가 하나 이상 있는지와 동치입니다.
- 비슷하게 Bob이 가진 A 카드를 b장이라고 하면, 위의 논리가 똑같이 적용됩니다. 즉, b번 혹은 $b+1$번의 Bob의 기회만 있으면 Bob도 자신 앞의 모든 카드를 제거 가능합니다.
- 이제 게임의 승패를 결정해 봅시다.
- Alice가 T_A번째 기회에 게임을 끝내고 Bob이 T_B번째 기회에 게임을 끝낸다면, Alice가 선공이므로 $T_A \leq T_B$일 때 Alice의 승리이고, 아니라면 Bob의 승리입니다.
- 추가적인 분석도 가능합니다: 카드 $2NM$장 중 A와 B가 각각 NM장으로 개수가 같으므로, $b=a$임을 알 수 있습니다.
- 이 관찰을 하면, 구현이 약간 편해집니다.

2E. Super Shy (Easy)

dp

출제진 의도 – **Medium** (Div. 2)

- 제출 98번, 정답 32명 (정답률 32.653%)
- 처음 푼 사람: **신지환**, 31분
- 출제자: `hidercpp`

2E. Super Shy (Easy)

- 자리에 사람이 앉는 것이 아니라 자리 사이의 공간을 기준으로 생각해봅시다.
- 이 문제는 길이가 $N-1$ 인 공간을 길이가 2 이상인 공간들로 분할하는 문제로 바뀝니다.
- 첫 분할은 원하는 대로 할 수 있으며, 나머지 분할은 자동으로 진행됩니다.
- $f(x)$ 를 길이가 x 인 공간을 자동 분할한 후 남는, 길이가 2 이상인 공간의 수로 두면, $f(0)=f(1)=0,\ f(2)=1,\ f(x)=f(\left\lfloor \frac{x}{2} \right\rfloor)+f(\left\lceil \frac{x}{2} \right\rceil)(x>2)$ 로 식을 세울 수 있습니다.
- 첫 분할 위치가 $a(0 \le a \le n-1)$ 일 때, 앉는 사람 수는 $f(a)+f((n-1)-a)+1$ 입니다.
- $f(x)(0 \le x \le n-1)$ 계산과 모든 a 에 대한 앉는 사람 수 계산이 전부 $O(N)$ 에 가능합니다.

1C2F. 나무 물 주기

`simulation`

출제진 의도 – **Hard** (Div. 2) / **Medium** (Div. 1)

- Div. 1 제출 54번, 정답 21명 (정답률 38.889%)
- Div. 1 처음 푼 사람: **이민제**, 16분
- Div. 2 제출 73번, 정답 14명 (정답률 19.178%)
- Div. 2 처음 푼 사람: **조성해**, 59분
- 출제자: `ksi4495`

1C2F. 나무 물 주기

- 1번 쿼리에서 열매가 물을 모두 흡수하면 자식 노드로 흘려주는 물이 없으므로 $O(1)$로 쿼리를 처리할 수 있습니다.
- 열매가 물을 흡수하고 물이 남는 경우는 흘려준 물의 양이 열매의 크기보다 큰 경우입니다.
- 이 때 열매는 자신의 크기만큼 물을 흡수하므로 열매의 크기가 두 배가 됩니다.
- 1번 쿼리에서 물을 흘려주는 양은 10^9 이하이므로 각 열매가 물을 흡수하고 남는 경우는 최대 $\lceil \log_2 10^9 \rceil$ 번 발생합니다.
- 따라서 각 간선은 최대 $\lceil \log_2 10^9 \rceil$ 번 사용되므로 문제를 그대로 시뮬레이션하되, 흘려줄 물이 없어지는 순간 멈추면 시간 복잡도가 $O(N \log X + Q)$가 됩니다.

1F2G. 가중치 복사 버그

`graphs, bfs, ad_hoc`
출제진 의도 – **Hard** (Div. 2) / **Medium** (Div. 1)

- Div. 1 제출 40번, 정답 19명 (정답률 47.500%)
- Div. 1 처음 푼 사람: **이성호**, 72분
- Div. 2 제출 33번, 정답 1명 (정답률 3.030%)
- Div. 2 처음 푼 사람: **조성해**, 169분
- 출제자: `sciencepark`

1F2G. 가중치 복사 버그

- "가중치가 c인 간선을 지나면, 그 직후 모든 간선의 가중치가 c만큼 증가한다."는 조건에 대해 생각해 봅시다.
- 지금까지 이동한 경로의 길이가 d라면, 처음 가중치가 c인 간선의 가중치는 $d + c$가 됩니다.
- 즉, 어떤 정점에 가장 빠르게 도달하는 것이 항상 이득입니다.
- 또한, 이 상태에서 처음 가중치가 c인 간선을 이용하면, 이동한 거리는 $d + (d + c) = 2d + c$가 됩니다.
- 거리를 이진수로 생각한다면, 이는 간선의 처음 가중치를 **뒤에 붙이는** 것으로 생각할 수 있습니다!

1F2G. 가중치 복사 버그

- BFS의 아이디어를 이용하여, 거리가 가까운 정점부터 탐색해봅시다.
- 시작 지점으로부터 거리가 d 인 정점이 있다면, 그 다음 정점은 거리가 $2d$ 혹은 $2d+1$ 입니다.
- 우선, 거리가 0 인 정점들을 모두 찾습니다. 이는 처음 가중치가 0 인 간선들만 이용해서 도달할 수 있는 정점들입니다. 그 정점들을 BFS의 시작 정점들로 합니다.
- BFS 도중에는 거리가 같은 모든 정점들에 대하여, 처음 가중치가 0 인 간선들을 모두 탐색한 뒤 처음 가중치가 1 인 간선들을 모두 탐색합니다.
- 이렇게 하면, 자연스럽게 거리가 증가하는 순서로 모든 정점을 탐색하게 됩니다.
- 탐색하게 되는 거리는 많아야 N 개이므로, 2차원 리스트 등 적절한 자료구조를 사용한다거나 하면 $O(N+M)$ 에 전체 문제가 해결됩니다.

2H. 여우 셰프

`greedy`, `parametric_search`
출제진 의도 – **Hard** (Div. 2)

- 제출 2번, 정답 0명 (정답률 0.000%)
- 출제자: `cozyyg`

- i 번 쿠키와 $i+1$ 번째 쿠키가 같은 상태면 0, 다른 상태면 1 인 새로운 배열 D_i 를 생각합니다.
- D_i 의 모든 값을 0 으로 만드는 것이 목표입니다.
- 쿠키를 한 번 뒤집을 때마다, 배열에서 바뀌는 값의 수는 최대 2 개며, 바뀌는 위치는 K 이상 떨어져 있습니다.
- 한쪽만 바뀌는 경우는 편의상 D_0 또는 D_n 이 같이 바뀐다고 생각할 수 있습니다.
- 이때 문제의 시행은 $j-i \geq K$ 인 D_i, D_j 를 선택하여 바꾸는 것이 됩니다.
- $i-0<K, n-i<K$ 인 D_i 의 값은 절대 바뀌지 않으며, 이 값이 1 이었다면 불가능합니다.

- 이외의 경우 항상 가능합니다.
- 쿠키를 뒤집는 방법을 바꿀 순서쌍의 나열로 나타낼 수 있습니다. (ex. $\{(2,7),(5,8),(6,9)\}$)
- 최소 횟수가 될 수 있는 방법을 추려내기 위해, 횟수가 늘어나지 않는 변환을 반복할 것입니다.
- 먼저 $D_x = 0$ 인데 (x,j) 또는 (j,x) 가 방법에 속한 경우를 살펴봅시다.
- 이때 D_x 가 바뀌는 횟수는 짝수여야 하기 때문에 다른 (x,k) 또는 (l,x) 가 방법에 속합니다.
- 이제 (x,j) 또는 (j,x) 를 각각 $(0,j)$ 또는 (j,n) 으로, (x,k) 또는 (l,x) 를 각각 $(0,k)$ 또는 (l,n) 으로 바꾸면, D_i 들의 최종 상태가 동일하면서 횟수가 늘어나지 않았습니다.
- 따라서 이 변환을 반복하면, 순서쌍의 모든 인덱스 x 가 $D_x = 1, x = 0, x = n$ 중 하나를 만족하게 됩니다.

- 이제 $D_x = 1$ 인 x 들을 크기 순서대로 $x_1, x_2, \cdots, x_k$ 라 합시다.
- 최적의 방법에서 각 순서쌍은 x_i 를 1개 또는 2개 포함합니다.
- 같은 x_i 가 3번 이상 나오는 경우에는 이전과 비슷하게 하나를 제외하고 모두 0 또는 n 으로 바꿀 수 있습니다.
- $(0, n)$ 은 최종 결과에 영향을 주지 않는 순서쌍이므로 제거할 수 있습니다.
- 위 변환을 더 이상 할 수 없을 때까지 반복하면, x_i 들은 정확히 하나의 순서쌍에만 등장합니다.
- 이때 순서쌍 중 x_i 를 2개 포함한 것의 개수를 y 라 하면, 그 방법의 시행 횟수는 $k - y$ 입니다.
- 따라서 y 의 최댓값을 구하면 됩니다. 즉, 겹치지 않도록 (x_i, x_j) 의 순서쌍을 최대한 많이 만들고 나머지 x_i 들을 0 또는 n 과 대응시키면 됩니다.

- $a < b, c < d$ 이고 $(x_a, x_d), (x_b, x_c)$ 가 방법에 속해있다면, 이것을 $(x_a, x_c), (x_b, x_d)$ 로 바꾸어도 올바른 방법입니다. ($\because x_d - x_b > x_c - x_b \geq K, x_c - x_a > x_c - x_b \geq K$)
- 위 변환을 반복하면 $a_1 < a_2 < \cdots < a_y, b_1 < b_2 < \cdots < b_y$ 에 대해 $(x_{a_1}, x_{b_1}), (x_{a_2}, x_{b_2}), \cdots, (x_{a_y}, x_{b_y})$ 가 방법에 속해 있는 형태가 됩니다.
- 한편 $a_i \geq i$ 이므로 a_1 부터 a_y 까지 순서대로 각각 $1, 2, \cdots, y$ 로 바꿀 수 있습니다. 마찬가지로 b_y 부터 b_1 까지 순서대로 각각 $k, k-1, \cdots, k-y+1$ 로 바꿀 수 있습니다.
- 따라서, (x_i, x_j) 의 순서쌍을 y 개 만들 수 있다면, **첫 y 개와 마지막 y 개를 크기 순서대로 짝지은 y 개의 순서쌍도 올바릅니다!**

- 이를 이용하면 y의 값을 구하기 위해 **매개 변수 탐색**을 이용할 수 있습니다.
- 이때 y가 가능한지 여부는 첫 y개와 마지막 y개를 짝짓는 경우만 확인하는 방법으로 $O(N)$에 해결할 수 있고, 전체 시간복잡도 $O(N \log N)$에 문제를 해결할 수 있습니다.
- $y \leq k/2$임에 유의합니다.

- 위 결과에서, b_y 부터 b_1 까지 순서대로 각각 $k, k-1, \cdots, k-y+1$ 로 바꾸지 않고, 대신에 b_1 부터 b_y 까지 순서대로 $x_{b_i} - x_{a_i} \geq K, b_i > b_{i-1}$ 인 최소의 b_i 로 바꿀 수도 있습니다. 이때 b_0 는 k 를 2 로 나눈 몫으로 정의합니다.
- 이때 $y+1$ 개의 순서쌍이 만들어질 수 있는지를 확인하는 과정에서 y 개의 순서쌍을 만든 결과를 이용할 수 있게 됩니다.
- 종합하면, x_i 들을 왼쪽 절반과 오른쪽 절반으로 나눈 다음, **투 포인터**를 이용하여 왼쪽 절반에 있는 원소들을 위 방법대로 오른쪽 절반에 순서대로 대응시켜서 순서쌍이 몇 개 만들어지는지를 확인하면 됩니다.
- 이때 전체 시간복잡도 $O(N)$ 에 문제를 해결할 수 있습니다.

1D. Natural Number Streamer

`divide_and_conquer, string`

출제진 의도 – **Hard** (Div. 1)

- 제출 8번, 정답 0명 (정답률 0.000%)
- 출제자: `sharaelong`

1D. Natural Number Streamer

- 문자열에 `01` 또는 `110`만 포함되어 있어도 답이 2 이상입니다.
- 그러므로 답이 1인 경우는 `100...00`, `11...11`, `00...00` 뿐입니다.
- 이제 답이 3 이상인지 판별할 방법을 생각해봅시다.
- 답이 3 이상이라면, '홀짝홀'이나 '짝홀짝'이 최적해에 포함되는데, 항상 '짝홀'이 포함되는 것을 알 수 있습니다.
- 따라서 이것에 이름을 붙여봅시다: Substring이 어떤 $x \geq 0$에 대해 $bin(2x)bin(2x+1)$로 표현될 때, 이를 *even-odd pair* (이하 eop)라 합니다.
- 만약 S에 존재하는 eop를 모두 찾을 수 있다면, eop들끼리 '이어 붙여서' maximal 점수를 갖는 substring들을 전부 파악할 수 있습니다.
- 물론 eop를 이어붙인 후에 앞뒤로 한 개의 수가 더 붙는 경우에 대한 처리는 해 주어야 합니다.

1D. Natural Number Streamer

- 이 시점에서 할 만한 가장 쉬운 생각은, eop의 총 개수가 $o(n^2)$ 인지 여부입니다. 만약 그렇다면 eop를 이어 붙인 maximal substring들을 전부 확인하여 답을 구해낼 수 있고, 시간 복잡도도 $o(n^2)$ 이 됩니다.
- 흥미롭게도, eop는 많아야 $O(n \log n)$ 개 존재합니다! 이를 증명해봅시다.
- $S\left[s, s+2k-1\right]$ 가 eop라고 합시다.
- 이 때 $S\left[i, i+2k-1\right]$ $(s < i < s+k)$ 가 모두 eop가 아닙니다. $S[s+k-1] \neq S[s+2k-1]$ 이기 때문입니다.
- 따라서 길이가 $2k$ 인 eop가 한 번 등장하면, 이후의 길이 $2k$ 짜리 substring $k-1$ 개가 모두 eop가 아닙니다.
- 그러므로 eop의 개수는 최대 $\displaystyle\sum_{k=1}^{n/2} \frac{n}{k} = O(n \log n)$ 입니다.

1D. Natural Number Streamer

- 뒤에서 살펴보겠지만, 이 관찰을 하지 못했다고 해도 eop를 모두 구하는 알고리즘을 고안할 수 있습니다. 이 알고리즘의 시간 복잡도가 $O(n \log n)$ 라면 eop가 $O(n \log n)$ 개라는 결론에 도달할 수 있습니다.
- 그러므로, 모든 eop를 $O(n \log n)$ 에 찾는 방법을 생각해봅시다.
- 여러 가지 접근이 가능하며, 여기서는 대회 준비 도중 발견된 3 개의 풀이를 모두 소개합니다!

1D. Natural Number Streamer

- 첫 번째 풀이는 위의 관찰을 조금 더 심층적으로 이용하는 풀이입니다. 이 풀이는 eunlin님께서 고안하셨습니다!
- 총 길이가 $2k$인 eop를 $O(n/k)$ 시간에 찾아내 봅시다.
- $S[i, i+2k-1]$가 eop라고 합시다.
- $[i, i+k)$ 중 k의 배수가 반드시 존재하는데, 이를 mk라고 합시다.
- $S[mk, (m+1)k-1]$와 $S[(m+1)k, (m+2)k-1]$를 생각했을 때, 이들의 LCP(longest common prefix)의 길이가 l이라고 하겠습니다.
- eop 조건에 따르면 $S[i, i+k-2] = S[i+k, i+2k-2]$이므로, $S[mk, i+k-2] = S[(m+1)k, i+2k-2]$도 성립합니다.
- 또한 $S[i+k-1] \neq S[i+2k-1]$이므로, eop의 왼쪽 끝 문자 위치가 $i+k-1 = mk+l$를 만족합니다!

1D. Natural Number Streamer

- S 의 i 번 문자로부터의 suffix를 S_i 라고 표기하겠습니다.
- SA, LCP, sparse table을 이용한 RMQ로 S_{mk} 와 $S_{(m+1)k}$ 의 LCP를 $O(1)$ 에 구하는 게 가능합니다.
- 따라서 모든 m 마다 LCP l 이 유일하게 정의되고, eop의 시작으로 가능한 후보 i 가 유일하게 결정됩니다!
- 따라서 $S\left[i, i+2k-1\right]$ 가 실제로 eop인지 판별해준다면 문제를 빠르게 해결 가능합니다.
- 이는 S_i 와 S_{i+k} 의 LCP를 똑같이 $O(1)$ 에 구해줌으로써 판별 가능합니다. leading zero로 시작하거나, $bin(2x+1)bin(2x)$ 를 구하지 않도록 구현에 유의가 필요합니다.

1D. Natural Number Streamer

- 두 번째 풀이는 자료구조를 활용합니다. 이 풀이는 mhy908 님께서 고안해주셨습니다!
- $S[i, i+2k-1]$ 가 eop라고 합시다.
- 앞선 풀이에서 언급했듯이, S_i 와 S_{i+k} 의 LCP(longest common prefix)의 길이가 $k-1$ 인 것입니다. (거의 충분조건에 가까운 강력한 명제이기도 한 것을 확인할 수 있습니다)
- S 의 모든 suffix를 trie에 넣은 상황을 생각합시다. (suffix tree) 알파벳이 `0`, 1 뿐이므로, 이 trie는 binary tree이기도 합니다.
- 여기서 S_i 와 S_{i+k} 는 깊이가 $k-1$ 인 어떤 node에서 처음으로 경로가 갈라지게 됩니다.
- 반대로 말하자면, 모든 eop는 trie에서 두 suffix가 갈라지는 node에 무조건 대응시킬 수 있습니다.

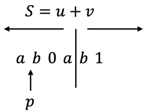
- 그러므로 eop의 개수 총합은, 각 node별로 그 node에 대응되는 eop 개수 합입니다.
- 깊이 $k-1$의 node T 하나를 고정합시다. T에서 `0`쪽으로 더 들어가는 suffix의 개수를 C_0, 1쪽은 C_1 개라고 합시다.
- 이 때, T에 대응되는 eop의 개수는 많아야 $\min(C_0, C_1)$ 개입니다.
- T의 `0`쪽으로 들어가는 어떤 suffix를 하나 고정했을 때, 이러면 이미 eop의 왼쪽 절반 $S[i, i+k-1]$이 결정된 것이므로, S_{i+k}가 T를 거쳐서 1쪽으로 들어가면 eop를 하나 찾은 것이고, 아니라면 그냥 eop가 없는 것이기 때문입니다.
- 이는 `0`과 1의 역할을 바꿔서 봐도 똑같습니다. 그러므로 위의 관찰이 성립합니다.
- 따라서 trie의 모든 node에서 $\min(C_0, C_1)$의 합은 $O(n \log n)$임을 보일 수 있습니다. Small-to-Large, HLD의 원리와 정확히 같은 argument를 사용하면 됩니다.

- 대략의 증명은 다음과 같습니다: 정점 개수가 n 개인 binary tree에서 $\min(C_0, C_1)$ 의 합의 최댓값을 $T(n)$ 이라고 합시다.
- 즉 여기서는 C_0, C_1 이 각 정점의 두 가지 subtree 크기를 의미합니다.
- 트리의 루트에서 0 쪽의 subtree 크기가 k 이면 1 쪽의 subtree 크기가 $n-1-k$ 입니다.
- 그러므로 점화식은 $T(n) = \max_{k \le n/2} T(k) + T(n-1-k) + k$ 입니다.
- 모든 $m < n$ 에 대해 $T(m) \le m \log m < n \log m$ 이었다면,
  $T(k)+T(n-1-k)+k \le k\log(2k)+(n-1-k)\log(n-1-k) \le (n-1)\log n < n\log n$
  이므로, 수학적 귀납법에 의해 성립합니다.

- 그러므로 S 의 suffix array와 LCP array를 만들고, suffix tree를 전부 순회하듯이 이 과정을 진행해주면 됩니다.
- suffix tree를 명시적으로 만들지 않아도 되며, 이 경우 divide and conquer을 구현하듯이 재귀적으로 시행이 가능합니다.

1D. Natural Number Streamer

- 마지막으로, 출제자의 정해는 Divide and Conquer 방식입니다.
- 문자열이 $S = u + v$ 로 반반 나뉘어 있을 때, u 와 v 에 걸친 eop들을 $O(n)$ 에 모두 찾을 수 있다면 전체 문제가 $O(n \log n)$ 에 해결될 것입니다.
- 먼저 eop가 왼쪽으로 쏠린 경우, 즉 $bin(2x)$ 파트가 u 에 완전히 포함될 때를 생각합시다.
- eop의 길이를 고정하면, eop의 시작점으로 가능한 후보가 최대 1 개입니다.
- eop가 u/v 가 나뉜 절반 지점에 의해, 그림과 같이 `ab0ab1` 꼴이 된다고 합시다.
- $bin(2x)$ 의 b 파트가 시작하는 지점의 index를 p 라고 합시다.



1D. Natural Number Streamer

- 핵심적인 관찰은 다음과 같습니다: eop의 길이, 즉 $k = |a| + |b|$을 고정하면, eop의 시작점 (a의 첫 글자의 index $= p - |a|$)으로 가능한 후보가 최대 1개입니다.
- 왜냐하면 $p + k + 1 = |u|$이므로 k가 고정되면 p도 고정되며, 이 때 $|b|$가 $u\left[p, p + i - 1\right] = v\left[0, i - 1\right]$인 i의 최댓값일 수밖에 없기 때문입니다. (이러한 i는 unique)
- $u\left[p + |b|\right] = \mathtt{0}, v\left[|b|\right] = \mathtt{1}$이여야 하기 때문입니다.
- 따라서 어떤 k가 eop를 만드는지 확인하려면, $k \to p \to |b| \to |a|$ 순으로 값을 계산하고, 다음 두 조건이 성립하는 것이 eop일 필요조건입니다.
- $u\left[p - |a|, p - 1\right] = u\left[|u| - |a|, |u| - 1\right], u\left[p, p + |b| - 1\right] = v\left[0, |b| - 1\right]$
- 또한 ab가 non-empty라면 ab의 첫 글자가 항상 `1`이어야 하며, `ab1ab0` 꼴은 구하면 안 됩니다. 이 4가지 조건은 eop일 필요충분조건이 됨을 확인할 수 있습니다!

1D. Natural Number Streamer

- 남은 작업은, 각 k 마다 위의 4 가지 조건 판별을 $O(1)$ 시간에 처리하는 것입니다.
- $|b|$ 값을 알고 있다면 모든 필요충분조건들은 substring 간의 equality를 판정하는 작업이므로, hashing으로 probabilistic하게 판정하거나 suffix array 등을 사용해 deterministic하게 $O(1)$에 해결 가능합니다.
- $|b|$ 값을 알아내는 것도 $O(1)$에 가능합니다. $v\#u$ 라는 문자열의 Z 배열을 구하고 $|v|+p+1$ 번째 위치의 값이 $|b|$ 가 되기 때문입니다.
- 따라서 이러한 Z 배열을 구하는데 $O(|S|)$ 시간이 걸리고, 가능한 k 의 범위는 $0 \le k \le |S|/2$ 이므로 dnc의 한 과정을 $O(|S|)$ 에 해결 가능합니다!
- eop가 오른쪽에 몰린 경우도 동일한 방식으로 해결할 수 있으나, eop의 구조 상 좌우 대칭이 아니므로 사소하게 달라지는 지점들이 있습니다. 예를 들어, 이 경우에는 v 에 대한 Z 배열만 구해도 됩니다.

1D. Natural Number Streamer

- 이제 어떻게든 모든 eop를 구했다면, eop를 이어 붙일 시간입니다.
- 다음의 dp를 정의합니다: $dp[i]$: $S[i]$를 마지막 글자로 가지는 substring의 점수 최댓값
- 이렇게 정의하면, 수가 이어지는 eop가 $S\left[i-2k+1, i\right]$에 위치할 때, $dp[i] \geq dp[i-2k+1]+2$가 성립합니다.
- 따라서 이를 전이식으로 두면 됩니다. 최종 답은 dp 배열 전체의 최댓값입니다.
- dp의 초기화는, 어떤 $S\left[i-2k+1, i\right]$에 위치하는 eop $bin(2x)bin(2x+1)$에 대해, $S[j, i-2k] = bin(2x-1)$인 j와 x가 존재하면 3으로, eop만 존재하면 2로, 아니라면 0으로 두면 됩니다. (답이 1인 경우는 맨 앞에서 예외로 처리했으므로)
- 특정 위치를 오른쪽 끝으로 갖고 특정 길이를 갖는 eop가 존재하는지 판별하는 작업은 set 등의 자료구조로 $O(\log n)$에 처리해줘도 되는 시간 제한을 설정했습니다.

1D. Natural Number Streamer

- 또한 이 과정 중에 어떤 인접한 두 substring $s_1 = bin(x), s_2 = bin(y)$ 에 대해 $y = x + 1$ 인지 판별하는 작업이 필요할 것입니다.
- Base를 2로 두는 rabin-karp hashing을 쓰면 해시함수 H에 대해 $H(y) = H(x) + 1$ 인지 판별하는 것과 동치입니다.
- Deterministic하게는, 직접 $x + 1$을 계산해보면 됩니다. x의 1의 자리 수부터 연속한 1의 길이를 알고 있으면, S의 substring equality를 판정하는 문제로 환원됩니다.

1D. Natural Number Streamer

- Open problem. 길이 n의 binary string이 가질 수 있는 eop의 개수가 $O(n)$개 이하일까요?
- 실제로 n이 30 정도로 작을 때 brute force를 돌려보면, n개를 넘는 경우가 전혀 존재하지 않습니다.
- 추가적으로, 작은 n에서 최대의 eop를 갖는 문자열은 0이 연속되어서 3개 이상 존재하지 않았고, 문자열이 1로 끝나는 경우를 제외하고 1이 연속되어서 3개 이상 존재하지 않았습니다.
- 다만, $n + 2 - \sqrt{2n}$개의 eop를 갖는 문자열은 항상 존재합니다. 이는 실제로 이 문제의 데이터로도 들어가 있습니다.
- 이 문제의 정해 시간복잡도가 $O(n \log n + |\text{eop}| \log |\text{eop}|)$이므로 만약 eop가 $O(n)$개임을 증명할 수 있다면 정해 시간복잡도의 상한 또한 감소합니다.
- 다만 SNUPC의 출제진은 이를 증명하지 못했기 때문에, 시간 제한은 $O(n \log^2 n + |\text{eop}| \log |\text{eop}|)$이더라도 통과하도록 설정되었습니다.

1E. Super Shy (Hard)

`math`, `ad_hoc`, `slope_trick`
출제진 의도 – **Medium** (Div. 1)

- 제출 56번, 정답 20명 (정답률 35.714%)
- 처음 푼 사람: **이민제**, 43분
- 출제자: `hidercpp`

1E. Super Shy (Hard)

- 자리에 사람이 앉는 것이 아니라 자리 사이의 공간을 기준으로 생각해봅시다.
- 이 문제는 길이가 $N-1$ 인 공간을 길이가 2 이상인 공간들로 분할하는 문제로 바뀝니다.
- 첫 분할은 원하는 대로 할 수 있으며, 나머지 분할은 자동으로 진행됩니다.
- $f(x)$ 를 길이가 x 인 공간을 자동 분할한 후 남는, 길이가 2 이상인 공간의 수로 두면, $f(0)=f(1)=0,\ f(2)=1,\ f(x)=f(\left\lfloor\frac{x}{2}\right\rfloor)+f(\left\lceil\frac{x}{2}\right\rceil)(x>2)$ 로 식을 세울 수 있습니다.
- 첫 분할 위치가 $a(0\le a\le n-1)$ 일 때, 앉는 사람 수는 $f(a)+f((n-1)-a)+1$ 입니다. 따라서 문제의 답은 $\max\limits_{0\le a\le n-1} f(a)+f((n-1)-a)+1$ 입니다.

1E. Super Shy (Hard)

- $x > 1$ 일 때 $x = x_k x_{k-1} ... x_0$ $(x_k = 1)$ 와 같이 이진수로 나타내면, x_{k-1} 의 값에 따라
$f(x) = \begin{cases} 2^{k-1} & (x_{k-1} = 0) \\ x - 2^k & (x_{k-1} = 1) \end{cases}$ 이라는 사실을 수학적 귀납법으로 보일 수 있습니다.
- $k = 1$ 이면 $f(2) = f(3) = 1$ 이므로 위 식이 성립합니다.
- $k = i$ 일때 위 식이 성립한다면, $k = i + 1$ 일 때 x_{k-1} 의 값에 따라
$f(x) = f(\left\lfloor \frac{x}{2} \right\rfloor) + f(\left\lceil \frac{x}{2} \right\rceil) = \begin{cases} 2^{k-2} + 2^{k-2} = 2^{k-1} & (x_{k-1} = 0) \\ (\left\lfloor \frac{x}{2} \right\rfloor - 2^{k-1}) + (\left\lceil \frac{x}{2} \right\rceil - 2^{k-1}) = x - 2^k & (x_{k-1} = 1) \end{cases}$
이므로 식이 성립합니다.

1E. Super Shy (Hard)

- $m := n - 1 = m_i m_{i-1} ... m_0 \ (m_i = 1)$ 와 같이 이진수로 나타내고, $m > 1$ 일때 $f(a) + f(m - a)$ 가 최대가 되는 $a(0 \le a \le m)$ 를 찾아봅시다.
- $a = x$ 와 $a = m - x$ 일때 식이 같으므로, $\left\lceil \frac{m}{2} \right\rceil \le a \le m$ 의 범위에서 탐색해도 무방합니다.
- f 에서 $f(x+1) = f(x)$ 또는 $f(x+1) = f(x) + 1$ 이 성립한다는 성질을 관찰할 수 있습니다.
- $f(x+1) = f(x)$ 라면 $f(m - (x+1)) \le f(m - x)$ 이므로 $a = x + 1$ 일 때 $a = x$ 일때보다 $f(a) + f(m - a)$ 의 값이 크거나 같음이 보장됩니다.
- $f(x+1) = f(x) + 1$ 이라면 $f(m - x) \le f(m - (x+1)) + 1$ 이므로 $a = x$ 일 때 $a = x + 1$ 일때보다 $f(a) + f(m - a)$ 의 값이 크거나 같음이 보장됩니다.
- 위의 사실들로부터 $a = \left\lceil \frac{m}{2} \right\rceil, a = 2^i, a = m$ 의 세 경우만 생각하면 됨을 알 수 있습니다.

1E. Super Shy (Hard)

- $a = \left\lceil \frac{m}{2} \right\rceil$ 또는 $a = m$ 인 경우 $f(a) + f(m-a) = f(m)$ 입니다.
- $a = 2^i$ 인 경우 $f(a) + f(m-a) = 2^{i-1} + f(m-2^i) \geq f(m)$ 임을 식 대입을 통해 확인할 수 있습니다.
- 즉, $f(a) + f(m-a)$ 가 최대가 되는 $a(0 \leq a \leq m)$ 중 하나는 $a = 2^i$ 입니다.
- 이 때 최대로 앉을 수 있는 사람 수는 $f(a) + f(m-a) + 1 = 2^{i-1} + f(m-2^i) + 1$ 이 됩니다.
- $m = 0$ 또는 $m = 1$ 일때 답은 자명하게도 1 입니다.
- 이 풀이의 시간복잡도는 $O(Q)$ 입니다.

1G. 지그재그 히스토그램 자르기

dp

출제진 의도 – **Hard** (Div. 1)

- 제출 30번, 정답 1명 (정답률 3.333%)
- 처음 푼 사람: **이동현**, 184분
- 출제자: `eunlin`

- 주어진 히스토그램을 조각에서 가장 큰 직사각형의 넓이가 지그재그가 되도록 최대한 많은 조각으로 나누어야 합니다.
- 조각을 나누었을 때 좌우 조각보다 넓이 값이 작은 조각을 valley, 큰 조각을 mountain이라고 하면 각각의 valley가 하나의 직사각형으로 구성된 최적해가 존재합니다.
- valley가 2개 이상으로 구성된 최적해에서 그 valley의 직사각형 중 하나를 인접해 있는 mountain 조각에 포함되도록 고쳐도 전체 조각의 수는 동일하고 여전히 지그재그 조건을 만족합니다. 따라서 이 과정을 반복하면 모든 valley가 하나의 직사각형으로 구성된 최적해를 얻을 수 있습니다.

- A_i 를 시작이 valley 또는 mountain 임과 상관없이, i 번째 직사각형이 마지막 valley 라고 할 때 최대 valley 의 개수라고 정의합시다. 그런 경우가 불가능하다면, $A_i = 0$ 으로 정의합니다.
- B_i 를 $\max\left(A_i, \max_{j<i, H_j<H_i} A_j\right)$ 으로 정의합니다. 그러면 $B_i - B_{i-1}$ 은 0 또는 1 입니다.
- 귀납법을 활용한 증명은 다음과 같습니다. 먼저, $A_1 = B_1 = 1$ 입니다.
- $H_1 < H_2$ 라면 $B_2 = B_1$ 으로 $B_2 = 1$ 이 됩니다.
- $H_2 \le H_1$ 이라면 1 이 mountain, 2 가 valley 가 되어 $1 = A_2 = B_2$ 입니다.
- 즉, $B_1 = B_2 = 1$ 입니다. $B_2 = 1$ 임을 보인 과정은 다음 페이지에서 다시 쓰입니다.

1G. 지그재그 히스토그램 자르기

- B_i 를 구하고자 하는 i 에 대해, j 를 $B_j = \max_{k<i} B_k = \max_{k<i} A_k$ 인 j 들 중 최솟값이라고 합시다.
- $B_i \le \max_{k<i} A_k + 1 = B_j + 1 = B_{i-1} + 1$ 임은 귀납가정과 A_i 의 정의에 의해 자명합니다.
- $B_{i-1} = B_j \le B_i$ 는 $B_2 = 1$ 인 것과 거의 같은 논리로 성립합니다.
- $H_j < H_i$ 라면 B 의 정의에 의해 $B_j \le B_i$ 입니다.
- j 의 최소 성질에 의해, $A_j = B_j$ 입니다. 즉, j 를 마지막 valley로 하는 valley가 B_j 개인 1 부터 j 까지의 분할이 존재합니다.
- $H_i \le H_j$ 라면 그 분할의 마지막 valley를 j 에서 i 로 옮긴 분할도 올바름을 보일 수 있습니다. 따라서 $B_j = A_j \le A_i \le B_i$ 입니다.
- 즉, $0 \le B_i - B_{i-1} \le 1$ 입니다. 이제 B_i 를 어떻게 계산할지 살펴봅시다.

1G. 지그재그 히스토그램 자르기

- 만약 $B_i = B_{i-1} + 1$ 이라면 B 의 최댓값을 갱신하였기 때문에 $A_i = B_i$ 입니다. 즉, valley i 가 추가된 해가 있는 것과 이는 동치입니다.
- j 를 앞의 슬라이드와 똑같이 (i 이전 B_k 의 최댓값을 주는 최소 k) 정의합시다.
- 만약 새로 찾은 해에서 j 이상 i 미만의 valley가 없다면, j 의 최소성에 모순입니다.
- 반대로 j 이상 i 미만의 valley가 2 개 이상이라면 A_j 를 주는 해의 첫 $A_j - 1$ 개의 valley들 이후, 새로운 해에서 j 이상 i 미만의 가장 큰 valley, valley i 의 해는 올바름을 보일 수 있습니다.
- 따라서, valley i 를 추가하고 j 에 있던 valley를 $[j, i)$ 구간의 원소 각각으로 옮겨 본 후 올바른 분할인지를 판별합니다. 물론 $i - 1$ 로 옮긴 분할은 두 valley가 연속하게 되므로 올바르지 않습니다.
- 만약 하나 이상이 올바르면 $B_i = B_{i-1} + 1$ 이고, 아니면 $B_i = B_{i-1}$ 입니다.

- 놀랍게도, $i - j$ 는 i 와 j 에 독립적으로 $O(\log H)$ 입니다!
- 만약 $j \le p < q < r \le i$ 이면서 $H_p \le H_q \ge H_r$ 인 p, q, r 이 존재한다면, j 부터 p 까지 H_k 가 가장 작은 k 중 하나를 생각합시다.
- A_j 최적해의 마지막 mountain을 $k - 1$ 까지 늘린 후 k 가 valley, $[k + 1, r - 1]$ 이 mountain, r 이 valley인 해를 생각하면 $H_k \le H_j, H_k \le H_p \le H_q, H_r \le H_q$ 임에서 항상 지그재그 조건을 만족합니다.
- $r < i$ 라면 위 해의 존재성에 의해 $B_r \ge A_r > A_j = B_j$ 이고 $r < i$ 입니다. 이는 B_j 가 최대라는 것에 모순입니다. 따라서 $r = i$ 입니다.

- j 부터 i 까지 중 H_x 가 가장 작은 x 들 중 가장 작은 것을 생각합시다.
- 앞 슬라이드의 내용에 의해, H 는 j 부터 x 까지 순감소한 후 x 부터 $i-1$ 까지 단조증가합니다.
- $j < k < x - 2$ 인 k 에 대해 $H_k \le 2H_{k+2}$ 라면 j 대신 k 와 x 를 valley로 잡았을 때 해를 구성하게 되어 B_j 가 최대임에 모순입니다. 따라서 $H_k > 2H_{k+2}$ 이고, $x - j$ 는 $O(\log H)$ 입니다.
- $x < k < i - 2$ 인 k 에 대해 $2H_k \ge H_{k+2}$ 라면 x 와 $k + 2$ 를 valley로 잡았을 때 해를 구성하게 되어 B_j 가 최대임에 모순입니다. 따라서 위와 같은 논리에 의해 $i - x$ 는 $O(\log H)$ 이고, $i - j$ 또한 $O(\log H)$ 입니다.
- 더 자세하게는, $i - j$ 는 $4(\log H + 1)$ 정도보다 항상 작습니다.
- 여전히 이 알고리즘을 그대로 구현하면 $O(N \log^2 H)$ 로 시간초과를 받습니다.

- $S[i:j]$를 히스토그램의 i번째부터 j번째 조각 안에서 가장 큰 직사각형의 넓이라고 합시다.
- $S[x+1:i-1] \geq H_i$라면 두 valley가 x, i인 해를 구성한 것입니다. 만약 그렇지 않으면 다른 모든 $x < l < i$에 대해서 $S[x+1:i-1] \geq S[l+1:i-1]$이므로 증가하는 구간에서는 해를 구성하지 못하고 나머지 구간을 봐야 합니다.
- H가 감소하는 구간에 대해서, 각 위치 $k(j \leq k < x)$를 valley로 두었을 때를 생각해봅시다. $k \neq j$인 k에 대해 j를 valley로 하는 solution에서 $H_k \leq H_j$이므로 j부터 $k-1$번째 직사각형을 이전 mountain에 포함시키고, valley를 k로 이동시킬 수 있습니다.
- $k+1$과 $i-1$을 양 끝으로 하는 직사각형의 넓이 S'에 대해 $S' \geq \max(H_k, H_i)$인지 검사하고, 만족한다면 k와 i를 valley로 둘 수 있는 것이므로 해를 구성한 것입니다. 만약 $S[k+1:i-1]$의 넓이를 주는 직사각형의 왼쪽 끝이 $k+1$이 아니라면 최대 넓이를 가지지 않고, 오른쪽 끝이 $i-1$이 아니라면 이미 i 이전에 해를 구성했어야 했거나, 최대 넓이를 가지지 않기 때문에 고려하지 않아도 됩니다. 이로써 로그가 하나 떨어집니다.

- 이제 답을 구할 시간입니다. 위의 알고리즘을 입력 그대로(1), 입력의 맨 앞에 0을 넣어서(2), 맨 끝에 0을 넣어서(3), 맨 앞과 맨 끝 모두에 0을 하나씩 넣어서(4) 시행합니다.
- 각각에서 나온 B의 마지막 원소 B_{-1}을 $M_i (1 \le i \le 4)$라고 하겠습니다. 답은 $\max (2M_1 - 1, 2M_2 - 2, 2M_3 - 2, 2M_4 - 3)$ 입니다.
- 이는 시작과 끝이 mountain인 경우를 고려해주기 위해 필요합니다. 만약 끝에 0을 넣었을 경우 $B_{-1} = A_{-1}$ 입니다. 이는 마지막 valley를 항상 마지막 0으로 옮길 수 있기 때문입니다.
- 만약 끝에 0을 넣지 않았을 때 $B_{-1} \ne A_{-1}$ 인 것은 문제가 되지 않습니다. $B_{-1} = A_k, H_k < H_{-1}$ 인 k가 존재하게 되는데, 이 경우 끝에 0을 넣었을 때 B_{-1} 이 1 큰 값이 나오게 되어 알고리즘의 잘못된 답이 무시됩니다.
- 앞에 0을 넣었을 때도 같은 논리가 적용됩니다. 시간 복잡도는 $O(N \log H)$ 입니다.

Open. 트리와 경로 뒤집기 쿼리

`heavy_light_decomposition`

출제진 의도 – **Hard** (Div. 1 Open)

- 출제자: `eunlin`

Open. 트리와 경로 뒤집기 쿼리

- 간선의 방향성을 뒤집는 쿼리가 주어졌을 때 도달 가능한 정점 쌍의 수를 세야 합니다.
- 우선 dp를 이용해 정점 쌍의 개수를 세는 법을 살펴봅시다. $dp[i]$를 정점 i를 root로 하는 subtree에서 해당 문제의 답이라고 하고, $A[i]$와 $B[i]$를 각각 정점 i를 root로 하는 subtree 안의 정점 u 중 i로 도달하는 경로가 존재하는 u의 개수, i에서 u로 도달하는 경로가 존재하는 u의 개수라고 정의합시다.
- 그러면 정점 i를 LCA로 가지는 도달 가능한 정점 쌍의 수는 $A[i] \times B[i]$이므로 $dp[i] = A[i] \times B[i] + \sum_{j \text{ is child of } i} dp[j]$이고, 이는 트리 dp로 쉽게 계산할 수 있습니다.
- 하지만 이 문제는 쿼리 문제입니다. Heavy-light decomposition을 이용해 트리를 관리하고, 각 chain에서 lazy segment tree를 이용해서 $A[i], B[i], dp[i]$ 값의 전이를 관리합시다.

- x 와 y 사이 경로의 간선의 방향성을 뒤집는 것은 x 와 root 사이 간선들의 방향성을 뒤집고, y 와 root 사이 간선들의 방향성을 뒤집는 것으로 생각할 수 있습니다. 즉, 쿼리의 형태를 항상 한 쪽에 root가 포함되게 바꿀 수 있습니다.
- 트리에서 자식의 $A[i]$ 또는 $B[i]$ 값의 변화가 부모의 $A[i], B[i], dp[i]$ 값에 영향을 미치기 때문에 세그먼트 트리에 세 가지 값을 단순하게 들고 있는 것만로는 값의 변화를 처리하기 어렵습니다.
- 세그먼트 트리의 노드마다 트리의 heavy chain에서 대응되는 연속한 정점들을 생각해줄 수 있습니다. 이 연속한 정점들 중 가장 위쪽의 정점을 p, 가장 아래쪽의 정점을 q 라고 합시다.

Open. 트리와 경로 뒤집기 쿼리

- p부터 q까지 정점들에 light edge 하나로 직접 연결된 모든 자식 정점의 dp값과 p부터 q까지 정점들에 대해서 $A[i] \times B[i]$의 합을 $r[p:q]$이라고 합시다.
- q의 자식 중 같은 chain에 속한 정점의 A, B 값이 주어졌을 때, $A[p], B[p], r[p:q]$를 구하는 함수를 세그먼트 트리의 각 노드가 관리하도록 구성합니다. 각 함수를 순서대로 f, g, h라고 하겠습니다.
- $f(A) = a_f A + b_f, g(B) = a_g B + b_g$ 꼴로 나타나며, a_f와 a_g는 0 또는 1입니다. 트리에 방향성이 있어 q에서 p로 도달 가능하면서 p에서 q로 도달 가능한 경우는 없기 때문에 한 노드의 a_f, a_g가 둘 다 1일 수는 없습니다.
- $h(A, B) = aA + bB + C$ 꼴로 나타납니다. $a_f a_g = 0$이므로 AB 항은 나타나지 않습니다.

- 이제 세그먼트 트리의 두 노드를 합치는 연산을 정의합시다. 두 노드 $(f_1(A), g_1(B), h_1(A, B))$, $(f_2(A), g_2(B), h_2(A, B))$를 합친 결과는 $(f_1 \circ f_2(A), g_1 \circ g_2(B), h_1(f_2(A), g_2(B)) + h_2(A, B))$입니다.
- f, g, h의 정의로부터 이와 같이 합치는 것이 타당함을 확인할 수 있으며, 위의 연산은 결합법칙이 성립하는 연산이므로, 세그먼트 트리를 이용하여 올바르게 관리할 수 있습니다.
- 한편, chain 내에서 간선을 여러 개를 뒤집어야 하기 때문에, 간선을 뒤집었을 때 f, g, h에 대한 정보도 똑같이 들고 다녀야 합니다. Lazy propagation을 이용하면 구간 뒤집기 쿼리를 $O(\log N)$에 처리할 수 있습니다.

- 이제 트리 전체에서 뒤집기 쿼리를 처리하는 방법을 확인해봅시다.
- 시작점 x가 주어졌을 때, x가 속한 체인에서 x의 부모 정점부터 체인의 가장 위쪽 정점인 $top[x]$까지 구간 뒤집기 쿼리를 처리합니다. 이후 세그먼트 트리를 이용하여 $top[x]$의 A, B, dp를 계산해줍니다. $top[x]$의 부모 정점이 존재할 경우, 그에 대응되는 세그먼트 트리의 노드의 f, g, h의 상수항을 구한 $top[x]$의 값을 바탕으로 점 업데이트해줍니다. 이후 $top[x]$의 부모 정점에 대하여 이를 반복합니다.
- 문제의 답은 root 정점이 포함된 chain에서 root의 dp를 계산하여 얻을 수 있습니다.
- 시간복잡도는 전처리 $O(N \log N)$, 쿼리 처리에 $O(Q \log^2 N)$ 입니다.