2025 SCSC 프로그래밍 경시대회 풀이

Official Solutions

by

2025 SCPC 출제진 및 검수진

Staff

운영

- 한성재 sungjae0506
- 김지훈 lycoris1600

출제

- sungjae0506
- lycoris1600
- sjh1224
- mujigae
- sjhi00
- ohwphil
- ksi4495
- pangitwise
- egwkim

검수

- hyperion1019
- yijw0930
- kdy40929
- jthis
- tony9402
- jinhan814
- utilForever
- qvixnh22

문제		의도한 난이도	출제자
3A	레퓨닛의 덧셈	**Easy**	sjhi00
3B/2A	주사위 피라미드	**Easy**	sungjae0506, mujigae
3C	SCSC 기차 놀이	**Easy**	pangitwise
3D/2B	물과 응애	**Easy**	mujigae
3E/2I	K-POP	**Easy**	sjh1224
3F/2G	현대모비스 V2X 자율주행 2	**Medium**	sungjae0506, mujigae, ohwp
1A	현대모비스 V2X 자율주행 1	**Medium**	sungjae0506, mujigae
3G/2F/1B	Sorting Replay at Jane Street	**Medium**	lycoris1600
3H/2E	SCSC 문자열 놀이	**Medium**	sjhi00
2J/1J	클-린드롬 부분 문자열	**Medium**	sjh1224

문제		의도한 난이도	출제자
2H	g-raph 신앙 (Easy)	**Hard**	mujigae
2C/1D	청군 백군	**Hard**	sjh1224
1I	g-raph 신앙 (Hard)	**Hard**	mujigae
2D/1G	비전 마법사 지환	**Hard**	mujigae, sungjae0506, ohwphil
1F	스시스시 왕국	**Hard**	ksi4495, sungjae0506
1E	격자 경로의 가중치	**Hard**	ohwphil
1H	관악산 정상에는 구름이 없다	**Hard**	sungjae0506
1C	최댓값과 쿼리	**Challenging**	sungjae0506
EX1	콩돌 놀이	**Easy**	mujigae
EX2	모모카와 열차 운행표	**Medium**	pangitwise

3A. 레퓨닛의 덧셈

`math, implementation`

출제진 의도 – **Easy**

- Div. 3 제출 97번, 정답 70명 (정답률 72.165%)
- Div. 3 처음 푼 사람: **dokdongsleeper**, 0분
- 출제자: `sjhi00`

- 풀이 1: 직접 계산하기
  - X 자리, Y 자리 레퓨닛을 직접 계산하여 더하면 됩니다.
  - X 자리 레퓨닛, 즉 $10^{X-1} + 10^{X-2} + \cdots + 10^1 + 1$은 반복문을 이용하여 계산할 수 있습니다. $\dfrac{10^X - 1}{9}$ 로도 계산할 수 있습니다. $2 \times 10^9 \le 2^{31} - 1$ 이므로, 32비트 정수 자료형을 이용하여 계산할 수 있습니다.
- 풀이 2: X와 Y의 대소 관계에 따라 답 출력하기
  - $X > Y$ 인 경우, $X - Y$ 개의 1과 Y 개의 2를 차례대로 출력하면 됩니다.
  - $X < Y$ 인 경우, $Y - X$ 개의 1과 X 개의 2를 차례대로 출력하면 됩니다.
  - $X = Y$ 인 경우, X 개의 2를 출력하면 됩니다.

3B/2A. 주사위 피라미드

`math`

출제진 의도 – **Easy**

- Div. 2 제출 147번, 정답 94명 (정답률 64.626%)
- Div. 2 처음 푼 사람: **motsuni04**, 2분
- Div. 3 제출 148번, 정답 62명 (정답률 41.892%)
- Div. 3 처음 푼 사람: **지현**, 7분
- 출제자: `sungjae0506`, `mujigae`

3B/2A. 주사위 피라미드

- 주사위에서 마주보는 면의 합은 항상 7입니다.
- 마주보는 면들은 제외하고 한 쪽만 보이는 경우를 생각해봅시다.
- 한 쪽만 보이는 면의 개수는 $1, 2, 3$개 중에 하나입니다.
- 위 세 경우에서 눈의 최대값과 최솟값의 합을 구해보면 각각 $7, 14, 21$입니다.
- 이를 통해 보이는 눈의 최댓값과 최솟값의 합은 $(\text{보이는 주사위 면의 개수}) \times 7$임을 알 수 있습니다.
- 바닥을 제외한 다섯 방향에서 볼 수 있고, 각 방향마다 $\dfrac{n(n+1)}{2}$개의 주사위 면이 보입니다.
- 따라서 $\dfrac{35n(n+1)}{2}$가 답입니다.

3C. SCSC 기차 놀이

`implementation`, `case_work`

출제진 의도 – **Easy**

- Div. 3 제출 127번, 정답 65명 (정답률 51.969%)
- Div. 3 처음 푼 사람: **roy1109**, 10분
- 출제자: `pangitwise`

- 임의의 두 지점 A와 B에 대하여, A에서 B로 갈 수 없다면 지점 A와 B는 서로 다른 공간으로 구분됩니다.
- 지점 A와 B가 서로 다른 공간에 있다는 것은, 공간 A'과 공간 B'이 서로의 공간 형성에 어떠한 영향도 미치지 않는다는 것을 뜻합니다.
- 설계도의 처음과 끝은 반드시 [과]이므로 그 사이에 들어가는 모든 차량의 왼쪽과 오른쪽은 서로 다른 공간으로 구분됩니다.

- 이제 차량들을 설계도에 따라 왼쪽에서 오른쪽으로 연결해 본다고 가정해 봅시다.
- n 번째 차량을 $n-1$ 번째 차량의 오른쪽에 연결하였을 때, n 번째 차량의 왼쪽에 있는 모든 기차간의 개수를 T_n 이라고 하고, $n+1$ 번째 차량을 새롭게 n 번째 차량의 오른쪽에 연결하였을 때 n 번째 차량과 $n+1$ 번째 차량 사이에 만들어지는 기차간의 개수를 Q_{n+1} 이라고 해 봅시다.
- 이때 $T_1 = 0$ 이고 $T_{n+1} = T_n + Q_{n+1}\ (n \geq 1)$ 입니다.
- 따라서 T_N 을 구하기 위해선 매번 Q_{n+1} 의 값을 알아내기만 하면 됩니다.

- 차량의 종류가 총 4개이므로 n번째 차량과 $n+1$번째 차량이 연결되는 방식의 가짓수는 $4 \times 4 = 16$가지입니다
- 정리하면 다음 표와 같습니다. 이를 이용해 매번 Q_{n+1}의 값을 알아낼 수 있습니다.

Q_{n+1}	C-[	C-]	S-2	S-5
C-[	1	1	1	1
C-]	0	1	1	1
S-2	1	1	2	1
S-5	1	1	1	2

- 16가지 경우를 하드 코딩을 해도 되지만 표를 잘 관찰하여 조건 분기를 줄일 수도 있습니다.
- 전체 시간복잡도 $\mathcal{O}(N)$으로 해결할 수 있습니다.

3D/2B. 물과 응애

`greedy`
출제진 의도 – **Easy**

- Div. 2 제출 362번, 정답 73명 (정답률 20.166%)
- Div. 2 처음 푼 사람: **숭실자전25김티나**, 9분
- Div. 3 제출 310번, 정답 25명 (정답률 8.065%)
- Div. 3 처음 푼 사람: **Gemini**, 27분
- 출제자: `mujigae`

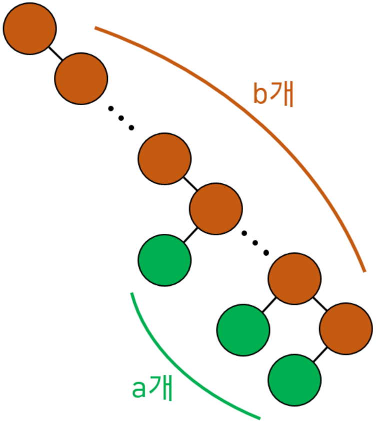
- 모든 O에 대해 자신의 왼쪽과 오른쪽에 각각 자신과 짝지을 수 있는 H가 하나씩 존재해야 합니다.
- 우선 각 O를 기준으로 왼쪽에 있는 H부터 짝지을 수 있는지 알아 봅시다. 왼쪽부터 O를 하나씩 세었을 때, 지금까지 나온 H의 개수가 O의 개수 이상이라면 지금까지 나온 모든 O에 대해 왼쪽의 H를 짝지어 줄 수 있습니다.
- 오른쪽의 H도 오른쪽부터 같은 방법을 적용하여 해결할 수 있습니다.
- 시간복잡도는 $\mathcal{O}(N)$ 입니다.

3E/2I. K-POP

`ad-hoc, constructive`
출제진 의도 – **Easy**

- Div. 2 제출 209번, 정답 83명 (정답률 40.670%)
- Div. 2 처음 푼 사람: ??, 27분
- Div. 3 제출 102번, 정답 23명 (정답률 22.549%)
- Div. 3 처음 푼 사람: **니은**, 52분
- 출제자: `sjh1224`

- $a \leq b$인 임의의 양의 정수 a, b에 대해, 다음과 같은 방법을 이용하면 리프 노드 수가 a이고 리프가 아닌 노드 수가 b인 이진 트리를 항상 구성할 수 있습니다.



- 따라서 $a \times b = K$ 인 (a, b) 중 노드 수인 $a + b$가 최소가 되는 (a, b) 를 찾아야 합니다.
- 이런 (a, b) 는 K 의 약수를 하나씩 확인해 보면 $\mathcal{O}(K)$ 또는 $\mathcal{O}(\sqrt{K})$ 시간에 찾을 수 있습니다.
- 이렇게 (a, b) 를 찾은 뒤, 위에서 논의한 방식으로 정답이 되는 이진 트리를 구성하면 됩니다.

3F/2G. 현대모비스 V2X 자율주행 2

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

- Div. 2 제출 105번, 정답 16명 (정답률 15.238%)
- Div. 2 처음 푼 사람: **숭실자전25김티나**, 29분
- Div. 3 제출 41번, 정답 0명 (정답률 0.000%)
- Div. 3 처음 푼 사람: **-**, -분
- 출제자: `sungjae0506`, `mujigae`, `ohwphil`

- 우선, 두 경로가 겹치지 않기 위해서는 한 경로는 U로 시작하여 R로 끝나야 하며, 다른 경로는 R로 시작하여 U로 끝나야 합니다.
- 일반성을 잃지 않고 첫 번째 경로가 U로 시작하여 R로 끝나며, 두 번째 경로가 R로 시작하여 U로 끝나게 경로를 수정한다고 합시다.
- 이제 $i = 1, 2, \cdots, 2N$에 대하여 (첫 번째 경로의 i번째 이동까지 나온 U의 개수) - (두 번째 경로의 i번째 이동까지 나온 U의 개수)를 관리하는 배열을 관리해 봅시다.
- 이 배열의 1번째 원소는 반드시 1이고, $2N-1$번째 원소 역시 반드시 1이고, $2N$번째 원소는 반드시 0입니다.
- 두 경로의 시작과 끝이 위에서 적은 대로 고정되었다고 할 때, 두 경로가 만나지 않기 위한 필요충분조건은 바로 위의 배열의 $2N-1$번째 원소까지 보았을 때 0 이하인 원소가 하나도 존재하지 않아야 합니다.

- 이제 이 배열의 모든 원소가 1 이상이 되도록 하고 싶습니다.
- 첫 번째 경로의 R과 U 이동을 교환하면, R 이동이 더 앞에 있었다면 R이 있던 위치부터 U가 있던 위치의 바로 앞 위치까지에 대하여 배열에 1 이 더해집니다.
- 비슷한 논리로, 두 번째 경로의 U와 R 이동에 대해서도 비슷한 논의를 해볼 수 있습니다.
- 그래서 두 경로의 시작과 끝을 고정한 직후 배열의 $2N-1$ 번째 원소까지의 최솟값을 m 이라고 하면 적어도 $1-m$ 번의 교환이 필요하다는 사실을 알 수 있습니다.

- 먼저, $m \le 0$이라면 $2N$ 번째 위치 이전에 반드시 0이 등장합니다.
- $2N$ 번째 위치를 제외한 위치에서 가장 먼저 나타나는 0이 있다면 그 0의 직전에는 반드시 1이 있어야 하고, 가장 나중에 나타나는 0이 있다면 그 0의 직후에는 반드시 1이 있어야 합니다.
- 처음 나타나는 0의 위치에서 반드시 첫 번째 경로에는 R 이동이 있으며, 두 번째 경로에는 U 이동이 있습니다. 나중에 나타나는 0의 직후 위치에서 반드시 첫 번째 경로는 U 이동이 있으며, 두 번째 경로에는 R 이동이 있습니다.
- 이제 첫 번째 경로에서 앞서 언급한 위치들의 이동을 교환하면 배열의 최솟값을 항상 1씩 증가시킬 수 있습니다.

- 참고로 처음에 경로의 시작 이동과 끝 이동을 고정할 때, 두 경로 모두에 대하여 최대 한 번의 교환으로 시작 이동과 끝 이동을 고정할 수 있습니다.
- 만약 한 경로의 시작 이동과 끝 이동이 모두 올바르지 않다면, 시작점과 끝점의 이동을 교환해 버리면 됩니다.
- 만약 시작 이동만 잘못되었다면, 시작 이동에 들어가야 하는 이동이 존재하는 가장 나중 위치와 교환하는 것이 최적입니다. 끝 이동도 비슷한 방법으로 교환하면 됩니다. 이는 앞서 논의한 배열의 관점에서 보면 증명할 수 있습니다.
- 최종적으로 두 경로의 시작 이동과 끝 이동을 고정하는 데에 드는 이동 수에 $1 - m$을 더한 만큼의 교환이 필요합니다.
- 두 경로의 역할을 바꿔서 계산하는 것도 잊지 말아 주세요!

1A. 현대모비스 V2X 자율주행 1

`ad_hoc`, `prefix_sum`

출제진 의도 – **Medium**

- Div. 1 제출 79 번, 정답 36 명 (정답률 45.570%)
- Div. 1 처음 푼 사람: **SNUPS 첩자**, 4 분
- 출제자: `sungjae0506`, `mujigae`

- R을 지날 때마다 $+1$, U를 지날 때 마다 -1을 더하는 누적합을 생각해봅시다.
- 두 경로가 겹치기 않게 하기 위해서는 처음과 끝을 제외한 누적합이 모두 양수거나 모두 음수여야 합니다.
- 한 번 swap연산을 할 때마다 누적합 배열의 한 항을 2만큼 줄이거나 늘릴 수 있습니다.
- 누적합 배열을 A라 할 때 $\min(\sum_{i=1}^{2N-1} \max(0, 1 + \frac{A[i]}{2}), \sum_{i=1}^{2N-1} \max(0, 1 - \frac{A[i]}{2}))$가 답입니다.

3G/2F/1B. Sorting Replay at Jane Street

`ad_hoc, recursion, sorting`

출제진 의도 – **Medium**

- Div. 1 제출 90번, 정답 39명 (정답률 43.333%)
- Div. 1 처음 푼 사람: **SNUPS 첩자**, 16분
- Div. 2 제출 233번, 정답 38명 (정답률 16.309%)
- Div. 2 처음 푼 사람: **숭실자전25김티나**, 47분
- Div. 3 제출 20번, 정답 1명 (정답률 5.000%)
- Div. 3 처음 푼 사람: **asterope**, 132분
- 출제자: `lycoris1600`

- 특정 인덱스를 기준으로 정렬을 한다고 생각해 봅시다.
- Stable sort를 하는 경우 가능한 정렬의 가짓수는 1가지 입니다.
- Unstable sort를 하는 경우 가능한 정렬의 가짓수는, 구분할 수 없는 원소들의 개수에 의해 결정되게 됩니다.
- 결국, 가장 마지막으로 나온 unstable sort와 그 이후에 나오는 쿼리들만 고려하면 됩니다.
- 의도한 풀이는 크게 2가지입니다.

- 1. Recursion $\mathcal{O}(NQ\log(N))$
  - 가장 마지막 쿼리부터 하나씩 처리합니다.
  - 현재 쿼리에 의해 순서가 결정되는 원소들끼리는 앞쪽 쿼리와 무관하게 순서가 고정됩니다.
  - 현재 쿼리에 의해 순서가 결정되지 않는 원소들끼리는
    - Stable sort인 경우 앞쪽 쿼리를 확인하기 위해 재귀를 한번 더 돌려야 합니다.
    - Unstable sort인 경우 순서가 결정되지 않는 원소 K개에 대해 모든 factorial(K) 가지의 경우가 다 가능합니다.

- 2. Lazy update $\mathcal{O}(N^2 \log(N) + Q)$
  - 마지막 Unstable sort 이후에 등장한 인덱스들만 저장해 둡니다.
  - 정렬 기준은 위에서 저장해 둔 인덱스들에 의해서 이루어집니다. 즉, 저 인덱스들에 있는 값이 하나라도 다른 배열들끼리는 구분이 가능하며, 모두 같은 배열들끼리는 구분이 불가능합니다.
  - 구분이 불가능한 배열 K 개에 대해서는 모든 factorial(K) 가지의 경우가 다 가능합니다.

- N이 Q보다 작다면 Lazy update가 유리하고, N이 Q보다 크다면 Recursion이 유리합니다.
- 두 방식 모두 좋은 풀이라고 생각해 제한을 $N = Q$로 설정했습니다.

3H/2E. SCSC 문자열 놀이

`math, bitmask, dp`
출제진 의도 – **Medium**

- Div. 2 제출 122번, 정답 5명 (정답률 4.098%)
- Div. 2 처음 푼 사람: **25학번 2회차 새내기**, 95분
- Div. 3 제출 29번, 정답 0명 (정답률 0.000%)
- Div. 3 처음 푼 사람: **-**, -분
- 출제자: `sjhi00`

- 만들어진 문자열의 길이를 M 이라고 하겠습니다.
- $1 \le i \le M$ 일 때, i 번째로 추가된 문자가 S이면 $b_i = 0$, C이면 $b_i = 1$ 이라고 하겠습니다. 또한, i 번째 문자까지만 추가했을 때 얻은 점수를 d_i 라고 하겠습니다. 편의상 $d_0 = 0$ 이라고 하겠습니다.
- 이때, $d_i = 2d_{i-1} + S + (C - S)b_i$ ($1 \le i \le M$) 이 성립합니다.
- $\frac{d_i}{2^i} = \frac{d_{i-1}}{2^{i-1}} + \frac{S + (C - S)b_i}{2^i}$ ($1 \le i \le M$) 이므로,
  $\frac{d_M}{2^M} = \sum_{i=1}^{M} \left(\frac{d_i}{2^i} - \frac{d_{i-1}}{2^{i-1}} \right) = \sum_{i=1}^{M} \frac{S + (C - S)b_i}{2^i}$ 입니다.
- 즉, $d_M = \sum_{i=1}^{M} 2^{M-i} (S + (C - S)b_i) = \left(2^M - 1\right) S + (C - S) \sum_{i=1}^{M} b_i 2^{M-i}$ 입니다.

- $P = \sum_{i=1}^{M} b_i 2^{M-i}$ 라 하면, $0 \le P \le 2^M - 1$ 을 만족합니다.
- 즉, $N = d_M = \left(2^M - 1\right) S + (C - S)P = \left(2^M - 1 - P\right) S + CP$ 입니다.
- $S, C \ge 1$ 이므로, $N \ge \left(2^M - 1 - P\right) + P = 2^M - 1$ 입니다.
- 만약 $S \ne C$ 인 경우, $N = \left(2^M - 1\right) S + (C - S)P$ 는 P 에 관한 일차방정식이 되며, 이 식을 만족하는 정수 P 는 많아야 1 개뿐입니다.
- 만약 정수 P 가 $0 \le P \le 2^M - 1$ 범위에서 유일하게 결정될 경우, $b_1, \cdots, b_M$ 도 유일하게 결정되므로, 만들어진 문자열도 자동으로 결정됩니다.
- 따라서 $2^M - 1 \le N$ 을 만족하는 각각의 양의 정수 M 에 대하여, 조건을 만족하는 P 값이 존재하는지, 존재한다면 만들어진 문자열이 SCSC를 부분 문자열로 가지는지 확인하면 됩니다.

- 이제 $S = C$ 인 경우만 고려하면 됩니다.
- $N = d_M = \left(2^M - 1\right) S$ 이므로, M 의 값으로 가능한 후보는 많아야 1 개뿐입니다.
- 만약 이를 만족하는 양의 정수 M 이 존재할 경우, S와 C로만 이루어진 길이가 M 인 임의의 문자열에 대하여 최종 점수는 항상 N 점이 됩니다.
- $M < 4$ 이면 답은 0 이며, $M \geq 4$ 인 경우만 고려하면 됩니다.
- $3 \leq i \leq M$, $0 \leq j < 8$ 인 각 i, j 에 대하여, 길이가 i 인 문자열 중에서, 길이가 3 인 접미사(suffix)가 H_j 가 되도록 하면서 SCSC를 부분 문자열로 가지지 **않는** 경우의 수를 $dp[i][j]$ 라고 하겠습니다.
- (단, $[H_0, \cdots, H_7] = [\text{SSS}, \text{SSC}, \text{SCS}, \text{SCC}, \text{CSS}, \text{CSC}, \text{CCS}, \text{CCC}]$)

- 이때, $dp[i][0], \cdots, dp[i][7]$ 을 각각 $dp[i-1][0], \cdots, dp[i-1][7]$ 에 관한 식으로 나타낼 수 있습니다. $(4 \leq i \leq M)$
- $dp[i][0] = dp[i][1] = dp[i-1][0] + dp[i-1][4]$
- $dp[i][2] = dp[i][3] = dp[i-1][1] + dp[i-1][5]$
- $dp[i][4] = dp[i-1][2] + dp[i-1][6], \qquad dp[i][5] = dp[i-1][6]$
- $dp[i][6] = dp[i][7] = dp[i-1][3] + dp[i-1][7]$
- 구하려고 하는 답은 $2^M - (dp[M][0] + \cdots + dp[M][7])$ 입니다.
- 각 테스트 케이스마다, $S \neq C$ 일 때 $\mathcal{O}\left((\log N)^2\right)$ 또는 $\mathcal{O}(\log N)$ 의 시간복잡도로 구현 가능하고, $S = C$ 일 때 $\mathcal{O}(\log N)$ 의 시간복잡도로 구현 가능합니다.
- 전체 시간복잡도는 $\mathcal{O}\left(T(\log N)^2\right)$ 또는 $\mathcal{O}(T \log N)$ 입니다.

2J/1J. 클-린드롬 부분 문자열

`manacher, hashing`
출제진 의도 – **Medium**

- Div. 1 제출 89번, 정답 35명 (정답률 39.326%)
- Div. 1 처음 푼 사람: **ㅁㄴㅇㄹasdf**, 11분
- Div. 2 제출 74번, 정답 10명 (정답률 13.514%)
- Div. 2 처음 푼 사람: **smots gaming**, 81분
- 출제자: `sjh1224`

– 풀이 1: 해싱을 이용하는 풀이

- 하나의 K 와 $1 \le i \le K$ 인 정수 i 를 고정해봅시다.
- 문자열을 $S[i...i+K], S[i+K+1...i+2K], \cdots$ 으로 분할하고, 분할된 각 문자열을 롤링 해시하여 배열 $H = [H_1, H_2, \cdots, H_M]$ 을 만듭니다. 이때 해시 값은 사전에 S 를 누적 합 형식으로 전처리해두면 각각 $\mathcal{O}(1)$ 에 계산할 수 있습니다.
- 배열 H를 하나의 문자열로 보고 매내처 알고리즘을 사용하면, H 의 팰린드롬 부분 문자열의 개수를 $\mathcal{O}(M) = \mathcal{O}(N/K)$ 시간에 구할 수 있습니다. (BOJ 16163 : 15164번_제보)
- 위에서 구한 수는 S 의 K-린드롬 부분 문자열 중, 시작 문자의 인덱스가 $\mod K$ 로 i 와 같은 것의 개수입니다.
- 이들을 모든 $i (1 \le i \le K)$ 에 대해 더해 주면, S 의 K-린드롬 부분 문자열의 개수를 $K \times \mathcal{O}(N/K) = \mathcal{O}(N)$ 시간에 구할 수 있습니다.
- 모든 K 에 대해 위 작업을 반복해 주면, 총 수행 시간은 $\mathcal{O}(N^2)$ 이 됩니다.

- 풀이 2: 해싱을 이용하지 않는 풀이
  - 다시, 하나의 K와 $1 \leq i \leq K$인 정수 i를 고정해봅시다. 길이 N인 배열 A를 구성하고자 합니다.
  - S에서 인덱스가 $\mod K$로 i와 같은 문자들을 모아, 순서대로 이어 붙인 문자열을 T라고 합시다. 즉 $T_j = S_{(j-1)K+i}, j = 1, 2, \cdots$ 입니다.
    T의 매내처 배열을 구합니다. 이때 길이 짝수인 팰린드롬은 무시하고, 길이 홀수인 팰린드롬에 해당하는 최대 반지름을 배열 M만 먼저 생각합시다.
  - 이때 $|M| = |T|$이므로, T를 가져온 위치에 해당하는 A의 자리에 그대로 M을 가져다 놓습니다. 즉, $A_{(j-1)K+i} = M_j, j = 1, 2, \cdots, |M|$으로 정의합니다.

- 각 i에 대해 위 작업을 수행하면, 배열 A를 $\mathcal{O}(K \times N/K) = \mathcal{O}(N)$ 시간에 완성할 수 있습니다.
- 각 $i = 1, 2, \cdots, N - K + 1$에 대해, $m_i = \min(A_i, A_{i+1}, \cdots, A_{i+K-1})$은 무엇을 의미할까요? 이 값은 $S[i...i + K - 1]$이 중심인 S의 K-린드롬 부분 문자열의 최대 반지름입니다.
- 따라서 각 i에 대해 m_i를 구해 주어 $S[i...i + K - 1]$이 중심인 K-린드롬 부분 문자열의 개수를 구하고 이들을 합하면, S의 홀수 길이 K-린드롬 부분 문자열의 개수를 구할 수 있습니다.

- 각 i에 대한 m_i 들의 배열은, std::set이나 우선순위 큐를 이용해 $\mathcal{O}(N \log N)$ 시간에 구하거나, 덱을 이용한 구간 최댓값/최솟값 트릭 (BOJ 11003 : 최솟값 찾기) 를 이용해 $\mathcal{O}(N)$ 시간에 구할 수 있습니다.
- 위 내용을 종합하면, $\mathcal{O}(N \log N)$ 또는 $\mathcal{O}(N)$ 시간에 S 의 홀수 길이 K-린드롬 부분 문자열의 개수를 구할 수 있습니다.
- 짝수 길이 K-린드롬 부분 문자열의 개수도 각 매내처 배열의 짝수 번째 값들을 이용해 동일한 방법으로 구해줄 수 있습니다.
- 위 작업을 모든 K 에 대해 반복해 주면, 정답을 구할 수 있고 총 수행 시간은 $\mathcal{O}(N^2 \log N)$ 또는 $\mathcal{O}(N^2)$ 이 됩니다.

2H. g-raph 신앙 (Easy)

`dp_tree`, `combinatorics`

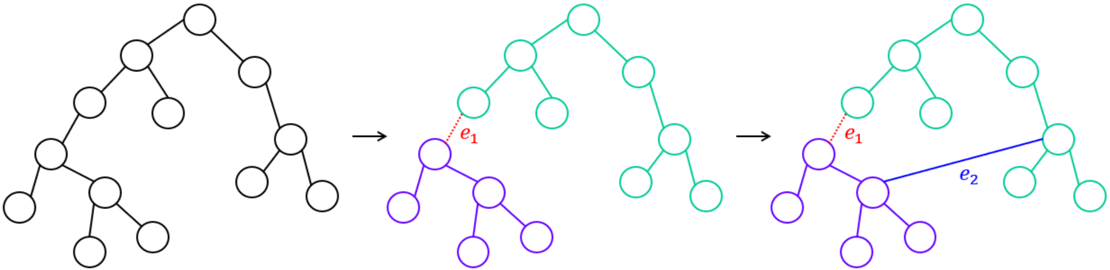
출제진 의도 – **Hard**

- Div. 2 제출 16번, 정답 1명 (정답률 12.500%)
- Div. 2 처음 푼 사람: **종이**, 197분
- 출제자: `mujigae`

2H. g-raph 신앙 (Easy)

- 전체 경우의 수와 조건을 만족하는 경우의 수를 각각 구하여 확률을 계산합시다.
- 전체 경우의 수는 단순한 계산으로 어렵지 않게 구할 수 있습니다. 첫 마술에서 지우기 위해 선택할 수 있는 간선의 수는 $N-1$ 개입니다. 다음으로 간선을 생성하기 위해 선택할 수 있는 정점쌍의 수는 $\binom{N}{2}-(N-2)$ 개입니다. 두 번째 마술도 이와 동일합니다.
- 전체 경우의 수는 $(N-1)^2(\binom{N}{2}-N+2)^2$ 가지입니다.
- 위의 식이 분모에 해당하기 때문에 주어진 제한에서 분모가 $998\,244\,353$ 의 배수가 될 수 없다는 점도 알 수 있습니다.

- 조건을 만족하는 경우의 수를 세기 위해 첫 번째 마술에서 일어나야 하는 일부터 분석해 봅시다.
- 첫 번째 마술에서 제거한 간선을 e_1 이라고 합시다. e_1 을 제거한 순간 트리는 두 개의 컴포넌트로 나뉩니다. 조건을 만족하기 위해서는 반드시 각 컴포넌트에서 정점을 하나씩 골라 간선을 생성해야 합니다. 이때 생성된 간선은 e_2 라고 합시다.

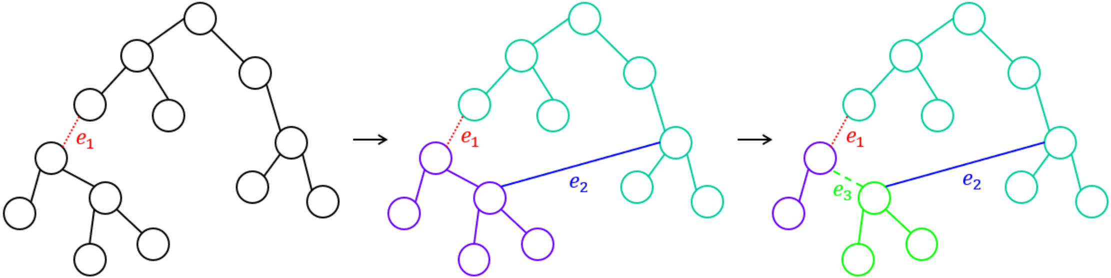


- 두 번째 마술에서도 마찬가지로 어떤 간선을 제거한 후 나타나는 두 개의 컴포넌트를 연결하는 간선을 이어 주어야 합니다.
- 간선을 제거했을 때 나타나는 컴포넌트의 크기는 DFS를 통해 얻을 수 있습니다. 초기 트리에서 DFS를 통해 컴포넌트의 크기를 전처리한 후 모든 e_1을 순회하며 새로운 트리를 생성하고, 다시 모든 e_2를 순회하며 DFS를 통해 컴포넌트의 크기를 계산하면 $\mathcal{O}(N^3)$에 원하는 결과를 얻을 수 있습니다.
- $\mathcal{O}(N^3)$으로는 주어진 제한에서 문제를 해결할 수 없기 때문에 더 빠른 방법을 생각해 보아야 합니다. 핵심은 두 번째 마술에서 선택할 수 있는 간선에 있습니다. e_2를 제외하면 모두 기존에 존재하는 간선이기 때문에, 초기 트리의 DFS에서 전처리한 컴포넌트의 크기를 활용할 수 있는 방법을 떠올려 봅시다.

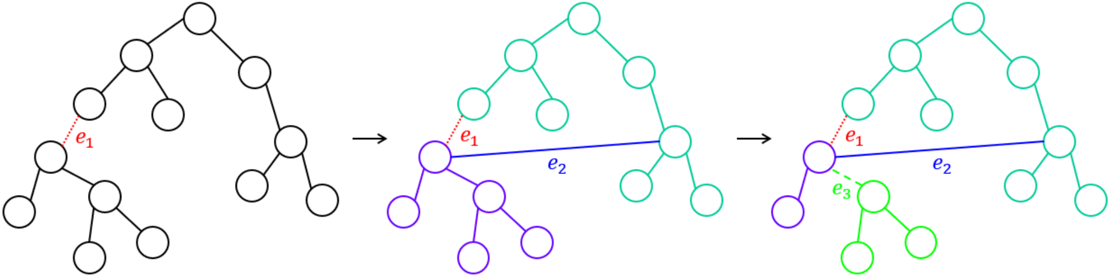
2H. g-raph 신앙 (Easy)

- 두 번째 마술에서 제거할 간선으로 e_2 를 선택할 경우, 어디에 e_2 를 생성했는지와 관계 없이 나타나는 컴포넌트의 크기는 e_1 을 제거했을 때 나타나는 컴포넌트의 크기와 동일하므로 이를 그대로 이용할 수 있습니다.
- e_2 를 제외한 다른 간선을 선택할 경우, 어떤 간선을 선택하든 기존에 초기 트리에 존재하는 간선 중 하나입니다. 이때 선택된 간선을 e_3 라 합시다.
- 초기 트리에서 임의의 루트를 기준으로 DFS를 실행하여 모든 간선에 대해 각 간선이 연결하는 두 정점 중 자식이 속한 컴포넌트의 개수를 전처리합시다. 간선 e_i 에 대해 이 값을 sub_i, 자식 정점을 x_i 라 하겠습니다.
- sub_i 를 이용하면 e_1 와 e_3 의 관계를 세 가지 경우로 나누어 원하는 값을 빠르게 계산할 수 있습니다.

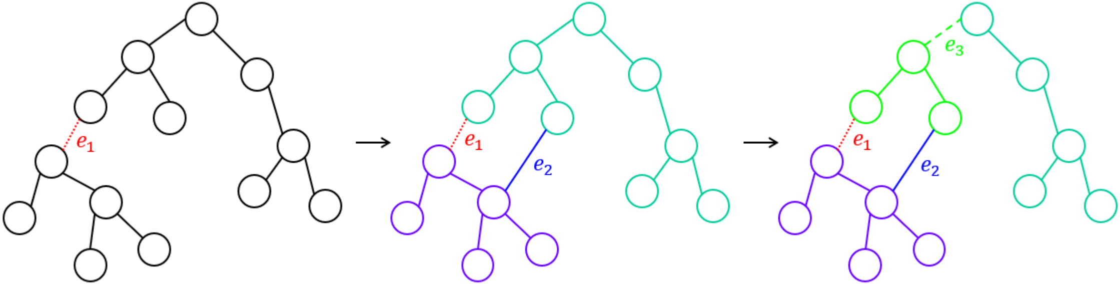
- 1. $LCA(x_1, x_3) = x_1$
- e_2 가 e_3 의 자식 컴포넌트와 연결된 경우
- $(sub_1 - sub_3) \cdot (N - sub_1 + sub_3) \cdot (N - sub_1) \cdot sub_3$



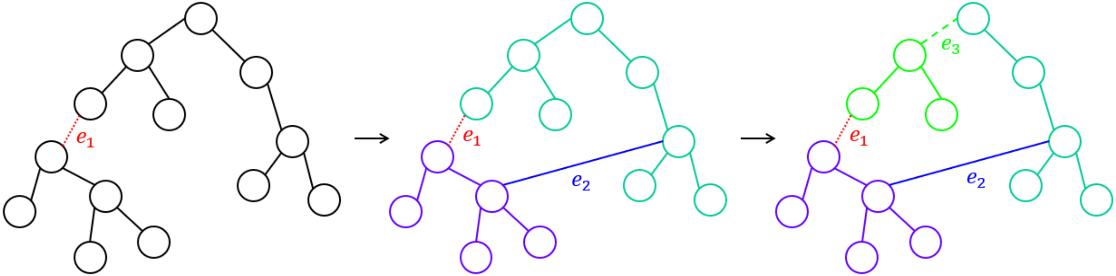
- 1. $LCA(x_1, x_3) = x_1$
- e_2 가 e_3 의 부모 컴포넌트와 연결된 경우
- $(N - sub_3) \cdot sub_3 \cdot (N - sub_1) \cdot (sub_1 - sub_3)$



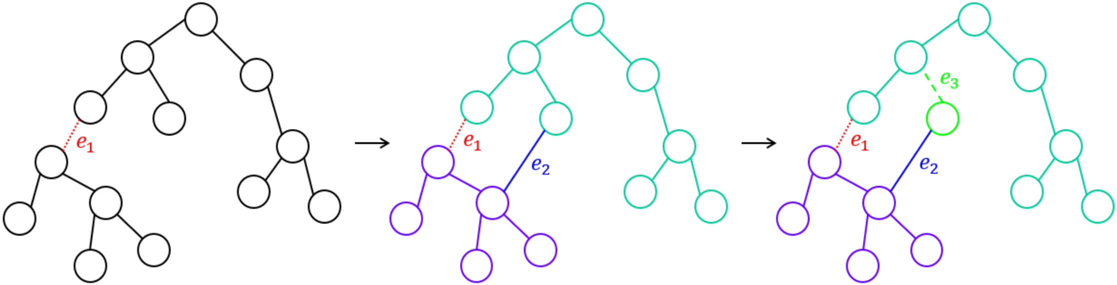
- 2. $LCA(x_1, x_3) = x_3$
- e_2 가 e_3 의 자식 컴포넌트와 연결된 경우
- $(N - sub_3) \cdot sub_3 \cdot sub_1 \cdot (sub_3 - sub_1)$



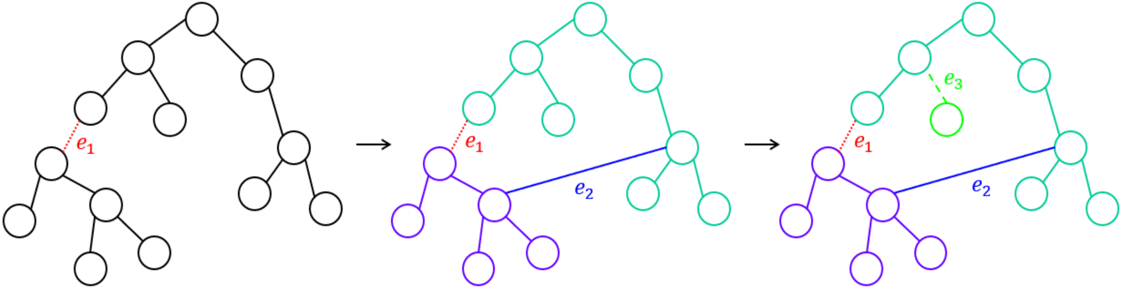
- 2. $LCA(x_1, x_3) = x_3$
- e_2가 e_3의 부모 컴포넌트와 연결된 경우
- $(sub_3 - sub_1) \cdot (N - sub_3 + sub_1) \cdot sub_1 \cdot (N - sub_3)$



- 3. $LCA(x_1, x_3) \neq x_1$ and $LCA(x_1, x_3) \neq x_3$
- e_2 가 e_3 의 자식 컴포넌트와 연결된 경우
- $(N - sub_3 - sub_1) \cdot (sub_3 + sub_1) \cdot sub_1 \cdot sub_3$



- 3. $LCA(x_1, x_3) \neq x_1$ and $LCA(x_1, x_3) \neq x_3$
- e_2 가 e_3 의 부모 컴포넌트와 연결된 경우
- $(N - sub_3) \cdot sub_3 \cdot sub_1 \cdot (N - sub_3 - sub_1)$



2H. g-raph 신앙 (Easy)

- 서로 다른 두 간선을 모두 순회하며 수식을 통해 얻은 값을 계산해 주면 두 번째 마술에서 e_2를 제외한 다른 간선을 선택하는 경우의 수를 얻을 수 있습니다.
- 모든 정점쌍에 대한 LCA를 DFS와 함께 전처리하면 $\mathcal{O}(N^2)$에 문제를 해결할 수 있습니다. 리루팅 또는 ETT 등을 활용한 $\mathcal{O}(N^2)$ 풀이도 존재합니다.
- 위에서 구한 식을 더 다듬으면 $\mathcal{O}(N)$에도 문제를 해결할 수 있습니다. 한 번 찾아 보세요.
- 참고로 g-raph는 지-래프로 읽습니다. 그들이 기린을 숭배하기 때문입니다.

2C/1D. 청군 백군

`parametric search, 2-SAT`
출제진 의도 – **Hard**

- Div. 1 제출 48번, 정답 22명 (정답률 45.833%)
- Div. 1 처음 푼 사람: **SNUPS 첩자**, 51분
- Div. 2 제출 13번, 정답 1명 (정답률 7.692%)
- Div. 2 처음 푼 사람: **숭실자전25김티나**, 141분
- 출제자: `sjh1224`

- 정수 p에 대해 "두 팀의 단합력 중 최솟값이 p 이상이 될 수 있는가??"를 판정하는 결정 문제를 먼저 풀어봅시다.
- 위 조건을 만족하려면, 팀 배정이 아래 두 가지 조건을 만족하면 됩니다.
  - (1) 각 조별로 신청한 팀이 아닌 팀에 배정되는 인원이 최대 1명
  - (2) F_{ij}가 p 미만인 모든 $i, j (i \neq j)$에 대해, i와 j가 다른 팀에 속함
- 이 두 가지 조건은 다음과 같은 2-SAT 문제를 푸는 것으로 완벽하게 모델링할 수 있습니다.

- 각 학생 i에 대해, boolean variable x_i를 i가 백군에 속하면 0, 청군에 속하면 1로 정의합니다.
- 조건 (1)을 모델링하기 위해, 학생 $i_1, i_2, \cdots, i_l$가 한 조를 짜 청군으로 신청했다고 해 봅시다. $i_1, i_2, \cdots, i_l$ 중 청군으로 배정하지 않는 인원이 최대 1명이어야 하는데, 이는 "임의의 $i_j, i_k (j \neq k)$에 대해 i_j와 i_k 중 한 명은 청군으로 배정해야 한다"와 동치입니다. 따라서 $\bigwedge_{j \neq k} (x_j \vee x_k)$로 모델링할 수 있습니다. 백군으로 신청한 조들에 대해서는 x_j, x_k 대신 $\neg x_j, \neg x_k$를 써 주면 될 것입니다.
- 다음으로 조건 (2)를 모델링해봅시다. i와 j가 다른 팀에 속한다는 것을 $(x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)$로 쓸 수 있습니다. 따라서 $\bigwedge_{F_{ij} < p, i \neq j} [(x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)]$로 모델링해줄 수 있습니다.

- 위와 같은 모델링에서 조건 (1)에 대한 CNF 절의 개수는 $\sum \binom{\text{각 조의 크기}}{2} = \mathcal{O}(N^2)$ 이고, 조건 (2)에 대한 CNF 절의 개수는 $\binom{N}{2}$ 에 바운드되므로 $\mathcal{O}(N^2)$ 입니다. 즉 CNF 절의 총 개수는 $\mathcal{O}(N^2)$ 입니다.
- 따라서 SCC 알고리즘을 이용해 위에서 정의한 2-SAT을 선형 시간에 풀어 주면, $\mathcal{O}(N^2)$ 시간에 "두 팀의 단합력 중 최솟값이 p 이상이 될 수 있는가??"를 판정할 수 있습니다.
- 위 결정 문제의 해답을 이용해 $[0, 10^9 + 1]$ 구간에서 parametric search를 수행하면, 두 팀의 단합력 중 최솟값을 $\mathcal{O}(N^2 \log(10^9))$ 시간에 구할 수 있습니다.

11. g-raph 신앙 (Hard)

`dp_tree, combinatorics`
출제진 의도 – **Hard**

- Div. 1 제출 9 번, 정답 6 명 (정답률 66.667%)
- Div. 1 처음 푼 사람: **ㅁㄴㅇㄹasdf**, 179분
- 출제자: `mujigae`

- 본 문제를 풀기 위해 더 작은 제한에서 가능한 $\mathcal{O}(N^2)$ 의 풀이를 정리했다면, 방법에 따라 큰 제한으로 넘어가기 위한 길이 더 빨리 보일 수 있습니다.
- g-raph 신앙 (Easy)에서 보았던 $\mathcal{O}(N^2)$ 풀이를 다시 살펴 봅시다. 해당 풀이를 기반으로 최적화하는 방법을 찾을 것이기 때문에, 용어도 그대로 빌려 사용하겠습니다.
- $\mathcal{O}(N^2)$ 풀이의 핵심은 두 번째 마술에서 선택하는 간선이 e_2 인 경우와 e_2 가 아닌 경우로 나누고, 다시 e_2 가 아닌 간선을 선택한 상황에서 x_1 과 x_3 의 관계에 따라 경우를 나누어 가능한 경우의 수를 세는 것이었습니다.
- x_1 과 x_3 의 관계에 따라 경우를 나누어 수식을 정리한 것에 집중해 봅시다.

- 각 경우마다 e_2가 e_3의 자식 컴포넌트와 부모 컴포넌트 중 어떤 컴포넌트에 연결되었는지에 따라 수식을 다르게 정리하여 계산했습니다. 두 식을 잘 관찰해 보면, 공통점이 있습니다.
- 두 식 모두 4개 항의 곱으로 이루어져 있습니다.
- 공통항을 2개씩 갖고 있습니다.
- 두 식을 더하여 하나의 식으로 합치면, 추가적인 전처리를 통해 값을 빠르게 계산할 수 있는 방법이 보이기 시작합니다.
- 각 간선이 e_1인 경우를 순회하며 계산하기 위해 필요한 값들을 알아 봅시다.

- 1. $LCA(x_1, x_3) = x_1$
- $(N - sub_1) \cdot (2N - sub_1) \cdot [sub_1 \cdot sub_3 - {sub_3}^2]$
- 기존에 전처리했던 sub_i 를 이용해 $\sum sub_i$ 와 $\sum {sub_i}^2$ 을 추가로 전처리하여 계산합니다.

- 2. $LCA(x_1, x_3) = x_3$
- $sub_1 \cdot (N + sub_1) \cdot [-{sub_3}^2 + (N + sub_1) \cdot sub_3 - N \cdot sub_1 \cdot 1]$
- 여기서 주의해야 하는 점은 sub_3 을 위해 기존에 전처리한 값을 사용할 수 없다는 점입니다. 그 이유는 바로, LCA 조건 때문입니다. 새로운 DP 값 stem을 다음과 같이 정의합시다.
- $stem_i :=$ i 번 정점부터 루트 정점까지 최단 경로로 이동할 때 거치는 정점의 개수
- 이제 $\sum stem_i, \sum {stem_i}^2$ 과 각 정점의 depth를 추가로 전처리하여 계산합니다.

- 3. $LCA(x_1, x_3) \neq x_1$ and $LCA(x_1, x_3) \neq x_3$
- $sub_1 \cdot (N + sub_1) \cdot [(N - sub_1) \cdot sub_3 - {sub_3}^2]$
- 여기서도 마찬가지로 모든 경우를 한 번에 처리하기 위해 단순히 sub_i 의 합 등을 사용하면 안 됩니다.
- 대신 우리가 1 번과 2 번 경우를 위해 전처리한 값을 이용해 계산하는 방법을 사용할 수 있습니다. 루트 정점에 대한 DP 값에서 i 번 정점의 sub_i 관련된 값과 $stem_i$ 관련된 값을 각각 뺀 값을 사용하면, LCA 조건을 만족하는 정점들에 대한 DP 값의 합을 이용할 수 있게 됩니다.

- $\mathcal{O}(N)$에 문제를 해결할 수 있습니다. 수식을 정리하는 방법이 다양하여, 자신이 편한 DP 구조를 이용해 조각을 맞추면 됩니다.
- 구현은 복잡하지만 리루팅을 이용한 풀이도 가능합니다.

2D/1G. 비전 마법사 지환

`ad_hoc`, `graph_traversal`
출제진 의도 – **Hard**

- Div. 1 제출 28 번, 정답 1 명 (정답률 3.571%)
- Div. 1 처음 푼 사람: **제16대 트릭컬 마이너 갤러리 갤주 셰럼**, 177분
- Div. 2 제출 41 번, 정답 1 명 (정답률 2.439%)
- Div. 2 처음 푼 사람: **Sapple**, 209분
- 출제자: `mujigae`

- 마력을 A 소모하는 행동을 1번 연산, 마력을 B 소모하는 행동을 2번 연산이라고 하겠습니다.
- 관찰 1: 2번 연산을 [(1번 연산) $+$ (전체 비트를 뒤집는 연산)]으로 분리할 수 있습니다.
- 관찰 2: 2번 연산을 짝수 번 진행한 결과는 0번 진행한 결과와 같고, 홀수 번 진행한 결과는 1번 진행한 결과와 같습니다.
- 관찰 3: 정수열의 i번째 항을 i번 정점으로, 마법진 $[L, R]$을 L번 정점과 $R+1$번 정점을 잇는 간선으로 보고 $R=N$인 경우를 위해 $N+1$번 정점까지 추가하면, 주어진 문제를 그래프에서 해결할 수 있습니다.

- 모든 항을 1로 만드는 경우를 크게 두 가지로 나눌 수 있습니다.
- 1. 1번 정점부터 $N+1$번 정점으로 가는 단순 경로
  - 2번 연산을 반드시 짝수 번 사용해야 합니다.
- 2. 단순 회로
  - 2번 연산을 반드시 홀수 번 사용해야 합니다.
  - 마력의 최솟값을 구해야 하므로 회로 중에 단순 회로만 고려해도 됩니다.
- 1번 연산과 2번 연산의 마력 소모량이 다르고, 경로의 홀짝성이 중요하기 때문에 BFS로 각 정점에서 출발하는 최단 경로와 최단 회로들을 찾으며 답을 갱신해주면 최솟값을 찾을 수 있습니다.
- 시간복잡도는 $\mathcal{O}(N(N+M))$입니다. $\mathcal{O}(M(N+M))$ 풀이도 가능합니다.

1F. 스시스시 왕국

`greedy`

출제진 의도 – **Hard**

- Div. 1 제출 20번, 정답 4명 (정답률 20.000%)
- Div. 1 처음 푼 사람: **SNUPS 첩자**, 172분
- 출제자: `ksi4495`, `sungjae0506`

- 먼저 한 도시에 도로가 추가될 때 어느 마을에 연결해야 하는지부터 고려해봅시다.
- 각 도시에 해당하는 트리의 센트로이드를 해당 도시의 대표 마을이라고 하겠습니다.
- 서로 다른 도시를 연결하는 도로를 추가할 때, 각 도시의 대표 마을에 도로를 연결하는 것이 최적입니다.
- 이제 모든 마을 사이의 거리의 합의 최솟값을 고려해봅시다.
- 같은 도시에 속한 두 마을 사이의 거리는 상수이므로 고려하지 않아도 됩니다.
- 서로 다른 도시 $X,\ Y$ 에 속한 두 마을 $A,\ B$ 사이의 거리는 $A \to C_X \to C_Y \to B$ 로 분해할 수 있습니다.
- 여기에서 $C_X,\ C_Y$ 는 각각 도시 $X,\ Y$ 의 대표 마을을 나타냅니다.

- $A \to C_X$ 와 $C_Y \to B$ 는 모두 상수이므로 고려하지 않아도 됩니다.
- 결국 모든 마을 사이의 경로에 $C_X \to C_Y$ 가 몇번 포함되는지가 최솟값에 영향을 줍니다.
- 이때 $C_X \to C_Y$ 가 경로에 포함되는 횟수는 도시 $X,\ Y$ 에 마을이 몇개 있는지에 의해서 결정됩니다.
- 따라서 각 도시에 마을의 개수만큼 가중치가 부여된 형태로 문제를 변형할 수 있습니다.
- 가중치가 부여된 정점과 각 정점의 차수가 주어질 때, 가중 합의 최솟값을 구하면 됩니다.
- 가중치가 클수록 차수가 단조증가하므로, 리프에 가중치가 작은 정점을 그리디하게 연결하는 방식이 항상 최적임을 증명할 수 있습니다.
- 우선순위 큐를 사용하여 $\mathcal{O}(N + M \log M)$ 의 시간복잡도로 문제를 해결할 수 있습니다.

- 풀이1: Generalized Huffman Tree (GHT)를 구현하면 됩니다.
- GHT는 (가중치, 차수) 쌍이 가장 작은 non-leaf 정점을 선택하여, 가중치가 작은 (차수 - 1)개의 leaf 정점을 연결하고 병합하는 과정을 반복합니다.
- 이때 병합된 정점은 가중치가 모두 더해져서 새로운 leaf 정점이 됩니다.
- 증명) Goubko, Mikhail. “Minimizing Wiener index for vertex-weighted trees with given weight and degree sequences.” *MATCH Communications in Mathematical and in Computer Chemistry*, vol. 75, 2016, pp. 3–27.
- cf. Prüfer sequence

- 풀이2: 풀이 1에서 정렬 기준을 (차수, 가중치)로 바꾸어도 동일한 결과를 얻을 수 있습니다.
- 증명) 문제의 조건으로 (가중치, 차수) 기준으로 정렬한 순서와 (차수, 가중치) 기준으로 정렬한 순서가 동일하므로, 병합 과정에서 선택되는 non-leaf 정점의 종류와 순서는 동일합니다.
- 초기 leaf 정점의 차수는 non-leaf 정점의 차수 이하이므로, 병합을 통해 새롭게 생성된 leaf 정점은 초기 leaf 정점이 모두 사용된 이후에야 선택됩니다.
- 병합 과정에서 leaf가 된 non-leaf 정점은 이후에도 동일한 정렬 기준에 따라 순서가 유지됩니다.
- 이는 차수와 leaf 정점의 가중치가 모두 단조 증가하기 때문입니다.

- 풀이3: (가중치, 차수) 쌍이 가장 큰 정점에 (가중치, 차수)가 그 다음으로 큰 정점들을 그리디하게 연결하는 것도 최적입니다.
- 증명) 풀이 2에서 non-leaf 정점에 연결되는 정점들을 역으로 고려하면 동치인 것을 알 수 있습니다.
- 풀이4: BFS 순서대로 넣으면 풀이 3과 동일한 풀이를 얻을 수 있습니다.
- 증명) 적당히 서브그래프를 바꿔서 더 작은 값을 얻을 수 있습니다.

1E. 격자 경로의 가중치

`math, fft, divide_and_conquer`

출제진 의도 – **Hard**

- Div. 1 제출 33번, 정답 15명 (정답률 45.455%)
- Div. 1 처음 푼 사람: **ainta**, 59분
- 출제자: `ohwphil`

- t에 대한 답을 b_t라고 정의하고, b_t에 대한 점화식을 찾아봅시다.
- 우선 $b_0 = a_0$입니다.
- 이제 (t, t)에 도착하는 한 경로가 (t, t) 이전에 방문한 직선 $y = x$ 상의 마지막 점을 (u, u)라고 합시다.
- (u, u)를 방문한 직후 $(u + K, u + K - 1)$을 방문하게 되고, 그 이후 (t, t)까지 $y = x$ 상의 점을 하나도 방문하지 않고 도착해야 합니다. 따라서 $t - u \geq K$여야 함을 알 수 있습니다.

- 이제 $(u+K, u+K-1)$ 에서 출발하여 (t,t) 까지 $y=x$ 상의 점을 하나도 방문하지 않고 도착하는 경우의 수를 생각해 봅시다.
- (t,t) 를 방문하기 직전에 $(t,t-1)$ 을 방문해야 하기 때문에 구하는 경우의 수는 $(u+K, u+K-1)$ 에서 출발하여 직선 $y=x-1$ 또는 그것보다 아래에 있는 점만을 지나며 $(t,t-1)$ 에 도착하는 경우의 수와 같게 됩니다.
- 이는 $u-t-K$ 번째 카탈란 수와 같게 됩니다.

- n 번째 카탈란 수를 $C_n := \dfrac{(2n)!}{n!(n+1)!}$ 으로 표기하겠습니다.
- 앞의 논의를 정리하면, $b_i = a_i \displaystyle\sum_{j=0}^{i-K} b_j C_{i-j-K}$ 임을 알 수 있습니다.
- 그러나, 이 점화식을 naïve하게 구현하면 시간복잡도 $\mathcal{O}(N^2)$ 로 시간 초과를 받게 됩니다.

- 어떻게 하면 이 풀이를 최적화할 수 있을까요? 일단 이 알고리즘의 유사코드를 적어봅시다.

```
b[0] <- 1
function naive(l, r, N)
  for i in range(l, r)
    b[i] <- b[i] * a[i]
    for j in range(i + 1, r)
      b[j] <- b[j] + b[i] * C[i - j - K]
```

- 조금 당황스러울 수 있지만, naive(l, r, N) 의 수행 결과는 아래의 dnc_slow(l, r, N) 의 수행 결과와 동일합니다.

```
b[0] <- 1
function dnc_slow(l, r, N)
  if r - l = 1
    b[l] <- b[l] * a[l]
    return
  m <- (l + r) / 2
  dnc_slow(l, m, N)
  for i in range(l, m)
    for j in range(m, r)
      b[j] <- b[j] + b[i] * C[i - j - K]
  dnc_slow(m, r, N)
```

- 왜 그런지 조금 더 자세히 알아봅시다.
- 우선 $r - l = 1$ 일 때는 자명합니다.
- 이제 재귀적인 부분을 생각해 보면, dnc_slow(l, m, N)과 naive(l, m, N)의 동작은 귀납 가정에 의해 동일합니다.
- 그리고 dnc_slow의 이중 for문에 의해 $l \leq i < m$ 인 모든 i에 대해서 naive(l, r, N)에서 일어나야 할 전이가 모두 일어나게 됩니다.
- 즉 naive(l, r, N)의 이중 for문 중 바깥 for문이 $[l, m)$ 에 대해서 수행된 것과 같은 효과가 발생합니다.
- 마지막으로 dnc_slow(m, r, N)을 호출하면 $[m, r)$ 에 대해서도 바깥 for문이 수행된 것과 같은 효과가 나기 때문에 두 함수의 실행 결과는 같습니다.

- 그렇다면 dnc_slow(l, r, N)의 의미는 무엇일까요?
- 구간을 반으로 나눈 뒤 왼쪽 구간에 재귀호출을 하여 왼쪽 구간의 b_i 값을 확정지은 후, 왼쪽 구간이 오른쪽 구간에 미치는 영향을 전이시킨 후, 오른쪽 구간에 다시 재귀호출을 하는 것으로 생각할 수 있습니다.

- 물론 dnc_slow(0, N, N)의 시간복잡도도 $\mathcal{O}(N^2)$로 아직은 역부족이지만, 자세히 보면 알고리즘 내부의 이중 for문 구조는 컨볼루션 형태로 나타낼 수 있기 때문에 FFT 또는 NTT를 통해 $\mathcal{O}(N \log N)$의 시간복잡도로 수행할 수 있습니다.
- 따라서 알고리즘의 시간복잡도는 $T(N) = 2T\left(\frac{N}{2}\right) + \mathcal{O}(N \log N)$이므로 Master theorem을 적용하면 시간복잡도가 $\mathcal{O}(N \log^2 N)$임을 알 수 있습니다. 이는 문제를 해결하기에 충분합니다.
- 이와 같이 FFT와 분할정복을 이용하여 DP 전이를 빠르게 하는 테크닉을 CDQ convolution 또는 online FFT라고 부릅니다.

1H. 관악산 정상에는 구름이 없다

`geometry`, `half_plane_intersection`
출제진 의도 – **Hard**

- Div. 1 제출 11번, 정답 0명 (정답률 0.000%)
- Div. 1 처음 푼 사람: **-**, -분
- 출제자: `sungjae0506`

- 관악산 옆면의 특징을 알아봅시다.
- 옆면 A, B, C 가 있을 때 A, B 가 인접하고 B, C 가 인접한 면이라 합시다.
- A, B, C 의 교점이 존재한다면 그 점은 B 가 최대 높이를 가지는 지점입니다.
- 밑면부터 시작하여 위로 올라오는 스위핑을 한다면 교점을 지날때마다 옆면은 1개 줄어듭니다.
- 따라서 교점은 중복을 포함하면 $N-2$ 개 있음을 알 수 있습니다.
- 이러한 사실을 이용하는 알고리즘을 생각해봅시다.

- 각 면을 인접한 순서로 링크드 리스트에 저장합니다.
- 링크드 리스트에서 연속된 3 개의 노드를 보면 교점의 높이를 구할 수 있습니다.
- 우선순위 큐로 교점 높이가 낮은 순으로 탐색하며, 최대 높이에 도달한 면을 링크드 리스트에서 삭제하고 우선순위 큐를 갱신하는 것을 반복합니다.
- 위 과정에서 얻은 교점을 통해 각 옆면을 밑면에 사영시킨 도형을 알아낼 수 있습니다.
- 사영시킨 면에서 구름과 닿는 면적을 반평면 교집합을 통해 계산하고 삼각함수를 이용해서 원래 면적을 구할 수 있습니다.

- 링크드 리스트에서 면을 삭제하고 우선순위 큐를 갱신하는데 $\mathcal{O}(\log N)$ 이 걸리고 이를 $N-2$ 번 반복하면 $\mathcal{O}(N \log N)$ 이 걸립니다.
- 반평면 교집합에도 $\mathcal{O}(N \log N)$ 이 걸리므로 전체 시간복잡도도 $\mathcal{O}(N \log N)$ 입니다.
- 구름과 닿는 면적을 구하는 과정에서 반평면 교집합을 사용하지 않는 구현도 가능합니다.
- 이 문제는 Weighted Straight Skeleton 을 구하는 문제와 관련이 있습니다.

1C. 최댓값과 쿼리

`data_structures, segtree, lazyprop, offline_queries`

출제진 의도 – **Challenging**

- Div. 1 제출 2번, 정답 0명 (정답률 0.000%)
- Div. 1 처음 푼 사람: **-**, -분
- 출제자: `sungjae0506`

- $A_{1,1}$ 이 최댓값이 되도록 cyclic shift 합니다.
- 최댓값이 전파되는 모양을 보면 아래와 같이 평행사변형 모양입니다.

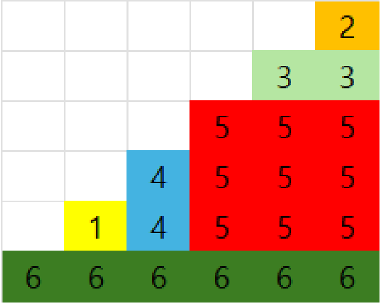
6	1	4	5	3	2
6	6	4	5	5	3
6	6	6	5	5	5
6	6	6	6	5	5
6	6	6	6	6	5
6	6	6	6	6	6

- 노드 하나가 평행사변형 하나의 정보를 저장하는 레이지 세그먼트 트리를 만들어봅시다.
- 각 노드에는 누적합, 누적합에 더할 값, 누적합에 더할 값에 더할 값, 영역의 길이, 영역의 길이에 더할 값을 저장합니다.
- 한 행마다 전체 업데이트 연산을 하고, 평행사변형의 꼭짓점 마다 각 노드의 값을 수정합니다.

- 쿼리를 수행하기 위해 행 순서로 정렬합니다.
- 구해야하는 직사각형 영역을 아래와 같이 3개의 영역으로 분할합니다.

- 세그 워크를 이용해 빨간색 영역을 구할 수 있습니다.
- 초록색 영역은 간단한 계산으로 구할 수 있습니다.
- 파란색 영역은 새로운 레이지 세그먼트 트리를 도입하여 구해줍니다.

- 새로운 레이지 세그먼트 트리는 아래 그림처럼 행렬을 45도 기울인 값을 저장합니다.



- 삼각형 모양의 영역을 이 세그먼트 트리에서 모두 구해서 결과값에 더합니다.
- 전체 시간 복잡도는 $\mathcal{O}((N+Q)\log N + Q\log Q)$ 이다.

EX1. 콩돌 놀이

`ad_hoc, math`

출제진 의도 – **Easy**

- Open Contest 제출 27번, 정답 15명 (정답률 55.556%)
- Open Contest 처음 푼 사람: **math_rabbit_1028**, 36분
- 출제자: `mujigae`

- 문제를 처음 보면 백트래킹 풀이를 시도하고 싶습니다.
- 제한이 작기 때문에 백트래킹을 통해서도 문제를 해결할 수 있습니다. 대신 구현이 많이 번거롭다는 단점이 있습니다.
- 출제자는 여러분이 구현으로 고통 받기를 원하지 않았습니다.
- 다른 방법은 없을까요?

- **S** 모양과 **C** 모양을 만드는 데 필요한 검은색 콩돌의 개수에 숨겨진 해답이 있습니다. 두 모양은 각각 11 개와 9 개의 검은색 콩돌로 이루어져 있습니다.
- 입력으로 주어지는 격자에는 최대 100 개의 검은색 콩돌이 들어갈 수 있습니다.
- **S** 모양 9 개를 채우거나 **C** 모양 11 개를 채우는 것이 불가능합니다.
- 단순히 검은색 콩돌의 개수를 세어 $11s + 9c = T$ 형태의 디오판토스 방정식을 해결하면 됩니다.

EX2. 모모카와 열차 운행표

`implementation, simulation, binary_search, priority_queue, sorting`

출제진 의도 – **Medium**

- Open Contest 제출 4번, 정답 1명 (정답률 25.000%)
- Open Contest 처음 푼 사람: **jeoffrey0522**, 172분
- 출제자: `pangitwise`

- 주어진 기차역 그래프 G에서 열차 운행표를 시뮬레이션하면서 오류를 검사해야 합니다.
- 먼저 그래프 G를 인접 리스트의 형태로 저장합니다.
- 그 다음 열차 운행표에서 각 열차들의 운행 경로를 받고, 해당 기차가 언제 몇 번 역에 도착하게 되는지를 저장합니다. 이 정보들의 집합을 K라고 해 봅시다.
- 이때 간선이 존재하지 않아 열차가 더이상 나아가지 못하는 경우가 있을 수 있습니다. 이를 탐지하기 위해 매번 현재 역에서 다음 역으로 가는 간선이 존재하는지를 검사해야 합니다.
- 이를 naive하게 검사하면 최악의 경우 시행마다 $\mathcal{O}(N)$ 만큼의 시간이 걸리게 됩니다.
- 이는 이분 탐색을 이용하여 시행마다 $\mathcal{O}(\log N)$ 의 시간으로 줄일 수 있습니다.
- 이분 탐색을 활용할 수 있도록 미리 인접 리스트 내의 간선 정보가 정렬되어 있어야 합니다.

- 기차가 마주하는 문제들을 정리한 집합을 E라고 해 봅시다.
- 앞선 과정에서 열차가 존재하지 않는 간선을 따라 이동해야 하는 경우를 따로 K에 저장해 둡시다.
- 이제 집합 K를 시간순으로 정렬하고, 오름차순으로 정보들을 하나씩 보면서 현재 기차들과 역들의 상태, 기차들의 이동을 시뮬레이션하면서 집합 E를 갱신해나가면 됩니다.
- 가능한 시각의 범위가 최대 $10^{10} + 1$이므로 모든 시각을 일일이 확인할 수는 없습니다.
- 따라서 같은 시각에 일어나는 정보들끼리 따로 모아 한꺼번에 처리해야 합니다

EX2. 모모카와 열차 운행표

- 시각 l에 처리해야 할 정보들의 순서는 다음과 같습니다.
  1. 시각 $l-1$에 간선이 없어 나아가지 못한 열차를 무기한 정차 처리하고 집합 E를 갱신한다. 이 정보는 유효해야 한다.
  2. 시각 l에 다음 역에 도달하게 되는 이동 가능한 열차들을 움직인다.
  3. 열차가 다음 역을 중복으로 방문하게 되는 경우 집합 E를 갱신한다.
  4. 시각 l에 같은 역에 동시에 도착하게 되는 열차들이 있거나, 이미 다음 역에 무기한 정차 중인 다른 열차가 있다면 충돌 판정 후 무기한 정차시키고 집합 E를 갱신한다.
- 시뮬레이션 과정에서 빠트린 부분이나 실수가 없도록 섬세한 구현이 필요합니다.
- 마지막으로 유효한 노선만을 운행하였을 때 모든 역의 최소 통과 요구 횟수를 만족시킬 수 있는지를 반복문으로 검사하면 됩니다.

- 인접 리스트에 저장된 간선을 정렬하는 데 최대 $\mathcal{O}(M \log N)$ 만큼의 시간이 걸립니다.
- 열차 운행표를 바탕으로 이분 탐색을 이용해 집합 K 를 구성하는 데 최대 $\mathcal{O}\left(\sum k_i \cdot \log N\right)$ 만큼의 시간이 걸립니다.
- 집합 K 를 우선순위 큐나 정렬을 이용해 시간순으로 시뮬레이션을 돌리는 데 최대 $\mathcal{O}\left(\sum k_i \cdot \log \sum k_i\right)$ 만큼의 시간이 걸립니다.
- 유효한 열차만을 운행하였을 때 모든 역의 최소 통과 요구 횟수를 만족하는지를 검사하는 데 최대 $\mathcal{O}\left(\sum k_i\right)$ 만큼의 시간이 걸립니다.
- 최종적으로 전체 $\mathcal{O}\left((M + \sum k_i) \cdot \log N + \sum k_i \cdot \log \sum k_i\right)$ 의 시간복잡도로 문제를 해결할 수 있습니다.