UCPC 2022

전국 대학생 프로그래밍 대회 동아리 연합
여름 대회 2022

Preliminaries

Official Solutions

예선 해설

전국 대학생 프로그래밍 대회 동아리 연합 · UCPC 2022 출제진

문제		의도한 난이도	출제자
A	코딩은 체육과목 입니다	**Easy**	ho94949
B	N수매화검법	**Medium**	man_of_learning
C	수소철도 충전 시스템	**Challenging**	functionx
D	functionx	**Hard**	heeda0528
E	solved.ac 2022	**Medium**	shiftpsh
F	Twitch Plays VIIIbit Explorer	**Medium**	ho94949
G	SCV 체인	**Hard**	doju
H	yo, i herd u liek ternary operators, so..	**Hard**	exqt
I	편지 배달	**Challenging**	queued_q
J	수 정렬하기, 근데 이제 제곱수를 곁들인	**Hard**	hyperbolic

A. 코딩은 체육과목입니다

`implementation`

출제진 의도 – **Easy**

- 제출 320번, 정답 293팀 (정답률 91.56%)
- 처음 푼 팀: **일감호는우리가지킨다** (건덕이, 건구스, 만쥬), 1분
- 출제자: ho94949
- 가장 짧은 정해: $41B^{Python}$, $145B^{C++}$

A. 코딩은 체육과목 입니다

- 혜아가 생각하는 N바이트 자료형에는 long이 $N/4$개 들어감을 알 수 있습니다.
- "long "을 $N/4$번 출력한 이후에 "int"를 출력하면 됩니다.

B. N수매화검법

`geometry`, `greedy`
출제진 의도 – **Medium**

- 제출 662번, 정답 144팀 (정답률 45.00%)
- 처음 푼 팀: **한양대학교3번째팀멤버를 찾습니다** (팀원1, 팀원2, 외부인), 7분
- 출제자: `man_of_learning`
- 가장 짧은 정해: 1,004B$^{\text{C++}}$, 1,006B$^{\text{Python}}$

- 베기 i의 경로는 두 점 $(sx_i, sy_i), (ex_i, ey_i)$로 정의되는 선분입니다.
- 따라서 **두 벡터의 외적**을 활용하면 두 베기 i, j의 경로의 교차 여부를 판단할 수 있습니다.
  - 벡터의 외적을 활용하면 임의의 세 점 p_1, p_2, p_3이 주어질 때 벡터 $\overrightarrow{p_1p_2}$에 대해 벡터 $\overrightarrow{p_1p_3}$의 상대적인 방향성을 알 수 있습니다.
  - 이때 $\overrightarrow{p_1p_3}$가 $\overrightarrow{p_1p_2}$의 시계 방향에 있다면 CW, 반시계 방향에 있다면 CCW라고 합니다.
  - 벡터 $\overrightarrow{s_ie_i}$에 대해 두 점 s_j, e_j 중 한 점이 CW, 다른 한 점이 CCW이고, 벡터 $\overrightarrow{s_je_j}$에 대해 두 점 s_i, e_i 중 한 점이 CW, 다른 한 점이 CCW이면 두 베기의 경로는 교차합니다.

- 베기를 행하는 순서가 정해지면 소모하는 내공의 양도 정해집니다.
- 가능한 순서가 $N!$개이기 때문에 모든 순서를 살펴볼 수는 없습니다. 그리디하게 최적의 순서를 결정할 수 있습니다.

- 편의를 위해 베기의 순서를 뒤집어 봅시다. m을 베기 i를 행한 순간에 아직 행하지 않은 (미래의) 베기가 아닌, 이미 행한 (과거의) 베기 중 베기 i와 경로가 교차하는 베기의 개수로 생각합시다.
- $(m+1) \times w_i$에서 $1 \times w_i$의 합은 베기의 순서가 변해도 달라지지 않습니다. 따라서 $m \times w_i$의 합을 최소화해야 합니다.
- 베기를 순서대로 하나씩 추가한다고 생각하면, 베기 i를 행할 때 생기는 각 교점마다 w_i를 더하는 문제가 됩니다.

- 각 교점의 관점에서 생각해 봅시다. 어떤 교점에서 만나는 베기를 순서대로 i, j 라고 하면, 나중에 추가되는 베기 j 를 행할 때 이 교점에서 가중치 w_j 가 더해집니다. 즉 가중치가 작은 베기를 나중에 수행하면 교점에서 더해지는 가중치가 작아집니다.
- 따라서 w_i 를 기준으로 내림차순 정렬한 뒤 검법을 펼치면, 모든 교점마다 최소의 가중치가 더해진다는 것을 알 수 있습니다.

- 다시 베기의 순서를 뒤집어 원래의 문제로 돌아오면 w_i **를 기준으로 오름차순 정렬한 뒤 검법을 펼치는 것이 최적**입니다.
- 직접 검법을 시뮬레이션하면서 내공을 계산할 필요는 없습니다. 먼저 w_i 의 총합을 구한 다음, 모든 교점을 보면서 두 베기의 가중치의 최솟값을 더해 주는 방식으로 쉽게 구현할 수 있습니다.

C. 수소철도 충전 시스템

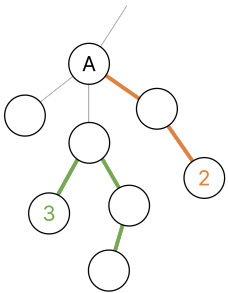
greedy, dp_tree

출제진 의도 – **Challenging**

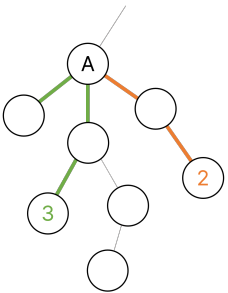
- 제출 61번, 정답 16팀 (정답률 41.03%)
- 처음 푼 팀: **초비상!!** (코딩 말렸어요, 시간초과 떴어요, Ofast가 해줘야 해요), 56분
- 출제자: functionx
- 가장 짧은 정해: 1,736B^{C++}, 2,218B^{Python}

C. 수소철도 충전 시스템

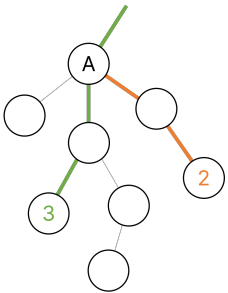
A의 부모 방향으로 무한한 길이의 레일이 있다고 가정할 때, 노드 A의 서브트리 위에 있는 기차를 배치해봅시다. 가능한 여러 배치 중에서도 우열이 있습니다.



1순위



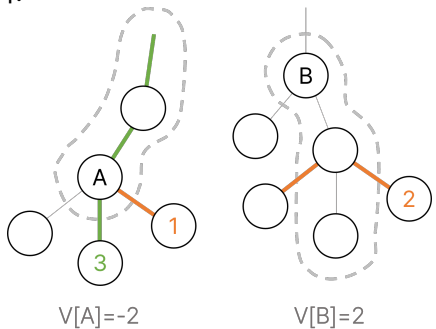
2순위



3순위

배치의 점수를 $V[A]$라 하면 $V[A]$를 최대화시켜야 합니다.

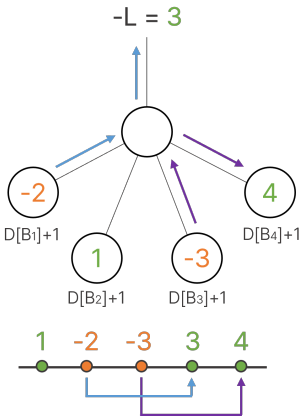
- 특정 기차가 A와 A의 부모를 잇는 간선을 지난다면, $V[A]$는 그 기차의 끝점(A의 조상 쪽 노드)과 A 사이의 거리를 -1로 곱한 값입니다.
- 이외의 경우에는 $V[A]$는 추가로 배치할 수 있는 A와 A의 자손을 잇는 기차의 최대 길이입니다 (없으면 0).



- $V[A]$의 최댓값 $D[A]$를 DP로 구해봅시다.
  - A의 자식 B_i의 서브트리에서 열차를 최적으로 배치합니다.
  - B_i의 서브트리에서 B_i와 A를 잇는 간선을 추가한다면, V 값은 1 늘어납니다.
  - 따라서 우리는 $D[B_i]+1$들을 가지고 A에서 최적의 배치를 찾으면 됩니다.

$D[A]$에 대한 Parametric Search를 하는데 $D[A] \geq L$인지 테스트합니다.

- $D[B_i]+1$들과 $-L$이 주어졌을 때 매칭을 할 수 있는지 구하면 됩니다.
- 모든 음수들이 절댓값이 크거나 같은 양수와 매칭을 할 수 있어야 됩니다.
- Greedy 방법으로 모든 음수들을 절댓값이 큰 순서대로 양수와 매칭하면 됩니다.



C. 수소철도 충전 시스템

- 정답이 YES이려면 아래 조건을 만족해야 합니다.
  - 모든 노드에 대하여 $D[i]$를 구할 수 있어야 한다.
  - 루트 노드 R에 대하여 $D[R] \le 0$이어야 한다.
- $D[A]$를 구하는 시간은 $\mathcal{O}\left(\deg_A \log(\deg_A)\right)$ 입니다.
- 정답을 구하는 시간은 $\mathcal{O}\left(N \log(N)\right)$ 입니다.
- 역추적 구현이 꽤 까다로운 편입니다.

D. functionx

`data_structures`

출제진 의도 – **Hard**

- 제출 1,098번, 정답 49팀 (정답률 23.56%)
- 처음 푼 팀: **저희 팀 이름 머함** (진짜 이걸로 해요?, ㅇㅇ, 팀원 이름 추천 받음), 21분
- 출제자: heeda0528
- 가장 짧은 정해: $907B^{C++}$, $1{,}424B^{Python}$

D. functionx

- 우선 $\mathrm{sgn}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$ 로 정의합시다.
- 각 2번 쿼리에 대해 그 전까지 들어온 1번 쿼리에 대한 $\mathrm{sgn}\left(\prod_i (a_i c + b_i)\right)$ 를 알고 싶습니다.
- $\mathrm{sgn}\left(\prod_i (a_i\, c + b_i)\right) = \prod_i \mathrm{sgn}\,(a_i\, c + b_i) = k \prod_{a_i \neq 0} \mathrm{sgn}\left(c + \frac{b_i}{a_i}\right)$
- $k = \prod_{a_i \neq 0} \mathrm{sgn}\,(a_i) \prod_{a_i = 0} \mathrm{sgn}\,(b_i)$

k를 관리하는 방법은 어렵지 않습니다.

- 초기값은 $k = 1$입니다.
- 1번 쿼리가 들어왔을 때 $a \neq 0$이면 $\mathrm{sgn}(a)$, 아니면 $\mathrm{sgn}(b)$를 k에 곱합니다.

D. functionx

$\prod_{a_i \neq 0} \operatorname{sgn}\left(c + \frac{b_i}{a_i}\right)$ 을 구해봅시다.

$$\operatorname{sgn}\left(c + \frac{b_i}{a_i}\right) = \begin{cases} -1 & \text{if } -c > \dfrac{b_i}{a_i} \\ 0 & \text{if } -c = \dfrac{b_i}{a_i} \\ 1 & \text{if } -c < \dfrac{b_i}{a_i} \end{cases}$$

- $-c = \frac{b_i}{a_i}$ 를 만족하는 i 가 하나라도 있으면 0,
- 그렇지 않고 $-c > \frac{b_i}{a_i}$ 를 만족하는 i 의 개수가 홀수면 -1, 짝수면 1 입니다.

- $-c = \frac{b_i}{a_i}$ 를 만족하는 i 가 하나라도 있는지 여부는 $\frac{b_i}{a_i}$ 를 tree set 혹은 hash set으로 관리하면 $\mathcal{O}(\log Q)$ 에 구할 수 있습니다.
- c 는 정수이기 때문에 b_i 가 a_i 의 배수인 경우만 고려해도 충분합니다.

- $-c > \frac{b_i}{a_i}$ 를 만족하는 i 의 개수는 적절한 자료구조를 사용해 $\mathcal{O}\left(\log Q\right)$ 에 구할 수 있습니다.
  1. Segment/Fenwick Tree
  2. Ordered Set in Policy-Based Data Structures

1. Segment/Fenwick Tree

- $\frac{b_i}{a_i}$ 의 범위가 크기 때문에 좌표 압축을 먼저 해야 합니다.
- $-c$ 보다 작은 $\frac{b_i}{a_i}$ 의 개수를 구하는 것이 목적이므로 $\frac{b_i}{a_i}$ 를 $\frac{b_i}{a_i}$ 보다 작거나 같은 정수로 변환해 압축해도 상관 없습니다.
- 이제 점 업데이트, 구간 합 쿼리를 지원하는 Segment/Fenwick Tree를 이용하면 각 쿼리를 처리할 수 있습니다.
- 홀짝성만이 필요한 정보이므로 합 대신 xor을 지원하는 자료 구조를 사용해도 좋습니다.

2. Ordered Set in Policy-Based Data Structures

- `std::set`에서 `find_by_order()`와 `order_of_key()` 함수가 추가된 G++의 자료 구조입니다.
- 이 중 `order_of_key(x)`는 set에서 x가 들어갈 위치, 즉 x보다 작은 원소의 개수를 리턴합니다.
- 비교 함수를 `std::less_equal<Key>`로 설정하면 `std::multiset`처럼 쓸 수 있습니다.
- 1에서와 같이 $\frac{b_i}{a_i}$를 정수로 바꾸어 set에 삽입하고, `order_of_key()` 함수를 이용해 $-c$보다 작은 원소의 개수를 구합니다.
- 구현이 훨씬 짧아, 정해는 이 방법으로 작성되었습니다.

E. solved.ac 2022

`implementation`, `string`, `parsing`
출제진 의도 – **Medium**

- 제출 650번, 정답 155팀 (정답률 31.89%)
- 처음 푼 팀: **개노답 3형제** (애드혹, 구성적, 많은조건분기), 14분
- 출제자: `shiftpsh`
- 가장 짧은 정해: 534B$^{\text{Python}}$, 836B$^{\text{C++}}$

$$p_i = \max\left(0.5^{(t_N - t_i)/365\text{일}}, 0.9^{N-i}\right)$$

$$X = \frac{p_1 l_1 + p_2 l_2 + \cdots + p_N l_N}{p_1 + p_2 + \cdots + p_N}$$

이 두 식을 구현해야 합니다.

l_i 는 문제에서 주어지므로 p_i 를 계산해야 하는데, t_i 는 날짜/시각 형식으로 주어집니다.

우선 시각을 파싱해 봅시다.

$$
\begin{aligned}
&\text{10:12:31} \\
&= \text{자정} + 10\text{시간} + 12\text{분} + 31\text{초} \\
&= \text{자정} + 10 \times 60 \times 60\text{초} + 12 \times 60\text{초} + 31\text{초} \\
&= \text{자정} + 36\,751\text{초}
\end{aligned}
$$

다음으로 날짜를 파싱해 봅시다.

2021/10/14

$$= (2021\text{년 } 1\text{월 } 1\text{일}) + \sum_{i=1}^{9} (i\text{월을 구성하는 날 수}) + 13\text{일}$$

$$= (2021\text{년 } 1\text{월 } 1\text{일}) + (31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30)\text{일} + 13\text{일}$$

$$= (2021\text{년 } 1\text{월 } 1\text{일}) + 286\text{일}$$

합치면 이렇게 됩니다.

$$
\begin{aligned}
&\text{2021/10/14 10:12:31} \\
&= (\text{2021년 1월 1일}) + 286\text{일} + 36751\text{초} \\
&= (\text{2021년 1월 1일}) + 286\text{일} + \frac{36751}{24 \times 60 \times 60}\text{일} \\
&= (\text{2021년 1월 1일}) + 286.425359\text{일}
\end{aligned}
$$

E. solved.ac 2022

어떤 날짜 S가 있다고 해 봅시다. 주어진 공식에서 $t_N - t_i$는 이렇게 바꿀 수 있습니다.

$$\begin{aligned} & t_N - t_i \\ &= t_N - S - t_i + S \\ &= (t_N - S) - (t_i - S) \end{aligned}$$

S를 적당히 잘 정하면 시각을 다루기 쉬워질 것 같습니다. 2019년 이후의 기록들만 들어오므로 2019년 1월 1일로 정해 봅시다.

2021/10/14 10:12:31

$$= (\text{2021년 1월 1일}) + 286.425359\text{일}$$

$$= (\text{2019년 1월 1일}) + \sum_{i=2019}^{2020} (i\text{년을 구성하는 날 수}) + 286.425359\text{일}$$

$$= (\text{2019년 1월 1일}) + 365\text{일} + 366\text{일} + 286.425359\text{일}$$

$$= S + 1017.425359\text{일}$$

이제 p_i를 쉽게 계산할 수 있게 되었습니다.

직접 파싱할 경우 2020 년이 윤년이라는 사실에 주의합니다.
언어에 따라 다음과 같은 방법들로 파싱하는 것도 가능하며, 일반적으로 1970 년 1 월 1 일부터 몇 초가 지났는지를 계산해 줍니다.

- **C++11**
  - `std::cin >> std::get_time(std::tm* t, "%Y/%m/%d %H:%M:%S")`
  - 이후 `std::mktime(std::tm* t)`
- **Python**
  - `datetime.timestamp(datetime.strptime(t, "%Y/%m/%d %H:%M:%S"))`

F. Twitch Plays VIIIbit Explorer

`greedy`, `implementation`
출제진 의도 – **Medium**

- 제출 567번, 정답 158팀 (정답률 35.19%)
- 처음 푼 팀: **삼성전자 DX부문** (네트워크사업부 서씨, 삼성리서치 안씨, 생활가전사업부 정씨), 14분
- 출제자: ho94949
- 가장 짧은 정해: $777B^{\text{Python}}$, $888B^{\text{C++}}$

- 이동 횟수에 제한이 없다면 아이템은 원하는 순서로 주울 수 있습니다.
- 아이템의 위치나 이동횟수를 무시하고 최대 강화횟수인 C를 구해봅시다.

F. Twitch Plays VIIIbit Explorer

- 닉네임에 어떤 알파벳이 X 번 등장한다고 합시다. $(X > 0)$
- 이 닉네임을 C 번 쓰기 위해서는 해당 알파벳이 CX 개 필요합니다.
- CX 는 던전에 있는 해당 알파벳이 적힌 아이템 수보다 작거나 같아야 합니다.
- 즉, 던전에 해당 알파벳이 적힌 아이템이 Y 개 있으면 $C \le \frac{Y}{X}$ 여야 합니다.
- 모든 문자에 대해 위 조건을 만족하는 최대의 C 를 구하면 됩니다.
- C 는 각 알파벳에 대해 $\left\lfloor \frac{\text{던전에 해당 알파벳이 적힌 아이템의 개수}}{\text{닉네임에 해당 알파벳이 등장한 횟수}} \right\rfloor$ 의 최솟값입니다.

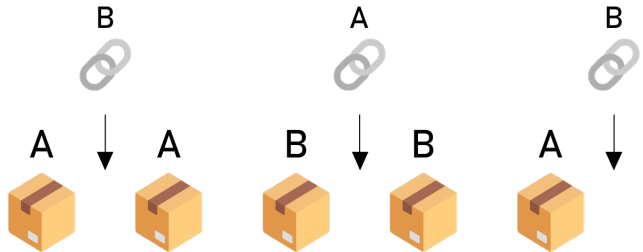
- 이제 닉네임을 C번 나열한 이후에, 아이템을 주울 순서를 임의로 정합니다.
  - 같은 아이템을 두 번 줍지만 않으면 어떤 방식으로 골라도 상관 없습니다.
- 이제 해당 아이템을 모두 줍는 경로를 찾아서 출력하면 됩니다.
- 현재 위치와 다음의 갈 위치의 행 번호 차이에따라 U, D를; 열 번호 차이에 따라 L, R를 출력합니다.
- 아이템이 있는 칸에 도착하면 P를 출력합니다.
- 문자열의 가능한 최대 길이는 $(N+M)NM$을 넘지 않으므로, 출력 조건에 부합합니다.

G. SCV 체인

ad_hoc, greedy
출제진 의도 – **Hard**

- 제출 281번, 정답 47팀 (정답률 29.38%)
- 처음 푼 팀: **UCPC의 최신 동향** (나폴리폴리, 고슬고슬비빈, 탕맛기픈), 31분
- 출제자: doju
- 가장 짧은 정해: 981B^{C++}, 1,164B^{Python}

- 주어진 블록 동작들로 놓인 블록들이 모두 올바른 블록체인(증가수열)이 되도록, 남은 블록들을 남김없이 체인 동작으로 이어 붙이는 문제입니다.
- 두 로봇이 놀이를 하고 있기 때문에 홀짝성을 따져야 합니다.



- 한 로봇이 연속으로 블록 동작을 하기 위해서는 그 사이에 반드시 상대방의 체인 동작이 있어야 합니다.
- 또한 마지막 블록 동작을 A 로봇이 한 경우에는 반드시 B 로봇의 체인 동작이 뒤따라야 합니다.

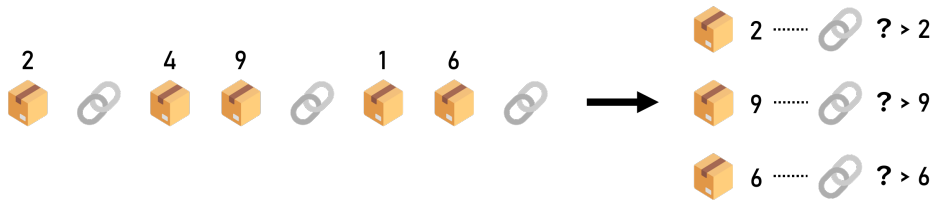
G. SCV 체인

- 이렇게 꼭 필요한 체인 동작들을 넣으면 이 기록은 일단 두 로봇이 번갈아가며 동작한 기록이 됩니다.
- 이 기록의 임의의 위치에 체인 동작을 짝수 개 묶어서 끼워 넣어도 기존의 동작들에 영향을 주지 않습니다.
  - 물론 놀이를 체인 동작으로 시작할 수 없으므로, 기록의 맨 앞에 넣으면 안 됩니다.

따라서 이 문제의 풀이는

- 체인 동작이 필요한 지점들에 체인 동작을 추가하고
- 남은 블록들을 짝수 개씩 묶음지어서 적절히 끼워 넣는

두 부분으로 나눌 수 있습니다.



- 체인 동작을 추가할 때, 이 동작으로 놓이는 블록의 번호는 바로 앞의 블록 동작으로 놓인 블록의 번호보다 커야 합니다.
- 이는 블록 동작으로 놓인 블록에 대해 그 블록보다 번호가 큰 블록을 하나 짝지어 주는 것으로 볼 수 있습니다.

만약 올바른 매칭이 존재한다면, 다양한 방법으로 매칭 하나를 찾을 수 있습니다.

- 짝이 필요한 블록들을 임의의 순서로 보면서, 남아 있는 자신보다 번호가 큰 블록들 중 가장 작은 번호의 블록을 매칭합니다.
- 짝이 필요한 블록들을 가장 큰 번호의 블록부터 보면서, 남아 있는 자신보다 번호가 큰 블록들 중에서 임의로 하나를 고릅니다.
- 짝이 필요한 블록들과 남은 블록들을 모두 번호순으로 늘어 놓고, 괄호를 짝지어 주듯 스택을 사용해 매칭합니다.

⋮

이때 N이 크므로 $\mathcal{O}(N)$ 또는 $\mathcal{O}(N \log N)$의 시간복잡도를 갖도록 구현해야 합니다.

이제 매칭을 하고 남은 블록들을 끼워 넣는 방법을 생각해 봅시다.

이제 매칭을 하고 남은 블록들을 끼워 넣는 방법을 생각해 봅시다.

놀이 규칙에서 알 수 있는 중요한 사실은, 1번 블록은 반드시 블록 동작으로 주어져야 한다는 것입니다.

- 이는 2번 예제를 통해 암시되기도 했습니다.

남는 블록은 짝수 개이고 모두 1번 블록보다 번호가 크므로, 남은 블록을 모두 1번 블록의 뒤에 번호 순서대로 이어 붙여 주면 문제를 해결할 수 있습니다.

다만, 1번 블록에 이미 매칭된 블록이 있는 경우는 주의가 필요할 수 있습니다.

이 경우 결국 답에서는 1번 블록과 처음 매칭된 블록이 어떤 것인지 구분할 수 없으므로, 매칭을 끊은 뒤 다른 남은 블록들과 한꺼번에 추가해 주면 쉽게 처리할 수 있습니다.

- 몇몇 그리디한 매칭 방법을 사용하면 자연스럽게 1번 블록과 매칭된 블록보다 번호가 큰(또는 작은) 블록들만 남게 되어, 매칭을 유지하면서도 간단히 처리할 수 있습니다.
- 또는 아예 매칭을 찾을 때부터 1번 블록을 제외하는 방법도 있습니다.

H. yo, i herd u liek ternary operators, so..

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

- 제출 293번, 정답 33팀 (정답률 35.11%)
- 처음 푼 팀: **삼성전자 DX부문** (네트워크사업부 서씨, 삼성리서치 안씨, 생활가전사업부 정씨), 12분
- 출제자: `exqt`
- 가장 짧은 정해: 855B$^{\text{C++}}$, 887B$^{\text{Python}}$

H. yo, i herd u liek ternary operators, so..

우선 다음과 같은 관찰이 필요합니다.

- 변수는 데코레이션일 뿐 필요가 없습니다. 답에 영향을 주는 것은 `?:` 의 배치뿐입니다.
  - 편의를 위해 식을 표기할 때 변수를 제외하고 표기하겠습니다. (`a?b?c:d:e` → `??::`)
- 모든 `?:` 는 유일하게 짝을 지어줄 수 있습니다. 괄호 문자열을 생각해 봅시다.

실제로 식을 파싱한다고 생각해 봅시다.

- 식에서 적절한 `?:` 쌍을 선택해서 `A?B:C`로 나눌 수 있습니다.
- 즉, 부분문제로 쪼개서 풀 수 있습니다. Python으로 구현하면 아래와 같은 느낌이 됩니다.

```
def f(S):
    if len(S) == 1: return 1 # just a variable
    return sum([f(A) * f(B) * f(C) for A,B,C in validSplits(S)])
```

하지만 이것을 그대로 쓰면 시간 안에 통과하기 힘들어 보입니다.

그래서 다음과 같은 관찰이 추가적으로 필요합니다. 만약 유효한 식이 2개 붙어 있다면 어떻게 될까요?

- 예를 들면 `?A:?B:`는 `(?A:)?B:`와 `?A:(?B:)`의 2가지 경우가 존재합니다.
  - 이 경우 답은 $2 \times f(\mathrm{A}) \times f(\mathrm{B})$가 됩니다.

만약 유효한 식이 3개 붙어 있다면 `(?A:)?B:?C:`, `?A:(?B:)?C:`, `?A:?B:(?C:)`가 가능할 것입니다. 각각의 경우의 수는

$$\texttt{(?A:)?B:?C:} \Rightarrow f(\mathrm{A}) \times f(\texttt{?B:?C:}) = f(\mathrm{A}) \times 2f(\mathrm{B})\, f(\mathrm{C})$$

$$\texttt{?A:(?B:)?C:} \Rightarrow f(\mathrm{A}) \times f(\mathrm{B}) \times f(\mathrm{C}) = f(\mathrm{A}) \times f(\mathrm{B}) \times f(\mathrm{C})$$

$$\texttt{?A:?B:(?C:)} \Rightarrow f(\texttt{?A:?B:}) \times f(\mathrm{C}) = 2f(\mathrm{A})\, f(\mathrm{B}) \times f(\mathrm{C})$$

이고, 이를 전부 더하면 $5f(\mathrm{A})\, f(\mathrm{B})\, f(\mathrm{C})$가 됩니다.

4개 붙어 있다면, 각각의 경우의 수는

$$\texttt{(?A:)?B:?C:?D:} \Rightarrow f(\text{A}) \times f(\texttt{?B:?C:?D:}) = f(\text{A}) \times 5f(\text{B})\,f(\text{C})\,f(\text{D})$$

$$\texttt{?A:(?B:)?C:?D:} \Rightarrow f(\text{A}) \times f(\text{B}) \times f(\texttt{?C:?D:}) = f(\text{A}) \times f(\text{B}) \times 2f(\text{C})\,f(\text{D})$$

$$\texttt{?A:?B:(?C:)?D:} \Rightarrow f(\texttt{?A:?B:}) \times f(\text{C}) \times f(\text{D}) = 2f(\text{A})\,f(\text{B}) \times f(\text{C}) \times f(\text{D})$$

$$\texttt{?A:?B:?C:(?D:)} \Rightarrow f(\texttt{?A:?B:?C:}) \times f(\text{D}) = 5f(\text{A})\,f(\text{B})\,f(\text{C}) \times f(\text{D})$$

이고, 이 경우에는 $14f(\text{A})\,f(\text{B})\,f(\text{C})\,f(\text{D})$가 됩니다.

- 이제 식이 여러 개 붙어 있다면 각각의 식의 경우의 수를 곱하고 특정 수를 곱해 준다는 사실을 알 수 있습니다.
- 이 특정 수들$(1, 2, 5, 14, \cdots)$을 구글링해 보면 이 수들을 카탈란 수(Catalan number)임 을 알 수 있으며, 실제로 위의 과정이 카탈란 수의 점화식과 동일함을 알 수 있습니다.

$$C_n = \begin{cases} 1 & \text{if } n = 0 \\ \sum_{i=0}^{n-1} C_i C_{n-i} & \text{otherwise} \end{cases}$$

H. yo, i herd u liek ternary operators, so..

- 위의 점화식을 이용해 N번째 카탈란 수를 구하면 $\mathcal{O}\left(N^2\right)$이 걸리지만,

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} = \prod_{k=2}^{n}\frac{n+k}{k}$$

  으로 구할 수 있음이 알려져 있기 때문에 $\mathcal{O}(N)$으로 최적화가 가능합니다.
- 모듈러에서 나눗셈을 하는 방법은 BOJ 13172번에 잘 나와 있습니다.
- 위에서 한 관찰로 f를 최적화해도 되지만 문자열을 하나씩 읽으면서 스택으로도 풀 수 있습니다.

H. yo, i herd u liek ternary operators, so..

```
st = []
for c in s:
  if c == '?': st.push('?')
  elif c == ':':
    subs = []
    while st[-1] != '?':
        subs.add(st[-1])
        st.pop()
    st.pop() # pop '?'
    st.push(catalan(len(subs)) * math.prod(subs))
answer = catalan(len(st)) * math.prod(st)
```

I. 편지 배달

`greedy`, `permutation_cycle_decomposition`
출제진 의도 – **Challenging**

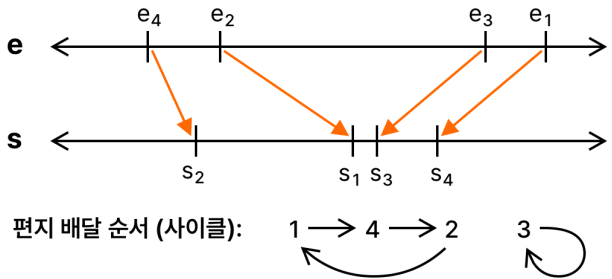
- 제출 52번, 정답 6팀 (정답률 40.00%)
- 처음 푼 팀: **초비상!!** (코딩 말렸어요, 시간초과 떴어요, Ofast가 해줘야 해요), 89분
- 출제자: queued_q
- 가장 짧은 정해: $848B^{\text{Python}}$, $906B^{\text{C++}}$

- 두 반 a, b 사이의 거리를 $d(a, b)$ 로 표기하겠습니다.
- 각 편지를 출발지부터 목적지까지 배달하는 데 필요한 이동 거리는 $d(s_i, e_i)$ 로 고정되어 있으므로, 그 외에 필요한 **추가 이동 거리**를 최소화하는 것이 목표입니다.
- 추가 이동 거리는 각 편지 배달과 배달 사이에 이동하는 거리와 교실로 돌아오는 데 필요한 이동 거리의 합입니다.

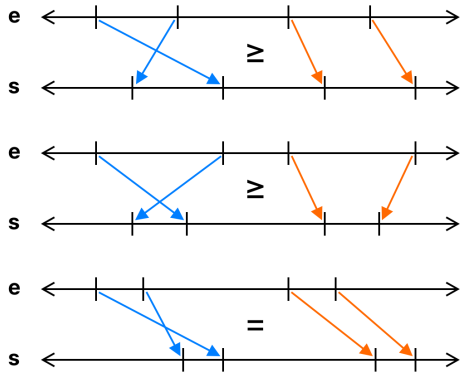
- 어떤 편지의 목록 $a_1, \cdots, a_k$ 를 한 명의 배달원에게 배정하는 상황을 생각합시다.
- 해당 목록을 첫 편지의 출발지(s_{a_1})에 있는 배달원에게 배정하면 추가 이동 거리가 최소가 됩니다.
- 이 때 배달원이 이동해야 하는 추가 이동 거리는 $d(e_{a_i}, s_{a_{i+1}})$ 의 합입니다.
  - 단, 마지막에 자기 반 교실로 돌아와야 하므로 a_k 다음 편지는 a_1 으로 간주합니다.
    따라서 배달할 편지의 목록은 사이클을 이룬다고 생각할 수 있습니다.
- 각 배달원마다 배달할 편지의 목록이 사이클을 이루므로,
  모든 편지에 대해 **그 다음 편지**가 하나씩 정해져 있음을 알 수 있습니다.
  (a_1 다음 편지는 a_2, a_2 다음 편지는 a_3, $\cdots$, a_k 다음 편지는 a_1)

- 모든 편지 i에 대해 그 다음 편지 p_i를 결정해 주었다고 가정합시다.
- $p_1, \cdots, p_m$은 순열을 이루며, 이는 몇 개의 사이클로 분리할 수 있습니다.
- 어떤 순열이 주어지든 각각의 사이클을 적절한 배달원에게 배분할 수 있습니다. 사이클의 첫 편지를 골라서, 그 출발지에 있는 반 배달원에게 해당 사이클을 배정하면 되기 때문입니다.
  - 만약 여러 사이클의 출발지가 같다면 해당 반의 배달원이 여러 사이클을 돌면 됩니다.
- 이 때 추가 이동 거리는 $d(e_i, s_{p_i})$의 합이 됩니다.

- 이제 $d(e_i, s_{p_i})$의 합이 최소인 순열 p를 결정해 주는 일만 남았습니다.
- 이는 e, s 좌표들을 각각 정렬하고 순서대로 매칭하는 그리디 알고리즘으로 찾을 수 있습니다.



– 그리디 알고리즘이 최적인 이유는, 두 매칭 선이 교차하는 경우보다 그렇지 않은 경우가 항상 이득이기 때문입니다.



J. 수 정렬하기, 근데 이제 제곱수를 곁들인

`number_theory`

출제진 의도 – **Hard**

- 제출 927번, 정답 83팀 (정답률 33.60%)
- 처음 푼 팀: **Rinjinbu_SSHS** (EaglesV7, greg3566, ravor8878), 4분
- 출제자: `hyperbolic`
- 가장 짧은 정해: $220B^{\text{Python}}$, $440B^{\text{C++}}$

- 수열 $A_1, A_2, \cdots, A_N$ 이 주어집니다.
- $A_1, A_2, \cdots, A_N$ 을 정렬한 수열을 $B_1, B_2, \cdots, B_N$ 으로 정의합시다.
- 그러면, 이 문제의 답이 YES인 것과 모든 i 에 대하여 $A_i \times B_i$ 가 제곱수인 것과 동치입니다.
- 수열 B 를 얻는데 $\mathcal{O}\left(N \log N\right)$, $A_i \times B_i$ 가 제곱수인지 판단하는데 각 i 마다 $\mathcal{O}\left(\log MAX\right)$ 의 연산이 소요되므로, $\mathcal{O}\left(N \log N + N \log MAX\right)$ 의 연산으로 이 문제의 답을 구할 수 있습니다. 여기서 MAX 는 10^{18} 입니다.

이 문제의 답이 YES인것과 모든 i 에 대하여 $A_i \times B_i$ 가 제곱수인 것과 동치란 것을 증명해봅시다.

먼저, 두 숫자의 값을 바꿀 수 있다는 성질은 equivalence relation이 됩니다. 즉, 다음 3가지 성질이 성립합니다.

- A_i 와 A_i 는 서로 값을 바꿀 수 있습니다. 왜냐하면 $A_i \times A_i$ 는 제곱수이기 때문입니다.
- A_i 와 A_j 가 서로 값을 바꿀 수 있다면 A_j 와 A_i 도 서로 값을 바꿀 수 있습니다. 왜냐하면 $A_i \times A_j$ 가 제곱수라면 $A_j \times A_i$ 도 제곱수이기 때문입니다.

- A_i 와 A_j, A_j 와 A_k 가 서로 값을 바꿀 수 있다면, A_i 와 A_k 도 서로 값을 바꿀 수 있습니다.
  왜냐하면 $A_i \times A_k = \frac{(A_i \times A_j)(A_j \times A_k)}{A_j^2}$ 고, $A_i \times A_j, A_j \times A_k, A_j^2$ 이 제곱수므로, $A_i \times A_k$ 도 제곱수이기 때문입니다.

  Equivalence relation이 존재하므로 이에 대응하는 equivalence class가 존재하며, 따라서 같은 class에 속하는 두 수는 무조건 값을 바꿀 수 있고, 다른 class에 속하는 두 수는 무조건 값을 바꿀 수 없다는 것을 알 수 있습니다.

수열 B 를 A 를 정렬한 수열이라고 정의합시다. 저희는 수열 A 를 수열 B 로 만들려고 합니다. 즉, 모든 i 에 대하여, A_i 의 값을 적절하게 바꾸어 B_i 로 만들어야 합니다. 이전 슬라이드에서 언급한 class에 대한 내용을 떠올려보면, A_i 의 값을 B_i 로 바꿀 수 있다는 것과 A_i 와 B_i 가 같은 class 안에 속한다는 것은 동치입니다. 즉, $A_i \times B_i$ 가 제곱수여야 합니다.
따라서, 모든 i 에 대하여 A_i 를 B_i 로 바꿀 수 있다는 것과 모든 i 에 대하여 $A_i \times B_i$ 가 제곱수여야 한다는 것은 동치입니다.

$A_i \times B_i$ 가 제곱수라는 것을 어떻게 판단할까요?

- 단순히 $\sqrt{A_i \times B_i}$ 를 계산하는 것은 무리가 있습니다. 왜냐하면 $A_i \times B_i$ 가 64bit 정수 범위를 벗어나기 때문입니다.
- $g = \gcd(A_i, B_i)$ 로 정의합시다. 그러면, $A_i \times B_i$ 가 제곱수라는 것과 $\frac{A_i}{g}, \frac{B_i}{g}$ 가 모두 제곱수라는 것이 동치입니다.
- $\frac{A_i}{g}, \frac{B_i}{g}$ 는 64bit 정수 범위 내에 들어가기 때문에, 이분탐색등의 방법을 사용하면 $\mathcal{O}\left(\log MAX\right)$ 의 연산으로 제곱수인지 정확하게 판단이 가능합니다.