한양대학교 ERICA 프로그래밍 경시대회 HEPC 2024

Official Solutions

by

한양대학교 ERICA캠퍼스 알고리즘 학회 영과일

2024년 5월 20일

후원

START LINK

운영





ZERO	ONE		의도한 난이도	출제자
A		손가락 게임	**Beginner**	박준성 rasauq1122
B		홀짝홀짝	**Easy**	이민구 qsccsq22
C		수박 게임	**Beginner**	박준성 rasauq1122
D		선배님 밥 사주세요!	**Easy**	길정민 kjmin622
E	**A**	버블버블	**Easy**	이민구 qsccsq22
	B	영일랜드	**Medium**	길정민 kjmin622
F	**C**	강의실	**Medium**	권혁준 khj20006
G	**D**	별 수호자 룰루	**Medium**	박준성 rasauq1122
	E	땅땅바 나누기	**Challenging**	박준성 rasauq1122
I	**F**	함수 루프 탐지	**Challenging**	오익준 putdata
H	**G**	자존심 강한 두 천재의 싸움	**Hard**	박준성 rasauq1122
	H	EDF	**Hard**	이민구 qsccsq22
	I	트리 스도쿠	**Medium**	박준성 rasauq1122
	J	바이러스 시뮬레이터	**Challenging**	권혁준 khj20006

0A. 손가락 게임

`implementation`
출제진 의도 – **Beginner**

@ ZERO division

- 제출 -번, 정답 -팀 (정답률 –.—%)
- 처음 푼 팀: -, -분
- 출제자: 박준성`rasauq1122`

@ ONE division

- 제출 -번, 정답 -팀 (정답률 –.—%)
- 처음 푼 팀: -, -분

0A. 손가락 게임



- ✓ 가능한 경우의 수가 $6 \times 6 = 36$ 가지이므로 모든 경우에 대해 예외처리를 진행해주면 됩니다.
- ✓ 혹은 지문에 나와있는대로 구현해주면 됩니다.

1. 둘 다 '가위', '바위', '보' 표식 중 하나를 냈다면 가위바위보로 승패를 가립니다.
2. 둘 중 한 사람만 냈다면, '무효' 표식을 낸 사람이 패배합니다.
3. 둘 다 내지 않았다면, 비깁니다.

0B. 홀짝홀짝

`implementation`
출제진 의도 – **Easy**

@ ZERO division

- 제출 -번, 정답 -팀 (정답률 –.—%)
- 처음 푼 팀: **-**, -분
- 출제자: 이민구qsccsq22

@ ONE division

- 제출 -번, 정답 -팀 (정답률 –.—%)
- 처음 푼 팀: **-**, -분

0B. 홀짝홀짝



- 자릿수 N 이 최대 100 만 자리여서 C++ 이나 Java 의 경우 `int` 나 `long long` 범위를 벗어나고, Python 의 경우 `int` 형으로 입력을 받을 경우 시간초과가 나게 됩니다.
- 따라서 입력을 정수형태가 아닌 문자열 형태로 입력을 받아야합니다.
- 입력으로 주어진 값에 대하여 각 자릿수가 짝수인지 홀수인지를 판별하는 과정을 모든 자릿수에 대해서 진행합니다.
- 시간 복잡도는 $\mathcal{O}(N)$ 입니다.

0C. 수박 게임

`implementation`
출제진 의도 – **Beginner**

@ ZERO division

- 제출 -번, 정답 -팀 (정답률 –.—%)
- 처음 푼 팀: -, -분
- 출제자: 박준성`rasauq1122`

@ ONE division

- 제출 -번, 정답 -팀 (정답률 –.—%)
- 처음 푼 팀: -, -분

0A. 수박 게임



- i 단계 과일이 m 개 있으면, $i+1$ 단계 과일은 $\lfloor \frac{m}{2} \rfloor$ 개를 만들 수 있습니다.
- 이는 반복문으로 간단하게 구현할 수 있습니다.
- 혹은 비트연산자를 이용해서 반복문 없이 구할 수도 있습니다.

0D. 선배님 밥 사주세요!

`implementation`
출제진 의도 – **Easy**

@ ZERO division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: -, -분
- ✓ 출제자: 길정민 `kjmin622`

@ ONE division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: -, -분

D. 선배님 밥 사주세요!



정민이가 선배와 잡은 약속의 기록과 선배가 소지한 돈이 주어졌을 때,
최대 며칠 연속으로 밥을 얻어먹을 수 있는지 출력합니다.

- ✓ 먼저 정민이의 모든 밥약속 정보인 이름, 가격, 날짜를 배열에 저장합니다.
  여기서 W 주차 D 번째 요일로 주어지는 날짜는 $W \times 7 + D$ 로 바꿔서 저장하는게 용이합니다.

D. 선배님 밥 사주세요!



- 다음으로 선배가 소지한 돈이 주어집니다.
  앞서 저장한 기록과 비교하며 해당 날짜에 약속을 잡을 수 있는지 판별합니다.

- W 는 최대 10, D 는 최대 6 이므로, 가능한 가장 늦은 약속은 76 일차입니다.
  따라서 가능한 약속을 기록하는, 크기가 77 인 0 으로 초기화된 약속 배열을 만듭니다.
  각 인덱스 자리에는 해당 일차에 가능한 약속이 있다면 1 을 저장합니다.

- 선배 이름과 소지한 돈을 받았을 때,
  저장한 기록에서 입력 받은 선배 이름과 같은 기록을 찾고,
  소지한 돈에서 기록된 가격을 뺀 값이 0 이상이라면 선배는 밥을 사줄 수 있습니다.
  따라서 소지한 돈 $-$ 기록된 가격 ≥ 0 일 때, 약속 배열[날짜]에 1 을 저장합니다.

D. 선배님 밥 사주세요!



- 마지막으로, 최대 연속 일수를 구하기 위해 모든 기록을 다시 순회합니다.
  어떤 기록에 대하여, 그 기록으로부터 며칠 연속으로 밥을 얻어먹을 수 있는지 계산합니다.
  즉, 배열에서 연속으로 1 이 나오는 가장 큰 부분 배열의 길이를 구합니다.

  이를 쉽게 구현하면 약속 배열을 처음부터 순회하며 1 이 나왔을 때의 인덱스로부터
  연속으로 몇 개의 1 이 나오는지 구하며 최대 길이를 구할 수 있습니다.

0E/1A. 버블버블

`sorting, greedy`
출제진 의도 – **Easy**

@ ZERO division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: -, -분
- ✓ 출제자: 이민구qsccsq22

@ ONE division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: -, -분



- ✓ swap 연산과 최대 딱 한번 배열 전체를 뒤집는 연산이 주어졌을 때, 최소 횟수로 주어진 배열을 오름차순으로 만드는 최적의 방법엔 2 가지 경우가 있습니다.

1. swap 연산을 통해 오름차순으로 만드는 방법
2. swap 연산을 통해 내림차순으로 만든 후 뒤집는 방법



첫 번째 방법

- ✓ 주어진 배열을 swap 연산으로 횟수를 세주면서 오름차순으로 만들어주면 됩니다.



두 번째 방법

- ✓ 오름차순 정렬까지 swap 연산 횟수가 k 번 남고, 내림차순 정렬까지 t 번 남았다고 가정합시다.
- ✓ $k > t$ 일 경우 내림차순으로 만들고 뒤집으면 횟수가 $(t+1)$ 번이므로 $k \geq (t+1)$ 이 성립합니다.

0E/1A. 버블버블



- 오름차순으로 정렬하다가 중간에 뒤집고 다시 오름차순으로 정렬하는 경우는 다음과 같기에 최적이 아닙니다.
- 오름차순으로 정렬하다가 중간에 뒤집는 것은 더 내림차순으로 만드는 것이고, 이는 다시 오름차순으로 만들 때의 횟수가 증가하는 것이기 때문에 해당 방법은 최적이 아닙니다.

0E/1A. 버블버블



- 내림차순으로 정렬하다가 중간에 뒤집고 오름차순으로 정렬하는 경우 다음과 같습니다.
- 내림차순으로 만드는데 필요한 횟수가 t번 이라고 가정합시다.
- 버블 정렬 특성상 내림차순으로 k번 정렬했다면 배열을 뒤집고 오름차순으로 만드는데 $(t-k)$번이 필요합니다.
- 따라서 어느 시점에서 배열을 뒤집던 내림차순으로 정렬하다가 뒤집고 오름차순으로 정렬하는 횟수는 동일합니다.

0E/1A. 버블버블



- 따라서 배열을 오름차순 정렬이 될 때까지 swap한 횟수를 a, 배열을 내림차순 정렬이 될 때까지 swap한 횟수를 b라고 하면 정답은 $\min(a, b+1)$ 입니다.
- 첫 번째 방법과 두 번째 방법 모두 배열을 정렬하는데 $\mathcal{O}\left(N^2\right)$ 이므로 총 시간 복잡도는 $\mathcal{O}\left(N^2\right)$ 입니다.

1B. 영일랜드

`backtracking`
출제진 의도 – **Medium**

@ ZERO division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: **-**, -분
- ✓ 출제자: 길정민`kjmin622`

@ ONE division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: **-**, -분

1B. 영일랜드

그래프가 주어졌을 때, 가장 긴 해밀턴 사이클의 크기를 구합니다.
해밀턴 사이클이란, 모든 노드를 정확히 한 번 방문하고 출발점으로 돌아오는 경로입니다.

- ✓ 다행히 N 이 작으므로, 백트래킹으로 답을 구할 수 있습니다.

- ✓ 그래프에는 루프가 존재하고, 두 노드를 잇는 간선이 여러 개일 수 있음을 주의합니다.
각 노드는 한 번씩만 방문해야하므로 자기 자신을 향하는 간선은 저장하지 않고,
최장 거리를 구해야하므로 간선 중 가장 시간이 오래걸리는 간선만 저장하면 됩니다.

0F/1C. 강의실

`greedy, sorting`
출제진 의도 – **Medium**

@ ZERO Division

- 제출 -번, 정답 -팀 (정답률 -.-%)
- 처음 푼 팀: -, -분
- 출제자: 권혁준 `khj20006`

@ ONE Division

- 제출 -번, 정답 -팀 (정답률 -.-%)
- 처음 푼 팀: -, -분



- 순간이동을 할 수 없다고 가정해봅시다.
- 단순히, 좌표의 오름차순으로 방문하는 것이 최단 시간임을 알 수 있습니다.
- 따라서 $A_1, A_2, \ldots, A_N$ 중 최댓값이 답이 됩니다.



- 주어진 수열 A를 오름차순으로 정렬시켜놓고 생각해봅시다.
- $x = 0$에서 A_1으로 가는 데 걸리는 시간을 B_1이라 하고,
  A_{i-1}에서 A_i로 가는 데 걸리는 시간을 B_i라고 합시다.
- 이동 과정 전체를 N개의 구간을 지나는 것으로 단순화한 것입니다.
- 그러면, 답을 $\sum_{i=1}^{N} B_i$로도 생각할 수 있습니다.



- 이제, 순간이동이라는 요소를 추가해봅시다.
- 순간이동을 하는 시점은 어느 강의실 하나에 도착했을 때 바로 하는 것이 최적입니다.
- 다른 강의실로 가는 도중에 순간이동을 하는 것은 지나온 거리만큼의 시간을 더 소모하기 때문입니다.
- 같은 이유로, 순간이동의 도착점은 반드시 아직 가지 않은 강의실 중 하나여야 합니다.
- 즉, 한 번의 순간이동은 B_i 중 하나를 0으로 만드는 것입니다.



- 따라서 수열 B 를 구해 오름차순 정렬한 뒤, 앞에서 $N - K$ 개의 값만을 더해준 것이 답이 됩니다.
- N 개의 원소에 대해 정렬을 수행하므로, 전체 시간복잡도는 $\mathcal{O}\left(N \log N\right)$ 입니다.

0G/1D. 별 수호자 룰루

ad_hoc
출제진 의도 – **Medium**

@ ZERO division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: -, -분
- ✓ 출제자: 박준성 `rasauq1122`

@ ONE division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: -, -분



- 가장 간단한 방법인 '번호 순으로 배정하기'에 대해서 생각해봅시다.
- K 가 짝수라면, 번호 순으로 배정해도 모든 조에서 전투력의 합이 K 로 나누어떨어지지 않습니다.
- 등차수열의 합 공식을 이용하면 이를 간단히 증명할 수 있습니다.



- 반대로 K 가 홀수라면, 모든 조에서 전투력의 합이 K 로 나누어떨어집니다.
- 모든 조에서 전투력의 합이 K 로 나누어떨어지지 않도록 조원을 바꾸는 방법에 대해서 생각해봅시다.

- 먼저 조를 바꾸는 두 조원의 전투력 a와 b는 K로 나눈 나머지가 같으면 안됩니다.
- 즉, $a \neq b \mod K$ 이어야 합니다.
- 만약 그렇지 않다면, 조원을 바꾸기 전의 전투력의 합이 S일 때 $S \equiv 0 \mod K$ 이므로 조원을 바꾼 후의 전투력의 합 또한 $S - a + b \equiv 0 \mod K$ 이기 때문입니다.

조의 개수가 2개일 경우 어떻게 풀 수 있을지 생각해봅시다. (즉, $N = 2K$)

1. 각 조에서 전투력을 K로 나눈 나머지가 다른 조원을 선택한다.
2. 두 조원을 교환한다.

이 방법을 이용하면 $N = 2K$인 경우에 대해서 문제를 해결 할 수 있습니다. 후술할 설명을 위해 이 방법을 **2-교환**이라고 부르겠습니다.

$N = 3K$ 인 경우에 대해서도 생각해봅시다.

1. 각 조에서 전투력을 K 로 나눈 나머지가 서로 다르도록 조원을 선택한다.
2. 원래 첫 번째 조에 있던 조원은 두 번째 조, 두 번째 조에 있던 조원은 세 번째 조, 세 번째 조에 있던 조원은 첫 번째 조에 소속되도록 한다.

이 방법을 이용하면 $N = 3K$ 인 경우에 대해서도 문제를 해결할 수 있고, 역시 이 방법을 **3-교환**이라고 부르겠습니다.

- **2-교환**과 **3-교환**을 이용하면 $N = 4K$ 인 경우, $N = 5K$ 인 경우에도 해결할 수 있습니다.
- $N = 4K$ 인 경우, 4개의 조를 2개 조/2개 조로 나눈 후 각각 **2-교환**을 진행합니다.
- $N = 5K$ 인 경우, 5개의 조를 3개 조/2개 조로 나눈 후 각각 **3-교환**과 **2-교환**을 진행합니다.
- 이처럼 **2-교환**과 **3-교환**을 이용하면, 조가 2 개 이상 있다면 항상 문제를 해결할 수 있을 것 같습니다.

K가 홀수고 $N = iK$라고 할 때, 문제를 다음과 같이 일반화하여 해결할 수 있습니다.

1. $i = 1$이라면 교환을 할 수 없으므로, 항상 전투력의 합이 K로 나누어떨어집니다.
2. $i = 2k$라면 $2k$개의 조를 2개씩 묶은 후, 총 k개의 조 쌍에 대하여 **2-교환**을 각각 진행합니다.
3. $i = 2k + 1$이라면 먼저 3개 조를 골라 **3-교환**을 진행한 후, 나머지 $2k - 2$개의 조를 2개씩 묶어 **2-교환**을 각각 진행합니다.

이를 구현하면 시간복잡도 $\mathcal{O}(N)$에 문제를 해결할 수 있습니다.

1E. 땅땅바 나누기

`geometry`, `partial_sum`
출제진 의도 – **Challenging**

@ ZERO division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: -, -분
- ✓ 출제자: 박준성`rasauq1122`

@ ONE division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: -, -분



다음과 같이 구현하면 됩니다.

1. 모든 보물을 시계 방향(혹은 반시계 방향)으로 정렬한다.
2. 정렬한 순서대로, 보물을 지나는 직선을 긋고 시계 방향(혹은 반시계 방향)으로 살짝 돌린다.
3. (2)에서 구한 직선에 대해 준성이 얻을 수 있는 보물들의 가치의 합을 구한다.
4. (3)에서 구한 보물의 가치의 합 중 최댓값일 때의 직선을 출력한다.



- 사용하는 알고리즘이 복잡하지 않지만 구현이 어려운 문제라고 생각합니다.
- 따라서 각 단계 별로 어떻게 구현하면 좋을지에 대해 간단히 소개하는 것으로 풀이를 대신하겠습니다.



1. 모든 보물을 시계 방향(혹은 반시계 방향)으로 정렬한다.

✓ 구조체(혹은 클래스)를 선언한 후, 연산자 오버로딩 혹은 구조체를 비교하기 위한 비교 함수를 구현하면 좋습니다.

✓ 먼저 사분면을 기준으로 분류하고, 같은 사분면에 있는 보물에 대해서는 기울기로 비교합니다.



2. 정렬한 순서대로, 보물을 지나는 직선을 긋고 시계 방향(혹은 반시계 방향)으로 살짝 돌린다.

- ✓ 사실 '살짝 돌린' 직선의 방정식을 현재 단계에서 정확히 구할 필요는 없습니다.
- ✓ 따라서 (4)에서 해당 내용에 대해 더 자세히 다루겠습니다.

3. (2)에서 구한 직선에 대해 준성이 얻을 수 있는 보물들의 가치의 합을 구한다.

- ✓ 보물들의 가치의 합은 누적 합 혹은 두 포인터로 $\mathcal{O}(1)$로 구할 수 있습니다.
- ✓ 현재 보물의 인덱스와 현재 보물에 정반대편에 위치한 보물의 인덱스를 잘 관리해주면 됩니다.
- ✓ 정반대편에 위치한 보물의 인덱스을 따로 저장하지 않고 이분 탐색을 이용하여 필요할 때마다 구해주는 방법도 가능합니다.
- ✓ 보물들의 가치의 합을 구할 때 정반대편에 위치한 보물의 가치와 현재 보물의 가치를 동시에 포함하여 구하지 않도록 주의해야 합니다. (즉, 둘 중 하나만 포함해야 합니다.)

4. (3)에서 구한 보물의 가치의 합 중 최댓값일 때의 직선을 출력한다.

- ✓ “직선의 방정식 $ax = by$은 점 (b, a)를 항상 지난다.”을 이용하면 됩니다.
- ✓ 보물의 좌표를 지나는 직선의 방정식을 그대로 출력하면, 그 보물은 성우가 갖게 되므로 약간 빗나가도록 출력해야 합니다.
- ✓ ‘살짝 돌린’ 직선의 방정식을 구하기 위한 유효한 방법은 해당 보물과 그 보물에서 시계 방향(혹은 반시계 방향)으로 가장 가까운 보물의 중점을 지나는 직선의 방정식을 출력하는 것입니다.

1E. 땅땅바 나누기



- 앞서 말한 두 보물의 좌표를 각각 (x_1, y_1), (x_2, y_2) 라고 하면 직선의 방정식은 $(y_1 + y_2)x = (x_1 + x_2)y$ 입니다.
- 출력해야 하는 a와 b의 범위가 보물의 최대 좌표의 2배 이므로 항상 출력할 수 있습니다.
- 다만 시계 방향(혹은 반시계 방향)으로 가장 가까운 보물을 고려할 때는 같은 사분면 뿐만 아니라 원점을 기준으로 대칭되는 사분면 또한 고려해야 합니다.
- 또한 출력하는 a와 b가 모두 0이 되지 않도록 잘 관리해주어야 합니다.

0I/1F. 함수 루프 탐지

`implementation, ad_hoc`
출제진 의도 – **Challenging**

@ ZERO Division

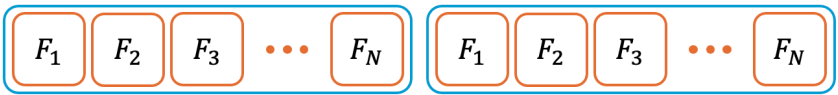
- ✓ 제출 -번, 정답 -팀 (정답률 -.-%)
- ✓ 처음 푼 팀: -, -분
- ✓ 출제자: 오익준`putdata`

@ ONE Division

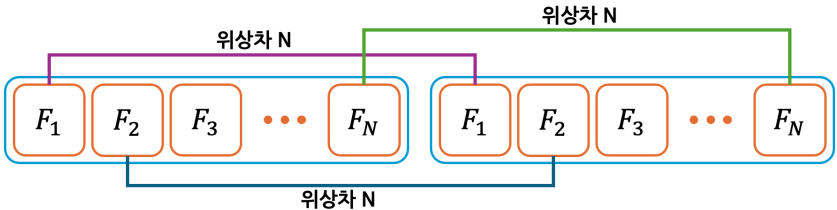
- ✓ 제출 -번, 정답 -팀 (정답률 -.-%)
- ✓ 처음 푼 팀: -, -분



- 어떤 함수 호출 구간 $[F_1, F_2, \ldots, F_N]$ 이 연속하여 두 번 반복될 경우 다음 그림과 같습니다.



- 두 구간의 어떤 함수명 F_i 에 대해 위상차는 모두 N 으로 동일하고 F_N 다음에 F_1 이 등장합니다.



- 마찬가지로 $K(2 \leq K)$ 번 반복될 경우에도 어느 연속한 두 구간에 대한 위상차 역시 N 으로 동일하고 F_N 다음에 F_1 이 등장하게 됩니다.



- 우리는 함수 호출 구간이 모두 서로 다른 원소로 이루어져 있음에 주목해야 합니다.
- 함수 호출 리스트 $[A_1, A_2, ...A_N]$ 가 있다고 해봅시다.
- $P(A_i)$ 를 A_1 부터 A_{i-1} 까지 A_i 와 동일한 함수명을 가진 원소의 가장 큰 위치라고 합시다.
  (여기서 A_i 의 위치는 i 이다.)



- 함수명 A_i 로 시작하는 함수 호출 구간이 연속하여 반복됐는지 확인하는 방법은 다음과 같습니다.
- 만약 $P(A_i)$ 가 존재한다면, $A_{P(A_i)}$ 부터 A_{i-1} 가 반복됐음을 가정하고 이 때 구간의 길이는 $i - P(A_i)$ 와 같습니다.
- 정수 $j(0 < j < i - P(A_i))$ 에 대해 $i + j$ 과 $P(A_{i+j})$ 의 차이가 모두 $i - P(A_i)$ 이고, $P(A_{i+i-P(A_i)-1}) + 1 = i$ 라면 $A_{P(A_i)}$ 부터 A_{i-1} 이 두번 반복된 것을 알 수 있습니다.
- 만일 중간에 차이가 $i - P(A_i)$ 가 아니거나, $P(A_{i+i-P(A_i)-1}) + 1 \neq i$ 이면 함수명 A_{i+j} 부터 다시 반복임을 가정하고 위의 과정을 시도하면 됩니다.



- 이와 같은 아이디어를 이용하여 처음부터 끝까지 함수명을 한 번씩만 훑어보며 여러 번 반복된 구간과 횟수를 구하면 됩니다.
- 문자열을 정수로 매핑 시켜주는 맵을 이용하면 문자열의 최대 길이가 10으로 작기에 총 시간복잡도는 $\mathcal{O}\left(N \log N\right)$입니다.

0H/1G. 자존심 강한 두 천재의 대결

`ad_hoc, math`

출제진 의도 – **Hard**

@ ZERO division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: -, -분
- ✓ 출제자: 박준성 `rasauq1122`

@ ONE division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: -, -분



- 길이가 k인 수를 진법 변환하는 방법의 시간 복잡도는 $\mathcal{O}\left(k^2\right)$입니다.
- k는 최대 $3\,000\,000$이므로 단순히 진법 변환을 통해 문제를 해결하려고 하면 TLE를 받게 됩니다.



- ✓ 따라서 문제를 효율적으로 풀기 위해서는 약간의 관찰이 필요합니다.
- ✓ 보다 쉬운 관찰을 위해 극단적인 상황을 가정해봅시다.



- 길이가 $3\,000\,000$인 두 수 a, b가 각각 10진법 정수, 2진법 정수라고 합시다.
- 그럼 a의 최솟값은 $10^{2\,999\,999}$고, b의 최댓값은 $2^{3\,000\,000} - 1$입니다.
- b의 최댓값은 $2^{3\,000\,000} = (2^{10})^{300\,000} < (10^4)^{300\,000} = 10^{1\,200\,000}$이므로 a의 최솟값인 $10^{2\,999\,999}$보다 항상 작습니다.
- 즉, a와 b는 진법 변환을 하지 않고도, a가 b보다 항상 크다는 것을 알 수 있습니다.



- 위 예제를 통해, 수의 길이가 어느 정도 길어지면 두 수를 비교해볼 것도 없이 더 큰 진수를 사용하는 수가 더 크다는 것을 알 수 있습니다.
- 그러한 수의 길이를 k_0 라고 하고, 그 값을 구해 봅시다.



- ✓ 10진법 정수 a와 9진법 정수 b가 있고, 두 수의 길이가 k_0라고 합시다.
- ✓ k_0가 a의 최솟값이 b의 최댓값보다 항상 크게 하는 값이라고 하면 다음 부등식이 성립합니다: $10^{k_0-1} > 9^{k_0} - 1$
- ✓ 이 부등식을 만족시키는 가장 작은 정수 k_0는 `python`을 통해 쉽게 구할 수 있으며, 그 값은 22입니다.
- ✓ 혹은 k_0를 정확히 구하지 않고도, 직관적으로 적당히 큰 수(ex. 100)를 골라도 문제를 해결할 수는 있습니다.



길이가 k인 u진법 정수 a와 v진법 정수 b가 주어질 때 다음과 같이 문제를 해결할 수 있습니다.

1. $u = v$라면, 가장 자릿수가 큰 수부터 하나씩 비교하여 두 수의 대소 관계를 구한다.
2. $u \neq v$이고 $k < k_0$라면, 두 수를 같은 진법으로 변환하여 두 수의 대소 관계를 구한다.
3. $u \neq v$이고 $k \geq k_0$라면, 더 큰 수를 진수로 사용하는 수가 더 큰 수다.

기본적으로 python으로 해결하는 편이 고려해야 할 요소가 적지만, C/C++로도 문제없이 해결할 수 있습니다.

1H. EDF

`implementation, priority_queue`
출제진 의도 – **Hard**

@ ZERO division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: -, -분
- ✓ 출제자: 이민구qsccsq22

@ ONE division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: -, -분

- Naive한 방법으로는 처음 N 개의 작업에 대해서 마감 시각을 기준으로 오름차순 정렬 후, 작업이 하나씩 추가 될 때마다 다시 마감 시각을 기준으로 오름차순 정렬을 해줌으로써 풀 수 있습니다.
- 하지만 위 방법은 $\mathcal{O}\left(M(N+M)\log(N+M)\right)$ 만큼의 시간복잡도가 걸리게 됩니다.
- 따라서 작업이 추가될 때마다 정렬해주는 방법이 아닌, 추가되는 즉시 마감 시각을 기준으로 정렬이 되게끔 우선순위 큐를 사용해야합니다.

1H. EDF

- 처음 N 개의 작업과 추가되는 M 개의 작업은 각 우선순위 큐에 넣어놓습니다.
- 이후 현재 시각 $+$ 현재 작업에 걸리는 시간을 a, 추가되는 작업의 추가되는 시각을 b라고 가정하고 비교를 합니다.
- 이때, a와 b를 비교해보면 2가지의 경우가 존재합니다.

첫 번째는 $(a \leq b)$ 인 경우입니다

- 이 경우는 현재 하고있는 작업을 새로운 작업이 추가되기 전에 완료할 수 있는 경우입니다.
- 현재 시각을 작업을 완료한 시각으로 갱신 후 우선순위 큐에서 해당 작업을 제거하고 다음 작업을 진행합니다.



두 번째는 $(a > b)$ 인 경우입니다

- 이 경우는 현재 하고있는 작업을 새로운 작업이 추가되기 전에 완료할 수 없는 경우입니다.
- 현재 하고 있던 작업을 k 만큼 했다면 기존의 작업에 걸리는 시간을 t 라고 했을 때, 작업에 걸리는 시간이 $(t - k)$ 인 새로운 작업으로 변경해줍니다.
- 이후 추가되는 작업을 우선순위 큐에 추가해준 후, 다시 마감 시각이 이른 작업부터 작업을 진행합니다.

- 앞선 두 가지 경우에 대해서 처리를 해주면 $(N+M)$ 개의 작업을 마감 시각이 이른 순으로 완료할 수 있습니다.
- 따라서 전체 시간 복잡도는 $\mathcal{O}\left((N+M)\log(N+M)\right)$ 입니다.

11. 트리 스도쿠

`tree, bfs, sorting`
출제진 의도 – **Medium**

@ ZERO division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: -, -분
- ✓ 출제자: 박준성 `rasauq1122`

@ ONE division

- ✓ 제출 -번, 정답 -팀 (정답률 –.—%)
- ✓ 처음 푼 팀: -, -분



- 인접한 두 정점의 매핑되는 값의 합(이하, 두 정점의 합)이 서로 다르기 위해서는 전략적으로 정점에 값을 매핑해야 합니다.
- 많은 전략이 있을 수 있겠지만, 그 중에서 **두 정점의 합이 점점 커지도록 매핑**하는 방법에 대해 소개하겠습니다.



결론부터 말하자면 아래와 같습니다.

1. 배열 A를 오름차순 정렬한다.
2. 트리를 너비 우선 탐색으로 방문한 순서대로의 정점 번호 리스트 T를 얻는다.
3. T_i번 정점(이하 T_i)에 A_i를 매핑한다.



- 정말 이 방법으로 항상 두 정점의 합이 점점 커지도록 매핑할 수 있을까요?
- $P(T_i)$를 T_i의 부모 정점 번호라고 합시다.
- T_i에 A_i를 순서대로 매핑한다고 상상해보면,
  매핑될 때마다 두 정점의 합이 $A_i + A_{P(T_i)}$인 간선이 하나씩 추가된다고도 볼 수 있습니다.

- 따라서 $A_i + A_{P(T_i)} < A_{i+1} + A_{P(T_{i+1})}$ 임을 증명하면 됩니다.
- 너비 우선 탐색의 특성에 의해서 $idx(P(T_i)) \leq idx(P(T_{i+1}))$ 가 항상 성립합니다.
  ($idx(t)$ 는 T 에서 t 의 인덱스)
- 수열 A 는 정렬되어있으므로, $A_{P(T_i)} \leq A_{P(T_{i+1})}$ 이고, $A_i < A_{i+1}$ 입니다.
- $A_i + A_{P(T_i)} < A_{i+1} + A_{P(T_i)} \leq A_{i+1} + A_{P(T_{i+1})}$ 이므로 위 명제는 참입니다.



- 항상 새로 추가되는 간선의 값(즉, 두 정점의 합)이 지금껏 존재한 간선의 값보다 크므로 모든 간선의 값이 다릅니다.
- 수열을 정렬하는데 $\mathcal{O}(n \log n)$, 너비 우선 탐색에 $\mathcal{O}(n)$ 이므로 총 시간 복잡도는 $\mathcal{O}(n \log n)$ 입니다.

1J. 바이러스 시뮬레이터

`segtree`

출제진 의도 – **Challenging**

@ ZERO Division

- ✓ 제출 -번, 정답 -팀 (정답률 -.-%)
- ✓ 처음 푼 팀: -, -분
- ✓ 출제자: 권혁준 `khj20006`

@ ONE Division

- ✓ 제출 -번, 정답 -팀 (정답률 -.-%)
- ✓ 처음 푼 팀: -, -분

- 1 번 쿼리는 $A_i = 0$ 인 사람에게는 적용할 필요가 없습니다.
- 주어진 1 번 쿼리들로 인해 A_i 가 0 으로 변하는 총 횟수를 N_1 이라 하면,
- 2 번 쿼리가 없다고 가정할 시 $N_1 \leq N$ 입니다.



- 2번 쿼리가 있다면 어떻게 될까요?
- 2번 쿼리로 인해 0이었던 원소가 0 초과가 되는 상황은 최대 Q번 일어납니다.
- 이 원소들이 다시 0으로 변할 수 있다고 보면, $N_1 \leq N + Q$입니다.
- 따라서, 1번 쿼리가 적용될 수 있는 총 횟수는 $\mathcal{O}(N + Q)$입니다.



- 최댓값과 그 인덱스를 저장하는 세그먼트 트리를 관리하면, 1 번 쿼리는 $\mathcal{O}\left((N+Q)\log N\right)$ 에 처리할 수 있습니다.
- 2 번 쿼리는 연산 당 $\mathcal{O}\left(\log N\right)$ 에 처리할 수 있습니다.
- 3 번 쿼리 또한, 구간 연속 합 테크닉으로 연산 당 $\mathcal{O}\left(\log N\right)$ 에 처리할 수 있습니다.
- 따라서 전체 시간복잡도는 $\mathcal{O}\left((N+Q)\log N\right)$ 입니다.