SNUPC 2022 풀이

Official Solutions

by

SNUPC 2022

2022년 9월 17일

잠시 주목해주세요!

- **2022 한국 대학생 프로그래밍 대회 (ACM-ICPC 2022 Seoul Regional)**가 열립니다!
- ACM-ICPC는 3명이 한 팀을 이뤄 알고리즘 문제를 해결하는 대회입니다. 매년 우수한 성적을 거둔 팀에게는 상금과 더불어 세계 대회 진출권이 부여됩니다.
- 팀원들과 함께 실력을 키우고 좋은 추억을 만들 수 있는 기회이니, 부담 없이 도전해 보세요!
- 참가 신청 마감일은 **9월 25일 (일)**입니다. 많은 신청 바랍니다!
  - 아직 팀원을 구하지 못하셨다면 SNUPS 슬랙에서 팀원을 모집해 보세요.
  - 참가 신청 방법은 `https://snups.org/icpc/`에 설명되어 있습니다.
  - 예선 대회는 **10월 7일 (금)-10월 8일 (토)**, 본선 대회는 **11월 18일 (금)-11월 19일 (토)**에 열립니다.
  - 기타 대회에 관한 자세한 정보는 `http://icpckorea.org/`에 나와 있습니다.

Div 2 문제		의도한 난이도	출제자
A	샤틀버스	**Easy**	cubelover, IHHI
B	SNUPTI	**Easy**	jhwest2, sait2000
C	3에 깃든 힘	**Medium**	snuphy
D	이상한 프로그래밍 언어	**Medium**	kipa00
E	자취방 정하기	**Medium**	jhwest2
F	콜라 줍기	**Hard**	kipa00, cakelemon
G	슬라임 키우기	**Hard**	TAMREF
H	포켓몬 대회	**Challenging**	cakelemon

	Div 1 문제	의도한 난이도	출제자
A	사회적 거리두기	**Medium**	IHHI
B	자리 바꾸기	**Medium**	cubelover
C	이름 부르기	**Challenging**	snuphy
D	정렬 프로그램	**Hard**	jhwest2
E	영희의 심부름	**Medium**	cakelemon
F	MST	**Challenging**	sebinkim
G	수 만들기	**Challenging**	cubelover
H	자취방 정하기	**Easy**	jhwest2
I	히스토그램 하나 빼기	**Hard**	TAMREF
J	넓이를 같게	**Hard**	jhwest2

2A. 샤틀버스

`arithmetic`

- 제출 52번, 정답 41명 (정답률 80.769%)
- 처음 푼 사람: **김상훈**, 1분
- 출제자: `cubelover`, `IHHI`

2A. 샤틀버스

- x가 편도 시간이고, y가 현재 버스가 종점까지 가는데 필요한 시간일 때, 버스가 시점에 언제 도착하는지를 구하는 문제입니다. ($1 \leq y < 2x; x \neq y$)
- 버스는 종점을 향해 움직이는 중이거나, 혹은 시점을 향해서 움직이는 중일 수 있습니다.
- $x > y$인 경우 버스가 종점을 향해 가는 중이므로, 종점에 도착한 뒤 편도 시간을 더해준 $x + y$가 답이 됩니다.
- $x < y$인 경우 버스가 시점을 향해 가는 중이므로, 종점까지의 이동 시간에서 편도 시간을 빼준 $y - x$가 답이 됩니다.

2B. SNUPTI

`string`, `implementation`

- 제출 136번, 정답 33명 (정답률 25.000%)
- 처음 푼 사람: **김성범**, 12분
- 출제자: `jhwest2`, `sait2000`

2B. SNUPTI

- 답이 YES라고 가정했을 때 정답으로 가능한 A_i 는 목록의 각 문자열에서 i 번째 문자를 모아 만든 집합으로 정해집니다.
  - 문자열의 i 번째 문자가 A_i 의 원소가 아니면 해당 문자열이 만들어지지 않습니다.
  - A_i 의 원소 중 문자열의 i 번째 문자로 나타나지 않는 문자가 있으면 목록에 없는 문자열이 만들어집니다.
- 따라서 이렇게 정해진 A_i 로부터 답이 YES가 될 수 있는지 확인해야 합니다.

2B. SNUPTI

- 답이 YES가 되기 위해서 확인해야 할 조건은 다음과 같습니다.
  - 각 척도의 결과에 사용되는 알파벳이 모두 서로 달라야 합니다.
  - 주어진 목록에 있는 각 문자열이 A_i 로부터 가능한 검사결과 문자열의 집합과 일대일 대응이 되어야 합니다.

2B. SNUPTI

각 척도의 결과에 사용되는 알파벳이 모두 서로 다름을 확인하기 위해서 $1 \leq i < j \leq N$ 에 대해 $A_i \cap A_j = \emptyset$ 임을 확인하면 됩니다.

2B. SNUPTI

- 일대일 대응을 확인하기 위해서 다음과 같은 두 조건을 확인하면 됩니다.
  - 주어진 목록에 중복이 없는지 확인합니다.
  - 가능한 검사결과의 개수가 주어진 목록의 문자열의 개수와 같은지 확인합니다.
- 중복이 없는지 확인하기 위해서 주어진 목록을 사전순으로 정렬하고 인접한 두 원소가 같은 경우가 없음을 확인하면 됩니다.
- 가능한 검사결과의 개수는 $|A_1| \times |A_2| \times \cdots \times |A_N|$ 으로 구할 수 있습니다.

2C. 3에 깃든 힘

`tree, greedy, dp`

- 제출 88번, 정답 19명 (정답률 21.591%)
- 처음 푼 사람: **김민재**, 25분
- 출제자: snuphy

2C. 3에 깃든 힘

- 정점이 $N = 3n$개 있는 트리를 크기 3인 서브트리들로 분해할 수 있는지 판별하고, 가능하다면 그 방법을 출력해야 합니다.

2C. 3에 깃든 힘

- 임의의 점을 루트로 잡고, 서브트리에서 문제를 푼다고 생각해봅시다.
- 만약 전체 트리에서 문제의 답이 참이라면, 해당 서브트리는 다음 3가지 상태 중 하나로 분류할 수 있습니다.
  a. 해당 서브트리가 분해 가능 : 서브트리 크기가 $3k$ 꼴
  b. 서브트리의 루트 정점을 제외했을 때 분해 가능 : 서브트리 크기가 $3k+1$ 꼴
  c. 서브트리의 루트 정점의 부모 정점을 추가했을 때 분해 가능 : 서브트리 크기가 $3k-1$ 꼴
- 앞으로 정점의 상태를 해당 정점을 루트로 하는 서브트리의 위의 분류에 해당하는 문자로 표현하고 분해 결과 나오는 크기 3의 서브트리를 3-트리라고 하겠습니다.

2C. 3에 깃든 힘

- 조금의 케이스 워크를 하면 어떤 정점에 대해 다음이 성립함을 알 수 있습니다.
  - 자식들 중 1개만 c이고 나머지는 a, 또는 자식들 중 2개만 b이고 나머지는 a일 때만 a
  - 자식들이 모두 a일 때만 b
  - 자식들 중 1개만 b이고 나머지는 a일 때만 c
  - 이외의 경우 분해 불가능
- 따라서 트리를 DFS로 순회하며 자식들의 상태를 알아낸 후 위의 수행을 반복하면 분해 가능성을 판별할 수 있습니다.
- 현재 순회하고 있는 정점이 3-트리에서 중간에 위치한 정점일 때 그 3-트리의 정점들을 뽑아 답에 추가한다면 분해 방법을 구할 수 있습니다.
- 시간복잡도는 DFS에 필요한 $\mathcal{O}(N)$ 입니다.

2C. 3에 깃든 힘

- 다른 방법으로 풀 수도 있습니다.
- 어떤 정점과 부모 정점이 서로 다른 서브트리로 분해되는 경우는 이 정점을 루트로 하는 서브트리의 크기가 $3k$ 꼴일 때밖에 없습니다. 즉, 서브트리의 크기가 3의 배수가 아닌 경우 무조건 부모 정점과 같은 3-트리로 분해됩니다.
- 따라서 리프부터 서브트리의 크기가 3이 아닌 정점을 부모 정점과 합치는 행동을 반복하여 새로 얻은 그래프에서 각 정점에 모인 정점의 개수가 모두 3일 때만 분해 가능합니다.
- `vector` 등으로 정점을 직접 옮기면서 크기가 4 이상이 되면 중지하는 방식 등으로 구현하면 역시 $\mathcal{O}(N)$에 해결할 수 있습니다.

2D. 이상한 프로그래밍 언어

`greedy`, `case_work`

- 제출 237번, 정답 18명 (정답률 7.595%)
- 처음 푼 사람: **김성범**, 42분
- 출제자: `kipa00`

2D. 이상한 프로그래밍 언어

- 변수 K와 이 변수에 적용할 연산이 $2N$개 주어집니다.
- 변수의 초기값은 K_0이며 연산은 다음 중 하나입니다.
  - +x: K가 $K + x$가 됩니다.
  - -x: K가 $\max(K - x, 0)$가 됩니다.
  - *x: K가 $x \cdot K$가 됩니다.
- 연산을 적용하기 전에, $(2i - 1)$번째 연산과 $2i$번째 연산 중 정확히 하나를 제거하고, 남은 N개의 연산을 **순서대로** 이어붙입니다.
  - 이렇게 제거하는 방법은 2^N가지 경우가 있습니다.
- 가능한 방법 중 모든 연산을 적용했을 때 K가 최대화되는 경우의 최종 변수의 값을 $10^9 + 7$로 나눈 나머지를 출력하는 문제입니다.

2D. 이상한 프로그래밍 언어

- 문제에서 정의한 모든 연산은 증가함수입니다. 즉, 문제에서 정의된 어떠한 연산 f 에 대해서도,

$$f(x) \leq f(y) \quad \text{if} \quad x \leq y$$

  가 성립합니다.
- 이는 매 순간 가장 큰 값이 되도록 하는 연산을 고르는 것이 최적의 방법이 된다는 것을 의미합니다!

2D. 이상한 프로그래밍 언어

- 따라서 아래 알고리즘:
  1. $K := K_0$.
  2. 두 연산 중에서 K의 값을 더 크게 만드는 쪽을 고르는 것을 N번 반복합니다.
  3. 최종 K의 값을 $10^9 + 7$로 나눈 나머지를 출력합니다.

  을 구현하면 **맞았습니다!!**를 받습니다...?

2D. 이상한 프로그래밍 언어

- 이 알고리즘은 놀랍게도 시간 초과를 받습니다. 왜일까요?
  - 연산을 i 번 반복하면 수의 길이가 $\mathcal{O}(i)$ 가 됩니다.
  - 덧셈, 뺄셈, 곱셈 등 연산을 시행해서 결과를 내어놓으려면, 최소한 입력을 읽어야 하므로 입력 크기만큼의 시간이 걸립니다.
  - 이 말은, 전체를 처리하는 데에

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$$

    이므로 $\mathcal{O}(N^2)$ 의 시간이 걸린다는 얘기입니다!
- 덧셈, 뺄셈, 곱셈이 항상 마법 같이 $\mathcal{O}(1)$ 시간에 일어나지는 않습니다.

- 그러면, 어차피 $10^9 + 7$로 나눈 나머지를 출력하면 되니까, 이렇게 수정한 알고리즘은 어떨까요?

    1. $K := K_0$.
    2. 두 연산 중에서 K의 값을 더 크게 만드는 쪽을 골라 연산하되, K에는 $10^9 + 7$로 나눈 **나머지를 저장하는 것**을 N번 반복합니다.
    3. 최종 K의 값을 출력합니다.

2D. 이상한 프로그래밍 언어

- 이번에는 틀렸습니다를 받습니다. 왜일까요?
  - 나머지만을 저장한다는 것은, 예를 들어 실제 K가 1인 경우와 $10^9 + 8$인 경우를 **같은 상태로** 저장한다는 것입니다.
  - 하지만 이 두 상태는 전혀 같지 않습니다!
    - 두 연산이 $*2$와 $+3$인 경우, $K = 10^9 + 8$이면 $*2$가 이득이지만 $K = 1$이면 $+3$이 이득입니다.
- K의 실제 값이 작은 경우가 문제인 것 같습니다.

2D. 이상한 프로그래밍 언어

K 의 실제 값이 작은 경우가 문제인 것 같습니다.

- 같은 연산끼리는, 빼기는 x가 작을수록, 더하기와 곱하기는 x가 클수록 이득인 것은 분명합니다.
- 빼기에 비해 더하기와 곱하기 (*0 제외) 가 이득인 것 역시 분명합니다.
- K 에 의존해서 다른 경우를 골라야 하는 상황은 더하기와 곱하기 중 하나를 골라야 할 때뿐입니다.
  - +x와 *y 중 하나를 골라야 한다고 합시다.
  - $K + x \geq yK$ 이려면 $(y-1)K \leq x$ 여야 합니다.
    - $y \leq 1$ 이라면 $yK \leq K \leq K + x$ 이므로 더하기를 고르는 것이 **무조건 이득**입니다.
    - $y \geq 2$ 라면 $K \leq \frac{x}{y-1} \leq 10^9$ 이이므로, 실제 K 가 10^9 이상일 때는 무조건 곱하기를 고르면 됩니다.

- K를 $10^9 + 7$로 나눈 나머지를 들고 있되, 실제 값이 10^9 이하인지 아닌지를 저장하는 `bool` 변수 b를 같이 들고 있읍시다.
- 이제 수정된 알고리즘은 다음과 같습니다.
  1. $K := K_0, b :=$ `true`.
  2. 다음을 N번 반복합니다:
     a. $b =$ `true`이면 두 연산 중에서 K의 값을 더 크게 만드는 쪽을 골라 연산합니다. 그렇지 않다면 직전 슬라이드의 설명대로 연산을 고릅니다.
     b. K의 값이 10^9 이상이라면 $b :=$ `false`.
     c. K에 K를 $10^9 + 7$로 나눈 나머지를 저장합니다.
  3. 최종 K의 값을 출력합니다.

2D. 이상한 프로그래밍 언어

- 여전히 틀렸습니다를 받습니다. 왜일까요?
  - 이번에는 연산을 연달아 적용하는 경우가 문제입니다.
  - 현재까지의 알고리즘에서 K의 실제 값이 $10^9 + 8$과 $2 \cdot 10^9 + 15$인 상태는 같은 상태입니다.
  - 하지만 이 두 상태는 연산을 여러 개 거치면 같지 않게 됩니다.
    - 10^9을 빼는 연산을 시행한 경우, K의 실제 값은 8과 $10^9 + 15$입니다.
    - 두 상태는 달라야 합니다!
- 뺄셈 연산이 문제가 되는 것 같습니다.

뺄셈 연산이 문제가 되는 것 같습니다.

- 뺄셈 연산 한 번에 최대 10^9 이 빠지고, 이 연산이 N 번 반복될 수 있습니다.
- 특정 순간에 $10^9 \cdot (N+1)$ 을 넘기면, 이후 실제 K 의 값마저도 뺄셈 연산으로 절대 10^9 아래로 떨어질 수가 없습니다.
- $B := 10^9 \cdot (N+1)$ 로 정의합니다.

2D. 이상한 프로그래밍 언어

- 기존에 10^9에서 끊었던 부분을 B까지 늘리면 될 거 같습니다.
- $B > 10^9 + 7$이라서 제대로 된 값을 출력하려면 조금 더 수정해야 합니다.
- 수정된 알고리즘은 다음과 같습니다.

1. $K := K_0, b := \texttt{true}$.
2. 다음을 N번 반복합니다:
    a. $b = \texttt{true}$이면 두 연산 중에서 K의 값을 더 크게 만드는 쪽을 골라 연산합니다. 그렇지 않다면 이전 슬라이드의 설명대로 연산을 고릅니다.
    b. K의 값이 B 이상이라면 $b := \texttt{false}$.
    c. $b = \texttt{false}$이면 K에 K를 $10^9 + 7$로 나눈 나머지를 저장합니다.
3. 최종 K의 값을 $10^9 + 7$로 나눈 나머지를 출력합니다.

2D. 이상한 프로그래밍 언어

- 이 정도 노력했는데 억울하게도 틀렸습니다를 받습니다. 왜일까요?
  - `*0 *0`이 문제가 됩니다.
  - 수가 얼마나 컸든 `*0`을 하면 실제 K의 값이 0이 되어버립니다.
- 이 부분을 수정하면 정말로 **맞았습니다!!**를 받을 수 있습니다.

전체 알고리즘

1. $K := K_0, b := \texttt{true}$.
2. 다음을 N 번 반복합니다:
   a. $b = \texttt{true}$ 이면 두 연산 중에서 K 의 값을 더 크게 만드는 쪽을 골라 연산합니다. 그렇지 않다면 이전 슬라이드의 설명대로 연산을 고릅니다.
   b. K 의 값이 B 이상이라면 $b := \texttt{false}$.
   c. $b = \texttt{false}$ 이면 K 에 K 를 $10^9 + 7$ 로 나눈 나머지를 저장합니다.
   d. 고른 연산이 `*0`이면 $b := \texttt{true}$.
3. 최종 K 의 값을 $10^9 + 7$ 로 나눈 나머지를 출력합니다.

2D. 이상한 프로그래밍 언어

- 시간 복잡도는 $\mathcal{O}(N)$ 입니다.
- Python과 같이 임의의 길이 정수를 언어에서 지원하는 언어가 아닌 경우, 곱셈의 오버플로우에 주의해야 합니다. $10^9 \cdot B$ 가 2^{64} 을 넘을 수 있습니다.
- 해결책 중 하나는 다음과 같습니다. x 와 y 를 곱하려 한다고 할 때,
  - 만일 `log2(x) + log2(y) >= 56`이면 $\log_2 B < 50$ 이므로 실수 오차를 고려하더라도 $xy > B$ 입니다.
    - x 와 y 를 $10^9 + 7$ 로 나눈 다음 곱합니다.
    - 나머지의 곱이 B 를 넘는지와 상관없이 실제 곱은 B 를 넘었습니다.
  - 그렇지 않다면, 실수 오차를 고려하더라도 $\log_2 xy \leq 62$ 이므로 두 수의 곱이 2^{62} 보다 작습니다.
    - 대부분의 언어에서 그냥 곱해서 처리해도 됩니다.

2E/1H. 자취방 정하기

`graph_theory`, `dijkstra`, `linearity_of_expectation`

- 제출 48번, 정답 17명 (정답률 35.417%)
- 처음 푼 사람: **신원석**, 50분
- 출제자: `jhwest2`

2E/1H. 자취방 정하기

- 어떤 정해진 이동을 하나 선택했다고 합시다.
- 이 이동에서 i 번째 간선을 c_i 번 사용한다고 했을 때, 기댓값의 선형성에 의해 이 이동에서 걸리는 시간의 기댓값은 $\sum_i c_i(a_i + b_i)/2$ 가 됩니다.
- 즉 각 간선의 가중치를 $(a_i + b_i)/2$ 로 설정한 그래프에서, 이동에서 선택한 간선의 가중치 합이 기댓값이 됩니다.

2E/1H. 자취방 정하기

- 주어진 그래프 G에서, 각 간선의 가중치를 $(a_i + b_i)/2$로 바꾼 그래프를 생각합시다.
- 각 정점에 대해 1번 정점으로 도달하는 가중치 합이 T 이하인 경로가 존재하는지 판별하는 문제가 됩니다.
- 모든 간선을 양방향으로 이용할 수 있으므로, 원래 경로를 뒤집어서 1번 정점에서 출발해 각 정점으로 도달하는 경로로 고려합시다.

2E/1H. 자취방 정하기

- 문제를 두 가지 케이스로 나눌 수 있습니다.
  - 도달 가능한 간선의 가중치가 모두 0 이상인 경우, 1 번 정점에서 출발하는 다익스트라 알고리즘으로 최단 경로를 구합니다. 이 최단 경로의 값이 T 이하인지 판별하면 됩니다.
  - 도달 가능한 간선 중 음수 가중치가 있는 경우, 해당 간선으로 이동해 해당 간선을 10^{100} 번 왔다 갔다 하면서 사용합니다. 그 다음에 원하는 정점으로 이동하면, 경로의 길이를 원하는 만큼 줄일 수 있습니다. 즉 도달 가능한 모든 정점이 정답이 됩니다.

2E/1H. 자취방 정하기

- DFS, BFS 등의 그래프 탐색 알고리즘을 이용해 음수 가중치 간선을 판별한 뒤, 없으면 다익스트라 알고리즘을 수행해서 문제를 해결할 수 있습니다.
- $\mathcal{O}(V^2)$ 시간에 작동하는 다익스트라 알고리즘은 너무 느립니다. Heap을 사용해서 $\mathcal{O}(E \log V)$ 시간에 작동하도록 해야 문제를 해결할 수 있습니다.
- 시간 복잡도는 $\mathcal{O}(V + E \log V)$ 입니다.

2F. 콜라 줍기

`dynamic_programming`, `prefix_sum`

- 제출 17번, 정답 5명 (정답률 35.294%)
- 처음 푼 사람: **김재윤**, 109분
- 출제자: `kipa00`, `cakelemon`

2F. 콜라 줍기

- $N \times N$ 격자의 $(1, 1)$과 (N, N)을 제외한 모든 격자칸 위에 두 개의 수 a_{ij}와 b_{ij}가 주어집니다.
- $(1, 1)$에서 (N, N)으로 이동하는 두 개의 경로를 찾을 것인데, 다음과 같은 조건을 만족해야 합니다.
  - 두 경로 모두 $(1, 1)$에서 (N, N)으로 이동하는 최단 경로여야 합니다.
  - 두 경로는 $(1, 1)$과 (N, N)을 제외하고 겹치는 칸이 있으면 안 됩니다.
- 한 경로에서는 지나간 칸의 a_{ij}의 값을, 다른 경로에서는 지나간 칸의 b_{ij}의 값을 더합니다.
- 두 값의 합을 최대화하면?

2F. 콜라 줍기

- 현재까지 그린 두 경로가 항상 서로 **같은** 행에 있다고 하고, 다음과 같이 정의합니다.

$$D[i][j][k] = \text{현재 } a_{ij} \text{ 경로가 } (i,j) \text{에, } b_{ik} \text{ 경로가 } (i,k) \text{에 있을 때 최댓값}$$

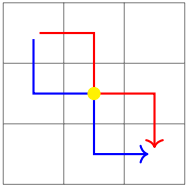
- 일반성을 잃지 않고 $j < k$라 합시다.
  - 이렇게 한 번, 표 전체를 대각선으로 대칭이동한 뒤 한 번 더 값을 구해 주면 답을 얻을 수 있습니다.

2F. 콜라 줍기

$D[i][j][k] =$ 현재 a_{ij} 경로가 (i, j)에, b_{ik} 경로가 (i, k)에 있을 때 최댓값

- 그러면 $D[i][j][k]$로 도달하는 방법은 세 가지가 있습니다.
  - a_{ij}를 더하는 경로가 $(i, j-1)$에서 오른쪽으로 이동 (단 $k \neq j-1$)
  - b_{ik}를 더하는 경로가 $(i, k-1)$에서 오른쪽으로 이동 (단 $j \neq k-1$)
  - 두 경로 모두 각각 $(i-1, j)$와 $(i-1, k)$에서 (i, j)와 (i, k)로 이동
- 따라서 이 세 값 중 가장 큰 값을 골라 주면 D를 채울 수 있습니다...?

- **틀렸습니다!** 왜 틀렸을까요?

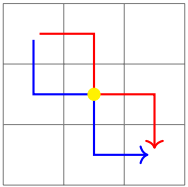


$N = 3$인 경우 이런 경로를 고려하게 됩니다.

- 오른쪽으로 각각 이동하게 되는 경우, D가 오른쪽으로 이동해서 방문한 칸을 기억하지 못합니다.
- 따라서 $D[2][1][2] \rightarrow D[2][1][3] \rightarrow D[2][2][3]$과 같이 값을 잘못 구하게 됩니다!

2F. 콜라 줍기

- 해결책 1: 왼쪽의 경로를 모두 이동한 후, 오른쪽의 경로를 이동합니다.



  - 현재 문제는 붉은 경로상에서 $(2,2) \to (2,3)$ 이동이 일어났지만, 이를 기억하지 못하고 푸른 경로상에서 $(2,1) \to (2,2)$ 이동이 일어나는 것입니다.
  - 이를 방지하기 위해서는, 한 행에서 D 값을 갱신할 때 우선 푸른 경로의 이동으로 인한 D 값 변화를 모두 적용하고, 이후 붉은 경로의 이동으로 인한 D 값 변화를 적용하면 됩니다.

- 해결책 2: $D[i-1][*][*]$ 에서 $D[i][*][*]$ 로 **한 번에** 전이합니다.
  - 이 경우, 지금 j 가 이전 k 를 넘지 못하도록 **점화식에서** 강제할 수 있기 때문에 도움이 됩니다.
  - $D[i-1][\alpha][\beta]$ 중 $D[i][j][k]$ 로 도달할 수 있는 칸은 $1 \le \alpha \le j < \beta \le k$ 를 만족해야 합니다.
  - 따라서 점화식을 다음과 같이 세울 수 있습니다.

$$D[i][j][k] = \max_{1 \le \alpha \le j < \beta \le k} \left(D[i-1][\alpha][\beta] + \sum_{c=\alpha}^{j} a_{ic} + \sum_{c=\beta}^{k} b_{ic} \right)$$

  - 이걸 naïve하게 계산하면 $\mathcal{O}(N^6)$ 이 걸립니다...!

2F. 콜라 줍기

$$D[i][j][k] = \max_{1 \le \alpha \le j < \beta \le k} \left(D[i-1][\alpha][\beta] + \sum_{c=\alpha}^{j} a_{ic} + \sum_{c=\beta}^{k} b_{ic} \right)$$

- 식이 다소 무서워 보이지만 차근차근 뜯어 봅시다.
  - 먼저 시그마 식을 최적화해 봅시다. 다음과 같은 prefix sum 배열을 정의합니다.

$$A_{ij} := \sum_{c=1}^{j} a_{ic} = A_{i(j-1)} + a_{ij}, \quad B_{ij} := \sum_{c=1}^{j} b_{ic} = B_{i(j-1)} + b_{ij}$$

  - 이 식은 $\mathcal{O}(N^2)$에 계산할 수 있습니다.

2F. 콜라 줍기

$$D[i][j][k] = \max_{1 \le \alpha \le j < \beta \le k} \left(D[i-1][\alpha][\beta] + \sum_{c=\alpha}^{j} a_{ic} + \sum_{c=\beta}^{k} b_{ic} \right)$$

$$A_{ij} := \sum_{c=1}^{j} a_{ic} = A_{i(j-1)} + a_{ij}, \quad B_{ij} := \sum_{c=1}^{j} b_{ic} = B_{i(j-1)} + b_{ij}$$

- 그러면 시그마 식 안쪽을 prefix sum을 이용해 정리할 수 있습니다.

$$\begin{aligned} D[i][j][k] &= \max_{1 \le \alpha \le j < \beta \le k} \left(D[i-1][\alpha][\beta] + \sum_{c=\alpha}^{j} a_{ic} + \sum_{c=\beta}^{k} b_{ic} \right) \\ &= \max_{1 \le \alpha \le j < \beta \le k} \left(D[i-1][\alpha][\beta] + A_{ij} - A_{i(\alpha-1)} + B_{ik} - B_{i(\beta-1)} \right) \end{aligned}$$

2F. 콜라 줍기

$$D[i][j][k] = \max_{1 \le \alpha \le j < \beta \le k} \left(D[i-1][\alpha][\beta] + A_{ij} - A_{i(\alpha-1)} + B_{ik} - B_{i(\beta-1)} \right)$$

- 이제 $\mathcal{O}(N^5)$ 이 되었습니다...!
- max 와 관련 없는 항을 밖으로 뺀 뒤 더 관찰해 봅시다.

$$D[i][j][k] = \max_{1 \le \alpha \le j < \beta \le k} \left(D[i-1][\alpha][\beta] - A_{i(\alpha-1)} - B_{i(\beta-1)} \right) + A_{ij} + B_{ik}$$

- max 안에 남은 항들이 j 및 k 와는 전혀 관련이 없어 보입니다...! 다음과 같이 정의합시다.

$$C[i][\alpha][\beta] := D[i-1][\alpha][\beta] - A_{i(\alpha-1)} - B_{i(\beta-1)}$$

2F. 콜라 줍기

$$D[i][j][k] = \max_{1 \le \alpha \le j < \beta \le k} C[i][\alpha][\beta] + A_{ij} + B_{ik}$$

- $D[i][j][k-1]$에 대해서 식을 써 보면 다음과 같이 됩니다.

$$D[i][j][k-1] - A_{ij} - B_{i(k-1)} = \max_{1 \le \alpha \le j < \beta \le k-1} C[i][\alpha][\beta]$$

- 이 식의 max는 $D[i][j][k]$의 경우에서 정확히 $\beta = k$인 경우를 제외하고 구한 값이 됩니다...!
- 따라서 다음과 같이 정리할 수 있습니다.

$$\begin{aligned} D[i][j][k] &= \max(D[i][j][k-1] - A_{ij} - B_{i(k-1)}, \max_{1 \le \alpha \le j} C[i][\alpha][k]) + A_{ij} + B_{ik} \\ &= \max(D[i][j][k-1] + b_{ik}, \max_{1 \le \alpha \le j} C[i][\alpha][k] + A_{ij} + B_{ik}) \end{aligned}$$

2F. 콜라 줍기

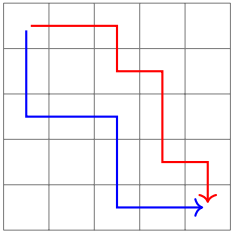
$$D[i][j][k] = \max(D[i][j][k-1] + b_{ik}, \max_{1 \le \alpha \le j} C[i][\alpha][k] + A_{ij} + B_{ik})$$

- 이 식은 $\mathcal{O}(N^4)$ 이며, 빠른 계산을 가로막는 것은 $\max_{1 \le \alpha \le j} C[i][\alpha][k]$ 입니다. 거의 다 왔습니다...!
- $F[i][j][k] := \max_{1 \le \alpha \le j} C[i][\alpha][k] = \max(F[i][j-1][k], C[i][j][k])$ 로 정의하면 max 를 F 로 대체할 수 있습니다.
- 이 경우 D 를 계산하는 데에는 F 가 계산되어 있다고 할 때 전체 $\mathcal{O}(N^3)$, F 를 계산하는 데 전체 $\mathcal{O}(N^3)$ 이므로 전체 $\mathcal{O}(N^3)$ 에 문제를 해결할 수 있습니다!

2F. 콜라 줍기

- 주어진 정의를 전부 그대로 배열로 정의해서 내면 **메모리 초과**가 일어납니다.
- 메모리 사용량을 줄여 봅니다.
  - A와 B를 저장하는 메모리는 $\mathcal{O}(N^2)$이므로 줄일 필요가 없습니다.
  - D는 반드시 저장해야 합니다.
  - C는 D, A, B가 있다면 계산하는 데 상수 시간이 들어가므로, 따로 저장할 필요가 없습니다.
  - F는 $F[i][*][*]$에 대한 값을 $D[i][*][*]$에서만 요구하므로, $D[i][*][*]$의 계산이 끝났다면 따로 저장할 필요가 없습니다.
    - $F[i][*][*]$을 저장하는 **2차원 배열**을 만들면 F에 대한 메모리 사용량을 $\mathcal{O}(N^2)$으로 줄일 수 있습니다.
- 이때 메모리 사용량은 $8 \times (400^3 + 3 \times 400^2) = 515.84 \times 10^6$이므로, 메모리 제한 안에 넉넉히 들어갑니다.

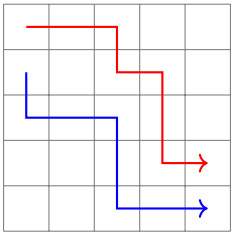
- 해결책 3: 관찰을 더 합니다.
- 경로에서 겹치는 칸이 없다는 것은 어떤 경로는 다른 경로보다 "위에" 있다는 말입니다.



$N = 5$인 경우의 예시.

2F. 콜라 줍기

- 그렇다면, 그림에서 빨간색 경로는 무조건 오른쪽으로 시작해서 아래로 끝나고, 파란색 경로는 무조건 아래로 시작해서 오른쪽으로 끝납니다.
- 경로 자체를 약간 다른 곳에서 시작하거나 끝나도록 바꿀 수 있습니다.



이렇게 바꾸었을 때 내려가는 횟수에 주목합시다.

2F. 콜라 줍기

- 서로의 경로에서, 마지막 칸에 도달하기까지 내려가는 횟수가 **같습니다**.
- 이런 상황에서 현재까지 그린 두 경로의 **내려간 횟수**가 i로 같다고 하고 식을 정의해 봅시다.

$$D[i][j][k] := \text{현재 } a_{ij} \text{ 경로가 } (i+\mathbf{2}, j) \text{에, } b_{ij} \text{ 경로가 } (i+\mathbf{1}, k) \text{에 있을 때 최댓값}$$

2F. 콜라 줍기

$D[i][j][k] :=$ 현재 a_{ij} 경로가 $(i+2, j)$에, b_{ij} 경로가 $(i+1, k)$에 있을 때 최댓값

- 이러면 어떤 점이 편할까요?
  - D의 두 경로가 서로 같은 행을 공유하지 않으므로, 이제 $j < k$를 지키는 한 두 경로가 마음대로 오른쪽으로 이동할 수 있습니다!
  - 아래로 동시에 내려갈 수 있는 제약 조건 역시 $j < k$입니다!
- 따라서 다음과 같은 비교적 간단한 식을 통해 계산할 수 있습니다.

$$\begin{aligned} D[i][j][k] = \max(&D[i-1][j][k] + a_{(i+2)j} + b_{(i+1)k}, \\ &D[i][j-1][k] + a_{(i+2)j} \text{ if } j > 1, \\ &D[i][j][k-1] + b_{(i+1)k} \text{ if } j < k-1) \end{aligned}$$

2F. 콜라 줍기

- 어떻게 계산하든, $\mathcal{O}(n^3)$에 문제를 해결할 수 있습니다.
- 역추적을 할 필요가 없기 때문에, $D[i][*][*]$를 계산할 때 $D[i-1][*][*]$만 남기는 방식으로 메모리 사용량을 $\mathcal{O}(n^2)$으로 줄일 수도 있습니다.

2G. 슬라임 키우기

`data_structure, ad_hoc`

- 제출 69번, 정답 11명 (정답률 15.942%)
- 처음 푼 사람: **신원석**, 93분
- 출제자: TAMREF

2G. 슬라임 키우기

- 수열 $a_1, \cdots, a_N$ 이 주어지고, 다음과 같은 형태의 연산을 총 Q 번 적용합니다.
  - 값이 x 이하인 수에 모두 y 를 곱한다. $(0 \leq x, y \leq 10^9)$
- 모든 연산을 적용한 후, 수열을 오름차순으로 정렬해 나타내는 문제입니다.
- 단순한 방법으로 $\mathcal{O}(QN)$ 시간에 구현할 수 있지만, 제한 시간에 통과하기 위해선 더 짧은 시간의 풀이를 생각해야 합니다.

2G. 슬라임 키우기

- $y = 0$ 혹은 $y = 1$인 경우는 처리하기 쉽습니다.
  - $y = 1$이라면 무시하면 되고, $y = 0$이면 $a_i \le x$였던 a_i가 0이 되므로, 그 a_i는 더 이상 고려하지 않아도 됩니다.
- $y \ge 2$인 경우, $x \le 10^9 \le 2^{30}$라는 사실에 주목합시다.
  - $a_i = 1$이고, $x = 10^9$, $y = 2$인 경우 이 a_i가 가장 천천히 증가하게 되는데, 그렇다 하더라도 30번만에 a_i가 10^9 이상으로 커지게 됩니다.

- 따라서 다음과 같은 방법으로 해결할 수 있습니다.
  1. 1 이상 10^9 이하인 수들을 우선 순위 큐 등을 비롯한 자료구조로 정렬된 상태를 유지합니다.
  2. 이후 단순하게 주어진 연산을 처리합니다. 가장 작은 수를 빼내고, 연산의 결과가 1 이상 10^9 이하일 때만 다시 자료 구조에 넣어줍니다.
- 자료구조의 삽입/삭제 연산 횟수는 많아야 $N \log_2(10^9) \sim 30N$ 번이므로, 우선 순위 큐를 사용하여 시간 복잡도 $\mathcal{O}(30N \log N)$ 에 문제를 해결할 수 있습니다.

2H. 포켓몬 대회

`bitwise_operations`, DP

- 제출 13번, 정답 0명 (정답률 0.000%)
- 처음 푼 사람: -, -분
- 출제자: `cakelemon`

2H. 포켓몬 대회

- 길이가 같은 두 증가수열 $a_1, \cdots, a_n$ 과 $b_1, \cdots, b_n$ 에 대해, 모든 $1 \leq i \leq n$ 에 대해 $a_i \leq b_i$ 가 만족되면 수열 a 는 수열 b 의 하위호환이라고 정의합니다.
- 1 부터 K 사이의 숫자만 사용하는 증가수열 N 개를 원소로 갖는 multiset S 가 주어질 때, multiset S 의 각 원소 s_i 에 대해 S 의 원소 중 s_i 의 하위 호환인 것의 수를 구해야 합니다.
- $N \leq 100\,000$ 이 크고 $K \leq 17$ 가 작은 제한으로 출제되었습니다.

2H. 포켓몬 대회

- 각 원소마다 집합 S를 순회하며 하위 호환인 것의 수를 세면 문제를 시간 복잡도 $\mathcal{O}(N^2K)$에 구현할 수 있습니다.
- 그러나 시간 제한을 만족하기 위해서는, 이보다는 빠른 방법을 찾아야 합니다.

2H. 포켓몬 대회

- 증가수열이 포함할 수 있는 수의 최대 크기를 K라고 합시다.
- $K \leq 17$이고 증가수열은 중복 원소를 가지지 않으므로, 다음과 같은 방식으로 증가수열 하나를 이진수 하나로 표현할 수 있습니다.

  증가수열 $a_1, \cdots, a_n$이 k를 포함한다면 오른쪽에서 k번째 비트는 *1*이며, 아니면 *0*이다.

2H. 포켓몬 대회

- 이제 두 수 a, b가 각각 어떠한 수열의 이진수 표현이라고 할 때, a의 비트 중 1인 것들을 오른쪽으로 옮겨서 b를 만들 수 있다면 b가 표현하는 수열은 a가 표현하는 수열의 하위 호환이라고 생각할 수 있습니다.
- 따라서 이제 문제는 $2^{17} - 1 = 131\,071$ 이하의 음이 아닌 정수를 N개 포함한 multiset S가 주어질 때, 각 원소에 대해 1인 비트를 오른쪽으로 움직여서 만들 수 있는 S의 원소 개수를 구하는 것이 됩니다.

2H. 포켓몬 대회

- DP를 활용하기 위해 아래와 같은 부분문제를 생각해 봅시다.

$$D[k][n] = [n\text{의 오른쪽에서 } k\text{번째 1까지의 비트의 값을 변경할 수 } \textbf{없을} \text{ 때,}\ n\text{의 비트들을 이동하여 만들 수 있는 } S\text{의 원소 개수}]$$

- 이때, 아래 두 가지 사실이 성립합니다. $\mathrm{pc}(n)$은 n에 있는 1인 비트의 개수입니다.
  - $D[\mathrm{pc}(n)][n] = (n\text{이 } S\text{에 포함된 횟수})$
  - $D[0][n] = (S\text{의 원소 중 수열 } n\text{의 하위호환인 것의 개수})$
- 따라서 충분히 빠르게 $D[k][n]$들을 계산해낼 수 있다면 문제를 해결할 수 있습니다.

2H. 포켓몬 대회

- n 의 오른쪽에서 k 번째 1인 비트를 bit_k 라고 하겠습니다.
- 오른쪽에서 bit_{k-1} 까지의 비트만 바꿀 수 없는 경우가 오른쪽에서 bit_k 까지의 비트를 바꿀 수 없는 경우에 비해 더 할 수 있는 일을 생각해 봅시다.
  - 두 경우의 차이는 bit_k 를 바꿀 수 있게 되었다는 점입니다.
- 따라서 bit_k 가 bit_{k-1} 과 **붙어 있다면** (bit_k 는 어차피 움직일 수 없으므로)
  $D[k-1][n] = D[k][n]$ 입니다.

2H. 포켓몬 대회

- 이제 bit_k 가 bit_{k-1} 과 **붙어 있지 않은** 경우를 생각해 봅시다.
- bit_k 는 오른쪽으로 최소 한 칸은 움직일 수 있습니다.
- n 의 하위 호환인 수들 중 bit_k 가 움직이지 않은 경우의 수는 앞선 설명처럼 $D[k-1][n]$ 입니다.
- n 의 하위 호환인 수들 중 bit_k 가 한 칸이라도 움직였다면,
  - 이 수들은 모두

$$f_k(n) := (n\text{에서 }\mathrm{bit}_k\text{를 한 칸 오른쪽으로 움직인 수})$$

    의 하위 호환입니다.
  - n 에서 bit_k 가 움직이지 않은 경우가 $f_k(n)$ 의 하위 호환일 일은 없습니다.
- 따라서 이 경우 $D[k-1][n] = D[k][n] + D[k-1][f_k(n)]$ 이 됩니다.

- 위에서는 $\mathrm{pc}(n) \geq 2$라고 가정했는데, $\mathrm{pc}(n) = 1$이면 비교적 간단한 base case 전처리가 가능합니다.
  - 식을 생각해 보면 $DP[0][1] = DP[1][1]$이고, $DP[0][n] = DP[1][n] + DP[0][n/2]$임을 알 수 있습니다.
- 위 사항까지 고려해서 점화식을 정리하면 다음과 같습니다.
  - $n = 1$이거나, $\mathrm{pc}(n) \geq 2$이고 bit_k와 bit_{k-1}이 붙어 있는 경우 $D[k-1][n] = D[k][n]$
  - 그렇지 않은 경우 $D[k-1][n] = D[k][n] + D[k-1][f_k(n)]$
- k를 **내림차순**으로, n을 오름차순으로 순회하여 시간복잡도 $\mathcal{O}(N + K \cdot 2^K)$에 모든 식을 계산할 수 있습니다.

스코어보드 보러 갑시다!

This Page is Intentionally Left Blank

1A. 사회적 거리두기

`geometry`, `sweeping`

- 제출 12번, 정답 2명 (정답률 16.667%)
- 처음 푼 사람: **김동현**, 220분
- 출제자: `IHHI`

1A. 사회적 거리두기

- 이 문제는 택시기하에서, N 개의 점이 주어지면 모든 점과의 거리의 최솟값이 K 가 되도록 하기 위해서 이동해야 하는 거리의 최솟값을 구하는 문제입니다.

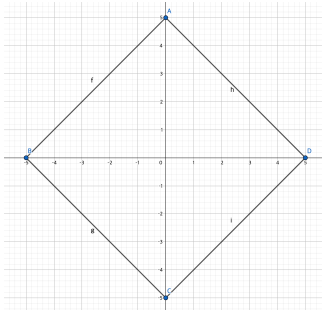
1A. 사회적 거리두기

- 우선 두 가지 경우를 고려해봅시다.
  - 민준이와 친구들의 거리의 최솟값이 K 보다 큰 경우
  - 민준이와 친구들의 거리의 최솟값이 K 이하인 경우
- 첫 번째 경우에 대해서는 쉽게 문제를 해결할 수 있습니다.
- 현재 가장 가까운 친구와의 거리를 D 라고 한다면, 그 친구 방향으로 $(D-K)$ 만큼 이동하게 되면 그 친구와의 거리가 K 가 되고, 다른 친구들과도 거리가 최대 $(D-K)$ 만큼 줄어들기 때문에, 거리의 최솟값이 K 가 됩니다.
- 미리 모든 친구들과의 거리를 계산하고, 거리의 최솟값을 구한 뒤, 이 경우에 대해서는 바로 답을 출력해주면 됩니다.

- ~~민준이와 친구들의 거리의 최솟값이 K 보다 큰 경우~~
- 민준이와 친구들의 거리의 최솟값이 K 이하인 경우

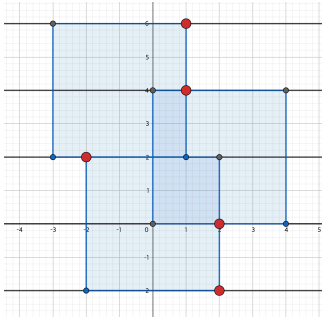
- 이제 두번째 경우가 남습니다.
- 목표를 약간 수정해서, 친구들과의 거리의 최솟값이 K 가 되는 위치를 구하는 것이 아니라 K **이상인** 위치를 구해도 됩니다.
  - 민준이와 친구들 사이의 거리의 최솟값이 K 이하이기에, 거리의 최솟값이 K 보다 큰 위치로 가기 위해서는, K 인 위치를 거쳐가야 하기 때문입니다.

1A. 사회적 거리두기



- 그러면, 거리의 최솟값이 K 이상인 위치의 경계를 그려보면 이는 위와 같은 45 도 기울어진 정사각형 형태로 나타납니다.
- 이를 45 도 돌려서 생각해주면, 좌표축에 평행한 정사각형이 되고, 이러한 영역의 합은 정사각형들의 합집합이 됩니다.

1A. 사회적 거리두기



- 이러한 정사각형들의 합집합에서 원점에서 거리가 최소가 되는 정점을 구해야 합니다. 이때 모든 y 좌표에서 영역에 속하지 않는 점들 중 x 좌표의 절대값이 가장 작은 점들이 최단 거리가 되는 후보가 될 것입니다.
- 이는 sweeping으로 해결할 수 있고, segment tree 등의 자료구조를 이용해서 $\mathcal{O}(N \log N)$ 시간에 해결할 수 있습니다.

1B. 자리 바꾸기

`permutation_cycle_decomposition`

- 제출 92번, 정답 25명 (정답률 28.261%)
- 처음 푼 사람: **조승현**, 4분
- 출제자: `cubelover`

1B. 자리 바꾸기

- 1 부터 N 까지의 수로 이루어진 순열 A 가 하나 주어집니다. i 번째 원소를 A_i 번째로 이동시키는 연산을 M 번 반복했을 때, 결과로 나타나는 순열을 구하는 문제입니다.
- 문제를 다시 state하면, 순열 $A : \{1, \cdots, N\} \to \{1, \cdots, N\}$ 이 주어질 때, $A^M(1), \ldots, A^M(N)$ 의 값을 구하는 문제가 됩니다.
- M 의 값이 $10^{200\,000}$ 으로 매우 크다는 점에 유의해야 합니다.

1B. 자리 바꾸기

- 순열-사이클 분할에 의해, 주어진 순열은 특정 주기로 원래 위치로 돌아오는 여러 개의 사이클들로 분해됩니다.
  - 예를 들어 주어진 순열이 $[3, 4, 1, 5, 2]$ 인 경우, 3과 1이 주기 2인 사이클을 이루고 $2, 4, 5$가 주기 3인 사이클을 이루게 됩니다.
- 사이클들로 순열을 분할하고 나면, 주기가 T인 사이클에 대해서는 $M \bmod T$번 만큼만 연산을 수행해보면 됩니다.
- 각 원소 i에 대해서 해당 원소가 속하는 사이클의 크기를 구합니다. 이 값이 T_i라고 할 때, $A^{M \bmod T_i}(i)$의 값을 계산해주면 됩니다.
- $M \bmod T_i$의 값을 $\mathcal{O}(\log M)$에 구하면, 위 값은 i가 속한 사이클의 $M \bmod T_i$칸 뒤에 있는 원소가 되므로 $\mathcal{O}(1)$ 시간에 구할 수 있습니다. 종합하면 $\mathcal{O}(N \log M + N)$ 시간에 작동하는 풀이를 얻습니다.

1B. 자리 바꾸기

- 이 풀이의 병목은 $M \bmod T_i$ 의 값을 구하는 과정에 있습니다.
- T_i 가 같은 원소에 대해서는 굳이 위 값을 계산해줄 필요가 없으므로, 서로 다른 T_i 값에 대해서만 계산해줍시다.
- 각 사이클에 대해서 주기의 합이 N 이므로, 서로 다른 주기의 개수는 많아야 $\mathcal{O}(\sqrt{N})$ 개뿐입니다. 따라서 $M \bmod T_i$ 를 $\mathcal{O}(\sqrt{N})$ 번만 계산해줘도 됩니다!
- 종합하면 전체 문제를 $\mathcal{O}(\sqrt{N} \log M + N)$ 시간에 해결할 수 있습니다.

1C. 이름 부르기

`combinatorics`, `fft`, `inclusion_and_exclusion`

- 제출 5번, 정답 0명 (정답률 0.000%)
- 처음 푼 사람: -, -분
- 출제자: snuphy

1C. 이름 부르기

- $N \times M$ 크기의 격자에 사람들이 앉아있습니다. ($1 \leq NM \leq 5 \times 10^5$)
- 사람들을 부르는 순서로 가능한 $(NM)!$ 가지 경우 중 앞뒤로 앉아있는 사람이 연달아 불리지 않는 경우의 수를 구해야 합니다.

1C. 이름 부르기

- 우선 1열만 있을 때를 생각해봅시다.
- 여기서 앞뒤로 앉아있는 사람 쌍이 연속해서 불리는 묶음이 k 개 이상인 순열의 수를 셀 수 있다고 한다면, 포함-배제의 원리를 통해 우리가 원하는 답을 구할 수 있게 됩니다.
- **한** 명 이상의 연달아 앉아있는 사람들의 묶음을 **블록**이라고 정의합시다.
  - 한 블록의 사람이 **두 명 이상**일 경우 앞에서부터 연속한 것과 뒤에서부터 연속한 것은 서로 다른 것으로 생각합시다. 예를 들어 $[1, 2, 3]$과 $[3, 2, 1]$은 다른 블록입니다.

1C. 이름 부르기

- n 명의 사람을 $(n-k)$ 개의 블록으로 분할하는 경우의 수를 $A_{n,k}$ 라고 합시다.
  - k 개 이상의 인접한 사람 쌍이 존재하는 순열의 수는 $(n-k)! \times A_{n,k}$ 가 됩니다.
  - 문제의 답은 포함-배제의 원리에 의해

$$\sum_{k=0}^{n-1}(-1)^k(n-k)!A_{n,k}$$

    가 됩니다.
- 그렇다면 $A_{n,k}$ 는 어떻게 구할 수 있을까요?

처음 구상했던 풀이는 이 부분을 $\mathcal{O}(n^2)$에 푸는 것이었습니다.

- $(n-k)$개의 블록 중 크기가 2 이상인 블록의 개수를 r이라고 하고 경우의 수를 구해봅시다.
  - 블록들은 연속한 수들의 묶음이므로 숫자들 사이를 자른다고 생각하면 블록들에 순서를 줄 수 있습니다. 이 $(n-k)$개의 블록 중 크기가 2 이상인 블록 r개를 고르는 경우는 ${}_{n-k}C_r$입니다.
  - 이후 각 r개 블록의 크기를 정하는 경우를 세어 봅시다. 블록의 크기를 정하는 것은 k개의 원소를 순서가 있는 r개의 각 집합에 적어도 1개를 할당하는 것으로 생각할 수 있고 이 경우의 수는 ${}_rH_{k-r} = {}_{k-1}C_{k-r}$입니다.

- 여기서 각 블록의 연속한 방향까지 고려해서 2^r 을 곱해준 ${}_{n-k}C_r \cdot {}_{k-1}C_{k-r} \cdot 2^r$ 이 우리가 원하는 경우의 수가 됩니다. 즉,

$$A_{n,k} = \sum_{r=1}^{\min\{k,n-k\}} {}_{n-k}C_r \cdot {}_{k-1}C_{k-r} \cdot 2^r.$$

- $A_{n,k}$ 들을 조금 더 빠르게 구할 수는 없을까요? $P_n(x) = \sum_{k=0}^{n-1} A_{n,k} x^k$ 로 정의합시다. 우리는 결국 다항식 $P_n(x)$ 의 계수를 빠르게 알아내면 됩니다.

1C. 이름 부르기

- 다음의 3 가지로 나누어 생각하면 P_n 를 P_{n-1} 과 P_{n-2} 로 표현할 수 있습니다. 편의상 i 번째 사람을 i 로 치환하여 표시하겠습니다.
  - $(n-1)$ 명을 적당히 나눈 상태에서 n 을 단독으로 추가: P_{n-1}
  - $(n-1)$ 명을 적당히 나눈 상태에서 $(n-1)$ 이 크기 2 이상의 블록에 속한다면 연속성을 잃지 않도록 $(n-1)$ 뒤에 추가, 그렇지 않다면 $[n-1, n]$ 이 하나의 블록이 되도록 추가: xP_{n-1}
  - $(n-2)$ 명을 적당히 나눈 상태에서 $[n, n-1]$ 을 추가: xP_{n-2}
- 이를 점화식으로 표현하면 $P_n = (1+x)P_{n-1} + xP_{n-2}$ 입니다.

- 이 점화식을 행렬로 표현할 수도 있습니다.

$$\begin{bmatrix} P_n \\ P_{n-1} \end{bmatrix} = \begin{bmatrix} 1+x & x \\ 1 & 0 \end{bmatrix} \begin{bmatrix} P_{n-1} \\ P_{n-2} \end{bmatrix}$$

- 이를 통해 P_N 을 다음과 같이 표현할 수 있습니다. 이때 $P_1 = 1, P_2 = 1 + 2x$ 입니다.

$$\begin{bmatrix} P_N \\ P_{N-1} \end{bmatrix} = \begin{bmatrix} 1+x & x \\ 1 & 0 \end{bmatrix}^{N-2} \begin{bmatrix} P_2 \\ P_1 \end{bmatrix}$$

- 이제 다항식 성분을 가진 행렬의 거듭제곱을 빠른 거듭제곱으로 구하면 됩니다.
- 다항식을 곱할 때 FFT를 이용한다면, $\mathcal{O}(N \log N)$의 시간복잡도로 $A_{N,k}$를 구할 수 있게 됩니다.
- 이제 본 문제인 M열의 경우에 대해 풀어봅시다. 편의상 $A_k = A_{N,k}$라 하겠습니다. 1열인 경우를 풀 때와 비슷한 철학을 이용해 푼다면, 답이 다음과 같음은 쉽게 알 수 있습니다.

$$\text{ans} = \sum_{k=0}^{(N-1)M} (-1)^k \left(\sum_{1 \le i \le M, 0 \le k_i < N, \sum k_i = k} \prod A_{k_i} \right) (NM-k)!$$

1C. 이름 부르기

- 가운데 괄호친 부분은 P_N 을 M 제곱한 다항식의 계수들과 정확히 일치합니다. 이 또한 FFT로 구해주면 최종적으로 $\mathcal{O}(NM \log (NM))$ 에 문제를 풀 수 있습니다.
- 여담으로 다항식 행렬을 거듭제곱할 때나 다항식을 거듭제곱할 때 convolution한 상태에서 모두 계산하고 다시 convolution하는 형태로 구현하면 시간복잡도는 변하지 않으나 convolution 횟수를 획기적으로 줄일 수 있어 상당히 빨라집니다.
- 이외에도 다항식 거듭제곱 시 특수한 형태의 행렬 거듭제곱임을 이용하면 곱셈 횟수를 대략 절반 이하로 줄일 수 있습니다.

1D. 정렬 프로그램

`sorting`, `math`, `combinatorics`, `sos_dp`

- 제출 5번, 정답 2명 (정답률 40.000%)
- 처음 푼 사람: **조승현**, 101분
- 출제자: `jhwest2`

1D. 정렬 프로그램

- 어떤 수열이 정렬되는지의 여부는 수열을 구성하는 수들의 대소 관계에만 달려있습니다.
- 즉 수열이 1 부터 N 까지의 수들로만 이루어진 경우에 문제를 해결한 뒤, 조합론을 이용해 범위를 M 까지로 확장했을 때의 경우의 수를 구할 수 있습니다.
- 서로 다른 수의 개수가 K 개일 때의 답을 ans_K 라고 합시다. 이때 ans_K 에 $\binom{M}{K}$ 를 곱해서 더한 것이 문제의 답이 됩니다.

1D. 정렬 프로그램

- 그러나 N^N 가지의 가능한 모든 수열을 다 정렬해보는 것은 여전히 매우 오래 걸리는 작업입니다. 개선이 필요합니다.
- 어떤 수열이 제대로 정렬된다는 것을 다음과 같이 생각할 수 있습니다. 수열의 서로 다른 수의 개수가 K 개일 때,
  - 가장 작은 수들이 모두 맨 앞으로 이동합니다.
  - 가장 작은 2 개의 수들이 모두 맨 앞으로 이동합니다.
  - …
  - 가장 작은 $K-1$ 개의 수들이 모두 맨 앞으로 이동합니다.

1D. 정렬 프로그램

- 가장 작은 k 개의 수들을 0 으로 바꾸고, 나머지 수들을 1 로 바꿔서 얻은 수열을 S_k 라고 합시다.
- 이때 $S_1, \cdots, S_{K-1}$ 이 모두 제대로 정렬된다는 것과 원래 수열이 제대로 정렬된다는 것은 동치입니다.
- 예를 들어 $[3, 3, 4, 2, 1]$ 이라는 수열이 제대로 정렬된다는 것은, $S_1 = [1, 1, 1, 1, 0]$, $S_2 = [1, 1, 1, 0, 0]$, $S_3 = [0, 0, 1, 0, 0]$ 의 3 개의 수열이 모두 제대로 정렬된다는 것과 같습니다.

1D. 정렬 프로그램

- S_i 들은 모두 길이 N 인 이진 수열이므로, S_i 로 가능한 2^N 개의 수열을 모두 하나하나 정렬해볼 수 있습니다.
- 이런 수열들 중 제대로 정렬되는 수열의 집합을 S 라고 합시다.
  - $S_1, \cdots, S_{K-1}$ 은 비트마스크로 해석했을 때 $S_{K-1} \subseteq S_{K-2} \subseteq \cdots \subseteq S_1$ 을 만족합니다.
  - 반대로 그런 $S_1, \cdots, S_{K-1}$ 이 정해졌을 때, 대응되는 수열이 정확히 $\binom{M}{K}$ 개씩 생겨납니다.
- 즉, $S_{K-1} \subseteq S_{K-2} \subseteq \cdots \subseteq S_1$ 를 만족하는 집합 쌍의 개수를 구하는 문제로 바뀌게 됩니다.

1D. 정렬 프로그램

- S에 속한 원소들을 포함 관계에 의해 간선을 이어봅시다.
- 여기에서 길이 $(K-1)$인 chain의 개수를 구할 수 있으면 문제가 해결됩니다.
- $\mathrm{dp}[i][S]$를 S로 끝나는 길이 i인 chain의 개수라고 정의하면, 다음 관계식이 성립합니다.

$$\mathrm{dp}[i][S] = \sum_{A \subsetneq S} \mathrm{dp}[i-1][A]$$

- 이 DP는 흔히 SOS DP로 불리는 테크닉을 이용해 $\mathcal{O}(2^N N^2)$ 시간에 해결할 수 있습니다.
- 종합하면 전체 문제를 $\mathcal{O}(2^N(K+N^2))$ 시간에 해결할 수 있습니다.

1D. 정렬 프로그램

- 여담으로, 이 풀이에서 병목이 되는 부분은 2^N 개의 수열을 직접 정렬해보는 부분입니다.
- 약간의 아이디어를 이용해서 실제 수행 시간을 줄일 수 있습니다.
  - B 번의 `conditional_swap` 연산에 의해 바뀌는 원소의 수는 많아야 $2B$ 개이고, 이 $2B$ 개의 위치가 같은 수열들에 대해서 B 번의 연산을 동시에 처리할 수 있습니다.
  - 비트 연산만을 이용해서 위 작업을 수행할 수 있습니다.
- $B = 5$ 로 잡으면 실제 수행 시간이 약 $\frac{1}{3}$ 정도로 줄어듭니다.

1E. 영희의 심부름

`DP, probability`

- 제출 27번, 정답 12명 (정답률 44.444%)
- 처음 푼 사람: **심유근**, 75분
- 출제자: `cakelemon`

- 기댓값의 성질에 의해 $\mathbb{E}[X+Y]=\mathbb{E}[X]+\mathbb{E}[Y]$ 가 만족됩니다.
- 따라서 존재하는 사탕과 초콜릿 리스트를 $c_1, c_2, \cdots$ 라고 하면,

$$\mathbb{E}\big[\text{획득한 사탕과 초콜릿의 수}\big]=\sum_x \mathbb{E}\big[\text{획득한 } c_x \text{ 의 수}\big]$$

  입니다.
- 이때, 각 초콜릿과 사탕은 하나씩밖에 없기 때문에 "획득한 c_x 의 수" 는 항상 0 또는 1 입니다.

1E. 영희의 심부름

- 편의를 위해 함수를 두 개 정의합니다.

$$\mathrm{Path}(X, Y) = [X \text{ 에서 } Y \text{ 까지의 최단 경로 수}]$$

$$\mathrm{CP}(X, Y) = [X \text{ 에서 } Y \text{ 까지의 최단 경로 중, 초콜릿이나 사탕을 하나 이상 지나는 경로의 수}]$$

  - 이때, 영희는 아래쪽 혹은 오른쪽으로만 이동할 수 있으며, 도착지인 Y 가 X 보다 왼쪽 혹은 위쪽에 있다면 Y 는 X 에서 도달 불가능한 칸이 됩니다.
- Y 가 X 에서 도달 불가능한 경우, 위 두 함수의 값은 0 으로 정의됩니다.

1E. 영희의 심부름

- Y 가 X 에서 도달 가능한 경우, $\mathrm{Path}(X, Y)$ 는 다음과 같이 구해집니다.

$$Y - X = (r, c)\text{일 때, } \mathrm{Path}(X, Y) = \binom{r+c}{r}$$

1E. 영희의 심부름

- Y 가 X 에서 도달 가능한 경우, $\mathrm{CP}(X, Y)$ 는 다음과 같이 구해집니다.
  - Y 에 사탕/초콜릿이 있다면:

$$\mathrm{CP}(X, Y) = \mathrm{Path}(X, Y)$$

  - Y 에 사탕/초콜릿이 없다면:

$$\mathrm{CP}(X, Y) = \mathrm{CP}(X, Y \text{의 왼쪽 칸}) + \mathrm{CP}(X, Y \text{의 위쪽 칸})$$

1E. 영희의 심부름

- 철수의 위치가 A로 정해진 상황을 생각해 봅시다.
- 맨 왼쪽 위의 칸을 O, c_x의 위치를 C라고 한다면 아래 식이 성립합니다.

$$\mathbb{E}\left[\text{철수의 위치가 } A \text{일 때, 획득한 } c_x \text{의 수}\right] = \frac{\mathrm{Path}(O, C) \times \mathrm{Path}(C, A)}{\mathrm{CP}(O, A)}$$

- 철수의 위치가 무작위인 경우 $\mathbb{E}\left[\text{획득한 } c_x \text{의 수}\right]$ 의 값은 모든 철수의 위치 A에 대한 기댓값의 평균입니다.

$$\begin{aligned}\mathbb{E}\left[\text{획득한 } c_x \text{의 수}\right] &= \frac{1}{N^2}\sum_{A}\frac{\text{Path}(O,C)\times\text{Path}(C,A)}{\text{CP}(O,A)} \\ &= \frac{\text{Path}(O,C)}{N^2}\sum_{A}\frac{\text{Path}(C,A)}{\text{CP}(O,A)}\end{aligned}$$

1E. 영희의 심부름

- 식의 정리를 위해 새로운 함수를 정의합니다.

$$F(X) = \sum_{A} \frac{\mathrm{Path}(X, A)}{\mathrm{CP}(O, A)}$$

- 위 함수를 활용하여 이전의 식을 정리할 수 있습니다.

$$\begin{aligned}\mathbb{E}\left[\text{획득한 } c_x \text{의 수}\right] &= \frac{\mathrm{Path}(O, C)}{N^2} \sum_{A} \frac{\mathrm{Path}(C, A)}{\mathrm{CP}(O, A)} \\ &= \frac{\mathrm{Path}(O, C)}{N^2} \times F(C)\end{aligned}$$

- 따라서 $F(X)$ 만 빠르게 구할 수 있다면, 문제는 해결됩니다.

1E. 영희의 심부름

- $X = (r, c)$ 라고 할 때, X 의 오른쪽에 있는 칸 $R = (r, c+1)$, X 의 아래쪽에 있는 칸 $D = (r+1, c)$, 그리고 X 에서 도달가능한 임의의 점 Y 가 존재합니다.
- 이때, $\text{Path}(X, Y)$, $\text{Path}(R, Y)$, $\text{Path}(D, Y)$ 간의 관계를 생각해 봅시다.
- $\text{Path}(X, Y)$, $\text{Path}(R, Y)$, $\text{Path}(D, Y)$ 간의 관계는 Y 의 위치에 따라 두 경우로 나뉩니다.
  - $X = Y$ 인 경우: $\text{Path}(X, Y) = 1$
  - $X \neq Y$ 인 경우: $\text{Path}(X, Y) = \text{Path}(R, Y) + \text{Path}(D, Y)$

- 위에서 구한 관계를 이용하면 다음과 같이 $F(X)$에 대한 점화식을 구할 수 있습니다.

$$\begin{aligned} F(X) &= \sum_{A} \frac{\text{Path}(X,A)}{\text{CP}(O,A)} = \sum_{A=X} \frac{\text{Path}(X,A)}{\text{CP}(O,A)} + \sum_{A\neq X} \frac{\text{Path}(X,A)}{\text{CP}(O,A)} \\ &= \frac{1}{\text{CP}(O,X)} + \sum_{A\neq X} \frac{\text{Path}(R,A)+\text{Path}(D,A)}{\text{CP}(O,A)} \\ &= \frac{1}{\text{CP}(O,X)} + \sum_{A} \frac{\text{Path}(R,A)+\text{Path}(D,A)}{\text{CP}(O,A)} \\ &= \frac{1}{\text{CP}(O,X)} + F(R) + F(D) \end{aligned}$$

1E. 영희의 심부름

- 결론적으로 다음 식이 성립하며, $F(X)$의 값을 DP를 이용해 $\mathcal{O}(N^2 \log M)$에 계산할 수 있습니다.

$$F(X) = \frac{1}{\mathrm{CP}(O, X)} + F(R) + F(D)$$

- 이때 $\mathrm{CP}(O, X) = 0$인 경우는 문제 정의상 0개의 사탕/초콜릿을 얻게 되므로, $F(X) = 0$으로 취급해도 무방합니다.

1E. 영희의 심부름

- 이제 $\mathbb{E}[\text{획득한 } c_x \text{의 수}] = \dfrac{\mathrm{Path}(O, C)}{N^2} \times F(C)$ 를 활용하여 각 초콜릿과 사탕이 최종 기댓값에 기여하는 값을 모두 계산할 수 있습니다.
- 따라서 초콜릿과 사탕의 기댓값을 합해 두고, 쿼리가 들어올 때마다 뒤집힌 칸에 해당하는 기댓값을 사탕의 기댓값/초콜릿의 기댓값에 더해 주거나 빼 주면 문제를 해결할 수 있습니다.
- 이때의 시간 복잡도는 $\mathcal{O}(N^2 \log M + Q)$ 이며, 이는 문제의 제한을 통과하는 데에 충분합니다.

1E. 영희의 심부름

- 임의의 정수 수열 $a_1, \cdots, a_N$ 에 대해, a_n 의 부분곱을 활용하여 모듈러 역원 수열 $b_n = \dfrac{1}{a_n}$ 을 $\mathcal{O}(N + \log M)$ 에 구하는 기법이 밝혀져 있습니다. 해당 기법을 사용하면, 시간 복잡도를 $\mathcal{O}(N^2 + Q + \log M)$ 으로 줄일 수 있습니다.
- 반면, 매 쿼리마다도 모듈로 역원을 구하는 $\mathcal{O}(N^2 \log M + Q \log M)$ 인 구현 또한 시간 제한을 충분히 통과합니다.

1F. MST

`mst, dsu`

- 제출 7번, 정답 1명 (정답률 14.286%)
- 처음 푼 사람: **이민제**, 219분
- 출제자: `sebinkim`

1F. MST

- N 개의 정점을 가진 그래프에서 다음 쿼리를 M 번 수행합니다.
  - $u\ v\ c\ k$: 모든 $0 \le i < k$ 에 대해 $(u+i) \bmod N$ 번 정점과 $(v+i) \bmod N$ 번 정점 사이 가중치 $(c+i)$ 인 간선을 추가한다.
- 쿼리를 모두 수행한 뒤 그래프의 최소 스패닝 트리의 가중치를 구하는 문제입니다.

1F. MST

- 간선의 개수는 최대 MN 개입니다. 최소 스패닝 트리를 구하는 알려진 알고리즘을 사용하려면, 그래프의 간선의 개수를 충분히 줄여야 합니다.
- 먼저 조금 더 쉬운 문제부터 해결해 봅시다.
  - 간선의 가중치를 무시하고 (즉, c를 무시하고), 그래프의 컴포넌트 개수를 구하는 문제를 생각합시다.
  - 또한, 모든 쿼리에 대해 $k = K$ 로 일정하다 가정합시다.

1F. MST

- 잠깐 다른 문제에 대해 얘기해 봅시다.
- 칠판에 숫자 1이 적혀 있습니다. 여러분은 다음 두 가지 동작을 원하는 만큼 수행할 수 있습니다.
  - 칠판에 적혀 있는 수 하나를 골라, 그 수를 원하는 만큼 bit shift(<< 혹은 >>) 한 결과를 칠판에 적습니다.
  - 칠판에 적혀 있는 두 수를 골라, 두 수를 bitwise or한 결과를 칠판에 적습니다.
- 목표는 이진수로 나타낸 결과가 K개의 연속한 1이 되는 수, 즉 $2^K - 1$을 만드는 것입니다. 몇 번의 동작이 필요할까요?

- $\mathcal{O}(\log K)$ 번의 동작으로 목표를 달성할 수 있습니다. 두 번의 동작으로 1 의 개수를 최대 두 배씩 늘릴 수 있기 때문입니다.
- 예를 들어, $K = 11$ 일 때 $2^{11} - 1 = 11111111111_{(2)}$ 를 만드는 방법은 다음과 같습니다.
  1. $1_{(2)}$ 를 왼쪽으로 1만큼 shift한 뒤 $1_{(2)}$ 와 or하여 $11_{(2)}$ 를 만든다.
  2. $11_{(2)}$ 를 왼쪽으로 2만큼 shift한 뒤 $11_{(2)}$ 와 or하여 $1111_{(2)}$ 를 만든다.
  3. $1111_{(2)}$ 를 왼쪽으로 4만큼 shift한 뒤 $1111_{(2)}$ 와 or하여 $11111111_{(2)}$ 를 만든다.
  4. $11111111_{(2)}$ 를 왼쪽으로 3만큼 shift한 뒤 $11111111_{(2)}$ 와 or하여 $11111111111_{(2)}$ 를 만든다.
- 어떤 수를 왼쪽으로 i 만큼 shift한 뒤 원래의 자기 자신과 or하는 연산을 i-연산이라 합시다. 위의 방법은 1에 1-연산, 2-연산, 4-연산, 3-연산을 순서대로 적용한 것입니다.

- 연산의 순서를 바꿔도 결과는 같습니다. 예를 들어, 3-연산, 4-연산, 2-연산, 1-연산을 순서대로 적용하면 이렇게 됩니다.
  1. $1_{(2)}$ 에 3-연산을 적용해 $1001_{(2)}$ 를 만든다.
  2. $1001_{(2)}$ 에 4-연산을 적용해 $10011001_{(2)}$ 를 만든다.
  3. $10011001(2)$ 에 2-연산을 적용해 $1011111101_{(2)}$ 를 만든다.
  4. $1011111101_{(2)}$ 에 1-연산을 적용해 $11111111111_{(2)}$ 를 만든다.
- 갑자기 다른 문제를 설명한 이유는 이것이 우리가 원래 해결하려던 문제의 풀이와 상당히 유사하기 때문입니다.

- 이제 원래 문제로 돌아갑시다. 모든 쿼리에 대해 $k = K$로 일정할 때, 그래프의 컴포넌트 개수를 어떻게 구할 수 있을까요?
- 주어진 모든 쿼리 (u, v, k)에 대해, 간선 (u, v)를 그래프 G에 추가합시다. 우리가 해결하고자 하는 문제는 G를 1만큼 shift한 그래프 (즉, 간선 (u, v)를 간선 $((u + 1) \mod N, (v + 1) \mod N)$으로 바꾼 그래프)를 원래 그래프 G와 or하는 연산 (즉, 간선들을 합집합하는 연산)을 K번 적용한 뒤, G의 컴포넌트 개수를 세는 것으로 간주할 수 있습니다.
- 그렇다면 앞의 문제에서처럼 $\mathcal{O}(\log K)$회의 i-연산으로 문제를 해결할 수 있습니다.
- 그래프 G의 연결 정보는 $\mathcal{O}(N)$개의 간선만 있으면 저장할 수 있으므로, i-연산을 적용한 뒤 필요없는 간선들을 모두 제거해 주면 그래프의 간선의 수를 $\mathcal{O}(N)$개로 유지할 수 있습니다.

1F. MST

- 그래프 G에 i-연산을 적용하는 방법은 다음과 같습니다.
  - 현재 그래프 G에 있는 모든 간선 (u, v)에 대해, 그래프 G에 간선 $((u+i) \mod N, (v+i) \mod N)$을 추가합니다. G의 간선의 수는 두 배가 됩니다.
  - Union-Find 알고리즘 등을 사용해 G에서 사이클을 이루는 간선들을 모두 제거합니다. 이때 $\mathcal{O}(N)$개의 간선만이 남습니다.
- 간선의 개수는 항상 $\mathcal{O}(N)$이고, i-연산을 총 $\mathcal{O}(\log K)$번 반복해야 하므로, 시간복잡도는 $\mathcal{O}(N \log K)$입니다.

- 이제 k의 값이 일정하지 않고 다양할 때도 생각해 봅시다. 모든 k가 2의 거듭제곱 꼴일 때가 조금 더 쉽습니다.
- $k = 2^d$인 쿼리 (u, v, k)가 주어졌을 때, 이 쿼리에 담긴 k개의 간선을 그래프에 추가하기 위해서는 간선 (u, v)에 2^{d-1}-연산, $\cdots$, 2-연산, 1-연산이 적용되어야 합니다.
  1. $N \le 2^D$인 정수 D를 잡고, 빈 그래프 G에 2^{D-1}-연산, $\cdots$, 2-연산, 1-연산을 순서대로 적용합시다.
  2. 이때, $k = 2^d$인 쿼리 (u, v, k)가 존재한다면, 2^d-연산이 이루어진 직후에 간선 (u, v)를 그래프 G에 추가합니다.
- 이렇게 하면 모든 k가 2의 거듭제곱인 경우를 해결할 수 있습니다.

1F. MST

- 2의 거듭제곱 꼴이 아닌 k가 존재한다면 어떻게 할까요?
  - $2^d \le k < 2^{d+1}$인 정수 d가 항상 존재합니다. 이때 쿼리 (u, v, k)는 두 개의 쿼리 $(u, v, 2^d)$와 $((u + k - 2^d) \mod N, (v + k - 2^d) \mod N, 2^d)$로 대체할 수 있습니다!
  - 이렇게 하면 모든 k가 2의 거듭제곱이 되도록 할 수 있으므로, 문제가 해결됩니다.
- 시간복잡도는 $\mathcal{O}(ND + M) = \mathcal{O}(N \log N + M)$입니다.

1F. MST

- 마지막으로, 그래프의 최소 스패닝 트리의 가중치를 구하는 원래 문제로 돌아갑시다.
- 이번에는 간선에 가중치가 있습니다. 그래프 G에 i-연산을 다음과 같이 적용할 수 있습니다.
  1. 현재 그래프 G에 있는 모든 간선 (u, v, c)에 대해, 그래프 G에 간선 $((u + i) \mod N, (v + i) \mod N, c + i)$을 추가합니다. G의 간선의 수는 두 배가 됩니다.
  2. Kruskal's algorithm 등을 사용해 G의 최소 스패닝 트리를 구하고, 최소 스패닝 트리에 포함되지 않는 간선들을 모두 제거합니다.
- 쿼리 (u, v, c, k)가 주어졌다 합시다. 이때 $2^d \le k < 2^{d+1}$인 정수 d가 항상 존재합니다.
- 쿼리 (u, v, c, k)는 두 개의 쿼리 $(u, v, c, 2^d)$와 $((u + k - 2^d) \mod N, (v + k - 2^d) \mod N, c + k - 2^d, 2^d)$으로 대체할 수 있습니다.

1F. MST

- 마찬가지로 $N \le 2^D$ 인 정수 D 를 잡고, 빈 그래프 G 에 2^{D-1}-연산, $\cdots$, 2-연산, 1-연산을 순서대로 적용합시다. 이때 $k = 2^d$ 인 쿼리 (u, v, c, k) 가 존재한다면, 2^d-연산이 이루어진 직후에 간선 (u, v, c) 를 그래프 G 에 추가합니다.
- 최종 상태에서 그래프 G 의 최소 스패닝 트리의 가중치를 구해 출력하면 됩니다. 시간복잡도가 $\mathcal{O}(N \log N)$ 인 Kruskal's algorithm 을 총 D 번 수행하므로, 시간복잡도는 $\mathcal{O}(N \log^2 N + M \log N)$ 입니다.

- 한편, 그래프의 간선이 모두 가중치 순서대로 정렬되어 있다면 Kruskal's algorithm을 $\mathcal{O}(N\alpha(N))$에 수행할 수 있습니다.
  - 처음에 모든 간선을 한 번만 정렬해 놓습니다.
  - 그래프에 i-연산을 적용할 때 새로 추가되는 각 간선들끼리는 정렬된 순서를 유지할 수 있습니다.
  - 따라서, 두 정렬된 배열을 $\mathcal{O}(N)$에 합치는 방법 (merge sort에서 사용하는 방법)을 사용하면 그래프의 간선을 계속해서 정렬된 상태로 유지할 수 있습니다.
- 따라서 $\mathcal{O}(N\alpha(N)\log N + M\log M)$에 문제를 해결할 수 있습니다.

1G. 수 만들기

`ad_hoc`, `case_work`

- 제출 12번, 정답 0명 (정답률 0.000%)
- 처음 푼 사람: -, -분
- 출제자: `cubelover`

1G. 수 만들기

- 1 부터 9 까지의 숫자가 A_i ($\leq 10^9$) 개 있습니다.
- 숫자를 일렬로 나열하고 사이사이에 나눗셈 기호를 적은 뒤 괄호를 쳐서 만들 수 있는 수의 개수를 구하는 문제입니다.
- 문제의 예제에서는 $1, 2, 3, 4, 6$ 이 각각 한 개씩 있습니다.
  - 만들 수 있는 수는 $\frac{1}{144}, \frac{1}{36}, \frac{1}{16}, \frac{1}{9}, \frac{1}{4}, \frac{4}{9}, 1, \frac{9}{4}, 4, 9, 16, 36, 144$ 입니다.
  - 13 가지이므로 13 이 답이 됩니다.

1G. 수 만들기

- 먼저 숫자의 순서가 고정된 경우를 생각해봅시다.
  - 1에 각 숫자를 곱하거나 나누는 것으로 생각하면, 첫 번째 숫자는 항상 곱해지고, 두 번째 숫자는 항상 나눠집니다.
  - 세 번째 이후의 모든 숫자는 **자유로이** 곱하거나 나눌 수 있습니다.
- 예를 들어 $3 \div 4 \div 5 \div 6 \div 7$인 경우, $\frac{3}{4}$에 $5, 6, 7$을 곱하거나 나눈 8가지 경우가 있습니다.
- 위 예제에서는 8가지 경우가 모두 결과가 다르지만, 실제로는 같은 게 있을 수도 있습니다.

1G. 수 만들기

- 숫자의 순서를 마음대로 배열할 수 있으므로, **숫자가 두 개 이상인 경우** 주어진 숫자 중 적어도 하나는 곱하고, 적어도 하나는 나눠서 만들 수 있는 수의 개수가 답이 됩니다.
  - 1이 한 개 이상 있으면, 1을 곱하거나 나누는 것은 수를 바꾸지 않으므로 나머지 숫자를 자유롭게 곱하거나 나눠서 만들 수 있는 수의 개수가 답입니다.
  - 1이 없으면, 모든 숫자를 곱하거나 모든 숫자를 나누는 두 가지 경우를 제외하고 모두 만들 수 있습니다. 나머지 숫자를 자유롭게 곱하거나 나눠서 만들 수 있는 수의 개수에서 2를 뺀 것이 답입니다.
- 숫자가 한 개인 경우 답은 1이 되므로 따로 처리하면 됩니다.
- 결과적으로, 모든 숫자를 자유롭게 곱하거나 나누어서 만들 수 있는 수의 개수를 구하면 문제를 해결할 수 있습니다.

1G. 수 만들기

- 주어진 숫자가 1 이상 9 이하이므로, 만든 수는 항상 $2^a 3^b 5^c 7^d$ (a, b, c, d는 정수) 꼴입니다.
- c와 d는 5와 7을 몇 번 곱했는지에 따라서만 결정됩니다.
  - c는 $(A_5 + 1)$가지 경우가 가능하고, d는 $(A_7 + 1)$가지 경우가 가능합니다.
- a는 $2, 4, 6, 8$을 몇 번 곱했는지, b는 $3, 6, 9$를 몇 번 곱했는지에 따라서 결정됩니다.
  - a와 b가 둘 다 6에 영향을 받기 때문에 c나 d처럼 따로 구할 수는 없습니다.
- (a, b)를 좌표평면의 점으로 생각하고, 어떤 점들을 만들 수 있는지를 생각해봅시다.

1G. 수 만들기

먼저, $2, 4, 8$만으로 만들 수 있는 a들을 구해봅시다.
만들 수 있으면 O, 없으면 X로 표시했습니다.

- $A_2 = 0, A_4 = 0$: OXXOXXOXXO . . . OXXOXXOXXO
- $A_2 = 0, A_4 = 1$: OXOOXOOXOO . . . OOXOOXOOXO
- $A_2 = 0, A_4 \geq 2$: OXOOOOOOOO . . . OOOOOOOOXO
- $A_2 = 1, A_4 = 0$: OOXOOXOOXO . . . OXOOXOOXOO
- $A_2 = 1, A_4 \geq 1$: OOOOOOOOOO . . . OOOOOOOOOO
- $A_2 \geq 2$: OOOOOOOOOO . . . OOOOOOOOOO

마지막 두 경우는 빈 공간 없이 모두 만들 수 있는 경우입니다.

1G. 수 만들기

$3, 9$만으로 만들 수 있는 b들을 구해봅시다. a보다는 경우가 적습니다.
만들 수 있으면 O, 없으면 X로 표시했습니다.

- $A_3 = 0$: OXOXOXOXO . . . OXOXOXOXO
- $A_3 \geq 1$: OOOOOOOOO . . . OOOOOOOOO

$2, 3, 4, 8, 9$만으로 만들 수 있는 (a, b) 쌍은 둘을 곱한 것입니다.
예를 들어 4가 2개, 8이 1개, 9가 1개인 경우 다음과 같습니다.

OXOOOOXO
XXXXXXXX
OXOOOOXO

1G. 수 만들기

- 6을 하나 추가하면 (a, b) 대신 $(a+1, b+1)$과 $(a-1, b-1)$을 만들 수 있게 됩니다.
- 여기서부터는 case work를 열심히 하면 됩니다. 앞에서는 한 가지 케이스로 처리했지만 더 나눠야 하는 경우도 있습니다.
- 끔찍했던 케이스를 소개합니다. $A_2 = 0, A_3 = 0, A_4 = 0$일 때,
  - $A_8 \geq 2, A_9 \geq 3, A_6 \geq 6$: $8^2 9^3 = 6^6$ 이라서 중복을 따로 처리해야 합니다.
  - $A_8 < 2$ 또는 $A_9 < 3$ 또는 $A_6 < 6$: $(A_6+1)(A_8+1)(A_9+1)$이 답입니다.

2E/1H. 자취방 정하기

`graph_theory`, `dijkstra`, `linearity_of_expectation`

- 제출 48번, 정답 25명 (정답률 52.083%)
- 처음 푼 사람: **이민제**, 41분
- 출제자: `jhwest2`

11. 히스토그램 하나 빼기

`data_structure`

- 제출 22번, 정답 13명 (정답률 59.091%)
- 처음 푼 사람: **이민제**, 60분
- 출제자: TAMREF

1I. 히스토그램 하나 빼기

- 너비가 높이가 $h_1, \cdots, h_N$ 인 히스토그램이 주어집니다.
- 각 $i = 1, \cdots, N$ 에 대해, h_i 를 제외한 히스토그램 $h_1, \cdots, h_{i-1}, h_{i+1}, \cdots, h_N$ 에 속하는 가장 큰 직사각형의 크기를 $\mathcal{O}(N^2)$ 보다는 빠른 시간에 구해야 합니다.

1I. 히스토그램 하나 빼기

- 기존 히스토그램에서 더 이상 커질 수 없는 직사각형들을 **maximal**하다고 합시다.
  - 히스토그램에서 maximal한 직사각형은 많아야 N 개입니다.
- $x = i$에서 i번째 막대 h_i를 빼낸 히스토그램에서, 답이 될 수 있는 직사각형은 아래 2가지 뿐입니다.
  1. 기존 히스토그램의 maximal한 직사각형에서 h_i를 빼낸 직사각형
  2. 높이가 h_i보다 크고, 그 외 부분에서는 히스토그램에 내접하는 maximal한 직사각형에서 h_i를 빼낸 것

1. 기존 히스토그램의 maximal한 직사각형에서 h_i를 빼낸 직사각형

- 기존 히스토그램의 maximal한 직사각형이 높이가 H이고 $[s, e]$ 구간의 막대를 사용한다고 합시다.
- $i \in [s, e]$일 때는 $(e - s) \cdot H$, 그렇지 않을 때는 $(e - s + 1) \cdot H$가 답의 후보가 됩니다.
- maximal한 직사각형이 최대 N개임을 이용해 각 $i = 1, \cdots, N$에 대한 답의 후보를 총 $\mathcal{O}(N \log N)$ 시간에 모두 구할 수 있습니다.

2. 높이가 h_i 보다 크고, 그 외 부분에서는 히스토그램에 내접하는 maximal한 직사각형에서 h_i 를 빼낸 것

- 이 직사각형의 높이로 가능한 h_j 를 생각하면, i 와 j 사이의 높이는 (h_i 만 제외하면) 전부 h_j 이상이어야 합니다.
- maximality를 생각하면 $h_s < h_j$ 를 만족하는 가장 큰 $s < i$, $h_e < h_j$ 를 만족하는 가장 작은 $e > j$ 에 대해 $[s+1, e-1]$ 구간이 밑변이 되어야 합니다.

- 이러한 점을 생각하면, j를 고정했을 때 가능한 i의 후보는 아래 둘뿐입니다.
  - $h_i < h_j$인 최대의 $i < j$
  - $h_j > h_i$인 최소의 $i > j$
- 이러한 i를 찾는 것과, 밑변의 길이를 찾는 것은 모두 "특정 구간에서 값이 x보다 작은 최대 인덱스"를 묻는 쿼리입니다.
  - RMQ, Union Find 등을 통해 $\mathcal{O}(N \log N)$ 시간에 찾을 수 있습니다.

1J. 넓이를 같게

`geometry`, `math`, `linear_algebra`, `case_work`

- 제출 9번, 정답 3명 (정답률 33.333%)
- 처음 푼 사람: **윤교준**, 158분
- 출제자: `jhwest2`

1J. 넓이를 같게

- 2차원 평면 위에 격자점을 끝점으로 가지는 N개의 선분이 놓여 있습니다.
- 적당한 점 P를 잡아 각 선분이 P와 이루는 삼각형의 넓이가 모두 같도록 할 수 있는지 판별하고, 가능하면 그러한 점 P를 하나 구해야 합니다.

1J. 넓이를 같게

- 선분의 두 끝점 $A(x_1, y_1)$ 과 $B(x_2, y_2)$, 그리고 점 $P(x, y)$ 에 의해 만들어지는 삼각형의 넓이를 구해봅시다.
- $\overrightarrow{PA} = (x_1 - x, y_1 - y)$, $\overrightarrow{PB} = (x_2 - x, y_2 - y)$ 로 쓸 수 있습니다.
- 삼각형의 넓이는 $\frac{1}{2}|\overrightarrow{PA} \times \overrightarrow{PB}| = \frac{1}{2}|(y_2 - y_1)x - (x_2 - x_1)y + (x_2y_1 - x_1y_2)|$ 로 계산됩니다.
- i 번째 선분에 대해서 $y_2 - y_1 = a_i$, $x_2 - x_1 = b_i$, $x_2y_1 - x_1y_2 = c_i$ 라고 둡시다.
- $|a_ix - b_iy + c_i|$ 가 모든 i 에 대해서 같도록 하는 x, y 의 값을 구하는 문제가 됩니다.

1J. 넓이를 같게

- $N = 2$인 경우 아래 두 식

$$a_1x - b_1y + c_1 = a_2x - b_2y + c_2$$
$$a_1x - b_1y + c_1 = -(a_2x - b_2y + c_2)$$

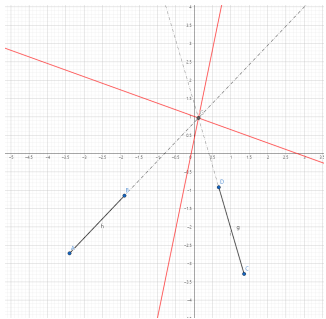
  중 적어도 하나는 해를 가지므로, 아무 해나 출력합니다.

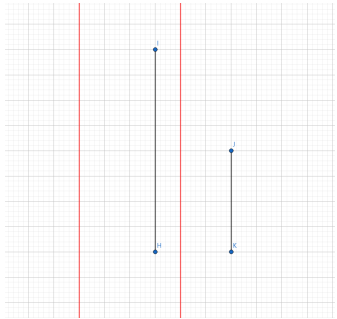
- $N \geq 3$인 경우, 다음 연립방정식에 주목합시다.

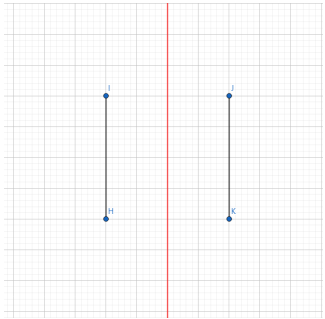
$$\begin{cases} |a_1x - b_1y + c_1| = |a_2x - b_2y + c_2| \\ |a_2x - b_2y + c_2| = |a_3x - b_3y + c_3| \end{cases}$$

- 이 방정식은 어떤 형태의 해를 가질까요?

1J. 넓이를 같게







- 위 세 가지 케이스 외에도, 두 선분이 길이가 같고 같은 직선 위에 놓이는 경우 평면 전체가 해가 됩니다.
- 따라서 위치 관계에 따라 총 4가지 중 한 가지 형태로 해가 나타납니다.

1J. 넓이를 같게

- 약간의 케이스 분류를 통해 연립방정식의 해가 다음 중 하나를 만족함을 알 수 있습니다.
  - 평행하지 않은 두 선분이 존재하는 경우, 최대 4개의 해를 가집니다.
  - 세 선분이 길이가 같고 한 직선 위에 놓이는 경우, 평면 전체가 해가 됩니다.
  - 그 외의 경우, 해가 아예 없거나 모든 해가 하나의 직선 위에 놓입니다.

1J. 넓이를 같게

- 처음에 주어지는 선분 중 평행하지 않은 두 선분이 있는 경우 답의 후보가 최대 4개로 제한됩니다.
  - 4개의 후보가 조건을 만족하는지 일일이 확인해보면 되고, 이는 $\mathcal{O}(N)$ 시간에 구현할 수 있습니다.
- 모든 선분이 평행한 경우를 살펴봅시다.
  - 우선 모든 선분이 길이가 같고 하나의 직선 위에 놓이면, 평면 전체가 해가 되므로 아무 점이나 하나를 고르면 됩니다.
  - 그렇지 않은 경우, 해가 존재한다면 해의 자취는 모든 선분들과 평행한 직선으로 나타납니다.
  - 두 개의 선분을 잘 선택했을 때 이 직선이 최대 2개로 제한되므로, 이 두 개의 직선이 조건을 만족하는지 일일이 확인해보면 됩니다. 역시 $\mathcal{O}(N)$ 시간에 할 수 있습니다.
- 종합하면 전체 문제를 $\mathcal{O}(N)$ 시간에 해결할 수 있습니다.