UCPC 2020 본선 풀이

Official Solutions

by

전국 대학생 프로그래밍 대회 동아리 연합

문제		의도한 난이도	출제자
A	만쥬의 식사	Easy	riroan
B	비숍 여행	Easy	riroan
C	함수 복원	Medium	moonrabbit2
D	소가 길을 건너간 이유 2020	Medium	functionx
E	지도 설치	Challenging	Diuven
F	애완 트리	Hard	moonrabbit2
G	그건 망고가 아니라 고양이예요	Medium	evenharder
I	빛의 전사 크리퓨어	Hard	pichulia
J	관광 사업	Challenging	moonrabbit2
K	데이터 제작	Hard	jhnah917
L	피자 배틀	Medium	16silver

A. 만쥬의 식사

greedy
출제진 의도 – **Easy**

- 제출 0번, 정답 0명 (정답률 0%)
- 처음 푼 사람:
- 출제자: riroan

- 밥그릇에 들어있는 츄르의 개수는 늘어나지 않습니다.
- 그래서 모든 츄르가 같아지기 위해 가장 작은 값으로 맞춰야 합니다.
- 츄르의 최솟값을 m 이라고 하면 $\sum_{i=1}^{N}(a_i - m)$ 의 값을 구하면 정답이 됩니다.
- 총 시간복잡도는 $\mathcal{O}(N)$ 입니다.

B. 비숍 여행

`implementation, math`

출제진 의도 – **Easy**

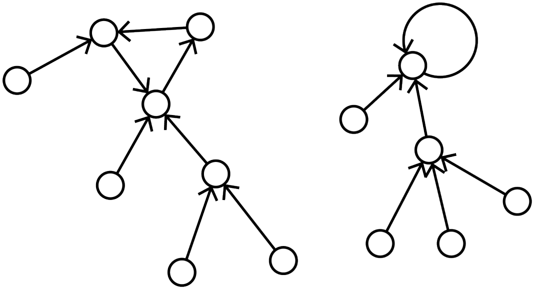
- 제출 0번, 정답 0명 (정답률 0%)
- 처음 푼 사람:
- 출제자: `riroan`

- 비숍은 한번 이동할 때마다 x좌표와 y좌표의 홀짝이 각각 변합니다.
- 예를들어 비숍이 (짝수, 홀수)좌표에 있다면 한번 이동했을 때 (홀수, 짝수)좌표로 이동합니다.
- 하지만 이동을 하더라도 **x좌표 + y좌표**의 홀짝은 변하지 않습니다.
- 따라서 비숍의 시작좌표 합의 홀짝과 같은 동전좌표 합의 홀짝인 개수를 구하면 됩니다.
- 총 시간복잡도는 $\mathcal{O}(N)$입니다.

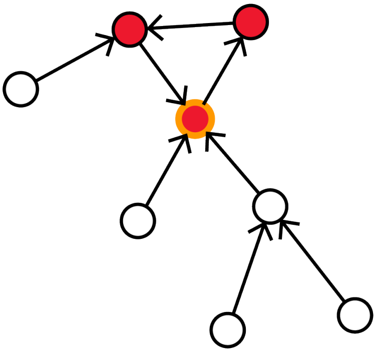
C. 함수 복원

combinatorics
출제진 의도 – **Medium**

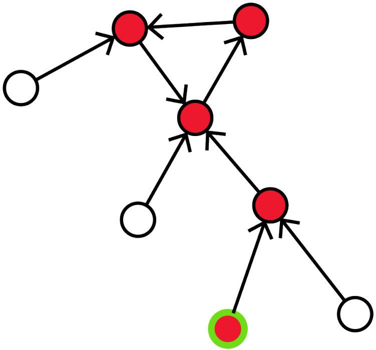
- 제출 406번, 정답 99팀 (정답률 24.63%)
- 처음 푼 팀: **여기가월파2020인가요** (월파, 치러, 왔어요), 10분
- 출제자: moonrabbit2



- 함수 그래프는 각 컴포넌트가 하나의 사이클에 연결된 트리들로 이루어져 있습니다.
- 자기보다 더 적은 수의 정점에 도달할 수 있는 정점이 있다면 트리에 속하고, 그렇지 않으면 사이클에 속하는 정점입니다.



- 사이클 정점에서는 사이클 상의 모든 정점이 도달 가능합니다.
- 사이클 상의 한 정점을 고정하고, 그 정점으로부터 도달 가능한 정점들을 묶는 것을 반복해 모든 사이클을 구할 수 있습니다,
- 각 사이클에 대해, 사이클의 길이가 L 이라면 원순열 $(L-1)!$ 을 답에 곱합니다.



- 트리 정점에서는 사이클 상의 정점과 함께 사이클까지의 트리 상의 경로에 있는 정점들이 추가로 도달 가능합니다.
- 트리 정점의 부모는 해당 정점보다 도달 가능한 정점의 수가 1 적습니다.

- 트리 정점에 대해, 부모도 트리 정점이면 부모가 유일하게 고정됩니다.
- 부모가 사이클 정점이면 사이클 상의 임의의 정점이 부모가 될 수 있습니다.
- 그러므로 부모가 사이클 정점이면 부모가 포함된 사이클의 크기 L 을 답에 곱합니다.

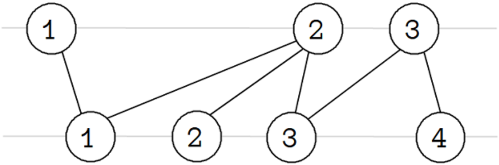
- 시간복잡도는 $\mathcal{O}\left(N^2\right)$ 입니다. 제한이 매우 여유롭게 주어져, $\mathcal{O}\left(N^3\right)$ 풀이도 가능합니다.
- 답이 0인지 판별하는 것은 어렵지 않습니다. 온갖 조건문을 넣고 싶지 않다면 실제 함수를 하나 만들면 편하게 구현이 가능합니다. 다만, 이 문제에서는 필요하지 않았습니다.

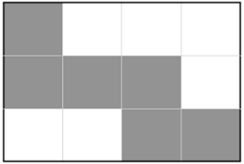
D. 소가 길을 건너간 이유 2020

`dynamic_programming`
출제진 의도 – **Medium**

- 제출 270번, 정답 30팀 (정답률 11.11%)
- 처음 푼 팀: **Let Us Win UCPC 2020** (윤교준, 김세빈, 이민제), 27분
- 출제자: `functionx`

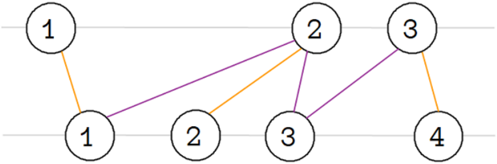
D. 소가 길을 건너간 이유 2020

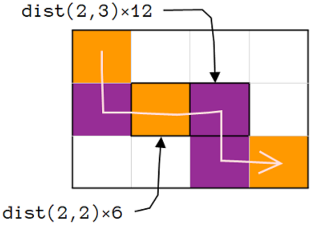




- $N \times M$ 격자를 그려봅시다.
- 위쪽 i 번째 헛간과 아래쪽 j 번째 헛간을 연결하는 항로가 있다면 격자의 i 행 j 열을 칠해줍시다.
- 놀랍게도 1행 1열에서 N 행 M 열까지 가는 최단경로 형태가 됩니다.

D. 소가 길을 건너간 이유 2020





- 역발상을 해보자면, 최단거리의 합은 각 항로에 대하여 (항로의 힘) × (항로를 지나는 경로의 수)의 합입니다.
- (항로를 지나는 경로의 수)는 $N + M - 1$ 또는 $(i + j - 1)(N + M - i - j + 1)$ 입니다.
- 격자에서 보면 직진할 때는 $N + M - 1$, 꺾을 때는 $(i + j - 1)(N + M - i - j + 1)$ 입니다.

DP 정의를 이렇게 합니다.

- $DP[i][j][dir]$: 격자의 i 행 j 열 까지 간 다음 dir(오른쪽/아래쪽) 방향으로 나갈 때 최소비용
- i 행 j 열을 지난 이후의 상황은 무시합니다.

(i, j) 직전에 어디를 지났는지 생각해보면 점화식을 세울 수 있습니다.

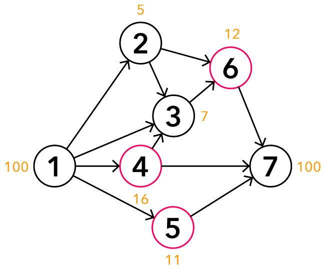
E. 지도 설치

network_flow
출제진 의도 – **Challenging**

- 제출 28번, 정답 3팀 (정답률 10.71%)
- 처음 푼 팀: **정우팬클럽** (회장, 부회장, 정우), 199분
- 출제자: Diuven

E. 지도 설치

- 단순 방향 그래프 $G = (V, E)$, 두 정점 S, E, 각 정점별 비용 C_v, 자연수 $K \le 5$
- 비용의 합이 최소이면서 다음의 조건을 만족하는 정점 집합 X 를 찾는 것이 목표입니다.
- 조건: S 에서 E 로 가는 모든 경로들이 X 의 원소를 적어도 K 개 포함하는 것



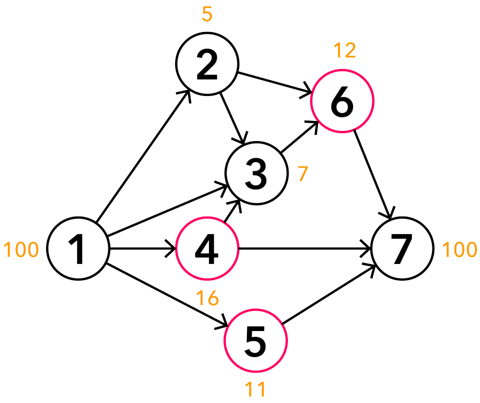
$K = 1$ 인 경우를 풀어봅시다.

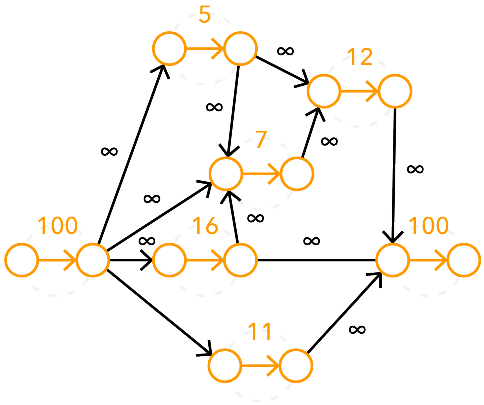
- 정점 집합 X 로 그래프 G 를 '쪼개면' 됩니다.
- 네트워크 플로우 알고리즘으로 s-t edge min cut을 구할 수 있습니다.
- 모든 정점을 v_{in}, v_{out} 두 개로 나누고, 들어오는 간선과 나가는 간선을 따로 담당합니다.
- $v_{in} \to v_{out}$ 간선을 추가하고, capacity를 C_v 로 정합니다.
- 나머지 간선들은 모두 capacity를 ∞ 로 설정합니다. (이론상 최대 유량보다 큰 값)

이 그래프에서 $S - E$ max flow를 구하면, 가능한 최소 비용을 구할 수 있습니다.

편의상 $v_{in} \to v_{out}$ 인 간선들을 Type V 간선, 나머지 간선들을 Type E 간선이라고 부릅시다.

E. 지도 설치

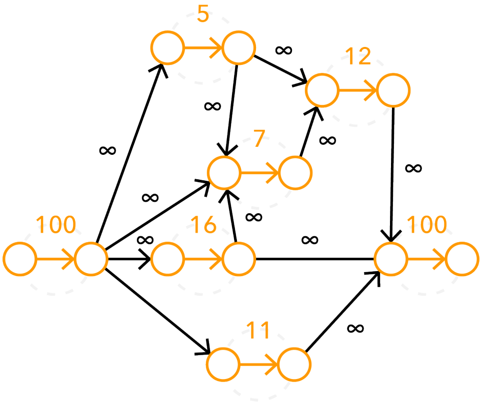


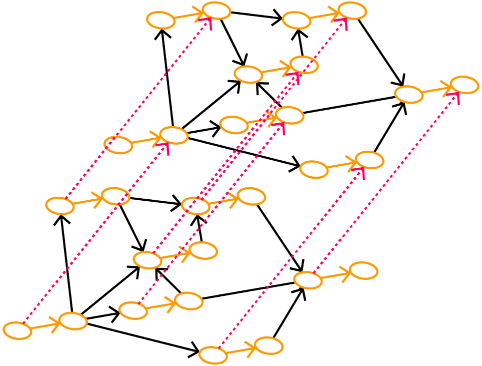


$K > 1$인 경우

1. $K = 1$인 경우의 플로우 그래프를 K개 만듭니다.
   z번째 플로우 그래프를 G_z이라고 하고, G_z에 속하는 정점들을 ${}^{z}v$처럼 부릅시다.
2. 모든 정점에 대해서, ${}^{z}v_{in}$과 ${}^{z+1}v_{out}$를 가중치가 무한대인 간선으로 잇습니다. (Type Z 간선)
3. super-source와 super-sink에 사용할 두 정점 A와 B를 만듭니다.
4. 모든 z에 대해 $A \to {}^{z}S_{in}, {}^{z}E_{out} \to B$를 가중치가 무한대인 간선으로 잇습니다. (Type S 간선)

다음의 그림처럼, 동일한 그래프를 K개 만들어 쌓았다고 생각하고, Type Z 간선들이 그 그래프를 z축으로 연결한다고 생각합시다.





그림에는 정점 A, B와 Type S 간선들이 생략되어 있습니다.

이 그래프에서 $K = 1$일 때처럼 $A - B$ max flow를 구하면 가능한 최소 비용을 구할 수 있습니다.

- 경로를 지나가면서, 지도를 무시하고 지나갈 수 있는 '찬스'가 $K - 1$개 있다고 합시다.
- 플로우를 구할 때 Type Z 간선을 사용한다는 것은 찬스를 사용하는 것을 의미합니다.
  즉, Type V 간선이 막혀 있을 때 Type Z 간선을 통해 한 층 위로 올라갈 수 있는 것입니다.
- 그래프에 max flow를 흘려 주었을 때, A에서 B로 가는 경로들은 전부 플로우로 막혀 있습니다.
- 막힌 간선은 모두 Type V 간선입니다. 따라서 A에서 지도를 K번 미만 거쳐서 B로 갈 수 없습니다.

일반적인 플로우 그래프에서 max flow를 흘려 주었을 때 min cut을 구하는 방법:

1. 플로우 그래프에서 '뚫린 간선'을 잔여 용량이 0이 아닌 간선들이라고 합시다.
2. (super) source 정점에서 '뚫린 간선'들만 사용해서 도달 가능한 정점들을 체크합니다.
3. 어떤 간선의 한쪽 끝점은 도달가능하고 다른 한쪽은 불가능한 경우 그 간선은 cut에 속하게 됩니다.

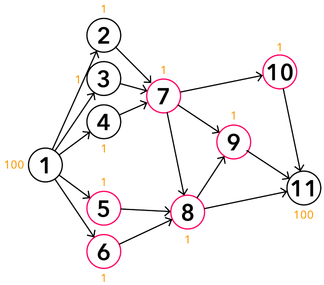
cut에 속하는 간선들은 모두 Type V 간선들이고, 그 간선에 해당하는 정점들이 우리가 구하고자 하는 정점 집합 X를 이룹니다.

E. 지도 설치

min cut을 구하고 cut 정점들의 C_v 를 무한대로 만드는 과정을 K 번 반복하는 풀이는 틀립니다.

한 min cut이 한 경로를 여러 번 막는 경우에 답을 구하지 못합니다.

아래와 같은 경우, 최적해가 아닌 $2, 3, 4, 5, 6, 7, 8$ 를 선택하게 됩니다.



구하려는 정점 집합 X 이 존재하지 않는 경우 (답 -1) 와 공집합인 경우 (답 0) 를 주의합시다.
supersource와 supersink를 사용하지 않은 경우 -1을 체크하지 못 할 수 있습니다.

그래프의 크기는 정점이 $\mathcal{O}(KN)$ 개, 간선이 $\mathcal{O}(K(N+M))$ 개입니다.

출제자의 풀이는 $\mathcal{O}(V^2E)$ 시간에 동작하는 Dinic 알고리즘을 사용합니다.
이론상 최대 계산량 $K^3N^2M \approx 3 \times 10^8$. BOJ 기준 실행 시간 10ms, 메모리 4MiB 이내

빠른 플로우 알고리즘을 사용할수록 문제는 빠르게 풀립니다.
가장 단순한 형태의 Ford-Fulkerson 알고리즘을 사용하면 시간 초과.
$\mathcal{O}(VE^2)$ 인 Edmonds-Karp 알고리즘을 사용해도 Dinic 풀이만큼 빠르게 돕니다.

F. 애완 트리

dynamic_programming
출제진 의도 – **Hard**

- 제출 71번, 정답 10팀 (정답률 14.09%)
- 처음 푼 팀: **정우팬클럽** (회장, 부회장, 정우), 39분
- 출제자: moonrabbit2

- 우선, S 나 E 가 $400(N-1)$ 보다 크면 의미가 없습니다. 더 크면 $400(N-1)$ 로 바꿔줍시다.
- 지름이 S 이상 E 이하인 경우의 수는 (지름이 S 이상인 경우의 수)-(지름이 $E+1$ 이상인 경우의 수) 입니다.
- 지름이 X 이상인 경우의 수를 구할 수 있으면 문제를 해결할 수 있습니다.

트리DP를 이용해 지름이 X 이상인 경우의 수를 구합니다.

- $D_{u,d}$는 u의 서브트리에서 지름이 X 미만이고 최대 깊이는 d인 경우의 수입니다.
- E_u는 u의 서브트리에서 지름이 X 이상인 경우의 수입니다.

루트를 1로 잡으면 답은 E_1 입니다. d는 최대 X 이므로 DP 테이블의 공간복잡도는 $\mathcal{O}(NX)$ 입니다.

현재 서브트리 위에 $[L, R]$ 간선이 생기면 DP값들이 갱신됩니다. 새 값들을 D'_u 와 E'_u 라 합시다. 다음과 같은 방법으로 D'_u 와 E'_u 를 구할 수 있습니다.

- $[L, R]$ 범위의 각 D 에 대해, $d + D < X$ 이면 $D'_{u,d+D}$ 에 $D_{u,d}$ 를 더합니다. 그렇지 않으면 E'_u 에 더합니다.
- E'_u 에 $(R - L + 1)E_u$ 를 더합니다.

이 과정을 $N - 1$ 번 수행하므로 총 시간복잡도는 $\mathcal{O}(400NX)$ 입니다.

자식에서 올라온 DP값과 기존의 DP값을 합쳐야 합니다. D_u 와 E_u, D_v 와 E_v 를 합쳐 D'_u 와 E'_u 가 되었다고 합시다.

- $d_1 + d_2 < X$ 라면 $D'_{u,\max(d_1,d_2)}$ 에 $D_{u,d_1} D_{v,d_2}$ 를 더합니다. 그렇지 않으면 E'_u 에 더합니다.
- E'_u 에 $D_{u,d}E_v$ 를 더합니다.
- E'_u 에 $D_{v,d}E_u$ 를 더합니다.
- E'_u 에 E_uE_v 를 더합니다.

이 과정을 $N-1$ 번 수행하므로 총 시간복잡도는 $\mathcal{O}\left(NX^2\right)$ 입니다.
이 과정에서 현재 DP테이블의 크기를 저장해 놓아 필요한 부분만 계산하면 $\mathcal{O}\left(400NX\right)$ 입니다. 증명은 생략합니다.

너무 느립니다. 각 부분을 최적화합시다.

- 아이디어는 $D'_{u,d}$ 와 E'_u 값들을 구할 때 $D_{u,d}$ 와 E_u 각각에 대해 새로운 값들을 순회하는 대신, $D'_{u,d}$ 와 E'_u 에 더해지는 값들을 구해 D'_u 를 순회하는 것입니다.
- 해당하는 $D_{u,d}$ 의 d 값들이 구간을 이뤄, 전처리 후 $O(1)$ 에 D_u 에서 d 의 구간에 대한 구간 합 쿼리를 구할 수 있음을 이용합니다.
- 편의상 $D_{u,[l,r]} = \sum_{i=l}^{r} D_{u,i}$ 로 정의합니다. $l > r$ 이라면 이 값은 0 입니다.

새 $[L, R]$ 간선이 생겼을 때 D'_u 와 E'_u 를 빠르게 구해봅시다.

- $D'_{u,d} = D_{u,[\max(0,d-R),d-L]}$ 입니다.
- $(R - L + 1)E_u$ 를 E'_u 에 더하는 것은 똑같이 하면 됩니다.
- $[\max(L, X), X - 1 + R]$ 범위의 d 에 대해, E'_u 에 $D_{u,[\max(0,d-R),d-L]}$ 을 더합니다.

총 시간복잡도는 $\mathcal{O}(NX)$ 입니다.

두 DP 테이블을 합쳤을 때 D'_u 와 E'_u 를 빠르게 구해봅시다.

- $D'_{u,d}$ 에 $D_{u,d}D_{v,[0,\min(d,X-1-d)]}$ 와 $D_{u,[0,\min(d-1,X-1-d)]}D_{v,d}$ 를 더합니다. 이는 두 서브트리 중 하나에서는 반드시 최대 깊이가 d 인 트리가 있어야 하기 때문입니다.
- 각 d 에 대해 $D_{u,d}D_{v,X-d,X-1}$ 을 E'_u 에 더합니다.
- E'_u 에 $D_{u,d}E_v$ 를 더합니다.
- E'_u 에 $D_{v,d}E_u$ 를 더합니다.
- E'_u 에 E_uE_v 를 더합니다.

총 시간복잡도는 $\mathcal{O}(NX)$ 입니다.

X를 S와 $E+1$로 두고 두 번 진행하므로, 최종적으로 총 시간복잡도는 $\mathcal{O}\left((S+E)N\right)$ 입니다.

시간제한이 여유롭게 주어졌지만 매우 비효율적으로 코딩했을 경우 시간 초과가 날 수 있습니다.

이 풀이 외에도 지름의 중점의 위치를 고정한 후 열심히 경우의 수를 구하는 풀이 등도 가능합니다.

G. 그건 망고가 아니라 고양이예요

`dfs, binary_search`
출제진 의도 – **Medium**

- 제출 139번, 정답 8팀 (정답률 5.75%)
- 처음 푼 팀: **여기가월파2020인가요** (월파, 치러, 왔어요), 68분
- 출제자: evenharder

G. 그건 망고가 아니라 고양이에요

문제의 모티브가 된 망고님입니다. 풀이랑 상관은 없지만 귀엽습니다.

S 를 \$ 단위로 $S = S_0\$S_1\$\cdots\$S_m$ 처럼 쪼개서 표현하면 M_i 를 다음과 같이 쓸 수 있습니다.

- $M_1 = S_0\underline{M_0}S_1\underline{M_0}\cdots\underline{M_0}S_m$
- $M_2 = S_0\underline{M_1}S_1\underline{M_1}\cdots\underline{M_1}S_m$
- $\vdots$
- $M_k = S_0\underline{M_{k-1}}S_1\underline{M_{k-1}}\cdots\underline{M_{k-1}}S_m$

즉 M_i 를 S 의 글자에 대응시킬 수 있습니다.

S 에 $\$$ 가 1개 있으면 어떻게 될까요? $S = S_0\$S_1$ 이라 하면,

- $M_1 = S_0\underline{M_0}S_1$
- $M_2 = S_0S_0\underline{M_0}S_1S_1$
- $\vdots$
- $M_k = S_0^k\underline{M_0}S_1^k$

가 되어, 반복문 등으로 문제를 해결할 수 있습니다. (S_i^k 는 S_i 가 k 번 반복된 문자열입니다.)

\$가 2개 이상 있으면 다음과 같은 DFS 기반 $\mathcal{O}(Qk|S|)$ 풀이를 생각할 수 있습니다.

- DFS를 통해 M_i의 글자를 조사하면서, \$면 M_{i-1}의 길이를, 아니면 1을 뺍니다.
- 남은 칸 수보다 M_{i-1}의 길이가 크면 목표 문자열이 포함되어 있으므로 재귀합니다.
- 원하는 부분문자열을 전부 확인할 때까지 반복합니다.

시간 복잡도를 어떻게 줄일 수 있을까요?

M_{i+1} 이 M_i 보다 2배 이상 길기 때문에, M_{60} 정도만 되어도 길이는 10^{18} 을 넘깁니다.

그러므로 적당한 60 이하의 k' 을 잡아, 길이가 10^{18} 이상인 $M_{k'}$ 만 조사해도 충분합니다.

- S 가 $S_1\$\cdots$ 꼴일 때, $M_k = S_1^{k-k'} M_{k'} \cdots$ 꼴입니다 ($k' \leq k$).
- 조건에 의해, $S_1^{k-k'} M_{k'}$ 이후는 탐색할 필요가 없습니다.

그러므로 $S_1^{k-k'}$ 는 따로 처리하고, $M_{k'}$ 에서 DFS를 적용하면 됩니다.

- DFS에서 각 M_i를 다 순회할 필요도 없이, 이분 탐색을 통해 원하는 시작 위치를 효율적으로 찾을 수 있습니다.
- 방문하는 칸의 개수는 세그먼트 트리와 비슷하게 출력하는 문자의 개수에 비례하므로, 시간복잡도 $\mathcal{O}\left(Qk' \lg(|S|) + \sum(b_i - a_i)\right)$의 풀이가 완성됩니다.
- M_0를 순회할 때 잘못 처리하면 시간복잡도에 $|M_0|$가 추가로 곱해질 수 있습니다.

I. 빛의 전사 크리퓨어

`greedy`
출제진 의도 – **Hard**

- 제출 124번, 정답 16팀 (정답률 12.90%)
- 처음 푼 팀: **1211789** (김현수, 신승원, 최석환), 45분
- 출제자: `pichulia`

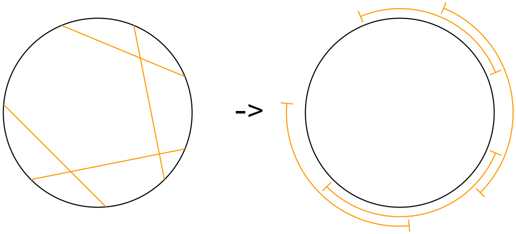
풀이를 열심히 준비해서 오니까 검수진한테

"이거 너무 Well-Known이라서 사전지식을 아는 사람만 풀듯"

이라고 구박받은 문제입니다.ㅠㅠ

검수진이 알려준 Well-Known 풀이가 있고
사전지식 없이 풀리는 재미있는 Ad-hoc 풀이가 있습니다.

두 풀이 모두 유용하기 때문에 모두 설명하겠습니다.



문제에서 고무줄이라고 되있는 것들을 어떤 시작점과 끝점이 있는 '구간'으로 변환해서 생각할 수 있습니다.

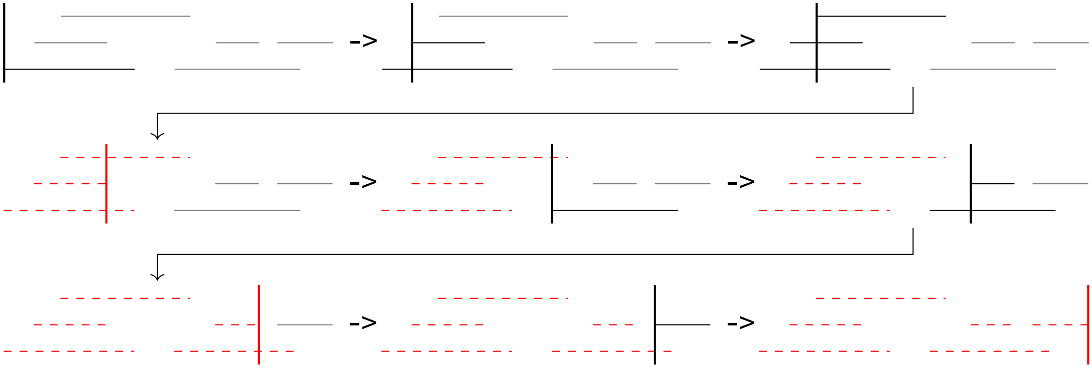
이제 본 문제는 N 개의 구간이 있고, 그 구간을 모두 파괴하는 빔의 최소값을 구하는 문제가 됩니다.

빔이 구간의 말단 지점에 스쳐도 구간을 파괴하므로, 구간의 중간이 아니라 시작점 or 끝점에 발사하는 것이 항상 이득입니다.

본 문제 해설에서는 구간의 시작점에서 끝점으로의 방향을 시계방향으로 생각했으며, 항상 구간의 끝점에 빔을 쏘는 것으로 설명할 것입니다.

직선일 때에는 조금 전형적이면서 유명한 Greedy 해법으로 풀립니다.

1. 시작점을 기준으로 구간을 정렬한다.
2. 위치를 조금씩 증가시켜가다가, 최초로 만나는 끝점이 있으면 빔을 발사한다.
   빔을 발사한 지점보다 시작점이 더 작은 구간은 모두 파괴당한다.
3. 빔을 발사한 지점 이후부터 다시 시작해서 위 과정을 반복한다.



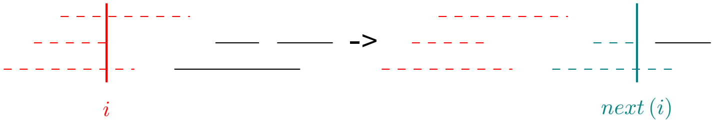
이렇게 진행하는 것이 항상 최소한의 빔을 사용합니다.

왜냐하면 끝점보다 더 큰 지점에 빔을 발사하면 그 구간이 파괴되지 않은 채로 남으며,
끝점보다 더 작은 지점에 빔을 발사하는 것은 항상 동률이거나 손해이기 때문입니다.

편의상 정렬 등의 각종 전처리는 이미 완료된 상태라고 가정하겠습니다.

이 풀이의 정답을 M 이라고 했을 때
단순비교로 $\mathcal{O}(N)$ 에 풀리는 해법이 있고
Segment Tree 등의 자료구조를 써서 $\mathcal{O}(M \log N)$ 로 푸는 해법이 있고
전처리를 적절하게 잘 해놔서 $\mathcal{O}(M)$ 에 풀리는 해법이 있습니다.

Well-Known 풀이 - Sparse Table



i 번 구간의 끝점에 빔을 발사했을 때, 살아남은 구간들 중 끝점의 좌표가 가장 작은 구간의 번호를 구합니다.

빔 한방에 모든 구간이 파괴되서 답이 1 이 되는 경우를 주의합시다.

i 번 구간의 끝점에 빔을 1회 발사했을 때 다음 구간의 번호를 알고 있으므로,

이를 활용해서 i 번 구간의 끝점에 빔을 2회, 4회, 8회, $\cdots$, 131072회 발사했을 때 다음 구간의 번호를 구할 수 있습니다.
(이런 식으로 데이터를 저장해놓는 기법을 Sparse Table 이라고 부른다고 합니다.)

원형이기 때문에 빙빙 돌 수 있는데, 이는 현재 구간의 번호를 구하는 과정이 몇바퀴를 돌았는지를 같이 저장해서 해결할 수 있습니다.

이제 각 1 번부터 N 번 구간의 끝점에서 빔 발사를 시작해, 한바퀴를 도는 데 필요한 빔의 발사 횟수를 $\mathcal{O}(\log N)$ 만에 구할 수 있게 됩니다.

전체 시간복잡도는$\mathcal{O}(N \log N)$ 공간복잡도는 $\mathcal{O}(N \log N)$

사전지식이 필요없는 풀이 - 비둘기집의 원리와 믿음 $(\cdots)$

이 문제를 풀기 위해선 최소 $\mathcal{O}(M \log N)$ 만에 직선문제를 해결할 수 있어야합니다.
대략적인 흐름은 다음과 같습니다.

1. 적당힌 곳을 '첫 발사 지점' 으로 지정하고, 그곳에 빔을 발사한다.
2. 파괴되지 않고 살아남은 구간들에 대해서 직선 풀이를 $\mathcal{O}(M \log N)$ 만에 해결한다.
3. 이 과정을 $\mathcal{O}(N/M)$ 번 반복한다.

각 구간의 끝점인 N 개의 지점이 '첫 발사 지점'의 후보지점들입니다.
이 후보지점들 중 정답이 나오도록 $\mathcal{O}(N/M)$ 개의 지점을 뽑아내는게 쉽지 않습니다.

짜잘한 증명은 잠시 생략하고 결론만 적어넣으면 다음과 같습니다.

Theorem

다른 구간의 끝점이 가장 적게 포함된 구간에는 끝점이 최대 $\mathcal{O}\left(N/M\right)$ 개 포함되어있다.
이 끝점들 중 하나에 정답이 되는 첫 발사 지점이 존재한다.

이 명제가 사실이라면, 각 구간을 좌표압축해서 구간의 범위를 $\mathcal{O}(N)$ 으로 줄인 뒤,
$e - s$ 값이 가장 작은 구간을 하나 찾은 다음에
s 부터 e 까지 빔을 한번씩 발사해 보면서 $\mathcal{O}(M \log N)$ 만에 직선문제를 해결하면 정답이 됩니다.

와! 신기해라!

결론이 매우 재밌지만 그것을 증명하는 과정은 아주 험난합니다. ㅠㅠ

이를 증명하기 전에 다음과 같은 6개의 명제가 먼저 증명되어야 합니다.

Lemma 1

적당히 아무 지점에 빔을 발사해서 구한 답이 M 이면, 전체 문제의 정답은 M 또는 $M-1$ 이다.

Lemma 2

만약 답이 $M-1$ 이 되는 첫 발사 지점이 있는 경우, 그 $M-1$ 개의 빔 중 어느 지점을 첫 발사 지점으로 잡아도 $M-1$ 개의 빔으로 모든 구간을 파괴할 수 있다.

Lemma 3

다른 구간을 완전히 포함하는 어떤 구간이 존재하는 경우, 그 구간을 지워도 답이 변하지 않는다. 이렇게 답에 영향을 주는 구간만 남긴 경우, '시작점' 기준으로 정렬한 것과 '끝점' 기준으로 정렬한 것의 순서가 같다.

Lemma 4

답에 영향을 주는 구간만 남긴 경우, 각 구간은 M 개의 영역 중 최대 2 개의 영역에 걸쳐서 존재한다.

Lemma 5

답에 영향을 주는 구간만 남긴 경우, 구간 내에 끝점의 개수가 $2N/M$ 개 이하인 구간이 존재한다.

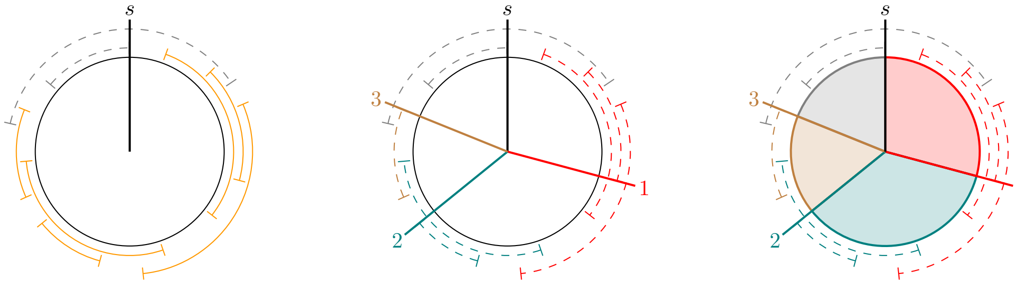
Lemma 6

구간 내에 끝점의 개수가 $2N/M$ 개 이하인 구간이 존재한다.

Lemma들끼리 잘 뭉치면 Theorem의 결론이 나옵니다.

다른건 다 그려려니 하는데
Lemma 1 이 아마 직관적이지 않을 것입니다.
대략적인 증명을 설명하겠습니다.

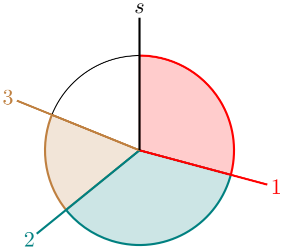
Proof.



M 개의 빔을 경계선 삼아서 전체 지점을 M 개의 영역으로 나눌 수 있습니다.
경계선이 되는 빔은 두 영역 중 반시계방향에 있는 영역에 포함시킵니다.

□

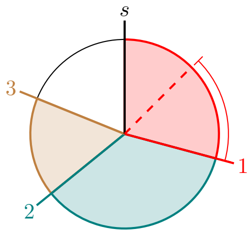
Proof.



맨 처음 발사한 빔과 마지막 빔 사이에 있는 영역 한 곳을 제외한 나머지 $M-1$개의 영역에는 반드시 빔이 한 개 이상 존재해야 합니다.

이는 직선으로 문제를 전개한 뒤 답을 구하는 과정을 잘 생각해보시면 알 수 있습니다. □

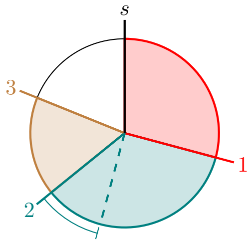
Proof.



첫 발사 지점 바로 다음에 발사한 1번 지점에 의해 구분된 1번 영역(그림에서 빨간색으로 표시)에 최소 한개의 빔이 발사되어야 합니다.

이는 어떤 구간이 1번 지점을 끝점으로 가졌고, 그 지점이 전체 구간의 끝점들 중 가장 작은 값이기 때문에 선택된 것입니다. 빨간색 영역 안에 빔이 하나도 발사되지 않았다면 해당 구간이 파괴되지 않기 때문에 모순입니다. □

Proof.



1번 영역에서 최소 한개의 빔을 발사한 논리가 그 다음 빔에 의해 생긴 영역에서도 똑같이 적용됩니다.

따라서 전체의 답은 $M - 1$ 이상이여야 합니다. □

Lemma 1 증명과정에서 보다보면, $M-1$ 개의 영역에는 빔이 무조건 한개 있어야한다는 내용이 있습니다.
전체 구간 끝점의 개수가 N 개 이므로 비둘기집의 원리로 첫 발사 지점 후보를 $N/(M-1)$ 개 이하로 추출해낼 수 있습니다.
그리고 Lemma 2에 의해서 이들 중 하나가 정답이 됩니다.
이대로 풀어도 정답이 나옵니다.

하지만 코드를 조금 더 간단하게 만들 수 없을까 싶어서 고민하다보니
Lemma 3 ~ Lemma 6 까지 확장을 할 수 있었습니다.

최종적으로 총 시간복잡도는 $\mathcal{O}(N \log N)$ 입니다.

출제자는 사전지식이 필요없는 풀이를 증명하는데 2~3주가 걸렸지만
응시자 여러분들은 다 젊고 똑똑하니까 5시간안에 증명해내거나, 혹은 증명을 못했더라도
믿음을 가지고 제출을 해볼 것이라 생각했기 때문에 맘편히 출제했습니다.

J. 관광 사업

`divide_and_conquer`
출제진 의도 – **Challenging**

- 제출 25번, 정답 3팀 (정답률 20.00%)
- 처음 푼 팀: **여기가월파2020인가요** (월파, 치러, 왔어요), 146분
- 출제자: `moonrabbit2`

- 정점 u 를 잡고, u 를 지나는 경로만 고려해 봅시다. u 는 센트로이드로 잡읍시다.
- D_v 를 u 에서부터 거리라 하면, $(D_v + D_w)(C_v + C_w)$ 의 최댓값을 구하면 됩니다.
- 단, v 와 w 는 같은 서브트리에서 오면 안 되고, 다른 집합에 있어야 합니다. u 또한 서브트리로 취급합시다.
- 위 조건들을 무시하면, $(-D_v, -C_v)$ 중 하나를 왼쪽 아래 꼭지점으로, (D_w, C_w) 중 하나를 오른쪽 위 꼭지점으로 하는 직사각형의 최대 넓이를 구하는 문제가 됩니다. 이는 전처리 후 분할 정복 최적화로 해결할 수 있음이 잘 알려져 있습니다. 17WF Money for Nothing 문제를 참조하세요.

- 다른 집합 조건은 A 집합, B 집합 각각을 따로 저장해두면 됩니다.
- 그러나 다른 서브트리 조건은 많이 어렵습니다.

- 분할 정복을 통해 다른 서브트리를 강제할 수 있습니다.
- 두 집합 S와 T를 만들어, 각 서브트리를 적절히 S와 T에 나누면, S-T 경로를 고려한 후, S와 T 각각에서 다시 해결하면 됩니다.
- S-T 경로를 고려할 땐 $S \cap A$의 정점-$T \cap B$의 정점 경로를 고려한 후, $S \cap B$의 정점-$T \cap A$의 정점 경로를 고려하면 됩니다.
- 단, 총 서브트리의 개수가 1개 이하가 되면 종료합니다.

- 새 서브트리가 들어오면, S 와 T 중 크기가 더 작은 것에 넣는다고 합시다.
- 모든 서브트리의 크기가 1이라면 이 방식을 이용하면 $\mathcal{O}\left(N \log^2 N\right)$ 로 모든 경로를 고려할 수 있습니다. $\log N$ 이 추가로 붙는 이유는 최대 넓이 직사각형을 구하는 과정이 포함되기 때문입니다.

사실, 서브트리들의 크기와 상관 없이 $\mathcal{O}\left(N \log^2 N\right)$에 할 수 있습니다!

- 가장 큰 서브트리의 크기가 $\frac{N}{2}$ 이상일 경우, 가장 먼저 큰 서브트리를 처리하면 한 집합은 그 서브트리만, 나머지 집합은 나머지 서브트리가 들어갑니다. 첫 번째 집합은 다음 재귀에서 아무 처리 없이 종료되고, 두 번째 집합은 크기가 $\frac{N}{2}$ 이하입니다.
- 가장 큰 서브트리도 크기가 $\frac{N}{2}$ 이하라면, 두 집합 각각의 크기가 최대 $\frac{3N}{4}$ 입니다. 한 서브트리를 넣을 때 두 집합간의 크기 차이가 최대 $\frac{N}{2}$ 이기 때문입니다.

이 방법을 통해 문제를 $M = \sum_{i=1}^{Q}(N_A + N_B)$이라 할 때, $\mathcal{O}\left(N \log N + M \log^3 N\right)$에 해결할 수 있습니다. 느려 보이지만 어째서인지 굉장히 빠르게 돕니다.

- 두 집합을 만드는 것은 센트로이드 분할 단계에서 함께 할 수 있습니다.
- 우선 센트로이드 u를 한 끝점으로 하는 모든 쿼리들을 처리합시다.
- 이제 서브트리들을 적절히 집합 S와 T로 나눕니다. 새 서브트리를 집합에 넣을 때에는 둘 중 작은 크기의 집합에 넣으면 각 서브트리의 크기는 앞선 풀이와 같은 이유로 $\frac{3N}{4}$ 입니다. 센트로이드에서는 모든 서브트리의 크기가 $\frac{N}{2}$ 이하이기 때문입니다.
- 같은 방법으로 S-T 경로를 고려합니다.
- u를 두 집합 각각에 넣으면 두 집합은 연결되어 다시 트리를 이룹니다. 각 트리들에 대해 다시 해결하면 됩니다.

- u를 따로 고려하는 이유는 각 정점이 $\mathcal{O}(\log N)$번만 방문된다는 것이 사실이 아니고, 그 이유는 두 집합 모두에 들어가는 u 때문이기에 이를 미리 처리하는 것입니다. 이 처리 없이는 각 쿼리마다 u를 너무 자주 고려하게 됩니다.
- 비슷한 이유로 현재 트리에서 인접한 간선만 저장하지 않으면 센트로이드 분할 과정에서 시간 초과가 납니다.
- 쿼리를 오프라인으로 처리할 수 있으므로 센트로이드 분할 과정에서 함께 처리하는 것이 편합니다.

- $M = \sum_{i=1}^{Q}(N_A + N_B)$이라 할 때, 시간복잡도는 $\mathcal{O}\left(N \log N + M \log^2 N\right)$ 입니다.
- 트리를 거리가 유지되는 이진 트리로 만들어서 해결하는 방법도 있습니다. 같은 시간복잡도를 가지며, 상수가 많이 커서 최적화가 필요할 수 있습니다. 자세한 설명은 생략합니다.

K. 데이터 제작

constructive

출제진 의도 – **Hard**

- 제출 36번, 정답 5팀 (정답률 13.89%)
- 처음 푼 팀: **여기가월파2020인가요** (월파, 치러, 왔어요), 171분
- 출제자: jhnah917

K. 데이터 제작

문제 만들어서 Call for Tasks 제출할 때까지만 해도 정도 예상했는데, 검수진분들이 을 주셨습니다. (???)

문제를 간단하게 요약해봅시다.

- 정점이 N개, 간선이 M개, 면이 K개인 평면 그래프를 만들자. (outer face 제외)
- self loop와 multi edge가 없어야 한다.
- **(중요) 좌표 범위 제한이 있다.**

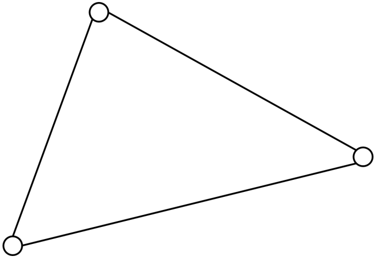
좌표 범위 제한만 없으면 아주 쉬운 문제인데, 좌표 제한 때문에 어려운 문제가 되었습니다.

평면 그래프의 V, E, F, C를 각각 정점, 간선, 면, 컴포넌트의 개수라고 하면,
$V - E + F = C + 1$이 성립합니다.
$N = V, M = E, K = F - 1$이므로 $N - M + K = C$입니다.

self loop와 multi edge가 없는 평면 그래프에서는 $M \leq 3N - 6$이 성립합니다.
그러므로 N개의 정점과 $3N - 6$개의 간선을 좁은 공간 안에 다 밀어넣는 방법을 찾은 뒤,
$M < 3N - 6$이라면 간선을 적당히 제거해주면 됩니다.

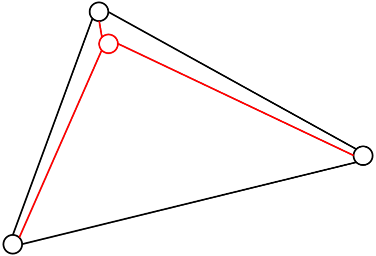
$C \neq 1$인 경우에는 정점 $C - 1$개를 따로 빼내면 $C = 1$인 상황으로 만들 수 있습니다.

좌표 범위를 신경쓰지 않고, $3N-6$ 개의 간선을 만드는 방법을 알아봅시다.

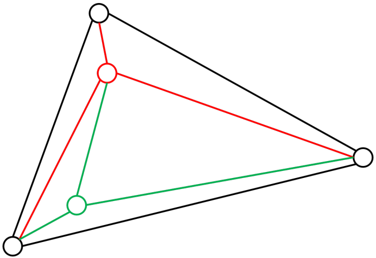


충분히 큰 삼각형에서 시작합니다.

정점은 3개, 간선은 $3(= 3 \times 3 - 6)$개 있습니다.



전 단계에서 만든 삼각형 내부에 점 하나를 찍으면
정점 1 개와 간선 3 개를 만들 수 있습니다.

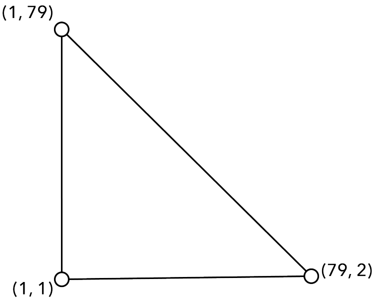


점 하나를 추가해서 만들어진 삼각형 내부에 점을 추가하면
정점 1 개와 간선 3 개를 또 만들 수 있습니다.

점이 N 개 생길 때까지 반복하면 됩니다.

모든 영역을 삼각형으로 만들어주면 간선이 $3N - 6$개가 된다는 것을 알았습니다.

이제 최대한 좁은 공간에 넣어봅시다.
정점 3개로 큰 삼각형을 만들고
그 안에 나머지 $N - 3$개의 점을 넣을 수 있어야 합니다.



큰 삼각형을 이렇게 만들어봅시다.

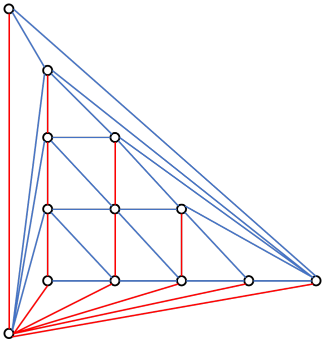
삼각형 내부에

- $y = 78$ 인 점 1개
- $y = 77$ 인 점 2개
- $\cdots$
- $y = i$ 인 점 $(79 - i)$ 개

를 넣을 수 있습니다.

총 $3 + \dfrac{78 \times 79}{2} = 3084$ 개의 점을 만들 수 있고, 이는 3000 개의 점을 넣기에 충분합니다.

$3N-6$ 개의 간선을 이용해 삼각형으로 쪼개는 방법은 아래 그림과 같이 하면 됩니다.



빨간 간선은 정점들이 연결 그래프를 이루기 위해서 꼭 필요한 간선이고
파란색 간선은 M 의 값에 따라 제거해도 되는 간선입니다.

M 값에 따라 파란색 간선을 적절히 추가/제거해서 원하는 그래프를 만들 수 있습니다.

$C \neq 1$ 인 경우에는 남은 정점 $C-1$ 개를 큰 삼각형 영역 바깥쪽에 배치해주면 됩니다.

L. 피자 배틀

dynamic_programming
출제진 의도 – **Medium**

- 제출 299번, 정답 63팀 (정답률 21.07%)
- 처음 푼 팀: **여기가월파2020인가요** (월파, 치러, 왔어요), 10분
- 출제자: 16silver

- 게임을 진행하면서 현재까지 고른 피자 조각의 크기 합의 차이는 클 수 없습니다.
- 적게 먹은 쪽이 피자 조각을 고르면서 균형이 맞춰지기 때문입니다.

현재까지 두 사람이 고른 피자 조각 크기 차이는 가장 큰 조각의 크기 M 이하이다.

Proof. 만약 어떤 순간에 X가 Y보다 $M + a$ 더 먹었다고 하자. 그렇다면 X가 가장 최근에 먹은 조각을 고르기 전에도 X는 최소 a 만큼 Y보다 앞서고 있었다. 하지만 그렇다면 Y가 자신에게 할당된 피자 조각을 먼저 다 먹었을 것이므로, X가 조각을 먼저 고르는 상황이 올 수 없다. 이는 모순이다.

- $dp(i, j, k)$: 남은 조각이 i 부터 시계 방향으로 연속한 j 개의 조각들이고, 현재까지 고른 피자 조각의 크기 합의 차가 k 일 때, 두 사람이 최선의 플레이를 했을 때 최종적으로 $(\text{선공이 먹은 양}) - (\text{후공이 먹은 양})$
  - 점화식 설명: k 에 따라 누가 피자 조각을 고를지가 정해집니다. 선공의 입장에서는 값이 클수록 피자를 많이 먹는 거고, 후공의 입장에서는 값이 작을수록 피자를 많이 먹는 것입니다.
- 이 때 k 는 $-M$ 부터 M 의 값을 가질 수 있습니다. (사실 $-M$ 은 가질 수 없습니다)

$dp(i,j,k)$: 남은 조각이 i 부터 시계 방향으로 연속한 j 개의 조각들이고, 현재까지 고른 피자 조각의 크기 합의 차가 k 일 때, 두 사람이 최선의 플레이를 했을 때 최종적으로 (선공이 먹은 양) − (후공이 먹은 양)

$$dp\left(i,j,k\right) = \begin{cases} k & \text{if } j=0 \\ \max(dp(i+1,j-1,k+c(i)), dp(i,j-1,k+c(i+j-1))) & \text{if } j \neq 0 \text{ and } k \leq 0 \\ \min(dp(i+1,j-1,k-c(i)), dp(i,j-1,k-c(i+j-1))) & \text{if } j \neq 0 \text{ and } k > 0 \end{cases}$$

모든 index는 $\mathrm{mod}\ N$ 으로 따집니다.

- 시간복잡도는 $\mathcal{O}\left(N^2M\right)$ 입니다.
- 그대로 짜면 공간복잡도도 $\mathcal{O}\left(N^2M\right)$ 입니다.
- 여기까지만 해도 문제를 풀 수는 있습니다.
- Bottom-Up 구조로, 남은 조각 수 j 에 슬라이딩 윈도우를 적용하면 $\mathcal{O}\left(NM\right)$ 으로 줄어듭니다.
- 실제로 코드를 돌려본 결과 메모리를 최적화하니 수행 시간이 반 정도로 줄어들었습니다.