AI Network Scholarium I Solutions

Official Solutions

AI NETWORK

2023년 3월 12일

문제		의도한 난이도	출제자
A	On My Way Dorm	**Easy**	`leehosu01`
B	aFan Event Planning	**Easy**	`tteuing2`
C	Appearance of the Runo	**Medium**	`august14`
D	Singularity of the Nim	**Medium**	`august14`
E	111111111111111	**Hard**	`leehosu01`
F	Decision Tree	**Hard**	`august14`
G	MiniEgg MiniGame	**Hard**	`august14`

A. On My Way Dorm

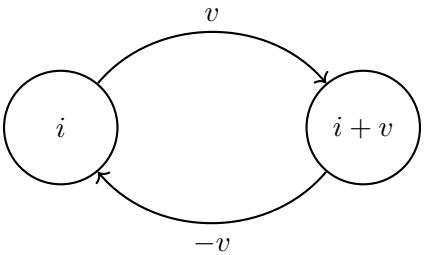
ad-hoc

출제진 의도 – **Easy**

- 제출 307번, 정답 107명 (총 정답률 34.85%, 정답자 정답률 67.30%)
- 처음 푼 사람: **cologne**, 3분
- 출제자: leehosu01

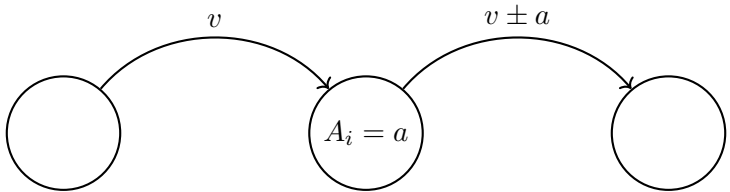
– 출근 커맨드를 준 이유가 있을 것 같습니다. 이를 적극적으로 이용해 봅시다.

- 기본적으로, i 층에서 v 의 속도를 가지면 $i+v$ 층으로 갈 수 있습니다.
- 반대로, $i+v$ 층에서 $-v$ 의 속도를 가질 수 있으면 i 층으로 돌아올 수 있습니다.



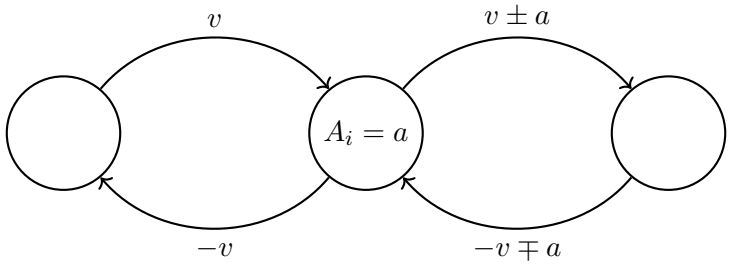
A. On My Way Dorm

– 어떤 층으로 v의 속도로 와서, $\pm a$만큼 가속해 $v \pm a$의 속도로 다음 층으로 갔다고 합시다.



A. On My Way Dorm

- 어떤 층으로 v의 속도로 와서, $\pm a$만큼 가속해 $v \pm a$의 속도로 다음 층으로 갔다고 합시다.



- $-v \mp a$의 속도로 해당 층으로 돌아왔다면, 또 $\pm a$만큼 가속해 이전 층으로 돌아갈 수 있습니다.
- 또한, 이전 층으로 갈 때의 속도도 $-v$가 되어 이 과정을 계속 반복할 수 있습니다.

- 사무실이 있는 층의 가속도를 α라고 합시다.
- 출근을 마치기 위해서는 속도가 0이 되어야 하므로, 정지 직전의 속도는 $\pm\alpha$가 됩니다.
- 이제 출근 커맨드 중 가장 마지막 단일 커맨드를 똑같이 실행하면 속도는 $\mp\alpha$가 됩니다.
- 이것은 이전 페이지에서 설명한 상황과 같습니다.
- 즉, 출근 커맨드를 거꾸로 실행한 것이 퇴근 커맨드가 됩니다.

B. aFan Event Planning

`prefix sum`, `set`
출제진 의도 – **Easy**

- 제출 280번, 정답 64명 (총 정답률 22.86%, 정답자 정답률 46.72%)
- 처음 푼 사람: **kyo20111**, 4분
- 출제자: `tteuing2`

- 1번 동작이 주어져 있지 않다면, 기초적인 누적 합 문제가 됩니다. (BOJ 11659)
- 누적 합은 $\mathcal{O}(N)$의 시간에 전처리할 수 있고, 합 구하기는 $\mathcal{O}(1)$의 시간에 처리할 수 있습니다.

- 1번 동작이 주어지면, 중간에 이벤트 토큰이 초기화되는 시점이 생깁니다.
- 그러므로 합을 구할 때, 가장 나중에 적용되는 초기화를 기준으로 합을 구합니다.

- 1번 동작이 주어지면, 중간에 이벤트 토큰이 초기화되는 시점이 생깁니다.
- 그러므로 합을 구할 때, 가장 나중에 적용되는 초기화를 기준으로 합을 구합니다.

- 초기화 시점을 적절히 관리하다가, 2번 동작이 주어질 때
  - 주어진 구간에 초기화되는 시점이 없다면, 구간 전체에 대한 합을 구합니다.
  - 초기화되는 시점이 있다면, 가장 나중에 초기화된 시점부터의 합을 구합니다.

- 1번 동작이 주어지면, 중간에 이벤트 토큰이 초기화되는 시점이 생깁니다.
- 그러므로 합을 구할 때, 가장 나중에 적용되는 초기화를 기준으로 합을 구합니다.

- 초기화 시점을 적절히 관리하다가, 2번 동작이 주어질 때
  - 주어진 구간에 초기화되는 시점이 없다면, 구간 전체에 대한 합을 구합니다.
  - 초기화되는 시점이 있다면, 가장 나중에 초기화된 시점부터의 합을 구합니다.

- 이벤트 토큰을 얻는 양에는 변화가 없으므로, 이전에 구한 누적 합을 그대로 사용 가능합니다.

- 이제, 초기화 시점을 관리하고 특정 구간에서 가장 최근의 시점을 찾는 것을 빠르게 하면 됩니다.
- C++에서는 `set` 등의 자료구조를 사용하면 둘 다 $\mathcal{O}(\lg N)$의 시간에 가능합니다.
- 전체 시간 복잡도는 $\mathcal{O}(N + Q \lg N)$이 됩니다.

C. Appearance of the Runo

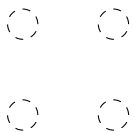
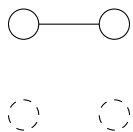
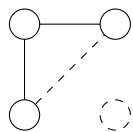
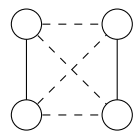
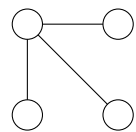
`inclusion and exclusion`
출제진 의도 – **Medium**

- 제출 83번, 정답 15명 (총 정답률 18.07%, 정답자 정답률 24.19%)
- 처음 푼 사람: **dlalswp25**, 32분
- 출제자: `august14`

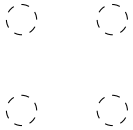
- 포함 배제의 원리를 적용하기 좋아 보입니다.
- 주어진 M 개의 쌍을 두 아이템을 같이 착용해야 한다는 규칙으로 생각합니다.
- k 개의 규칙이 동시에 적용될 때의 경우의 수를 구하고 $(-1)^k$ 를 곱해 모두 더하면 답이 됩니다.

- 루노는 같은 속성의 아이템을 여럿 착용할 수 없습니다.
- 즉, 여러 규칙에 의해 같은 속성의 다른 아이템을 여럿 착용해야 되면 그 경우의 수는 0입니다.
- 다르게 말해서, 어떤 규칙이 적용되면 특정 속성은 해당 규칙에 사용된 아이템으로 확정됩니다.

- 앞으로 각 속성은 정점으로, 각 규칙은 간선으로 생각합니다.
- 각 속성과 규칙이 확정된 상태에 따라 다음과 같이 다섯 가지로 경우를 나눌 수 있습니다.

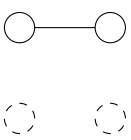
1. 
2. 
3. 
4. 
5. 

- 실선은 확정된 부분이며, 점선은 확정해야 할 부분입니다. 하나씩 살펴보겠습니다.

1. 

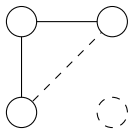
- 아무것도 확정되지 않은 상태입니다.
- 각 속성의 아이템을 착용하는 경우의 수 N^4 를 답에 더하면 됩니다.

2.



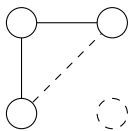
- 규칙 하나를 적용하여 두 속성의 아이템이 확정된 상태입니다.
- M 개의 규칙 각각에 대해 남은 두 속성의 아이템을 착용하는 경우의 수 N^2 가지가 있습니다.
- 포함 배제의 원리에 의한 계수는 -1 이므로 MN^2 을 답에서 빼면 됩니다.

3.



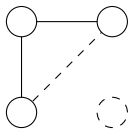
- 세 속성의 아이템이 확정된 상태입니다.
- 적용된 규칙이 길이가 2인 경로가 되거나 길이가 3인 사이클이 되는 경우입니다.
- 한 아이템을 기준으로, 이 아이템과 같이 착용 가능한 두 아이템의 쌍을 모두 봅니다.
- 이를 열거한 결과의 개수가 경로의 개수가 됩니다.

3.



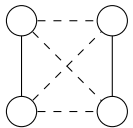
- 경로의 양 끝을 연결하는 규칙의 유무를 검사하면 사이클의 개수도 알 수 있습니다.
- 다만, 사이클에는 세 개의 경로가 포함되어 있으므로 평범하게 개수를 세면 중복이 생깁니다.
- 이를 처리하는 방법은 두 가지 정도가 있습니다.
- 첫 번째 방법은 평범하게 개수를 센 다음, 사이클에 포함된 경로의 개수인 3으로 나누는 것이고,
- 두 번째 방법은 기준인 아이템의 속성 번호가 제일 낮을때만 사이클의 개수를 세는 것입니다.

3.



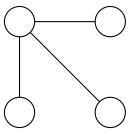
- 그렇게 구한 경로의 개수를 P, 사이클의 개수 C라고 합시다.
- 이 각각에 대해 남은 한 속성의 아이템을 착용하는 경우의 수 N 가지가 있습니다.
- 포함 배제의 원리에 의한 계수는 각각 $1, -1$ 이므로 $(P - C)N$을 답에 더하면 됩니다.

4.



- 네 속성의 아이템이 확정된 상태 두 가지 중 하나입니다.
- 속성이 하나도 겹치지 않는 두 규칙이 적용되어 있습니다.
- 그 중간을 잇는 규칙의 유무에 따라 최대 2^4 가지 경우가 생깁니다.
- 각각의 경우에 대해 포함 배제의 원리에 의한 계수를 계산해서 답에 더하면 됩니다.
- 사이클의 개수를 셀 때처럼 중복을 적절히 처리해야 합니다.

5.



- 네 속성의 아이템이 확정된 상태 두 가지 중 나머지 하나입니다.
- 속성이 하나도 겹치지 않는 두 규칙을 뽑을 수 없는 경우입니다.
- 한 아이템을 기준으로 나머지 속성마다 연결 가능한 것의 개수를 곱하면 경우의 수가 됩니다.
- 포함 배제의 원리에 의한 계수는 -1 이므로, 이를 답에서 빼면 됩니다.

- 각 경우를 계산하기 위해 필요한 시간 복잡도를 생각해보면 다음과 같습니다.
  1. N^4 을 계산하면 되므로 $\mathcal{O}(1)$
  2. MN^2 을 계산하면 되므로 $\mathcal{O}(1)$
  3. 한 아이템을 기준으로, 이 아이템과 연결된 두 규칙의 쌍을 보는 것이므로 $\mathcal{O}\left(N + M^2\right)$
  4. 두 규칙의 쌍에 대해 최대 16가지 경우가 생기므로 $\mathcal{O}\left(M^2\right)$
  5. 미리 한 아이템을 기준으로 차수들을 구하고, 그 곱을 계산하는 것이므로 $\mathcal{O}(M + N)$

- 그러므로, 총 $\mathcal{O}\left(N + M^2\right)$ 의 시간에 문제를 해결할 수 있습니다.

D. Singularity of the Nim

`game theory, ad-hoc`
출제진 의도 – **Medium**

- 제출 55번, 정답 19명 (총 정답률 34.55%, 정답자 정답률 57.58%)
- 처음 푼 사람: **angello10**, 39분
- 출제자: `august14`

- 이 게임은 `Impartial game`의 일종으로, 두 플레이어간의 차이는 선공과 후공밖에 없습니다.
- 또한 차례가 진행될 때마다 $[C_N, C_{N-1}, \cdots, C_1]$ 가 사전순으로 빨라지기만 합니다.
- 즉, 같은 상태로 다시 되돌아오는 일은 생기지 않습니다.
- 그러므로, 각 게임 상태마다 두 플레이어가 최선을 다하면 누가 이기는지 결정이 됩니다.

- 이때 최선을 다한다는 것은 다음과 같습니다.
  - 선공은 차례를 진행 가능한 모든 방법과 그때의 다음 게임 상태를 생각합니다.
  - 그중에 후공(=지금의 나)이 이길 수 있는 방법이 있다면, 그 방법을 선택하면 승리합니다.
  - 선공(=지금의 상대)이 이기는 방법밖에 없다면 패배합니다. 즉, 후공이 승리합니다.

- 그러나 이 게임은 모든 게임 상태를 따지기에는 그 개수가 너무 많습니다.

- $N = 1$일 때의 답을 귀납적으로 생각해 봅시다.

코인 개수(C_1)	승자	선공의 승리 전략 / 패배 이유
0	후공	아무것도 할 수 없습니다.
1	선공	모든 코인을 가져갑니다.
$\vdots$	$\vdots$	$\vdots$
P	선공	모든 코인을 가져갑니다.
$P+1$	후공	몇 개의 코인을 가져가도 패배하는 상황이 됩니다.

- $N = 1$일 때의 답을 귀납적으로 생각해 봅시다.

코인 개수(C_1)	승자	선공의 승리 전략 / 패배 이유
$P+1$	후공	몇 개의 코인을 가져가도 패배하는 상황이 됩니다.
$P+2$	선공	$P+1$개의 코인이 남도록 한개의 코인을 가져갑니다.
⋮	⋮	⋮
$2P+1$	선공	$P+1$개의 코인이 남도록 P개의 코인을 가져갑니다.
$2(P+1)$	후공	몇 개의 코인을 가져가도 패배하는 상황이 됩니다.

- $N = 1$일 때의 답을 귀납적으로 생각해 봅시다.

코인 개수(C_1)	승자	선공의 승리 전략 / 패배 이유
$k(P+1)$	후공	몇 개의 코인을 가져가도 패배하는 상황이 됩니다.
$k(P+1)+1$	선공	$k(P+1)$개의 코인이 남도록 한개의 코인을 가져갑니다.
$\vdots$	$\vdots$	$\vdots$
$k(P+1)+P$	선공	$k(P+1)$개의 코인이 남도록 P개의 코인을 가져갑니다.
$(k+1)(P+1)$	후공	몇 개의 코인을 가져가도 패배하는 상황이 됩니다.

- 이렇게 코인 $P+1$ 개를 단위로 같은 패턴이 반복되고, 따라서 승패는 다음과 같이 정해집니다.
  - C_1 이 $P+1$ 의 배수이면 후공이 승리합니다.
  - C_1 이 $P+1$ 의 배수가 아니면 선공이 승리합니다.
- $N > 1$ 인 경우는 어떨까요?

- 이렇게 코인 $P+1$ 개를 단위로 같은 패턴이 반복되고, 따라서 승패는 다음과 같이 정해집니다.
  - C_1 이 $P+1$ 의 배수이면 후공이 승리합니다.
  - C_1 이 $P+1$ 의 배수가 아니면 선공이 승리합니다.
- $N > 1$ 인 경우는 어떨까요?

- 사실은, 밑에서 첫 번째 칸에는 코인이 지수적으로 추가되지 않습니다.
- 그래서 $N = 1$ 때와 같은 논리를 계속해서 적용할 수 있습니다.

- C_1 이 $P+1$ 의 배수이면 후공이 승리합니다.
  - 선공이 어떤 식으로 코인을 가져가도 C_1 은 $P+1$ 의 배수가 아니게 됩니다.
- C_1 이 $P+1$ 의 배수가 아니면 선공이 승리합니다.
  - 선공은 C_1 이 $P+1$ 개의 배수가 되도록 만들면 됩니다.
- 이는 $[C_N, C_{N-1}, \cdots, C_1]$ 을 사전순으로 나열하여 수학적 귀납법으로 증명이 가능합니다.

- 생각하는 것만으로는 이 사실을 잘 알 수가 없을 수도 있습니다.
- 그럴 때는 작은 범위에 대해 실험을 해서 규칙을 발견하면 좋습니다.

E. 11111111111111

`xor basis`, `bitmask`, `dp`

출제진 의도 – **Hard**

- 제출 113번, 정답 4명 (총 정답률 3.54%, 정답자 정답률 10.00%)
- 처음 푼 사람: **cologne**, 125분
- 출제자: `leehosu01`

- XOR Basis를 알고 있다면, 이 문제가 이와 크게 관련되어 있음을 쉽게 눈치챌 수 있습니다.
- 또한, 이 풀이는 독자가 XOR Basis에 대해 알고 있다고 가정하고 작성되었습니다.

E. 111111111111111

- 현재까지 주어진 정수들을 통해 만들어지는 기저(Basis)의 개수를 B라고 합시다.
- B가 작다면 $\mathcal{O}\left(2^B\right)$의 시간에 기저들을 XOR하는 모든 경우를 보면 됩니다.
- B가 크다면 다른 방법이 필요합니다.

E. 111111111111111

- 기저를 구하는 과정은 $\mathbb{Z}_2$ 상에서의 가우스 소거법입니다.
- 그래서, 적절한 처리를 해서 각각의 기저가 특정 비트를 혼자만 1로 가지게 만들 수 있습니다.
- 또한, 비트의 순서를 섞어도 비트의 수가 바뀌지는 않으므로, 다음처럼 나타내 봅시다.

```
10000???
01000???
00100???
00010???
00001???
```

- ? 부분을 XOR해서 어떤 수를 만들 때, 기저를 최대한 많이 쓴 것이 답의 후보가 됩니다.
- ? 부분의 비트 수를 R이라고 하면, $\mathcal{O}\left(B \cdot 2^R\right)$ 시간에 동적 계획법으로 구할 수 있습니다.

E. 111111111111111

- 문제에서 주어지는 정수의 최대 크기 $111\ 111\ 111\ 111\ 111$ 은 2^{47} 보다 작습니다.
- 그러므로 기저는 최대 47 개 생길 수 있습니다. 즉, $R = 47 - B$ 입니다.

E. 111111111111111

- 문제에서 주어지는 정수의 최대 크기 $111\ 111\ 111\ 111\ 111$ 은 2^{47} 보다 작습니다.
- 그러므로 기저는 최대 47 개 생길 수 있습니다. 즉, $R = 47 - B$ 입니다.

- 이제 47 을 절반쯤으로 나눠서,
  1. $B \leq 24$ 정도일 때는 기저들을 XOR 하는 모든 경우를 봅니다.
  2. 나머지 경우는 동적 계획법을 사용합니다.
- 이 방식으로 충분히 정답으로 처리될 수 있으나 상수를 타이트하게 줄여야 할 수 있습니다.
- 그러니 더 빠른 방식을 알아봅시다.

E. 111111111111111

- 두 번째 방법이 작동되는 방법을 응용하면, 한 번에 모든 기저에 대한 답을 구할 수 있습니다.
- 먼저, 입력에 의해 추가되는 기저들을 미리 구해둡니다.
- 기저들을 순서대로 볼 때, 왼쪽 아래에 0인 비트의 삼각형이 생기도록 비트 순서를 섞습니다.

```
1???!???
01??!???
001?!???
0001!???
--------
00001???
```

- ! 인 비트들을 XOR한 결과는 가장 밑의 기저가 추가된 이후에 더 이상 바뀌지 않습니다.
- 즉, 동적 계획법에서 관리하던 최상위 비트를 없애고 적절히 비트 개수를 갱신하면 됩니다.

E. 111111111111111

- 물론 여전히 기저의 개수가 많으므로, 첫 번째 방법을 섞어야 합니다.
- 24개의 기저를 처리할 때 까지는 기저들을 XOR하는 모든 경우를 봅니다.
- 이제 남은 23개의 비트에 대한 동적 계획법을 하면 됩니다.
- 기저가 하나씩 추가될 때 마다 동적 계획법 배열이 반으로 줄어듭니다.
- 이렇게 $\mathcal{O}\left(2^{24}+2^{23}+\cdots+2^{0}\right)=\mathcal{O}\left(2^{25}\right)$의 시간에 문제를 해결할 수 있습니다.

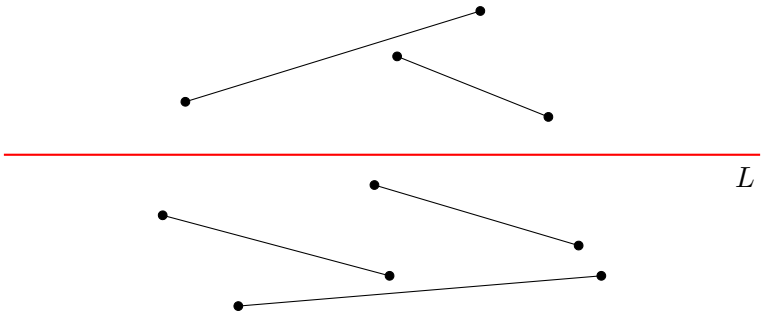
F. Decision Tree

`geometry`
출제진 의도 – **Hard**

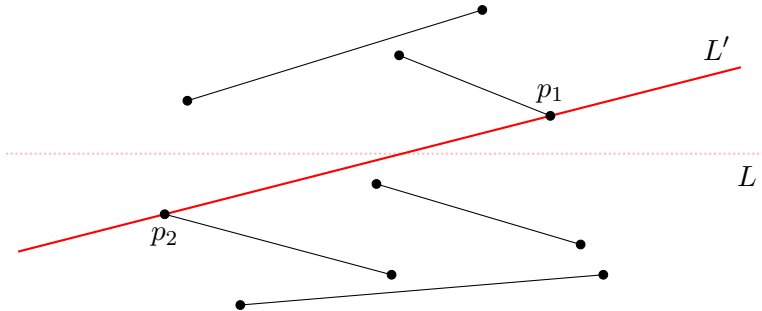
- 제출 24번, 정답 1명 (총 정답률 4.17%, 정답자 정답률 6.25%)
- 처음 푼 사람: **cologne**, 210분
- 출제자: `august14`

- 2차원 평면 위에 선분이 몇 개 있으면, 이를 임의의 두 그룹으로 나누는 직선을 구해야 합니다.
- 가능한 모든 직선을 테스트할 수는 없고, 적절한 후보들만 테스트해야 합니다.
- 어떤 후보들을 보면 좋을지 알아봅시다.

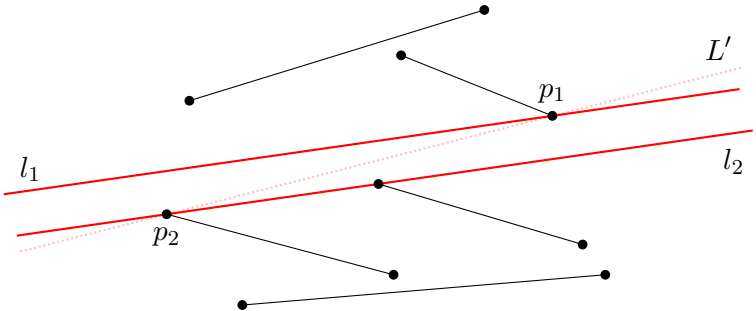
- 이미 선분들을 두 그룹으로 나누는 직선 L을 찾았다고 해 봅시다.
- 편의상 회전변환을 통해 L을 x 축에 평행한 직선으로 그렸습니다.



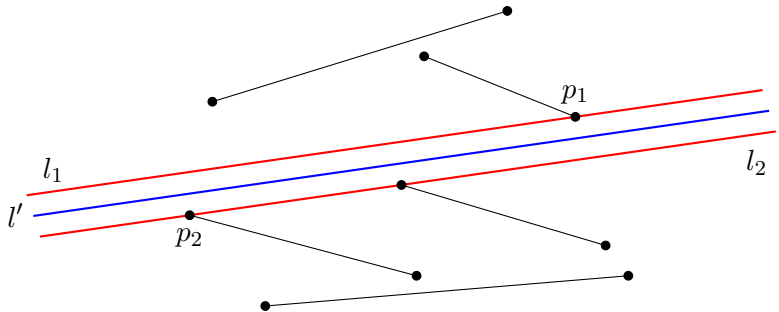
- L 을 반시계 방향으로 돌리다 선분에 막혀서 더 이상 돌릴 수 없는 상태를 L' 이라 합시다.
- L' 은 어떤 선분의 끝점 p_1, p_2 에 붙어 있어 이를 결정 트리에 사용할 수는 없습니다.



- 이제, L' 을 복사해 각각 p_1, p_2 에 고정된 두 직선 l_1, l_2 를 만듭니다.
- l_1, l_2 를 시계 방향으로 동시에 돌려서, 하나가 다른 선분에 막히면 다른 하나도 멈춥니다.



- l_1, l_2 사이에 이와 평행한 직선 l' 를 넣으면 이 직선은 결정 트리에 사용할 수 있습니다.
- 하지만 정확히 어떤 위치인 것이 좋을까요?



- $l_k := ax + by + c_k = 0$, 일반성을 잃지 않고 $c_1 < c_2$ 라고 합시다.
- c에 0.5의 간격을 줘서 l'을 다음 중 하나로 만듭니다.
  1. $ax + by + (c_1 + 0.5) = 0$
  2. $ax + by + (c_2 - 0.5) = 0$
- 이 둘은 l_1, l_2가 아무리 붙어있어도, 즉, $c_2 - c_1 = 1$ 이어도 사용할 수 있습니다.

- $l_k := ax + by + c_k = 0$, 일반성을 잃지 않고 $c_1 < c_2$ 라고 합시다.
- c에 0.5의 간격을 줘서 l'을 다음 중 하나로 만듭니다.
  1. $ax + by + (c_1 + 0.5) = 0$
  2. $ax + by + (c_2 - 0.5) = 0$
- 이 둘은 l_1, l_2가 아무리 붙어있어도, 즉, $c_2 - c_1 = 1$ 이어도 사용할 수 있습니다.

- 우리는 계수가 정수인 직선들만 출력해야 하므로, 양변에 2를 곱해서 쓰면 됩니다.
  1. $2ax + 2by + (2c_1 + 1) = 0$
  2. $2ax + 2by + (2c_2 - 1) = 0$

- l' 의 기울기는 어떤 두 끝점을 잇는 기울기 중 하나입니다.
- 그러므로, 어떤 두 끝점을 지나는 직선이 $ax + by + c = 0$ 이라고 하면
- 두 가지 후보 $2ax + 2by + (2c \pm 1) = 0$ 를 테스트하는 것으로 충분합니다.
- 즉, $\mathcal{O}\left(N^2\right)$ 개의 후보만 보면 가능한 모든 상황을 테스트할 수 있습니다.

- 각각의 후보에 대해 이 직선과 교차하는 선분의 집합을 `bitset`으로 만들어 둡니다.
- 또한 각각의 후보에 대해 나뉘게 되는 두 선분의 집합도 `bitset`으로 만들어 둡니다.
  1. 선분 그룹을 나누는 단계는 $N-1$번 있습니다.
  2. 각 단계마다, $\mathcal{O}\left(N^2\right)$개의 후보를 테스트합니다.
  3. 각 후보마다, 이 후보가 적절한지 $\mathcal{O}\left(N/64\right)$에 확인합니다.
- 그래서, 총 $\mathcal{O}\left(N^4/64\right)$의 시간에 모든 과정을 마칠 수 있습니다.

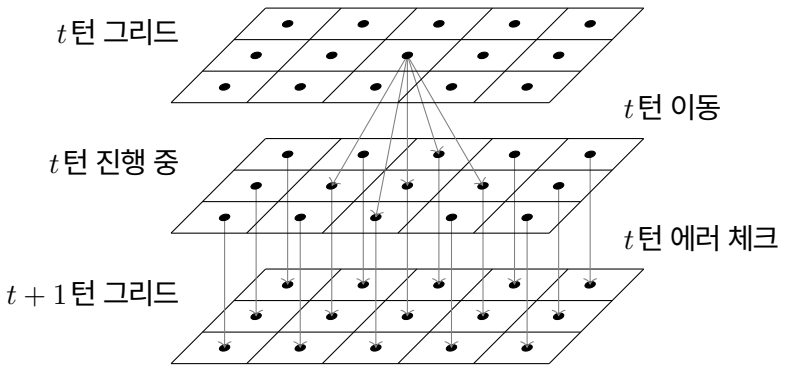
- 다만, 모든 선분이 한 직선 위에 있는 경우에 반례가 있으므로 적절한 예외 처리가 필요합니다.
- Rotating sweep line으로 $\mathcal{O}\left(N^3\right)$ 의 시간에 풀 수도 있지만, 여기서는 설명하지 않습니다.

G. MiniEgg MiniGame

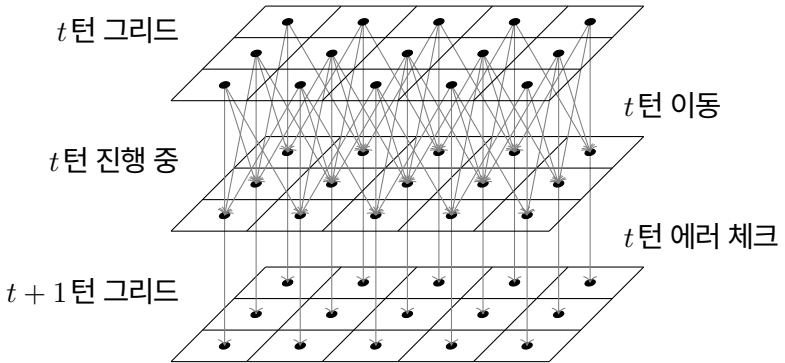
`minimum-cost flow`
출제진 의도 – **Hard**

- 제출 2번, 정답 0명 (총 정답률 0.00%, 정답자 정답률 0.00%)
- 처음 푼 사람: **N/A**
- 출제자: `august14`

- Minimum-cost flow 문제입니다.
- 사람 한 명을 하나의 플로우로 본다면 직관적인 모델링을 만들 수 있습니다.



- Minimum-cost flow 문제입니다.
- 사람 한 명을 하나의 플로우로 본다면 직관적인 모델링을 만들 수 있습니다.



- 각 미니에그는 st 턴 그리드에서 et 턴 진행 중으로 가는 비용 $(-pt)$ 인 간선이 됩니다.
- 모든 간선의 용량을 1 로 두면, 같은 칸에 여러 사람이 올 수 없도록 처리가 됩니다.
- 아직 몇 가지 문제가 남아 있습니다.

– 먼저, 두 사람이 마주보고 이동하지 못하도록 해야 하는데, 다음과 같은 방법들이 있습니다.

1. 두 사람이 마주보는 이동이 교차하는 곳에 중간 간선을 만들어 엮는다.
2. 그냥 역추적을 할 때 두 사람이 마주보고 이동하면, 이를 막고 이후의 커맨드를 교환한다.
3. 미니에그의 비용을 $(-pt) \cdot KNM$ 으로 만들고, 가만히 있는 커맨드의 비용을 -1 로 만든다.

- 그리고, 그래프의 크기가 크고 비용에 음수가 있기 때문에 SPFA를 사용하면 안 됩니다.
  1. 첫 플로우는 그래프가 DAG라는 점을 이용해 동적 계획법으로 최단경로를 찾아야 합니다.
  2. 다음부터는 포텐셜을 구할 수 있어서 Dijkstra 알고리즘을 쓸 수 있습니다.
- 포텐셜에 대한 정보는 다음 링크를 참조하세요.