# SNUPC 2020 풀이

Official Solutions

by

SNUPC 2020

# 2A. 수면 패턴

`implementation`

- 제출 95번, 정답 61명 (정답률 64.21%)
- 처음 푼 사람 : **현재익**, 4분
- 출제자 : `minty99`

수면 패턴이 뒤집어진 민티의 건강한 삶을 도와줍시다.
규칙적인 수면 패턴이 그 무엇보다 건강에 좋다는 것 잊지 마시길!

- 민티는 일주일 동안 $T$ 시간 이상 자야 합니다.
- 평일 동안 잠든 시각과 깨어난 시각의 쌍이 $N$ 개 주어집니다.
- 주말에 최소 몇 시간이나 자야 하는지를 구하는 문제입니다.

이때, 다음과 같은 부분에 주의하여야 합니다.

- 민티가 평일 동안 잠을 자지 않을 수도 있습니다.
- 주말은 $24$ 시간이 아니라 $48$ 시간입니다.
- 민티가 한 번에 잘 수 있는 시간에 제약이 없습니다.
  즉, 자는 중에 요일이 두 번 이상 넘어갈 수도 있습니다.

# 2A. 수면 패턴

민티가 평일 동안 잠을 잔 총 시간을 구해봅시다.
요일 때문에 구현이 헷갈릴 수 있지만, 다음과 같이 시간으로 바꾸면 쉽게 구할 수 있습니다.

- 월요일 $0$ 시 $\sim 23$ 시 $: 0 \sim 23$
- 화요일 $0$ 시 $\sim 23$ 시 $: 24 \sim 47$

...

이와 같이 변환하면 쉽게 총 수면 시간을 구할 수 있고, 이 값을 $T$ 에서 빼기만 하면 됩니다.
$48$ 시간보다 많이 필요한 경우나, 평일에 이미 $T$ 시간을 채운 경우 등 예외 처리에 주의하세요!

# 2B. 단어 개수 세기

`implementation, string`

- 제출 149번, 정답 53명 (정답률 35.57%)
- 처음 푼 사람 : **현재익**, 12분
- 출제자 : `kipa00`

키파는 프랑스어를 실제로도 공부하고 있습니다. 너무 어려워요…

- 프랑스어의 단어 개수를 세는 방법이 문제에 주어져 있습니다.
- 이대로 단어의 개수를 세어서 그 개수를 출력하면 되는 문제입니다.

본문에 설명된 단어의 개수를 세는 법을 정리하면 다음과 같습니다.

- 띄어쓰기와 하이픈 단위로 끊은 "부분문자열" 이 기본
- "부분문자열" 이 `c'`, `j'`, `n'`, `m'`, `t'`, `s'`, `l'`, `d'`, `qu'` 로 시작하고 `'` (어포스트로피) 뒤가 모음인 경우[1] 이 부분문자열의 단어의 개수는 $2$
- 그렇지 않으면, 이 부분문자열의 단어의 개수는 $1$



[1] TMI: 이렇게 단어가 줄어드는 현상을 리애종 (liaison) 이라고 부릅니다.

**잠깐!** 각 "부분문자열"이 두 번 이상 줄어들었을 가능성은 없을까요?

- 한 번 이상 줄어들었다면, 첫 번째 '뒤에 오는 문자가 **모음**이었다는 뜻이었습니다.
- 문자열을 한 번 분리하고 나면 앞쪽은 관심사가 아니고, 뒤쪽은 모음으로 시작합니다.
  - `c'`, `j'`, `n'`, `m'`, `t'`, `s'`, `l'`, `d'`, `qu'`가 모두 자음으로 시작합니다.
- 따라서 뒤쪽이 줄어든 형태일 수는 없습니다.

시간 제한이 매우 널널하기 때문에 앞 경우 $9$개, 뒤 모음 $6$개를 모두 문자열로 만들어서 처리해도 만점을 받을 수 있습니다.

구현을 편하게 하는 여러 가지 방법이 있습니다.

- Python의 경우 “부분문자열”을 명시적으로 구할 수 있는데, 이때 공백과 -(하이픈)이 같은 것이므로 `input().replace('-', ' ').split()` 같은 방법을 쓸 수 있습니다.
- `qu'`를 제외한 나머지는 한 글자 뒤에 어포스트로피가 바로 나오므로, 두 번째 글자가 어포스트로피임을 확인했다면 `strchr("cjnmtsld", s[0])`나 `s[0] in 'cjnmtsld'` 같은 방식으로 한 글자짜리 경우를 한꺼번에 처리할 수 있습니다.

# 2C. 넴모넴모 2020

`binary_search`

- 제출 205번, 정답 35명 (정답률 17.07%)
- 처음 푼 사람: **신원석**, 21분
- 출제자: `koosaga`, `TAMREF`

문제를 해석하면 결국 각각의 질문 $(x, y)$ 에 대해 두 값을 빠르게 구해야 한다는 것을 알 수 있습니다.

- $y$ 층에서 제거되는 넴모의 수
- $x$ 번째 칸에서 제거되는 넴모의 수

$y$ 층에서 제거되는 넴모의 수를 생각해 봅시다.

- $x > a_y$ 라면 $y$ 층에서 제거되는 넴모는 없습니다.
- $x \leq a_y$ 라면, 정확히 $(a_y - x + 1)$ 마리의 넴모가 제거됩니다.

이제, $x$ 번째 칸에서 제거되는 넴모의 수를 구해봅시다. 이는 곧 $a_i \geq x$ 를 만족하는 $y \leq i \leq N$ 의 개수와 같습니다.

- $x > a_y$ 라면, 역시 제거되는 넴모가 없습니다.
- $x \leq a_z$ 인 가장 큰 $y \leq z \leq N$ 이 존재한다면, $\boldsymbol{a_y}$ **가 단조감소하므로** $y$ 층부터 $z$ 층까지는 모두 제거되는 넴모가 생깁니다. 따라서 답은 $(z - y + 1)$ 입니다.

두 경우를 종합하면 다음과 같이 문제를 풀 수 있습니다.

- $x > a_y$ 라면 답은 무조건 $0$.
- $x \le a_y$ 라면, $x \le a_z$ 인 가장 큰 $z$ 에 대해 답은 $z - y + a_y - x$.

결국 “$x \le a_z$ 인 가장 큰 $z$” 를 구하는 게 이 문제의 핵심이 됩니다.

- 모든 $y \le z \le N$ 을 다 두드려 보면 매번 $\mathcal{O}(N)$ 의 시간이 걸리므로 효율적이지 못합니다.
- $a_y$ 가 단조감소한다는 조건을 이용하면 **이진 탐색**을 이용하여 $z$ 를 빠르게 구할 수 있습니다.
- 이진 탐색을 직접 구현해도 되고, `std::lower_bound`, `upper_bound`등 라이브러리 함수를 이용해도 좋습니다.
- 시간복잡도는 $\mathcal{O}(Q \log N)$ 입니다.

# 2D. 신기한 연산

`ad_hoc`

- 제출 22번, 정답 16명 (정답률 72.73%)
- 처음 푼 사람 : **현재익**, 39분
- 출제자 : `doju`

TMI: 문제에서 소개하는 **신기한 연산**은 **xor 연산**입니다.

```
1 xor 3 xor 2 xor 1 xor 2 = 3
```

응용 문제로 COCI 2014-2015 UTRKA(BOJ 10546)가 있습니다.

직관에 크게 의존하는 문제이기 때문에 사람마다 가지각색의 과정을 거쳐서 결론에 다다르게 됩니다.

이 풀이에서는 가능한 접근 중 하나를 소개합니다.

출력 형식에 무서운 표현이 적혀 있으므로, 먼저 **조건을 만족하는 문자열이 존재할 조건**을 생각해 보는 것이 좋습니다.

이때 효과적인 접근은 가능한 모든 입력에서 답이 존재한다고 가정하고, 최악의 경우에 대해 문제를 풀어 보는 것입니다. 그리고 보통 풀립니다.

이 문제에서 최악의 경우는 만들 수 있는 모든 구간이 전부 입력으로 주어지는 상황입니다.

## 2D. 신기한 연산

모든 올바른 구간에 대해 조건을 만족하는 문자열이 하나 있다고 가정합니다.

이 문자열의 뒤에 알파벳을 덧붙여서 더 긴 문자열을 만들어 보려고 합니다.

## 2D. 신기한 연산

조건을 만족하는 부분 문자열이 있을 때, 뒤에 알파벳을 더 붙여도 여전히 조건을 만족하도록 하는 전략은 다음의 두 가지가 있습니다.

- 홀수 번 등장하는 알파벳을 하나 붙여서 짝수 번으로 만들고, 임의의 알파벳을 하나 더 붙입니다.

a c a b a + c a

- 임의의 알파벳 하나를 두 번 연달아 붙입니다.

a c a b a + b b

지금은 각 부분 문자열에 어떤 알파벳이 홀수 번 등장하는지 모르기 때문에, 두 번째 전략을 선택하는 것이 안전해 보입니다.

알파벳을 추가함으로써 새로 생기는 부분 문자열은 잠시 무시하고, 일단 알파벳을 붙여 봅니다.

$$\boxed{\quad S \quad} + \text{xxyyzz}\ldots$$

여기서 잠깐 관찰해 보면, xxyyzz... 꼴의 문자열은 어떤 홀수 길이의 부분 문자열을 고르더라도 **홀수 번 등장하는 알파벳이 단 하나 있음**이 자명합니다!

모든 알파벳은 두 개씩 짝을 지어서 등장하는데, 부분 문자열의 시작 또는 끝 위치에서만 짝이 하나 끊어지고 한 글자만 남게 되기 때문입니다.


남은 것은 $N$ 종류의 알파벳이 모두 등장해야 한다는 조건인데, 이 조건 역시 간단하게 해결할 수 있습니다. 각 알파벳을 **순서대로** 사용하면 됩니다.

따라서 다음과 같은 문자열이 답이 됩니다.

aabbccaabbcc...

Ex. $N = 3$인 경우

직관으로 답을 찾지 못했다면, 완전 탐색으로도 접근할 수 있습니다.

어떤 문자열이 모든 구간에 대해 조건을 만족한다면, 거기서 마지막 한 글자를 지우더라도 여전히 성립합니다.

따라서 알파벳 하나를 붙이고 여전히 조건이 만족되는지 확인해 보는 과정을 반복해서 문자열의 길이를 늘려 나갈 수 있습니다.

$N = 3$에 대해 이렇게 탐색을 진행하면 앞에서 제시한 답과 같은 패턴을 빠르게 찾을 수 있습니다.

# 2E. 여우 신탁

`ad_hoc, dynamic_programming`

- 제출 121번, 정답 18명 (정답률 14.88%)
- 처음 푼 사람 : **신원석**, 41분
- 출제자 : doju

## 2E. 여우 신탁


여우는 사랑입니다.

첫 번째 여우가 받는 수를 정하면 나머지 연산을 순서대로 진행하여 마지막 여우가 받는 수를 알 수 있습니다.

그러므로 첫 번째 여우가 받을 수 있는 $A_1$ 가지 수에 대해 마지막 여우가 받는 수를 모두 계산해 보면 됩니다.

그러나 이것을 그대로 구현하면 $N \times A_1$ 번 연산을 해야 하므로 시간 초과를 받습니다.

마지막 여우가 받는 수의 **분포**가 중요하므로, 다음과 같은 문제를 생각해 볼 수 있습니다.

- 마지막 여우가 $x$ 를 받았을 때, 첫 번째 여우가 받았을 가능성이 있는 수는 몇 종류인가?

이 값을 $C_x$ 라고 하면, 모든 $x$ 에 대해 $x \times \dfrac{C_x}{A_1}$ 를 전부 더해 주면 답을 구할 수 있습니다.

두 번째 예제인 $(9, 4, 2)$ 에 대해 $C$ 배열을 구해 봅시다.

먼저 첫 번째 여우에게만 신탁을 내린 뒤에는 당연히 $C_0$ 부터 $C_8$ 까지 전부 $1$ 로 채워져 있습니다.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

## 2E. 여우 신탁

다음으로 $4$개의 수 중 하나를 원하는 여우에게 신탁을 내려 봅시다.

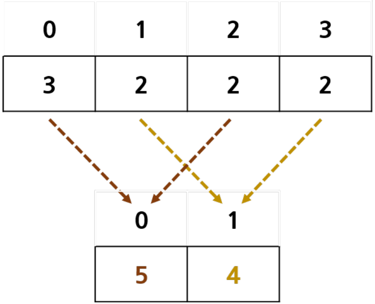
이 여우에게 신탁을 내린 뒤 새로 만들어지는 $C$는 기존의 $C$에서 인덱스를 $4$로 나눈 나머지가 서로 같은 항들을 합쳐 주면 됩니다.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 3 | 2 | 2 | 2 |

## 2E. 여우 신탁

마지막으로 $2$개의 수 중 하나를 원하는 여우에게도 신탁을 내려 줍니다.




이제 마지막 여우가 받게 되는 수의 기댓값이 $\frac{4}{9}$ 라는 것을 알 수 있습니다.

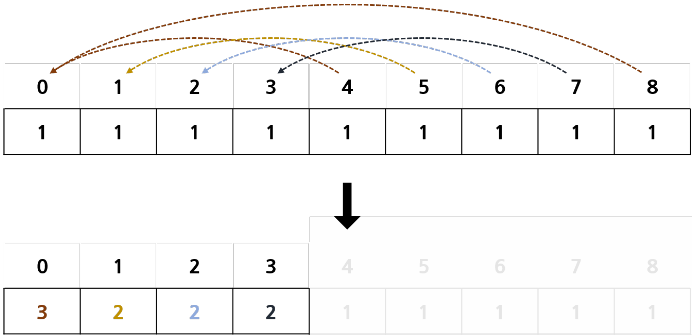
이 방법은 처음의 방법보다는 효율적인 것 같지만, $\mathcal{O}\left(\sum A_i\right)$ 번의 연산이 필요하므로 최악의 경우 크게 다르지 않습니다.

그러나 이 과정에는 한 가지 눈여겨볼 점이 있습니다.

두 번째 여우에게 신탁을 내릴 때, 첫 번째 여우가 이미 $4$ 미만의 수를 받았다면 두 번째 여우도 같은 수를 받게 된다는 점입니다.

## 2E. 여우 신탁

따라서 첫 번째 여우가 $4$ 이상의 수를 받았을 때에만 나머지를 계산해서 배열을 변경해도 같은 결과를 얻을 수 있습니다.



이 관찰은 다음 여우가 받게 되는 수의 분포를 계산할 때 $A$ **의 값이 줄어드는 만큼만** 계산하면 된다는 것을 의미합니다.

예를 들어 $(1\,000\,000, 999\,999)$ 와 같은 입력이 주어지더라도 단 한 번의 연산으로 새로운 분포를 구할 수 있습니다.

중간에 $A$ 의 값이 증가하는 경우는 분포가 전혀 변하지 않으므로 완전히 무시해도 됩니다.

$A$ 의 값은 $A_1$ 에서 시작해 $1$ 까지 줄어들 수 있으므로, 최종적으로 $\mathcal{O}(N + A_1)$ 의 시간복잡도로 문제를 해결할 수 있습니다.

답을 구할 때 많은 실수의 합을 구해야 하고, 허용하는 오차 범위가 작기 때문에 오차에 주의하는 것이 좋습니다. C/C++의 경우 `float` 자료형으로는 통과하기 어렵습니다.

# 1A/2F. 빈 문자열 만들기

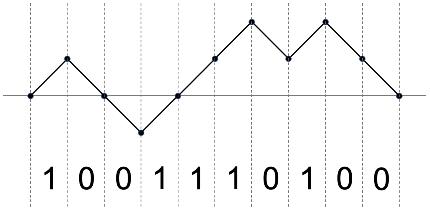
`ad_hoc`

- 제출 23번, 정답 5명 (정답률 21.74%)
- 처음 푼 사람: **김상범**, 74분
- 출제자: `ainta`

- 0과 1이 같은 횟수로 나타나는 문자열 $S$가 주어집니다.
- $S$의 길이 $2k$짜리 부분문자열이 $0^k1^k$ 형태이거나 $1^k0^k$ 형태일 때 이를 제거할 수 있습니다. ($x^k$는 문자 $x$가 $k$번 연속으로 나타나는 문자열)
- 이 때 최소 횟수의 작업으로 $S$를 빈 문자열로 만드는 방법을 출력하는 문제입니다.

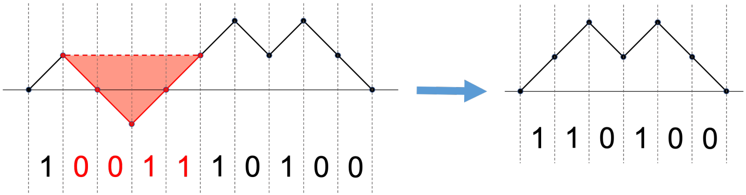
## 1A/2F. 빈 문자열 만들기

처음 원점에서 시작하여, 문자가 $1$ 일 경우 오른쪽 위 격자점으로, 문자가 $0$ 일 경우 오른쪽 아래 격자점으로 이동하는 경로를 고려해 봅시다.



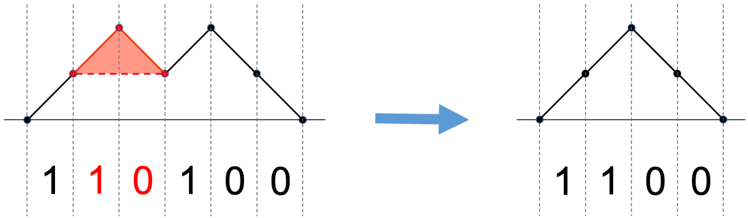

$0$ 과 $1$ 의 개수가 같으므로, 문자열 길이가 $L$ 일 때 끝점은 $(L, 0)$ 이 됩니다.

$0^k1^k$ 형태의 부분문자열을 없애는 연산은 아래 그림와 같이 내려갔다 올라가는 V자 형태의 부분 경로를 제거한 후, 그 앞과 뒤를 이어붙이는 연산이 됩니다.

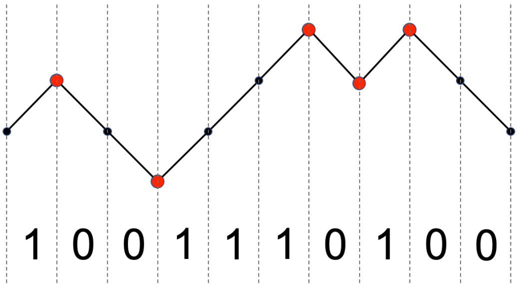


## 1A/2F. 빈 문자열 만들기

마찬가지로, $1^k0^k$ 형태의 부분문자열을 없애는 연산은 ㅅ자 형태 부분 경로를 제거한 뒤 나머지를 이어붙이는 연산이 됩니다.



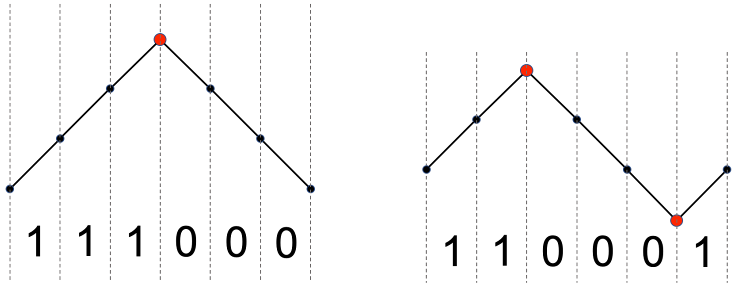
조건을 만족하는 부분문자열의 가운데에는 항상 꺾이는 점이 있을 것입니다. 이 꺾이는 점에 주목해 봅시다.



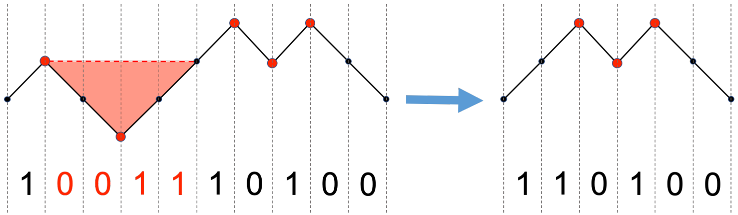
## 1A/2F. 빈 문자열 만들기

꺾이는 점이 1개면 답은 1입니다.

꺾이는 점이 2개면 답은 2입니다.



꺾이는 점이 3개 이상이면, 꺾이는 점 개수를 두 개 줄이는 방법이 항상 존재합니다.
꺾이는 점들 중에서 가장 왼쪽도, 가장 오른쪽도 아닌 어떤 점을 중심으로 하여, 없앨 수 있는 최대 길이만큼을 없애면 됩니다.



- 조건을 만족하는 부분문자열을 없앨 때, 꺾이는 점 개수는 최대 두 개 줄어듭니다. 이는 경우를 나누어 쉽게 증명할 수 있습니다.
- 따라서 꺾이는 점이 $3$ 개 이상일 때는 $2$ 개씩 줄이다가, $2$ 개 이하로 남으면 $1$ 개씩 줄이는 것이 최적입니다.
- 꺾이는 점 개수를 $D$ 라 하면 답은 $\left\lfloor \frac{D}{2} \right\rfloor + 1$ 가 됩니다.

## 1A/2F. 빈 문자열 만들기

- 역추적을 열심히 해야 합니다.
- 두 번째 꺾이는 점을 중심으로 하여 최대한으로 없애는 것을 반복하다가, 꺾이는 점이 하나만 남았을 때 전체 문자열을 없애면 됩니다.
- 이는 스택 등을 사용하면 비교적 짧게 구현할 수 있습니다.
- 문자열을 없앨 때 뒤쪽 부분 인덱스가 바뀜에 주의합시다.

# 2G. 문제지 나르기

`ad_hoc`, `backtracking`

- 제출 24번, 정답 3명 (정답률 12.50%)
- 처음 푼 사람: **심유근**, 215분
- 출제자: `koosaga`, `TAMREF`

$N$ 개의 11 차원 점 $(x_{i,1} \cdots, x_{i,11})$ 이 주어져 있을 때, $Q$ 개의 질의를 처리해야 합니다.
각 질의는 주어진 점 $(y_1, \cdots, y_{11})$ 에 대해

$$\max_{1 \le i \le N} \sum_{k=1}^{11} |x_{i,k} - y_k|$$

의 값을 구하는 것입니다. $N$ 개의 점을 전부 보면서 거리를 구하는 방법은 $\mathcal{O}(11NQ)$ 로 시간 안에 맞기 어렵습니다.

$$\max_{1\le i\le N} \sum_{k=1}^{11} |x_{i,k} - y_k|$$

위 식에서 가장 처리하기 어려운 부분은 ‘절댓값’ 입니다. 만약 절댓값이 없다면 어떻게 될까요?

$$\max_{1\le i\le N} \sum_{k=1}^{11} (x_{i,k} - y_k) = \max_{1\le i\le N} (\sum_{k=1}^{11} x_{i,k}) - \sum_{k=1}^{11} y_k$$

식에서 공통으로 등장하는 $y_k$ 를 바깥으로 꺼내두면, 단순히 $x_{i,1} + \cdots + x_{i,11}$ 의 최댓값을 저장하여 문제를 풀 수 있습니다.

절댓값이 없으면 풀기 쉽다는 데 착안하여 문제를 해결해 봅시다. 다음의 두 가지 아이디어가 필요합니다.

- $|x| = \max(x, -x)$ 이다.
- $\max(a, b) + \max(c, d) = \max(a + c, a + d, b + c, b + d)$ 이다.

이 아이디어를 이용하여 절댓값에 들어간 식 $2$개의 합을 구해보면

$$\begin{aligned}|x_{i,1} - y_1| + |x_{i,2} - y_2| &= \max(x_{i,1} - y_1, y_1 - x_{i,1}) + \max(x_{i,2} - y_2, y_2 - x_{i,2}) \\ &= \max\big(+x_{i,1} + x_{i,2} - y_1 - y_2, \\ &\qquad +x_{i,1} - x_{i,2} - y_1 + y_2, \\ &\qquad -x_{i,1} + x_{i,2} + y_1 - y_2, \\ &\qquad -x_{i,1} - x_{i,2} + y_1 + y_2\big)\end{aligned}$$

가 됩니다.

## 2G. 문제지 나르기

각각의 $4$ 가지 식에 대해서 공통으로 들어 있는 $y_1, y_2$ 를 빼내고 따로 최댓값을 구해줍시다.

- $M(++) = \max_{1 \le i \le N} (x_{i,1} + x_{i,2})$
- $M(+-) = \max_{1 \le i \le N} (x_{i,1} - x_{i,2})$
- $M(-+) = \max_{1 \le i \le N} (-x_{i,1} + x_{i,2})$
- $M(--) = \max_{1 \le i \le N} (-x_{i,1} - x_{i,2})$

$$\max(M(++) - y_1 - y_2, M(+-) - y_1 + y_2, M(-+) + y_1 - y_2, M(--) + y_1 + y_2)$$

가 원하는 답이 됩니다. $M(\pm\pm)$ 를 $\mathcal{O}(N)$ 에 전처리할 수 있으니, 대략 $\mathcal{O}(4(N+Q))$ 에 문제를 풀 수 있습니다.

$11$ 차원이라고 해서 달라지는 건 없습니다. 각각의 $2^{11} = 2\,048$ 가지 식에 대해서 공통으로 들어있는 $y_1, \cdots, y_{11}$ 을 빼내고 따로 최댓값을 구해줍시다.

- $M(\pm\pm\cdots\pm) = \pm x_{i,1} \pm x_{i,2} \cdots \pm x_{i,11}$

$$\max(M(\pm\pm\cdots\pm) \mp y_1 \mp y_2 \cdots \mp y_{11})$$

가 원하는 답이 됩니다. $M(\pm\pm\cdots\pm)$ 들은, 각각의 셀에 대해서 백트래킹을 하는 방식으로 $\mathcal{O}(2048N)$ 에 전처리할 수 있으니, 대략 $\mathcal{O}(2\,048 \cdot (N+Q))$ 에 문제를 풀 수 있습니다.

복잡도가 $\mathcal{O}(2\,048 \cdot 11 \cdot (N+Q)) \approx 10^8$ 이 되면 구현에 따라 맞을 수도, 시간 초과가 날 수도 있습니다.

$M$ 값을 전처리하는 과정에서 매번 $x_{i,j}$ 들의 합을 구하지 않고 잘 최적화하면 $\mathcal{O}(2\,048 \cdot (N+Q))$ 에 문제를 풀 수 있습니다. 설명은 생략합니다.

# 1F/2H. $2 \times M$ 타일링

`dynamic_programming`

- 제출 5번, 정답 1명 (정답률 20.00%)
- 처음 푼 사람 : **심유근**, 184분
- 출제자 : `doju`

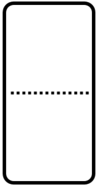
## 1F/2H. $2 \times M$ 타일링

- $2 \times M$ 격자판에 두 칸짜리 타일을 채우는 익숙한 배경의 문제입니다.
- 타일의 각 칸에는 정해진 개수만큼의 점이 찍혀 있고, 격자판을 채울 때에도 점의 개수에 대한 조건을 지켜야 합니다.

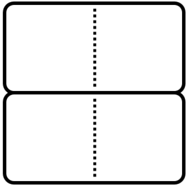
## 1F/2H. $2 \times M$ 타일링

먼저 점을 무시하고 격자판을 채워 봅시다.

이때는 다음의 두 가지 패턴을 적절히 이어 붙이면 가능한 배치를 모두 만들 수 있음이 잘 알려져 있습니다.



2×1 패턴



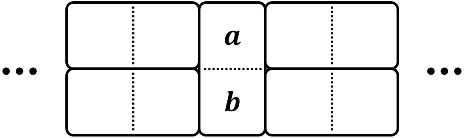
2×2 패턴

## 1F/2H. $2 \times M$ 타일링

이제 각 세로줄에 찍힌 점의 개수를 모두 $K+1$개로 맞춰 봅시다.

먼저 다음과 같은 사실을 쉽게 알 수 있습니다.

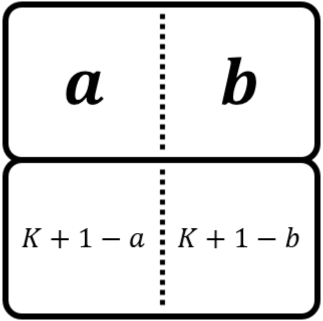
- 두 칸에 찍혀 있는 점의 개수의 합이 $K+1$인 타일은 세로로 놓을 수 있습니다.
- 그렇지 않은 타일은 세로로 놓을 수 없습니다.




$$a + b = K + 1$$

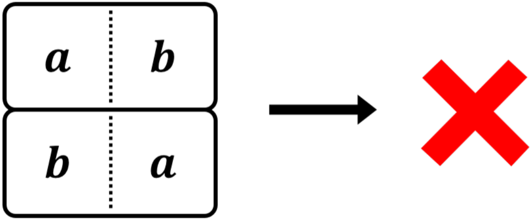
## 1F/2H. $2 \times M$ 타일링

$(a, b)$ 타일을 가로로 놓으면 반드시 $(K+1-a, K+1-b)$ 타일을 그와 짝지어서 놓아야 합니다.



이때 $a + b = K + 1$ 이라면, $(a, b)$ 타일과 짝지어지는 타일은 똑같이 $(a, b)$ 입니다. 그러나 타일 상자에는 같은 종류의 타일이 하나씩만 있으므로, 이는 불가능한 배치입니다.

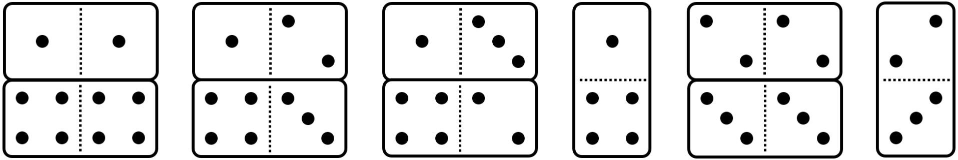
따라서 두 칸에 찍혀 있는 점의 개수의 합이 $K + 1$ 인 타일은 반드시 세로로 놓아야 합니다.



## 1F/2H. $2 \times M$ 타일링

이제 답에서 등장할 수 있는 패턴을 모두 나열할 수 있습니다.

예를 들어 $K = 4$일 때 등장할 수 있는 패턴은 다음과 같습니다. 각 패턴은 180도 돌려서 위아래를 뒤집어 사용할 수도 있습니다.



## 1F/2H. $2 \times M$ 타일링

이제 두 가로줄에 찍힌 점의 총 개수가 같도록 해 봅시다.

$K$ 의 범위가 상당히 작아 $\mathcal{O}(K!)$ 또는 $\mathcal{O}(2^K)$ 꼴의 시간복잡도를 떠올리기 쉽지만, 의도된 풀이는 **Dynamic Programming**입니다.

각 패턴을 최대 한 번씩만 사용해서 격자판을 채우는 모습에서 배낭 채우기 문제(BOJ 12865)를 떠올리면 좋습니다.

## 1F/2H. $2 \times M$ 타일링

배낭 채우기 문제의 핵심은 다음과 같습니다.

- 앞에서부터 $N$ 개의 물건 중에서 적당히 골라서
- $K$ 크기의 배낭을 채울 때
- 최대 가치는 얼마인가?

이 문제에도 같은 접근을 사용해 봅시다.

## 1F/2H. $2 \times M$ 타일링

이 문제를 풀기 위해 필요한 정보를 비슷하게 정리하면 다음과 같습니다.

- 앞에서부터 $N$ 개의 패턴 중에서 적당히 골라서
- 너비 $M$ 의 격자판을
- 윗줄에는 $A$ 개의 점이 있고
- 아랫줄에는 $B$ 개의 점이 있도록
- 채우는 것이 가능한가?

하지만 등장할 수 있는 $(N, M, A, B)$ 쌍의 개수가 어마어마하게 많기 때문에, 아직 문제를 풀 수는 없습니다.

여기서 눈여겨볼 점은 윗줄과 아랫줄의 점의 개수가 정확히 몇 개인지는 중요하지 않고, 서로 같기만 하면 된다는 점입니다.

따라서 윗줄과 아랫줄을 각각 다루지 않고 둘의 **차이**를 다루면 상태의 개수를 대폭 줄일 수 있습니다.

3년 전에도 같은 테크닉을 소개한 적이 있습니다 :P

구해야 하는 정보를 다시 정리하면 다음과 같습니다.

- 앞에서부터 $N$ 개의 패턴 중에서 적당히 골라서
- 너비 $M$ 의 격자판을
- **윗줄과 아랫줄에 찍힌 점의 개수의 차이가 $D$ 가 되도록**
- 채우는 것이 가능한가?

물론 격자판을 채울 수 있다면 답을 역추적하기 위한 단서도 같이 저장해야 합니다.

여기서 등장할 수 있는 $(N, M, D)$ 쌍의 개수는 $\mathcal{O}\left(K^3M^2\right)$ 이며, 이것이 곧 풀이의 시간복잡도가 됩니다.

처리하기 까다로운 부분이 많으므로 설계를 꼼꼼히 한 뒤 구현에 들어가는 것이 좋습니다.

# 1D/2I. 버거운 버거

`square_root_decomposition, segment_tree_with_lazy_propagation`

- 제출 9번, 정답 0명 (정답률 0.00%)
- 처음 푼 사람 : —
- 출제자 : `kipa00`

- 버거 빵에 구간 뒤집기 연산과 구간에서 버거운 버거 만들기 연산을 수행하는 자료구조 문제입니다.
- 버거운 버거는 올바른 괄호 쌍과 비슷한 조건을 만족해야 합니다.
  - 그러니 풀이에서는 버거운 버거를 $90$ 도 돌렸다고 생각하고 문제에서 주어진 대로 그냥 괄호쌍 표기를 사용하겠습니다.

먼저 버거운 버거를 만들 때 필요한 연산이 무엇인지부터 생각해 보겠습니다.

- **(**를 이전 값에 $+1$, **)**를 이전 값에 $-1$을 해서 현재 값을 생각한다고 했을 때 다음 두 연산이 필요합니다:
  - 최솟값을 구하는 연산. 여는 괄호 전에 닫는 괄호가 너무 많은 경우, 닫는 괄호를 보상할 방법은 여는 괄호를 앞에 붙이는 것뿐이기 때문입니다.
  - 맨 마지막 값을 구하는 연산. 닫는 괄호 전에 여는 괄호가 너무 많은 경우, 현재 괄호쌍이 몇 번 열렸는지를 세어서 닫아 주어야 하기 때문입니다.
- 구간에 대해 첫 번째 값을 $a$라고 하고 두 번째 값을 $b$라고 하면, 추가되는 괄호의 개수는 최소 $(-2a + b)$개입니다.

이 최솟값 조건부터 증명해 봅시다.

- 먼저, $a \leq 0$ 이고 $a \leq b$ 인 점에 주목합시다.
- $(-a)$ 가 의미하는 값은 여는 괄호와 맞지 않는 닫는 괄호의 개수입니다.
  - (가 등장했을 때 더한 $1$ 을 )에서 소모한다고 생각합시다.
- 최소한 이만큼은 여는 괄호를 반드시 앞에 붙여 주어야 합니다.
- $(b - a)$ 가 의미하는 값은 닫는 괄호와 맞지 않는 여는 괄호의 개수입니다.
  - 최솟값까지는 어쨌든 여는 괄호는 모두 소모되었을 것이고, 그 이후로 나온 여는 괄호만큼은 닫는 괄호를 소모하지 못했다는 뜻이기 때문입니다.
- 최소한 이만큼은 닫는 괄호를 반드시 뒤에 붙여 주어야 합니다.

그런데 실제로 항상 $(-2a + b)$ 번의 삽입만 이용해서 올바른 괄호 쌍을 만들 수 있습니다.

- 위 증명에서 여는 괄호는 **모두** 맨 앞에, 닫는 괄호는 **모두** 맨 뒤에 붙이면 그것이 올바른 괄호 쌍임을 증명할 수 있습니다.
- 증명은 매우 비슷하므로 대신 예시 몇 개를 들면 다음과 같습니다.
  - `((()`, $a = 0, b = 2$, `((()))`
  - `()))`, $a = -3, b = -3$, `((((())))`
  - `()))(((`, $a = -1, b = 2$, `(()))((()))`

이제 사업의 번창을 위해서 $[L, R)$ 구간만 전담하는 부하 직원을 고용한다고 합시다.

- 사장 입장에서는 주문에 $[L, R)$ 구간이 포함될 때마다 직원에게 $a$와 $b$를 물어볼 수 있습니다. 이제부터 구별을 위해 이를 $a_{L,R}$과 $b_{L,R}$로 쓰겠습니다.
- 주문 $[l, r)$에 대해 $l \leq L$이고 $R \leq r$이면, 다음과 같은 방식으로 직원을 활용할 수 있습니다:
  - $a_{l,r} = \min\{a_{l,L}, \quad b_{l,L} + a_{L,R}, \quad b_{l,L} + b_{L,R} + a_{R,r}\}$
  - $b_{l,r} = b_{l,L} + b_{L,R} + b_{R,r}$
- 확인해야 할 빵의 개수가 $(L - l) + (r - R) = (r - l) - (R - L)$개로 부하 직원이 담당하는 구간의 길이만큼 줄었습니다!

직원 입장에서는 이걸 어떻게 처리하는 게 좋을까요?

- 빵을 뒤집지도 않았는데 계속 같은 값을 물어보면 짜증이 날 겁니다! 그래서 빵을 뒤집지 않는 한, $a_{L,R}$과 $b_{L,R}$은 기억하게 될 겁니다.
- 담당하는 모든 구간의 빵을 뒤집으라고 하면, $b_{L,R}$은 $-b_{L,R}$이 될 것을 알기 때문에 $a_{L,R}$만 어떻게 해서 농땡이를 피우고 싶을 겁니다.
  - 여는 괄호가 모두 닫는 괄호, 닫는 괄호가 모두 여는 괄호가 되기 때문에, 이제 최솟값은 이전의 최댓값과 관련이 있습니다.
  - 기존의 최댓값을 $c_{L,R}$이라고 합시다. 그러면 $a_{L,R}$은 $-c_{L,R}$이 됩니다!
- 따라서 구간의 일부분을 질의하지 않는 이상, 세 값만 기억해 두면 직원은 계속 농땡이를 피울 수 있습니다.

이제 직원을 $M$ 명 고용해서 $N$ 개의 기계를 모두 다룬다고 해 봅시다.

- 직원당 다루어야 하는 기계의 수는 대략 $\frac{N}{M}$ 개입니다.
- 앞에서부터 적절히 $\frac{N}{M}$ 개씩의 기계를 담당하도록 했다고 합시다.
- 빵을 뒤집거나 질문을 했을 때 농땡이를 피우지 않는 직원은 **최대 $2$ 명입니다!**
    - 구간을 어떻게 잡든, 맨 끝점들이 포함되지 않은 직원은 구간 전체를 뒤집거나 빵을 하나도 뒤집지 않게 되기 때문입니다.

이 방법의 시간을 계산해 봅시다.

- 농땡이를 피우는 직원 : 직원 개인의 응답은 상수 시간이므로, 다 합해서 $\mathcal{O}(M)$ 만큼 걸립니다.
- 농땡이를 피우지 못하는 직원 : 최대 $\frac{N}{M}$ 개의 빵을 처리해야 하므로 $\mathcal{O}\left(\frac{N}{M}\right)$ 만큼 걸립니다.

따라서 시간은 질의당 $\mathcal{O}\left(M + \frac{N}{M}\right)$ 만큼 걸립니다.

$M$ 을 얼마로 잡아야 적절할까요? 이는 산술기하평균부등식이 알려 줍니다:

$$M + \frac{N}{M} \geq 2\sqrt{M \cdot \frac{N}{M}} = 2\sqrt{N}.$$

등호는 $M = \dfrac{N}{M}$ 일 때 성립하므로, 대략 $M = \sqrt{N}$ 으로 두면 되는 것을 알 수 있습니다.
따라서 이 방법의 질의당 시간복잡도는 $\mathcal{O}\left(\sqrt{N}\right)$ 이고, 전체 시간복잡도는 $\mathcal{O}\left(Q\sqrt{N}\right)$ 입니다.

구현에 따라 만점을 받을 수 있습니다.

사업이 더 커져서, 이제 직원들을 관리하는 직원이 필요하게 되었다고 해 봅시다.

- 말단 직원이 $D$ 단계 직원이라고 해 봅시다. 전의 경우는 $D = 1$ 인 경우였습니다.
- $0 \leq i < D$ 에 대해 각 $i$ 단계 직원이 $C$ 명씩 부하 직원을 담당한다고 합시다.
  - $C \geq 2$ 여야 하는데, $C = 1$ 이면 유일한 부하 직원이 자기와 같은 구간을 담당하게 되기 때문입니다.
- 마찬가지로, 모든 직원은 담당하는 전 구간을 질의받을 경우 **부하 직원에게 어떤 정보도 제공하지 않고** 곧바로 답변합니다.
- 말단 직원이 아닌 직원이 전 구간이 아닌 구간을 질의받을 경우 관련 부하 직원에게 물어보고 결과를 합쳐서 알려 줍니다.

먼저 직원의 수는 너무 많지만 않으면 되니 이것부터 계산해 봅시다:

$$\sum_{i=0}^{D} C^i = \frac{C^{D+1}-1}{C-1}$$

- 엄청나게 큰 값이어 보이지만 그렇지 않습니다! $C \geq 2$에 주목합시다.
- 각 직원이 담당하는 구간 길이가 $1$보다는 커야 하므로, $\frac{N}{C^D} \geq 1$에서 $C^D \leq N$을 얻습니다.

따라서,

$$\frac{C^{D+1}-1}{C-1} \leq \frac{CN-1}{C-1} < \frac{CN}{C-1} = \left(1+\frac{1}{C-1}\right)N \leq 2N.$$

그 다음으로, 시간을 계산해 봅시다. 먼저 말단 직원이 아닌 경우입니다.

- 각 단계별로 일을 시켜야 하는 직원은 최대 $2$ 명입니다! 이는 전과 똑같은 논리입니다.
  - 이 직원들은 일을 하는 데 $C$ 만큼의 시간이 걸립니다.
- 각 단계별로 일을 해야 하는 직원은 최대 $2C$ 명입니다!
  - 전 단계에서 부하 직원에게 일을 시켜야 하는 직원이 최대 $2$ 명이기 때문입니다.
- 이 말은, 최대 $2C$ 명의 직원 중 $(2C-2)$ 명은 **상수 시간에 일할 수 있다**는 뜻입니다!

따라서 어떤 질의가 들어오든 $i < D$ 에 대해 각 단계별로 걸리는 시간 $T(i)$ 는 $\mathcal{O}(C)$ 입니다.

$T(D)$ 는 좀 다른데, 말단 직원은 더 맡길 직원이 없으므로 실제로 일을 해야 하기 때문입니다.

- $D$ 단계에서도 실제로 일을 해야 하는 직원이 최대 $2$ 명인 것은 변하지 않습니다.
- 농땡이를 피우는 직원 : 최대 $(2C-2)$ 명이고 각각 상수 시간입니다.
- 농땡이를 피우지 못하는 직원 : 최대 $2$ 명이고 각각 $\dfrac{N}{C^D}$ 시간입니다.

따라서, $D$ 단계에서 걸리는 총 시간은 $\mathcal{O}\left(C+\dfrac{N}{C^D}\right)$ 입니다.

총 시간을 계산합시다.

$$\sum_{i=0}^{D} T(i) = \sum_{i=0}^{D-1} T(i) + T(D) = \mathcal{O}\left(CD + \frac{N}{C^D}\right).$$

역시 산술기하평균을 사용해 $C$에 대해 최솟값을 구할 수 있습니다.

$$\begin{aligned} CD + \frac{N}{C^D} &= C + C + \cdots + C + \frac{N}{C^D} \\ &\geq (D+1)\sqrt[D+1]{C^D \cdot \frac{N}{C^D}} = (D+1)\sqrt[D+1]{N}. \end{aligned}$$

이며, 등호는 $C = \frac{N}{C^D}$, 즉 $C = \sqrt[D+1]{N}$일 때 성립합니다. 이때 최솟값이 $C(D+1)$이 된다는 점도 유념합시다.

이제 $x = D + 1$ 이라고 놓고 다음 함수 $f$ 를 최소화하면 됩니다:

$$f(x) = xN^{1/x}.$$

다음 슬라이드에 있는 계산을 통해 $x = \ln N$ 일 때 최소화됨을 알 수 있습니다. 이때 항상 정수여야 하는 $C$ 에 먼저 주목해 봅시다.

$$C = \sqrt[D+1]{N} = N^{1/\ln N} = N^{\log_N e} = e.$$

따라서 $C = 2$ 나 $C = 3$ 이 가장 빠르다는 것을 알 수 있습니다.

## 계산입니다.

$$y = x \cdot N^{1/x}$$

$$\ln y = \ln x + \frac{1}{x} \ln N$$

$$\frac{y'}{y} = \frac{1}{x} - \frac{1}{x^2} \ln N = \frac{1}{x}\left(1 - \frac{1}{x} \ln N\right)$$

$$y' = N^{1/x} \cdot \left(1 - \frac{1}{x} \ln N\right)$$

$$y' = 0 \quad \Rightarrow \quad 1 - \frac{1}{x} \ln N = 0 \quad \Rightarrow \quad x = \ln N.$$

실제 컴퓨터가 이진수에 대한 최적화가 잘 되어 있으므로 $C = 2$를 사용한다고 합시다.[2]

- 산술기하평균 등호 조건 $C^{D+1} = N$에 의해 $x = D + 1 = \log_2 N$입니다.
- 이때의 $f$ 값은 $\log_C N \cdot C = 2 \log_2 N$입니다.

즉, 우리는 모든 질의를 $\mathcal{O}(\log N)$ 시간에 할 수 있는 방법을 만들어 냈습니다! 이 방법의 전체 시간복잡도는 $\mathcal{O}(N + Q \log N)$ 혹은 $\mathcal{O}((N + Q) \log N)$입니다.

안정적으로 만점을 받을 수 있습니다.





[2] 이론적으로 모든 연산을 같은 시간에 하는 컴퓨터는 $C = 3$이 더 빠를까요? 이건 잘 모르겠습니다.

$D = 1$일 때의 이 방법을 **Square Root Decomposition**이라고 부릅니다.

- $D = 1$일 때는 $C = \sqrt[D+1]{N} = \sqrt{N}$이 되므로, 흡사 각 구간들을 루트로 쪼갠 것처럼 보이는 점 때문에 이런 이름이 붙었습니다.
- $D = 2$인 경우 $C = \sqrt[3]{N}$이 되고, **Cube Root Decomposition**이라고 부를 수 있습니다.
  - 그렇게 자주 쓰이지는 않지만, 루트 단위로 처리하면 느리고, $D$를 더 늘리기에는 메모리가 부족할 때 쓸 수 있는 기법입니다.
  - 이 문제에서는 각 직원이 적은 수의 정보를 기억해도 되었지만, 한 직원이 기억해야 할 정보의 개수가 많아지면 $C$를 상수로 두는 방법은 메모리 때문에 쓸 수 없습니다. (BOJ 17410)

$C = 2$일 때의 이 방법을 **Segment Tree with Lazy Propagation**이라고 부릅니다.

- **Segment Tree**는 "직원들을 관리하는 직원을 여러 단계로 나누어 생각하는 것"을 얘기합니다.
  - $D$가 단순히 여러 단계인 것으로는 이 이름을 사용하지 않고, 주로 $C$가 상수일 때 이 이름을 쓰는 것 같습니다.
- **Lazy Propagation**은 "부하 직원이 알 필요가 있을 때까지 질의를 알려주지 않는 것"을 얘기합니다.

# 1A/2F. 빈 문자열 만들기

`ad_hoc`

- 제출 55번, 정답 15명 (정답률 27.27%)
- 처음 푼 사람 : **박종훈**, 39분
- 출제자 : `ainta`

# 1B. 공정한 회의

`ad_hoc`, `minimum_spanning_tree`

- 제출 26번, 정답 8명 (정답률 30.77%)
- 처음 푼 사람 : **김종범**, 40분
- 출제자 : `tlwpdus`

## 1B. 공정한 회의

결론부터 얘기하자면 조건을 만족하는 $C$ 가 존재하고 $\sum_{i=1}^{N} \sum_{j=i+1}^{N} C(i,j)$ 가 최소일 때, $C(i,j)$ 의 값은 다음처럼 구할 수 있습니다.

- 주어진 $A_i$ 와 $B_i$ 를 잇는 가중치 $D_i$ 인 양방향 간선을 그어 그래프를 만든다.
- 그래프의 **Maximum Spanning Tree**를 찾는다. (연결되지 않은 컴포넌트들은 가중치 $1$ 인 간선으로 적당히 잇는다.)
- $C(i,j)$ 는 MST에서 정점 $i$ 와 정점 $j$ 를 잇는 **경로 상의 가중치 중 최솟값**이어야 한다.

왜 그런지 증명해봅시다.

$i, j, k$ 로 이루어진 회의가 **공정할** 조건을 먼저 생각해봅시다.

$$\begin{aligned}
&\text{not}(C(i,j) < C(i,k) \text{ 이고 } C(i,j) < C(k,j)) \\
&\Leftrightarrow C(i,j) \geq C(i,k) \text{ 거나 } C(i,j) \geq C(k,j) \\
&\Leftrightarrow C(i,j) \geq \min(C(i,k), C(k,j))
\end{aligned}$$

정점 $i$ 와 정점 $j$ 사이 가중치가 $C(i,j)$ 인 완전 그래프를 생각해보면, 위 조건은 다음과 같이 풀어쓸 수 있습니다.

"$i$ 에서 $j$ 로 가는 가중치가 $i$ 에서 $\boldsymbol{k}$ **를 거쳐** $j$ 로 가는 가중치의 min 값보다 크거나 같다."

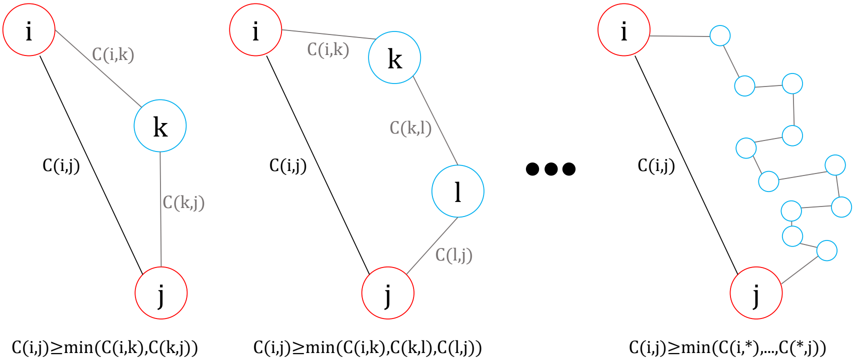
중간에 $k$ 만 거쳐야 할까요? $l$ 도 거쳐봅시다.

$$\begin{aligned} C(i,j) &\geq \min(C(i,k), C(k,j)) \\ &\geq \min(C(i,k), \min(C(k,l), C(l,j))) \\ &= \min(C(i,k), C(k,l), C(l,j)) \end{aligned}$$

다시 한 번 위 조건은 다음과 같이 풀어쓸 수 있습니다.

"$i$ 에서 $j$ 로 가는 가중치가 $i$ 에서 $\boldsymbol{k}$**,** $\boldsymbol{l}$ **을 거쳐** $j$ 로 가는 가중치의 min 값보다 크거나 같다."

이를 일반적으로 $i$에서 $j$로 가는 모든 경로에 대해 일반화할 수 있습니다. 지금까지의 상황을 그림으로 요약하면 다음과 같습니다.



일반적으로 그래프에서 $i$ 에서 $j$ 로 가는 모든 경로 상의 min 값의 max값을 구하는 문제는 Maximum Spanning Tree 위에서 $i, j$ 경로 상의 min 값을 구해서 풀 수 있습니다.

"모든 경로"에는 $i$ 에서 바로 $j$ 로 가는 경로도 포함되므로, $C$ 가 조건을 만족할 때 MST를 구해 $i, j$ 경로 상의 가중치 min 값을 구하면 이 값이 곧 $C(i, j)$ 가 됨을 알 수 있습니다.

하지만 이는 모든 $C(i, j)$ 를 동원해서 그래프를 만들었을 때의 이야기일 뿐, 주어진 $A_i, B_i, D_i$ 쌍으로 만들어진 그래프만으로 MST를 만들어도 되는지는 다른 이야기입니다.

우선 $A_i, B_i, D_i$ 로 만들어진 (양방향) 그래프가 연결 그래프라고 가정해봅시다.

주어지지 않은 간선을 사용해 spanning tree를 만드는 해가 존재한다면, 그 간선을 빼고 주어진 간선 중 하나를 잘 넣어서 더 나은 (더 작거나 같은 $\sum_{i=1}^{N} \sum_{j=i+1}^{N} C(i,j)$ 값을 가지는) spanning tree를 찾을 수 있습니다.

연결 그래프가 아닌 경우에는 $C(i,j)$ 의 가능한 최솟값인 1의 가중치로 적당히 연결해줘도 상관없습니다.

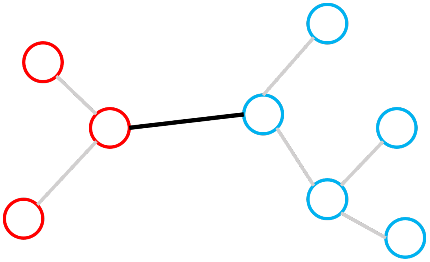
좋습니다. 이제 MST를 구해서 모든 $(i, j)$에 대해 $i$에서 $j$로 가는 경로 상 최솟값들의 합을 잘 구해야 합니다. 이를 위해서 MST를 구하는 알고리즘 중 하나인 **크루스칼**[Kruskal] **알고리즘**을 생각해봅시다.

크루스칼 알고리즘은 다음과 같이 수행됩니다.

- 간선을 가중치가 큰 순서대로 보면서
- 사이클이 생기지 않는다면 추가한다.

크루스칼 알고리즘 수행 중 검정색 간선이 추가되어 빨간 컴포넌트와 파란 컴포넌트가 합쳐졌다고 합시다. 크루스칼은 **가중치가 큰** 간선을 먼저 보기 때문에 이미 추가된 회색 간선들은 검정 간선보다 가중치가 크거나 같습니다.

따라서 빨간 정점과 파란 정점 사이의 경로 최솟값은 모두 검정 간선의 가중치가 됩니다.



그러므로 크루스칼 알고리즘을 수행하면서 Union-Find로 **컴포넌트의 크기**를 같이 관리해주고

$$\mathrm{size}(\mathrm{find}(A_i)) * \mathrm{size}(\mathrm{find}(B_i)) * D_i$$

를 union이 성공할 때마다 더해주면 됩니다.

계획이 가능한지 아닌지를 판단하는 것은 MST를 만든 이후 sparse table로 경로 최솟값을 구해 일일이 확인하는 것도 가능하지만, Union-Find를 관리할 때에 **컴포넌트와 인접한 정점 리스트**를 관리해주고 작은 것 큰 것 트릭을 사용해 합쳐줘도 확인이 가능합니다.

어떤 식으로 확인하든 총 시간복잡도는 $\mathcal{O}(M \log N)$입니다.

# 1C. 전국일주

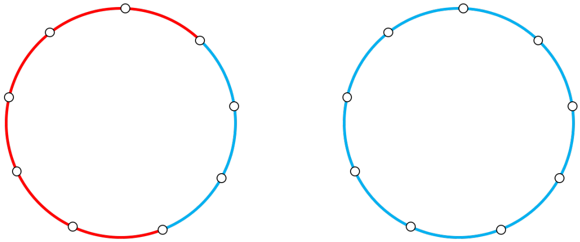
`graph`

- 제출 19번, 정답 6명 (정답률 31.58%)
- 처음 푼 사람 : **최민기**, 49분
- 출제자 : `ainta`

- $N$ 개의 마을이 있고 $N(N-1)/2$ 개의 모든 마을 쌍에 대해 도로가 있습니다.
- 각 도로는 자갈 도로 혹은 진흙 도로입니다.
- 어느 마을에서 시작하여, 모든 마을을 한 번씩 방문한 후 처음 마을로 돌아오면서 도로 종류가 최대 한 번 바뀌는 경로를 찾아야 합니다.
- 도로의 종류는 질문을 통해 알아내야 하는데, 질문은 최대 $2N$ 번 할 수 있습니다.

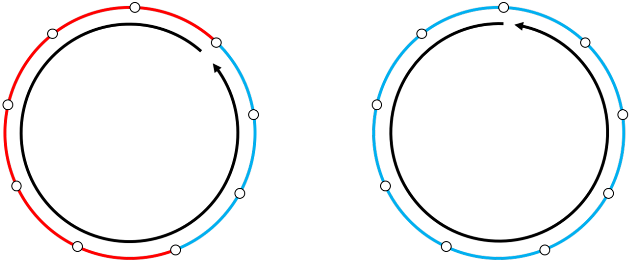
## 1C. 전국일주

각 마을을 정점으로, 자갈 도로를 붉은 간선으로, 진흙 도로는 푸른 간선으로 생각합시다.
모든 정점을 한 번씩 지나는 사이클에 대해, 사이클 안에서 같은 색 간선이 전부 연속하여 있으면 답을 찾을 수 있습니다.



## 1C. 전국일주

두 종류 간선이 사이클에서 모두 나타난 경우, 색이 바뀌는 지점에서 시작해 한 바퀴 돌면 됩니다.
간선이 모두 같은 색이면 아무 곳에서 시작해도 됩니다.

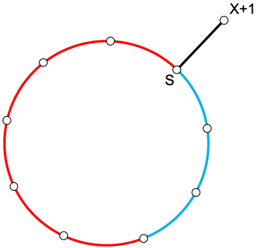


- 이런 사이클은 귀납적으로 찾을 수 있습니다.
- $1, 2, \ldots, X$ 번 정점만 고려했을 때, 같은 색 간선이 모두 연속해 있는 사이클을 찾았다고 해 봅시다.
- 이때 $(X+1)$ 번 정점을 이 사이클에 적절히 추가할 수 있습니다.

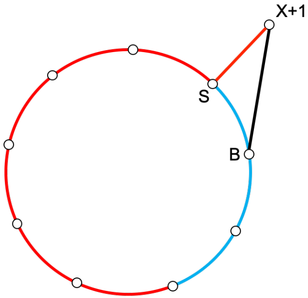
## 1C. 전국일주

$1, 2, \ldots, X$ 번을 지나면서 조건을 만족하는 사이클을 구했다고 가정합니다. 이 사이클에는 두 종류 간선이 모두 나타난다고 해 봅시다.

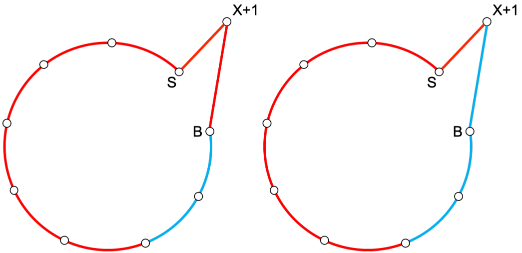
사이클에서 색이 바뀌는 지점 $S$와 $X+1$번 정점 사이 색을 질문합니다.



빨간색이면 $S$ 에서 파란 쪽으로 이어진 정점 $B$ 와 $X+1$ 번 정점 사이 색을 질문합니다.

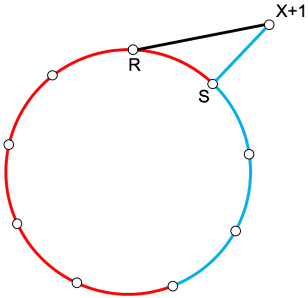


이 간선의 색과 관계없이 새로운 사이클을 아래와 같이 만들 수 있습니다.

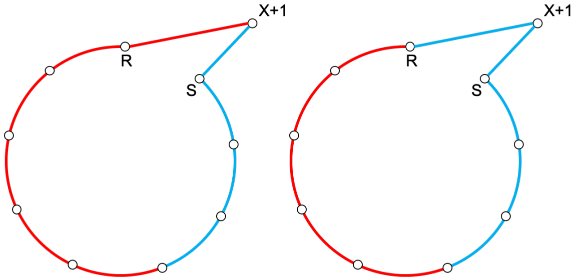


## 1C. 전국일주

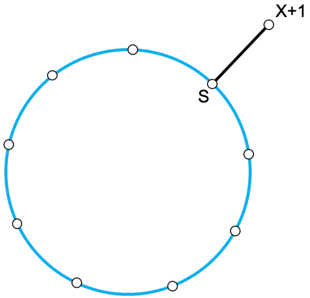
파란색이면 $S$에서 빨간 쪽으로 이어진 정점 $R$과 $X+1$번 정점 사이 색을 질문합니다.



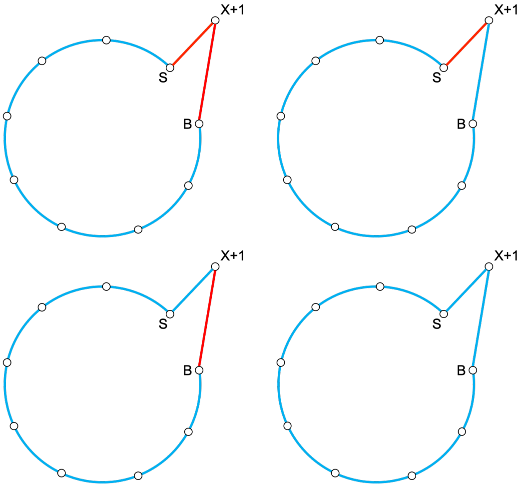
같은 방식으로 새로운 사이클을 만들 수 있습니다.



원래 사이클의 간선이 모두 같은 색일 때도 비슷하게 하면 됩니다.



# 1C. 전국일주



- 한 번의 질문으로 1번 정점과 2번 정점 사이 색을 알아냅니다.
- 그 후 3번 정점부터 $N$번 정점까지는 각각 두 번의 질문을 통해 추가할 수 있습니다.
- 즉 $(2N - 3)$번의 질문으로 문제의 조건을 만족하는 경로를 찾을 수 있습니다.

# 1D/2I. 버거운 버거

`square_root_decomposition, segment_tree_with_lazy_propagation`

- 제출 61번, 정답 20명 (정답률 32.79%)
- 처음 푼 사람 : **김창동**, 37분
- 출제자 : `kipa00`

# 1E. 직선형 분자 만들기

divide_and_conquer, link_cut_tree, graph_theory

- 제출 0번, 정답 0팀 (정답률 0.00%)
- 처음 푼 사람 : —
- 출제자 : koosaga

단순한 풀이는, 모든 $[L, R]$ 구간을 시도해 본 후, 구간에 속하는 정점과 간선들이 직선형인지를 확인하는 것입니다.

직선형인지는 어떻게 확인할까요?

직선형 그래프는 차수가 $3$ 이상인 정점이 없는 트리입니다.

## 1E. 직선형 분자 만들기

트리라는 조건 역시 풀어서 생각해 봅시다. 다음 세 조건 중 두 가지 이상의 조건을 만족하는 그래프를 트리라고 부릅니다.

- $V - E = 1$
- 연결되어 있다.
- 사이클이 없다.

저 중에 만만해 보이는 조건 두 개를 골라서 효율적으로 해결해야 합니다.

잘 모르겠으니 조금 돌아갑시다. 사실 원래 그래프에 사이클이 없으면 3번 조건은 무조건 참입니다. 고로 입력이 트리라고 가정하고, 1번과 3번 조건을 골라서 문제를 해결합니다.

우리는 다음 두 조건을 만족하는 구간 $[L, R]$ 의 개수를 알고 싶습니다:

- 구간 안에 차수가 $3$ 이상인 정점이 없음.
- $V - E = 1$

이 중 첫 번째 조건은 Two pointers를 사용해서 단순화 할 수 있고, 두 번째 조건은 Segment tree를 사용해서 단순화할 수 있습니다. 각각 어떤 방식인지 자세히 살펴보도록 합시다.

**조건 1: 구간 안에 차수가 $3$ 이상인 정점이 없음.** $f(i) =$ (구간 $[i, j]$ 에 차수가 $3$ 이상인 정점이 있는 최소 $j$) 라 정의합시다. (없을 시 $N+1$). $f(i) \leq f(i+1)$ 입니다.

$[i, f(i)]$ 구간 안에 있는 정점들의 차수를 배열에 관리합시다. 모든 정점의 차수가 $2$ 이하라면 구간을 오른쪽으로 한 칸씩 늘려줍니다. 새 정점의 인접한 간선을 순회하면서 차수를 관리합니다. $3$ 이상인 정점이 존재하면 늘리기를 멈추고, $f(i)$ 를 결정합니다.

이제 $i$ 번 정점을 제거합니다. 역시 모든 인접한 간선을 순회하며 차수를 관리합니다.

이런 식으로 $f(i)$ 를 $\mathcal{O}(n+m)$ 시간에 구할 수 있습니다. 구간 $[L, R]$ 안에 차수가 $3$ 이상인 정점이 없다는 것은 $R < f(L)$ 이라는 것입니다. 그러한 구간의 개수는 $\sum_{L=1}^{n} f(L) - L$ 입니다.

**조건 2:** $V - E = 1$. $EC[i][j] = (j - i + 1) - ([i, j]$ 구간에 완전히 포함되는 간선의 개수) 라고 합시다. 우리는 $i \leq j, EC[i][j] = 1$ 인 엔트리의 개수를 세고 싶습니다.

$EC[i][*]$ 배열을 $EC[i + 1][*]$ 로 바꿉시다. $(j - i + 1)$ 항 때문에, 배열의 모든 원소는 $1$ 씩 감소합니다. 또한, $i$ 번 정점과 인접한 정점 $i < j$ 에 대해, $j, j + 1, \ldots, N$ 번 원소가 $1$ 씩 증가합니다. 이는 구간에 특정 수를 더하고 빼는 연산입니다.

구간 덧셈 뺄셈은 할 수 있어도, $1$ 의 개수를 세는 자료 구조를 관리하기는 어려워 보입니다. 이때, 사이클이 없는 모든 그래프에 대해서 $V - E \geq 1$ **이 만족한다는 것을 관찰**합시다. 이제 $1$ 의 개수 대신 구간 최솟값과 그의 개수를 관리하면 됩니다. 세그먼트 트리에 Lazy propagation 을 사용하여 해결 가능합니다 - 지면 상 자세한 설명은 생략합니다.

**조각 맞추기.** 조건 1은 $\mathcal{O}(n+m)$ 시간에 판별할 수 있고, 조건 2는 $\mathcal{O}((n+m)\log n)$ 시간에 판별할 수 있습니다. 이제 두 조건을 한 번에 판별해야 합니다.

조건 1은 굉장히 편리하게도, 고정된 $L$에 대해 $R$의 범위를 $L \le R \le f(L) - 1$로 한정시키는 형태의 조건이 됩니다. 이는 세그먼트 트리에 적용시키기 용이합니다.

전체 구간을 쿼리하는 대신 $[L, f(L) - 1]$ 구간에 대해서 쿼리합시다.

이렇게 트리 케이스에 대한 문제가 $\mathcal{O}((n+m)\log n)$에 해결됩니다.

원래 문제로 돌아옵시다. 이제 여기에 **연결 조건** 아니면 **사이클 조건**을 추가해야 합니다. 이 풀이에서는 사이클 조건을 추가하겠습니다. $g(i) =$ (구간 $[i, j]$ 에 사이클이 있는 최소 $j$) 라 정의합시다. (없을 시 $N + 1$).

$g$ 를 계산할 수 있으면, 아까 구한 세그먼트 트리에서 쿼리를 하는 구간을 $[L, \min(f(L), g(L)) - 1]$ 으로 바꿔주면 됩니다. 아까 세그먼트 트리를 쓸 때, 사이클이 없는 모든 그래프라고 가정한 것을 일반 그래프에서도 자연스럽게 사용할 수 있습니다.

$g$ 역시 $g(i) \leq g(i + 1)$ 를 만족합니다. 이 점을 활용해서 크게 두 가지 방식의 풀이를 생각할 수 있습니다.

**풀이 1: Link-cut tree with two pointers.** Link-cut tree (LCT) 는 그래프를 관리하는 자료 구조입니다. 간선을 추가, 제거할 수 있고, 두 정점이 연결되어 있는지를 확인할 수 있습니다. 다만, 자료 구조 안에 저장된 간선이 사이클을 이루면 안 된다는 조건이 있습니다.

1980 년대 Tarjan/Sleator가 개발한 이 자료구조는 Union Find의 상위호환입니다. 매우 유용하지만, 어려운 것으로도 악명이 높습니다.

앞서 조건 1을 사용했을 때 사용한 방법을 그대로 활용해서, $[i, f(i)]$ 구간에 있는 간선을 어떤 자료구조에 관리해줍니다.

추가 제거는 자료구조 기능을 그대로 쓰면 됩니다. 사이클 판별은, 추가하고자 하는 간선의 양 끝점이 연결되어 있다면 사이클이 있는 것입니다.

이 풀이는 **Link Cut Tree 그 자체를 제외하면** 아이디어와 구현이 모두 간단한 편입니다.

LCT가 가장 큰 문제입니다. 대회 중에 구현하긴 시간이 너무 많이 들 것 같습니다. 좋은 구현을 미리 해 두시거나, 아니면 인터넷에서 좋은 구현을 검색하시면 됩니다. 어찌됐든, LCT에 대한 연습을 해 봐야 대회 중 풀기 수월할 것 같습니다.

LCT에 대해 배우고 싶다면 topology-blog.tistory.com/5를 참고하세요.

시간 복잡도는 $\mathcal{O}\left((n+m)\log n\right)$ 입니다.

**풀이 2: Divide and conquer.** 분할 정복을 사용하시면, 상대적으로 간단한 자료 구조로도 충분합니다. 그렇다고 위 풀이보다 간단하다는 뜻은 아닙니다.

여기서는 Union-Find를 사용합니다. $f(L, R, L_{\text{opt}}, R_{\text{opt}}) = g(L), g(L+1), \ldots, g(R)$ 이 $[L_{\text{opt}}, R_{\text{opt}}]$ 구간에 있음이 보장될 때, $g(L) \ldots g(R)$ 을 계산하는 함수로 정의합시다. 이때, $[R, L_{\text{opt}}]$ 구간에 있는 간선이 Union-Find에 저장되어 있다는 전제조건이 있습니다.

$g(M)$ 을 $(R_{\text{opt}} - L_{\text{opt}}) \log N$ 시간에 구하고, 이를 토대로 $f(L, M-1, L_{\text{opt}}, g(M))$, $f(M+1, R, g(M), R_{\text{opt}})$ 을 호출하는 식으로 재귀적으로 내려갑니다. $g(i) \le g(i+1)$ 이라 이 재귀 함수는 올바릅니다. 이런 식의 분할 정복 형태를 처음 보셨다면, **Divide and conquer optimization**으로 검색해 보세요.

$g(M)$ 을 $(R_{\text{opt}} - L_{\text{opt}}) \log N$ 시간에 구하기 위해서는, $L_{\text{opt}}, \ldots, R_{\text{opt}}$ 구간을 증가시켜 나가면서 간선을 추가해 보고, 사이클이 있는지를 판별하면 됩니다... 사실 이때 인접 리스트를 순회해야 하기 때문에, 수행 시간은 $[L_{\text{opt}}, R_{\text{opt}}]$ 구간의 차수 합이 됩니다. 그래서 $g(M)$ 을 원하는 시간에 구할 수는 없으나, **큰 틀에서는** 별 상관이 없습니다. 어떤 디테일이 깨졌는지를 아는 것은 여러분의 몫으로 남기겠습니다 :)

이제 재귀 함수를 호출하기 전 조건을 만족시켜 줍시다. 대략 위와 비슷한 계산량으로 가능합니다. 재귀를 빠져나올 때는, UF에 한 변경을 전부 롤백합시다. Amortization이 안 되니 Rank compression을 써야 함에 유의하세요. 복잡도는 $\mathcal{O}\left((n+m)\log^2 n\right)$ 입니다.

지면 상 생략한 내용이 많고, 구현이 상당히 까다로운 편이라고 생각합니다. 저는 이 풀이로 구현하려고 생각했으나, 포기하고 LCT의 힘을 빌렸습니다. 행운을 빕니다!

# 1F/2H. $2 \times M$ 타일링

`dynamic_programming`

- 제출 41번, 정답 10명 (정답률 24.39%)
- 처음 푼 사람: **구준서**, 43분
- 출제자: `doju`

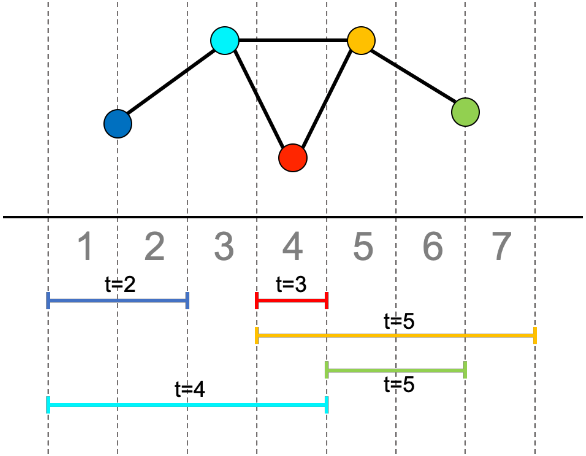
# 1G. 삼분 그래프 리턴즈

flows, dijkstra

- 제출 0번, 정답 0명 (정답률 0.00%)
- 처음 푼 사람 : –
- 출제자 : koosaga

- 구간 그래프가 주어집니다.
- 각 정점 $i$ 는 $t_i$ 의 맛을 갖고 있습니다.
- 사이클이 생기지 않도록 정점 집합을 잘 골랐을 때, 고른 정점들 맛의 합을 최대화하는 문제입니다.

## 1G. 삼분 그래프 리턴즈

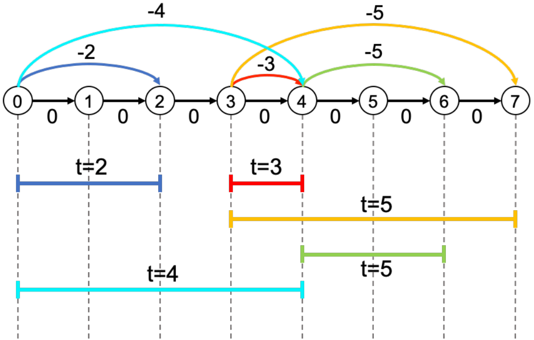


- 만약 한 점을 $3$개의 구간이 지난다면 그 구간들에 해당하는 정점들은 서로 사이클을 이룹니다.
- 즉 구간을 골랐을 때 임의의 점을 지나는 구간은 $2$개 이하여야 합니다.
- 반대로, 구간 그래프 상의 임의의 사이클을 잡아 봅시다. 사이클을 이루는 구간 중 끝점이 가장 작은 것을 골랐을 때, 이 구간의 끝점에는 최소 $3$개의 구간이 겹칩니다.
- 구간 $3$개가 겹치는 점이 없다면 그 구간들로 만든 그래프에 사이클이 없습니다.
- 즉 각 지점을 $2$개 이하의 구간이 지나도록 적당히 골라서, 고른 구간들의 $t_i$의 합을 최대화하면 됩니다.

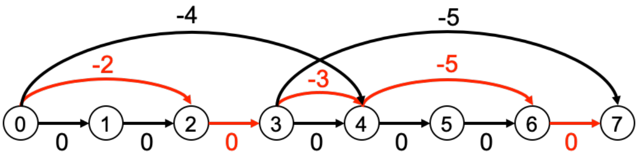
- 각 지점을 $1$ 개 이하의 구간이 지나도록 하는 문제는 잘 알려진 문제입니다.
- 이는 DP로 해결할 수 있습니다.
- 추가로 한 가지 특별한 방법을 소개합니다.

- $1 \leq i \leq 500,000$ 인 $i$ 에 대해, $(i-1)$ 번 정점에서 $i$ 번 정점으로 가는 가중치 $0$ 짜리 간선을 만듭니다.
- 맛이 $t_i$ 인 구간 $[s_i, e_i]$ 에 대해, $s_i - 1$ 번 정점에서 $e_i$ 번 정점으로 가는 가중치 $-t_i$ 짜리 간선을 만듭니다.
- $0$ 번 정점에서 $500,000$ 번 정점으로 가는 최단경로 길이가 $D$ 라면, 답은 $-D$ 입니다.

구간으로 그래프를 구성하면 아래와 같습니다.

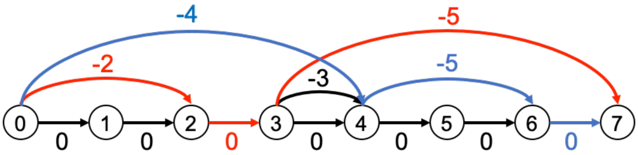


최단경로의 길이는 $-10$ 이므로, 답은 $10$ 입니다.



- 한 점을 지날 수 있는 구간 개수가 1개가 아니라 2개라면 어떨까요?
- 위와 같은 그래프에서, $0$번 정점에서 $500,000$번 정점으로 가는 **두 개**의 경로를 고릅니다. 이 때 두 경로는 가중치가 음수인 간선을 공유하지 않아야 합니다.
- 이 두 경로 길이 합의 최솟값이 $D$라면, 답은 $-D$입니다.
- 왜 그럴까요? 한 점을 지나는 구간 개수가 최대 $K$개라면, $K$개의 겹치지 않는 구간 집합으로 분할partition 할 수 있기 때문입니다.
- 여기서 구해 주는 경로 하나는 겹치지 않는 구간 집합에 대응됩니다.

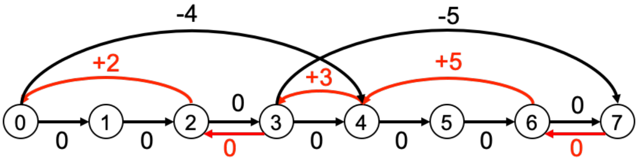
위 예시에서는 두 경로를 아래와 같이 구하는 것이 최적이고 이 때 경로 길이의 합은 $-16$ 이므로, 답은 $16$ 입니다.



- 이 문제를 minimum cost flow problem으로 모델링할 수 있습니다.
- $1 \le i \le 500,000$인 $i$에 대해, $i-1$번 정점에서 $i$번 정점으로 가는 capacity $2$, cost $0$ 간선을 만듭니다.
- 맛이 $t_i$인 구간 $[s_i, e_i]$에 대해, $s_i - 1$번 정점에서 $e_i$번 정점으로 가는 capacity $1$, cost $-t_i$ 간선을 만듭니다.
- $0$번 정점에서 $500,000$번 정점으로 가는 min cost flow를 두 번 구해 합한 것이 $D$라면, 답은 $-D$입니다.

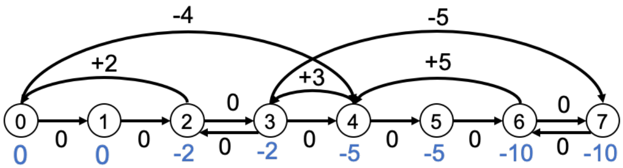
- min cost flow를 벨만 포드나 SPFA를 사용해 구하면 $\mathcal{O}\left(N^2\right)$로 시간 초과를 받습니다.
- 첫 번째 min cost flow는 DP로 $\mathcal{O}\left(N\right)$에 구할 수 있습니다.
- 첫 번째 경로의 간선들을 적절히 뒤집은 후, 두 번째 min cost flow를 빠르게 구해 봅시다.

위 예시에서 첫 번째 최단경로를 구해 뒤집으면 그래프는 아래와 같이 바뀝니다. capacity가 $0$보다 큰 간선만 나타내었고 간선에 적힌 숫자는 그 간선의 cost입니다.



- 음수 가중치 간선이 없으면 다익스트라를 쓸 수 있습니다.
- 간선의 가중치를 전부 $0$ 이상으로 만들어 봅시다.
- 첫 번째 min cost flow를 구하기 전, $0$번 정점에서 $i$번 정점까지 최단경로를 $P(i)$로 놓습니다.
- 이 $P(i)$ 값은 DP를 사용해 구할 수 있습니다.

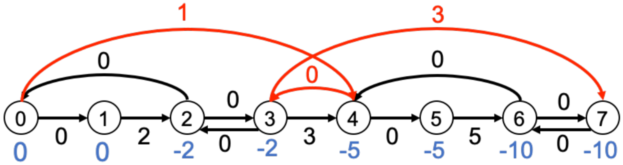
예시에서 $P(i)$ 를 나타내면 아래와 같습니다.



- $u$에서 $v$로 가는 cost $c$ 간선이 있을 때, cost를 $c$에서 $c + P(u) - P(v)$로 바꿉니다.
- 이렇게 만든 새로운 그래프는 가중치가 전부 $0$ 이상입니다.
- 방향을 뒤집지 않은 간선들은, 최단 경로의 정의에 의해 $P(v) \le P(u) + c$를 만족합니다.
- 방향을 뒤집은 간선들은, 최단 경로의 정의에 의해 $P(u) = P(v) + c$를 만족하고, 해당 간선과 뒤집은 간선 모두 가중치가 $0$입니다.
- 원래 그래프에서 $s$에서 $t$로 가는 최단경로 길이를 $D$라 하면, 새로운 그래프에서 최단경로 길이는 $D + P(s) - P(t)$입니다.

새로운 그래프는 아래와 같습니다. 모든 가중치가 $0$ 이상이므로 다익스트라를 사용할 수 있습니다.

최단경로 길이가 $4$ 이므로, 원래 그래프에서 최단경로 길이를 $x$라 하면 $x - P(7) + P(0) = 4$ 이고 $x = -6$입니다.



- 총 시간복잡도는 $\mathcal{O}(n \log n)$ 입니다.
- 간선의 가중치를 수정하는 것은 Johnson's algorithm을 알고 있다면 이해가 쉬울 것입니다.
- 일반적으로 MCMF를 구할 때에도, 매번 SPFA를 하는 것보다는 처음 한 번만 SPFA로 구하고 두 번째 최단경로부터는 다익스트라로 구할 수 있으며 이렇게 하면 $\mathcal{O}(f|E||V|)$ 에서 $\mathcal{O}(|E||V| + f|E| \log |V|)$ 로 시간복잡도를 줄일 수 있습니다.

# 1H. 계주 코스 정하기

`ad_hoc`, `plane_sweeping`

- 제출 0번, 정답 0명 (정답률 0.00%)
- 처음 푼 사람: —
- 출제자: `ainta`

- 길이 $N$ 인 배열 $A$ 와 길이 $M$ 인 배열 $B$ 가 주어집니다.
- $C_{i,j} = A_i + B_j$ 이고, $C_{i,j} \geq 0$ 인 칸만 이동할 수 있습니다.
- 시작 지점은 $1$ 열, 끝 지점은 $M$ 열의 칸입니다.
- 오른쪽 또는 아래로만 이동할 수 있을 때, 시작점에서부터 끝점까지 이동이 가능한 (시작 지점, 끝 지점) 의 쌍을 구하는 문제입니다.

- 조금 더 간단한 문제를 생각해 봅시다.
- 시작 지점이 $(1, 1)$, 끝 지점이 $(N, M)$ 으로 정해져 있을 때, 이동이 가능한지 판별해 봅시다.
- 이 문제를 $N(N+1)/2$ 번 해결 하는 것으로 원래 문제도 해결할 수 있습니다.

# 1H. 계주 코스 정하기

| B | | | | | |
|---|---|---|---|---|---|
| 5 | 1 | 3 | 4 | 5 | 6 |

| A |
|---|
| -4 |
| -2 |
| -3 |
| -1 |
| -6 |

| | | | | | |
|---|---|---|---|---|---|
| 1 | -3 | -1 | 0 | 1 | 2 |
| 3 | -1 | 1 | 2 | 3 | 4 |
| 2 | -2 | 0 | 1 | 2 | 3 |
| 4 | 0 | 2 | 3 | 4 | 5 |
| -1 | -5 | -3 | -2 | -1 | 0 |

- $\min A_i + \max B_j < 0$ 인 경우, 한 row가 이동이 불가능하므로 불가능한 경우입니다.
- $\max A_i + \min B_j < 0$ 인 경우 역시 한 column이 이동이 불가능하므로 불가능합니다.

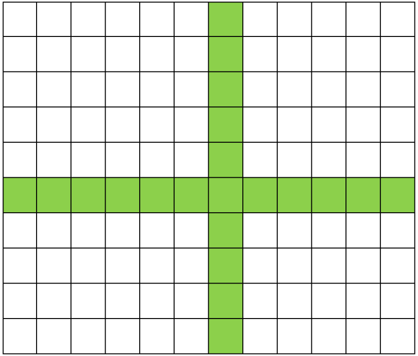
# 1H. 계주 코스 정하기

B

| | 5 | 0 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| **A** | | | | | | |
| -4 | 1 | -4 | -1 | 0 | 1 | 2 |
| -2 | 3 | -2 | 1 | 2 | 3 | 4 |
| -3 | 2 | -3 | 0 | 1 | 2 | 3 |
| -1 | 4 | -1 | 2 | 3 | 4 | 5 |
| -6 | -1 | -6 | -3 | -2 | -1 | 0 |

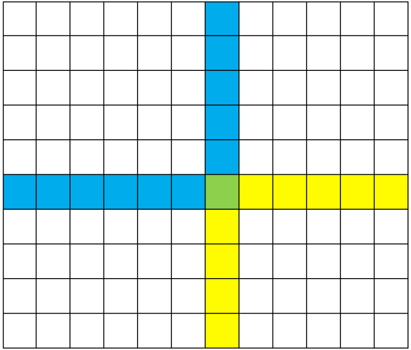
- $\min A_i + \max B_j < 0$인 경우, 한 row가 이동이 불가능하므로 불가능한 경우입니다.
- $\max A_i + \min B_j < 0$ 인 경우 역시 한 column이 이동이 불가능하므로 불가능합니다.
- 따라서, 이동이 가능한 경우에는 $\min A_i + \max B_j \geq 0$, $\max A_i + \min B_j \geq 0$
- 즉, 모든 칸이 이동이 가능한 row와 column이 적어도 하나씩 존재합니다.

따라서, 아래 그림과 같이 녹색 부분이 모두 이동 가능한 십자 모양이 존재합니다.



## 1H. 계주 코스 정하기

시작 칸에서 끝 칸으로 이동 가능할 조건은 시작 칸에서 파랑색 칸 중 하나로, 그리고 노랑색 칸 중 하나에서 끝 칸으로 이동 가능한 것입니다.

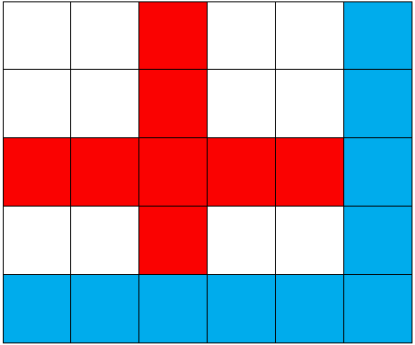


- 시작 칸에서 파랑색 칸으로 가는 것이 가능한지, 그리고 노랑색 칸 중 하나에서 끝 칸으로 가능한지 두 가지 문제를 해결하면 됩니다.
- 시작 칸에서 파랑색 칸으로 가는 것에 집중해 봅시다.
- 십자 모양이 $p$ 번째 행과 $q$ 번째 열이라고 할 때, 왼쪽 위 $(p-1) \times (q-1)$ 크기의 격자를 생각해봅시다.

- $(p-1) \times (q-1)$ 격자에서 모든 칸이 이동이 불가능한 row와 column이 모두 존재하는 경우, 시작 칸에서 파랑색 칸에 도달하는 것이 불가능합니다.

# 1H. 계주 코스 정하기

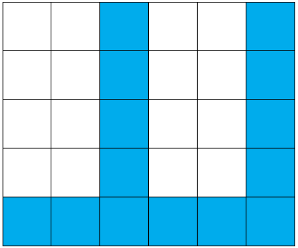
빨간색 칸을 지나갈 수 없는 경우, 시작 칸에서 파랑색 칸에 도달할 수 없습니다.

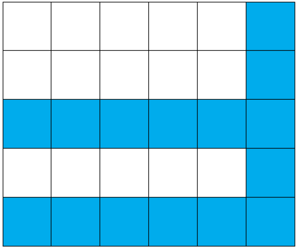


- $(p-1) \times (q-1)$ 격자에서 모든 칸이 이동이 불가능한 row와 column이 모두 존재하는 경우, 시작 칸에서 파랑색 칸에 도달하는 것이 불가능합니다.
- 그렇지 않은 경우 모든 칸이 이동이 가능한 column이 존재하거나, 모든 칸이 이동이 가능한 row가 존재합니다.
- 그 이유는 처음에 녹색 십자가 존재했던 이유와 동일합니다.
- 이 경우, 똑같은 모양의 크기가 작은 문제로 변형됩니다.

# 1H. 계주 코스 정하기

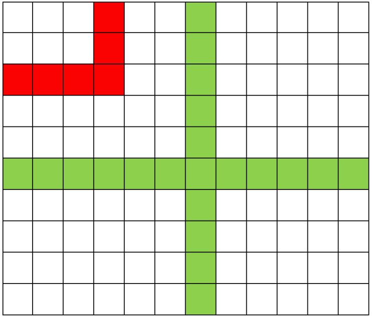
새로 생긴 파랑색 row나 column에 의해 $p$ 또는 $q$가 감소한 꼴의 똑같은 모양이 생깁니다.



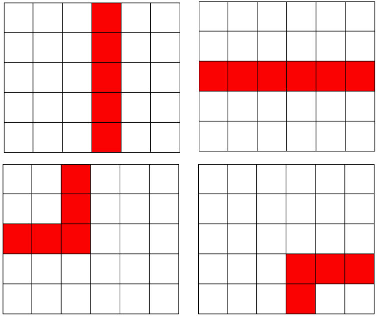


- 따라서, 파랑색 칸에 도달하는 것이 불가능한 경우는 $p$와 $q$가 감소하다가 결국에는 $(p-1) \times (q-1)$ 격자에서 모든 칸이 이동이 불가능한 row와 column이 존재할 때입니다.
- 이는 $C_{x,j} < 0$ for $1 \leq j \leq y$, $C_{i,y} < 0$ for $1 \leq i \leq x$가 성립하는 $x, y$가 존재하는 것과 동치입니다.

간단히 말해, 아래 빨간색 칸들처럼 이동할 수 없는 뒤집어진 ㄴ자 모양이 존재하는 경우에만 이동이 불가능하다는 뜻입니다.



정리하면, 이동이 불가능한 경우는 아래 4가지 패턴 뿐입니다.



- 4가지 패턴을 식으로 나타내면 다음과 같습니다.
- type 1. $\max A_i + \min B_j < 0$인 경우 (이동 불가능한 column)
- type 2. $\min A_i + \max B_j < 0$인 경우 (이동 불가능한 row)
- type 3. $C_{x,j} < 0$ for $1 \leq j \leq y$, $C_{i,y} < 0$ for $1 \leq i \leq x$가 성립하는 $x, y$가 존재하는 경우
- type 4. $C_{x,j} < 0$ for $y \leq j \leq M$, $C_{i,y} < 0$ for $x \leq i \leq N$이 성립하는 $x, y$가 존재하는 경우

- 이제 원래 문제로 돌아가 시작점이 $(b, 1)$ 이고 끝점이 $(e, M)$ 인 경우가 가능한지 판단해 봅시다.
- type 1: 시작점과 끝점에 관계없이 $\min B_j$ 는 일정하므로 $\max A_i$ 가 $-\min B_j$ 미만인 $(b, e)$ 에 대해 type 1 패턴이 존재함을 알 수 있습니다.
- type 2: 시작점과 끝점에 관계없이 $\max B_j$ 는 일정하므로 $\min A_i$ 가 $-\max B_j$ 미만인 $(b, e)$ 에 대해 type 2 패턴이 존재함을 알 수 있습니다.

- type 3
- $C_{i,1} < 0$ 인 $i$ 들에 대해, $C_{i,j} \geq 0$ 인 가장 작은 $j$ 를 $f(i)$ 로 정의합시다.
- $1 \leq j \leq f(i) - 1$ 인 $j$ 중 가장 $B_j$ 가 작은 $j$ 에 대해, $C_{x,j} \geq 0$ 인 가장 큰 $x \leq i$ 를 $g(i)$ 로 정의합시다.
- type 3 패턴이 존재할 조건은 어떤 $i$ 에 대해 $g(i) < b \leq i, i \leq e$ 가 성립하는 것입니다.
- 모든 $i$ 에 대해 $g(i)$ 를 총 $\mathcal{O}(N \log N)$ 시간에 계산할 수 있습니다.

## 1H. 계주 코스 정하기

아래 그림에서 $b \in [2, 3], e \geq 3$ 인 경우는 type 3 이 존재하여 불가능한 경우임을 알 수 있습니다.

| B | | | | | |
|---|---|---|---|---|---|
| 5 | 1 | 3 | 4 | 5 | 6 |

g(3) = 1

| A | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 6 | 2 | 4 | 5 | 6 | 7 |
| -2 | 3 | -1 | 1 | 2 | 3 | 4 |
| -6 | -1 | -5 | -3 | -2 | -1 | 0 |
| -1 | 4 | 0 | 2 | 3 | 4 | 5 |
| -3 | 2 | -2 | 0 | 1 | 2 | 3 |

f(3) = 6

- $(b, e)$ 를 좌표평면에 나타냈다고 생각해봅시다.
- type 1 이 나타나는 영역은 직사각형 영역들의 합집합으로 쉽게 표현 가능합니다.
- type 2 와 type 3 도 마찬가지입니다.
- type 4 는 type 3 과 동일한 방법으로 표현 가능합니다.

- type 1,2,3,4가 적어도 하나 이상 나타내는 영역은 직사각형 $\mathcal{O}(N)$ 개의 합집합으로 표현 가능합니다.
- 이 직사각형들의 합집합의 넓이는 plane sweeping이나 segment tree로 $\mathcal{O}(N \log N)$ 시간에 계산 가능합니다.
- 이를 통해 이동이 가능한 $(b, e)$ 쌍의 개수를 구할 수 있습니다.
- 총 시간복잡도는 $\mathcal{O}(M + N \log N)$ 입니다.

# 11. 연금술사

`greedy, binary_search`

- 제출 92번, 정답 17팀 (정답률 18.48%)
- 처음 푼 사람 : **김현수**, 20분
- 출제자 : `ainta`

최종적으로 남기게 될 광물 하나의 가중치를 $k$라고 합시다. $k$를 만들기 위해서는, $\{0, 1, \ldots, k-1\}$이 각각 최소 하나씩 필요합니다. 이 때 제공하게 될 $\{0, 1, \ldots, k-1\}$ 각각을 만드는 데도, 비슷한 논리가 반복해서 적용됩니다.

여기서 얻을 수 있는 교훈은:

- 큰 원소일수록 얻기가 지수적으로 힘들다.
- 그러니까 답도 $N$에 비해서 그렇게 크지는 않을 것이다.
- 고정된 원소 $k$를 얻을 수 있는지를 판별하는 식으로 접근해 볼 수 있다.

문제를 바꿔서, 고정된 원소 $k$를 얻을 수 있는지를 YES/NO로 판별해 봅시다.

$k$ 를 만들기 위해서는 $\{0, 1, \ldots, k-1\}$ 이 필요하고, 그 집합을 만들기 위해서는 원소 $i$ 에 대해 $\{0, 1, \ldots, i-1\}$ 이 필요하고... 가 반복됩니다.

**과정을 거꾸로 생각합시다.** $k$ 를 $\{0, 1, \ldots, k-1\}$ 로 바꾸는 식으로, 문제에서 주어진 집합 (multiset) $C$ 를 만든다고 생각해 봅시다. 이때 사용 가능한 연산은 다음 두 가지입니다.

- 1. $\{i\} \rightarrow \{i$ 미만 수 각각 1개 이상, $i$ 초과 수 총 0개 이상$\}$ $(\mathrm{MEX} > 0$에 해당)
- 2. $\{0\} \rightarrow \{0$ 초과 수 총 1개 이상$\}$ 연산 $(\mathrm{MEX} = 0$ 에 해당)

각각의 수를 꼭 1개 **초과**, 0개 **초과**로 만들 이유는 없어 보입니다. 웬만한 연산을 하면 이미 수가 많이 불어나기 때문입니다. 갖고 있는 수들을 집합 $C$가 되도록 맞춰나가야 하는데, 괜히 일을 늘리는 것 같은 느낌이 듭니다.

예외적인 경우를 제외하면, 연산을 각각의 수를 **정확히** $1$개 만들거나 안 만들고, 대신 $C$의 **부분집합을** 만들 수 있으면 끝나는 것으로 해도 됩니다.

대강의 증명은 이렇습니다. $k > 0$이고, 연산을 한 번 이상 했다고 합시다 (아닐 경우 예외 처리). $C$의 부분집합을 만드는 연산 과정의 첫 단계는 1번 연산을 사용하는 것입니다. 이 과정에서 0을 하나 더 만든 후, 그 0에 2번 연산을 하여, $C$에 없는 남은 원소들을 채워주면 됩니다. 이제 부분집합을 만드는 걸로 문제를 대체했으니, 더 많은 수를 만들 이유가 없고, 연산에 **정확히** 조건을 넣을 수 있습니다.

$k$가 주어졌을 때, 다음 두 연산을 통해서 $C$의 부분집합을 만들어야 합니다.

1. $\{i\} \to \{0, 1, \ldots, i-1\}\ (i > 0)$
2. $\{0\} \to \{i\}\ (i > 0)$

1번 연산으로 $k$를 분해하고, 작은 수가 되었을 때 2번 연산을 사용해서 수의 크기를 $C$에 맞게 불려준다고 생각할 수 있습니다.

이 말이 성립하기 위해서는, 1번 연산을 한번에 한 후, 그 다음 2번 연산을 한번에 하는 식으로, 2번 연산 다음에 1번 연산을 하는 일이 없는 최적해가 존재함을 증명해야 합니다. 이는 사실이고, exchange argument로 증명이 가능합니다.

본격적인 문제 해결에 들어가기 위한 준비작업이 다 끝났습니다. 이제 $k$ 에서 시작해서 $C$ 의 부분집합으로 원소를 줄여 줍시다.

$\{k\}$ 집합에서 시작해서, 1 번 연산을 적용해 줍시다. 관리할 $C$ 의 부분집합을 $D$ 라고 합시다. 현재 집합 안에 있는 원소 중 $D$ 에 넣을 수 있는 원소가 있다면, 1 번 연산을 더 적용할 필요 없이 넣어주면 됩니다. 그렇지 않다면, 1 번 연산을 계속 반복해야 합니다.

나중에는 1 번 연산을 더 이상 할 수 없어서, 0 여러 개가 남게 됩니다. 이들은 이제 2 번 연산을 통해서 적절히 $D$ 에 끼워줍니다. 이걸로도 부족하면, $k$ 를 만들 수 없는 것입니다.

처음에 $k$가 들어왔을 때, $D$에 넣을 수 있는지 판단합니다. 만약 잘 안 된다면, $\{0, \ldots, k-1\}$의 원소가 1개씩 생깁니다. 이제 $\{0, \ldots, k-1\}$의 최댓값인 $(k-1)$에 대해서 처리합시다. 만약 $D$에 넣었다면, $(k-2)$에 대해서 계속 반복하면 됩니다. 못 넣었다면, $(k-1)$을 없애야 합니다. 이제 $\{0, \ldots, k-2\}$ 원소가 $2$개씩 존재합니다.

이렇게 최댓값 $M$을 계속 지워주는 식으로 구현하면, $\{0, \ldots, M-1\}$ 구간의 원소는 모두 등장 횟수가 동일합니다. 이 횟수를 변수 하나로 저장하면, 집합을 쉽게 관리할 수 있습니다.

집합의 크기가 변수 하나에 저장이 안 될 정도로 매우 커질 수 있습니다. 하지만, 크기가 $|C|$를 초과하면 $C$에 넣는 것 자체가 불가능하니, 쉽게 처리 가능합니다. 2번 연산은, 마지막에 남은 0의 개수와, $|C| - |D|$를 비교하면 됩니다.

특정 수 $k$를 $C$의 부분집합으로 만들 수 있는지를 판별했습니다. 조금만 더 하면 됩니다!

**관찰.** $k \le n + 100$입니다. $k > n + 100$이라면 위 알고리즘 실행 시 카운트가 $2^{100}$을 넘어가기 때문입니다. 이제 모든 $k$를 시도해 보면 시간 복잡도는 $\mathcal{O}\left(n^2\right)$ 입니다.

**관찰.** $(k+1)$을 만들 수 있으면 $k$를 만들 수 있습니다. 첫 연산에서 $\{k+1\} \to \{0, \ldots, k\}$이 될 것이고, 이것보다는 $k$ 하나로 시작하는 것이 좋기 때문입니다. 이제 $k$를 이분 탐색으로 찾으면, 시간 복잡도는 $\mathcal{O}(n \log n)$으로 만점입니다.

마지막으로, 위에서 언급한 $k > 0, |C| = 1$ 코너 케이스를 조심하셔야 합니다. 다행히도, 예제에 이 코너 케이스가 전부 기록되어 있습니다.