# UCPC 2021 본선 풀이

Official Solutions

by

전국 대학생 프로그래밍 대회 동아리 연합

| 문제 | | 의도한 난이도 | 출제자 |
|---|---|---|---|
| **A** | A+B와 쿼리 | Medium | jh05013 |
| **B** | Distance on Triangulation 2 | Challenging | molamola |
| **C** | UCP-Clustering | Medium | jihoon |
| **D** | 츠바메가에시 | Medium | pichulia |
| **E** | 가위바위보 버블 정렬 | Hard | benedict0724 |
| **F** | 간단한 문제 | Medium | molamola |
| **G** | 돌 가져가기 2 | Hard | ainta |
| **H** | 봉화대 | Easy | evenharder |
| **I** | 붉은색 푸른색 | Challenging | tlwpdus |
| **J** | 시험 문제 출제 | Medium | jjwdi0 |
| **K** | 은퇴한 자들의 게임 | Challenging | tlwpdus |
| **L** | Make Different | Challenging | functionx |

# A. A+B와 쿼리

data_structure
출제진 의도 – **Medium**

- 제출 207번, 정답 40팀 (정답률 19.32%)
- 처음 푼 팀: **atcRXDyUNl4HGA7** (iGaM30GZzw0lTMU, G27eK6TJLM1yO0E, XYvSgfxJBXX2N0s), 22분
- 출제자: jh05013

- A와 B에 큰 노력을 기울일 필요는 없습니다. 중요한 건 C입니다.
- A 또는 B의 특정 자리를 바꾸는 것은 C의 특정 자리에 한 자리 수를 더하거나 빼는 것과 같습니다.

## A. A+B와 쿼리

| 자 리 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... |
|---|---|---|---|---|---|---|---|---|---|---|
| C | 1 | 2 | 5 | 9 | 9 | 9 | 7 | 7 | 7 | ... |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| C | 1 | 2 | 7 | 9 | 9 | 9 | 7 | 7 | 7 | ... |

- 편의를 위해 C를 반대 방향으로 써 보았습니다. 즉 맨 왼쪽이 일의 자리입니다.
- $i$번째 자리에 $d$를 더했다고 합시다.
- 만약 $d = 0$이라면 쿼리의 답은 0입니다.
- $d > 0$인데 받아올림이 일어나지 않는다면 답은 1입니다.

| 자 리 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... |
|---|---|---|---|---|---|---|---|---|---|---|
| C | 1 | 2 | 5 | 9 | 9 | 9 | 7 | 7 | 7 | ... |

↓

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| C | 1 | 2 | 2 | 0 | 0 | 0 | 8 | 7 | 7 | ... |

- 받아올림이 일어나면 어떻게 될까요?
- $i+1$ 번째부터 시작해서 연속으로 몇 개의 자리가 9인지 찾습니다. $k$ 개라고 합시다.
- 그러면 답은 $k+2$ 이고, $i+1$ 번째부터 $i+k$ 번째까지의 자리는 전부 0이 되며, $i+k+1$ 번째 자리는 1 올라갑니다.

그러므로 다음 연산을 모두 효율적으로 지원하는 자료구조가 필요합니다.

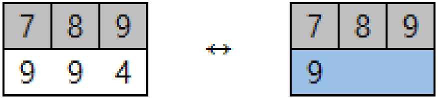
1. **연산 1:** $i$번째 자리를 특정 숫자로 바꾸기
2. **연산 2:** $i$번째부터 시작해서 연속으로 몇 개의 자리가 9인지 (0인지) 찾기
3. **연산 3:** $i$번째부터 $j$번째까지 자리가 모두 9라는 (0이라는) 가정 하에 모두 0으로 (9로) 바꾸기

다양한 방법이 있습니다. 이 문제를 해결하는 세 가지 방법을 소개합니다.

## 1. Sqrt decomposition

| 자리 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| C | 1 | 2 | 9 | [9] | | | 9 | 9 | 4 | [0] | |

- 배열을 길이 $\mathcal{O}\left(\sqrt{N}\right)$ 의 조각 $\mathcal{O}\left(\sqrt{N}\right)$ 개로 나눕니다.
- 각 조각은 다음 중 하나의 정보를 갖고 있습니다.
  - 조각 내 각각의 숫자 (위 그림에서 흰색)
  - 조각 내의 모든 숫자가 9라는 (0이라는) 정보 (위 그림에서 파란색)




**연산 1:** $i$번째 자리를 특정 숫자로 바꾸기

- 흰색 조각은 모든 자리가 9가 (0이) 되는 순간 파란색 조각으로 바뀝니다.
- 반대로, 파란색 조각은 한 자리라도 바뀌는 순간 흰색 조각이 됩니다.

| 자 리 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| C | 1 | 2 | 9 | [9] | | | 9 | 9 | 4 | [0] | |

**연산 2:** $i$번째부터 시작해서 연속으로 몇 개의 자리가 9인지 (0인지) 찾기

- 맨 첫 조각에서는 한 자리씩 봅니다.
- 파란색 조각을 만나면, 그 조각이 9인지 (0인지) 보고 한번에 건너뜁니다. 반대로 0이라면 (9라면) 바로 종료합니다.
- 흰색 조각을 만나면 한 자리씩 봅니다. 적어도 한 자리는 9가 (0이) 아니기 때문에, 그 조각에서 종료하게 될 것입니다.

- **연산 3**은 비슷하므로 생략합니다.
- 각각의 연산을 $\mathcal{O}\left(\sqrt{N}\right)$에 구현할 수 있으므로, 전체 시간 복잡도는 $\mathcal{O}\left(N+Q\sqrt{N}\right)$입니다.

## 2. Segment tree with lazy propagation

- 각 노드마다 "그 노드가 담당하는 범위의 최솟값과 최댓값"을 저장합니다.
- 이제 다음 연산을 지원하면 됩니다.
  - 특정 구간에 특정 값 더하기
  - $i$가 주어졌을 때, 구간 $[i, j]$의 최솟값이 9가 아닌 (최댓값이 0이 아닌) 가장 작은 $j \geq i$ 구하기
- 모두 $\mathcal{O}(\log N)$에 구현할 수 있습니다.
- 매우 다양한 방법으로 활용되는 자료구조이니 꼭 알아두도록 합시다.

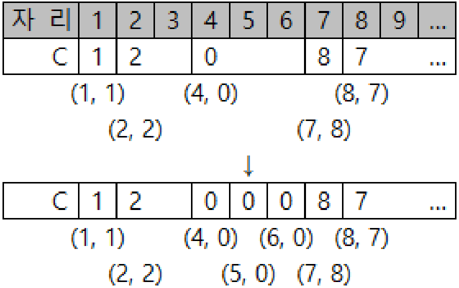
## 3. Set

| 자리 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... |
|---|---|---|---|---|---|---|---|---|---|---|
| C | 1 | 2 | | 0 | | | 8 | 7 | | ... |

(1, 1) (4, 0) (8, 7)

(2, 2) (7, 8)

- 출제자의 풀이입니다.
- 연속된 숫자를 한 블록으로 묶어 (시작 위치, 숫자)의 쌍으로 나타냅니다. 이를 std::set 등의 balanced binary search tree에 넣어 관리합니다.




**연산 1:** $i$ 번째 자리를 특정 숫자로 바꾸기

- 우선 $i$ 번째 자리를 분리하여 길이 1의 블록으로 바꿉니다.

| 자 리 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... |
|---|---|---|---|---|---|---|---|---|---|---|
| C | 1 | 2 | | 0 | 0 | 0 | 8 | 7 | | ... |

(1, 1) (4, 0) (6, 0) (8, 7)

(2, 2) (5, 0) (7, 8)

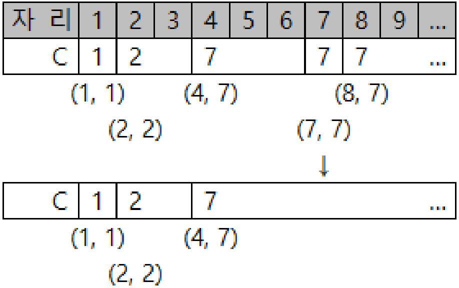
↓

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| C | 1 | 2 | | 0 | 5 | 0 | 8 | 7 | | ... |

(1, 1) (4, 0) (6, 0) (8, 7)

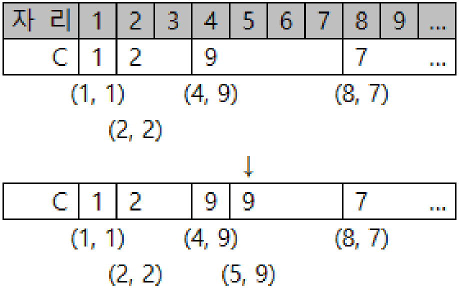
(2, 2) (5, 5) (7, 8)

- 그 블록의 값을 바꿉니다.




- 바꾼 뒤의 값이 왼쪽이나 오른쪽 블록의 값과 같을 수도 있습니다. 그 경우에는 하나로 합쳐줍니다.

## A. A+B와 쿼리




**연산 3:** $i$ 번째부터 $j$ 번째까지 자리가 모두 9라는 (0이라는) 가정 하에 모두 0으로 (9로) 바꾸기

- 마찬가지입니다. 이번에는 $i$ 번째부터 $j$ 번째까지 자리를 하나의 블록으로 분리합니다.

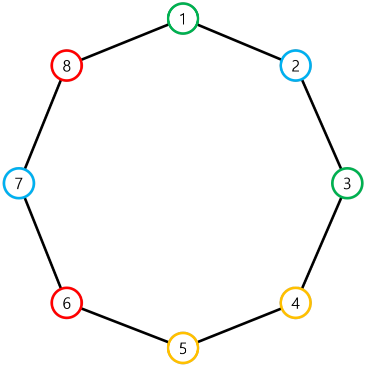
- **연산 2**는 BST에서의 탐색으로 구현할 수 있습니다.
- 각 연산의 시간 복잡도는 $\mathcal{O}(\log N)$입니다.

Challenge: 펜윅 트리로 풀어보세요!

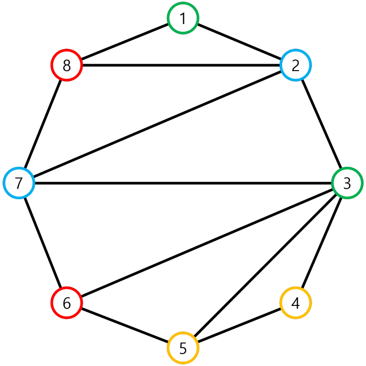
# B. Distance on Triangulation 2

ad_hoc
출제진 의도 – **Challenging**

- 제출 19번, 정답 0팀 (정답률 0.00%)
- 처음 푼 팀: –
- 출제자: `molamola`

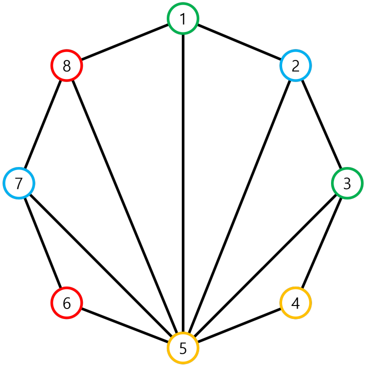



- 같은 사람이 소유한 집들을 같은 색으로 칠해 봅시다.



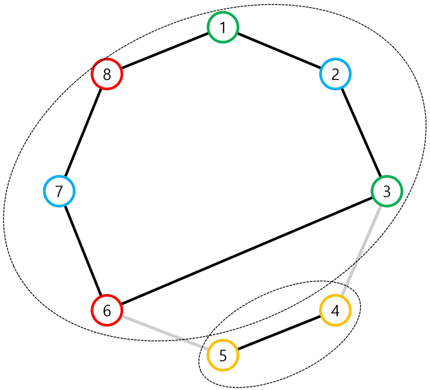

- 주어진 다각형을 삼각분할해서 같은 색 정점들 사이 거리 합을 최소화하는 문제입니다.
- 두 정점 사이 거리는 거쳐야 하는 선분 개수를 의미합니다.

- Claim. 문제의 답은 $2N - 1$ 이하입니다.

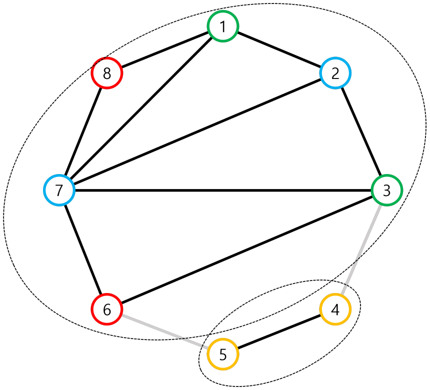



- Proof. 성게 모양으로 삼각분할하면 $2N-1$짜리 답을 얻을 수 있습니다.

- $2N - 1$은 (많은 경우에) 꽤 좋은 답입니다.
- 하지만 이보다 더 작은 해가 존재하는 경우도 있습니다.
- 위의 성게 방법을 조금 더 확장해 봅시다.

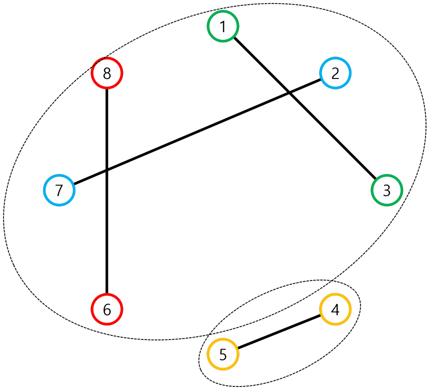



- 위처럼 분리된 영역으로 나눌 수 있는 경우, 이를 따로따로 해결할 수 있습니다.




- 각 영역을 성게 모양으로 삼각분할하면 더 작은 해를 얻을 수 있습니다.

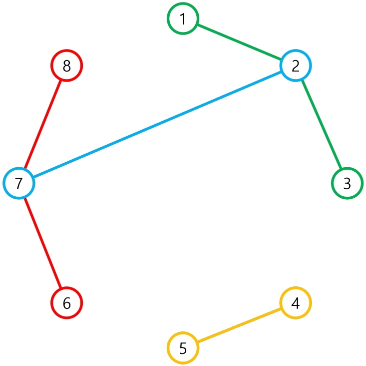
- 영역을 더 잘 정의해 봅시다.
- 같은 색인 두 정점을 잇는 선분을 모두 긋습니다.
- 두 선분이 교차한다면 이들이 같은 컴포넌트에 속한다고 합시다.
- 이 때, 문제의 답은 $2N-$ (컴포넌트 개수) 입니다.




- 예시에서는 컴포넌트 수가 2개이므로 답은 $2N-2=6$ 입니다.

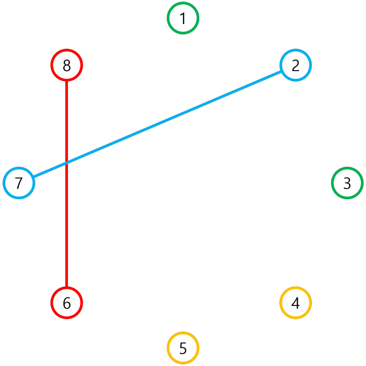
- 각 컴포넌트를 성게 모양으로 삼각분할하면 $2N-$ (컴포넌트 개수)짜리 답이 구해집니다.
- 이것이 최적임을 증명할 것입니다.

- 새로운 문제를 생각해 봅시다.
- 도로가 하나도 없는 상태에서 시작합니다.
- 도로를 적당히 추가해서, 같은 색을 갖는 두 정점끼리 **연결**되도록 하려 합니다.
- 추가해야 하는 도로는 최소 몇 개일까요?

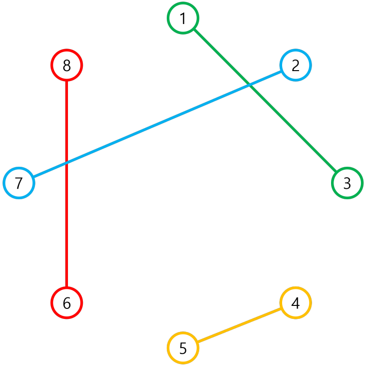



- 빨간색은 빨간색끼리, 파란색은 파란색끼리 ... 연결되어 있어야 합니다.
- spanning tree보다 약한 조건이므로 답은 forest 형태일 것입니다.

## B. Distance on Triangulation 2




- 위와 같이 파란 정점을 이은 선분과 빨간 정점을 이은 선분이 교차하는 경우를 봅시다.
- 이 때 4개의 정점은 같은 컴포넌트에 있어야 합니다.

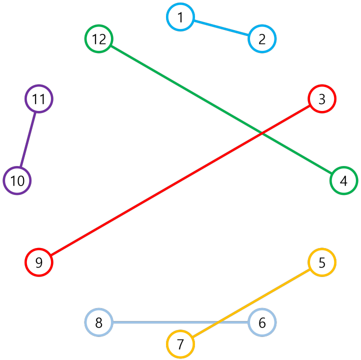



- 즉 위 그림에서 1, 2, 3, 6, 7, 8번 정점은 같은 컴포넌트에 속해야 합니다.
- 4, 5번 정점도 같은 컴포넌트에 속합니다.

- forest의 간선 개수는 (정점 개수) $-$ (컴포넌트 개수) 이므로
- $2N -$ (컴포넌트 개수)가 새로운 문제의 답이 됩니다.

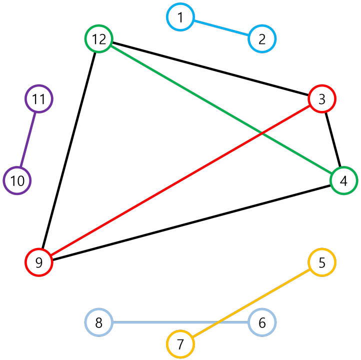
- 새로운 문제의 답은 원래 문제의 답보다 작거나 같습니다.
- 원래 문제에서 각 최단경로에 포함된 간선들을 전부 모았을 때,
- 같은 색의 정점은 모두 연결되어 있기 때문입니다.

- 정리하면,
- $2N-$ (컴포넌트 개수) $=$ (새로운 문제의 답) $\leq$ (원래 문제의 답) 이면서
- 원래 문제에는 $2N-$ (컴포넌트 개수) 짜리 해가 존재하므로
- 이것이 최적해입니다.




- 컴포넌트를 나누는 예시

# B. Distance on Triangulation 2



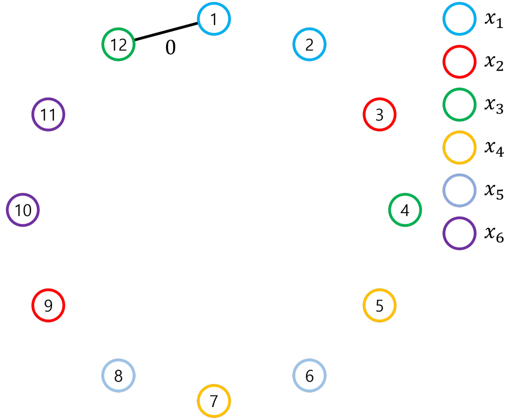

- 한 컴포넌트에 속한 정점들 번호가 인접할 필요가 없음에 유의합시다.

- 컴포넌트를 나이브하게 나누면 $\mathcal{O}\left(N^2\right)$ 으로 느립니다.
- 이를 빠르게 나누는 방법을 알아봅시다.

- 가능한 방법은 여러 가지가 있습니다.
- stack을 사용하는 방법
- DFS + segment tree를 사용하는 방법
- 해싱을 사용하는 방법
- 등등등...

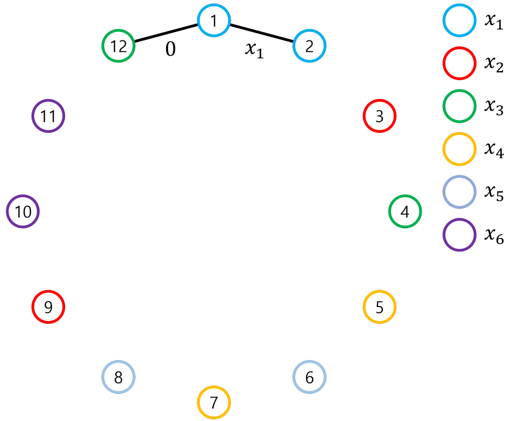
- 특이하면서도 가장 코딩이 간단한 해싱 풀이를 소개합니다.
- $N$ 개의 색에 각각 $[0, 2^{64})$ 사이 랜덤한 수를 지정합니다.
- 이를 각각 $x_1, x_2, \ldots, x_N$ 이라고 합시다.

## **B.** Distance on Triangulation 2



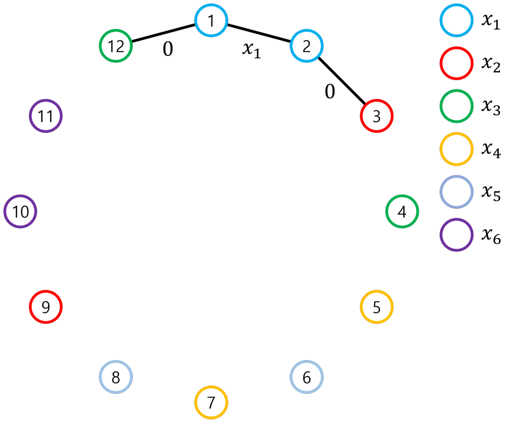

- 먼저 $1$번 정점과 $2N$번 정점 사이 간선에 $0$을 적습니다.

- 시계방향으로 한 바퀴 돌면서...
- 만나는 정점의 색이 $c$라면 (이전 간선에 적힌 수) XOR $x_c$를 새 간선에 적어 줍니다.



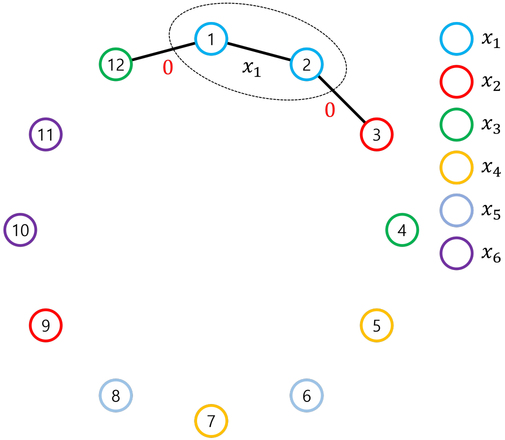

- 1번 정점 색이 1이므로 새 간선에는 $0 \oplus x_1 = x_1$ 을 적습니다.

## **B.** Distance on Triangulation 2

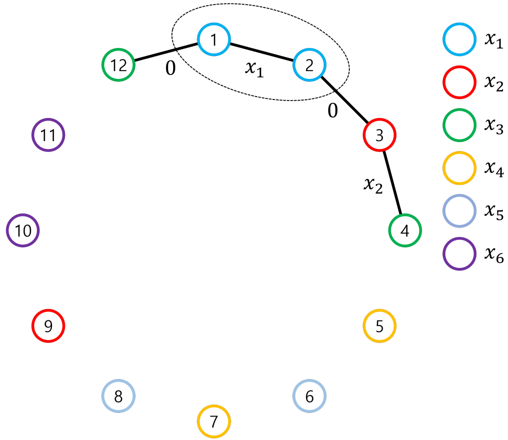



- 2번 정점 색이 1이므로 새 간선에는 $x_1 \oplus x_1 = 0$을 적습니다.

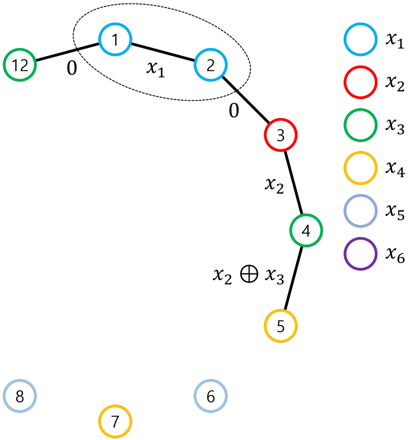
- 만약 두 간선에 적힌 수가 같다면, 두 간선 사이의 정점들은 그 밖의 정점들과 색이 겹치지 않습니다.
- 이미 적었던 수를 한 번 더 적은 경우, 두 간선 사이 정점들을 한 컴포넌트로 묶어 줍시다.




- $0$이 적힌 두 간선 사이에 있는 1, 2번 정점을 같은 컴포넌트로 묶습니다.

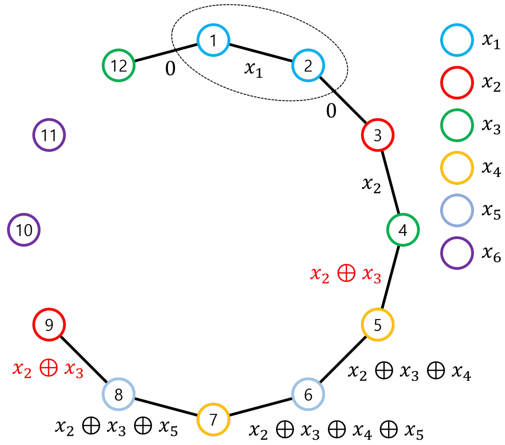



- 3번 정점 색이 2이므로 새 간선에는 $0 \oplus x_2 = x_2$을 적습니다.

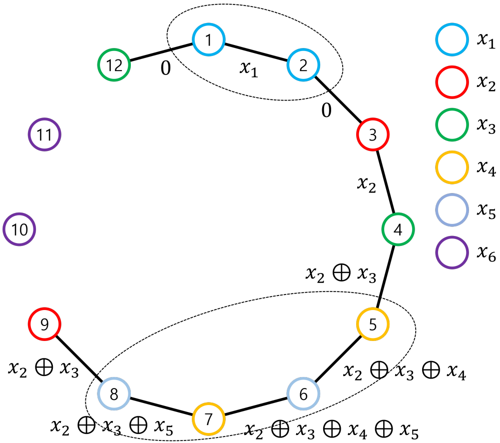


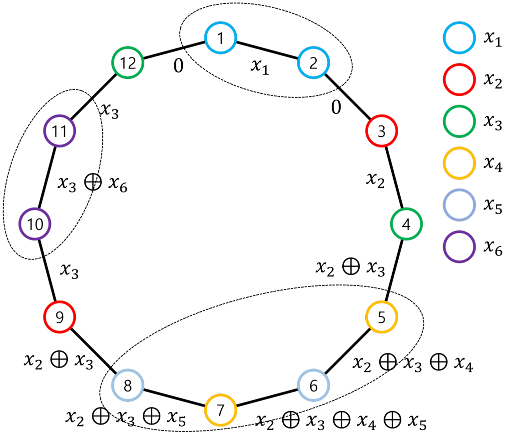

- 4번 정점 색이 3이므로 새 간선에는 $x_2 \oplus x_3$을 적습니다.

(U)

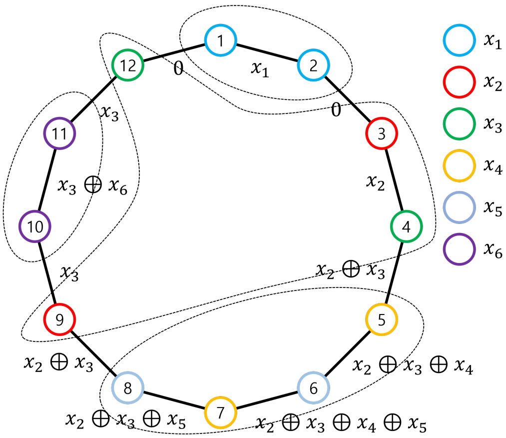



- 이를 반복하다가 같은 값이 나오면 그 사이 정점들을 묶어 줍니다.


1
2
12
3
4
5
6
7
8
9
10
11
0
$x_1$
0
$x_2$
$x_2 \oplus x_3$
$x_2 \oplus x_3 \oplus x_4$
$x_2 \oplus x_3 \oplus x_4 \oplus x_5$
$x_2 \oplus x_3 \oplus x_5$
$x_2 \oplus x_3$
$x_1$
$x_2$
$x_3$
$x_4$
$x_5$
$x_6$




— 한 바퀴 돌고 나면 나머지 정점을 한 컴포넌트로 묶어 줍니다.


1
2
3
4
5
6
7
8
9
10
11
12
$x_1$
0
0
$x_2$
$x_2 \oplus x_3$
$x_2 \oplus x_3 \oplus x_4$
$x_2 \oplus x_3 \oplus x_4 \oplus x_5$
$x_2 \oplus x_3 \oplus x_5$
$x_2 \oplus x_3$
$x_3$
$x_3 \oplus x_6$
$x_3$
$x_1$
$x_2$
$x_3$
$x_4$
$x_5$
$x_6$

- 역추적을 할 때 유의해야 합니다.
- 무조건 $2N-3$개의 간선을 출력해야 하기 때문에, 쓸모 없는 간선도 억지로 추가해야 합니다.
- 이는 컴포넌트를 구하면서 자연스럽게 처리할 수 있습니다.

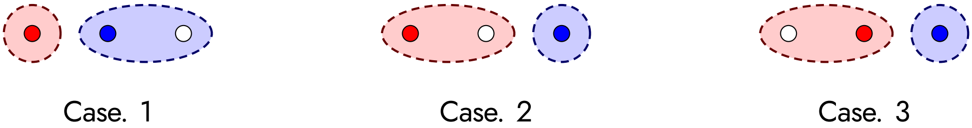
- 시간복잡도는 $\mathcal{O}(N \log N)$ 혹은 $\mathcal{O}(N)$ 입니다.

# C. UCP-Clustering

`graph, implementation`

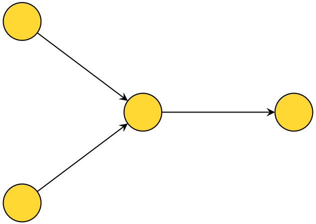
출제진 의도 – **Medium**

- 제출 35번, 정답 4팀 (정답률 11.43%)
- 처음 푼 팀: **[URGENT]** (ENCAPTURED BY THE SHADOW, Unleashed World, JUST FUN), 188분
- 출제자: `jihoon`




- 총 $\binom{N}{2}$ 가지의 초기 값을 고려하여 UCP-Clustering을 진행하였을 때, 가능한 최종 결과 및 평균 반복 횟수를 구하는 문제입니다.

- 두 클러스터의 중심이 주어졌을 때, 모든 데이터에 대해서 가장 가까운 클러스터의 중심은 $\mathcal{O}(N)$의 시간에 찾을 수 있습니다.
- 그 후 새로운 두 클러스터의 중심은 각 차원별로 중앙값을 찾아서 구할 수 있으므로 마찬가지로 $\mathcal{O}(N)$의 시간에 찾을 수 있습니다.
- 그리고 새로운 두 중심은 이전의 결과와 무관하게, 현재의 두 클러스터의 중심에 의해서 결정됩니다.



- 두 클러스터의 중심을 일종의 state로 생각하고, 새로운 두 클러스터의 중심을 정하는 것을 transition으로 생각할 수 있습니다.
- 그러므로, out degree가 1인 유향 그래프에서 가능한 시작 노드 $\binom{N}{2}$ 개가 주어졌을 때, 만나게 되는 self-loop를 가지는 노드와 그 노드까지의 거리의 기댓값을 구하는 것으로 모델링할 수 있습니다.

- 총 가능한 state의 수를 $M$ 개라고 한다면, 다음 state를 구하는데 $\mathcal{O}(N)$ 의 계산이 필요하므로 전체 알고리즘의 시간 복잡도는 $\mathcal{O}(NM)$ 이 될 것입니다.
- 그렇다면 $M$ 의 upper bound는 어떻게 구할 수 있을까요?

1. 중앙값의 성질을 사용

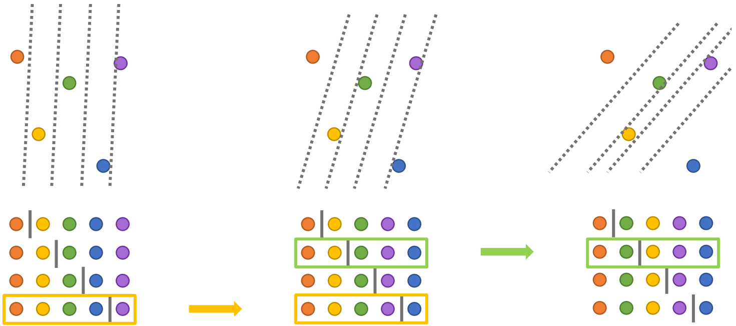
- 중앙값의 성질을 이용하면, 각 차원별 UCP-Clustering의 중심으로 가능한 값은 $\mathcal{O}\left(N^2\right)$ 임을 알 수 있습니다. $2$차원의 데이터에 대하여 문제를 풀고 있으므로, UCP-Clustering의 중심으로 가능한 좌표는 $\mathcal{O}\left(N^4\right)$ 개가 됩니다.
- 두 개의 클러스터의 중심을 나타내야 하므로 이 때 구할 수 있는 upper bound는 $\mathcal{O}\left(N^8\right)$ 로, 주어지는 $N$ 을 감안한다면 굉장히 큰 수가 됩니다.
- 과연 이것은 tight한 upper bound일까요?

## 2. 기하학적 접근

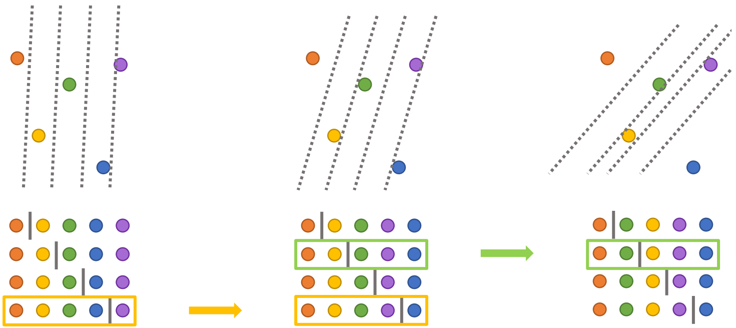
- 현재 두 클러스터의 중심을 양 끝으로 하는 선분을 생각해봅시다.
- 선분의 중점에서 선분에 수직인 직선을 그어보면, 이 직선을 경계로 하여, 새로운 두 개의 클러스터가 결정되는 것을 알 수 있습니다!
- 그러므로, 직선을 그었을 때 만들어지는 서로 다른 클러스터 구성의 경우의 수가 indegree가 0이 아닌 노드의 수의 upper bound가 됨을 알 수 있습니다.
- 실제로 탐색하게 되는 indegree가 0인 노드들은 시작 노드들 뿐이며, 이는 총 $\mathcal{O}\left(N^2\right)$ 개입니다.

- 같은 기울기를 가진 직선을 통해서 최대 $N-1$ 가지의 다른 클러스터 구성을 만들 수 있습니다.
- 그리고 직선의 각도를 서서히 바꾸면서 360도 회전했을 때, 클러스터 구성을 바꿀 수 있는 기울기의 개수는 $\mathcal{O}\left(N^2\right)$ 개 밖에 되지 않음을 알 수 있습니다.
- 그러므로, $M$ 은 $\mathcal{O}\left(N^3\right)$ 를 만족합니다.
- 그런데 여기서 조금 더 생각을 해보면, upper bound를 $\mathcal{O}\left(N^2\right)$ 로 줄일 수 있습니다!

# C. UCP-Clustering



- 현재 상태에서 각도를 아주 미세하게 바꾸어서, 클러스터의 구성을 바꿔봅시다.



– 각도를 바꾸기 전의 가능한 구성과 각도를 바꾼 후의 가능한 구성 간에는 정확히 한 가지 다른 경우만 존재함을 확인할 수 있습니다.

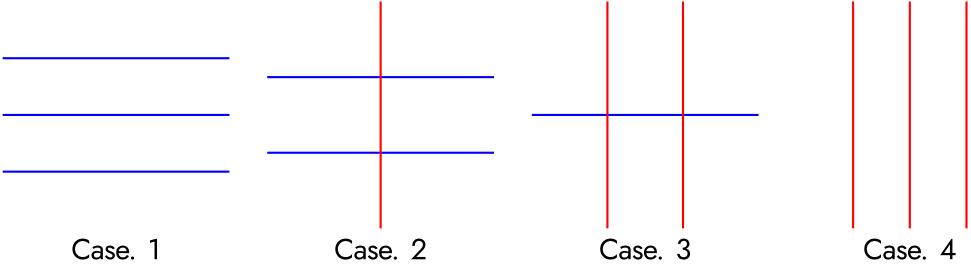
- 구성을 다르게 만드는 서로 다른 기울기의 종류는 $\mathcal{O}\left(N^2\right)$ 이므로, $M$ 의 upper bound 또한 $\mathcal{O}\left(N^2\right)$ 이 됩니다. 그러므로, 전체 알고리즘의 시간 복잡도는 $\mathcal{O}\left(N^3\right)$ 이며, $N = 512$ 인 경우에 제한 시간 안에 문제를 해결할 수 있습니다.
- 그래프 모델링을 별도로 하지 않고 naive하게 모든 pair로부터 UCP-Clustering을 시작하더라도 memoization을 함께 사용하여 문제를 해결할 수 있습니다. C++의 std::map을 이용하여 구현할 경우, memoization 관련 시간 복잡도를 고려하더라도 $\mathcal{O}\left(N^3 + N^2 \log N\right)$ 에 문제를 해결할 수 있습니다.
- 의도한 풀이는 $\mathcal{O}\left(N^3\right)$ 이지만, $\mathcal{O}\left(N^2 \log N\right)$ 풀이도 있으니 관심 있으신 분은 도전해보셔도 좋을 것 같습니다.

# D. 츠바메가에시

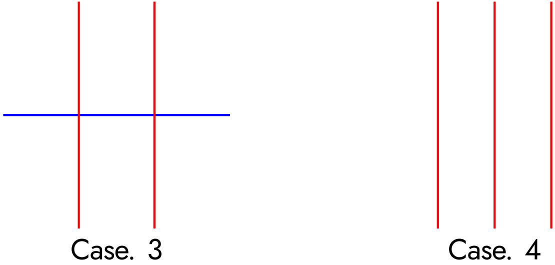
`priority_queue, data_structures`

출제진 의도 – **Medium**

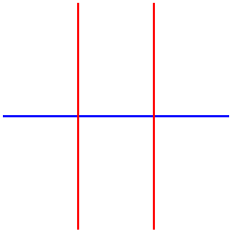
- 제출 111번, 정답 57팀 (정답률 51.35%)
- 처음 푼 팀: **https://youtu.be/JLY6st2rPKg** (구독, 좋아요, 알림 설정까지), 11분
- 출제자: `pichulia`




- 검격을 어떤 방향으로 수행하느냐에 따라 총 네 가지 경우로 나누어서 생각해볼 수 있습니다.
- 그 중 두 가지는 $X, Y$ 좌표를 서로 바꿔서 한번 더 계산하면 됩니다.
  [Case. 1] 는 [Case. 4] 의 계산 방식으로 처리할 수 있고,
  [Case. 2] 는 [Case. 3] 의 계산 방식으로 처리할 수 있습니다.

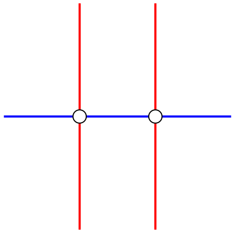



- [Case. 4] 는 아주 간단한 논리로 해결 가능합니다.
- $Y$ 축과 평행한 검격마다 얻을 수 있는 점수의 합을 미리 계산해놓고, 그 중 값이 가장 큰 세 개의 검격의 점수의 합을 구하면 됩니다.
- 검격을 두 번 이하로 사용해 모든 제비를 베는 경우 등은 적절히 예외처리를 합니다.

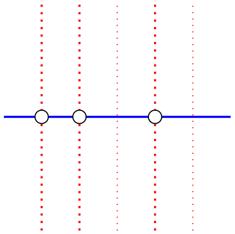


Case. 3

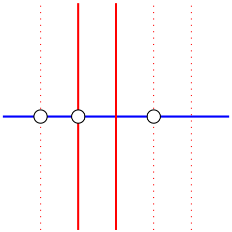
- 따라서 [Case. 3] 와 같이 $X$ 축과 평행한 검격 하나와, $Y$ 축과 평행한 검격 두 개에 대한 점수의 최대값을 계산하는 방법만 있으면 전체 정답을 구할 수 있습니다.
- 이후 풀이에서는 $X$ 축과 평행한 검격을 '$X$ 검격', $Y$ 축과 평행한 검격을 '$Y$ 검격' 이라고 축약해서 서술하겠습니다.



- 언뜻 보면 [Case. 4] 와 비슷하게, $X$ 검격 중 최대 점수 1개와, $Y$ 검격 중 최대 점수 2개의 합을 계산하면 되는 것 처럼 보입니다.
- 하지만 교차하는 검격에 의해서 두 번 베어지는 제비는 점수가 두 번 더해져서는 안된다는 점 때문에 골치가 아픕니다.



- 우선 $X$ 검격 을 고정시켜놓고 생각을 해보겠습니다.
- $X$ 검격 에 의해 제거된 제비들은 이제 점수를 더해져서는 안되므로, $Y$ 검격 점수에서 해당 제비들의 점수만큼 제거를 해줍니다.



- 점수가 보정된 $Y$ 검격 들 중 최대 점수 2개의 합을 구해봅니다.
- $X$ 검격 점수와, 위 2 개의 점수의 합을 구하면, $X$ 검격 하나가 고정되어있을 때 얻을 수 있는 점수의 최대값을 구할 수 있습니다.
- 위 과정을 가능한 모든 $X$ 검격 에 대해서 수행하면 최종 정답을 구할 수 있습니다.

- 해당 풀이를 쿼리 문제로 바꿔서 생각해 보겠습니다.

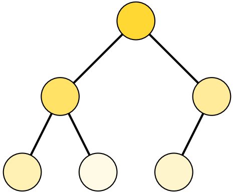
0. 초기에 $Y$ 검격 의 가지 수 만큼의 값이 저장된 수열이 주어지고
1. $x$ 번째 수열의 값을 $v$ 만큼 감소시킨다.
2. 전체 수열 중 가장 큰 값과, 두 번 째로 큰 값을 얻어온다.
3. $x$ 번째 수열의 값을 $v$ 만큼 증가시킨다. ( 1. 쿼리로 인해 값이 감소하기 이전으로 되돌린다 )

- 1. 과 3. 쿼리는 각각 제비의 개수 $N$ 만큼 발생하고, 2. 쿼리는 $X$ 검격 의 가지 수 만큼 발생합니다.

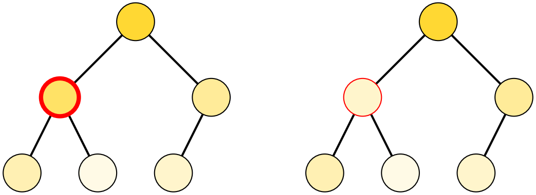
- 각각의 쿼리를 $\mathcal{O}(\log M)$ 수준으로 빠르게 처리할 수 있는 적절한 자료구조를 사용해서 해결할 수 있습니다. ($M$ 은 좌표의 범위 = $1\,000\,000$)

1. 각 노드마다 최대값을 2개씩 들고다니는 Segment Tree 를 사용
2. 임의의 노드를 자유롭게 삽입/삭제 할 수 있는 Multi Set 을 사용
3. 각 노드의 값을 임의로 수정할 수 있는 Indexed Heap ( Indexed Priority Queue ) 을 사용

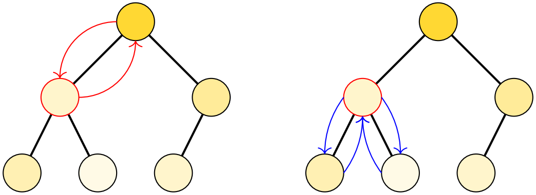
- 본 슬라이드에서는 조금 생소할 Indexed Heap 를 기준으로 설명하겠습니다.



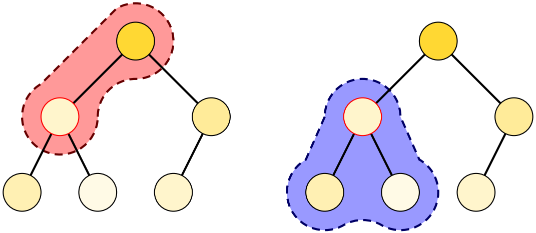
- Indexed Heap 은 특정 데이터가 Heap 의 어느 노드에 위치해있는지를 빠르게 알아내기 위해, 노드 번호를 같이 기록한 자료구조입니다.
- 일반적인 Heap 의 root 노드는 $1$ 번으로 지정되고, 번호가 $x$ 인 노드의 자식은 각각 $2x$, $2x+1$ 이 되도록 해서 말단 노드의 번호가 $N$ 이 되도록 번호를 부여받습니다. ( 물론 root 노드를 0번으로 지정하는 방법도 있습니다. )



- 예를 들어 좌표가 $x$ 인 $Y$ 검격 위에 제비가 있어서 $v$ 만큼 값을 감소시켜야하는 상황을 생각해봅시다.
- 만약 $x$ 좌표에 대응되는 노드의 Index 값이 $2$ 임을 알고있다면, 왼쪽 그림에 빨간 색으로 표시된 노드라는 것을 한번에 알 수 있습니다.
- 그리고 해당 노드의 값에 $v$ 를 바로 빼 줄 수 있습니다.



- 저런! $v$ 값을 빼주는 바람에 Heap 구조를 만족하지 않게 되버렸습니다!
- 왼쪽 그림처럼 부모노드와 자리를 교환하거나, 오른쪽 그림처럼 자식노드 중 하나와 자리를 교환하는 것을 반복해서, Heap 구조를 만족하도록 만들어야합니다.
- **값이 변경된 노드는 둘 중 한가지의 연산만을 반복해서 수행하게 됩니다.** 자세한 증명은 생략합니다. 어렵지 않습니다.



- 부모노드와 교환하는 것은, 해당 노드가 말단인 Sub-Heap 에서 Push 연산을 수행한 것과 정확히 같은 동작을 합니다.
- 자식노드와 교환하는 것은, 해당 노드를 root 로 하는 Sub-Heap 에서 Pop 연산을 수행한 것과 정확히 같은 동작을 합니다.
- 따라서 Heap 의 전체적인 구조를 만족하면서, 임의의 노드의 값을 수정할 수 있는 Heap 을 만들 수 있게 됩니다.

- 노드값 1개를 수정하면 최대 Heap의 높이 ( == $\mathcal{O}(\log M)$ ) 만큼 노드가 이동하므로, 1. 과 3. 쿼리를 $\mathcal{O}(\log M)$ 에 해결할 수 있습니다.
- 또한, 이렇게 만든 Heap 에서 가장 큰 값은 1번 노드, 두 번 째로 큰 값은 2번 노드 또는 3번 노드에 있으므로, 2. 쿼리를 $\mathcal{O}(1)$ 에 해결할 수 있습니다.

- 전체 시간복잡도는 $\mathcal{O}(M + N \log M)$, 공간복잡도는 $\mathcal{O}(M + N)$ 입니다.
- 좌표압축을 사용했다면 $M$ 대신 $N$ 이 사용되어 각각 $\mathcal{O}(N \log N)$, $\mathcal{O}(N)$ 으로 줄어듭니다만, 굳이 좌표압축을 하지 않아도 정답을 받을 수 있게 설계하였습니다.

- 이 외에도 '이게 왜 맞았지?' 싶은 $\mathcal{O}\left(N\sqrt{N}\right)$ Greedy 풀이가 있으니 심심하신 분들은 직접 풀어보고 증명도 해보시기 바랍니다.

# E. 가위바위보 버블 정렬

ad_hoc

출제진 의도 – **Hard**

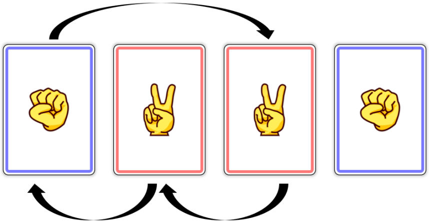
- 제출 86번, 정답 45팀 (정답률 52.33%)
- 처음 푼 팀: **atcRXDyUNl4HGA7** (iGaM30GZzw0lTMU, G27eK6TJLM1yO0E, XYvSgfxJBXX2N0s), 37분
- 출제자: `benedict0724`

## 문제 요약

- 가위, 바위 또는 보가 그려진 카드 N개가 나열되어 있습니다.
- 문제에서 정의된 버블 소트와 유사한 시행을 T번 했을 때, 최종 배열을 구하는 문제입니다.
- N과 T의 제한이 매우 크기 때문에, 시행을 직접 T번 시도하는 $\mathcal{O}(NT)$ 해법으로는 문제를 해결할 수 없습니다.

- 우선 가위, 바위, 보 중 두 가지 종류의 카드만 있는 경우에 문제를 풀어봅시다.
- 두 종류의 카드 A, B가 있습니다. A는 B를 이기는 카드입니다.

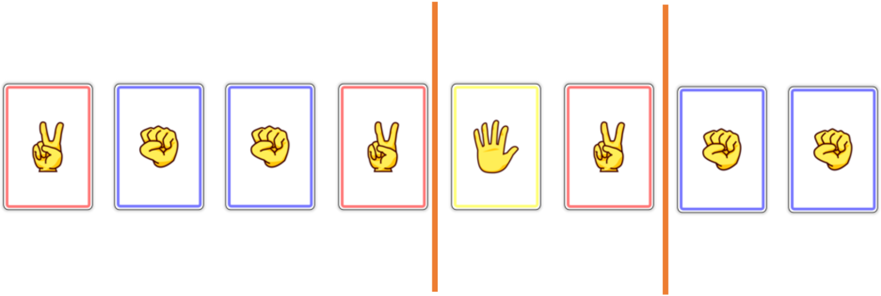
- A 카드(승리하는 카드)보다는 B 카드(패배하는 카드)들의 움직임에 주목해야 합니다.
- 만약 B 카드 왼쪽에 A 카드가 하나라도 있다면, 놀이를 1번 하면 B 카드는 왼쪽으로 1칸 이동하게 됩니다.



- B 카드 왼쪽에 A 카드가 하나도 없다면, 놀이를 1번 했을 때 B 카드의 위치가 변하지 않습니다.
- B 카드끼리는 순서가 바뀌지 않으므로, 왼쪽에서 $k$ 번째로 나타나는 B 카드는 $k$ 번째 위치보다 왼쪽에 놓일 수 없습니다.
- 따라서 T번의 시행 후 $i$ 번째 위치에 $k$ 번째로 나타나는 B 카드의 최종적인 위치는 $max(i - T, k)$ 가 됩니다.
- 이렇게 B 카드를 채운 후, 남은 공간에 A 카드를 채우면 $\mathcal{O}(N)$ 에 문제가 해결됩니다.

- 가위, 바위, 보 세 가지 종류의 카드가 모두 등장할 경우, 위의 방법을 그대로 사용할 수는 없을 것처럼 보입니다.
- 하지만 간단한 아이디어를 통해 세 가지 종류의 카드가 모두 있더라도 위의 방법을 사용할 수 있습니다.

- 앞에서부터 카드들을 보면서 가위, 바위, 보 중 몇 가지 종류의 카드가 나타났는지 셉니다.
- 현재 배열에 세 가지 종류의 카드가 모두 등장하는 순간, 배열을 분리합니다.
- 즉, 분리된 배열에는 마지막 배열을 제외하면 정확히 두 가지 종류 카드만 나타납니다.
- 분리된 배열 중 마지막 배열은 한 가지 또는 두 가지 종류의 카드가 나타날 수 있습니다.



**Observation 1**: 첫 번째 시행에서 분리된 배열들 사이에서 카드가 바뀌지 않습니다.

- **Proof**: 분리된 배열에 카드 A와 B가 있다고 가정해봅시다. 이때 A 카드가 B 카드를 이깁니다.
- 첫 번째 시행이므로 분리된 배열 다음으로 오는 카드는 반드시 C 카드입니다.
- 분리된 배열에서 마지막 카드는 항상 A입니다.
- A는 C에게 패배하기 때문에, 교환이 일어나지 않습니다.

**Observation 2**: 계속되는 시행에서도 분리된 배열들 사이에서 카드가 바뀌지 않습니다.

- **Proof**: 분리된 배열의 마지막 카드는 A이고, 다음으로 오는 배열에는 반드시 C가 포함됩니다.
- 다음으로 오는 배열에 A와 C가 포함된 경우, 다음 배열 첫 번째 카드가 어떤 것이든 교환이 일어나지 않습니다.
- 다음으로 오는 배열에 B와 C가 포함된 경우, B 카드는 C 카드를 이기므로 맨 왼쪽의 C 카드는 오른쪽으로 이동하지 않습니다. 따라서 교환이 일어나지 않습니다.
- 다음으로 오는 배열에 C만 포함되는 경우, 배열 사이에 교환이 일어나지 않습니다.

- 따라서 분리된 배열 사이에서 교환이 절대 일어나지 않으며, 각각의 배열에는 두 가지 종류의 카드만 있기 때문에 두 가지 카드만 있을 때의 해법을 그대로 적용할 수 있습니다.
- 배열을 분리할 때 $\mathcal{O}(N)$, 각각의 분리된 배열에서 문제를 해결할 때 $\mathcal{O}(N)$의 시간이 걸리므로 최종적으로 $\mathcal{O}(N)$에 문제를 해결할 수 있습니다.

- 비슷한 관찰을 적용하면 배열에서 오른쪽으로 이동하며 (현재 카드, 카드의 개수)를 관리하거나 linked list를 이용하는 풀이도 존재합니다.
- 두 풀이 모두 $\mathcal{O}(N)$에 문제를 해결할 수 있습니다. 아이디어가 크게 다르지 않기 때문에 자세히 다루지는 않습니다.

# F. 간단한 문제

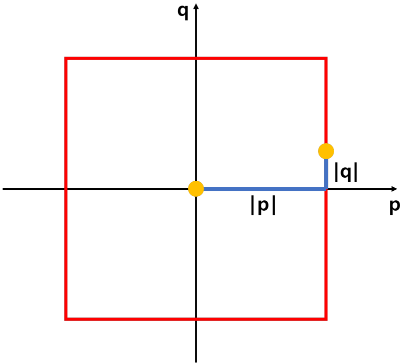
ad_hoc
출제진 의도 – **Medium**

- 제출 115번, 정답 34팀 (정답률 29.57%)
- 처음 푼 팀: **Noto Sans CJK** (cubelover, jhuni, kajebiii), 18분
- 출제자: `molamola`

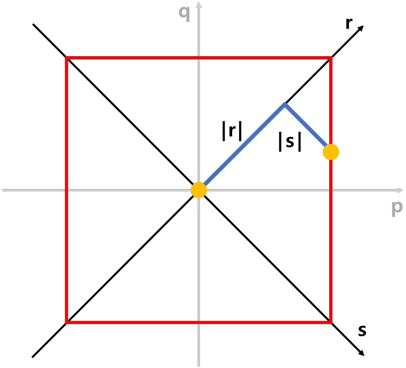
- 두 수열 $(p_1, p_2, \ldots, p_N), (q_1, q_2, \ldots, q_N)$ 이 주어집니다.
- 다음 값을 구하는 문제입니다.

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \min(|p_i - p_j|, |q_i - q_j|)$$

- $r = \dfrac{p+q}{2}$, $s = \dfrac{p-q}{2}$라 할 때, $\max(|p|, |q|) = |r| + |s|$입니다.




- 좌표평면에서 $\max(|p|, |q|)$가 같은 점들을 모으면 붉은 사각형으로 나타납니다.




- $|r| + |s|$ 가 같은 점들도 같은 사각형으로 나타납니다.

- 마찬가지로, $r_i = \dfrac{p_i + q_i}{2}, s_i = \dfrac{p_i - q_i}{2}$ 라고 하면
- $\max(|p_i - p_j|, |q_i - q_j|) = |r_i - r_j| + |s_i - s_j|$ 가 성립합니다.

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\min(|p_i - p_j|, |q_i - q_j|)$$

$$= \sum_{i=1}^{N}\sum_{j=1}^{N}|p_i - p_j| + |q_i - q_j| - \max(|p_i - p_j|, |q_i - q_j|)$$

$$= \sum_{i=1}^{N}\sum_{j=1}^{N}|p_i - p_j| + |q_i - q_j| - |r_i - r_j| - |s_i - s_j|$$

$$= \sum_{i=1}^{N}\sum_{j=1}^{N}|p_i - p_j| + \sum_{i=1}^{N}\sum_{j=1}^{N}|q_i - q_j| - \sum_{i=1}^{N}\sum_{j=1}^{N}|r_i - r_j| - \sum_{i=1}^{N}\sum_{j=1}^{N}|s_i - s_j|$$

- $F(x) = \sum_{i=1}^{N}\sum_{j=1}^{N}|x_i - x_j|$ 로 정의하면 $F(p) + F(q) - F(r) - F(s)$ 가 답이 됩니다.
- $F(x)$ 를 빠르게 구해 봅시다.

- 수열을 정렬해도 결과는 같습니다.
- $x_i$ 가 오름차순 정렬되어 있다고 하면 $F(x)$ 는 다음과 같습니다.

$$F(x) = 2\sum_{i=1}^{N}\sum_{j=1}^{i-1}(x_i - x_j)$$

- $\sum_{i=1}^{N}\sum_{j=1}^{i-1}(x_i - x_j)$ 에서 $x_a$ 는 $a-1$ 번 더해지고 $N-a$ 번 빼집니다.
- $F(x) = 2\sum_{a=1}^{N}((a-1)-(N-a))x_a = 2\sum_{a=1}^{N}(2a-1-N)x_a$ 입니다.

- 수열 $p, q, r, s$를 정렬해야 하므로 시간복잡도는 $\mathcal{O}(N \log N)$ 입니다.

# G. 돌 가져가기 2

`combinatorics, fft`

출제진 의도 – **Hard**

- 제출 35번, 정답 7팀 (정답률 20.00%)
- 처음 푼 팀: **[URGENT]** (ENCAPTURED BY THE SHADOW, Unleashed World, JUST FUN), 119분
- 출제자: `ainta`

- 예선의 돌 가져가기 문제와 같은 세팅입니다.
- 최대 점수가 아닌 모든 경우에 얻을 수 있는 총 점수의 합을 구하는 문제입니다.

- 만약 검은색 돌을 가져가 얻는 총 점수를 구할 수 있다면 마찬가지 방법으로 흰색 돌로 얻는 총 점수도 구할 수 있습니다.
- 검은색 돌을 가져가 얻는 총 점수 $Bscore$를 구해봅시다.

- 왼쪽부터 $1, 2, .., N$ 번 돌이라고 번호를 붙였을 때, $x$ 번 돌을 가져가면서 점수를 얻었다고 합시다.
- $x$ 번 돌을 가져갔을 때 바로 왼쪽 돌이 $l$ 번 돌, 오른쪽 돌이 $r$ 번 돌이라고 합시다.
- 모든 $N!$ 가지 경우 중 이런 일이 일어나는 경우의 수는 얼마일까요?

- $l$ 번부터 $r$ 번까지 $r-l+1$ 개의 돌을 먼저 가져간 돌의 번호부터 나열했을 때 $x, l, r$ 또는 $x, r, l$ 로 끝나는 경우에 점수를 얻게 됩니다.
- 이 사건이 발생할 확률은 $\frac{2}{_{r-l+1}P_3}$ 이고, 따라서 경우의 수는 $\frac{2N!}{(r-l+1)(r-l)(r-l-1)}$ 이 됩니다.
- $f(r-l) = \frac{2N!}{(r-l+1)(r-l)(r-l-1)}$ 라 둡시다.

- $T_k$ 를 첫 $k$ 개 돌 중 검은색 돌의 무게의 합이라고 합시다.
- 두 흰색 돌 $l, r$ 번에 대해서, 검은색 돌을 가져갈 때 왼쪽이 $l$ 번, 오른쪽이 $r$ 번이어서 얻는 점수의 총합은 $f(r-l) \cdot (T_r - T_l)$ 이 됩니다.
- 따라서, $Bscore = \sum_{l \le r, S_l = W, S_r = W} f(r-l) \cdot (T_r - T_l)$

- $Bscore = \sum_{l \le r, S_l = W, S_r = W} f(r-l) \cdot (T_r - T_l)$
- $Bscore = \sum_{i=1}^{N} \{f(i) \sum_{r-l=i, S_l=W, S_r=W} (T_r - T_l)\}$
- 모든 $i$ 에 대해 $g(i) = \sum_{r-l=i, S_l=W, S_r=W} (T_r - T_l)$ 을 구하면 충분합니다.

$$A_i = \begin{cases} T_i & \text{if } S_i = W \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$B_i = \begin{cases} 1 & \text{if } S_{N-i} = W \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

- 위와 같이 정의된 수열 $A$와 $B$에 대해 $C_i = \sum_{j+k=i} A_j B_k$ 인 수열 $C$를 FFT로 계산할 수 있습니다.
- 이 때, $g(i) = C_{N+i} - C_{N-i}$ 이 성립합니다.

- 따라서 FFT를 통해 $Bscore$를 계산할 수 있고, 최종 답 역시 구할 수 있습니다.
- modulo가 998244353이므로 NTT를 이용해 실수 오차 없이 문제를 해결할 수도 있습니다.
- NTT 또는 FFT를 사용하므로 시간복잡도는 $\mathcal{O}(N \log N)$입니다.

# H. 봉화대

dynamic_programming
출제진 의도 – **Easy**

- 제출 82번, 정답 60팀 (정답률 73.17%)
- 처음 푼 팀: **탐색기원툴32등도전합니다** (탐, 새, 기), 8분
- 출제자: evenharder

- 마을의 높이나 구간 배치가 어떻든 항상 봉화대가 설치되어야 하는 마을이 있습니다. 바로 가장 높이 있는 마을입니다.
- 가장 높은 마을이 $x$번째 마을이라면, 그 뒤에 있는 마을은 가장 높은 마을과 같은 구간에 있어야 합니다.
- 그럼 $x$번째 마을을 포함하는 구간의 시작은 어떻게 될까요?
  - $x$번째 마을이나 그 앞의 마을 어디든 가능합니다.

## H. 봉화대

(U)

- $f(n)$를 1번째 마을부터 $n$번째 마을까지 조건에 맞추어 봉화대를 설치하는 배치의 경우의 수라고 정의합시다.
  - 기저조건으로 $f(0) = 1$입니다.
- $x$를 $h_x = \max(\{h_1, h_2, \cdots, h_n\})$를 만족하는, 즉 가장 높은 마을의 위치라고 하면 다음 식이 성립합니다.

$$f(n) = \sum_{k=0}^{x-1} f(k)$$

$x$번째 마을을 포함하는 구간은 $[1, n], [2, n], \cdots, [x, n]$이 가능하기 때문입니다.
- 그러므로 $f(n)$의 누적 합을 저장하며 계산하면 시간 복잡도 및 공간 복잡도 $\mathcal{O}(N)$에 해결 가능합니다.

- 조합론을 이용하여 해결할 수도 있습니다.
- $h_x = \max(\{h_1, h_2, \cdots, h_n\})$를 만족하는 $x$를 $n$별로 $a_1, a_2, \cdots, a_N$으로 정의하면 배치할 수 있는 경우의 수는 다음과 같습니다.

$$(a_2 - a_1 + 1) \times (a_3 - a_2 + 1) \times \cdots \times (a_N - a_{N-1} + 1)$$

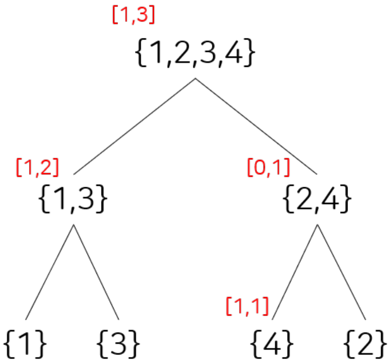
- $i$번 마을이 $a_i = i$를 만족할 때만 봉화대를 설치할 수 있다는 사실을 이용하면, 그 사이에 경계를 배치하는 경우의 수를 구하는 문제로 바꿀 수 있습니다.

# I. 붉은색 푸른색

`flow`
출제진 의도 – **Challenging**

- 제출 33번, 정답 1팀 (정답률 3.03%)
- 처음 푼 팀: **longest path to victory** (Aeren, edenooo, cgiosy), 205분
- 출제자: `tlwpdus`

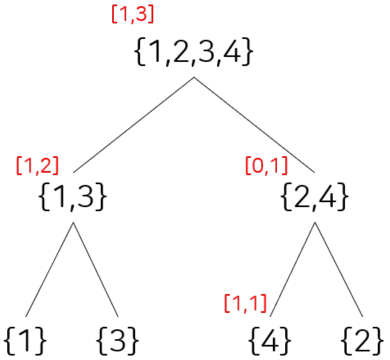
- 우선은 2번 쿼리(제거)가 없다고 해 봅시다.
- 구슬 주머니가 합쳐지기만 하므로 트리 형태로 나타낼 수 있습니다.



- 이 경우 문제는 다양한 방식으로 풀 수 있습니다.
- 트리 형태로 나타내고 나면 DFS order로 번호를 정렬해서 각 주머니의 구슬들을 구간으로 나타낼 수 있습니다.
- 그렇다면 모든 원소가 0 또는 1인 수열에서 몇몇 구간의 합의 범위가 주어졌을 때에 해당하는 해가 있는지 묻는 문제가 됩니다.
- 이는 벨만-포드로 해결하는 방식이 잘 알려져 있습니다.

- 하지만 벨만-포드로 해결하는 방식은 2번 쿼리가 있는 경우로 확장하기 어렵습니다.
- 확장 가능한 방향의 풀이를 떠올려 봅시다.

- 우선 $i$번째 구슬의 색이 붉은색이면 $x_i = 1$, 푸른색이면 $x_i = 0$으로 놓아보면, 3번 쿼리의 조건들이 선형 부등식들임을 확인할 수 있습니다.




$$1 \le x1+x2+x3+x4 \le 3$$

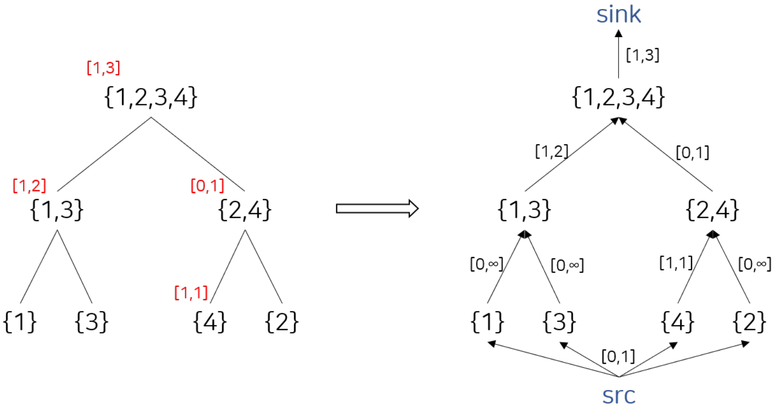
$$1 \le x1+x3 \le 2$$

$$0 \le x2+x4 \le 1$$

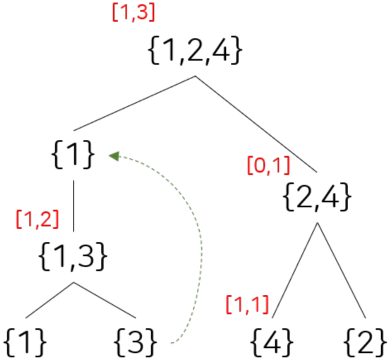
$$1 \le x4 \le 1$$

- 특수한 형태의 선형 부등식들을 모델링하는 잘 알려진 방법으로 flow가 있으니 어쩌면 flow로 풀 수 있지 않을까요?
- 위에서 얘기한 트리의 구조를 그대로 활용하면 가능합니다.
- 각 트리의 리프 구슬이 붉은색이면 1의 flow가, 푸른색이면 0의 flow가 흐른다고 하면,
- 각 주머니에서 붉은색 구슬의 개수는 해당 주머니를 나타내는 정점에 흐르는 flow와 같게 됩니다.
- 이제 각 정점마다 흐를 수 있는 flow의 하한과 상한을 모두 정하면 모델링이 가능합니다.

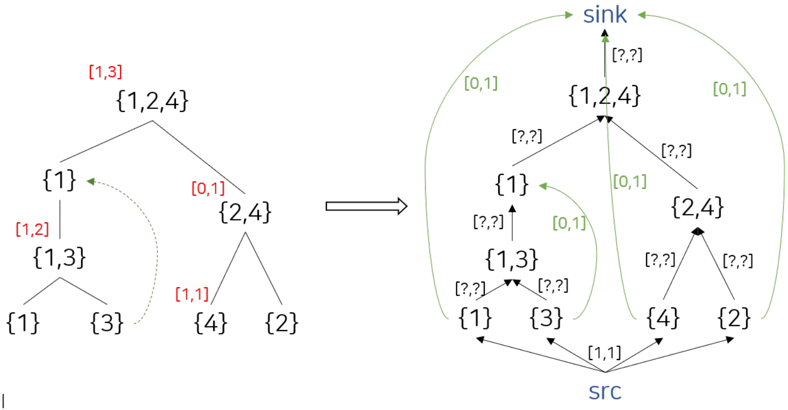
- 즉, 아래처럼 flow graph를 구성해 봅시다.
- 이처럼 간선에 흐를 수 있는 flow의 하한도 존재하는 문제는 **lr flow**로 해결이 가능합니다.



- 이제 2번 쿼리를 모델링해야 합니다.
- 구슬이 사라진 경우 그 부모 노드를 새로 만들어 상태를 기록해 봅시다.
- 가령, 1,3번 주머니에서 3번 구슬이 사라진 경우 아래와 같습니다.

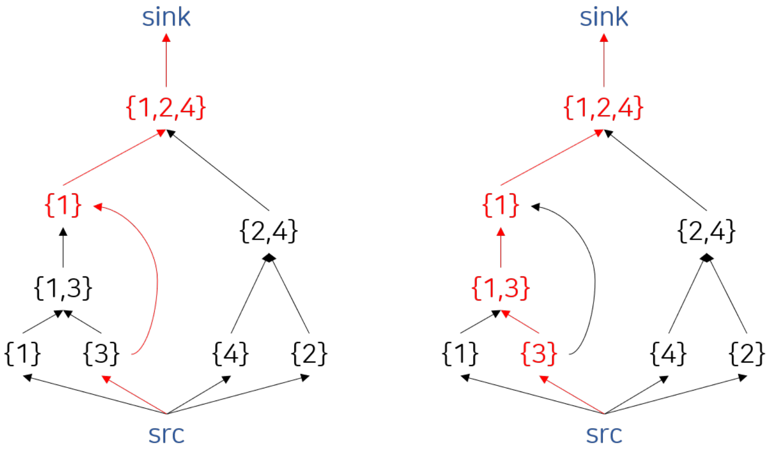


- flow로 모델링하기 위해서 이번에는 각 구슬이 없어지는 지점으로 용량 1인 간선을 이어봅시다.
- 또, 이전과 다르게 이번에는 리프로 들어오는 간선에 무조건 1의 flow가 흐르게 해 봅시다.



- 각 리프에 들어오는 flow가 어디로 흐르는지 살펴봅시다.
- 새로 추가한 간선에 flow가 흐르면 해당 flow는 $i$번째 구슬이 있는 주머니들은 건너뛰고 삭제된 이후에만 영향을 주게 됩니다.
- 한편 새로 추가한 간선에 flow가 흐르지 않는다면 리프로 들어온 flow는 $i$번째 구슬이 있었던 주머니들을 다 거친 뒤 삭제된 이후에도 영향을 주게 됩니다.

– 두 경우의 차이는 정확히 $i$번째 구슬이 있었던 주머니를 나타내는 정점들이 됩니다.



- 따라서 각 주머니에 붉은색 구슬의 개수가 $l$개 이상 $h$개 이하라는 조건을
- 적절한 계산을 거쳐 $l'$개 이상 $h'$개 이하라는 조건으로 변형할 수 있습니다.
- 그러므로 비슷하게 lr flow를 사용하면 문제를 해결할 수 있습니다.

# J. 시험 문제 출제

`meet_in_the_middle`

출제진 의도 – **Medium**

- 제출 189번, 정답 43팀 (정답률 22.75%)
- 처음 푼 팀: **lcp.jpg** (gs15120, psb0623, 이채준), 23분
- 출제자: `jjwdi0`

- 가능한 모든 경로의 가짓수는 얼마나 될까요?

- 가능한 모든 경로의 가짓수는 얼마나 될까요?
- $N$ 이 20인 경우, 오른쪽 방향과 아래 방향으로 각각 19번 이동해주어야 하고 방향의 순서를 자유롭게 바꿀 수 있으므로 총 $\binom{38}{19}$ 의 가짓수가 나올 수 있습니다.
- 이는 매우 크므로 naive하게 세어주는 것은 무리가 있습니다.

- $(1,1)$ 에서 시작하는 경로는 무조건 $(1,N)$ 과 $(N,1)$ 을 잇는 대각선을 지난다는 사실을 이용합시다. (이를 앞으로 주대각선 이라 하겠습니다)
- 그렇다면 길이 $2N-1$ 인 경로는 $(1,1)$ 에서 시작해 주대각선에 도착할 때까지의 경로와 주대각선 바로 다음에서 시작해 $(N,N)$ 에 도착하는 경로로 나눌 수 있습니다.

- $(1,1)$에서 시작하는 경로는 무조건 $(1,N)$과 $(N,1)$을 잇는 대각선을 지난다는 사실을 이용합시다. (이를 앞으로 주대각선 이라 하겠습니다)
- 그렇다면 길이 $2N-1$인 경로는 $(1,1)$에서 시작해 주대각선에 도착할 때까지의 경로와 주대각선 바로 다음에서 시작해 $(N,N)$에 도착하는 경로로 나눌 수 있습니다.

- 두 경로를 별도로 생각하면, $(1,1)$에서 시작해 주대각선에 도착하는 모든 경로와 $(N,N)$에서 시작해 주대각선 바로 전에 도착하는 모든 경로를 나열할 수 있습니다.
- 각각의 가짓수는 많아봤자 50만($\approx 2^{19}$) 정도이기 때문입니다.

- 이제 두 경로를 합친 전체 경로에서 최대 구간합을 구해야 합니다.
- 본 문제를 해결하기에 앞서 일차원 배열에서 최대 구간합을 구하는 방법을 알아보겠습니다.

- 이제 두 경로를 합친 전체 경로에서 최대 구간합을 구해야 합니다.
- 본 문제를 해결하기에 앞서 일차원 배열에서 최대 구간합을 구하는 방법을 알아보겠습니다.
- 흔히 동적계획법을 이용하는 방법과 분할정복을 이용하는 방법이 알려져 있는데, 두 방법 모두 살펴보겠습니다.

- 일차원 배열 이름을 $A$라고 하겠습니다.

- 일차원 배열 이름을 $A$라고 하겠습니다.
- 동적계획법 풀이:

- 일차원 배열 이름을 $A$라고 하겠습니다.
- 동적계획법 풀이:
  - $dp[i]$ =($A[i]$를 오른쪽 끝으로 무조건 포함하는 최대 구간합)으로 정의하면
  - $dp[i] = max(dp[i-1] + A[i], A[i])$ 와 같은 점화식을 얻을 수 있고
  - 최종 답으로는 $max_{1 \leq i \leq N}(dp[i])$를 구하면 됩니다.

- 분할정복 풀이:

- 분할정복 풀이:
  - 배열 가운데를 잘라 왼쪽 배열과 오른쪽 배열에서 각각 (전체 배열의 최대 구간합, 왼쪽 원소를 포함한 최대 구간합, 오른쪽 원소를 포함한 최대 구간합, 전체 배열 원소의 합)을 재귀적으로 구합니다.
  - 둘로 나뉜 배열의 정보를 이용하여 전체 배열의 정보를 구해줍니다.
  - 최종 답으로는 배열 $A$의 (전체 배열의 최대 구간합)에 해당하는 값을 구하면 됩니다.

- 다시 본 문제로 돌아와, 앞에서 설명한 두 방법을 이용해 해결하겠습니다.

- 다시 본 문제로 돌아와, 앞에서 설명한 두 방법을 이용해 해결하겠습니다.
- 주대각선(또는 그 직전)에 도착하는 모든 경로를 나열할 때, 각각의 경로마다 (최대 구간합, 마지막 원소를 포함하는 최대 구간합)의 pair를 구해줍니다. 이를 $(p, q)$라 하겠습니다.

- 다시 본 문제로 돌아와, 앞에서 설명한 두 방법을 이용해 해결하겠습니다.
- 주대각선(또는 그 직전)에 도착하는 모든 경로를 나열할 때, 각각의 경로마다 (최대 구간합, 마지막 원소를 포함하는 최대 구간합)의 pair를 구해줍니다. 이를 $(p, q)$라 하겠습니다.
- 이제 두 경로를 합쳐 전체 경로를 만들 때, 만들어진 배열의 최대 구간합은 $max(p_1, p_2, q_1 + q_2)$가 됩니다. 이때 $p_1, q_1$은 $(1, 1)$에서 시작하는 경로의 $(p, q)$에 해당하고, $p_2, q_2$는 $(N, N)$에서 시작하는 경로의 $(p, q)$에 해당합니다.

- 이제 $max(p_1, p_2, q_1 + q_2) = K$ 를 만족하는 경로의 개수를 세어주면 됩니다.
- 출제자가 처음에 생각했던 Persistent Segment Tree를 사용한 풀이도 있지만, 여기서는 다른 출제자분들이 알려주신 더 쉽고 빠른 풀이를 다루겠습니다.

- $max(p_1, p_2, q_1 + q_2) = K$를 만족하는 경로의 개수를 세는 것이 아닌 $max(p_1, p_2, q_1 + q_2) \leq K$를 만족하는 경로의 개수를 세어 보겠습니다. 이를 $f(K)$라 하겠습니다.

- $max(p_1, p_2, q_1 + q_2) = K$를 만족하는 경로의 개수를 세는 것이 아닌 $max(p_1, p_2, q_1 + q_2) \leq K$를 만족하는 경로의 개수를 세어 보겠습니다. 이를 $f(K)$라 하겠습니다.
- $(p_1, q_1)$에서 $p_1 \leq K$를 만족하는 $q_1$을 따로 모으고, $(p_2, q_2)$에서 $p_2 \leq K$를 만족하는 $q_2$를 따로 모으겠습니다.

- $max(p_1, p_2, q_1 + q_2) = K$ 를 만족하는 경로의 개수를 세는 것이 아닌 $max(p_1, p_2, q_1 + q_2) \leq K$ 를 만족하는 경로의 개수를 세어 보겠습니다. 이를 $f(K)$ 라 하겠습니다.
- $(p_1, q_1)$ 에서 $p_1 \leq K$ 를 만족하는 $q_1$ 을 따로 모으고, $(p_2, q_2)$ 에서 $p_2 \leq K$ 를 만족하는 $q_2$ 를 따로 모으겠습니다.
- 이제 모아준 $q_1, q_2$ 사이에서 $q_1 + q_2 \leq K$ 를 만족하는 쌍의 개수를 세어 주면 됩니다.

- $max(p_1, p_2, q_1 + q_2) = K$ 를 만족하는 경로의 개수를 세는 것이 아닌 $max(p_1, p_2, q_1 + q_2) \leq K$ 를 만족하는 경로의 개수를 세어 보겠습니다. 이를 $f(K)$ 라 하겠습니다.
- $(p_1, q_1)$ 에서 $p_1 \leq K$ 를 만족하는 $q_1$ 을 따로 모으고, $(p_2, q_2)$ 에서 $p_2 \leq K$ 를 만족하는 $q_2$ 를 따로 모으겠습니다.
- 이제 모아준 $q_1, q_2$ 사이에서 $q_1 + q_2 \leq K$ 를 만족하는 쌍의 개수를 세어 주면 됩니다.
- 이는 정렬 후 투 포인터 혹은 이진 탐색을 이용하면 쉽게 구할 수 있습니다.
- 위 방법을 이용하면, 구해야 하는 답은 $f(K) - f(K-1)$ 임을 알 수 있습니다.

- 두 경로를 합쳐줄 때에는 주대각선 위 어느 한 점에서 만나는 경로만 합쳐줘야 하기 때문에 구현상의 난이도가 어느 정도 있을 수 있습니다.

- 두 경로를 합쳐줄 때에는 주대각선 위 어느 한 점에서 만나는 경로만 합쳐줘야 하기 때문에 구현상의 난이도가 어느 정도 있을 수 있습니다.
- 이와 같이 절반까지의 완전탐색을 수행하고, 그 결과를 합쳐주는 알고리즘을 Meet in the Middle이라고 합니다.

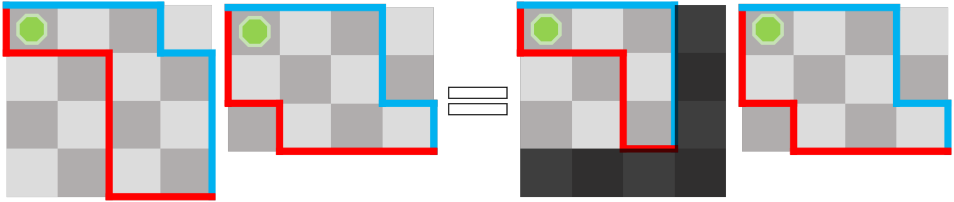
- 두 경로를 합쳐줄 때에는 주대각선 위 어느 한 점에서 만나는 경로만 합쳐줘야 하기 때문에 구현상의 난이도가 어느 정도 있을 수 있습니다.
- 이와 같이 절반까지의 완전탐색을 수행하고, 그 결과를 합쳐주는 알고리즘을 Meet in the Middle이라고 합니다.
- 나열한 모든 경로의 가짓수가 $\mathcal{O}\left(2^N\right)$ 정도 되고, 이에 대해 정렬을 수행해야 하므로 전체 시간복잡도는 $\mathcal{O}\left(N \cdot 2^N\right)$ 입니다.

# K. 은퇴한 자들의 게임

`ad_hoc`, `math`, `game_theory`

출제진 의도 – **Challenging**

- 제출 45번, 정답 5팀 (정답률 11.11%)
- 처음 푼 팀: **CSI** (cs71107, Sait2000, IHHI), 89분
- 출제자: `tlwpdus`

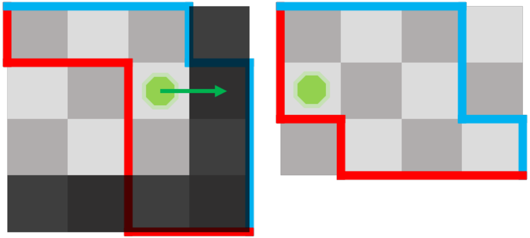


- Claim : $2 \leq N_i, M_i$ 인 판을 골라서 마지막 행과 열을 동시에 삭제해도 게임의 결과는 같습니다.
- 왜 그럴까요?

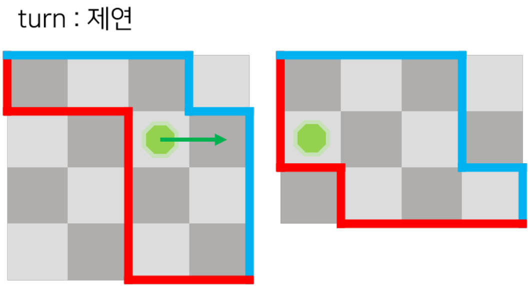
- Proof : 우선 원래 상태에서 제연이가 이기는 상태였다고 해봅시다.
- 그 말은 덕인이가 **어떤 전략을 가지고 있든 간에** 제연이가 이길 방법이 있다는 의미입니다.
- 여기서 전략이라 함은, $K$ 개 말의 위치가 주어졌을 때에 어느 판의 말을 움직여야 하는지 내뱉는 함수로 볼 수 있습니다.

- 이제 일반성을 잃지 않고, 첫 번째 판의 마지막 행과 열을 삭제했다고 해봅시다.
- 이 상태에서도 제연이에게 필승 전략이 있음을 보이면 됩니다.
- 제연이는 마지막 행과 열이 삭제되지 않은 것처럼 말을 움직일 것입니다. 즉, 기존 전략을 그대로 사용할 것입니다.
- 이 전략은 첫 번째 판의 마지막 행과 열을 삭제하기 전에는 상대가 어떤 전략을 가지고 있든 이기는 전략이었으므로,
- **대부분의 경우**에는 제연이가 이길 것 같다는 생각을 할 수 있습니다.

turn : 제연

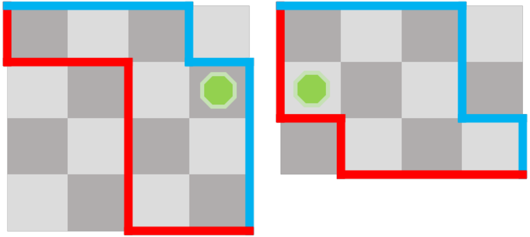


- 유일하게 제연이의 승리가 보장되지 않는 경우는 제연이가 첫 번째 말을 움직여야 하는데 이미 마지막 열에 위치하고 있을 때입니다.
- 이 경우를 다루기 위해 행과 열을 삭제하기 전 상태를 다시 생각해 봅시다.

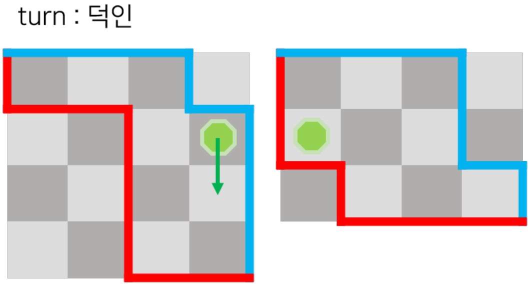



- **기존의 필승 전략**을 따라 움직였기 때문에
- 이 상태에 도달했으며, 1번 말을 움직이려 한다는 것은 곧,
- 이 상태에서 1번 말을 움직여도 **제연이가 이길 방법이 존재**한다는 뜻입니다.

turn : 덕인

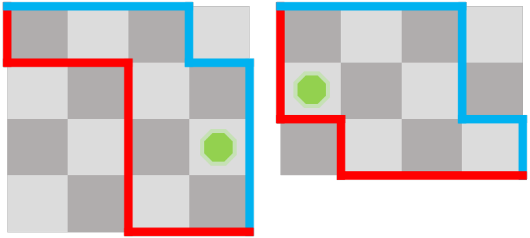


– 즉, 이 상황은 **덕인이가 어떻게 하더라도** 제연이가 이기는 상황입니다.

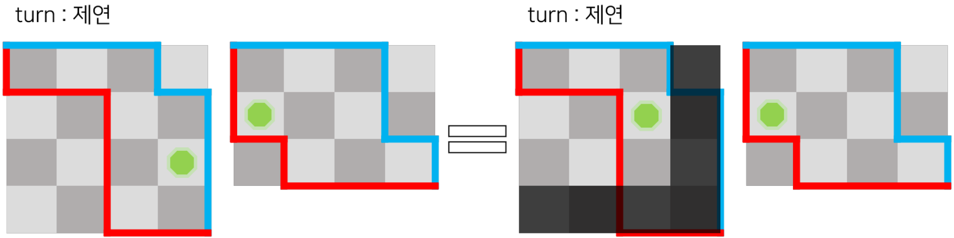



- 특히, 덕인이가 **1번 말**을 움직여도 제연이에게는 필승 전략이 존재합니다.

turn : 제연



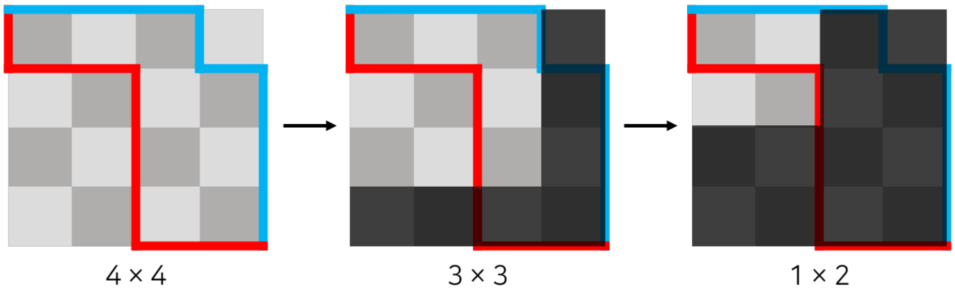
- 따라서 위 상황 또한 **제연이가 이기는 상황**일 것입니다.




- 그런데 이 상황은 원래 상황과 전혀 다르지 않습니다.
- 이미 지나온 왼쪽 또는 윗 부분은 승부에 영향을 주지 못하기 때문입니다.
- 따라서 원래 상황에서도 제연이는 필승 전략을 갖고 있었을 것입니다.

- 따라서 원래 제연이가 이기는 상태였다면,
- $i$ 번째 판의 마지막 행과 열을 제거해도 제연이가 이깁니다.
- 마찬가지로, 원래 덕인이가 이기는 상태였을 때에도 비슷한 방식으로 승자가 바뀌지 않음을 보일 수 있습니다.
- 따라서 증명이 완료됩니다.

- 이 사실을 활용하면 $K$ 개의 판 모두 $N_i = 1$ 또는 $M_i = 1$ 일 때까지 판을 축소할 수 있습니다.
- 이 때 주의할 점은, 항상 $N_i, M_i$ 가 1씩 줄어드는 것은 아니라는 것입니다.
- 아래의 경우 두 번째 축소에서 $N_i$ 가 2 줄어든 것을 볼 수 있습니다.



- 각 게임판을 $1 \times M_i$ 또는 $N_i \times 1$ 형태로 축소하고 나면 누가 이기는지를 구하는 것은 쉽습니다.
- 축소한 뒤 각 판의 크기를 $N_i' \times M_i'$ 라 하면 각 판에서 제연이가 움직일 수 있는 기회는 덕인이의 움직임에 상관 없이 $M_i' - 1$ 회이고, 비슷하게 덕인이가 움직일 수 있는 기회는 $N_i' - 1$ 회이므로
- $\sum_{i=1}^{K} N_i'$ 과 $\sum_{i=1}^{K} M_i'$ 를 비교해서 후자가 크다면 제연이의 승리, 전자가 크거나 같다면 덕인이의 승리가 됩니다.

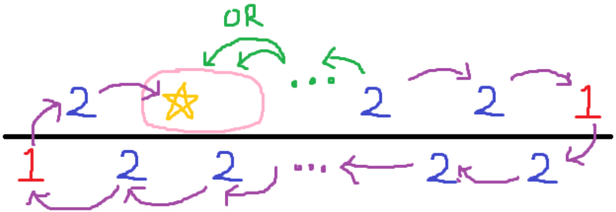
- 물론 판을 직접 만들어서 위와 같은 조작을 하면 $\mathcal{O}\left(\sum_{i=1}^{K} N_i M_i\right)$ 의 시간이 소요되므로 더 효율적인 방법을 생각해야 합니다.
- 이를 위해서는 여러 가지 방법이 가능합니다. 대부분 위에서 설명한 과정을 잘 시뮬레이션하는 방식입니다.
- 두 RD-경로를 뒤에서부터 보면서 적절히 $N_i, M_i$ 를 줄여주는 식으로 $\mathcal{O}\left(\sum_{i=1}^{K} N_i + M_i\right)$ 에 구현할 수 있습니다. 어렵지 않으니 설명은 생략하겠습니다.
- 추가로 상대와 실제로 게임을 진행할 때 매 순간 움직여야 하는 말이 무엇인지 또한 효율적으로 구할 수 있습니다.

# L. Make Different

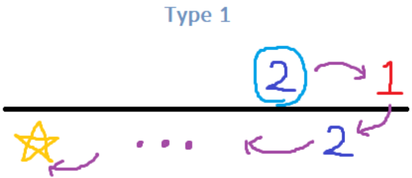
`suffix_array`
출제진 의도 – **Challenging**

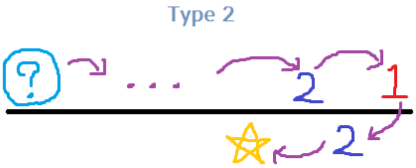
- 제출 18번, 정답 0팀 (정답률 0.00%)
- 처음 푼 팀: –
- 출제자: `functionx`

- 방향을 바꾸는 횟수는 무조건 1회 이하입니다.
- 방향을 2번 이상 바꾸는 경로가 있다면, 방향을 덜 바꾸면서 더 짧은 경로를 만들 수 있습니다.

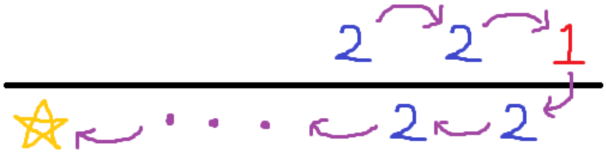


- 처음에 시계방향으로 움직이는 경로만 구해봅시다. 처음에 반시계방향으로 시작하는 경우에도 시계방향과 똑같은 알고리즘으로 문제를 해결할 수 있습니다.
- 경로를 시계방향으로 움직이는 횟수와 반시계방향으로 움직이는 횟수 차이에 따라 두 가지 유형으로 나눌 수 있습니다.

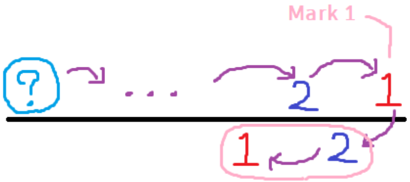


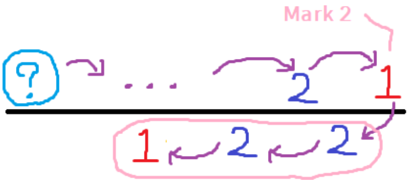



- 첫 번째 유형에서는 방향을 바꾸는 지점은 무조건 1이고 시작점에서 꺾는 지점까지는 모두 2여야 합니다.
- 방향을 꺾을 수 있는 지점이 무조건 한 가지만 나옵니다.
- 방향을 꺾은 이후 최단거리는 Suffix Array와 LCP를 이용하여 구합니다.

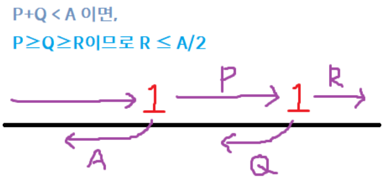


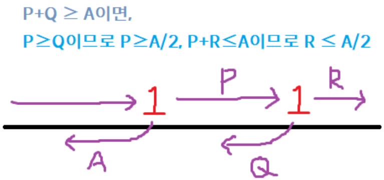
- 두 번째 유형에서는 방향을 바꾸는 지점은 무조건 1이고 방향을 바꾼 이후에는 계속 2번 스프링을 밟다가 끝납니다.
- 모든 1에 대하여 반시계방향으로 쭉 갈 때 밟는 2의 개수를 구해줍니다.
- 매 쿼리마다 방향을 꺾을 수 있는 지점을 Suffix Array와 LCP를 이용하여 구합니다.



- 놀랍게도 방향을 꺾는 지점은 최대 $\log_2 N + 1$ 개만 봐도 됩니다.
- 첫 번째 경로에서 반시계방향으로 움직인 횟수를 A라고 하면, 첫 번째 꺾는 지점에서 거리가 A 미만인 경로만 추가로 보면 됩니다.
- 두 번째 경로에서 최단경로가 경신이 되든 안되든, 그 다음에는 두 번째 꺾는 지점에서 거리가 A/2 미만인 경로만 추가로 보면 됩니다.



(U)

- Suffix Array를 구축하는 데 $\mathcal{O}(N \log N)$ 또는 $\mathcal{O}\left(N \log^2 N\right)$ 가 필요합니다.
- 쿼리를 처리하는데 $\mathcal{O}\left(Q \log^2 N\right)$ 가 필요합니다.