

# 2022 ICPC Sinchon Summer Algorithm Camp Contest Solution

Official Solutions

신촌지역 대학생 프로그래밍 동아리 연합

2022년 8월 20일

문제	의도한 난이도	출제자
<b>1A</b> 단순한 문제 (Small)	<b>Beginner</b>	tlsdydaud1
<b>1B</b> 에어팟	<b>Beginner</b>	lms0806
<b>1C</b> 장신구 명장 임스	<b>Easy</b>	lms0806
<b>1D</b> 기술 연계마스터 임스	<b>Easy</b>	lms0806
<b>1E</b> 핸들 뭉로 하지	<b>Medium</b>	gumgood
<b>1F</b> Tipover Transform	<b>Hard</b>	tlsdydaud1
<b>1G</b> 무자비한 최단경로	<b>Hard</b>	djs100201
<b>2A</b> 단순한 문제 (Large)	<b>Easy</b>	tlsdydaud1
<b>2B</b> 토큰	<b>Medium</b>	tlsdydaud1
<b>2C</b> 반짝반짝 3	<b>Medium</b>	lky7674
<b>2D</b> 수열의 점수	<b>Hard</b>	gumgood
<b>2E</b> Mexor Tree	<b>Hard</b>	djs100201
<b>2F</b> 전깃줄 연결	<b>Challenging</b>	gumgood
<b>2G</b> 까다로운 형제	<b>Challenging</b>	gumgood

# 1A. 단순한 문제 (Small)

bruteforcing, math

출제진 의도 - **Beginner**

- ✓ 제출 59번, 정답 52명 (정답률 88.136%)
- ✓ 처음 푼 사람: **junlee1216**, 2분
- ✓ 출제자: t1sdydaud1

## 1A. 단순한 문제 (Small)

- ✓  $T$ 와  $a, b, c$ 의 범위가 작으므로 가능한 쌍을 모두 탐색하면 됩니다.
- ✓ 시간복잡도는  $O(Tabc)$ 입니다.

## 1B. 에어팟

simulation, implementation

출제진 의도 – **Beginner**

- ✓ 제출 113번, 정답 46명 (정답률 40.708%)
- ✓ 처음 푼 사람: **kgmin604**, 11분
- ✓ 출제자: lms0806

## 1B. 에어팟

- ✓ 이전 핸드폰 종류와 이후 연결할 핸드폰의 종류를 비교하면서 풀면 됩니다.
- ✓ 현재 에어팟의 소모량이 100 이상이 되면, 소모량은 초기상태로 돌아갑니다.
- ✓ 출력은 누적 소모량이 아닌, 현재 에어팟의 소모량을 출력하시면 됩니다.

# 1C. 장신구 명장 임스

sort, greedy

출제진 의도 - **Easy**

- ✓ 제출 78번, 정답 49명 (정답률 62.821%)
- ✓ 처음 푼 사람: **gong29**, 11분
- ✓ 출제자: lms0806

## 1C. 장신구 명장 임스

- ✓ 메이플스토리 전문 기술 시스템을 문제로 만들어 보았습니다.
- ✓ 각각의 장신구를 만드는데 소모되는 피로도를 기준으로 정렬하여 푸시면 됩니다.
- ✓ 피로도가 200 미만인 경우 만들 수 있는 장신구가 남았을 때, 만들 수 있습니다.

# 1D. 기술 연계마스터 임스

stack, implementation, simulation

출제진 의도 - **Easy**

- ✓ 제출 109번, 정답 39명 (정답률 36.697%)
- ✓ 처음 푼 사람: **youvebeen**, 20분
- ✓ 출제자: lms0806

- ✓ 1 - 9, L - R, S - K로 기술 종류가 주어집니다.
- ✓ L이나 S를 사용하지 않고, R이나 K를 사용하는 경우 더이상 스킬을 사용할 수 없게 됩니다.
- ✓ L과 S 스킬을 여러번 사용할 경우, R과 K 스킬도 여러번 사용이 가능해집니다.
- ✓ L과 S 스킬의 사용한 횟수들을 저장한 후, R과 K가 주어졌을 때 1번씩 없애는 방식으로 풀면  $O(N)$  풀이가 가능합니다.

# 1E. 핸들 뭘로 하지

bfs, greedy

출제진 의도 - **Medium**

- ✓ 제출 185번, 정답 0명 (정답률 00.000%)
- ✓ 처음 푼 사람: —, —분
- ✓ 출제자: gumgood

## 1E. 핸들 뭉로 하지

- ✓ 모든 노드는 루트에서 해당 노드로 가면서 얻을 수 있는 문자열과 대응됩니다.
  - 리프 노드는 효규가 얻을 수 있는 문자열에 해당합니다.
  - 리프를 제외한 노드는 효규가 얻을 수 있는 문자열의 접두사에 해당합니다.
- ✓ 모든 리프 노드에 해당하는 문자열 중 사전 순으로 가장 마지막에 오는 것을 찾는 문제입니다.
- ✓ 하지만 모든 리프 노드에 대해 문자열을 단순 비교하는 경우 시간복잡도는  $O(N^2)$ 으로 TLE(시간 초과)를 받게 됩니다.

## 1E. 핸들 뭉로 하지

다음과 같이 관찰해봅시다.

- ✓ 두 노드의 깊이가 같다면 대응되는 문자열의 길이 또한 같습니다.
- ✓ 이때 두 문자열이 다르다면 사전 상 앞에 오는 노드와 그 자식들은 정답이 될 수 없습니다.
  - 따라서 정답이 될 수 있는 후보 노드들은 모두 같은 문자열에 대응됩니다.
  - 즉, 이 문자열이 정답의 접두사가 됩니다.
- ✓ 따라서 트리를 깊이 순으로 탐색하면서 정답 문자열을 앞 글자부터 하나씩 결정할 수 있습니다.

## 1E. 핸들 뭉로 하지

1. 먼저, 루트 노드가 후보 노드가 됩니다.
2. 후보 노드들의 자식 노드 중 사전 순으로 가장 마지막에 오는 노드들을 고릅니다.
  - 각 깊이의 후보 노드들은 앞 글자들이 모두 결정되어있기 때문에 노드에 적혀 있는 문자 하나만 비교하면 됩니다.
  - 이렇게 고른 노드들이 새로운 후보 노드가 됩니다.
3. 후보 노드를 갱신하는 2. 과정을 더 이상 자식 노드가 없을 때까지 반복합니다.
4. 마지막에 남아있는 후보 노드들이 정답 문자열에 대응하는 노드가 됩니다.

## 1E. 핸들 뭉로 하지

- ✓ 각 깊이마다 문자가 하나가 결정되므로 이를 이어붙이면 정답이 됩니다.
- ✓ 트리를 깊이 순으로 탐색하는 BFS 알고리즘으로 구현할 수 있습니다.
- ✓ 따라서 전체 시간복잡도  $O(N)$ 에 해결할 수 있습니다.

# 1F. Tipover Transform

dp

출제진 의도 - **Hard**

- ✓ 제출 1번, 정답 0명 (정답률 00.000%)
- ✓ 처음 푼 사람: —, —분
- ✓ 출제자: t1sdydaud1

- ✓ 게임을 클리어하기 위해 추가해야 하는 큐브 블록의 수는 블록들을 쓰러뜨린 후 블록이 놓이지 않은 칸의 수와 같습니다.
- ✓ 추가해야 하는 큐브 블록의 수를 최소화하려면 블록들을 적절하게 쓰러뜨려 최대한 많은 칸에 블록이 놓이게 해야 합니다.
- ✓ 블록들을 적절히 쓰러뜨려 최대 몇 개의 칸을 덮을 수 있는지 구해 봅시다.

- ✓ 0번 칸에 있는 큐브 블록을 제외하고, 1번부터  $N$ 번 칸에 존재하는 블록들에 왼쪽부터 차례로 번호를 매긴 다음, 다음과 같이 DP 상태를 정의합니다.
  - $a[i]$ :  $i$ 번째 블록까지 고려했을 때,  $i$ 번째 블록을 왼쪽으로 쓰러뜨려서 덮을 수 있는 칸 수의 최대값
  - $b[i]$ :  $i$ 번째 블록까지 고려했을 때,  $i$ 번째 블록을 쓰러뜨리지 않고 덮을 수 있는 칸 수의 최대값
  - $c[i]$ :  $i$ 번째 블록까지 고려했을 때,  $i$ 번째 블록을 오른쪽으로 쓰러뜨려서 덮을 수 있는 칸 수의 최대값
- ✓ 만약 어떤 블록을 어떤 방향으로 쓰러뜨리는 것이 불가능할 경우, 해당하는 DP 값은  $-\infty$ 로 설정합니다. 또한  $a[0] = b[0] = c[0] = 0$ 으로 간주합니다.

- ✓  $i$ 번째 블록이 놓인 칸 번호를  $pos[i]$ , 그 블록의 높이를  $h[i]$  라고 하고, DP 식을 세워 봅시다.  
편의상  $pos[0] = h[0] = 0$ 으로 간주합니다.
- ✓  $a[i]$ 
  - $i$ 번째 블록을 왼쪽으로 쓰러뜨렸을 때 덮이는 칸의 번호는  $(pos[i] - h[i])$  부터  $(pos[i] - 1)$  까지입니다.
  - $pos[i] - h[i] < 1$ 인 경우, 왼쪽에 충분한 공간이 없으므로  $a[i] = -\infty$ 입니다.
  - $pos[i - 1] > pos[i] - h[i]$ 인 경우,  $(i - 1)$ 번째 블록 때문에  $i$ 번째 블록을 왼쪽으로 쓰러뜨릴 수 없으므로  $a[i] = -\infty$ 입니다.

✓  $a[i]$

– 그 밖의 경우,  $(i - 1)$  번째 블록의 위치에 따라 DP 식이 달라집니다.

▶  $pos[i - 1] = pos[i] - h[i]$  인 경우:  $a[i] = a[i - 1] + h[i]$

▶  $pos[i - 1] < pos[i] - h[i] \leq pos[i - 1] + h[i - 1]$  인 경우:

$$a[i] = \max(a[i - 1], b[i - 1]) + h[i]$$

▶  $pos[i - 1] + h[i - 1] < pos[i] - h[i]$  인 경우:

$$a[i] = \max(a[i - 1], b[i - 1], c[i - 1]) + h[i]$$

- ✓  $b[i]$
- $i$  번째 블록을 쓰러뜨리지 않습니다. DP 식의 조건 분기가 비교적 간단해집니다.
    - ▶  $pos[i - 1] + h[i - 1] \geq pos[i]$  인 경우:  $b[i] = \max(a[i - 1], b[i - 1]) + 1$
    - ▶  $pos[i - 1] + h[i - 1] < pos[i]$  인 경우:  $b[i] = \max(a[i - 1], b[i - 1], c[i - 1]) + 1$

- ✓  $c[i]$
- $i$  번째 블록을 오른쪽으로 쓰러뜨립니다. 이때 블록이 덮는 칸의 번호는  $(pos[i] + 1)$  부터  $(pos[i] + h[i])$  까지입니다.
  - 일단 오른쪽에 충분한 공간이 있어야 하므로,  $pos[i] + h[i] > N$  인 경우  $c[i] = -\infty$  입니다.
  - $c[i]$  는  $i$  번째 블록까지만을 고려하므로 오른쪽에 있는 블록의 존재는 여기서 고려하지 않습니다. 따라서 DP 식은 다음과 같이 정리됩니다.
    - ▶  $pos[i - 1] + h[i - 1] > pos[i]$  인 경우:  $c[i] = \max(a[i - 1], b[i - 1]) + h[i]$
    - ▶  $pos[i - 1] + h[i - 1] \leq pos[i]$  인 경우:  $c[i] = \max(a[i - 1], b[i - 1], c[i - 1]) + h[i]$

- ✓ 모든 DP 값을 다 구했다면 답을 출력합니다.  $1 \sim N$  번 칸에 존재하는 블록의 수가  $v$  일 경우, 출력해야 하는 답은  $N - \max(a[v], b[v], c[v])$  입니다.

# 1G. 무자비한 최단경로

dijkstra

출제진 의도 - **Hard**

- ✓ 제출 10번, 정답 0명 (정답률 00.000%)
- ✓ 처음 푼 사람: —, —분
- ✓ 출제자: djs100201

- ✓ 잘 알려진 테크닉을 배웠으면 하는 마음에 출제하게 되었습니다.
- ✓ 좌표 차이가 비용인 여러 문제가 출제되기에 꼭 알아가도록 합시다.

- ✓ 우선  $x, y$  좌표 부터 해결하고,  $z$  좌표를 해결해 보도록 합시다.
- ✓ 단순히 모든 정점 쌍에 대해서 간선을 계산 후 이어주게 되면, 너무 많은 간선이 생겨서 문제를 시간 내에 해결할 수 없습니다.
- ✓  $x, y$  좌표가 인접한 정점 끼리만 간선을 이어주면 된다는 것이 핵심입니다.
- ✓ 이는 인접하지 않은 쌍 끼리의 간선은, 인접한 정점만이 간선으로 연결되어 있는 경로로 동일하게 나타낼 수 있기 때문입니다.

- ✓ 이제  $z$  좌표를 해결해 봅시다.
- ✓ 이는 추가 정점을 설정함으로써 문제를 해결할 수 있습니다.
- ✓ 각각  $k$ 로 나눈 나머지에 따라서  $n + 1, n + 2, \dots, n + k$  번 정점을 만들어 줍시다.
- ✓ 우리는  $z$  좌표를 이용하는 간선은  $n + 1, n + 2, \dots, n + k$  번의 추가 정점을 중간에 순회하여 이동할 것입니다.

- ✓ 정점  $z_i$  를  $k$  로 나눈 나머지를  $x$  라고 하고,  $k - x$  를  $k$  로 나눈 나머지를 다시  $y$  라고 합시다.
- ✓ 정점  $i$  에서  $n + x + 1$  로 가중치  $z_i$  인 간선을, 정점  $n + y + 1$  에서 정점  $i$  로 가중치  $z_i$  인 간선을 이어줍니다.
- ✓ 이렇게 이어주게 되면, 문제에 주어진 모든 간선을 그래프가 정확하게 표현할 수 있습니다.
- ✓ 간선은  $4N$  개 정도로, Dijkstra 알고리즘을 사용하여 문제를 해결할 수 있습니다.

## 2A. 단순한 문제 (Large)

number\_theory, math

출제진 의도 - **Easy**

- ✓ 제출 13번, 정답 12명 (정답률 92.308%)
- ✓ 처음 푼 사람: **luciaholic**, 5분
- ✓ 출제자: t1sdydaud1

## 2A. 단순한 문제 (Large)

- ✓ 입력값의 범위가 크므로 완전 탐색으로는 제한 시간 안에 문제를 풀 수 없습니다.
- ✓ 주어진 식을 잘 관찰해 봅시다.

- ✓  $(x \bmod y) = (y \bmod z) = (z \bmod x) = k$ 라고 하면, 나머지 연산의 성질에 의해서  $k < x$ ,  $k < y$ ,  $k < z$ 입니다.
- ✓ 따라서  $(x \bmod y) < x$ ,  $(y \bmod z) < y$ ,  $(z \bmod x) < z$ 입니다.
- ✓  $(x \bmod y) < x$ 는  $x \geq y$ 와 동치입니다. 같은 논리로  $y \geq z$ ,  $z \geq x$ 입니다.
- ✓ 종합하면  $x \geq y \geq z \geq x$ 이며, 이것이 성립하는 경우는  $x = y = z$ 인 경우뿐입니다.
- ✓ 따라서  $\min(a, b, c)$ 가 조건을 만족하는 쌍의 개수와 같음을 알 수 있습니다.

## 2B. 토큰

SCC

출제진 의도 - **Medium**

- ✓ 제출 23번, 정답 10명 (정답률 43.478%)
- ✓ 처음 푼 사람: **ok10p**, 15분
- ✓ 출제자: t1sdydaud1

- ✓ 첫 번째 상태에서 두 번째 상태를 거쳐서 다시 첫 번째 상태로 돌아오기 위한 조건을 알아봅시다.
- ✓ 간선을 따라 이동해서 어떤 정점  $u$  로 갈 수 있는 정점의 집합을  $S(u)$  라고 하면, 토큰을 어떻게 움직이더라도  $S(u)$  에 속하는 정점들 위에 놓인 토큰의 수  $c[u]$  는 증가하지 않습니다.
- ✓ 첫 번째 상태에서 두 번째 상태를 만드는 것이 가능하려면 각각의 정점  $i$  에 대해  $c[i]$  가 줄어드는 경우가 없어야 합니다. 이것은 두 번째 상태에서 첫 번째 상태를 만들 때도 똑같이 적용되므로, 첫 번째 상태와 두 번째 상태를 비교했을 때 모든  $i$  에 대해 첫 번째 상태의  $c[i]$  값과 두 번째 상태의  $c[i]$  값이 같아야 합니다.

- ✓ 토큰을 움직였을 때  $c[i]$ 의 값이 하나라도 변했을 경우, 토큰이 움직이기 전에 있던 정점과 움직인 정점은 서로 다른 SCC에 속합니다.
- ✓ 따라서 미션을 성공적으로 수행하기 위해서는 토큰을 다른 SCC로 움직이면 안 되며, 그래프의 어떤 SCC에 속한 정점들에 놓인 토큰의 수가 두 상태에서 서로 다른 경우가 존재한다면 답이 NO임을 알 수 있습니다.

- ✓ 그런 경우가 존재하지 않을 경우 답이 항상 YES임을 증명합니다.
- ✓ 앞에서 답이 NO인 경우를 걸러내고 나면 모든 SCC가 두 상태에서 같은 수의 토큰을 포함하는 경우만 남습니다. 또한 토큰을 다른 SCC로 움직일 일이 없어졌기 때문에 각각의 SCC를 따로 처리해도 됩니다. 따라서 한 SCC 안에서 가능한 모든 상태 간에 서로 전환이 가능함을 보이면 됩니다.

- ✓ 어떤 SCC에서 토큰이 놓인 정점  $u$ 와 토큰이 놓이지 않은 정점  $v$ 를 임의로 선택하고,  $u$ 에서  $v$ 로 가는 경로를 아무거나 하나 고릅니다. 이 경로는 같은 정점을 두 번 이상 지나지 않아야 합니다.
- ✓ 이 경로에는 토큰이 하나 이상 놓여 있습니다. 토큰이 놓인 정점의 번호를 경로에서 나타나는 순서대로  $p_1, p_2, \dots, p_k$ 라고 합시다.
- ✓ 경로에서 정점  $p_k$ 와  $v$  사이에는 토큰이 존재하지 않습니다. 따라서 정점  $p_k$ 에서 경로를 따라 정점  $v$ 로 토큰을 움직일 수 있습니다.

- ✓ 같은 방식으로 정점  $p_{k-1}$  에서 정점  $p_k$  로 토큰을 움직이고, 정점  $p_{k-2}$  에서 정점  $p_{k-1}$  로 토큰을 움직이고, ... 정점  $p_1$  에서 정점  $p_2$  로 토큰을 움직입니다. 이제 정점  $p_1$  에 있던 토큰이 없어지고 정점  $v$  에 토큰이 놓였고, 나머지 정점의 상태는 처음과 동일합니다.  $p_1$  과  $u$  는 동일한 정점이므로 정점  $u$  에서 정점  $v$  로 토큰을 움직인 것과 같은 효과를 내는 데 성공했습니다.
- ✓ 이 방법을 반복하면 한 SCC 안에서 가능한 모든 상태 간에 전이가 가능함을 알 수 있습니다.

## 2C. 반짝반짝 3

linearity\_of\_expectation, probability, tree  
출제진 의도 - **Medium**

- ✓ 제출 37번, 정답 4명 (정답률 10.811%)
- ✓ 처음 푼 사람: **portal3046**, 60분
- ✓ 출제자: 1ky7674

- ✓ 중급 4회차 필수 문제이자 SUAPC 2021 Summer의 기출문제인 반짝반짝 2에서 확장시킨 문제입니다.
- ✓ 우선 쉬운 버전인 반짝반짝 2부터 풀어보도록 하겠습니다.
- ✓ 불이 들어온 기존 전구 개수의 확률변수를  $X$ , 불이 들어온 추가 전구 개수의 확률변수를  $Y$  라고 합시다.
- ✓ 불이 들어온 전구 개수의 기댓값은  $E(X + Y)$ 로 표기할 수 있습니다.

**Theorem** : 임의의 확률변수  $X, Y$ 에 대해서  $E(X + Y) = E(X) + E(Y)$ 를 만족한다.

- ✓ 위의 Theorem은 확률변수  $X, Y$ 가 서로 독립이 아니어도 성립합니다.
- ✓ 때문에  $E(X)$ 와  $E(Y)$ 를 따로 계산하면 됩니다.

- ✓  $i$  번째 전구가 불이 들어오는지에 대한 확률변수를  $X_i$  로 정의합니다.

$$X_i = \begin{cases} 1, & \text{if } i\text{번째 전구가 켜진 경우} \\ 0, & \text{if } i\text{번째 전구가 꺼진 경우} \end{cases}$$

- ✓  $E(X_i) = p_i$  를 만족합니다.

- ✓  $X_i$ 의 정의에 따라서 확률변수  $X$ 를  $X = \sum_{i=1}^N X_i$ 로 표현할 수 있습니다.
- ✓ 따라서  $E(X) = E\left(\sum_{i=1}^N X_i\right) = \sum_{i=1}^N E(X_i) = \sum_{i=1}^N p_i$ 를 만족합니다.

- ✓  $i$  번째 전구와  $i + 1$  번째 전구 사이에 있는 추가 전구를  $i$  번째 추가 전구라고 합니다.
- ✓  $i$  번째 추가 전구에 불이 들어오는지에 대한 확률변수를  $Y_i$  로 정의합니다.

$$Y_i = \begin{cases} 1, & \text{if } i\text{ 번째 추가 전구가 켜진 경우} \\ 0, & \text{if } i\text{ 번째 추가 전구가 꺼진 경우} \end{cases}$$

- ✓  $E(Y_i) = p_i(1 - p_{i+1}) + p_{i+1}(1 - p_i)$  를 만족합니다.

- ✓  $Y_i$ 의 정의에 따라서 확률변수  $Y$ 를  $Y = \sum_{i=1}^{N-1} Y_i$ 로 표현할 수 있습니다.
- ✓ 따라서  $E(Y) = E\left(\sum_{i=1}^{N-1} Y_i\right) = \sum_{i=1}^{N-1} E(Y_i) = \sum_{i=1}^{N-1} \{p_i(1 - p_{i+1}) + p_{i+1}(1 - p_i)\}$ 를 만족합니다.

- ✓ 이 풀이를 트리 버전으로 확장시킬 수 있습니다.
- ✓ 주어진 트리를  $T$  라고 하겠습니다.
- ✓ 정점에 붙어있는 불이 들어온 전구 개수의 기댓값은  $\sum_{i=1}^N p_i$  로 동일합니다.
- ✓ 간선에 붙어있는 불이 들어온 전구 개수의 기댓값은  $\sum_{(u,v) \in T} p_u(1 - p_v) + p_v(1 - p_u)$  가 됩니다.

- ✓ 이제  $p_i$  를  $p'_i$  로 바뀌었을 때, 불이 들어온 전구 개수의 기댓값을  $O(1)$  에 구해봅시다.
- ✓ 정점 부분은 간단합니다.  $\sum_{i=1}^N p_i$  을 저장하는 변수 하나를 만들고, 이 값에  $p_i$  를 빼고  $p'_i$  를 더하는 식으로 관리하면 됩니다.

- ✓ 간선 부분은 조금 생각해야 합니다.
- ✓ 정점의 degree의 최댓값이  $N - 1$  개이기 때문에, 이웃한 간선들을 모두 탐색하면서 기댓값을 계산하면 시간복잡도가  $O(QN)$  이 되므로 시간 초과가 발생합니다.
- ✓ 이를 좀 더 최적화 해봅시다.

$$\sum_{(u,v) \in T} p_u(1-p_v) + p_v(1-p_u) = \sum_{i=1}^N \left( p_i \sum_{j \in \text{child}(i)} (1-p_j) + (1-p_i) \sum_{j \in \text{child}(i)} p_j \right)$$

- ✓ 간선에 붙어있는 불이 들어온 전구 개수의 기댓값에 대한 수식을 위와 같이 변형할 수 있습니다.
- ✓ 각 정점 별로 자식들의  $p_j$ 의 합과  $(1-p_j)$ 의 합을 저장하는 변수를 만들고 관리하면 됩니다.
- ✓  $p_i$ 를  $p'_i$ 로 바뀌었을 때,  $i$ 번 노드의 부모 노드에 저장된 2개의 변수 값을 변경하고 이에 따라서 기댓값을 계산하면 됩니다.
- ✓ 최근에 이와 비슷한 방법으로 풀 수 있는 문제가 KOI 2022 2차 고등부 1번으로 출제되었습니다. 이 문제도 같이 풀어보시면 좋습니다.

## 2D. 수열의 점수

divide\_and\_conquer, prefix\_sum

출제진 의도 - **Hard**

- ✓ 제출 4번, 정답 0명 (정답률 00.000%)
- ✓ 처음 푼 사람: —, —분
- ✓ 출제자: gumgood

## 2D. 수열의 점수

Divide and Conquer 전략으로 해결할 수 있습니다.

- ✓ 구간  $[s, e]$ 에 속하는 모든 부분 수열의 점수를 구하는 함수를  $f(s, e)$ 라고 합시다.
- ✓ 해당 구간을 절반으로 나눠 생각해봅시다. 구간의 중앙을  $m$ 이라 하면, 이 구간에 속하는 모든 부분 수열을 다음과 같이 나눌 수 있습니다.
  1.  $[s, m]$ 에 속하는 부분 수열
  2.  $[m + 1, e]$ 에 속하는 부분 수열
  3.  $b_m$ 과  $b_{m+1}$ 을 모두 포함하는 부분 수열
- ✓ 1, 2번의 경우는 각각  $f(s, m)$ 와  $f(m + 1, e)$ 의 값과 같습니다.
- ✓ 3번 경우인  $b_m$ 과  $b_{m+1}$ 을 모두 포함하는 부분 수열의 점수를 구해보겠습니다.

## 2D. 수열의 점수

- ✓ 3번에 해당하는 수열은  $s \leq l \leq m$  과  $m + 1 \leq r \leq e$  를 만족하는  $b_l, \dots, b_r$  과 같습니다.
- ✓  $r$  을 고정해놓고 모든  $l$  에 대해 수열의 점수를 구해봅시다.
- ✓ 먼저 오른쪽 구간에 해당하는  $\min(b_{m+1}, b_r)$  와  $\max(b_{m+1}, b_r)$  를 계산합니다.
- ✓ 그러면 왼쪽 구간은  $\min, \max$  값에 따라 각각 두 구간으로 나뉘게 됩니다.
  - $\min(b_l, b_m) \geq \min(b_{m+1}, b_r)$  인 구간과  $\min(b_l, b_m) < \min(b_{m+1}, b_r)$  인 구간
  - $\max(b_l, b_m) \leq \max(b_{m+1}, b_r)$  인 구간과  $\max(b_l, b_m) > \max(b_{m+1}, b_r)$  인 구간
- ✓ 즉, 특정  $r$  에 대해 왼쪽 구간의 모든  $l$  을 네 가지의 경우로 나눌 수 있습니다.

## 2D. 수열의 점수

3.1  $\min(b_l, b_m) \geq \min(b_{m+1}, b_r), \max(b_l, b_m) > \max(b_{m+1}, b_r)$

- $\min(b_l, b_r) = \min(b_{m+1}, b_r)$  이고  $\max(b_l, b_r) = \max(b_l, b_m)$  인 구간입니다.
- 이 구간의 점수는 여기에 속하는 모든  $l$ 에 대해  $\max(b_l, b_m)$ 의 합과  $\min(b_{m+1}, b_r)$ 을 곱한 값으로 계산할 수 있습니다.

3.2  $\min(b_l, b_m) < \min(b_{m+1}, b_r), \max(b_l, b_m) \leq \max(b_{m+1}, b_r)$

- $\min(b_l, b_r) = \min(b_l, b_m)$  이고  $\max(b_l, b_r) = \max(b_{m+1}, b_r)$  인 구간입니다.
- 이 구간의 점수는 여기에 속하는 모든  $l$ 에 대해  $\min(b_l, b_m)$ 의 합과  $\max(b_{m+1}, b_r)$ 을 곱한 값으로 계산할 수 있습니다.

## 2D. 수열의 점수

3.3  $\min(b_l, b_m) \geq \min(b_{m+1}, b_r), \max(b_l, b_m) \leq \max(b_{m+1}, b_r)$

- $\min(b_l, b_r) = \min(b_{m+1}, b_r)$  이고  $\max(b_l, b_r) = \max(b_{m+1}, b_r)$  인 구간입니다.
- 해당 구간에 속하는 각 부분 수열의 점수는  $\min(b_{m+1}, b_r) \cdot \max(b_{m+1}, b_r)$  입니다.

3.4  $\min(b_l, b_m) < \min(b_{m+1}, b_r), \max(b_l, b_m) > \max(b_{m+1}, b_r)$

- $\min(b_l, b_r) = \min(b_l, b_m)$  이고  $\max(b_l, b_r) = \max(b_l, b_m)$  인 구간입니다.
- 해당 구간에 속하는 각 부분 수열의 점수는  $\min(b_l, b_m) \cdot \max(b_l, b_m)$  입니다.

## 2D. 수열의 점수

- ✓ 3.3에서  $\min(b_{m+1}, b_r) \cdot \max(b_{m+1}, b_r)$ 은 고정된 값으로  $O(1)$ 에 계산할 수 있습니다.
- ✓ 3.1, 3.2, 3.4는 각각  $\max(b_l, b_m)$ ,  $\min(b_l, b_m)$ ,  $\min(b_l, b_m) \cdot \max(b_l, b_m)$ 에 대해 구간 합이 필요합니다.
  - prefix array를 이용해 전처리해두면 모두  $O(1)$ 에 얻을 수 있습니다.
- ✓ 이를 모든  $m + 1 \leq r \leq e$ 에 대해 반복하면 됩니다.

- ✓ 구간의 길이를  $n = e - s + 1$  이라고 했을 때, 구간을 나누는 경계는  $O(n)$  는 계산할 수 있습니다.
- ✓ prefix array 또한  $O(n)$  에 한 번만 전처리해두면 모든  $r$  에 대해  $O(1)$  에 구간 합을 계산할 수 있습니다.
- ✓ 따라서 구간  $[s, e]$  에서  $b_m$  과  $b_{m+1}$  을 모두 포함하는 부분 수열의 점수를  $O(n)$  에 계산할 수 있습니다.

- ✓  $f(s, e)$ 의 시간복잡도를  $T(n)$ 이라고 하면  $T(n) = 2 \cdot T(n/2) + O(n)$ 을 만족하므로 전체 시간복잡도  $O(N \log N)$ 에 해결할 수 있습니다.
- ✓ monotone stack, segment tree를 이용하여  $O(N \log N)$ 에 해결할 수도 있습니다.

## 2E. Mexor Tree

lca, dfs

출제진 의도 - **Hard**

- ✓ 제출 10번, 정답 0명 (정답률 00.000%)
- ✓ 처음 푼 사람: —, —분
- ✓ 출제자: djs100201

- ✓ 이 문제는 dfs를 잘 쓰길 바라는 마음에 문제를 출제하게 되었습니다.
- ✓ 문제 제목은 MEX 연산과 XOR 연산을 합친 언어유희입니다.

- ✓ 이 문제는 크게 두 부분으로 나누어져 있습니다.
- ✓ XOR 쿼리를 처리하는 첫 번째 부분과 수열  $b_1, b_2, \dots, b_N$  을 구하는 두 번째 부분입니다.
- ✓ 우선 XOR 쿼리를 먼저 처리해 봅시다.
- ✓ *hld*(*heavy light decomposition*) 라는 잘 알려진 테크닉을 이용하면  $O(M(\log N)^2)$  정도에 해결할 수 있지만, 더 쉽게 해결하는 방법이 있습니다.

- ✓ 쿼리  $(X, Y, Z)$ 가 입력으로 들어오게 됐을 때를 생각해 봅시다.
- ✓ 루트와  $X$ 까지의 단순경로, 루트와  $Y$ 까지의 단순경로를  $Z$ 와 XOR 연산을 시행한 값으로 업데이트 해 봅시다.
- ✓ 그렇다면  $X$ 와  $Y$ 의  $lca(X, Y)$ 를 제외한 모든 정점의 업데이트가 올바르게 이루어짐을 알 수 있습니다.
- ✓ 따라서  $lca(X, Y)$ 만 따로 추가적인 처리를 해 주면 됩니다.

- ✓ 왜 이런식으로 쿼리를 처리할까요?
- ✓ 이는 임의의 두 정점쌍에 대한 쿼리를 루트와 정점으로의 쿼리로 바꿀 수 있기 때문입니다.
- ✓ XOR 연산은 교환법칙이 성립합니다. 따라서 배열을 만들고 모은 후, 한번에 처리합니다.
- ✓  $val[300001]$  이라는 *int* 형 배열을 만든 후, 쿼리마다  $val[X], val[Y], a[lca(x, y)]$  이 3가지 부분에  $Z$ 와 XOR 연산을 해줍니다.
- ✓ 그 후 *dfs*를 이용해서 적절히 처리해주면 XOR 쿼리를 해결할 수 있습니다.

- ✓ 두 번째 부분을 처리해 봅시다.
- ✓ 이는 첫 번째 부분의 원리를 깨달았다면 비슷하게 처리할 수 있습니다.
- ✓ 여기서는  $S$ 가 고정되어 있기 때문입니다.
- ✓ 마찬가지로  $dfs$ 를 이용하면  $S$ 와 임의의 정점까지의 단순 경로상의 존재하는 모든 정점 값들의 집합을 알 수 있습니다.

- ✓ 따라서  $S$ 를 루트로 한 후  $dfs$ 를 하면서  $b_i$ 들을 구해주면 됩니다.
- ✓  $b_i$ 를 구하는 방법은  $set$ 이나  $segment\ tree$ 와 같은 자료구조로 처리해 줍시다.
- ✓ 따라서 전체 문제는  $O((M + N) \log N)$  정도에 해결됩니다.

## 2F. 전깃줄 연결

dp, monotone\_queue

출제진 의도 - **Challenging**

- ✓ 제출 1번, 정답 0명 (정답률 00.000%)
- ✓ 처음 푼 사람: —, —분
- ✓ 출제자: gumgood

## 2F. 전깃줄 연결

Dynamic Programming으로 해결할 수 있습니다.

- ✓  $dp[i]$  를  $i$  번째 전봇대에 전력을 공급하는 최소 비용으로 정의합니다.
- ✓  $1 \leq j < i$  인  $j$  번째 전봇대까지 최소 비용으로 전력을 공급한 후,  $i$  번째 전봇대에 전깃줄을 이어 전력을 공급하는 비용은 다음과 같습니다.

$$dp[j] + C_j + C_i - 2 \cdot \gcd(C_j, \dots, C_i) + \sum_{k=j+1}^{i-1} B_k$$

- ✓  $sB_i = \sum_{k=1}^i B_k$  로 정의합니다. 이때, 다음과 같은 식으로  $dp[i]$  값을 구할 수 있습니다.

$$dp[i] = C_i + sB_{i-1} + \min(dp[j] + C_j - sB_j) - 2 \cdot \gcd(C_j, \dots, C_i)$$

## 2F. 전깃줄 연결

- ✓ 각  $i$ 마다 모든  $j$ 에 대해 값을  $O(1)$ 에 갱신하더라도  $O(N^2)$ 으로 TLE(시간 초과)를 받게 됩니다.
- ✓ 다음과 같이 관찰해봅시다.
  - $j$ 부터  $i$  사이에 대해 가능한  $\gcd(C_j, \dots, C_i)$ 의 종류는  $\log C_i$ 개를 넘지 않습니다.
  - 즉, 1에서  $i - 1$ 까지의 구간을 gcd값에 따라 최대  $C_i$ 개의 구간으로 나눌 수 있습니다.
  - 각 구간마다 gcd값과  $dp[j] + C_j - sB_j$ 의 최소값을 저장하여  $\log C_i$ 개의 구간에 대해서만 확인하면 됩니다.

## 2F. 전깃줄 연결

- ✓ 이때 각 구간은  $j$ 가 증가함에 따라  $\gcd(C_j, \dots, C_i)$ 도 증가합니다.
  - $\gcd(C_j \dots C_i) \leq \gcd(C_{j+1} \dots C_i)$ 이기 때문입니다.
- ✓ 단조성을 유지하는 monotone stack으로 각 구간의 gcd값과  $dp[j] + C_j - sB_j$ 의 최소값을 관리할 수 있습니다.
- ✓ monotone stack의 최대 길이는 gcd의 종류와 같은  $\log C_i$ 개로  $O(\log C_i)$ 에 새로 구간을 나누고 각 구간에 저장할 값을 갱신할 수 있습니다.
- ✓ 따라서 전체 시간복잡도  $O(N \log(\max(C_i)))$ 에 해결할 수 있습니다.

## 2G. 까다로운 형제

dp, lis, sliding\_window

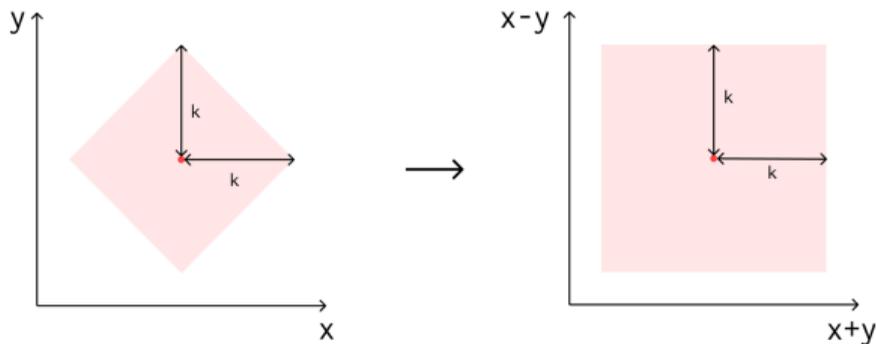
출제진 의도 - **Challenging**

- ✓ 제출 1번, 정답 0명 (정답률 00.000%)
- ✓ 처음 푼 사람: —, —분
- ✓ 출제자: gumgood

- ✓ 비슷한 문제가 많은 전형적인 유형의 문제입니다.
- ✓ 다양한 풀이가 존재하지만 그 중 가장 쉽다고 생각하는 풀이를 소개하겠습니다.

## 2G. 까다로운 형제

- ✓  $(x, y)$  좌표계 위 한 여행지에서 거리가  $k$  이하인 곳은 왼쪽 그림과 같습니다.
- ✓ 편의를 위해  $(x, y)$  좌표계를  $(u, v) = (x + y, x - y)$  좌표계로 변환하겠습니다.
- ✓ 좌표계를  $-45$ 도 회전하여 늘린 모양으로 모든 좌표값을 정수로 유지할 수 있습니다.



## 2G. 까다로운 형제

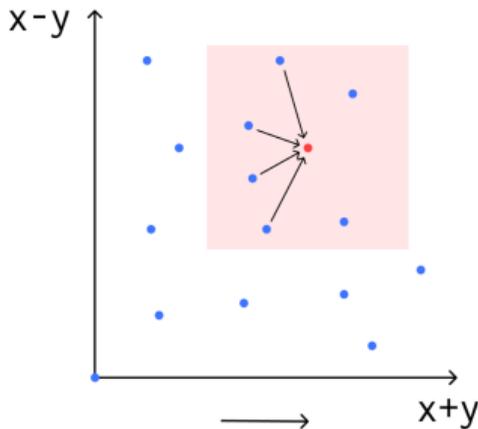
여행 경로에 필요한 조건은 다음과 같습니다.

1. 여행지는  $(x, y)$  좌표계에서 원점에서 멀어지는 순으로 선택해야 합니다.
  - 회전한 좌표계에서  $u = x + y$ 가 증가하는 순과 같습니다.
2. 연속한 두 여행지는  $(x, y)$  좌표계에서 거리가  $K$  이하를 만족해야 합니다.
  - 회전한 좌표계 위 두 점  $(u_1, v_1), (u_2, v_2)$ 에 대해  $|u_1 - u_2| \leq K$  이고  $|v_1 - v_2| \leq K$  인 것과 같습니다.

- ✓ 먼저 첫 번째 조건을 위해  $u$ 가 증가하는 순으로 여행지를 정렬합니다.
- ✓ 그러면  $i$  번째 여행지에 도달할 수 있는 후보를 1 번째부터  $i - 1$  번째 여행지로 제한하게 됩니다.
- ✓ 이제 Dynamic Programming으로 접근할 수 있습니다. 다음과 같이 DP 식을 정의하겠습니다.
  - $dp[i] = i$  번째 여행지에 도착했을 때 얻을 수 있는 최대 (만족도 합, 여행지 개수)

## 2G. 까다로운 형제

- ✓  $i$  번째 여행지의 변환된 좌표를  $(u_i, v_i)$  라고 합시다.
- ✓  $j < i$  인  $j$  번째 여행지에 대해  $u_i - K \leq u_j \leq u_i + K$  와  $v_i - K \leq v_j \leq v_i + K$  를 만족하는 곳에서  $i$  번째 여행지로 갈 수 있습니다.



## 2G. 까다로운 형제

- ✓ segment tree에 각  $v(= x - y)$  값에 해당하는 여행지의 DP값을 저장합니다.
- ✓ 각  $i$  번째 여행지를 기준으로  $v_i - K \leq v_j \leq v_i + K$  에 있는 여행지의 최소 DP값을 알 수 있습니다.
- ✓ 하지만  $u_j < u_i - K$  에 있는 여행지의 값은 제외해야 합니다.
- ✓  $u(= x + y)$  축을 따라 sliding window를 하면서 segment tree에 DP값을 추가하거나 제거하여 해결할 수 있습니다.

- ✓ 좌표의 범위가 크기 때문에 좌표 압축이 필요합니다.
- ✓ segment tree에 값을 제거하기 위해 queue 자료구조를 활용하거나 압축 전 좌표를 통해 쿼리의 범위를 잘 설정하는 방법이 있습니다.
- ✓ 전체 시간복잡도  $O(N \log N)$ 에 해결할 수 있습니다.