2019 AlKor 1학기 고급반 2주차 모의고사 풀이

고려대학교 정보보호학부 알고리즘 문제해결 동아리

Prepared by @evenharder

2019년 4월 13일

머릿말

- **풀이를 보기 전에 문제에 대해 더 생각을 해보세요!**
- **BOJ 101**과 **자주 틀리는 요인** 문서는 숙지하셔야 합니다.
- 곰곰히 생각해보면 풀 수 있는 문제들로 구성했습니다.
- 1주차보다 쉽게 했고, 보다 만족스러운 참여도와 결과가 보였습니다.
  처음 3문제는 모두 접근해볼만한 난이도로 준비하는 게 좋아보입니다.
  - 풀이 순서는 제가 느낀 난이도순이지만 애매한 부분이 많습니다.

머릿말

- 모든 문제를 지금 푸는 건 어려울 수 있습니다. 하나하나씩 풀어보세요.
- 때문에 (가능하면) 각 문제 별로 정답 코드를 제공하려고 합니다.
  - 대부분 정해 코드가 공식 대회 사이트에 있기에 비공개할 이유가 없습니다.
  - 코드를 보면 풀이를 보다 쉽게 이해할 수 있습니다.
  - 물론 어느 상황에서든 **표절은 절대 금지입니다.**
- 제출, 정답 및 정답율은 2019년 4월 6일 16:00 (UTC +9) 기준입니다.
- 힘들어서 해설지 퀄리티가 점점 하락하고 있습니다...

Problem A

속이기 (BOJ 11895)

출처 : (oj.uz) GA5 1번

제출 : 15 / 정답 : 15

평균 시도 : 1.27

FYI : bitwise xor(exclusive or)는 보통 ^(caret)을 씁니다.

FYI : bitwise xor(exclusive or)는 보통 ^(caret)을 씁니다.

$X_1 \oplus X_2 \oplus \cdots \oplus X_k = Y_1 \oplus Y_2 \oplus \cdots \oplus Y_{n-k}$?

FYI : bitwise xor(exclusive or)는 보통 ^(caret)을 씁니다.

$X_1 \oplus X_2 \oplus \cdots \oplus X_k = Y_1 \oplus Y_2 \oplus \cdots \oplus Y_{n-k}$?

- $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 0$와 동치.

FYI : bitwise xor(exclusive or)는 보통 ^(caret)을 씁니다.

$X_1 \oplus X_2 \oplus \cdots \oplus X_k = Y_1 \oplus Y_2 \oplus \cdots \oplus Y_{n-k}$?

- $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 0$와 동치.

즉, 모든 수를 xor해서 0일 때만 위의 조건을 만족하며 분할할 수 있다.

FYI : bitwise xor(exclusive or)는 보통 ^(caret)을 씁니다.

$X_1 \oplus X_2 \oplus \cdots \oplus X_k = Y_1 \oplus Y_2 \oplus \cdots \oplus Y_{n-k}$?

- $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 0$와 동치.

즉, 모든 수를 xor해서 0일 때만 위의 조건을 만족하며 분할할 수 있다.

- 분할이 가능하면 집합 Y에는 최소값만 넣으면 된다.

FYI : bitwise xor(exclusive or)는 보통 ^(caret)을 씁니다.

$X_1 \oplus X_2 \oplus \cdots \oplus X_k = Y_1 \oplus Y_2 \oplus \cdots \oplus Y_{n-k}$?

- $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 0$와 동치.

즉, 모든 수를 xor해서 0일 때만 위의 조건을 만족하며 분할할 수 있다.

- 분할이 가능하면 집합 Y에는 최소값만 넣으면 된다.

문제 이름 그대로 속이는, 좋은 낚시 문제입니다. 항상 지문을 꼼꼼히 읽읍시다!

Code : http://boj.kr/f9a9fb1dcd1a4f3bbbaf6f1ac8960e51

Problem C

이진 트리 (BOJ 13325)

출처 : ACM ICPC Daejeon Nationalwide Internet Competition 2016 A번

제출 : 10 / 정답 : 10

평균 시도 : 2

포화이진트리(perfect binary tree)가 주어진다.

포화이진트리(perfect binary tree)가 주어진다.

- root를 1번째로 두면, x번째 정점의 자식은 $2x$번째, $2x+1$번째

포화이진트리(perfect binary tree)가 주어진다.

- root를 1번째로 두면, x번째 정점의 자식은 $2x$번째, $2x+1$번째

leaf 바로 위의 정점들부터 보자.

포화이진트리(perfect binary tree)가 주어진다.

- root를 1번째로 두면, x번째 정점의 자식은 $2x$번째, $2x+1$번째

leaf 바로 위의 정점들부터 보자.

- 정점 x에 대해, 한쪽은 길이가 $d[2x]$, 한쪽은 $d[2x+1]$

포화이진트리(perfect binary tree)가 주어진다.

- root를 1번째로 두면, x번째 정점의 자식은 $2x$번째, $2x+1$번째

leaf 바로 위의 정점들부터 보자.

- 정점 x에 대해, 한쪽은 길이가 $d[2x]$, 한쪽은 $d[2x+1]$
- 이 경우 적은 쪽에 차만큼 더해주어야 한다.

포화이진트리(perfect binary tree)가 주어진다.

- root를 1번째로 두면, x번째 정점의 자식은 $2x$번째, $2x+1$번째

leaf 바로 위의 정점들부터 보자.

- 정점 x에 대해, 한쪽은 길이가 $d[2x]$, 한쪽은 $d[2x+1]$
- 이 경우 적은 쪽에 차만큼 더해주어야 한다.
- 그러면 그쪽 경로는 $\max(d[2x], d[2x+1])$로 통일된다.

포화이진트리(perfect binary tree)가 주어진다.

- root를 1번째로 두면, x번째 정점의 자식은 $2x$번째, $2x+1$번째

leaf 바로 위의 정점들부터 보자.

- 정점 x에 대해, 한쪽은 길이가 $d[2x]$, 한쪽은 $d[2x+1]$
- 이 경우 적은 쪽에 차만큼 더해주어야 한다.
- 그러면 그쪽 경로는 $\max(d[2x], d[2x+1])$로 통일된다.
- 이를 계속 반복하면 한 level를 해결 가능하고, 그럼 바로 위의 level도 마찬가지로...!

http://boj.kr/6535d79448104e9cb7e2e7ca225ec804

- $2^n - 1$에서 1까지 줄여가면 모든 정점을 순회 가능하다!
- 종종 쓰이는 트릭이니 (예시 : segment tree) 잘 알아두자.

Problem B

순열 그래프의 연결성 판별 (BOJ 7982)

출처 : AMPPZ 2012 I번

제출 : 13 / 정답 : 10

평균 시도 : 3.6

그래프는 수열 $\{a_i\}$에서 연결해서 생성할 수 있다. 뭘 알 수 있을까?

그래프는 수열 $\{a_i\}$에서 연결해서 생성할 수 있다. 뭘 알 수 있을까?

- 컴포넌트는 **연속**되어 나타난다.

그래프는 수열 $\{a_i\}$에서 연결해서 생성할 수 있다. 뭘 알 수 있을까?

- 컴포넌트는 **연속**되어 나타난다.
- 어떤 구간 L이 컴포넌트가 되었다고 가정하자. 그러면

그래프는 수열 $\{a_i\}$에서 연결해서 생성할 수 있다. 뭘 알 수 있을까?

- 컴포넌트는 **연속**되어 나타난다.
- 어떤 구간 L이 컴포넌트가 되었다고 가정하자. 그러면
  - L 뒤로는 L의 최대 원소보다 큰 값만 와야 하며

그래프는 수열 $\{a_i\}$에서 연결해서 생성할 수 있다. 뭘 알 수 있을까?

- 컴포넌트는 **연속**되어 나타난다.
- 어떤 구간 L이 컴포넌트가 되었다고 가정하자. 그러면
  - L 뒤로는 L의 최대 원소보다 큰 값만 와야 하며
  - L 앞에는 L의 최소 원소보다 작은 값만 와야 한다.

그래프는 수열 $\{a_i\}$에서 연결해서 생성할 수 있다. 뭘 알 수 있을까?

- 컴포넌트는 **연속**되어 나타난다.
- 어떤 구간 L이 컴포넌트가 되었다고 가정하자. 그러면
  - L 뒤로는 L의 최대 원소보다 큰 값만 와야 하며
  - L 앞에는 L의 최소 원소보다 작은 값만 와야 한다.
- 조금 정리해보면, $\max(a_1, a_2, \cdots, a_k) = k$일 때 컴포넌트가 분리된다.

그래프는 수열 $\{a_i\}$에서 연결해서 생성할 수 있다. 뭘 알 수 있을까?

- 컴포넌트는 **연속**되어 나타난다.
- 어떤 구간 L이 컴포넌트가 되었다고 가정하자. 그러면
  - L 뒤로는 L의 최대 원소보다 큰 값만 와야 하며
  - L 앞에는 L의 최소 원소보다 작은 값만 와야 한다.
- 조금 정리해보면, $\max(a_1, a_2, \cdots, a_k) = k$ 일 때 컴포넌트가 분리된다.
  - $\max(a_1, a_2, \cdots, a_k) = k$이면 $\{a_1, a_2, \cdots, a_k\} = \{1, 2, \cdots, k\}$

정답 출력은 각 컴포넌트의 최대 원소를 저장하면 어렵지 않게 할 수 있다.

Code : http://boj.kr/f88c861f7c114fabbb214e24840d2fd5

Problem D

Bricks (BOJ 10510)

출처 : CERC 2014 I번

제출 : 9 / 정답 : 8

평균 시도 : 3.38

검은 벽돌이 b개, 흰 벽돌이 w개 있으면 나눌 수 있는 비율은

$$\frac{b}{\gcd(b,w)} : \frac{w}{\gcd(b,w)}.$$

검은 벽돌이 b개, 흰 벽돌이 w개 있으면 나눌 수 있는 비율은

$$\frac{b}{\gcd(b,w)} : \frac{w}{\gcd(b,w)}.$$

- 해당 비율을 만족하며 벽돌을 분리할 수 있으면 분리해야 한다.

기하학적으로 좌표평면에 놓고 접근해보자.

기하학적으로 좌표평면에 놓고 접근해보자.

- 좌표평면의 원점에서 시작해서 `B` 1개당 x좌표, `W` 1개당 y좌표 1 증가

기하학적으로 좌표평면에 놓고 접근해보자.

- 좌표평면의 원점에서 시작해서 `B` 1개당 x좌표, `W` 1개당 y좌표 1 증가
- 이 때 $wx - by = 0$과의 교점 중 격자점의 개수를 구하면 된다.

기하학적으로 좌표평면에 놓고 접근해보자.

- 좌표평면의 원점에서 시작해서 `B` 1개당 x좌표, `W` 1개당 y좌표 1 증가
- 이 때 $wx - by = 0$과의 교점 중 격자점의 개수를 구하면 된다.

선분 하나하나씩 옮겨가며 따지면 $O(N)$에 해결 가능.

기하학적으로 좌표평면에 놓고 접근해보자.

- 좌표평면의 원점에서 시작해서 `B` 1개당 x좌표, `W` 1개당 y좌표 1 증가
- 이 때 $wx - by = 0$과의 교점 중 격자점의 개수를 구하면 된다.

선분 하나하나씩 옮겨가며 따지면 $O(N)$에 해결 가능.

문제를 굳이 이렇게 바꾸지 않아도 구현은 비슷할 것 같지만 예외처리가 곤혹스러울 수 있다.

Code : http://boj.kr/742141a744d1427cb61e2e782ed0f74c

Problem H

생물학자 (BOJ 3116)

출처 : Croatian Highschool Competitions in Informatics 2009 National Competition #2 Seniors 1번

제출 : 7 / 정답 : 7

평균 시도 : 2.29

서로 다른 두 박테리아의 쌍에 대해

서로 다른 두 박테리아의 쌍에 대해

- 이 두 박테리아가 만나는지

서로 다른 두 박테리아의 쌍에 대해

- 이 두 박테리아가 만나는지
- 만날 경우, 그 자리에 있는 다른 박테리아가 있는지

를 계산해볼 수 있다.

두 박테리아가 만나는지는 어떻게 확인할 수 있을까?

두 박테리아가 만나는지는 어떻게 확인할 수 있을까?

- 상대 위치와 상대속도를 고려하면 확인 가능.

두 박테리아가 만나는지는 어떻게 확인할 수 있을까?

- 상대 위치와 상대속도를 고려하면 확인 가능.

그럼 이 때 다른 박테리아가 있는지는?

두 박테리아가 만나는지는 어떻게 확인할 수 있을까?

- 상대 위치와 상대속도를 고려하면 확인 가능.

그럼 이 때 다른 박테리아가 있는지는?

- 만날 수 있는 박테리아의 초기 위치와 방향은 (각 방향별로) 8가지밖에 없다!

두 박테리아가 만나는지는 어떻게 확인할 수 있을까?

- 상대 위치와 상대속도를 고려하면 확인 가능.

그럼 이 때 다른 박테리아가 있는지는?

- 만날 수 있는 박테리아의 초기 위치와 방향은 (각 방향별로) 8가지밖에 없다!
- 이분 탐색이나 mapping 등을 통해 확인 가능.

두 박테리아가 만나는지는 어떻게 확인할 수 있을까?

- 상대 위치와 상대속도를 고려하면 확인 가능.

그럼 이 때 다른 박테리아가 있는지는?

- 만날 수 있는 박테리아의 초기 위치와 방향은 (각 방향별로) 8가지밖에 없다!
- 이분 탐색이나 mapping 등을 통해 확인 가능.

박테리아 쌍은 $O(N^2)$ 개 있으므로 총 시간복잡도 $O(N^2 \lg N)$ 에 가능하다.

Code : http://boj.kr/c9fdc71d9b244ec3a20b60acbe4841d4

Problem G

로봇 조종하기 (BOJ 2169)

출처 : KOI 2002 고등부 1번

제출 : 9 / 정답 : 9

평균 시도 : 2.11

기본적으로 '칸 (i, j) 까지 거치면서 얻을 수 있는 최대 가치'을 저장하는 동적 계획법으로 접근해야 한다.

기본적으로 '칸 (i, j) 까지 거치면서 얻을 수 있는 최대 가치'을 저장하는 동적 계획법으로 접근해야 한다.

- 칸에 도달하는 방법 : 왼쪽, 오른쪽, (위).

기본적으로 '칸 (i, j) 까지 거치면서 얻을 수 있는 최대 가치'을 저장하는 동적 계획법으로 접근해야 한다.

- 칸에 도달하는 방법 : 왼쪽, 오른쪽, (위).
- 현재 칸 기준으로 왼쪽 또는 바로 위에서 오는 경우만 고려해보자.

기본적으로 '칸 (i, j) 까지 거치면서 얻을 수 있는 최대 가치'을 저장하는 동적 계획법으로 접근해야 한다.

- 칸에 도달하는 방법 : 왼쪽, 오른쪽, (위).
- 현재 칸 기준으로 왼쪽 또는 바로 위에서 오는 경우만 고려해보자.
  - 그럼 $l[i][j] = min(l[i][j-1], dp[i-1][j]) + a[i][j]$.

기본적으로 '칸 (i, j) 까지 거치면서 얻을 수 있는 최대 가치'을 저장하는 동적 계획법으로 접근해야 한다.

- 칸에 도달하는 방법 : 왼쪽, 오른쪽, (위).
- 현재 칸 기준으로 왼쪽 또는 바로 위에서 오는 경우만 고려해보자.
  - 그럼 $l[i][j] = min(l[i][j-1], dp[i-1][j]) + a[i][j]$.
  - $(i, j-1)$ 또는 $(i-1, j)$ 를 방문해야 하기 때문!
  - 오른쪽에서 오는 경우는 고려할 필요가 없다 (한 번 밟은 칸은 다시 밟을 수 없음)

기본적으로 '칸 (i, j) 까지 거치면서 얻을 수 있는 최대 가치'을 저장하는 동적 계획법으로 접근해야 한다.

- 칸에 도달하는 방법 : 왼쪽, 오른쪽, (위).
- 현재 칸 기준으로 왼쪽 또는 바로 위에서 오는 경우만 고려해보자.
  - 그럼 $l[i][j] = min(l[i][j-1], dp[i-1][j]) + a[i][j]$.
  - $(i, j-1)$ 또는 $(i-1, j)$ 를 방문해야 하기 때문!
  - 오른쪽에서 오는 경우는 고려할 필요가 없다 (한 번 밟은 칸은 다시 밟을 수 없음)
- 오른쪽도 비슷하게 정의할 수 있다.

기본적으로 '칸 (i, j) 까지 거치면서 얻을 수 있는 최대 가치'을 저장하는 동적 계획법으로 접근해야 한다.

- 칸에 도달하는 방법 : 왼쪽, 오른쪽, (위).
- 현재 칸 기준으로 왼쪽 또는 바로 위에서 오는 경우만 고려해보자.
  - 그럼 $l[i][j] = min(l[i][j-1], dp[i-1][j]) + a[i][j]$.
  - $(i, j-1)$ 또는 $(i-1, j)$ 를 방문해야 하기 때문!
  - 오른쪽에서 오는 경우는 고려할 필요가 없다 (한 번 밟은 칸은 다시 밟을 수 없음)
- 오른쪽도 비슷하게 정의할 수 있다.
- 각 칸별로 두 값의 최대값을 취하면 된다. 행마다 반복하자.

기본적으로 '칸 (i, j) 까지 거치면서 얻을 수 있는 최대 가치'을 저장하는 동적 계획법으로 접근해야 한다.

- 칸에 도달하는 방법 : 왼쪽, 오른쪽, (위).
- 현재 칸 기준으로 왼쪽 또는 바로 위에서 오는 경우만 고려해보자.
  - 그럼 $l[i][j] = min(l[i][j-1], dp[i-1][j]) + a[i][j]$.
  - $(i, j-1)$ 또는 $(i-1, j)$ 를 방문해야 하기 때문!
  - 오른쪽에서 오는 경우는 고려할 필요가 없다 (한 번 밟은 칸은 다시 밟을 수 없음)
- 오른쪽도 비슷하게 정의할 수 있다.
- 각 칸별로 두 값의 최대값을 취하면 된다. 행마다 반복하자.

총 시간 복잡도는 $O(NM)$.

Code : http://boj.kr/729bb42602d14ee88b7e81568b9dce9b

- 토글링 (toggling) 을 이용한 공간 복잡도 $O(M)$ 코드
- 이렇게 바로 전 열의 상태만 알아도 되는 경우 공간 복잡도 (와 약간의 수행시간) 을 줄일 수 있습니다.

Problem E

알 수도 있는 사람 (BOJ 13358)

출처 : ???

제출 : 5 / 정답 : 5

평균 시도 : 2.4

2-친구와 3-친구 관계를 모두 없애야 한다.

2-친구와 3-친구 관계를 모두 없애야 한다.

- 2-친구의 경우, A랑도 B랑도 친구인 경우이므로 없애야 한다.

2-친구와 3-친구 관계를 모두 없애야 한다.

- 2-친구의 경우, A랑도 B랑도 친구인 경우이므로 없애야 한다.

문제가 되는 것은 3-친구 관계.

2-친구와 3-친구 관계를 모두 없애야 한다.

- 2-친구의 경우, A랑도 B랑도 친구인 경우이므로 없애야 한다.

문제가 되는 것은 3-친구 관계.

- A의 친구들과 B의 친구들끼리 연결된 그래프에서, 최소한의 친구들 (과 각자의 연결관계) 을 제거해서 선분을 모두 없애야 한다.

사실 대단히 전형적인 문제...

- 3-친구 제거 문제는 이분 그래프에서의 vertex cover 문제와 동치이다.

사실 대단히 전형적인 문제...

- 3-친구 제거 문제는 이분 그래프에서의 vertex cover 문제와 동치이다.
- 그리고 이분 그래프에서 minimum vertex cover = maximum matching (König's Theorem)

사실 대단히 전형적인 문제...

- 3-친구 제거 문제는 이분 그래프에서의 vertex cover 문제와 동치이다.
- 그리고 이분 그래프에서 minimum vertex cover = maximum matching (König's Theorem)
- 즉 이분 그래프에서의 매칭을 진행하면 된다.

사실 대단히 전형적인 문제...

- 3-친구 제거 문제는 이분 그래프에서의 vertex cover 문제와 동치이다.
- 그리고 이분 그래프에서 minimum vertex cover = maximum matching (König's Theorem)
- 즉 이분 그래프에서의 매칭을 진행하면 된다.

네트워크 플로우 이론을 이용한 간단한 $O(VE) = O(N^3)$ 이분 매칭 알고리즘이 존재한다.

Code : `http://boj.kr/4a3936b589a8442faf55800884e9759a`

- 네트워크 플로우 관련 블로그 (kks227)
- 이분 매칭 관련 블로그 (kks227)
- Hopcroft-Karp Algorithm (kks227) : 시간복잡도 $O(EV^{0.5})$.

Problem I

장난감 정리 로봇 (BOJ 8875)

출처 : IOI 2013 5번

제출 : 4 / 정답 : 4

평균 시도 : 2.5

모든 장난감 로봇이 각자 k개 이하의 장난감을 정리하면서 모든 장난감을 치울 수 있는지 판별할 수 있을까?

모든 장난감 로봇이 각자 k개 이하의 장난감을 정리하면서 모든 장난감을 치울 수 있는지 판별할 수 있을까?

- Yes! Parametric search를 해보자

모든 장난감 로봇이 각자 k개 이하의 장난감을 정리하면서 모든 장난감을 치울 수 있는지 판별할 수 있을까?

- Yes! Parametric search를 해보자

기본적인 접근은 그리디.

모든 장난감 로봇이 각자 k개 이하의 장난감을 정리하면서 모든 장난감을 치울 수 있는지 판별할 수 있을까?

- Yes! Parametric search를 해보자

기본적인 접근은 그리디.

- 연약한 로봇은 크기에 상관없이, 작은 로봇은 무게에 상관없이 짐을 들 수 있다.

우선 연약한 (무게 제한이 있는) 로봇부터 생각해보자.

우선 연약한 (무게 제한이 있는) 로봇부터 생각해보자.

- $x < y$일 때, 제한 x인 연약한 로봇이 정리할 수 있는 장난감은 제한 y인 연약한 로봇도 정리할 수 있다.

우선 연약한 (무게 제한이 있는) 로봇부터 생각해보자.

- $x < y$일 때, 제한 x인 연약한 로봇이 정리할 수 있는 장난감은 제한 y인 연약한 로봇도 정리할 수 있다.
- 그러므로 (무게) 제한에 대한 오름차순으로 연약한 로봇을 정렬하자.

우선 연약한 (무게 제한이 있는) 로봇부터 생각해보자.

- $x < y$일 때, 제한 x인 연약한 로봇이 정리할 수 있는 장난감은 제한 y인 연약한 로봇도 정리할 수 있다.
- 그러므로 (무게) 제한에 대한 오름차순으로 연약한 로봇을 정렬하자.
- 그럼 각 로봇이 추가되면서 제거할 수 있는 장난감들이 (무게에 따라) 추가된다.

우선 연약한 (무게 제한이 있는) 로봇부터 생각해보자.

- $x < y$일 때, 제한 x인 연약한 로봇이 정리할 수 있는 장난감은 제한 y인 연약한 로봇도 정리할 수 있다.
- 그러므로 (무게) 제한에 대한 오름차순으로 연약한 로봇을 정렬하자.
- 그럼 각 로봇이 추가되면서 제거할 수 있는 장난감들이 (무게에 따라) 추가된다.
- 이 상태에서, 각 로봇당 가장 큰 장난감 k개를 제거하자.

왜 이렇게 할까?

왜 이렇게 할까?

- 나머지를 맡을 작은 로봇들은 무게의 제약을 받지 않기 때문에 큰 장난감부터 정리해야 한다.

왜 이렇게 할까?

- 나머지를 맡을 작은 로봇들은 무게의 제약을 받지 않기 때문에 큰 장난감부터 정리해야 한다.

이젠 작은(크기 제한이 있는) 로봇을 생각해볼 차례.

왜 이렇게 할까?

- 나머지를 맡을 작은 로봇들은 무게의 제약을 받지 않기 때문에 큰 장난감부터 정리해야 한다.

이젠 작은(크기 제한이 있는) 로봇을 생각해볼 차례.

- 제한이 큰 로봇부터 가장 무거운 장난감들을 최대 k개까지 정리한다.

왜 이렇게 할까?

- 나머지를 맡을 작은 로봇들은 무게의 제약을 받지 않기 때문에 큰 장난감부터 정리해야 한다.

이젠 작은(크기 제한이 있는) 로봇을 생각해볼 차례.

- 제한이 큰 로봇부터 가장 무거운 장난감들을 최대 k개까지 정리한다.
- 정리할 수 없으면 k개일 때 정리 불가.

왜 이렇게 할까?

- 나머지를 맡을 작은 로봇들은 무게의 제약을 받지 않기 때문에 큰 장난감부터 정리해야 한다.

이젠 작은(크기 제한이 있는) 로봇을 생각해볼 차례.

- 제한이 큰 로봇부터 가장 무거운 장난감들을 최대 k개까지 정리한다.
- 정리할 수 없으면 k개일 때 정리 불가.

전반적인 구현 : priority queue 등의 자료구조를 이용

왜 이렇게 할까?

- 나머지를 맡을 작은 로봇들은 무게의 제약을 받지 않기 때문에 큰 장난감부터 정리해야 한다.

이젠 작은(크기 제한이 있는) 로봇을 생각해볼 차례.

- 제한이 큰 로봇부터 가장 무거운 장난감들을 최대 k개까지 정리한다.
- 정리할 수 없으면 k개일 때 정리 불가.

전반적인 구현 : priority queue 등의 자료구조를 이용

총 복잡도는 $O(N \lg N \lg(A+B))$.

Code : http://boj.kr/f3599f35f0f64a988dd3413d492f2f36

- C++의 `priority_queue`는 max-heap이라, 큰 값이 먼저 올라온다.
- `priority_queue`에 원하는 비교 연산자를 넣는 방법?
  - C++11의 경우 `auto f = [] () {};` 꼴로 익명 함수를 통해 비교 함수를 만들고, `priority_queue<T, vector<T>, decltype(f)> pq(f);`꼴로 선언하면 된다 (`T`는 타입).

Problem F

A highway and the seven dwarfs (BOJ 7057)

출처 : CEOI 2002 4번

제출 : 2 / 정답 : 1

평균 시도 : 10

직선이 평면에 있는 점들을 분할하는지를 확인하자.

직선이 평면에 있는 점들을 분할하는지를 확인하자.

- convex hull을 만들고, 직선이 이를 통과하는지 판별하는 것과 동치.

직선이 평면에 있는 점들을 분할하는지를 확인하자.

- convex hull을 만들고, 직선이 이를 통과하는지 판별하는 것과 동치.

standard한 문제이기에 다양한 풀이가 있다.

직선이 평면에 있는 점들을 분할하는지를 확인하자.

- convex hull을 만들고, 직선이 이를 통과하는지 판별하는 것과 동치.

standard한 문제이기에 다양한 풀이가 있다.

- 여기서는 보다 간결한 이분탐색을 이용한 풀이를 설명한다.

convex hull을 통과하는 방향을 생각해보자.

convex hull을 통과하는 방향을 생각해보자.

- 이 방향에서 가장 멀리 떨어진 두 점을 생각해볼 수 있다.

convex hull을 통과하는 방향을 생각해보자.

- 이 방향에서 가장 멀리 떨어진 두 점을 생각해볼 수 있다.
- 각 방향별로 이 두 점을 알면...?

convex hull을 통과하는 방향을 생각해보자.

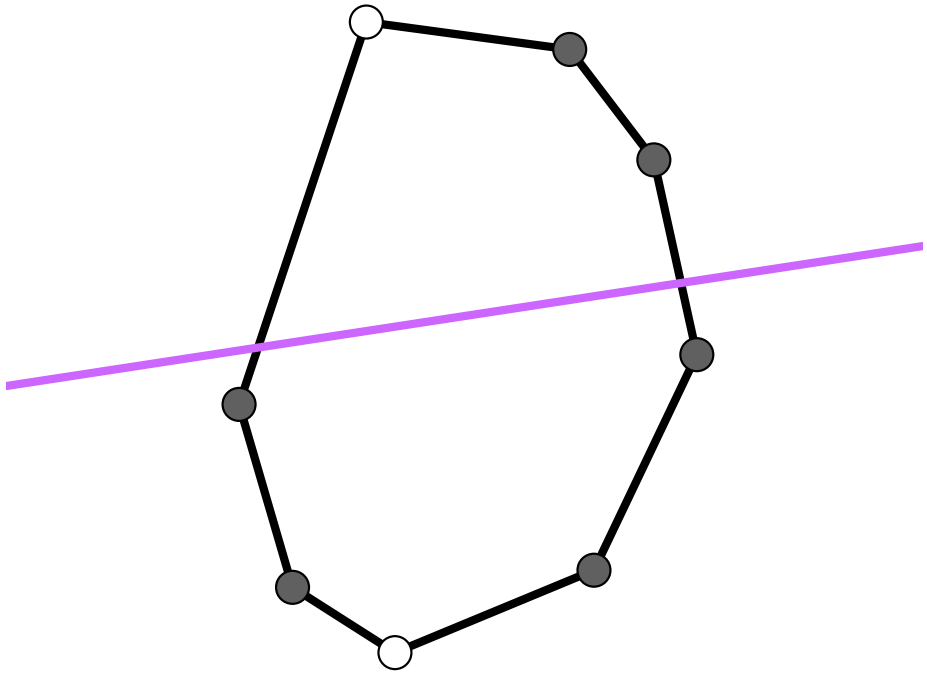
- 이 방향에서 가장 멀리 떨어진 두 점을 생각해볼 수 있다.
- 각 방향별로 이 두 점을 알면...?
  - 두 점을 연결한 선분과 직선이 교차하는지 판별하면 된다!

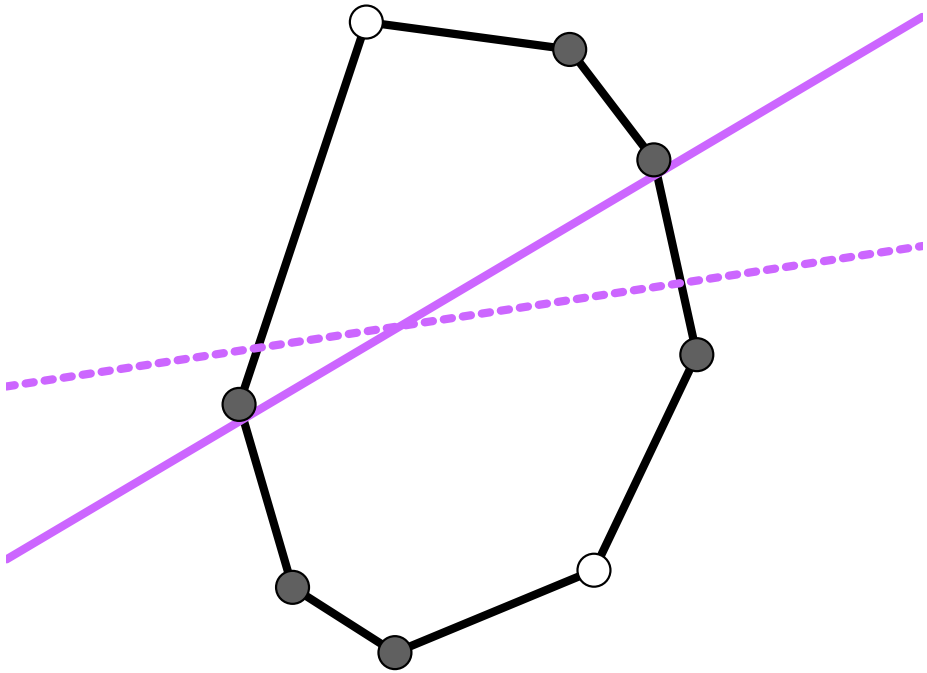
convex hull을 통과하는 방향을 생각해보자.

- 이 방향에서 가장 멀리 떨어진 두 점을 생각해볼 수 있다.
- 각 방향별로 이 두 점을 알면...?
  - 두 점을 연결한 선분과 직선이 교차하는지 판별하면 된다!
- 그럼 각도를 조금씩 기울여보자. 언제 두 점이 바뀔까?

convex hull을 통과하는 방향을 생각해보자.

- 이 방향에서 가장 멀리 떨어진 두 점을 생각해볼 수 있다.
- 각 방향별로 이 두 점을 알면...?
  - 두 점을 연결한 선분과 직선이 교차하는지 판별하면 된다!
- 그럼 각도를 조금씩 기울여보자. 언제 두 점이 바뀔까?
- 직선의 기울기가 두 점이 속한 convex hull의 선분의 기울기를 넘어서면 된다!





convex hull을 위쪽과 아래쪽으로 쪼개자.

convex hull을 위쪽과 아래쪽으로 쪼개자.

- Andrew's Monotone Chain Algorithm으로 쉽게 구축 가능.

convex hull을 위쪽과 아래쪽으로 쪼개자.

- Andrew's Monotone Chain Algorithm으로 쉽게 구축 가능.

양 반껍질은 기울기 순으로 정렬이 되어 있다.

convex hull을 위쪽과 아래쪽으로 쪼개자.

- Andrew's Monotone Chain Algorithm으로 쉽게 구축 가능.

양 반껍질은 기울기 순으로 정렬이 되어 있다.

- 여기서 이분탐색을 진행해서 해당 기울기 이상이 되는 점을 잡자.

convex hull을 위쪽과 아래쪽으로 쪼개자.

- Andrew's Monotone Chain Algorithm으로 쉽게 구축 가능.

양 반껍질은 기울기 순으로 정렬이 되어 있다.

- 여기서 이분탐색을 진행해서 해당 기울기 이상이 되는 점을 잡자.
- 이 점이 주어진 직선의 기울기로부터 '가장 멀리 떨어져있는 점'이 된다!

convex hull을 위쪽과 아래쪽으로 쪼개자.

- Andrew's Monotone Chain Algorithm으로 쉽게 구축 가능.

양 반껍질은 기울기 순으로 정렬이 되어 있다.

- 여기서 이분탐색을 진행해서 해당 기울기 이상이 되는 점을 잡자.
- 이 점이 주어진 직선의 기울기로부터 '가장 멀리 떨어져있는 점'이 된다!
- 선분과 직선이 만나는지 확인만 하면 끝.

Code : `http://boj.kr/514a4fadbe0d4521b22c45bef6bb8cea`

- Andrew's Monotone Chain Algorithm
- CCW도 익숙해집시다

A. 낚시, 그리디

B. 스위핑

C. 트리

D. 구현, 정수론

E. 이분 매칭

F. 기하, 볼록 껍질

G. 동적 계획법

H. 구현

I. 이분탐색, 우선순위 큐, 그리디