Sogang ICPC Team

2021-1 중급 스터디 4회차

문제풀이

이윤제, 임지환
yjyj102742@gmail.com
raararaara@gmail.com

목차

- 사탕 항아리
- Trade
- 수열과 쿼리 0
- Springboards
- Container With Most Water
- 구두 수선공
- Collatz Conjecture

- $N(1 \leq N \leq 100{,}000)$ 개의 항아리
- 각각 $K(1 \leq K \leq 500{,}000), K+1, K+2, \ldots, K+N-1$개씩 사탕이 들어있다.

- 항아리 몇 개를 골라 같은 수만큼 꺼내기 -> 1회
- 뽑기 횟수를 최소화해서 모든 사탕을 꺼내는 방법을 구해보자.

- 관광객이 물건을 사려 한다.
- 예산은 $S(0 \le S \le 10^9)$, 상인은 $n(1 \le n \le 100{,}000)$명

- 각 상인마다 파는 물건의 초기 판매가 $c_i(1 \le c_i \le 10^9)$, 가중치 $p_i(0 \le p_i \le 10^9)$
  - p_i all distinct
- 관광객이 k개의 물건을 갖고 있다면 $c_i + k \cdot p_i$가격에 물건을 팔게 된다.

- 살 수 있는 최대 물건 수를 구해보자.

- 어떤 기준을 먼저 고려해야 할까?

1) 원가가 낮은 물건
2) 가중치(p_i)가 높은 물건

- 어떤 기준을 먼저 고려해야 할까?

1) 원가가 낮은 물건
2) 가중치(p_i)가 높은 물건

원가는 어차피 고정이니, 가중치가 높은 물건을 먼저 고려하는 것이 맞다!

Naïve solution

- 각각의 물건에 대해 살지(0), 안살지(1) -> $O(2^n)$

little optimization to Naïve Solution :

- $d(i,k)$: i번 물건까지 고려하여 k개의 물건을 살 때 드는 최소 비용

$$d(i,k) = \min\big(d(i-1,k-1) + c_i + (k-1)p_i, d(i-1,k)\big)$$

little optimization to Naïve Solution :

- $d(i,k)$: i번 물건까지 고려하여 k개의 물건을 살 때 드는 최소 비용

$$d(i,k) = \min\left(d(i-1,k-1) + c_i + (k-1)p_i, d(i-1,k)\right)$$

time & space complexity: $O(n^2)$

- 물건을 몇 개를 살 수 있을까?

focus on "p_i all distinct",
k개의 물건을 산다면:

$$(c_0 + 0 \cdot p_0) + (c_1 + 1 \cdot p_1) + \cdots + (c_{k-1} + (k-1)p_{k-1})$$
$$= \sum_{i=0}^{k-1} (c_i + i \cdot p_i) \geq \sum_{i=0}^{k-1} i \cdot (k - i - 1)$$

$$= k \sum_{i=0}^{k-1} i - \sum_{i=0}^{k-1} i^2 - \sum_{i=0}^{k-1} i = \frac{k^2(k-1)}{2} - \frac{k(k+1)(2k+1)}{6} - \frac{k(k-1)}{2} = O(k^3)$$

k개의 물건을 구매할 때 ck^3만큼의 비용이 든다면:

$$ck^3 \leq S,$$
$$k \leq \sqrt[3]{c'S}$$

k는 예산의 세제곱근에 bound된다..

little optimization to Naïve Solution 1:

- $d(i,k)$: i번 물건까지 고려하여 k개의 물건을 살 때 드는 최소 비용

$$d(i,k) = \min\big(d(i-1,k-1) + c_i + (k-1)p_i, d(i-1,k)\big)$$

time & space complexity: $O\left(n \cdot \sqrt[3]{S}\right)$

$O\left(n \cdot \sqrt[3]{S}\right)$로 될까요?

- $\sqrt[3]{c'S}$를 분석해보면, 256MB로 빠듯할 수 있습니다.
- 슬라이딩 윈도우를 이용합니다.

$$d(i,k) = \min\left(d(\boxed{i-1},k-1) + c_i + (k-1)p_i, d(\boxed{i-1},k)\right)$$

13545. 수열과 쿼리 0

- 1과 -1로만 이루어져 있는 길이 $N(1 \leq N \leq 100{,}000)$인 수열

- $i \quad j$: $a_i, a_{i+1}, \ldots, a_j$로 이루어진 부분 수열 중에서 합이 0이면서 가장 긴 연속하는 부분 수열의 길이

- focus on 연속 부분수열:

- static array에서 구간합을 구하는 방법 = prefix sum
- 합이 0인 연속 부분수열 = psum[R] == psum[L-1]인 구간 [L,R]

⬇

psum이 같은 index들끼리 관리

- focus on multi-static array query:

- can think Mo's algorithm, Mo's 를 사용하기 위해서는:

1) 구간 확장&축소 과정에서 쿼리 처리
2) 구간 확장&축소에 따른 정보 관리

- 구간 확장&축소 과정에서 쿼리 처리

psum이 같은 index관리를 increasing order로 할 수 있다면?
⇒list의 end와 list의 front의 차이로 계산 가능

- 구간 확장&축소에 따른 정보 관리

index관리를 increasing order로 할 수 있으려면?

- left index append -> push_front
- left index reduce -> pop_front
- right index append -> push_front
- right index reduce -> pop_back

front&back 양쪽에 대한 push&pop 연산을 지원하는 자료구조: deque or list

- 구간 확장&축소 과정에서 쿼리 처리

psum이 같은 index관리를 increasing order로 할 수 있다면?
⇒list의 end와 list의 front의 차이로 계산 가능

이거로 충분한가요?
구간 확장&축소 상에서 반영된 업데이트가 아닌 경우가 최대라면?

- 구간 확장&축소 과정에서 쿼리 처리

psum이 같은 index관리를 increasing order로 할 수 있다면?
⇒list의 end와 list의 front의 차이로 계산 가능

⬇

이거로 충분한가요?
구간 확장&축소 상에서 반영된 업데이트가 아닌 경우가 최대라면?

Mo’s 에 bucket를 얹어보자..

Bucket이 관리하는 정보

- 연속부분수열의 합이 0인 수열의 길이에 대한 count
- count 배열을 bucket으로 관리
- 쿼리에 대한 길이 count를 반영한 후 bucket단위로 쿼리처리=> $O(\sqrt{N})$per query

18318. Springboards

- (0,0)에서 출발하여 (N,N) $(1 \leq N \leq 10^9)$에 도달하고자 한다.

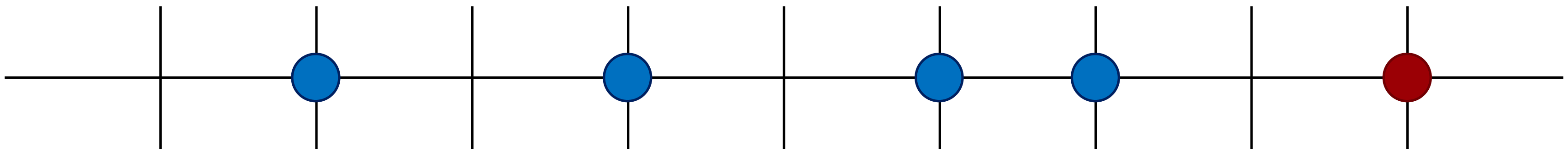
- 이동은 인접한 칸 중 위쪽 혹은 오른쪽으로만 가능하다.
- 격자판에는 $P(1 \leq P \leq 10^5)$개의 스프링보드가 있어, (x_1, y_1)을 밟으면 (x_2, y_2)에 도달할 수 있다. $(x_1 \leq x_2, y_1 \leq y_2)$

- (N,N)에 도달하기 위해 걸어야 하는 최소 거리를 구해보자.

18318. Springboards

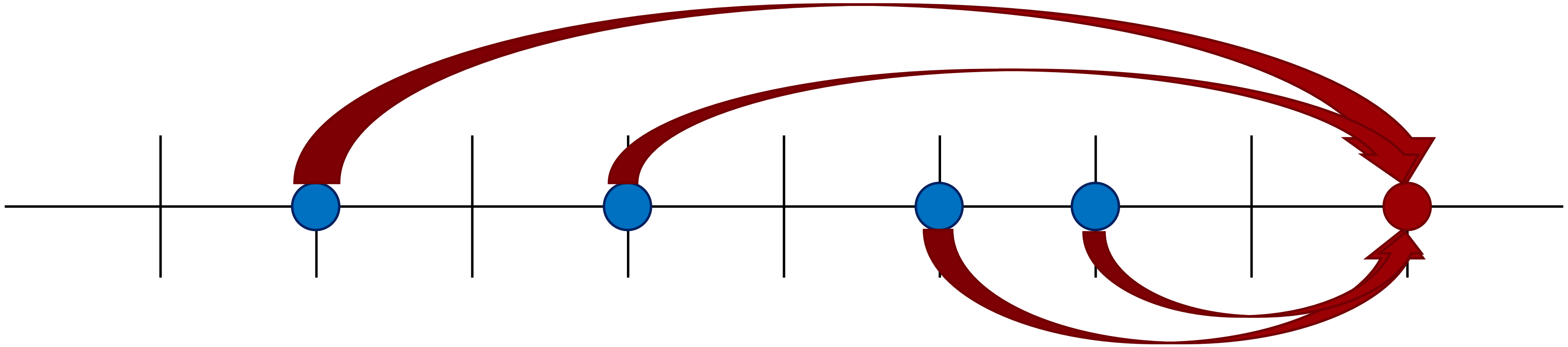
- 공간 상의 좌표의 개수: 최대 2P+2(2P for spring board), 2 for start & end location
- 2차원 평면이 아닌 1차원 상에서의 문제라면? => 그래도 어렵습니다. 하지만..

- 1차원 상에서의 문제



- 1차원 상에서의 문제

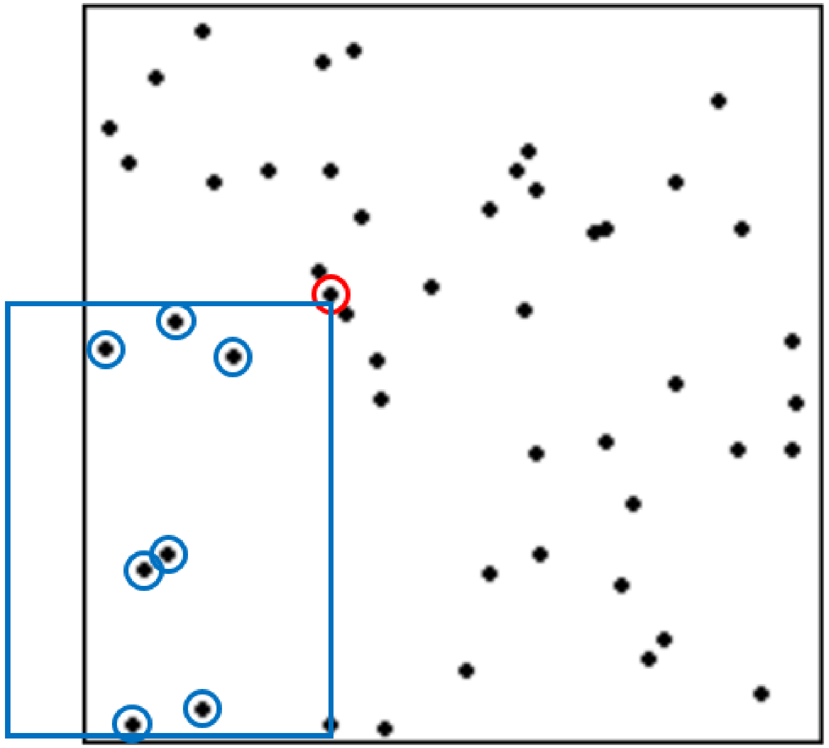
$$dist[i] = \min_{j<i}\left(dist[j] + \left|x_i - x_j\right|\right)$$



- 2차원 상의 문제

$$dist[i] = \min_{j<i}\left(dist[j] + |x_i - x_j| + |y_i - y_j|\right)$$

problem: 평면상의 idx는 어떻게 부여?

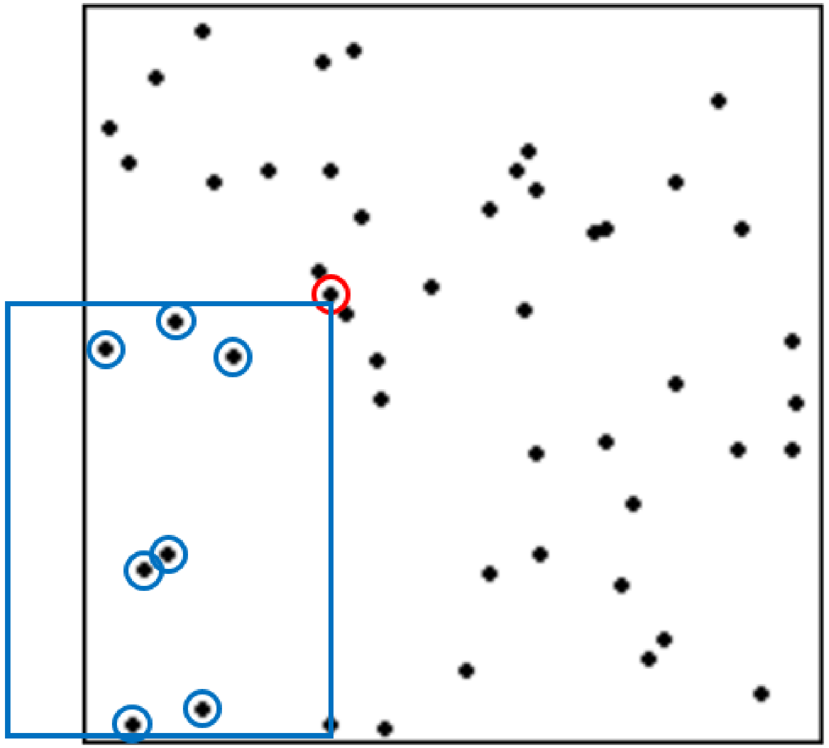


- 2차원 상의 문제

$$dist[i] = \min_{j<i}\left(dist[j] + |x_i - x_j| + |y_i - y_j|\right)$$

problem: 평면상의 idx는 어떻게 부여?

segment tree



- 2차원 상의 문제

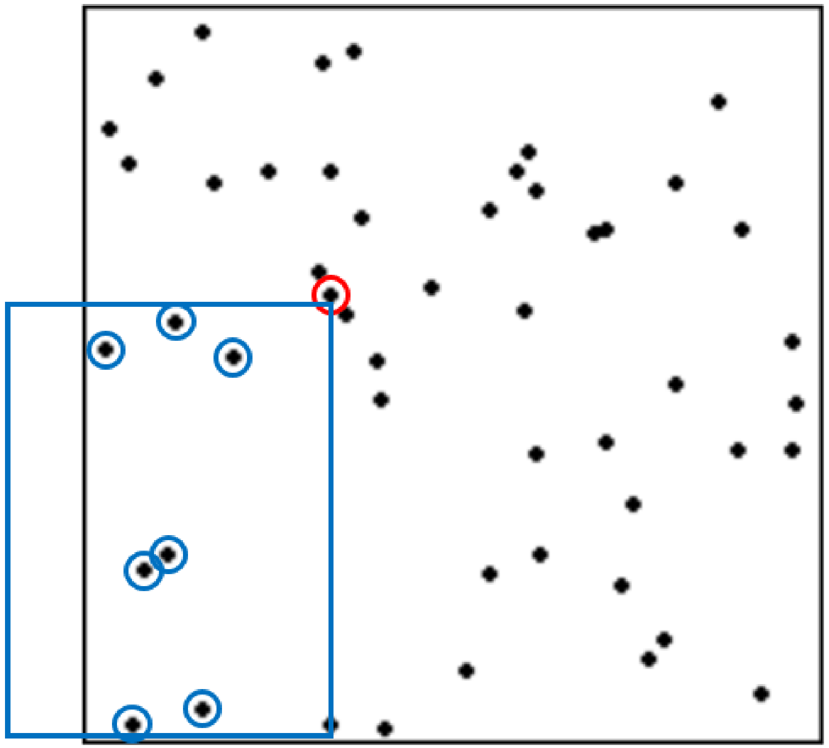
$$dist[i] = \min_{j<i}\left(dist[j] + |x_i - x_j| + |y_i - y_j|\right)$$

problem: 평면상의 idx는 어떻게 부여?

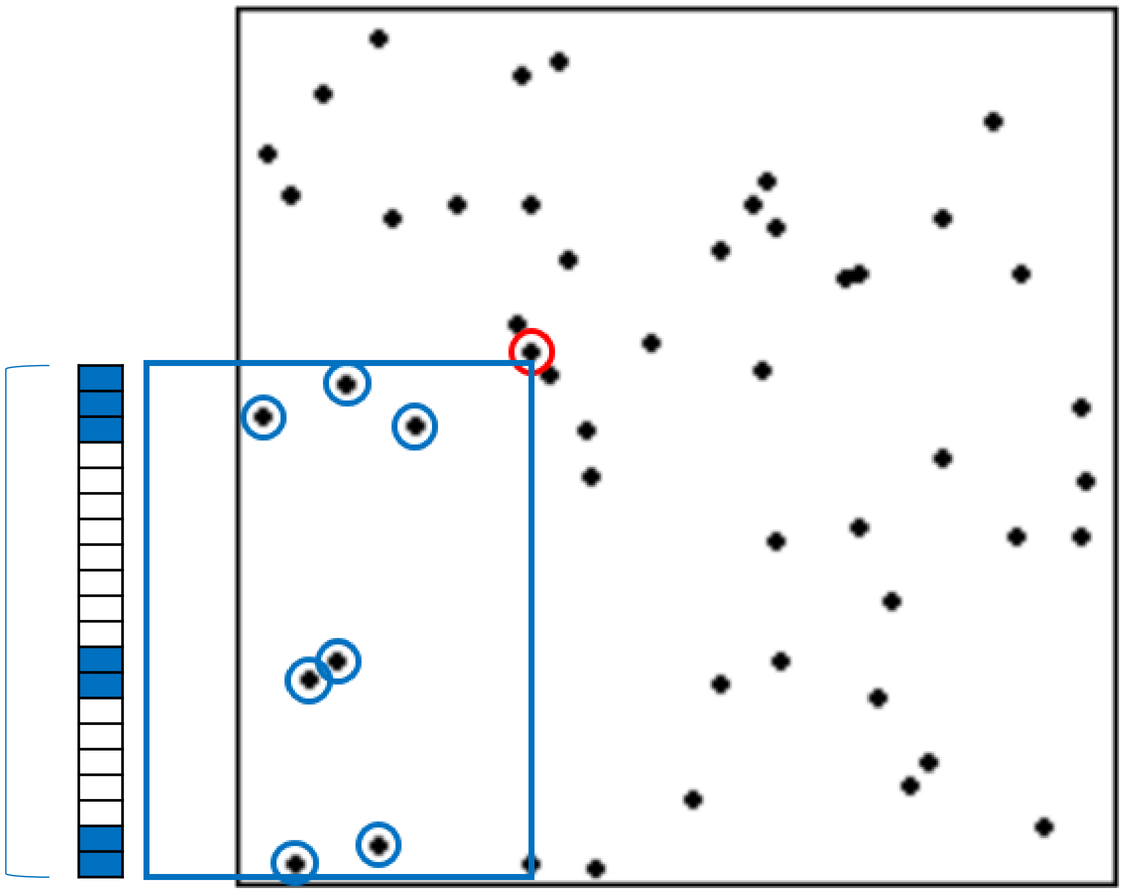
segment tree

x축의 점들을 하나씩 보면서:

각 y좌표에 대한 최소 dist 관리



18318. Springboards



segment tree
for y coordinates

$$dist[i] = \min_{j<i}\left(dist[j] + |x_i - x_j| + |y_i - y_j|\right)$$

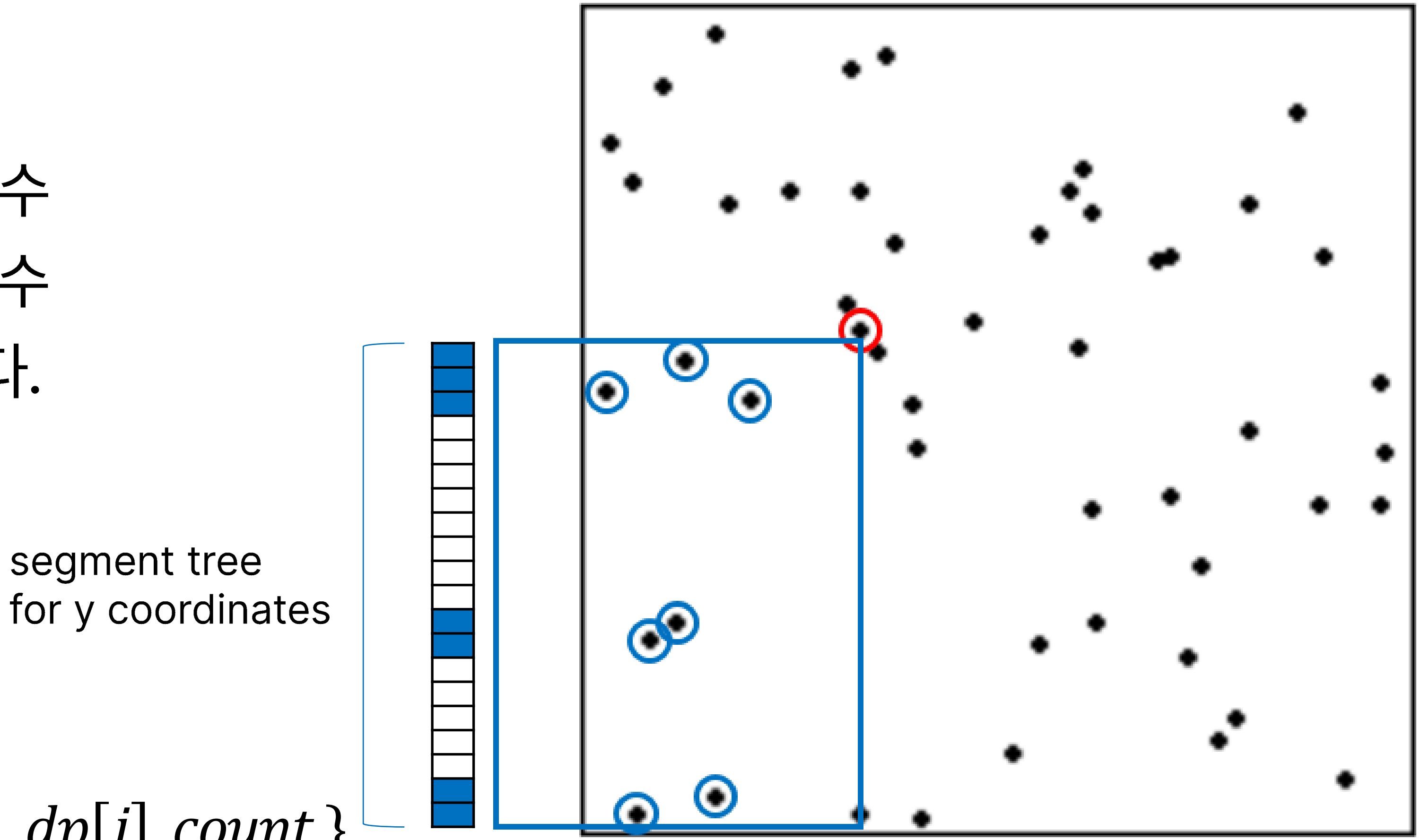
$$\text{->} \min_{j<i}\left(\boldsymbol{dist[j] - (x_j + y_j)} + (x_i + y_i)\right)$$

$\therefore\ target =$ minimize $dist[j] - (x_j + y_j)$ <= handle with segment tree

- Springboards 문제와 전개방식이 차이가 없습니다.
- 똑같이 x값에 대한 sweeping + y값 max distance를 segment tree로 관리합니다.

- 요구하는 값이
1) 최대로 될 수 있는 식물 개수
2) 그렇게 될 수 있는 방법의 수

이므로 이를 pair로 관리합니다.



$$dp[i] = \max\{ dp[j].dist + 1, dp[j].count \}$$

Container With Most Water

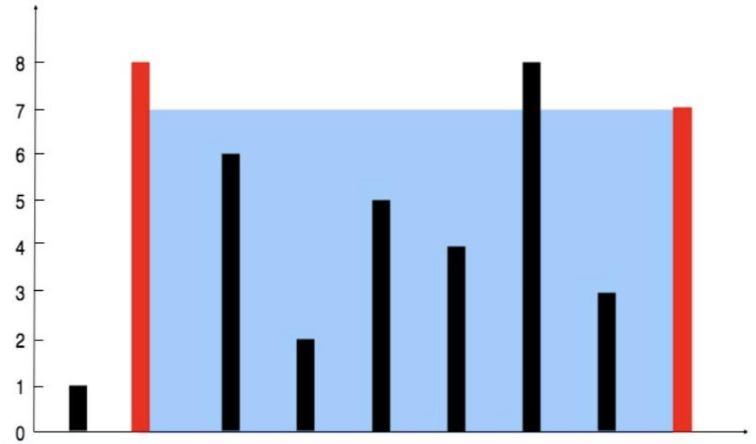
문제 설명

- 일직선 상에 n개의 막대들이 있고, 높이가 주어진다.
- 두 막대를 골라 양 끝으로 삼은 벽에 물을 최대한 많이 담고 싶다.
- 담을 수 있는 물의 양의 최대값 구하기

- 즉, 어떤 두 높이 h1, h2에 대해
  - Min(h1, h2) * (r – l + 1) 들의 최대값

예시

Example 1:



Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Container With Most Water

문제 설명

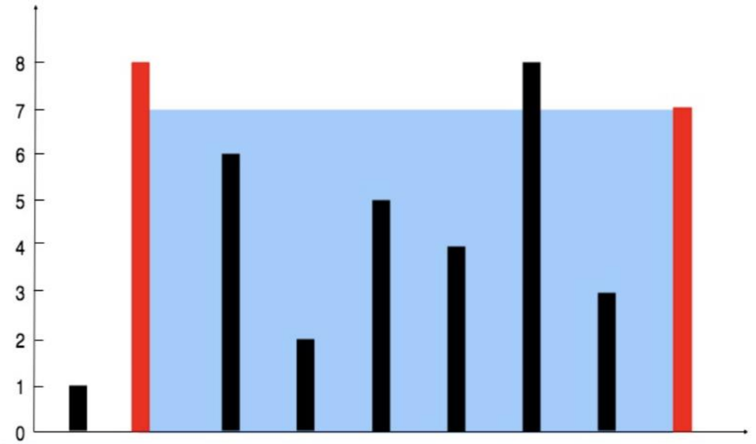
- 일직선 상에 n개의 막대들이 있고, 높이가 주어진다.
- 두 막대를 골라 양 끝으로 삼은 벽에 물을 최대한 많이 담고 싶다.
- 담을 수 있는 물의 양의 최대값 구하기

- 즉, 어떤 두 높이 h1, h2에 대해
  - Min(h1, h2) * (r – l + 1) 들의 최대값

- Naive Approach: O(n^2)

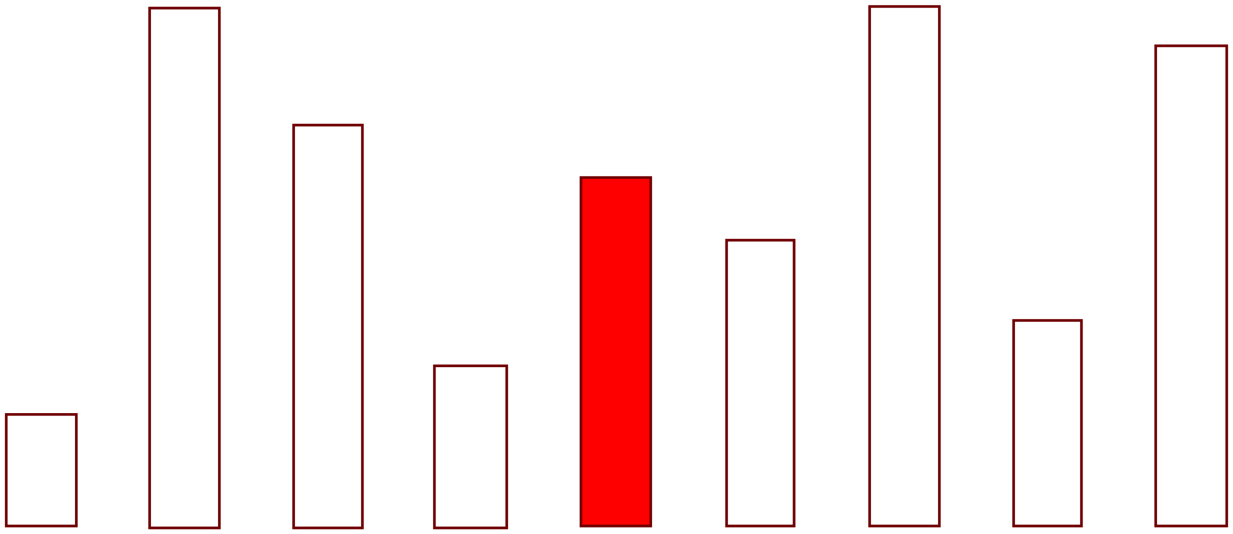
예시

Example 1:

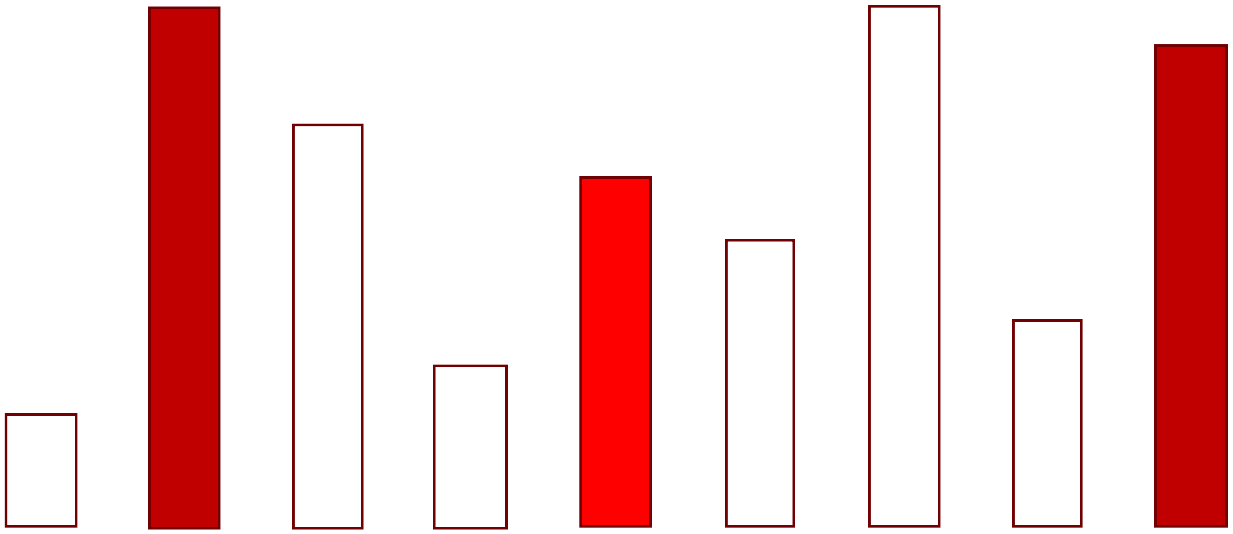



```
Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.
```

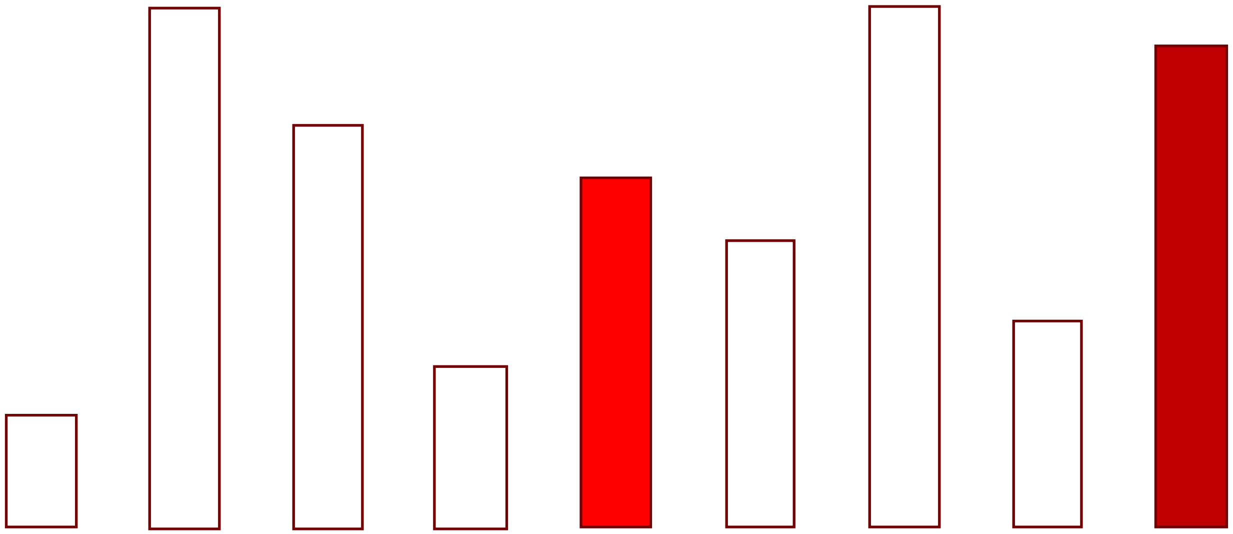
- 각 막대에 대해 최적해가 뭘까?



- 각 막대에 대해 최적해가 뭘까?
- 나보다 길면서 가장 멀리 있는 막대
  - 왼쪽, 오른쪽



- 각 막대에 대해 최적해가 뭘까?
- 나보다 길면서 가장 멀리 있는 막대
  - 가장 멀리있는 오른쪽 막대를 찾아보자.



- 각 막대에 대해 최적해가 뭘까?
- 나보다 길면서 가장 멀리 있는 막대
  - 가장 멀리있는 오른쪽 막대를 찾아보자.

- Priority_queue에 좌표값 우선순위로 담으면
- top이 가장 오른쪽에 있는 막대

- 각 막대에 대해 최적해가 뭘까?
- 나보다 길면서 가장 멀리 있는 막대
  - 가장 멀리있는 오른쪽 막대를 찾아보자.

- Priority_queue에 좌표값 우선순위로 담으면
  - 해당 막대보다 짧은 막대들을 pop
- top이 현재 막대보다 높으면서 가장 오른쪽에 있는 막대

- 각 막대에 대해 최적해가 뭘까?
- 나보다 길면서 가장 멀리 있는 막대
  - 가장 멀리있는 오른쪽 막대를 찾아보자.

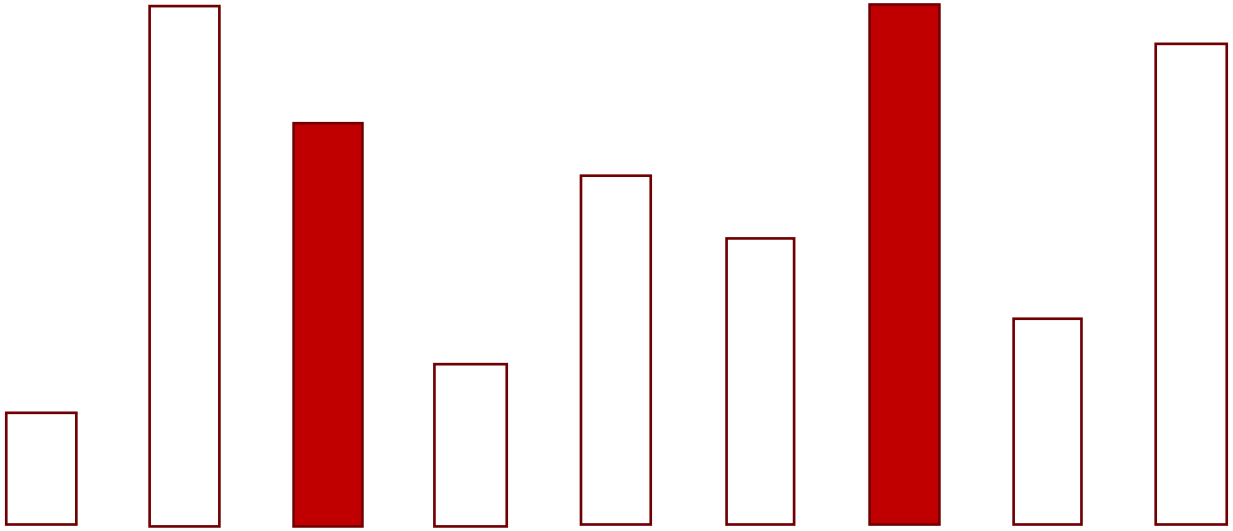
- Priority_queue에 좌표값 우선순위로 담으면
  - 해당 막대보다 짧은 막대들을 pop
- top이 현재 막대보다 높으면서 가장 오른쪽에 있는 막대

- 짧은 막대부터 순서대로진행
  - 한번 pop된 막대가 다시 필요하지 않도록

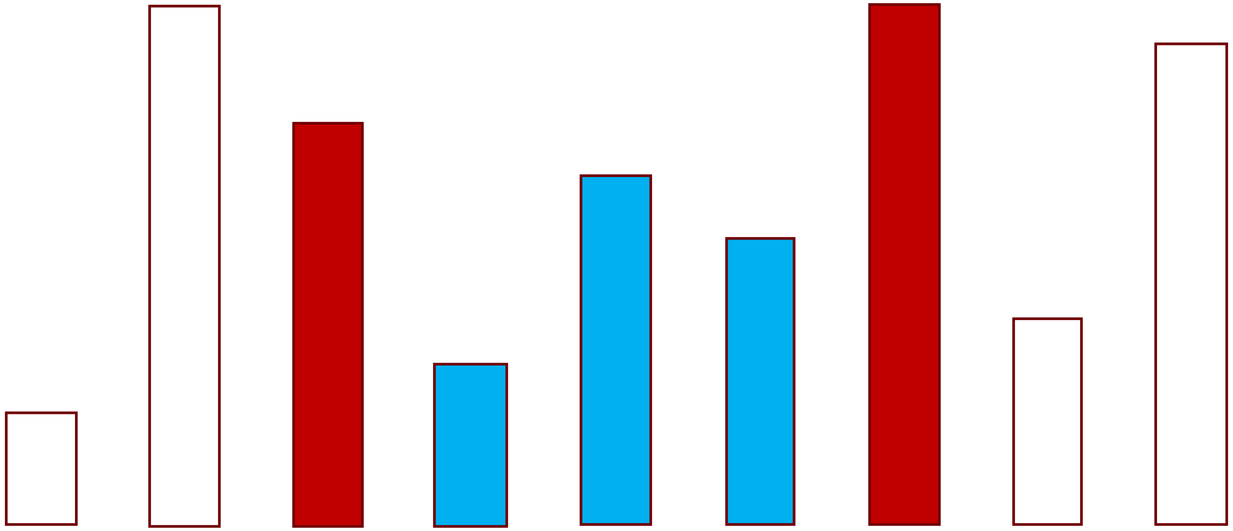
- O(nlogn)

- 더 줄일 수 없을까?

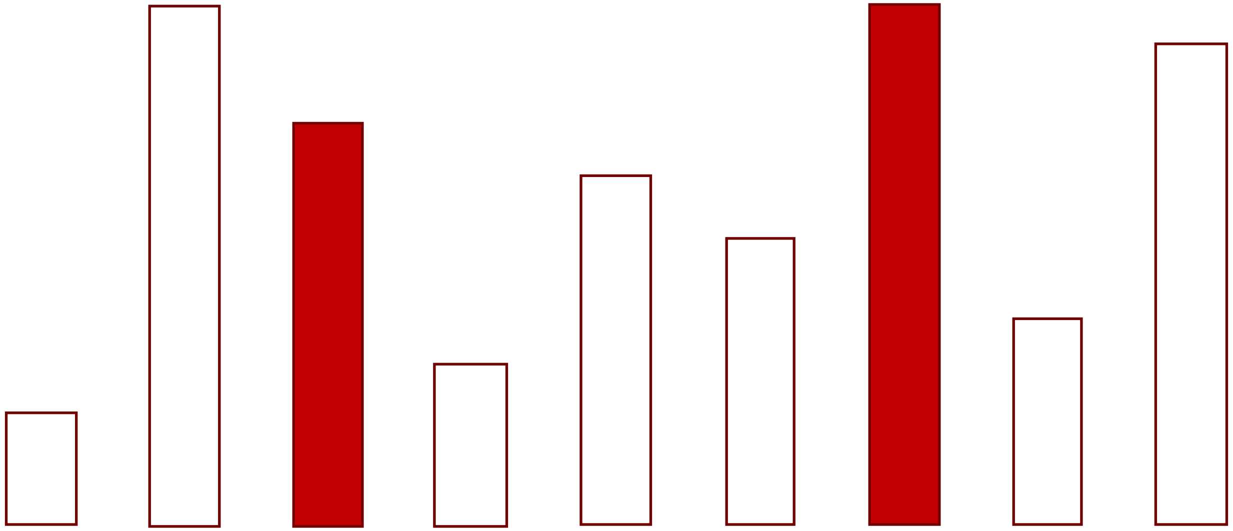
- 다시 Brute Force
- 어떤 시점에 두 막대로 답을 업데이트했다고 가정



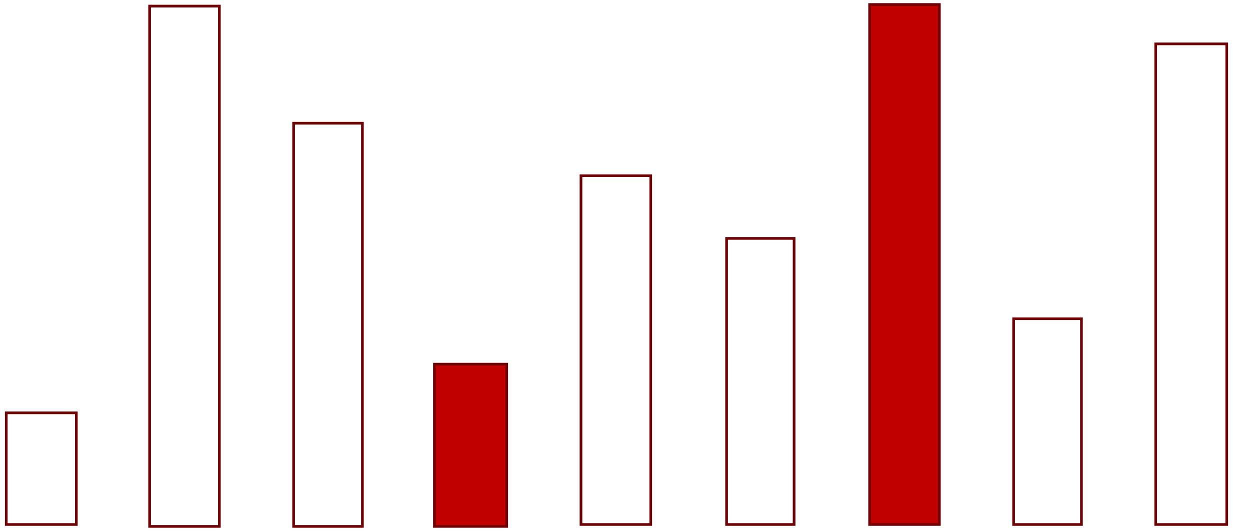
- 다시 Brute Force
- 어떤 시점에 두 막대로 답을 업데이트했다고 가정
- 둘 중 낮은 막대 입장에서, 높은 막대와 사이에 있는 막대들과?



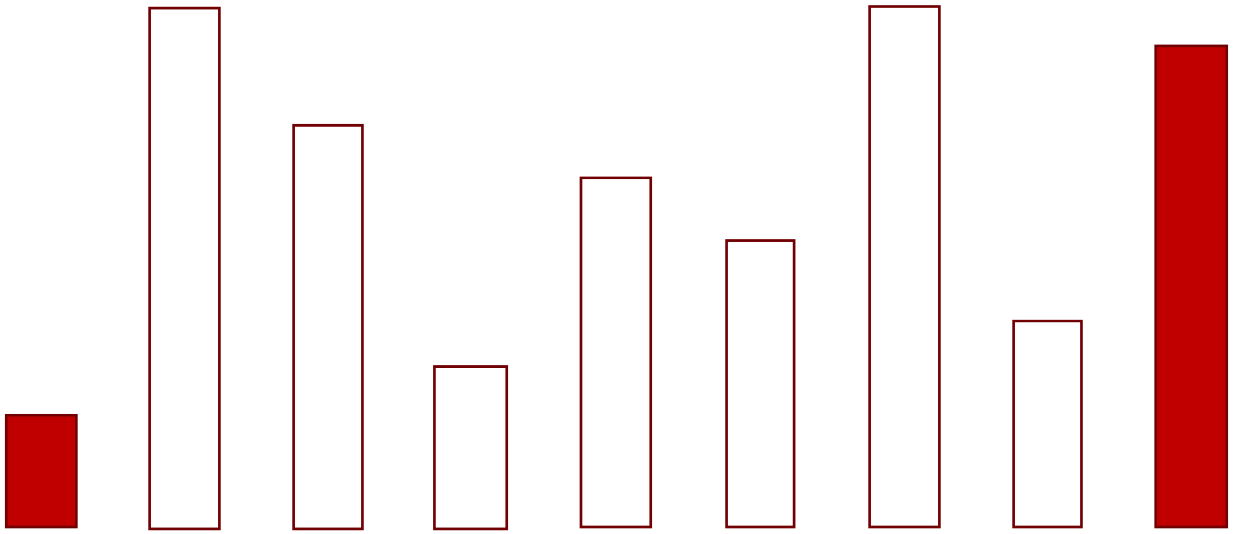
- 다시 Brute Force
- 어떤 시점에 두 막대로 답을 업데이트했다고 가정
- 둘 중 낮은 막대 입장에서, 높은 막대와 사이에 있는 막대들과?
  - 볼 필요 없음



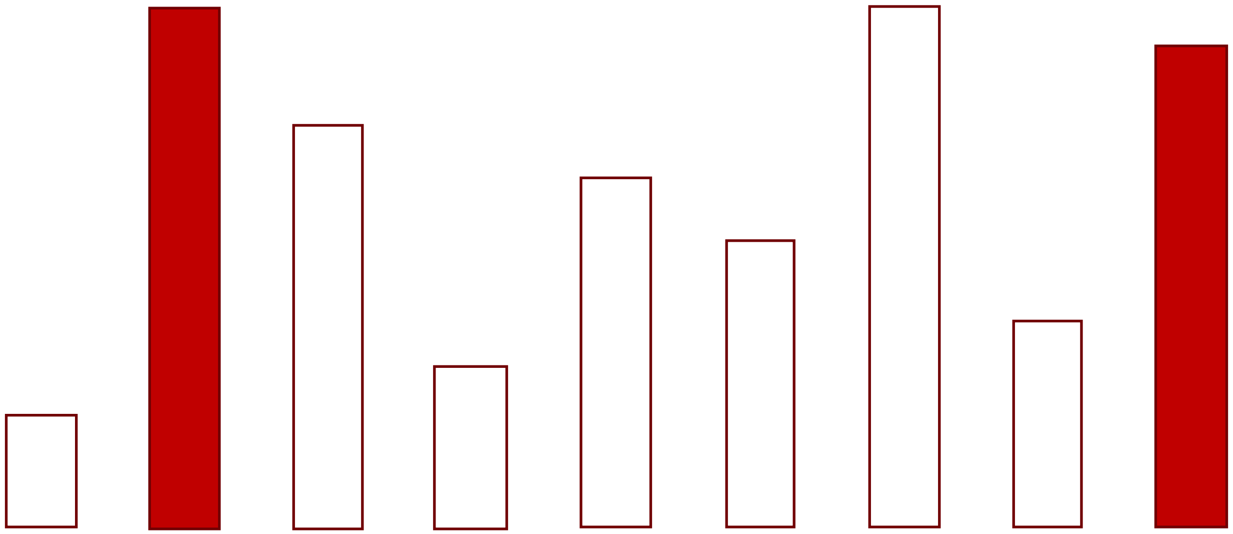
- 다시 Brute Force
- 어떤 시점에 두 막대로 답을 업데이트했다고 가정
- 둘 중 낮은 막대 입장에서, 높은 막대와 사이에 있는 막대들과?
    - 볼 필요 없음
- 좁혀주자



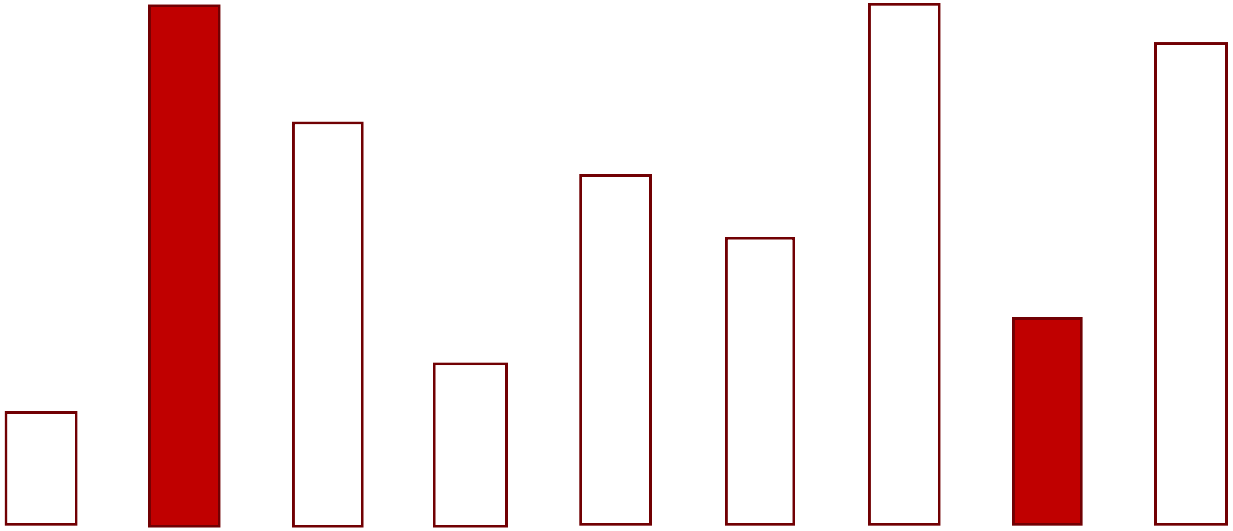
- 최초에 양 끝 막대에서 시작
  - 현재 답을 계산
  - 더 낮은 막대쪽을 좁히고



- 최초에 양 끝 막대에서 시작
  - 현재 답을 계산
  - 더 낮은 막대쪽을 좁히고



- 최초에 양 끝 막대에서 시작
  - 현재 답을 계산
  - 더 낮은 막대쪽을 좁히고
- 반복
- O(n)



Container With Most Water - 정리

- 조건을 만족하는 pair / 구간을 구하는 문제
- O(N^2) --> O(nlogn), O(n) 으로 줄여야 하는 고민
  - Seg Tree, Priority Queue, Two Pointer...

- 투 포인터
  - 투 포인터를 쓸 수 있는지 의심
  - 진짜 투 포인터로 풀리는 지 증명

문제 설명

- 해야 할 일들의 목록이 주어진다.
- 각 일들에 대해
  - T_i: 걸리는 시간
  - S_i: 1일 지연될때마다 발생하는 손해
- 이 주어질 때, 손해를 최소화하기 위한 일의 순서는?

예시

- 4
- 3 4 ← 3일 걸리는 일이고, 하루 지연될때마다 4원 손해
- 1 1000
- 2 2
- 5 5

답: 2 1 3 4

문제 설명

- 해야 할 일들의 목록이 주어진다.
- 각 일들에 대해
  - T_i: 걸리는 시간
  - S_i: 1일 지연될때마다 발생하는 손해
- 이 주어질 때, 손해를 최소화하기 위한 일의 순서는?

사고과정

- 어떤 일들의 중요도가 다를 때 순서 매기기
- 중요도가 숫자로 주어져 있다면…
  - 그냥 정렬하면 되는데

- 완전탐색 – 순열
  - O(n!)

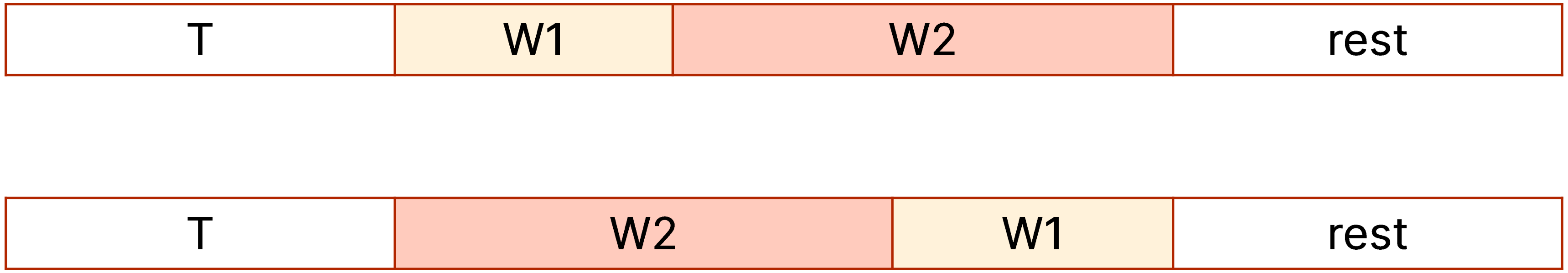
- 완전탐색 – 순열
  - O(n!)

- 각 일의 중요도를 우리가 구할 수 있나?
- 구했다면, 이 중요도는 순서 무관하게 적용될 수 있는 것인가?
  - 두 작업 간 우열관계
  - 앞순서에서 비교할 때 / 뒤쪽 순서에서 비교할 때 다른가?

- 두 일 W1, W2의 우열을 대해 비교해 보자.
  - 다른 조건은 전부 같게 해 주어야 함
- 두 경우 각각의 손해 계산

나머지	W1	W2	나머지
나머지	W2	W1	나머지

- 두 일 W1, W2의 우열을 대해 비교해 보자.
  - 다른 조건은 전부 같게 해 주어야 함
- 두 경우 각각의 손해 계산
- 1번케이스: $T * S1 + (T + T1) * S2 + rest$
- 2번케이스: $T * S2 + (T + T2) * S1 + rest$



- W1을 더 먼저 해야 한다

= 1번케이스가 손해가 적다

- $T * S1 + (T + T1) * S2 + rest \leq T * S2 + (T + T2) \ * S1 + rest$

T	W1	W2	rest

T	W2	W1	rest

- W1을 더 먼저 해야 한다

= 1번케이스가 손해가 적다

- $T * S1 + (T + T1) * S2 + rest \leq T * S2 + (T + T2) \ * S1 + rest$
- $T1 * S2 \leq T2 * S1$
- $T1/S1 \leq T2/S2$

T	W1	W2	rest

T	W2	W1	rest

- 동작의 순서를 정하거나 / 여러 선택지들 중 뭔가 선택해야 할 때
  - 그리디 알고리즘은 좋은 해답이 됨

- 이 방법으로 해결될 수 있다는 증명
  - 다른 조건들을 모두 같게 세팅
  - 최선의 전략이 정말 최선임을 보이자
  - 귀류도 하나의 좋은 방법

- 이거 그냥 될 것 같은데? 를 경계

문제 설명

- 배열이 주어진다
- f(i, j): gcd of {a[i] ~ a[j]} 라고 정의하자.
- 배열에서 만들어지는 서로 다른 f(i, j) 값은 몇 개일까?
- 배열 크기 ≤ 500,000
- 각 원소 값 ≤ 10^18

예시

- 4
- 9 6 2 4

- $f(1,1) = 9,\ f(1,2) = 3,\ f(1,3) = 1,$ $f(1,4) = 1$
- $f(2,2) = 6,\ f(2,3) = 2,\ f(2,4) = 2$
- $f(3,3) = 2,\ f(3,4) = 2$
- $f(4,4) = 4$
- $\{1,2,3,4,6,9\}$

답: 6가지

- Naive Approach: O(n^3)
  - 모든 (i, j) 에 대해 f(i, j) 계산
- 줄여봅시다.

- Naive Approach: O(n^3)
  - 모든 (i, j) 에 대해 f(i, j) 계산
- gcd를 계속 계산하지 말고 누적하면 O(n^2) 가능
  - $f(i,j) = \gcd(f(i,j-1), a[j])$

		i		j	

- gcd를 계속 계산하지 말고 누적하면 O(n^2) 가능
  - $f(i,j) = \mathrm{gcd}(f(i,j-1), a[j])$
- $set[j]$: j에서 끝나는 f값들의 집합
  - $\{f(0,j), f(1,j), \dots, f(j,j)\}$

		i		j	

- gcd를 계속 계산하지 말고 누적하면 O(n^2) 가능
  - $f(i,j) = \gcd(f(i,j-1), a[j])$
- $set[j]$: j에서 끝나는 f값들의 집합
  - $\{f(0,j), f(1,j), \dots, f(j,j)\}$
- $set[i] = \gcd(set[i-1], a[i]) \cup \{a[i]\}$

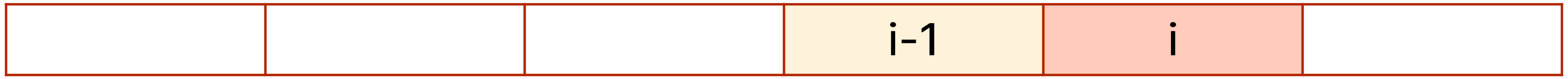
			i-1	i	

Collatz Conjecture

- $set[j]$: j에서 끝나는 f값들의 집합
  - $\{f(0,j), f(1,j), \ldots, f(j,j)\}$
- $set[i] = \gcd(set[i-1], a[i]) \cup \{a[i]\}$

- 이렇게 set으로 관리하면 중복되는 값들에 대해 제거가 가능
  - 총 개수는 n^2개지만 중복되는 값들이 계속 제거
  - 조금은 더 효율적일 것이라는 생각

- 여전히 최악 O(n^2)...



- $set[j]$: j에서 끝나는 f값들의 집합
  - $\{f(0,j), f(1,j), \ldots, f(j,j)\}$
- $set[i]$의 중복제거한 개수는?

			i-1	i	

- $set[j]$: j에서 끝나는 f값들의 집합
  - $\{f(0,j), f(1,j), \dots, f(j,j)\}$
- $set[i]$의 중복제거한 개수는?
  - $f(i,j) = \gcd(f(i-1,j), a[i])$
  - $f(i-1,j) = \gcd(f(i-2,j), a[i-1])$
  - $\dots$
- 같은 set 안의 모든 f는 약수-배수관계
  - $f(i,j) | f(i-1,j) | f(i-2,j) \dots | f(j,j) = a[j]$

- $set[j]$: j에서 끝나는 f값들의 집합
  - $\{f(0,j), f(1,j), \dots, f(j,j)\}$
- 같은 set 안의 모든 f는 약수-배수관계
  - $f(i,j) \mid f(i-1,j) \mid f(i-2,j) \dots \mid f(j,j) = a[j]$

- 약수-배수관계는 서로 값이 같거나 절반 이하

→ $set[j]$: $\{a[j], a[j]/2, a[j]/4, \dots\}$

- $set[j]$: j에서 끝나는 f값들의 집합
  - $\{f(0,j), f(1,j), \ldots, f(j,j)\}$
- 같은 set 안의 모든 f는 약수-배수관계
  - $f(i,j) \mid f(i-1,j) \mid f(i-2,j) \ldots \mid f(j,j) = a[j]$

- 약수-배수관계는 서로 값이 같거나 절반 이하

→ $set[j]$: $\{a[j], a[j]/2, a[j]/4, \ldots\}$
→set[j] 개수: 최대 log(a[j]) 개 !!!
→전체 연산수: log(10^18) 사이즈 집합에 대해 n번 연산

- $set[j]$: j에서 끝나는 f값들의 집합
  - $\{f(0,j), f(1,j), \ldots, f(j,j)\}$
- 같은 set 안의 모든 f는 약수-배수관계
  - $f(i,j) \mid f(i-1,j) \mid f(i-2,j) \ldots \mid f(j,j) = a[j]$

- 약수-배수관계는 서로 값이 같거나 절반 이하

→ $set[j]$: $\{a[j], a[j]/2, a[j]/4, \ldots\}$
→set[j] 개수: 최대 log(a[j]) 개 !!!
→전체 연산수: log(10^18) 사이즈 집합에 대해 n번 연산
→O(n*60)

- 이 문제를 통해 배울 수 있는 것
    - 정수론을 잘하자? *NO!!!*

- 우리가 옳다고 믿고 있는 것들이 정말 옳은가?
    - 언제나 의심하고 시도해 볼 필요가 있다...

마무리

- 개인적으로 좋아하는 세 문제들을 다뤄 봤습니다.
- 각 문제를 해결하는 과정에 대해 곱씹어 보시길 바랍니다.
- 풀이 자체보다는 왜 이렇게 될 수 밖에 없는지
- 확장성 있는 공부
- 좋은 문제들을 제보해 주세요.

- 개인적으로 좋아하는 세 문제들을 다뤄 봤습니다.
- 각 문제를 해결하는 과정에 대해 곱씹어 보시길 바랍니다.
- 풀이 자체보다는 왜 이렇게 될 수 밖에 없는지
- 확장성 있는 공부
- 좋은 문제들을 제보해 주세요.

Q&A