



Trapping rain water

☑ 다시 풀어보기	<input type="checkbox"/>
🔗 링크	https://leetcode.com/problems/trapping-rain-water/
🕒 생성일	@2020년 12월 30일 오후 11:42
# 소요시간(분)	
☰ 유형	Two pointer
☰ 출처	Leet Code

문제

음이 아닌 정수로 막대의 높이가 주어졌을 때, 비가 온 후 물을 얼마나 가두어 놓을 수 있는지 구해라

접근 방법

- 시간 복잡도 $O(n)$
- Two pointer를 이용하여 왼쪽에서 오른쪽으로, 오른쪽에서 왼쪽으로의 최대 높이들을 구한다
- 왼쪽에서 시작해서 구한 높이와 오른쪽에서 시작해서 구한 높이 중 작은값에서 막대 높이를 빼준다 = 물을 가두어둘 수 있는 공간

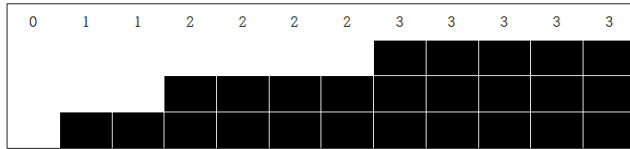
예제

$height = [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]$



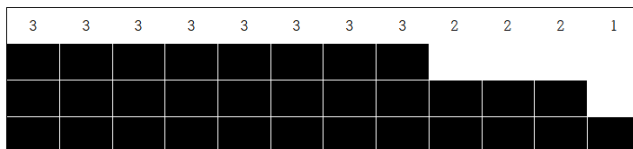
Left → Right

왼쪽에서 오른쪽으로 탐색하며 최대 높이를 저장한다

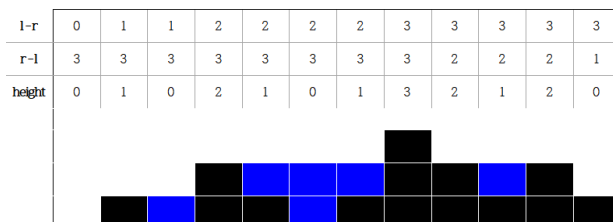


Right → Left

오른쪽에서 왼쪽으로 탐색하며 최대 높이를 저장한다



(각 방향에서 구한 높이의 최소값) - (막대의 높이) = 물을 채울 수 있는 높이



소스 코드

```
class Solution {
    public static int trap(int[] height) {

        if (height.length == 0) return 0;

        int[] leftH = new int[height.length];
        int[] rightH = new int[height.length];

        leftH[0] = height[0];
        rightH[height.length - 1] = height[height.length - 1];
```

```

    int leftIndex = 1, rightIndex = height.length - 2;

    while (leftIndex < height.length) {
        leftH[leftIndex] = Math.max(leftH[leftIndex - 1], height[leftIndex]);
        rightH[rightIndex] = Math.max(rightH[rightIndex + 1], height[rightIndex]);

        leftIndex++;
        rightIndex--;
    }

    int answer = 0;

    for (int i = 0; i < height.length; i++) {
        answer += Math.min(leftH[i], rightH[i]) - height[i];
    }


    return answer;
}
}

```

마무리

20년 하반기 코테에서 비슷한 문제가 나왔었고 백준 [빗물](#) 문제와 같은 유형이라 큰 어려움 없이 풀었다.

그러나 제일 처음 풀었을 때를 떠올려보면,

- Two pointer를 몰라서 2번의 for문을 돌렸었고
- leftH[i] 와 rightH[i]의 최소값으로 구하지않고 최대 정점의 인덱스를 저장해둔 뒤 인덱스에 따라서 따로 계산을 해주었었다 (더다지만 조금씩은 성장하고 있나보다 껄껄 )