查阅更多的题解，请点击

# Problem

## 42. Trapping Rain Water(Hard)

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.


The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped. Thanks Marcos for contributing this image!

**Example:**

```
Input: [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
```

# Solution

该题很有意思，可以当作是在砌墙：输入一个数组，数组元素可以当作当前位置墙的高度，问该种情况，能装多少水（可以这么理解，不过实际上是2维平面，只考虑面积）

类似于“木桶效应”，能装多少水取决于最短的那块木板，即若3个即即以上连续的元素呈现V型，则可以“装水”，数量取决于两个“峰”中较少的那一个。

先分享一个错误的想法：

可以通过一次遍历找到所有可以“装水”的gap，然后去计算。但是这里存在一个问题：[3,1,2,1,6](gap为[3,1,2]和[2,1,6])和[5,2,1,2,1,5](gap为[5,2,1,2]和[2,1,5)这样的输入，我需要去合并多个gap，分情况讨论时比较复杂，并不可取

## Two Pointers

**O(n) time, O(1) space**

为了得到结果，显然知道问题的全部信息而非局部信息，即需要遍历整个输入数组。通常的数据遍历一个指针从前向后或是从后向前遍历，不妨设从前向后遍历，当前已遍历到第i个元素,前i个元素的最大值为max,下标为k(k<i)：那么很直观地想到，对于第i个位置，能装的水是否为**max-height[i-1]**?

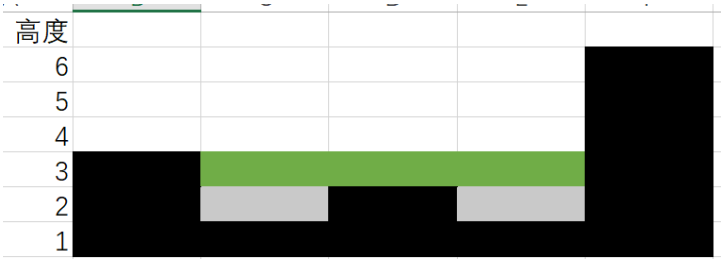
很遗憾不能，由于能装的水取决于木桶的短板，max只是单侧桶壁，若其比另一侧的桶壁短，则计算正确。

那么不妨用两个指针进行遍历，一个从前向后，一个从后向前，并记录maxLeft，maxRight，那么若maxLeft<maxRight，则在左边计算，反之在右边计算。

## Stack

**O(n) time, O(n) space**

考虑之前失败的idea，问题在于某些gap需要merge，比如：输入为[3,1,2,1,6]，两个gap为[3,1,2]和[2,1,6]，需要merge，这时可以有一个smart的merge方式：
如下图所示，黑色部分为高度，灰色和绿色部分之和为所求结果，我们可以先求灰色部分，再求绿色部分，该过程可以用栈实现




GitHub传送门-solution one
GitHub传送门-solution two

## Solution One

```
class Solution
{
  public:
    int trap(vector<int> &height)
    {
        int n = height.size();
        if (n < 3)
            return 0;
        int ret = 0;
        int left = 0, right = n - 1;
        int maxLeft = 0, maxRight = 0;
        while(left<right){
            if(height[left]<height[right]){
                if(height[left]>maxLeft)
                    maxLeft = height[left];
                else
                    ret += maxLeft - height[left];
                left++;
            }else{
                if(height[right]>maxRight)
                    maxRight = height[right];
                else
                    ret += maxRight - height[right];
                right--;
            }
        }
        return ret;
    }
};
```

## Solution Two

```
class Solution
{
  public:
    int trap(vector<int> &height)
    {
        int n = height.size();
        if (n < 3)
            return 0;
        int ret = 0;
        stack<int> s;
        for (int i = 0; i < n; ++i)
        {
            while (!s.empty() && height[i] > height[s.top()])
            {
                int low = height[s.top()];
                s.pop();
                if (!s.empty())
                {
                    int width = i - s.top() - 1;
                    int h = min(height[i], height[s.top()]) - low;
                    ret += width * h;
                }
            }
            s.push(i);
        }
        return ret;
    }
};
```