1. Trapping Rain Water

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

解法1 暴力枚举。对于每一个柱子，只有被两个高柱子夹住，它的上方才能存储雨水

class Solution {  
public:  
 int trap(vector<int>& height) {  
 int ans = 0;  
 for(int i = 0; i < height.size(); ++i){  
 int max\_left = 0, max\_right = 0;  
 for(int j = 0; j < i; ++j){  
 max\_left = max(max\_left, height[j]);  
 }  
 for(int j = i + 1; j < height.size(); ++j){  
 max\_right = max(max\_right, height[j]);  
 }  
 ans += max(0, min(max\_left, max\_right) - height[i]);  
 }  
 return ans;  
 }  
};

解法2 动态规划。暴力枚举中每次都在重新找最大值，这个过程可以一次遍历并存储下来

class Solution {  
public:  
 int trap(vector<int>& height) {  
 int ans = 0, n = height.size();  
 if(n == 0)return 0;  
 int max\_left[n], max\_right[n];  
 max\_left[0] = 0, max\_right[n - 1] = 0;  
 for(int i = 1; i < n; ++i){  
 max\_left[i] = max(max\_left[i - 1], height[i - 1]);  
 }  
 for(int i = n - 2; i >= 0; --i){  
 max\_right[i] = max(max\_right[i + 1], height[i + 1]);  
 }  
 for(int i = 0; i < height.size(); ++i){  
 ans += max(0, min(max\_left[i], max\_right[i]) - height[i]);  
 }  
 return ans;  
 }  
};

解法3 使用栈

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

解法4 two pointers。将dp方法中寻找最大值的过程和计算水容量的过程合二为一。对于左右两边，只要，一定不能存水，因此此时只要更新最值即可；当时，处有可能存水，按照此种方式移动可知，，存水量为，另一种情况同理

class Solution {  
public:  
 int trap(vector<int>& height) {  
 int ans = 0;  
 int i = 0, j = height.size() - 1;  
 int max\_left = 0, max\_right = 0;  
 while(i < j){  
 if(height[i] < height[j]){  
 if(height[i] < max\_left){  
 ans += max\_left - height[i++];  
 }else{  
 max\_left = height[i++];  
 }  
 }else{  
 if(height[j] < max\_right){  
 ans += max\_right - height[j--];  
 }else{  
 max\_right = height[j--];  
 }  
 }  
 }  
 return ans;  
 }  
};