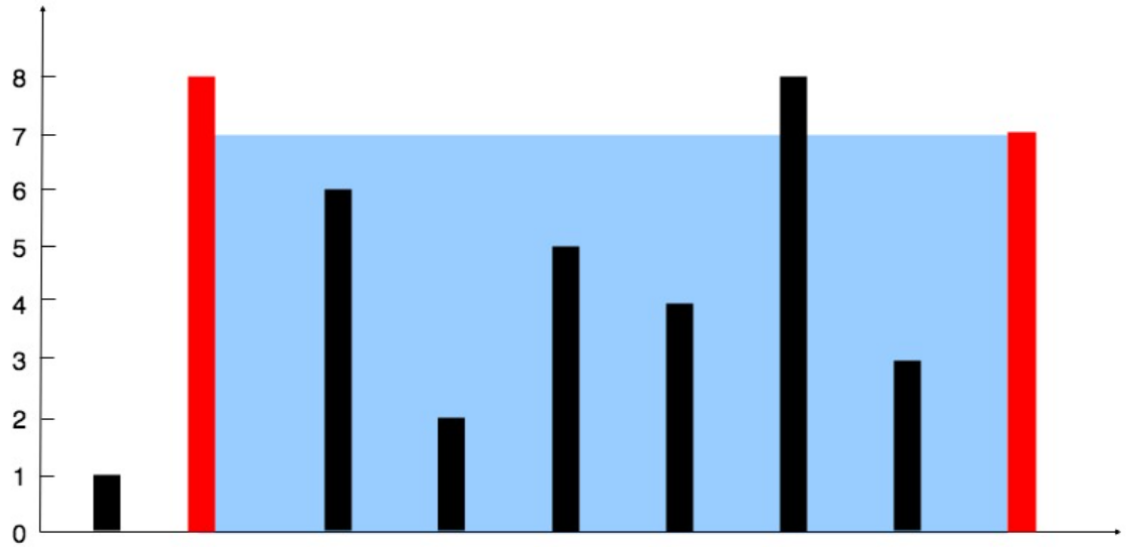
黎锦灏　518021910771　0420作业

1、给定正整数 a1, a2, ..., an，代表 n 条线段（由点(i,ai)和(i,0)构成，i=1,2,...,n），从中找出两条线段，使之与 x 轴构成的容器能够包含尽可能多的水 。



例如：输入 1,8,6,2,5,4,8,3,7，则输出为 49，见上图，蓝色表示水

解：考虑以下做法：初始双指针 l, r 分别指向 1 和 n ，以两端的两条线段更新一次容器容量

- 若 $a_1 < a_n$，则 $l++$;
- 若 $a_1 \geq a_n$，则 $r++$;

  算法结束后得到的最大容量即为目标所求。

下面证明算法的正确性：

考虑第一步，假设当前左指针和右指针指向的数分别为 l 和 r（不妨假设 $a_l < a_r$ ）设此时两个指针之间的距离为 $d = r - l$。那么，它们组成的容器的容量为：$a_l * d$。

不妨考虑若保持较小的值不动，移动指向此时较大的值的指针 $r' < r$，则有

- 若$a_{r'} >= a_r$，则 $min(l, r') <= min(l, r)$
- 若$a_{r'} < a_r$，则 $min(l, r') < min(l, r)$

考虑到此时宽度比之前的 d 更小，而高度也比先前更小，所以移动较大的值的指针，容量只会更小，一定不能得到更好的结果。

因此我们可以得到结论：双指针中，每次移动指向较小的值，并以当前两指针指向的两条线段为容器边界计算容量， 最后能得到最优值。

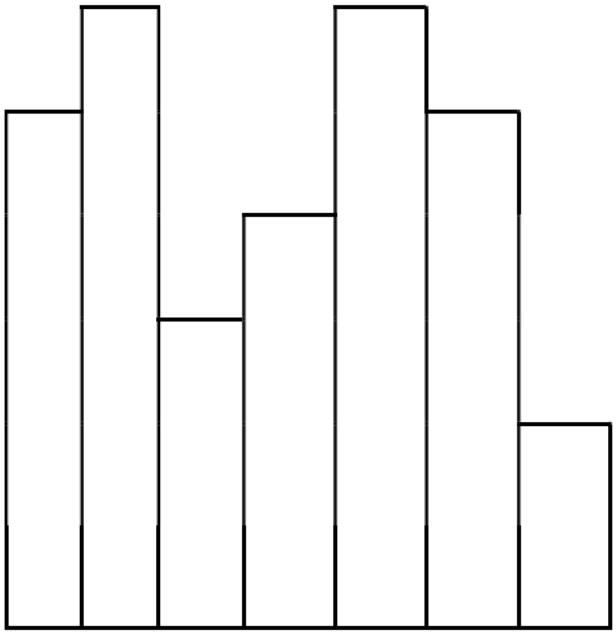
故时间复杂度为 $O(n)$。

C++代码如下：

```
Input:height
Output:container with most water
int maxArea(vector<int>& height) {
        int n = height.size();
```

```
        int l = 0, r = n - 1, ans = 0, now;
        while(l < r) {
            if(height[l] < height[r]) {
                now = height[l] * (r - l);
                if(now > ans) ans = now;
                l++;
            }
            else {
                now = height[r] * (r - l);
                if(now > ans) ans = now;
                r--;
            }
        }
        return ans;
}
```

2、海报墙由 n 块宽度相同高度不同的木板组成，那么在此海报墙上能够张贴的最大海报面积是多少？设木板宽度为 1，高度为 h1,h2,...,hn，海报必须整体都粘贴在墙上，并且不能斜贴。



解：对于一张海报的张贴，实际上是取左右端点 l, r，面积为$(r-l)*min_{l\le i\le r}(a[i])$，那么对于固定高度的右端点而言，它能取到的最左端点即恰好比它小的第一个左边 l，则面积为 $(r-l)*a[r]$。

考虑单调栈算法，维护一个高度单调上升的木板序列栈 S（栈中元素记录木板的位置、木板的高度、最左可以拓展到的位置），从左往右扫描木板，设当前扫描到 i：

- 若 $a_i > S.top().h$，则证明此时左边不会限制当前木板的高度，所以只需将当前木板入栈即可；
- 若$a_i \le S.top().h$，则需要退栈直到栈顶刚好小于当前木板高度，则这段区间即为以 i 为右端点的海报最大面积，且记录这一位置+1为当前木板能拓展的最左位置。

从左到右扫描一遍，每次入栈时更新当前最优值即可，最后得到的即为最大面积。

由于只需要一次扫描，故时间复杂度为 $O(n)$。

伪代码见下：

```
Algorithm: Board
Input: A {a set of board's height}
Output: the maximum area
begin
    ans := 0;
    for i:=0 to n do
        board[i].i:=i;
        board[i].pre:=0;
        board[i].h:=A[i];

    for i:=0 to n do
        while !s.empty() and board[i].h < s.top().h do

        tmp:=s.top();
        s.pop();
        board[i].pre := tmp.pre+1;
        ans := max(ans , tmp.v*(i - tmp.i + tmp.pre));
        s.push(board[i]);

    while !s.empty() do
        tmp:=s.pop();
        ans := max(ans,tmp.v*(i - tmp.i + tmp.pre));

    output ans
end
```

3、平面有两组点，如何证明是否存在直线可以将这两组点分开 。

证：对于两组点分别求凸包，得到包裹住两组点的两个凸多边形。

对于两个凸边形可以求交集：

- 若无交集，则证明两组点可以用一条直线分开；
- 若有交集，则证明无法分开。

黎锦灏　518021910771　0422作业

1、已知 n 个矩形，这些矩形的边都平行于坐标轴，1）求出所有这些矩形的交集；2）求出这些矩形能够覆盖的面积

解：1）要求所有的矩形的交集可以拆成若干子问题，求两个矩形的交集复杂度为$O(1)$，而且求得的交集仍然是一个矩形，于是可以拿矩形交集继续与下一个矩形求交集即可，与 n 个矩形依次求交集所得的结果即为所有矩形的交集，时间复杂度为 $O(n)$。

2）考虑扫描线算法求矩形覆盖的面积。先按 x 坐标排序，从左向右扫描，扫描线的事件列表为每个矩形的竖直边，每遇到一个矩形的左边则加入扫描线维护的集合，遇到右边则从集合中删除。

扫描线维护的状态为当前扫描线穿过的矩形集合，在一个新的扫描线事件发生，即扫描线状态发生变化时，计算当前事件点到上一事件点之间的覆盖面积（若干矩形），并且在新事件加入或删除一个矩形之后，维护扫描线上的有效矩形集合。

伪代码如下：

```
Algorithm:Rectangle Area
Input: R the set of rectangles {R1, R2, … Rn}
Output: The maximum coverage area
begin
    sort all vertices in non-decreasing map order of their x-coordinates
    Initialize the empty stack S {every time push new elements into the stack,
it will keep the orderaccording to the y-coordinates}

    for i:=0 to n do
        area :=  area + S(Ri)

    for i:=0 to 4n do
        if V[i] is the left-end point
            S.push(V[i])
        if V[i] is the right-end point
            if V[i] is point in the upper bound
                find the symmetrical left-end point u
                S.pop(u)

                find the point v whose y-coordinates is bigger than V[i] and
push them in a queue{from the order of non-decreasing order of
their x-coordinates}

                flag:=false

                while !Q.empty()do
                    v := Q.dequeue()
                    if flag then area:=area - the area formed by {v, V[i]}
                    else area:=area + the area formed by {v, V[i]}
            else
                find the point v whose y-coordinates is smaller than V[i] and
push them in a queue{from the order of non-decreasing order of
their x-coordinates}

                flag:=false
                while !Q.empty()do
                    v := Q.dequeue()
                    if flag then area:=area - the area formed by {v, V[i]}
                    else area:=area + the area formed by {v, V[i]}
    output area
end
```