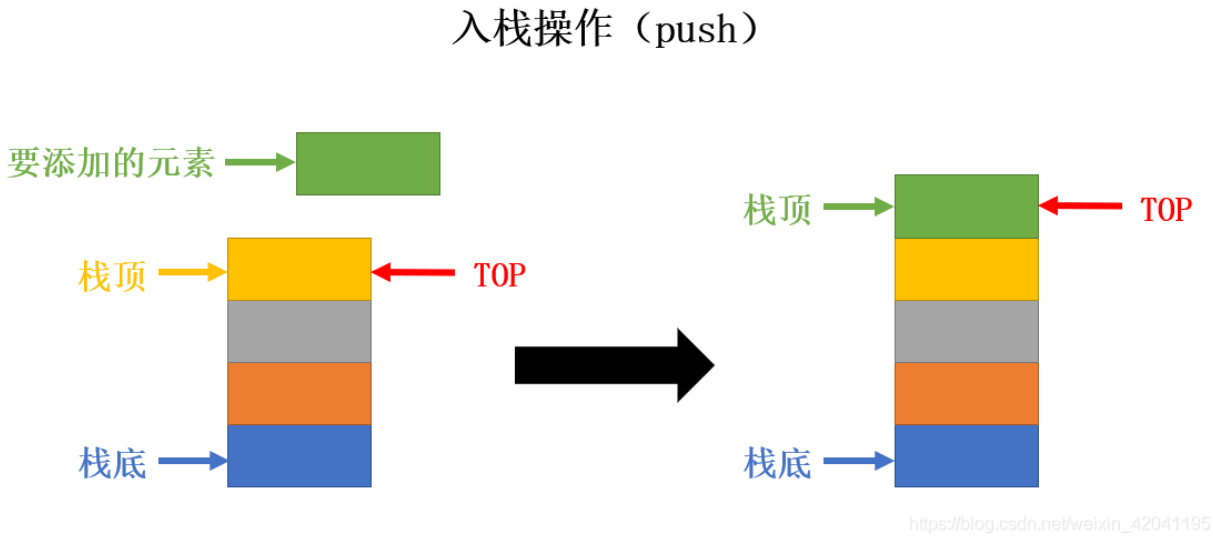
单调栈

定义

栈是一种数据结构，特点是先进后出，只有顶部可以进出。



那么单调栈就是栈里面的元素按照他们的位置关系，满足一定单调性的特殊的栈

例题引入

- 给定一个 n 个数的数组 a，定义 $f(i)$ 是 $a[i]$ 左边第一个小于 $a[i]$ 的元素的下标，若不存在，则 $f(i)=0$，试求出所有数的 $f(i)$
- $n \leq 1e6$

3 4 2 5 9 7

0 1 0 3

1. 最朴素的想法显然是两层 for 循环，就可以求出，时间复杂度 $O(n^2)$
2. 看到要求的东西里有说第一个这种话，思考是否可以二分。假如现在要求 $f(i)$，然后有一段区间是 $[L,i]$，已知里面的最小值小于 $a[i]$，那么 $[L,i]$ 区间内肯定存在答案，假如没有，那么显然不存在答案，这样就显然具有二分的性质了，所以我们需要维护区间最小值。
    - 关于这个的写法有很多
    - 先二分区间然后再在线段树上查询最小值，这样对于每一个数就是 $log(n)^2$ ，无法通过
    - 或者直接在线段树上二分，对于 $[1,i]$ 区间来说，假如 $[mid+1,i]$ 存在答案了，那么肯定往右儿子查，反之才差左儿子。这样对于每个数就是 log
    - 不过第一种先二分区间，再查询最小值的，可以用 st 表来把查询的复杂度优化掉，那么也是 log
3. 思考一下如何用单调栈解决这个问题

- 假如有两个数 $a[i]$ 和 $a[j]$，$a[i] < a[j]$ 并且 $i > j$，意思就是 $a[i]$ 在 $a[j]$ 的右边并且还比 $a[j]$ 小。那么 $a[j]$ 对于后面的数来说，必然不会作为答案，相当于 $a[j]$ 这个值没有用了，那么我们就可以想象栈。对于每个数进来，因为维护的相当于是前面的数的一个集合，所以我们可以把前面的比他大的数都弹出去。那么停止的时候， 就是遇到了第一个小于 $a[i]$ 的数，其实用单调栈的思想可以看作只是暴力的优化或者说减枝，因为栈里维护的也是前面的集合，或者说是可行的答案的集合，因为每加进来一个数，都会弹出前面不可能作为答案的数。显然每个数最多只能被加进来一次，弹出去一次，那么时间复杂度显然是 $O(n)$ 的。那么就比前面的方法优秀很多。

```cpp
void solve() {
    int n;
    cin >> n;

    stack<int> st;//栈里面存放的是下标
    vector<int> a(n + 1), f(n + 1);
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
        while (!st.empty() && a[st.top()] >= a[i]) {
            // 不同题目的符号不同，注意分辨
            st.pop();
        }
        f[i] = (st.size() == 0 ? 0 : st.top());
```

```
14          //f[i] = (st.size() == 0 ? n + 1 :
    st.top());
15          st.push(i);// 注意栈里面放进去的是下标
16      }
17
18      for (int i = 1; i <= n; ++i) {
19          cout << f[i] << " \n"[i == n];
20      }
21  }
```

一般我习惯求左边的，for i 从 1 到 n ，求右边，for i 从 n 到 1 。其他部分没有任何的区别

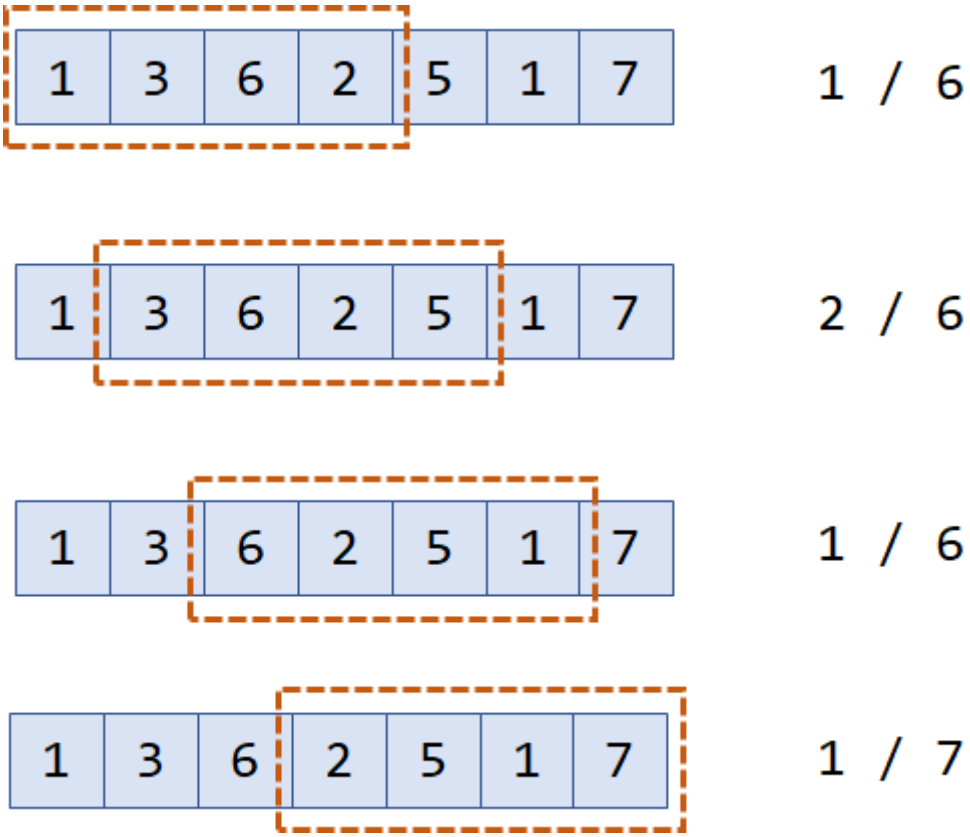
对于上面的符号问题，最好是每道题都去分析一下，有个小结论，取反。

单调队列

“如果一个选手比你小还比你强，你就可以退役了。”——单调队列的原理

貌似单调栈也有这种性质

单调队列是一种主要用于解决**滑动窗口**类问题的数据结构在长度为 n 的序列中，求每个长度为 m 的区间的区间最值。它的时间复杂度是 n，在这个问题中比线段树或者 st 表优秀。



1. 思考一下如何用普通队列做，那么只要用一个队列来维护当前窗口里的所有值，然后线性扫描一下队列，每次新加进来一个数，就把队尾加一个数，队头弹一个数出去，很显然时间复杂度是 $O(nk)$
2. 类似于单调栈的想法，观察一下哪些数永远不可能作为答案了，也就是说没有用了。

m = 4

1 3 6 2 5 1 7

所以假如要求区间最小值，有一个数如果更靠后并且更小，那么前面的数字就没有用了。所以新加进来一个数，可以把靠右的比他大的数弹出去，然后因为区间往右边挪了，维护的数的左端点也要合法，所以弹出左侧不合法的下标，因为停止的条件是碰到了一个比他小的数，所以容易想到队列里面的元素排列是递增

的，所以叫单调队列，因为维护的都是可能的答案加上单调递增。所以处理完之后，最小值就是队头的元素，就可以 $O(1)$ 得到。

因为需要左侧，右侧，可以联想到 $deque$ 这个数据结构。

```
void solve() {
    int n, m;
    cin >> n >> m;

    vector<int> a(n + 1);
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
    }

    deque<int> q;
    for (int i = 1; i <= n; ++i) {、
        //这个其实每次最多执行一次
        while (!q.empty() && i - q.front() >= m) {
            q.pop_front();
        }
        while (!q.empty() && a[q.back()] >= a[i]) {
            q.pop_back();
        }
        q.push_back(i);
        if (i >= m) {
```

```
            cout << a[q.front()] << " \n"[i == n];
        }
    }
}

```