栈

概念

栈是受限的序列。

只能在 top 插入和删除，bottom 的情况不可知。

实现：基于向量或列表派生。

应用

调用栈 & 消除尾递归

调用栈，就是把函数的实例和寄存器等等需要保存的信息都塞到一个栈里面。注意这个栈是从高往低去用的。

如果我们显式地维护调用栈，并利用我们知道但编译器不知道的优化，就可以“消除递归”。

尾递归是可以消除的。因为是“一连串的return”，连续地在 调用栈上 pop。

进制转换

在除法取余之后，得到的余数需要倒序输出。这个时候就可以采用栈来对这个问题进行优化。

括号匹配

如何判断一个括号序列的括号是否是匹配的？

消去一对**紧邻**的左右括号，不影响全局判断。

紧邻——上一个——最近的一个——栈的先进后出性质。

方法：

- 遇到左“括号”，将其入栈

- 遇到右“括号”，考察栈顶是否匹配，栈顶出栈。

栈混洗 - Catalan / 括号序列

考察栈 $A=\langle a_1,\cdots,a_n]$ 和栈 $B=\emptyset$，中间栈 $S=\emptyset$。

只允许：`S.push(A.pop())`、`B.push(S.pop())`

最终要达到 $A=S=\emptyset$

栈混洗的结果个数 $\mathrm{SP}(n)=\sum_{k=1}^{n}\mathrm{SP}(k-1)\mathrm{SP}(n-k)$。

> 考虑 S 第一次变空（此时 pop 的一定是第一个元素 a_1）的时候，假设在 B 里面已经有 k 个元素。那么在 A 进入右侧之前，右侧已经有 $k-1$ 个元素，左侧还要有 $n-k$ 个元素，这是唯一的枚举方法。

这是卡特兰数，$\mathrm{SP}(n)=\dfrac{(2n)!}{n!(n+1)!}$。栈混洗是等价于括号序列的。

禁形：312模式的序列。 Knuth, 1968.

判断禁形，最快的方法是用栈直接模拟。

中缀表达式求值

核心思想：需要延迟缓冲

> 即，仅根据表达式的前缀，不足以确定各运算符的计算次序。

但是，什么情况下是可以确定一定会算的呢？

也就是在一个高优先运算符之后，紧接着出现了一个低优先级运算符，那么前面那个高优先级的运算符就可以先算了。

所以本质上是维护一个运算符优先级单调升高的单调栈，然后每次遇到新的运算符就一直pop，直到栈顶的运算符优先级比新入栈的低。

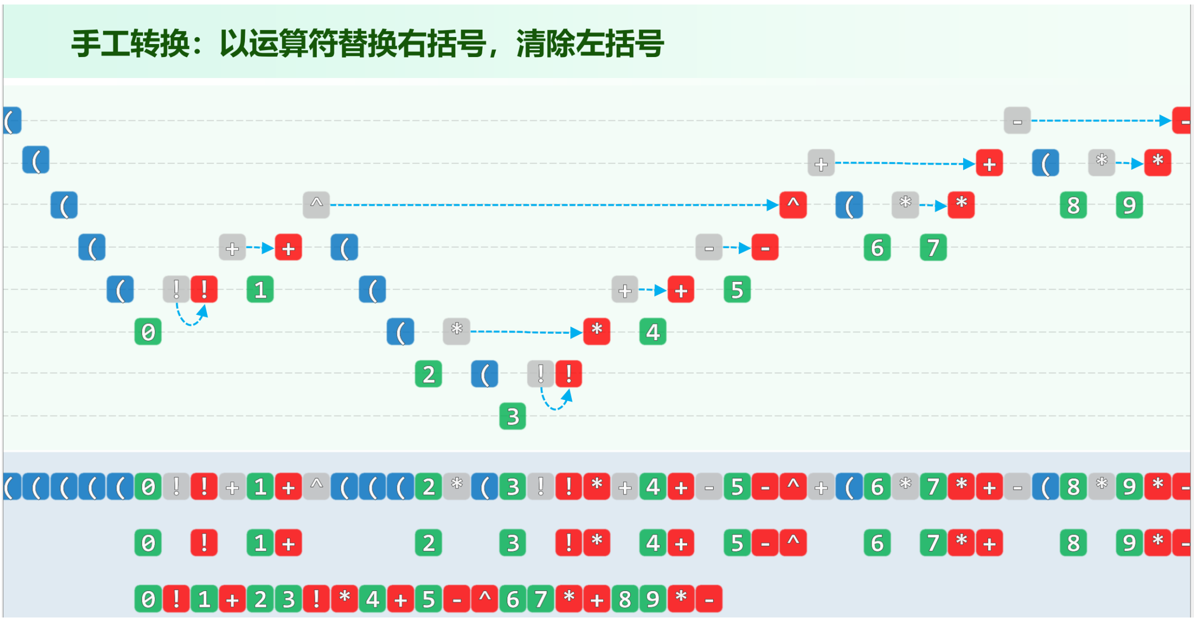
可以用哨兵来简化计算和求值。

逆波兰表达式(后缀表达式)

求值

硬求。需要数的时候从栈里面取数，计算出来结果 push 回栈里面。

转换



队列

概念

也是受限的队列，只能在队尾插入，查询队首。

应用

资源分配、银行服务模拟。

双栈当队

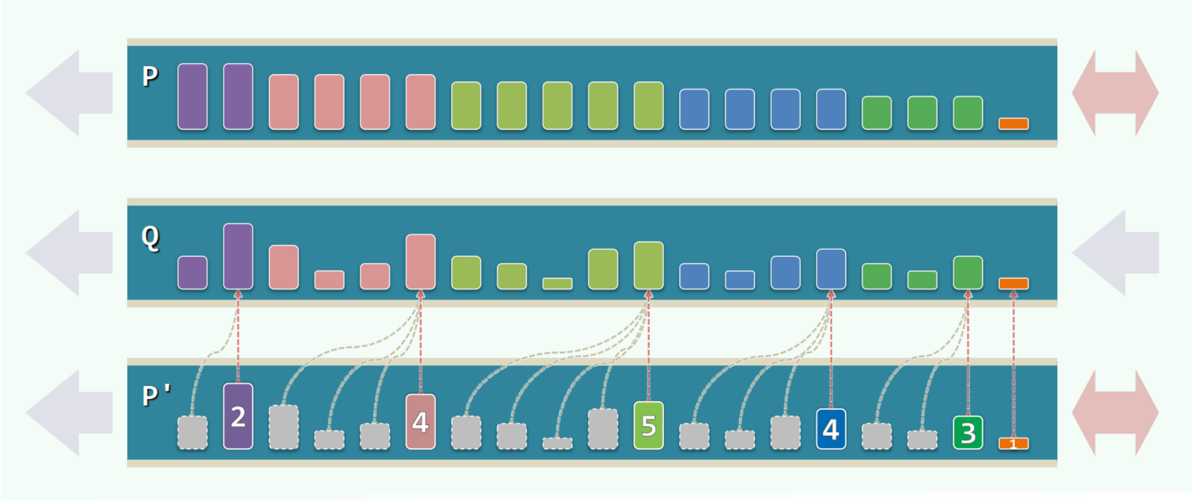
均摊仍然是 $O(1)$ 的（更确切地说，是 $O(4n)$ 的）。

分摊可以用 cost 的方法（给每个元素一些钱，元素进行操作需要付钱，本质就是换个方法数数）；也可以用聚合的办法（就是数总共多少次操作，每次操作的成本，求和之后除以操作次数）。

单调队列

内部元素具有单调性的队列。

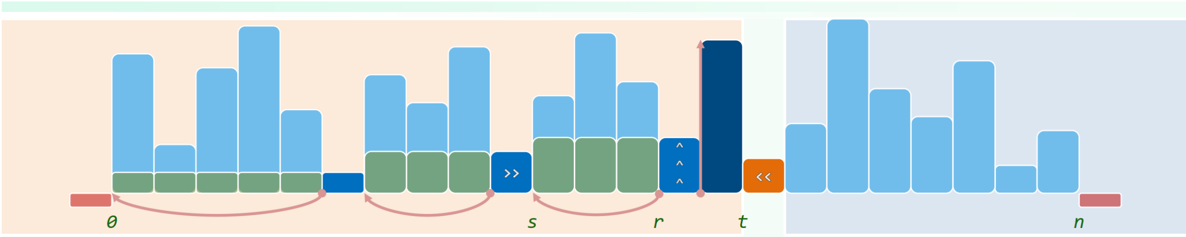
讲义里管这个叫 Queap，但这个概念不如单调队列直观。



时间复杂度为dequeue $\mathrm{O}(1)$，enqueue均摊$\mathrm{O}(1)$

直方图内最大矩形

定义$s[r], t[r]$表示r左右方向上第一个高度小于r的柱高度的坐标，也就是r的左右“限位”，定义这样一组特殊的限位：前一项是后一项的左限位。用一个栈来存储这些限位，每当扫描到一个新的柱时，如果其高度小于栈顶柱高，则说明这是一个新的限位，此时不断取出栈中元素，直到到达一个小于新柱的限位，最并最大矩形大小后让新柱入栈，继续向下扫描。



时间复杂度为均摊$\mathrm{O}(n)$，且为**就地**和**在线**算法。