数据结构

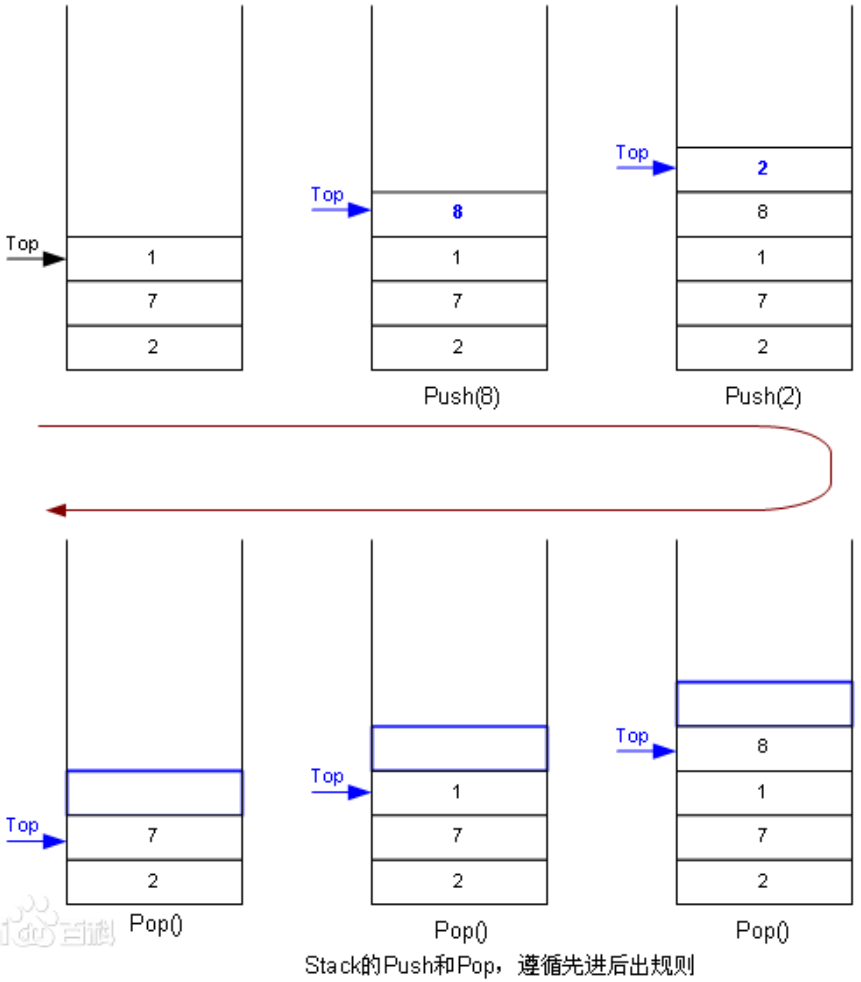
吴清月

数据结构

- 栈
- 队列
- 优先队列（堆）
- 单调队列
- 线段树、树状数组
- 分块

栈

- 栈是一种支持在栈顶插入和删除的数据结构。
- 实现非常简单，开一个stack数组再用一个top指向栈顶就可以了。
- 应用：
  - 括号匹配
  - Tarjan算法
  - dfs中的系统堆栈
- 在STL中是stack



单调栈

- 首先看一个问题：
- 你有一个长度为n的序列，你需要对于里面的每一个元素，找到它左边第一个比它大的数。
- 要求复杂度$O(n)$。

- 注意到，如果对于$x < y$有$a_x < a_y$，那么x这个元素就没用了。
- 所以用一个栈从左到右扫，遇到一个数就不断弹出栈顶比它小的数，最后把它压入栈中。
- 时间复杂度是$O(n)$的。

队列

- 队列是一种支持在队尾插入，队首删除的数据结构。
- 可以用一个数组加两个指针实现。
- 应用：
  - bfs
  - SPFA（它死了）
- 在STL中是queue。

优先队列（堆）

- 写一个数据结构，支持插入一个元素、查询最小值和删除最小值。
- 可以用堆来实现。
- 堆本质上是一棵完全二叉树。
- 应用：
  - 堆优化Dijkstra
  - 结合启发式合并等各种东西
- 在STL中是priority_queue，需要引用queue头文件。

单调队列

- 洛谷1886 滑动窗口
- 给你一个长度为n的序列，求这个序列里面所有长度为k的连续子序列的最大值和最小值。
- 要求复杂度$O(n)$。

- 假设求的是最大值。和单调栈一样，如果对于$x < y$有$a_x < a_y$，那么x这个元素就没用了。
- 唯一的一个区别是要求长度为k，这个比较好处理，如果队首元素太靠前了就不断弹出即可。
- 时间复杂度$O(n)$。

线段树

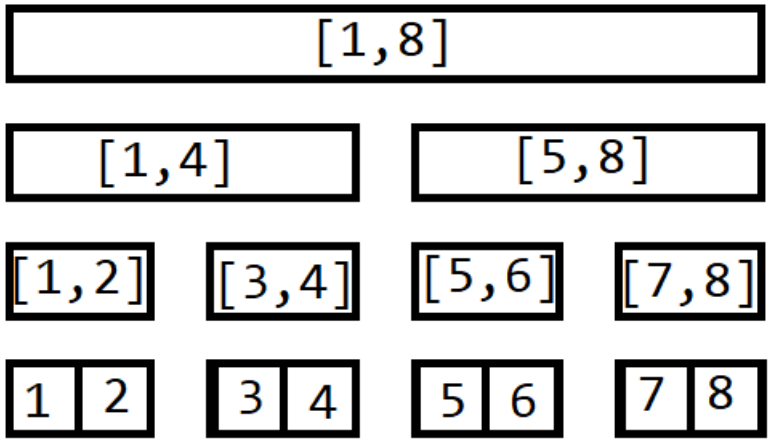
- 写一个数据结构，支持单点修改，单点查询。
- ？？？
- 用数组实现

- 写一个数据结构，支持区间求和。
- ？
- 用前缀和实现。

- 写一个数据结构，支持单点修改，区间求和。
- 线段树！

线段树

- 首先假设n是2的幂。
- 在整个序列上建立如图所示的二叉树：
- 每一个节点存储这个区间的和是多少。
- 修改直接更改相应的位置即可。
- 每一次只会修改$O(\log n)$个节点。
- 查询时，一个区间会被分成 $O(\log n)$ 个小区间，时间复杂度还是$O(\log n)$。

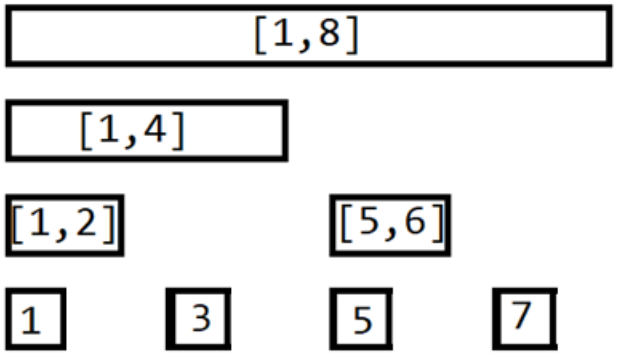


线段树

- 写一个数据结构，支持区间修改，区间查询。
- 还是用线段树来实现。
- 区间查询好做，如何实现区间修改呢？
- 注意到一个区间会被分割成$O(\log n)$个小区间，所以我们直接把这些小区间打上一个标记表示被修改过。
- 查询到这个点的时候把这个标记下放到子节点。

树状数组

- 如果维护的信息是可减的，那么可以在线段树上删掉一些没用的节点；
- 这样形成的结构叫做树状数组。
- 树状数组是一个数组，常数比线段树要小很多。
- 复杂度还是一个log。



洛谷1739 表达式括号匹配

- 输入一个字符串，问括号匹配是否合法。
- $n \leq 10^5$

- 直接用栈就好了。

洛谷1908 逆序对

- 给你一个序列，求逆序对数量。
- $n \leq 5 \cdot 10^5$

洛谷1908 逆序对

- 方法一：归并排序。
- 尝试在归并排序的时候统计逆序对。
- 每次归并的时候，只需要考虑左边的元素和右边的元素产生的逆序对。
- 两边用两个队列，每次弹出最小的，然后看它形成了多少逆序对。
- 方法二：树状数组
- 先离散化，然后从左到右插入到树状数组中，查询它左边有多少元素比它大。

- 时间复杂度都是$O(n \log n)$。

NOIP2004 合并果子

- 有n堆果子，每一堆有一个重量，每次可以选择将两堆果子合并，付出的体力值为两堆的重量之和。
- 求将所有果子合并为一堆的最小体力。
- $n \leq 10000$（2004年的电脑）

NOIP2004 合并果子

- 每次贪心找最小的两堆进行合并一定是最优的。
- 所以用一个堆维护一下，每一次取出最小的两个元素，然后把它们的和加入到堆里面即可。
- 时间复杂度$O(n\log n)$。

- 实际上还有一个更加优秀的解法。
- 注意到，每次合并出来的元素一定是单调的。
- 所以拿两个队列维护，先把所有数sort一下放到第一个队列里面，然后每次看是第一个队首更大还是第二个队首更大，找出最小的两个，弹出后加在一起放到第二个队列末尾。
- 时间复杂度还是$O(n\log n)$，但是复杂度瓶颈是排序。

NOIP2016 蚯蚓

- 有n段蚯蚓，每次你会选择一条最长的，假设长度为x，并且把它切成长度为$\lfloor px \rfloor$和$x - \lfloor px \rfloor$的两段，同时其余的所有蚯蚓长度都会增加q。
- 问你m秒内切割的每一条蚯蚓的长度和最后的所有蚯蚓的长度。
- 要求复杂度线性。

NOIP2016 蚯蚓

- 首先可以用一个堆实现$O\big((n+m)\log n\big)$。
- 如何线性呢？
- 注意到，如果不切割，那么所有蚯蚓长度的大小关系是不会变的。
- 同时，如果我们在t_1时刻切割了一条初始长度为x蚯蚓，在t_2时刻切割了一条初始长度为y的蚯蚓，其中$t_1 < t_2, x > y$。
- 然后我们注意到，第一次切下的蚯蚓初始长度长，增加的长度还更多（先切下来增加的长度更多），所以第一次切下的一定比第二次切下的要长。
- 也就是满足单调性的。和上一道题一样，开三个队列，第一个队列存储原来的蚯蚓，第二个队列存储切下来的第一条蚯蚓，第三个队列存储切下来的第二条蚯蚓。这样时间复杂度就降到$O(n\log n+m)$了。

多重背包问题

- 有n种物品和一个大小为m的背包，每一种物品有一个重量w和一个价值v，第i种物品有c_i个，求总容量不超过上限的情况下的最大价值。
- $n, m \leq 5000$

多重背包问题

- 可以二进制拆分：
- 假设这种物品有40个，那么就把它捆成1,2,4,8,16,9六个包，这样40个物品就变成了6个。
- 时间复杂度是$O(nm\log n)$。

- 能不能更优呢？
- 单调队列优化。

单调队列优化

- 我们考虑用单调队列来优化DP。
- 这样的DP状态转移方程如下：
- $f_i = \max\limits_{i-r \le j < i-l} f_j + a_i$
- 由于max不满足可减性，所以不能直接记录前缀和。
- 怎么办呢？

单调队列优化

- 如果一个数x在另一个数y右边而且$f_x > f_y$，那么y就没用了。
- 所以我们维护一个单调递减的队列，当我们处理到第i个位置的时候，我们把$i-l$这个位置加入到单调队列中，并且把$i-r-1$这个位置弹出。
- 时间复杂度是$O(n)$。

单调队列优化

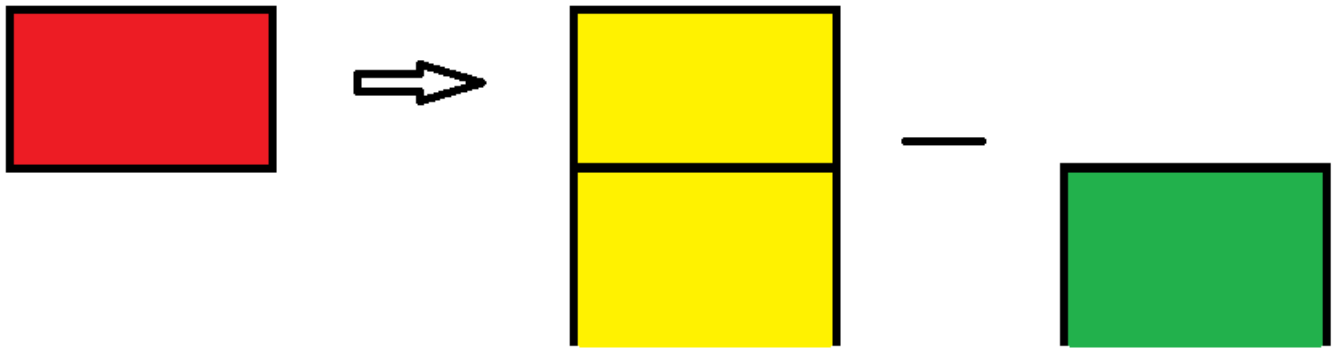
- 我们设$f_{i,j}$表示前i个物品，背包容量是j的最大价值。
- 则状态转移方程为$f_{i,j}=\max\left(f_{i-1,j-kw_i}+kv_i\right)$，也就是枚举这个物品选了多少个。
- 我们考虑单调队列优化的模型，$f_i=\max\limits_{i-k\le j<i} f_j+a_i$，是不是和这个有点像？
- 唯一一个不同点是模型是一段连续的区间，而这里面是一段等差序列。
- 我们按照$\bmod w_i$把整个DP数组第二维分成w_i个不同的部分，每一部分分别转移即可。
- 换句话说，假设背包容量是12，物品体积是3，那么1,4,7,10是一个部分，2,5,8,11是一个部分，3,6,9,12是一个部分，每一个部分内部都是可以单调队列优化的。
- 时间复杂度降到$O(nm)$。

二维数点问题

- 平面上有n个点(x_i, y_i)，还有q组询问，每次询问一个矩形里面有多少个点。
- $n, q \leq 10^5, -10^9 \leq x_i, y_i \leq 10^9$
- 允许离线

二维数点问题

- 矩形可以差分。



- 这样对一个矩形的询问就被我们拆成了两个对前缀的询问相减。
- 把所有的询问排序，所有的点按照横坐标排序，然后依次扫过去，点就变成了在y_i处+1，询问就变成了区间和。
- 用线段树/树状数组实现。注意离散化。

HDU 6315

- 给你一个长度为n的排列b_i，你需要维护一个初始全0的序列a_i，支持以下操作：
  - 把$[l,r]$内的a_i都+1
  - 求$\sum_{l\le i\le r}\left\lfloor\frac{a_i}{b_i}\right\rfloor$
- $n,q\le 10^5$

HDU 6315

- 首先，每一个元素最多加到n。
- 所以，所有$\frac{a_i}{b_i}$总共只会被更改$\frac{n}{1}+\frac{n}{2}+\frac{n}{3}+\cdots+\frac{n}{n}=O(n\log n)$次。
- 每次修改暴力就好了。
- 也就是说，线段树上每一个节点维护这段区间还需要加多少就会有一个$\frac{a_i}{b_i}$发生改变。一旦出现0就暴力更改。
- 时间复杂度$O(n\log^2 n)$

SPOJ GSS3

- 维护一个长度为n的序列，有两种操作：
  - 单点修改
  - 询问所有$[l,r]$的子区间中权值之和最大的子区间的权值和。
- $n,q \leq 50000$

SPOJ GSS3

- 线段树上每一个节点维护如下信息：
  - 整体和sum
  - 最大的前缀和pre
  - 最大的后缀和suf
  - 最大的子段和maxx
- 这些都可以$O(1)$维护。
- 如何查询呢？
- 维护一个全局的ans和s，分别表示当前的最大子段和和最大后缀和。
- 到达一个点之后用s+pre和maxx更新ans，用suf和s+sum更新s，最后ans就是答案。
- 时间复杂度$O(n \log n)$。

CodeForces 1140C

- 有n首歌，每一首歌有一个长度t_i和一个美妙值b_i，你需要选出k首歌，最大化选出的$\sum t_i \cdot \min(b_i)$。
- $1 \leq k \leq n \leq 10^5$

CodeForces 1140C 1600

- 假设我们选择了一首美妙值为v的歌，那么所有美妙值大于等于v的歌都不会对min造成影响，我们只需要选择长度最长的k个。
- 所以先按照v排序，依次枚举美妙值最小的那首歌，同时维护前面的所有歌中长度最长的k个。
- 用一个小根堆实现即可。如果size太大了就往外弹出最短的。
- 时间复杂度$O(n\log n)$。

CodeForces 1208D

- 你有一个长度为n的排列，但是你不知道它是什么。
- 对于每一个数，你知道它前面所有小于它的元素之和s_i。
- 求这个排列。
- $n \leq 2 \cdot 10^5$

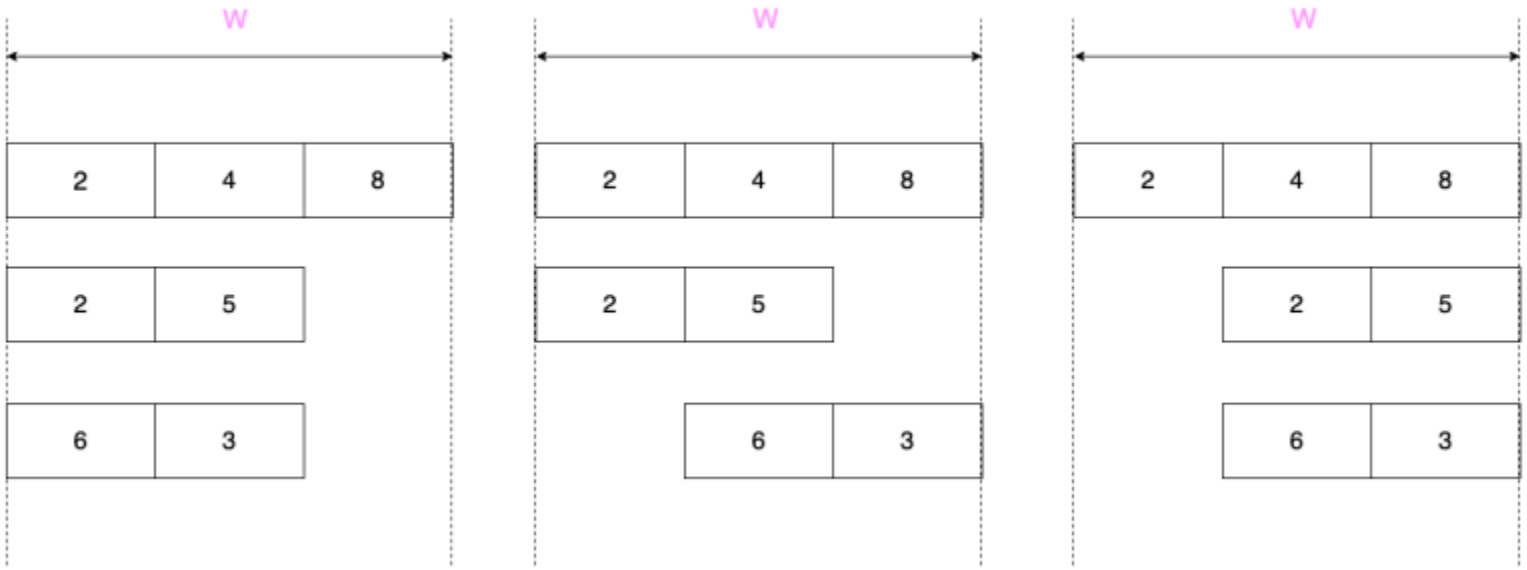
CodeForces 1208D 1900

- 首先考虑1。它前面一定没有比它更小的元素，同时它后面所有的元素都要比它大，也就是s_i是最后一个0。
- 这样我们就确定了1的位置。它后面的元素的s肯定都包含它，所以先把后面的所有的s减去1。
- 考虑2。这时候2的位置就是最后一个0的位置了。找到这个位置并且把它后面的元素都减去2。
- 重复这个过程，就可以得到整个排列。
- 用线段树实现区间加和求最后的一个0。
- 时间复杂度$O(n \log n)$。

CodeForces 1208E

- 有n个序列，第i个序列的长度是l_i。这些序列并排放在一起，每一个序列都放在一个长度为w的框里面。
- 这些序列可以在框里面滑动，但是不能划出框。
- 对于每一个位置i，你需要求出怎样滑动可以使得这一个位置上的数的和最大。
- 序列总长度不超过10^6，$-10^9 \leq a_{i,j} \leq 10^9$。

CodeForces 1208E



CodeForces 1208E 2200

- 要想这一个位置所有数的和最大，只需要使得每一个序列的这个位置都最大。
- 现在我们的问题就是如何通过滑动第i个序列使得第p个位置上的数最大。
- 首先如果这个位置距离两端都大于等于l_i，那么p这个位置可以是这个序列里面的任何一个数。
- 否则，可以发现可行的一定是原序列里面的一个区间。超过这个区间的因为紧贴墙而不可行。
- 所以对于每一个序列，中间的部分就是整个序列里面的最大值和0取max，两边的部分可以暴力。
- 用一个线段树记录每一个位置的答案，用ST表实现求一个区间的最大值。
- 时间复杂度$O(\sum l \log l)$。

CodeForces 1167F

- 有一个长度为n的数列a，里面所有的元素都互不重复。定义函数$f(l,r)$表示将$[l,r]$内的数升序排序后，第i个位置的数乘i后的所有元素和。
- 求所有$f(l,r)$的和。
- $n \leq 5 \cdot 10^5$

CodeForces 1167F 2300

- 对于每一个元素，考虑它对答案的贡献。
- 可以发现，只要这个区间多包含了一个比它小的元素，那么它就会被多算一次。
- 所以我们只需要看对于每一个小于它的元素，有多少个区间包含了它，那么答案就多算几次。
- 比如这个序列长下面这种情况：
- 10 3 10 5 10 3 10 3
- 那么5就会被计算2*5（第一个3）+4*5（它自己）+4*3（第二个3）+4*1（第三个3）=46次。

CodeForces 1167F 2300

- 更进一步，假设这个数的位置是x，它左边有a个比它小的数，位置分别是$p_1, p_2, p_3, \cdots, p_a$，右边有$b$个比它小的数，位置分别为$q_1, q_2, q_3, \cdots, q_n$，那么它就会被计算$(n-x+1)\sum p_i + x\sum(n-q_i+1)$次。
- 也就是说只要知道了它左边所有元素到开头的距离之和以及它右边所有元素到结尾的距离之和，就可以求出答案。
- 可以用线段树来维护。将所有的数排序后挨个插入，并且维护上面的两个和。
- 时间复杂度$O(n\log n)$。

CodeForces 1158C

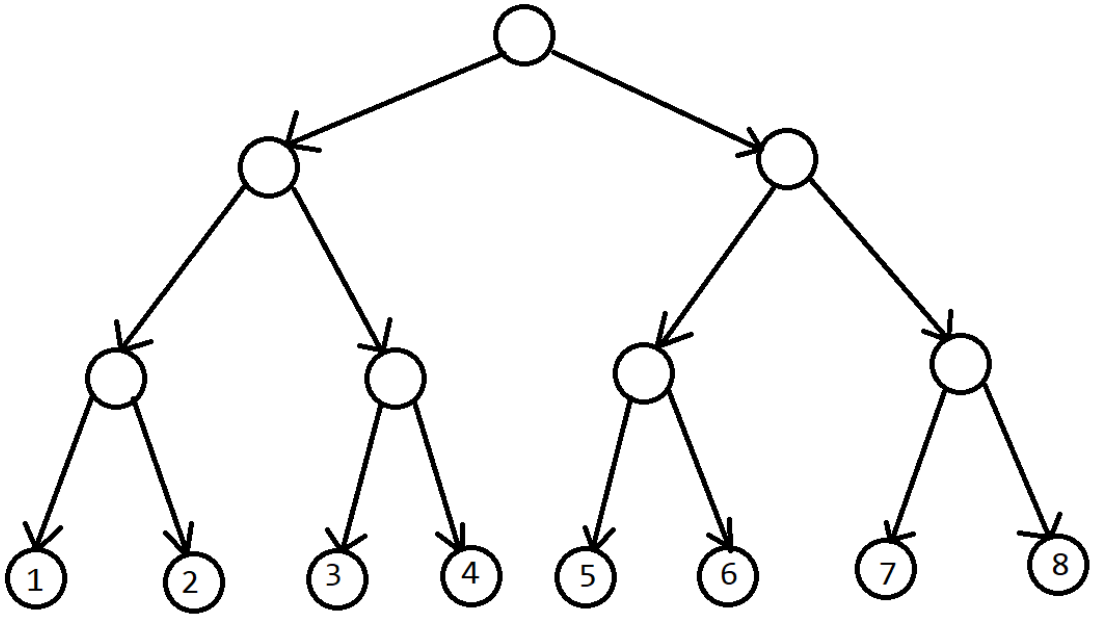
- 有一个排列，你知道每一个元素右面第一个大于它的元素的位置，让你还原出这个排列。
- 无解输出-1。
- $n \leq 100000$

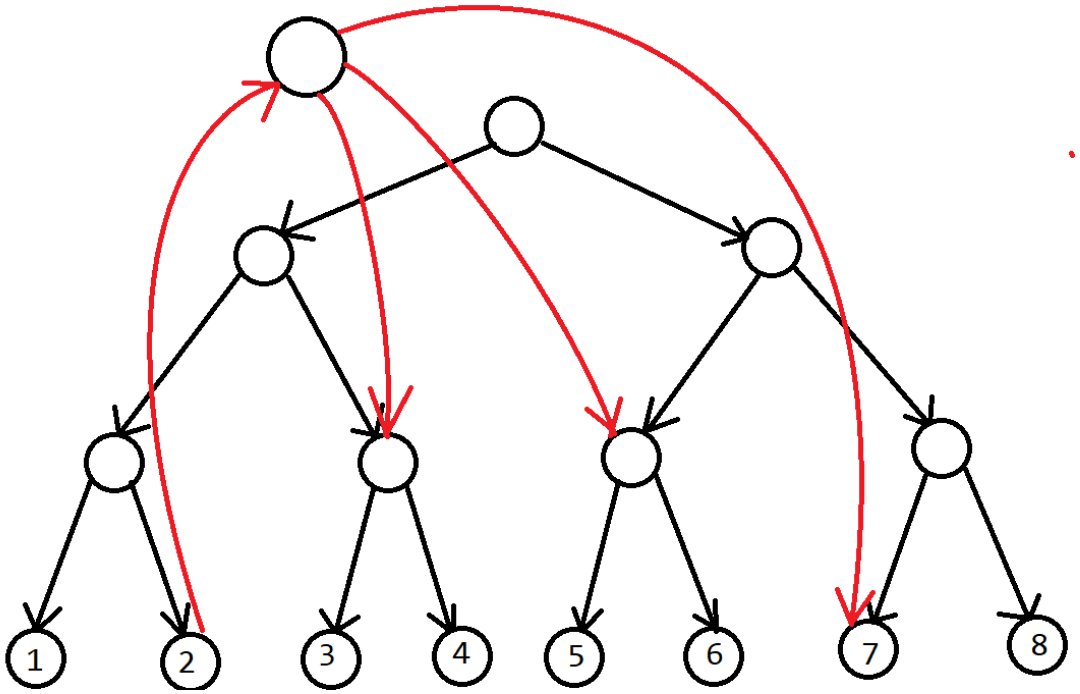
CodeForces 1158C 2300

- 假设我们知道了p_x右边第一个大于它的数是p_y，那么就能知道$p_x < p_y$，同时$p_x > x$和y之间的所有元素。
- 也就是说我们得到了一个偏序关系。建出图来，然后看是否存在拓扑序即可。
- 然而$n \leq 100000$，直接建图是$O(n^2)$的，所以我们接下来介绍一个黑科技——线段树优化建图。

线段树优化建图

- 一个点要向一个区间连边，怎么办呢？
- 线段树的复杂度很优秀，所以直接把线段树建到图上！
- 比如我们要将2连向[3,7]这个区间：





线段树优化建图

- 这样一次连边就是$O(\log n)$而不是$O(n)$的了。
- 点向区间连边？按照上面那个树建。
- 区间向点连边？上面那个线段树所有的边取反。
- 区间向区间连边？建立两棵线段树，一棵边向下指，一棵边向上指，然后分别把两个区间对应到两棵线段树上。
- 这样上面一道题就可以在$O(n \log n)$的时间复杂度内完成了。

CodeForces 1216F

- 有n间屋子，第i间屋子联网的花费是i。
- 有一些屋子可以建Wi-Fi，在第i间屋子建立Wi-Fi的花费还是i，但是信号可以覆盖它左右各k间屋子，这些屋子就不需要再联网了。
- 你需要将所有的屋子都联网，求最小花费。
- $n, k \leq 2 \cdot 10^5$

CodeForces 1216F 2300

- 考虑DP。设f[i]表示将1到i全部联网的最小花费。
- 怎么转移呢？
- i可以装Wi-Fi：$f_{i+k} = \min\limits_{i-k \le j < i+k} f_j + i$
- i不能装Wi-Fi：$f_i = f_{i-1} + i$
- 每次求完后，让$f_i = \min\limits_{j \ge i}(f_j)$，也就是说既然前j个都接好了，那么前i个也接好了。
- 用线段树维护整个DP数组，更新就对应了线段树区间查询。注意边界条件。
- 时间复杂度$O(n \log n)$。

根号平衡

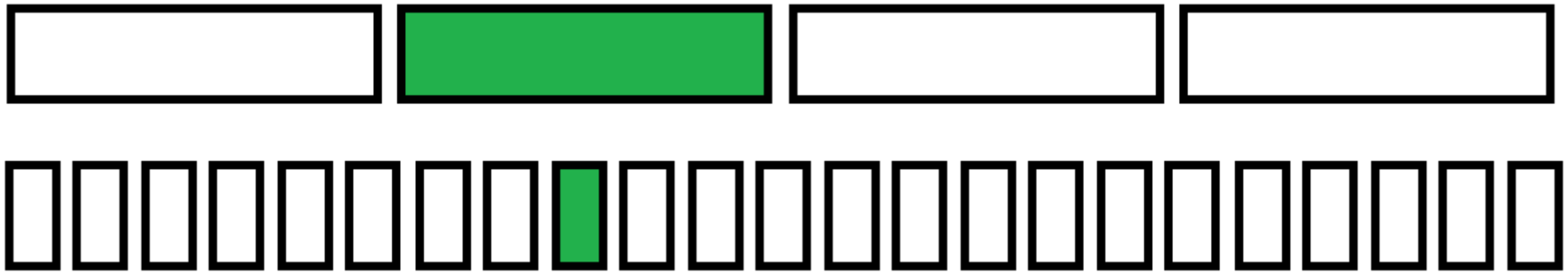
- 维护一个序列，支持：
- O(1)单点修改，O(sqrt(n))区间和

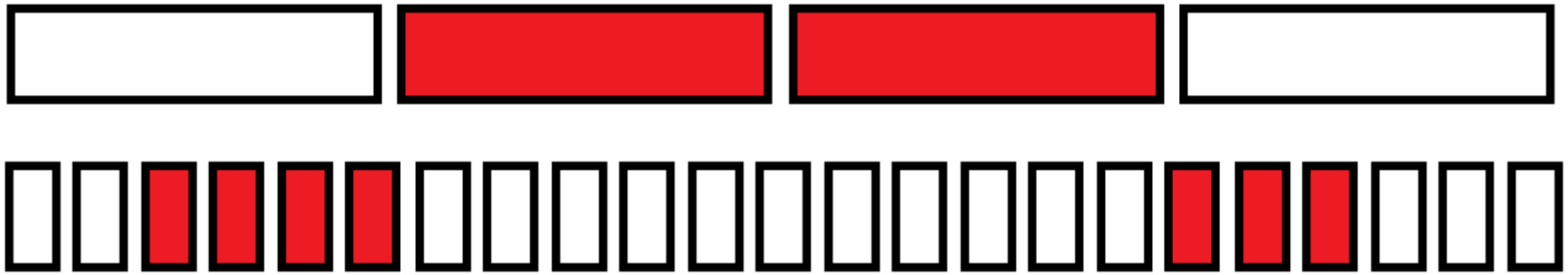
根号平衡

- 分块维护块内和，每次修改的时候更新块内和以及该位置在数组上的值
- 查询的时候就和普通分块一样查

根号平衡

- 修改：





根号平衡

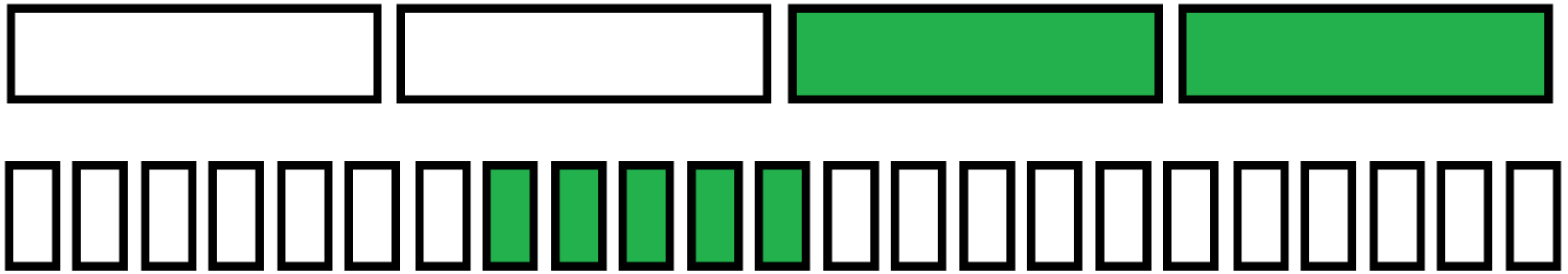
- 维护一个序列，支持：
- O(sqrt(n))单点修改，O(1)区间和

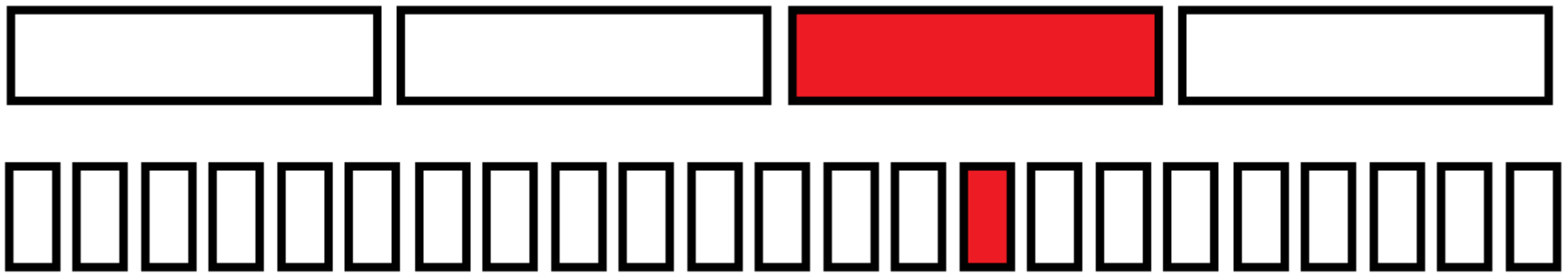
根号平衡

- 分块维护块内前缀和和块外前缀和
- 也就是说维护每个块块内前x数的和
- 以及维护前x的块的和
- 更新的时候分别更新这两个前缀和
- 查询的时候把这两个前缀和拼起来

根号平衡

- 修改：





根号平衡

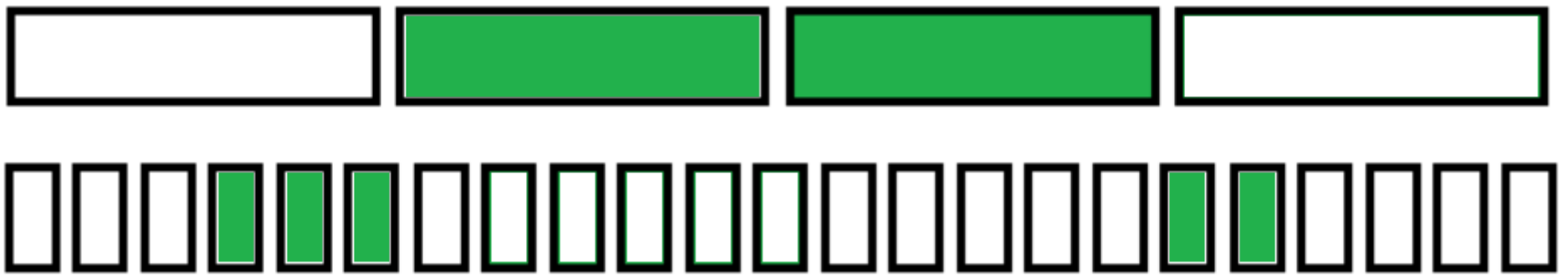
- 维护一个序列，支持：
- O(sqrt(n))区间加，O(1)查单点

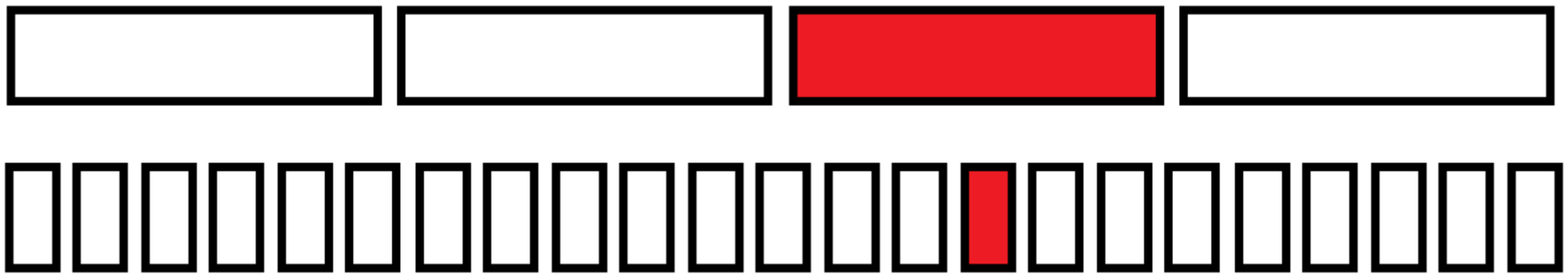
根号平衡

- 直接分块

根号平衡

- 修改：





根号平衡

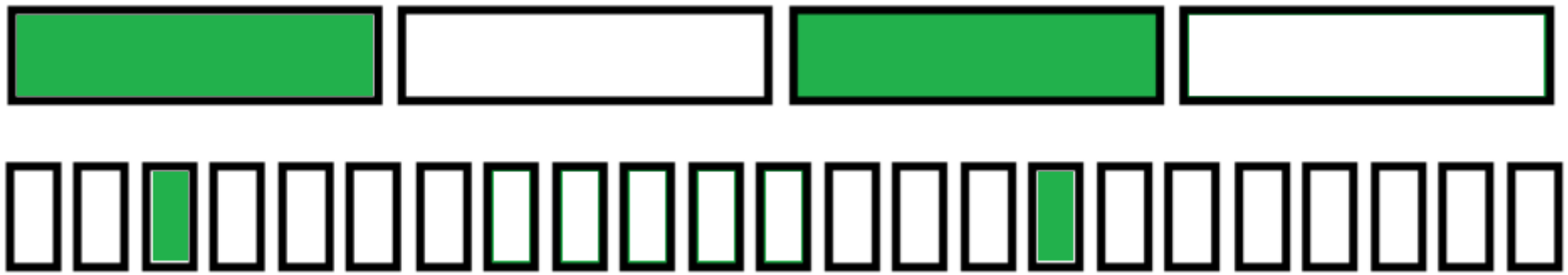
- 维护一个序列，支持：
- O(1)区间加，O(sqrt(n))查单点

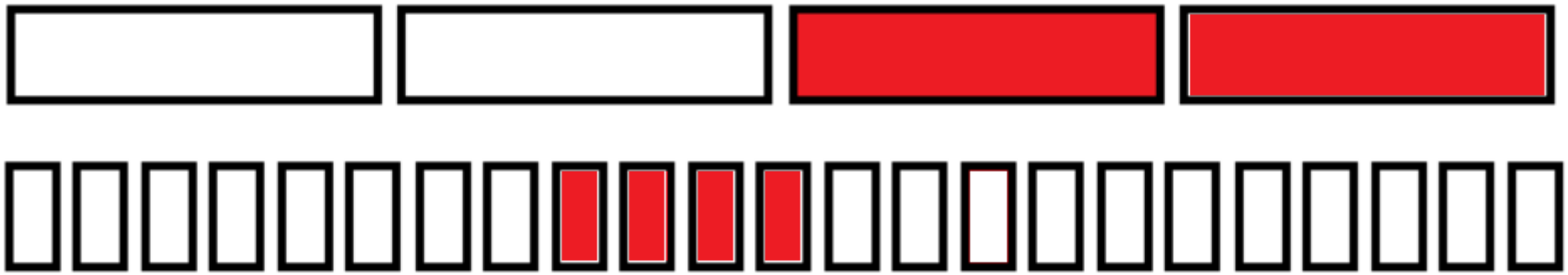
根号平衡

- 每次对区间[l,r]加x的时候
- 差分为前缀[1,l-1]减x，前缀[1,r]加x
- 同时在数组上和块上打标记
- 使得区间[l,r]加x
- 查询的时候就扫过块外的标记和块内的标记即可

根号平衡

- 修改：





根号平衡

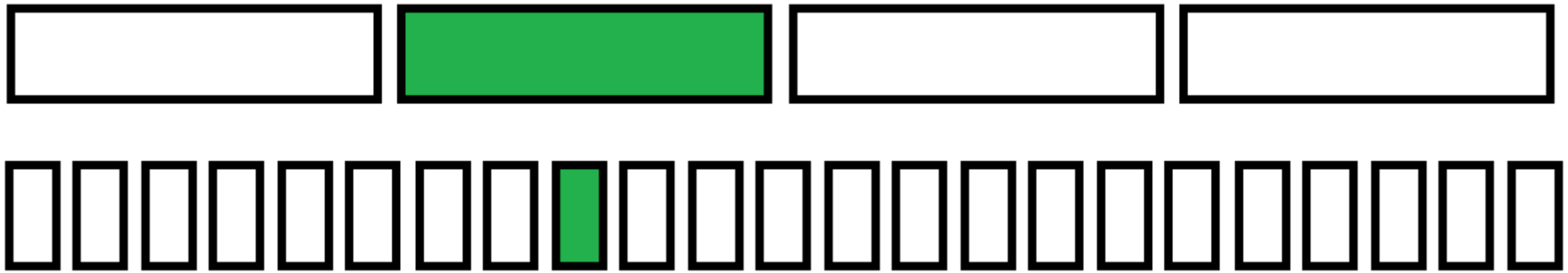
- 维护一个集合，支持：
- O(1)插入一个数
- O(sqrt(n))查询k小

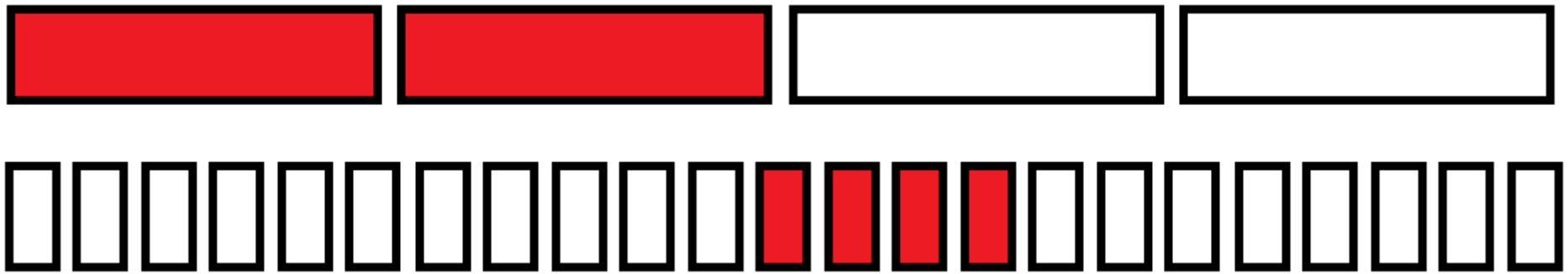
根号平衡

- 离散化后对值域进行分块
- 还是维护第i个块里面有多少个数
- 查询的时候从第一个块开始往右跑
- 最多走过sqrt(n)个整块和sqrt(n)个零散的数

根号平衡

- 修改：





根号平衡

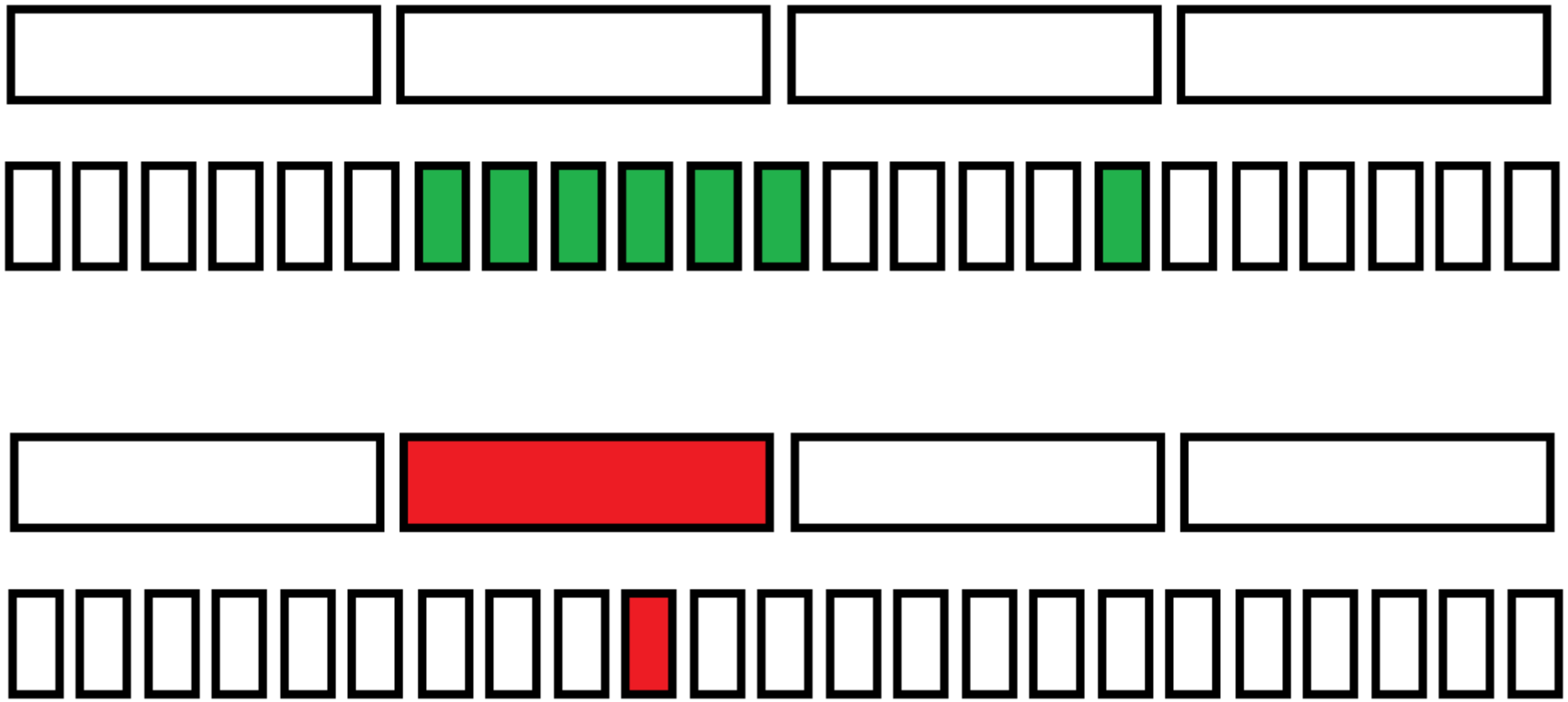
- 维护一个集合，支持：
- O(sqrt(n))插入一个数
- O(1)查询k小

根号平衡

- 值域分块
- 对于每个数维护一下其在哪个块里面
- 对于每个块维护一个OV（有序表）表示这个块内的所有数存在的数，从小到大
- 这样我们修改的时候只会改变sqrt(n)个数所从属的块
- 查询的时候定位到其所属于的块，然后找到其在该块中对应的值

根号平衡

- 修改：



CodeChef Chef and Churu

- 给n个数，给定m函数，每个函数为序列中第li 到第ri个数的和
- 有q个询问，有两种类型的操作：

- 1 x y 把序列中的第x个数改为y
- 2 x y 求第x个函数到第y个函数的和

Solution

- 因为函数不变，所以把函数分块
- 维护一个前i个块的函数的前缀和，代表前i个块中每个序列上的点的出现次数
- 然后再维护一个前i个块的函数的答案
- 每次修改只需要查询这个序列上的点在前i个块的函数中的出现次数即可
- 然后零散的部分，即用一个O(sqrt(n))修改，O(1)查询的分块维护即可

- O(nsqrtn + msqrtn) = O(msqrtn)