Segment Tree

$n + e$

Tsinghua University

2016 年 7 月 16 日

Introduction

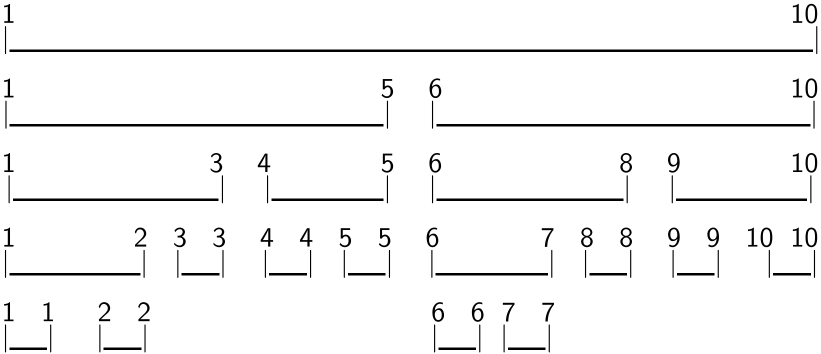
- 给你一段长度为 N 的序列 A[], 求:
  1. 单点修改, 单点查询;
  2. 单点修改, 区间查询;
  3. 区间修改, 单点查询;
  4. 区间修改, 区间查询;
- $N \leq 100000$
- 修改操作包括但不限于 + C, * C, 强制 = C, ……
- 查询操作包括但不限于 max, min, sum, ……

1. Introduction
   - 长啥样?
   - 修改
   - 查询
   - Lazy tag
   - 代码
2. Application

- 线段树将一个区间划分成一些单元区间，每个单元区间对应线段树中的一个叶结点.
- 根节点对应区间为 $[1, N]$
- 对于线段树中的每一个非叶子节点 $[a, b]$，它的左儿子表示的区间为 $[a, \frac{a+b}{2}]$，右儿子表示的区间为 $[\frac{a+b}{2}+1, b]$. 因此线段树是平衡二叉树，最后的叶子节点数目为 N.
- 为提高运行效率，$\frac{a+b}{2}$ 常常写成 $a+b \gg 1$
- 堆式存贮：父亲节点编号为 i，则左儿子编号为 $i*2$，右儿子编号为 $i*2+1$
- 为提高运行效率，常常写成 $i \ll 1$ 和 $i \ll 1|1$

举个栗子

假设 N=10

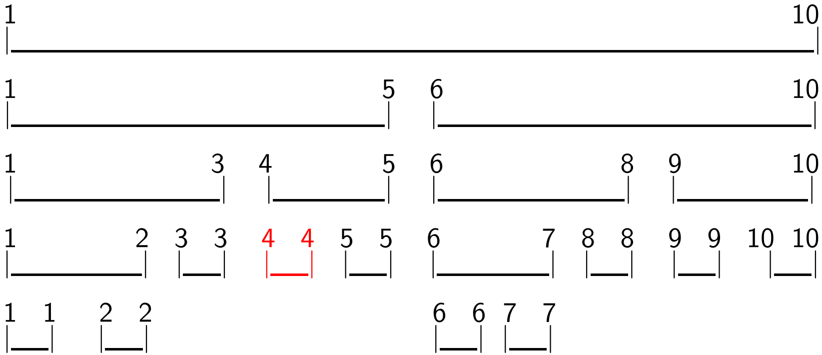


- 易知树高为 logN

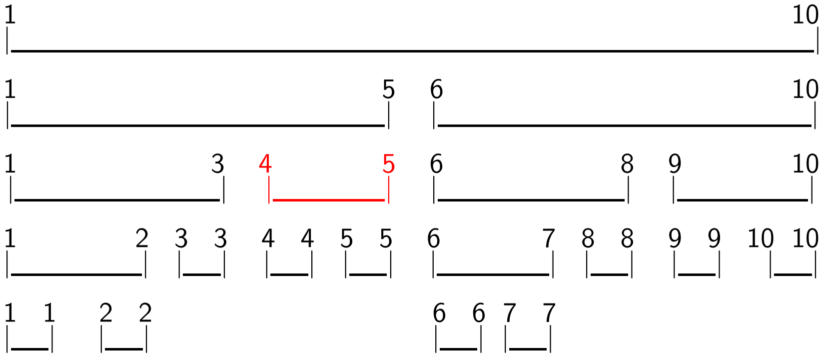
```
void Build_Tree(int o, int l, int r) {
    if (l == r) { sum[o] = a[l]; return;}
    int mid = l + r >> 1;
    Build_Tree(o << 1, l, mid);
    Build_Tree(o << 1 | 1, mid + 1, r);
    sum[o] = sum[o << 1] + sum[o << 1 | 1];
}
```

1. Introduction
   - 长啥样?
   - **修改**
   - 查询
   - Lazy tag
   - 代码
2. Application

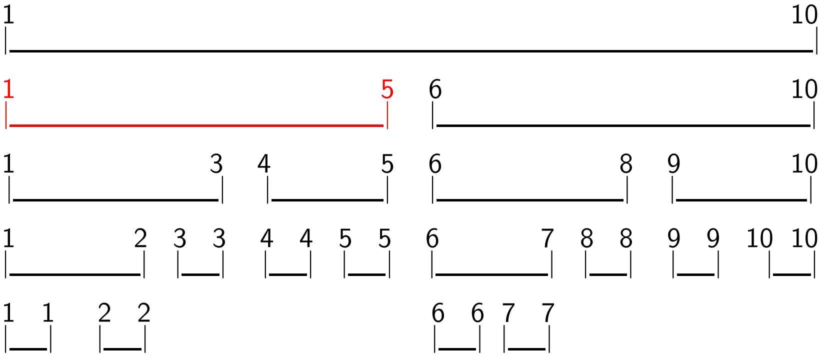
修改 A[4]



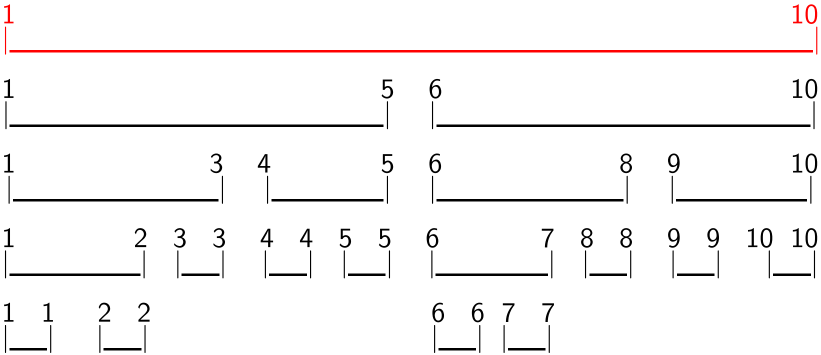
修改 A[4]



修改 A[4]



修改 A[4]

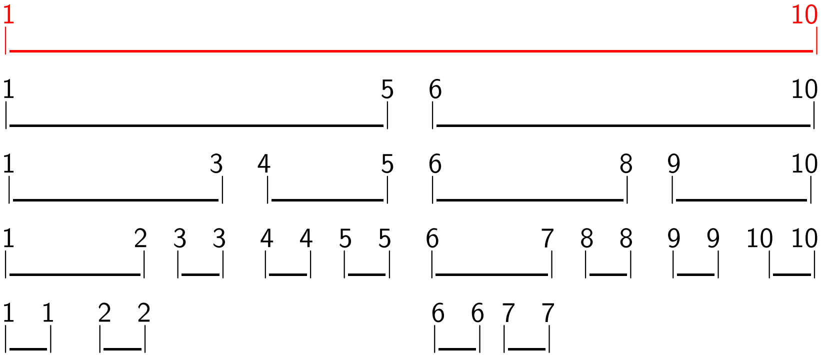


- 易知单点修改复杂度为 logN
- 单点查询类似

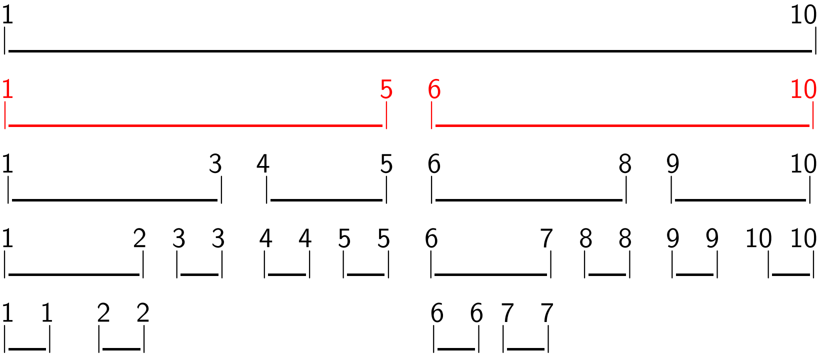
```
void Update(int o, int l, int r) {//A[x] = y
    if (l == r) { sum[o] = y; return;}
    int mid = l + r >> 1;
    if (x <= mid) Update(o << 1, l, mid);
    else Update(o << 1 | 1, mid + 1, r);
    sum[o] = sum[o << 1] + sum[o << 1 | 1];
}
```

1. Introduction
   - 长啥样?
   - 修改
   - **查询**
   - Lazy tag
   - 代码
2. Application

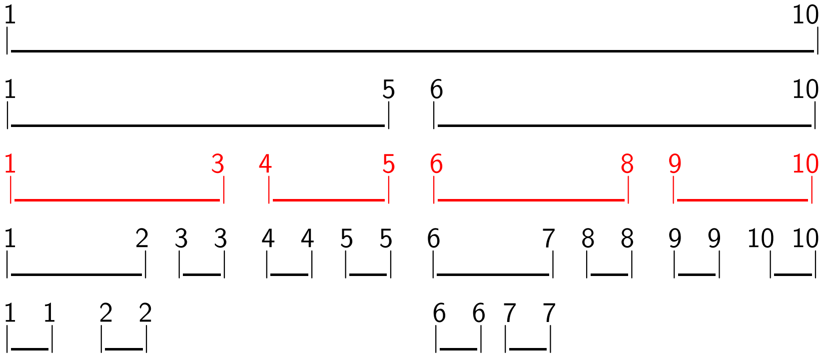
查询 A[2..9]



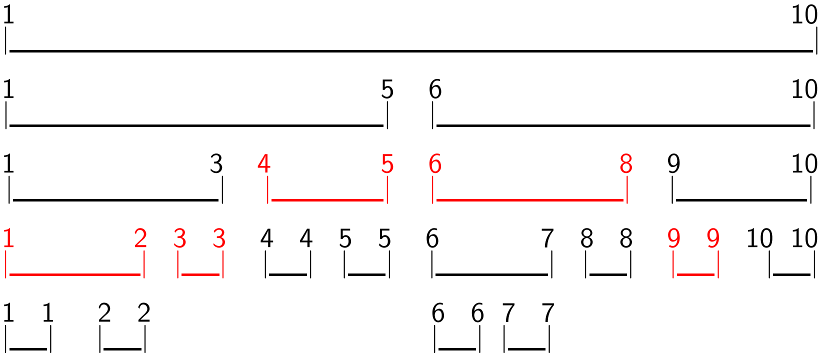
查询 A[2..9]



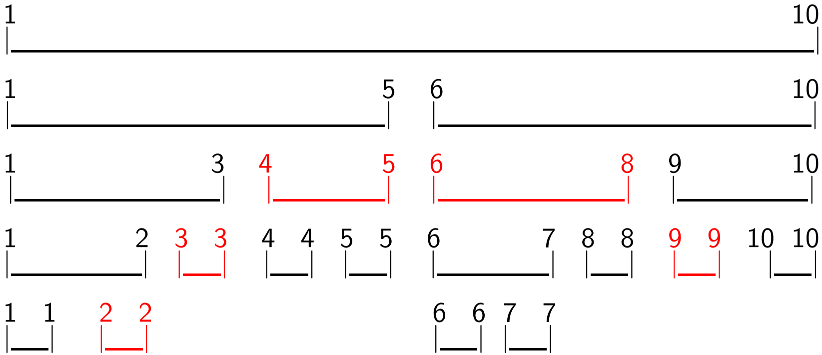
查询 A[2..9]



查询 A[2..9]



查询 A[2..9]



- Q: 为什么不直接查 [2,9]? A: 因为没有 [2,9] 这段区间啊 · · ·
- 易知能够通过访问不超过 2*logN 个线段树上的区间来获得任意区间 [l,r] 的答案

```
void Query(int o, int l, int r) {//A[x..y]
    if (x <= l && r <= y) { ans += sum[o]; return;}
    int mid = l + r >> 1;
    if (x <= mid) Query(o << 1, l, mid);
    if (mid < y) Query(o << 1 | 1, mid + 1, r);
}
```

- 区间修改类似

- Q: 为什么不直接查 [2,9]? A: 因为没有 [2,9] 这段区间啊 · · ·
- 易知能够通过访问不超过 2*logN 个线段树上的区间来获得任意区间 [l,r] 的答案

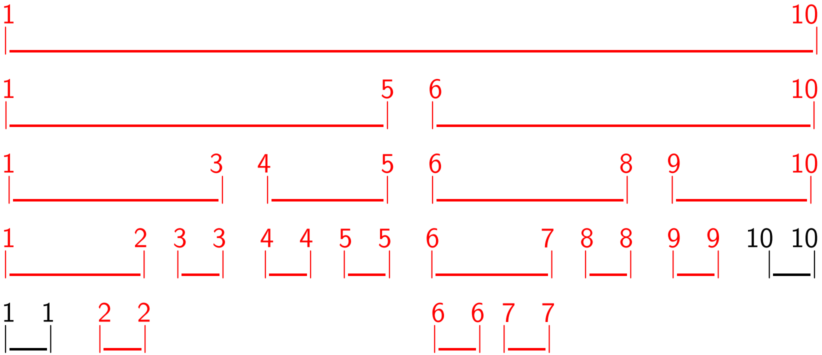
```
void Query(int o, int l, int r) {//A[x..y]
    if (x <= l && r <= y) { ans += sum[o]; return;}
    int mid = l + r >> 1;
    if (x <= mid) Query(o << 1, l, mid);
    if (mid < y) Query(o << 1 | 1, mid + 1, r);
}
```

- 区间修改类似吗?

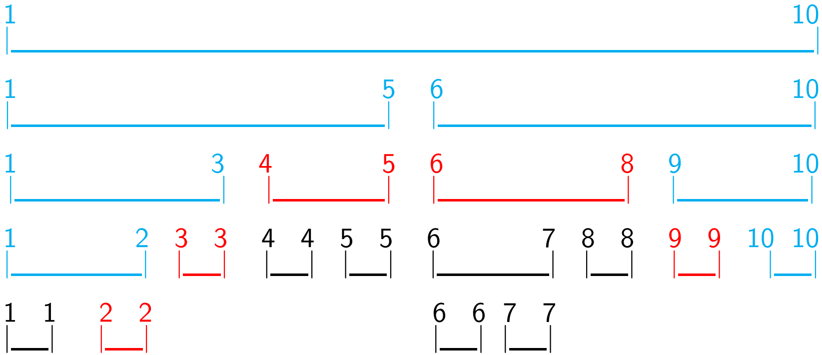
1. Introduction
   - 长啥样?
   - 修改
   - 查询
   - Lazy tag
   - 代码
2. Application

- Lazy-Tag 记录的是每一个线段树节点的变化值
- 当这部分区间的**一致性**被破坏时，就将这个变化值传递给子区间
- 每个节点存一个 Tag 值，表示这个区间进行的变化
- 每当访问到某一个节点时,Tag 下传

如果把 A[2..9] 每个数都 +C, 那么真正要修改的节点大概这么多

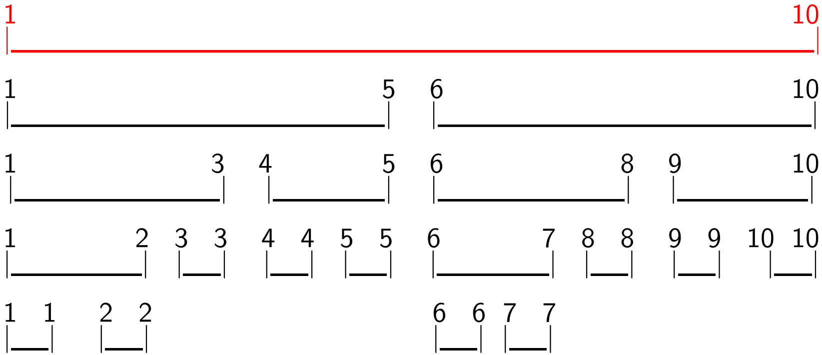


使用 Lazy Tag 以后，情况是这样的:

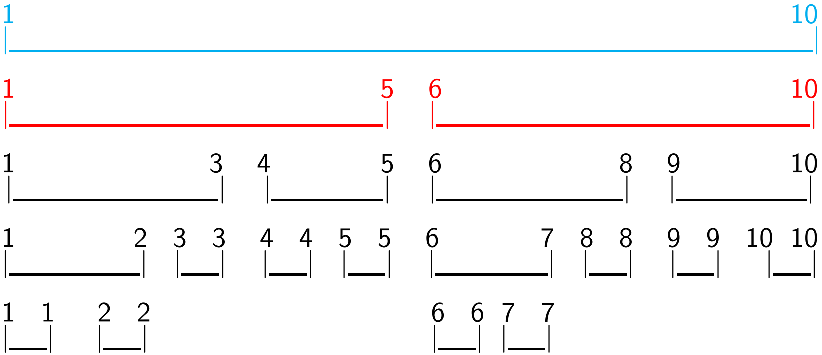


- 蓝色是要把信息及时维护的节点, 红色是本次区间修改操作 Lazy Tag 下传停止的位置.

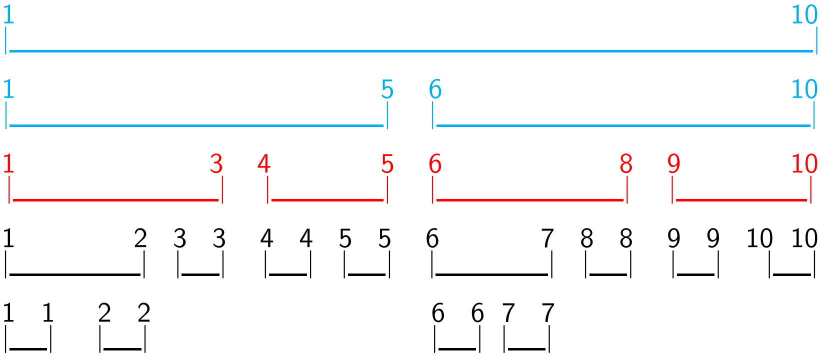
修改 A[2..9]



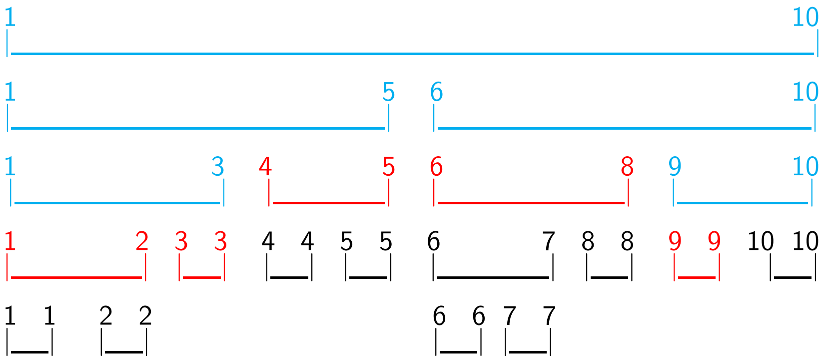
修改 A[2..9]



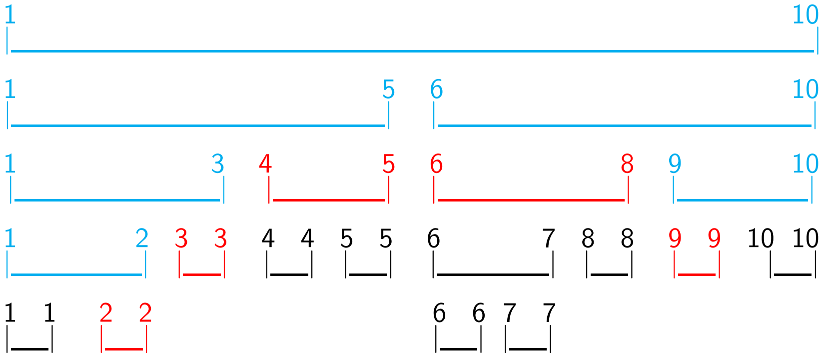
修改 A[2..9]



修改 A[2..9]

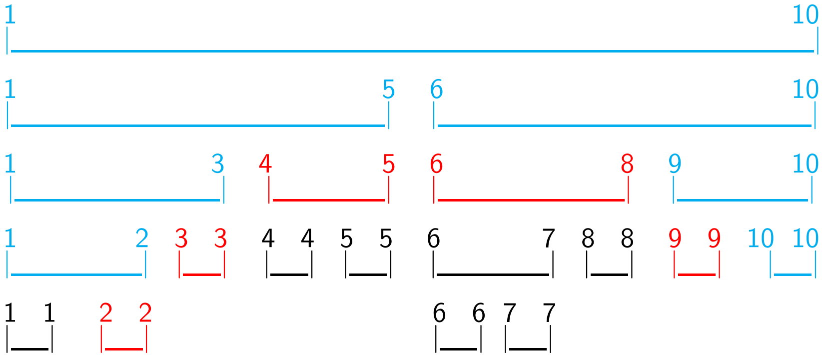


修改 A[2..9]

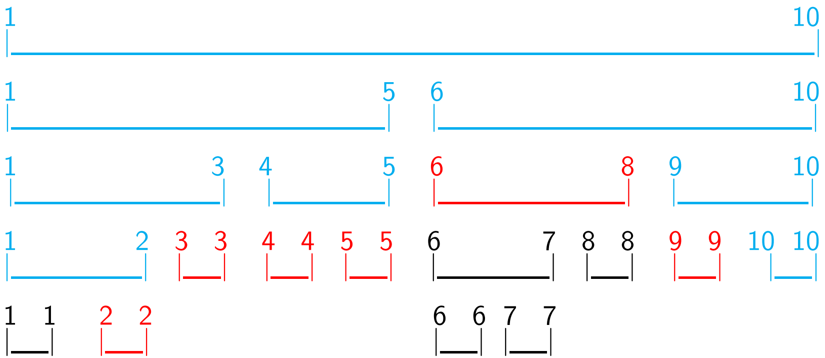


- 由于每一行最多只有两个蓝色区间和两个红色区间, 因此线段树区间修改的自带常数为 4.
- zkw 线段树: 满二叉树, 靠蓝色区间维护上去

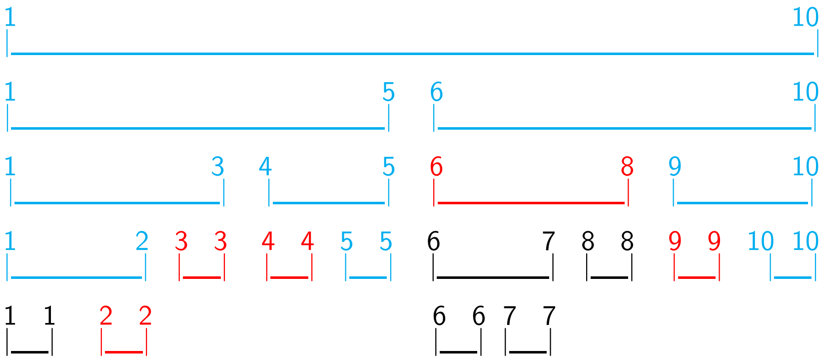
要查询 A[5]



要查询 A[5]



要查询 A[5]



- 多个 Lazy Tag 咋办?
- 考虑打标记运算的优先级, 优先级高的先下传.

1. Introduction
   - 长啥样?
   - 修改
   - 查询
   - Lazy tag
   - 代码
2. Application

- 这里
- 线段树里面每个节点都要记录统计量和 Lazy Tag (修改量)
- 建议写相关的函数都传 3 个参: (int o, int l, int r)
- 网络上说线段数要开 4 倍空间, 实际上如果没写挂的话, 正常只要这样开:
  先把 N 补成 2 的幂次, 再 $\times 2$ 即可

1. Introduction
2. Application
    - 什么时候要用线段树?
    - 例题 1
    - 例题 2
    - 例题 3
    - 例题 4
    - 例题 5
    - 例题 6
    - 例题 7
    - 例题 8
    - 例题 9

- 统计量可合并
- 修改量可合并
- 通过修改量可直接修改统计量

- 统计量可合并
- 修改量可合并
- 通过修改量可直接修改统计量

- 一句话: 满足区间加法即可使用线段树维护信息

接下来的题目中如果 N 和 M 没说明范围，默认 10w

1 Introduction

2 Application

- 什么时候要用线段树?
- 例题 1
- 例题 2
- 例题 3
- 例题 4
- 例题 5
- 例题 6
- 例题 7
- 例题 8
- 例题 9

请优化以下代码:

```
void operation1 (int l, int r, int c) {
    for (int i = l; i <= r; i++) a[i] = (a[i] + c) % p;
}
int operation2 (int l, int r) {
    int cnt = 0;
    for (int i = l; i < r; i++) cnt += a[i] > a[i + 1];
    return cnt;
}
```

Solution

- ~~差分~~
- ~~单点修改，区间查询小于 0 的数的个数~~
- 不会做！求大神指点迷津

- 对于一个序列维护以下操作:
  1. 修改某个 a[i] 的值
  2. 输入 l,r, 选出区间 [l,r] 中的所有 a[i], 问他们两两之差的和是多少, 差的平方和是多少.
- 答案 mod $10^9 + 7$ 输出

Solution

- 第一问答案是 0 hhhh
- 第二问拆式子，发现要维护区间和、区间平方和

- 给出 A[], 要求支持:
  1. 询问 A[l..r] 中最大的数
  2. 删除 A[x], 并且 x+1 以后的元素整体前移
  3. 在末尾增加一个数
- 1853

Solution

- 用 Splay 也是可以的我并没有意见
- 用线段树记录每个位置的数是否存在，存在则标为 1，不存在则标为 0
- 当然相应的 l,r 也要修改
- 转成线段树中第 k 小的数是哪个，在线段树上二分

```
if (k <= sum[o]) Query(o << 1, l, mid);
else k -= sum[o], Query(o << 1 | 1, mid + 1, r);
```

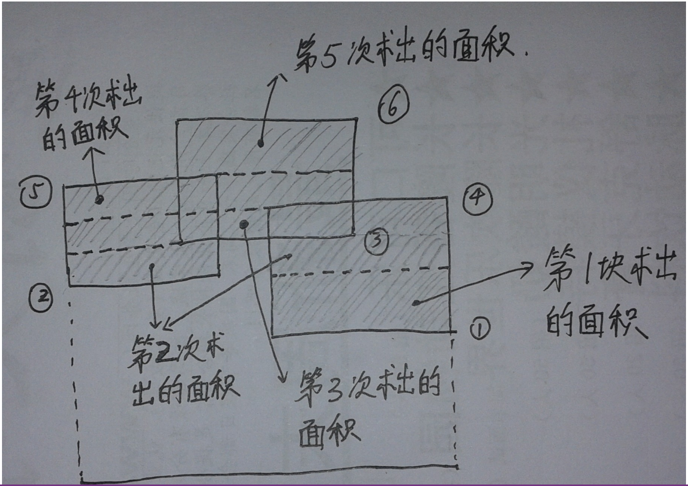
1. Introduction
2. Application
   - 什么时候要用线段树?
   - 例题 1
   - 例题 2
   - 例题 3
   - **例题 4**
   - 例题 5
   - 例题 6
   - 例题 7
   - 例题 8
   - 例题 9

矩形面积并

- 二维平面上给出 N 个矩形，求它们覆盖的总面积
- 1820

Solution

- 扫描线 ~~灰不似窝的图!!! 窝的字木有喇么丑!!!~~



1. Introduction

2. Application

线段树求最大子段和

- 给出 A[], 要求支持:
  1. A[x]=y
  2. 询问 A[x..y] 的最大子段和
- 1647

Solution

- 都是套路.
- 一段答案的存在方式只有三种: 完全被左区间包含 / 完全被右区间包含 / 跨过左右区间分界点
- 只要合并答案就好
- 维护 sum && 从左/右端点开始的最大子段和, 答案可顺便维护

1. Introduction
2. Application
   - 什么时候要用线段树?
   - 例题 1
   - 例题 2
   - 例题 3
   - 例题 4
   - 例题 5
   - **例题 6**
   - 例题 7
   - 例题 8
   - 例题 9

- 对长度为 n 的数列进行 m 次操作, 操作为:
  1. a[l..r] 每一项都加一个常数 C, 其中 $0 \le C \le 10^{11}$
  2. 求 F[a[l]]+F[a[l+1]]+...F[a[r]] mod 10000 的余数
- 其中 F[i] 表示斐波那契数列. 即 F[0]=F[1]=1, F[n+2]=F[n+1]+F[n].

Solution

- 对于每个位置，保存 F[a[i]] 和 F[a[i]+1]
- 矩阵乘法具有分配律，Lazy Tag 只要记录幂次即可
- 预处理出循环节，就不用每次都来矩阵快速幂了

- 对长度为 n 的数列进行 m 次操作, 操作为:
  1. 对 $i \in [l, r]$ 执行: $a[i] = a[i]^2$
  2. 求 $\sum_{i=l}^{r} a[i]$ mod p 的余数, p 在程序开始运行时给出
- 2164

Solution

- 找循环节
- 进了循环节后，打上在循环节整体移动的 Lazy Tag
- 如果还没进，**暴力**递归修改，根据均摊复杂度的那套理论，这一部分复杂度不会超过 $O(NlogN)$

- 有一个 2*n 的点阵，平行于坐标轴的方向上相邻的点之间可以连边，维护以下操作：
  1. 在某相邻两点之间连边
  2. 删除某条边
  3. 询问某两点是否连通
- BZOJ1018

Solution

- 这个……用 LCT 我是没有意见的，要试试能不能过.
- 维护 A[x..x+1][0..1] 四个格子之间的连通性
- 查询 [l,r] 是否联通，注意有可能先掉头后直行

1. Introduction
2. Application
   - 什么时候要用线段树?
   - 例题 1
   - 例题 2
   - 例题 3
   - 例题 4
   - 例题 5
   - 例题 6
   - 例题 7
   - 例题 8
   - **例题 9**

- 求 A[] 的一个最长子序列 B[], 满足 $B_i - B_{i-1} \leq d$, 输出长度即可
- 听说 A 和 d 很大? 2442

Solution

- 说好的 DP: $f[i] = \max\{f[j] | 1 \le j < i,\ |a[j] - a[i]| \le d\} + 1$
- 按照 A[] 排序，线段树中存 f[]，维护区间最大值
- 思路：按顺序枚举右端点，查询答案/左端点