线段树基础

王文斌

天津大学

2023 年 7 月 22 日

目录

1. 线段树简介
2. 线段树入门
3. 区间修改与懒标记
4. 动态开点
5. 线段树上二分
6. 二维偏序

目录

1 线段树简介

2 线段树入门

3 区间修改与懒标记

4 动态开点

5 线段树上二分

6 二维偏序

线段树简介

Description

线段树是一颗二叉树，树上每个点都对应了一个区间，区间一分为二为左右儿子。每个点维护对应区间的信息。

目录

1. 线段树简介
2. 线段树入门
3. 区间修改与懒标记
4. 动态开点
5. 线段树上二分
6. 二维偏序

认识线段树

给出 $a = [1, 9, 2, 6, 0, 8, 1, 7]$, 可以构建出线段树如下:

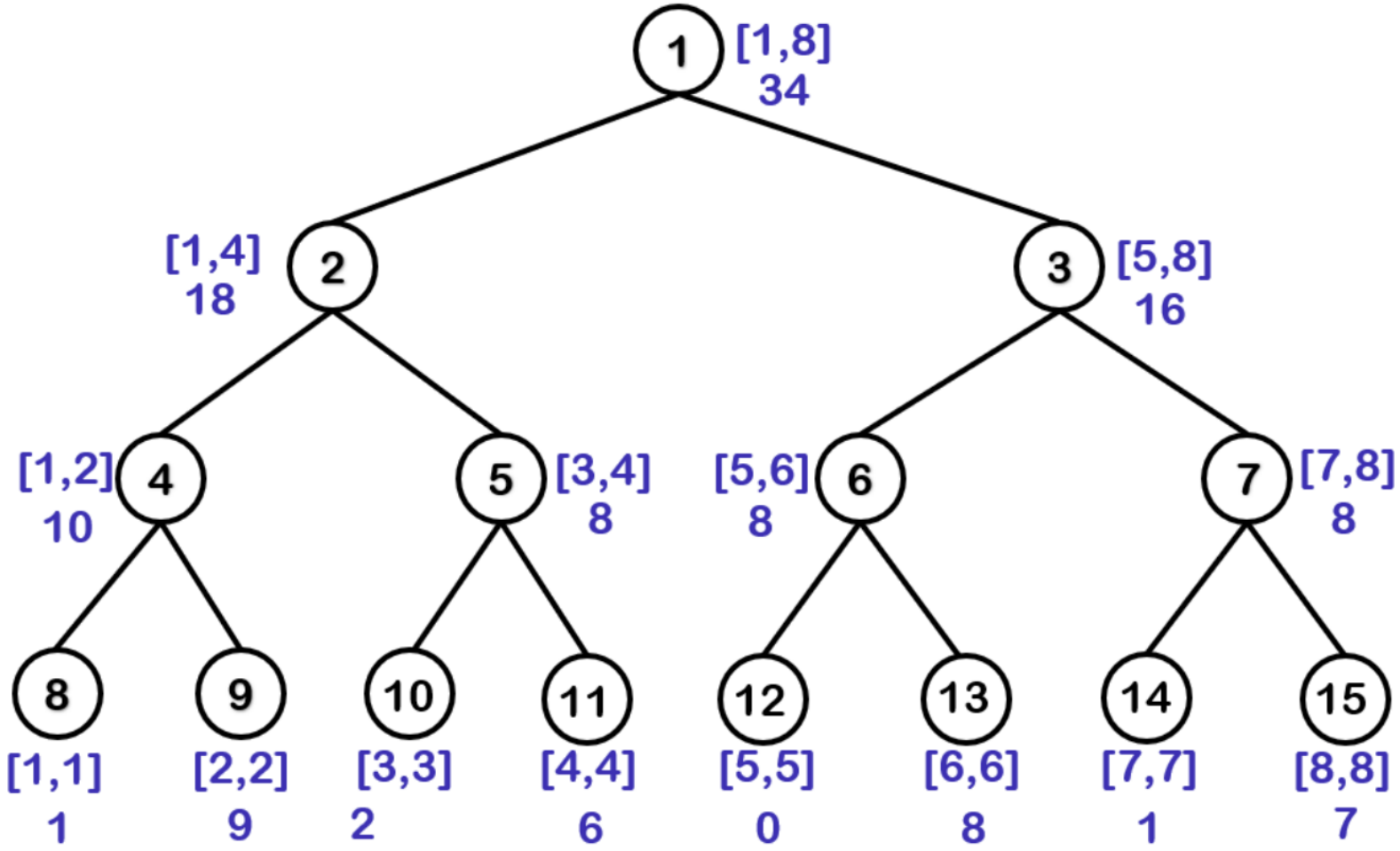

图: 线段树示例

建树操作

Description

我们用递归的方式，从一个数组构建出一颗线段树。建树代码如下：

```
1 void build(int rt, int l, int r){
2     if(l == r){
3         sum[rt] = a[l]; return;
4     }
5     int mid = (r+l)/2;
6     build(rt*2, l, mid);
7     build(rt*2+1, mid+1, r);
8     sum[rt] = sum[rt*2] + sum[rt*2+1];
9 }
```

从一道题开始

Description

n 个数字 $a_1, \ldots a_n$，有 q 次操作，每次操作有 2 种类型：

1. 修改一个位置 a_i 的值为 x
2. 查询区间 $[l, r]$ 的数字和

$1 \leq n, q, a_i, l, r \leq 10^6$,

查询操作

Description

查询区间 $[l, r]$ 的数字和

```
int qry(int rt, int l, int r, int L, int R){
    if(L <= l && r <= R) return sum[rt];
    int mid = (r+l)/2;
    if(R <= mid) return qry(rt*2, l, mid, L, R);
    else if(L > mid) return qry(rt*2+1, mid+1, r, L, R);
    else return qry(rt*2, l, mid, L, R) +
    qry(rt*2+1, mid+1, r, L, R); //信息合并
}
```

单点修改操作

Description

单点修改：修改 $a_i = x$

```
1 void update(int rt, int l, int r, int i, int x){
2     if(l == r){
3         sum[rt] = a[l] = x; return;
4     }
5     int mid = (r+l)/2;
6     if(i <= mid) update(rt*2, l, mid, i, x);
7     else update(rt*2+1, mid+1, r, i, x);
8     sum[rt] = sum[rt*2] + sum[rt*2+1];
9 }
```

单点修改泛化

Description

如果我们查询的信息不是区间和，而是区间最大/最小值，该怎么做？

单点修改泛化

Description

对于这种单点修改，查询区间统计信息的问题，只需要满足区间信息可合并，就可以用线段树来做。

```
#define mid (l+r)/2
#define lson rt*2, l, mid
#define rson rt*2+1, mid+1, r
struct node{
    //some data
}nd[maxn*4];
```

单点修改泛化

```
1  void update(int rt, int l, int r, int i, int x){
2      if(l == r){
3          nd[rt] = ...;//单点修改
4          return;
5      }
6      if(i <= mid) update(lson, i, x);
7      else update(rson, i, x);
8      nd[rt] = merge(nd[rt*2], nd[rt*2+1]);
9  }
10 node qry(int rt, int l, int r, int L, int R){
11     if(L <= l && r <= R) return nd[rt];
12     if(R <= mid) return qry(lson, L, R);
13     else if(L > mid) return qry(rson, L, R);
14     else return merge(qry(lson, L, R), qry(rson, L, R));
15 }
```

单点修改泛化

Description

考虑下列操作哪些可以用线段树维护，哪些不可以？

1. 修改一个元素，查询区间最大值/最小值
2. 修改一个元素，查询区间和的平方 (即 $(a_l + a_{l+1} + \ldots a_r)^2$)
3. 修改一个元素，查询区间最小值的个数
4. 修改一个元素，查询区间最大前缀和
5. 修改一个元素，查询区间最大子段和
6. 修改一个元素，查询区间 gcd
7. 修改一个元素，查询区间众数（如果有多个，输出值最大的那个）
8. 修改一个元素，每个位置都是一个二元组 (a, b) 表示方程 $f_i(x) = a_i * x + b_i$ ，每次查询给出 t，求区间 $[l, r]$ 的方程迭代结果带入 t 的值，即 $f_l(f_{l+1}(f_{l+2}(\ldots f_r(t))))$

例题讲解：区间最大子段和

```
1  struct node{
2      int pre_sum, suf_sum, sum, ans;
3  };
4
5  node merge(node x, node y){
6      node z;
7      z.pre_sum = max(x.pre_sum, x.sum+y.pre_sum);
8      z.suf_sum = max(y.suf_sum, x.suf_sum+y.sum);
9      z.sum = x.sum + y.sum;
10     z.ans = max(x.sum, y.sum, x.suf_sum + y.pre_sum);
11 }
```

目录

1 线段树简介

2 线段树入门

3 区间修改与懒标记

4 动态开点

5 线段树上二分

6 二维偏序

区间修改，区间查询

Description

我们经常会遇到一类问题，每次修改是对区间进行修改，比如：
n 个数字 $a_1, \ldots a_n$，有 q 次操作，每次操作有 2 种类型：
1. 对一个区间 $[L, R]$ 整体加上一个值 x
2. 查询区间 $[l, r]$ 的数字和

$1 \leq n, q, a_i, l, r, x \leq 10^6$,

懒标记

Description

如果我们按照单点修改的思路去做，每次区间修改的复杂度高达 $O(nlog(n))$.
一个区间只需要 log 个节点就可以表达，我们可以利用这个性质，给大区间打上标记。

懒标记

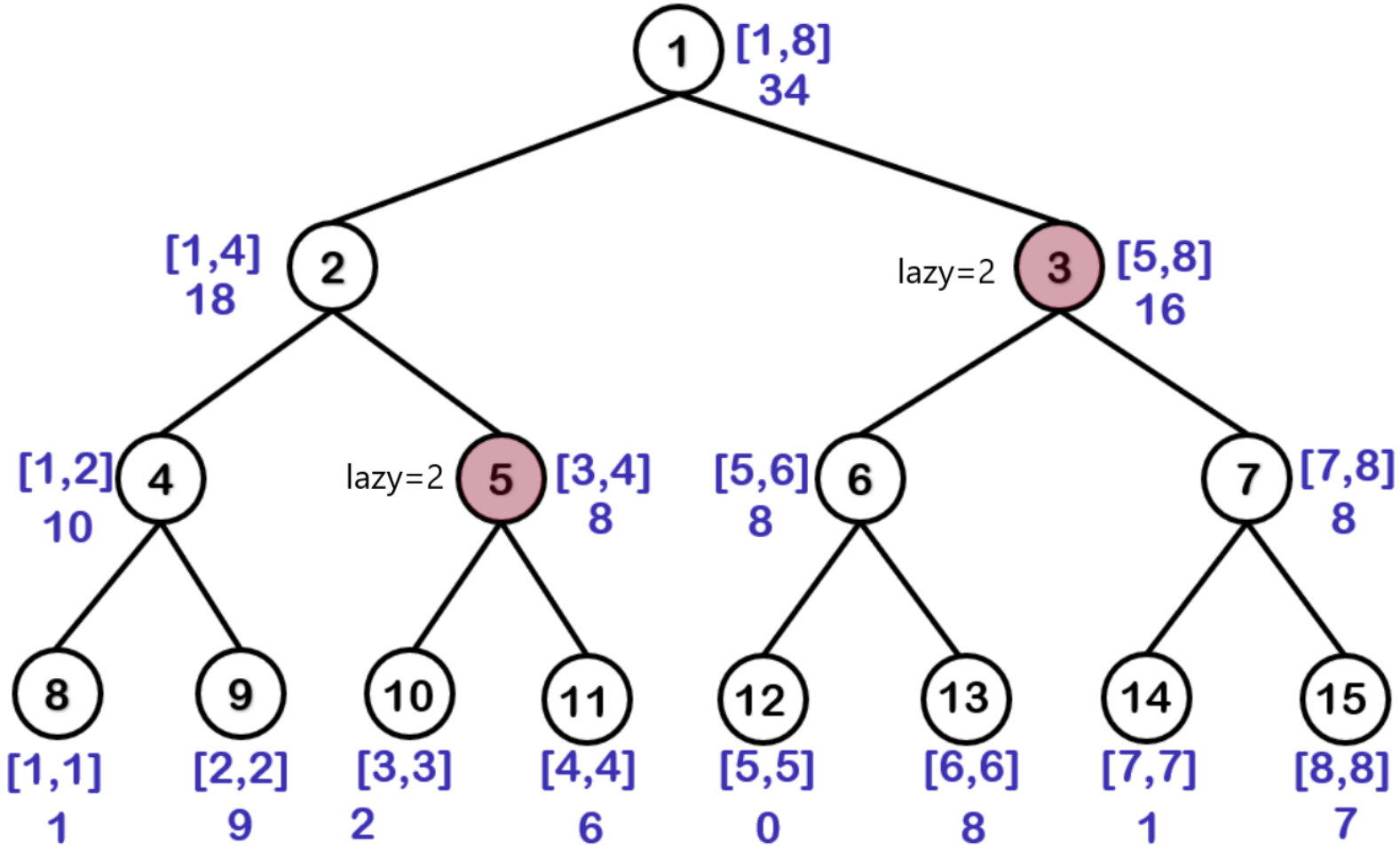
1
[1,8]
34
[1,4]
18
2
lazy=2
3
[5,8]
16
[1,2]
10
4
lazy=2
5
[3,4]
8
[5,6]
8
6
7
[7,8]
8
8
9
10
11
12
13
14
15
[1,1]
1
[2,2]
9
[3,3]
2
[4,4]
6
[5,5]
0
[6,6]
8
[7,7]
1
[8,8]
7

图: 懒标记示例

懒标记

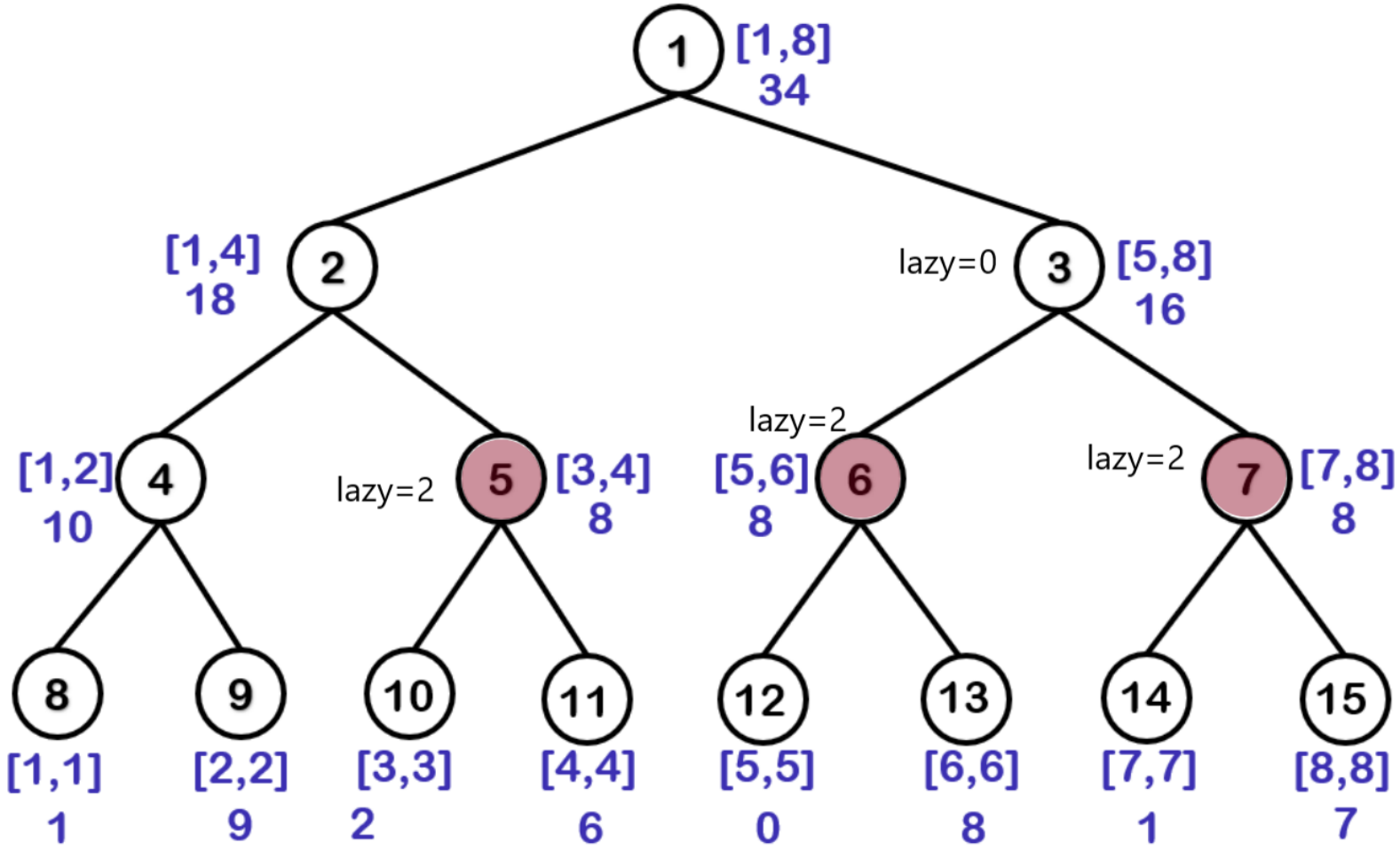
1
[1,8]
34
[1,4]
18
2
lazy=0
3
[5,8]
16
lazy=2
[1,2]
10
4
lazy=2
5
[3,4]
8
[5,6]
8
6
lazy=2
7
[7,8]
8
8
9
10
11
12
13
14
15
[1,1]
1
[2,2]
9
[3,3]
2
[4,4]
6
[5,5]
0
[6,6]
8
[7,7]
1
[8,8]
7

图: 懒标记示例

懒标记

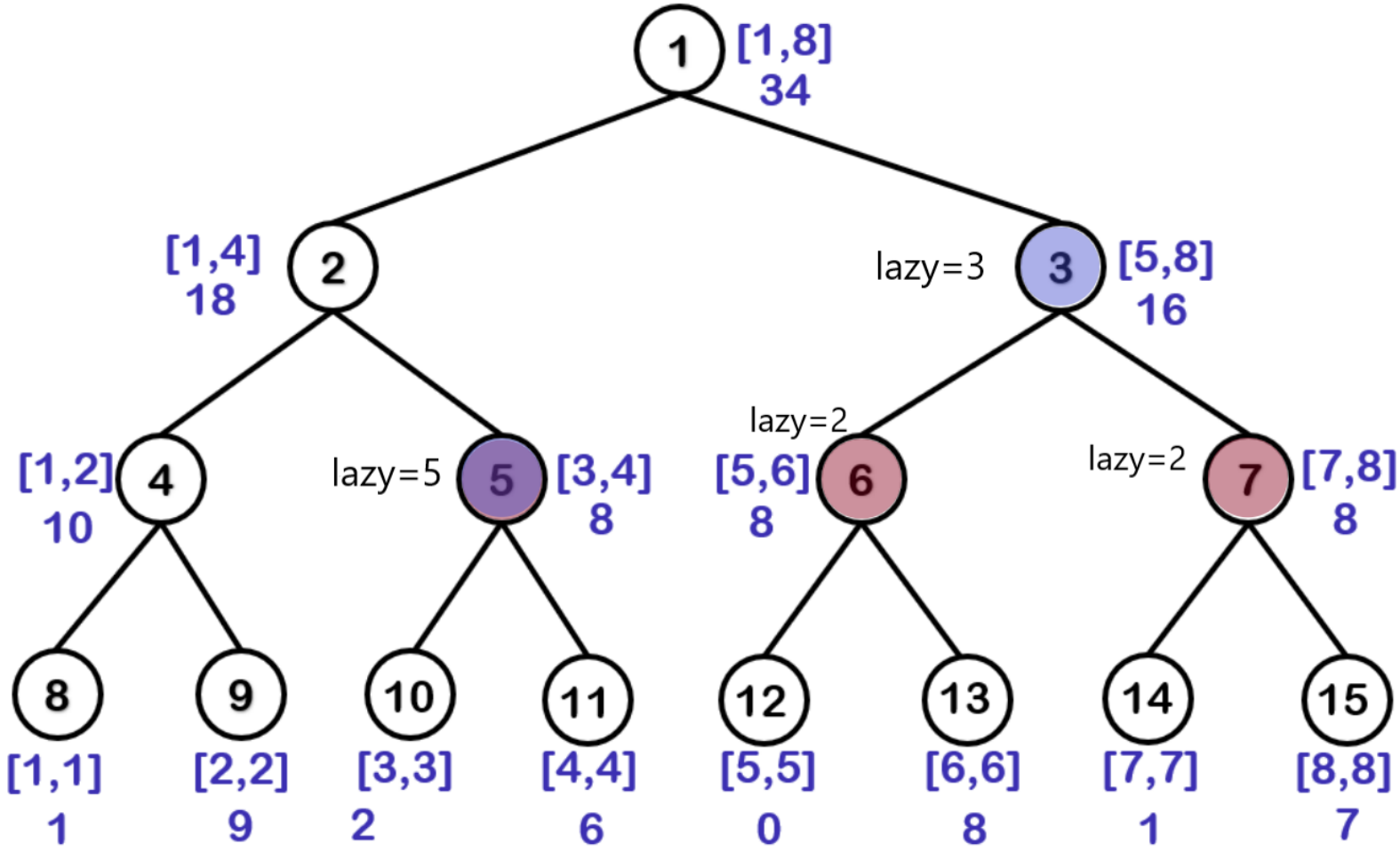
1
[1,8]
34
[1,4]
18
2
lazy=3
3
[5,8]
16
[1,2]
10
4
lazy=5
5
[3,4]
8
lazy=2
[5,6]
8
6
lazy=2
7
[7,8]
8
8
9
10
11
12
13
14
15
[1,1]
1
[2,2]
9
[3,3]
2
[4,4]
6
[5,5]
0
[6,6]
8
[7,7]
1
[8,8]
7

图: 懒标记示例

代码示例

```
1  void update(int rt, int l, int r, int L, int R, int x){
2      if(L <= l && r <= R){
3          put_tag(rt, l, r, x); //新东西
4          return;
5      }
6      push_down(rt, l, r); //新东西
7      if(i <= mid) update(lson, i, x);
8      else update(rson, i, x);
9      nd[rt] = merge(nd[rt*2], nd[rt*2+1]);
10 }
11 node qry(int rt, int l, int r, int L, int R){
12     if(L <= l && r <= R) return nd[rt];
13     push_down(rt, l, r); //新东西
14     if(R <= mid) return qry(lson, L, R);
15     else if(L > mid) return qry(rson, L, R);
16     else return merge(qry(lson, L, R), qry(rson, L, R));
17 }
```

代码示例

```
1 void put_tag(int rt, int l, int r, int x){
2     nd[rt].sum += (r-l+1) * x;
3     nd[rt].lazy += x;
4 }
5 void push_down(int rt, int l, int r){
6     put_tag(lson, nd[rt].lazy);
7     put_tag(rson, nd[rt].lazy);
8     nd[rt].lazy = 0;
9 }
```

区间修改泛化

Description

如果我们要做的不是区间加，而是区间乘，能做吗？

区间修改泛化

Description

对于这种**区间修改**，查询区间统计信息的问题，想用线段树做，需要满足以下条件：1. 标记信息可叠加 2. 当对区间叠加懒标记的时候，同时也可以很方便的更新需要维护的信息 (例如区间和)

区间修改泛化

Description

考虑下列操作哪些可以用线段树维护，哪些不可以？

1. 需要进行区间赋值，查询区间求和
2. 需要进行区间加，查询区间最小值
3. 需要进行区间赋值，查询区间最小值
4. 需要进行区间加，查询区间每个数字平方的和，即：$a_l^2 + a_{l+1}^2 \ldots + a_r^2$
5. 需要进行区间对 x 取 min，查询区间最小值
6. 需要进行区间对 x 取 min，查询区间和
7. 需要进行区间赋值，查询区间 gcd
8. 需要区间对 x 取 gcd（即 $a[i] = gcd(a[i], x)$，求区间和

区间修改泛化

Description

4. 需要进行区间加，查询区间每个数字平方的和，即：$a_l^2 + a_{l+1}^2 ... + a_r^2$

区间修改泛化

Description

4. 需要进行区间加，查询区间每个数字平方的和，即：$a_l^2 + a_{l+1}^2 \ldots + a_r^2$

```
1  struct node{ ll sum2, sum, lazy; }nd[maxn*4];
2  void put_tag(int rt, int l, int r, int x){
3      nd[rt].sum2 += 2*x*nd[rt].sum + x*x*(r-l+1);
4      nd[rt].sum += (r-l+1) * x;
5      nd[rt].lazy += x;
6  }
7  void push_down(int rt, int l, int r){
8      put_tag(lson, nd[rt].lazy);
9      put_tag(rson, nd[rt].lazy);
10     nd[rt].lazy = 0;
11 }
```

区间修改泛化

Description

5. 需要进行区间对 x 取 min，查询区间最小值

区间修改泛化

Description

5. 需要进行区间对 x 取 min，查询区间最小值

```
1  struct node{ ll min, lazy; }nd[maxn*4];//注：lazy初始化为
       inf，即伪无穷大
2  void put_tag(int rt, int l, int r, int x){
3      nd[rt].min = min(nd[rt].min, x);
4      nd[rt].lazy = min(nd[rt].lazy, x);
5  }
6  void push_down(int rt, int l, int r){
7      put_tag(lson, nd[rt].lazy);
8      put_tag(rson, nd[rt].lazy);
9      nd[rt].lazy = inf;
10 }
```

区间修改泛化

Description

6. 需要进行区间对 x 取 $\min$，查询区间和

区间修改泛化

Description

7. 需要进行区间赋值，查询区间 gcd

区间修改泛化

Description

7. 需要进行区间赋值，查询区间 gcd

```
1  struct node{ ll gcd, lazy; }nd[maxn*4];//注：lazy=nothing
       表示无赋值操作，可以用-1或者-114514等特殊值来表达
       nothing
2  void put_tag(int rt, int l, int r, int x){
3      nd[rt].gcd = x;
4      nd[rt].lazy = x;
5  }
6  void push_down(int rt, int l, int r){
7      put_tag(lson, nd[rt].lazy);
8      put_tag(rson, nd[rt].lazy);
9      nd[rt].lazy = nothing;
10 }
```

区间修改泛化

Description

8. 需要区间对 x 取 gcd（即 $a[i] = gcd(a[i], x)$，求区间和

区间修改泛化

Description

9. 需要区间加，求区间最大子段和

区间修改泛化

Description

10. 需要对区间进行取相反数，求区间和

区间修改泛化

Description

10. 需要对区间进行取相反数，求区间和

```
1  struct node{ ll sum, lazy; }nd[maxn*4];
2  void put_tag(int rt, int l, int r, int x){//x = 1 or -1
3      nd[rt].sum *= x;
4      nd[rt].lazy *= x;
5  }
6  void push_down(int rt, int l, int r){
7      put_tag(lson, nd[rt].lazy);
8      put_tag(rson, nd[rt].lazy);
9      nd[rt].lazy = 1;
10 }
```

再探懒标记的叠加性

Description

当修改操作有多种不同类型，例如：
n 个数字 $a_1, \ldots a_n$，有 q 次操作，每次操作有 3 种类型：
1. [L,R] 每个位置的值 +x
2. [L,R] 每个位置的值 *x
3. 查询区间 $[l, r]$ 的数字和% 1000000007 的结果

$1 \leq n, q, a_i, l, r \leq 10^6$,

例题讲解

Description

维护多个懒标记再分类讨论进行处理，麻烦，通用性差。
把懒标记看成一个**整体**的二元组 (a, b)
表示这个区间所有数字 $x = a * x + b$
此时考虑两个懒标记 $(a_1, b_1), (a_2, b_2)$ 叠加：
x -> $a_1 * x + b_1$ -> $a_1 * a_2 * x + b_1 * a_2 + b_2$
即我们可以用 $(a_1 * a_2, b_1 * a_2 + b_2)$ 来表达 $(a_1, b_1), (a_2, b_2)$ 这两个标记叠加的结果。
操作 1 可以用 $(1, x)$ 来表达
操作 2 可以用 $(x, 0)$ 来表达

例题讲解

```
1  struct lazyTag{
2      ll a, b;
3      lazyTag operator + (lazyTag x){
4          return (lazyTag){a * x.a, b*x.a + x.b};
5      }
6  }
7  struct node{ ll sum; lazyTag lz; }nd[maxn*4];
8  void put_tag(int rt, int l, int r, lazyTag lz){
9      nd[rt].sum = nd[rt].sum * lz.a + lz.b*(r-l+1);
10     nd[rt].lz = nd[rt].lz + lz
11 }
12 void push_down(int rt, int l, int r){
13     put_tag(lson, nd[rt].lz);
14     put_tag(rson, nd[rt].lz);
15     nd[rt].lazy = (lazyTag){1, 0};
16 }
```

区间修改泛化

Description

11. 给 01 数列，区间赋值/翻转，求区间最长的连续的 1 长度

区间修改泛化

Description

11. 给 01 数列，区间赋值/翻转，求区间最长的连续的 1 长度

```
1  struct lazyTag{ //表示先进行a赋值，再进行b翻转
2      ll a, b; //a!=nothing表示赋值，b=1表示整体翻转
3      lazyTag operator + (lazyTag x){ //一些讨论}
4  }
5  struct node{ ll pre[2], suf[2], ans[2]; lazyTag lz; };
6  void put_tag(int rt, int l, int r, lazyTag lz){
7      if (lz.a != nothing)
8          nd[rt].pre[lz.a] = ... = r-l+1,
9          nd[rt].pre[lz.a^1] = ... = 0;
10     if (lz.b != 0) swap(nd[rt].pre[0],nd[rt].pre[1]),...;
11     nd[rt].lz = nd[rt].lz + lz
12 }
13 void push_down(int rt, int l, int r){
14     put_tag(lson, nd[rt].lz), put_tag(rson, nd[rt].lz);
15     nd[rt].lazy = (lazyTag){nothing, 0};
16 }
```

目录

1 线段树简介

2 线段树入门

3 区间修改与懒标记

4 动态开点

5 线段树上二分

6 二维偏序

Sample Problem

Description

有一类问题，无法进行常规的建树
n 个位置 $a_1, \ldots a_n$，一开始全为 0，有 q 次操作，每次操作有 2 种类型：
1. 对一个区间 $[L, R]$ 整体加上一个值 x
2. 查询区间 $[l, r]$ 的数字和

强制在线
$1 \leq q, a_i, l, r, x \leq 10^6, 1 \leq n \leq 1e18$

代码示例

Description

当需要遍历的时候才开辟空间，每次操作最多增加 log 级别的空间

```
1  struct node{
2      int ls, rs;
3      data...;
4  }nd[maxn]
5  int new_node(){
6      init(nd[++tot]); return tot;//对新开辟的node初始化
7  }
8  void update(int &rt, int l, int r, int L, int R, lz){
9      if(!rt) rt = new_node();
10     if(L <= l && r <= R){put_tag(rt,l,r, lz); return;}
11     push_down(rt, l, r); //懒标记下传
12     if(L <= mid) update(nd[rt].ls, l, mid, L, R);
13     if(R > mid) update(nd[rt].rs, mid+1, r, L, R);
14     push_up(rt, l, r);//合并两个子树信息
15 }
```

代码示例

```
node qry(int rt, int l, int r, int L, int R){
    if(L <= l && r <= R){return nd[rt];}
    push_down(rt, l, r); //懒标记下传
    if(R<=mid) return qry(nd[rt].ls, l, mid, L, R);
    else if(L>mid) return qry(nd[rt].rs, mid+1, r, L, R);
    return merge(qry(nd[rt].ls, l, mid, L, R),
    qry(nd[rt].rs, mid+1, r, L, R));
}
void push_down(int rt, int l, int r){
    if(!rt) return;
    if(!nd[rt].ls) nd[rt].ls = new_node()
    put_tag(nd[rt].ls, l, mid, nd[rt].lz);
    if(nd[rt].rs) nd[rt].rs = new_node();
    put_tag(nd[rt].rs, mid+1, r, nd[rt].lz);
    nd[rt].lz = nothing;
}
```

目录

1. 线段树简介
2. 线段树入门
3. 区间修改与懒标记
4. 动态开点
5. 线段树上二分
6. 二维偏序

概念

Description

一个区间在一个线段树上表现为 log 个区间，当我们查询的信息在线段树上满足单调性的时候，我们可以利用线段树的结构做到 log(n) 复杂度的查询

从一道题开始

Description

你有一个集合，一开始为空，q 次查询，每次查询有两种操作：
1. 往集合里面加入一个数字 2. 查询集合中第 k 大的数字 (保证集合中至少有 k 个数字)
$q, k \leq 1e61e6, x <= 1e6$

二分

Description

太好啦，我会二分！
相当于有一个长度 1e6 的数组 a，a_i 表示数字 i 的个数。二分答案！
复杂度？

代码示例

Description

直接在线段树上跑查询，每次判断左子树的 size 来决定往哪边走，直到走到叶子即为答案。

```
1 int sum[maxn*4];
2 int qry(int rt, int l, int r, int k){
3     if(l == r){return l;}
4     if(sum[rt*2] >= k) return qry(lson, k);
5     else return qry(rson, k-sum[rt*2]);
6 }
```

区间内二分

Description

你有一个集合，一开始为空，q 次查询，每次查询有两种操作：
1. 往集合里面加入一个数字 2. 查询集合中值域为 [L,R] 的数字中，第 k 大的数字 (保证答案存在)
$q, k \leq 1e6, x <= 1e6$

代码示例

Description

落在区间内的时候再判断

```
1  int sum[maxn*4];
2  int qry(int rt, int l, int r, int L, int R, int &k){
3      if(l == r){return l;}
4      if(L <= l && r <= R){
5          //消去区间,返回
6          if(sum[rt] < k){k -= sum[rt]; return -1;}
7      }
8      int ans = -1, mid = (r+l)/2;
9      if(L <= mid) ans = qry(lson, L, R, k);
10     if(ans != -1) return;
11     if(R > mid) ans = qry(rson, L, R, k);
12     return ans;
13 }
```

例题演示

Description

给数组 a, q 次查询，每次查找给出 l,r,x, 求区间 [l,r] 中最左边的 >= x 的位置，支持区间加。$a_i, q, l, r, x \leq 1e6$

例题演示

Description

给数组 a，q 次查询，每次查找给出 l,r，求区间 [l,r] 的最小值第一次出现的位置，支持区间加

目录

1 线段树简介

2 线段树入门

3 区间修改与懒标记

4 动态开点

5 线段树上二分

6 二维偏序

概念

Description

抽象：一个对象有两个属性 (a_i, b_i)，每个查询 (x, y)，查询两个属性分别满足偏序关系的对象集合信息。
具体：2D 平面上 n 个点 (a_i, b_i)，每次查询给出 (x, y)，求有多少点在查询点的左下角。
（进一步，查询给出一个矩阵，求有多少个点在矩阵内）

解题思路

Description

一般解决方法：

1. 离线，利用 sort 让一维默认满足，再用数据结构解决另一维。
2. 强制在线？树套树！

解题思路

Description

对所有点（平面点和查询点）全部按照第一维排序，然后从左向右扫。
当遇到一个平面点 (x,y)，让 a[y] += 1
当遇到一个查询点 (x,y)，相当于查询 a[0]+a[1]...+a[y] 的总和
单点修改，区间查询。

一些典中典

Description

线段覆盖问题：
给定若干个线段 $[l_i, r_i]$，每次查询给出 $[L, R]$，查询有多少线段被给定的区间完全覆盖/部分覆盖。

一些典中典

Description

求逆序对个数。
二维偏序转化：每个 a[i] 可以视为二元组 (i, a[i])，相当于对于每个 (i, a[i]) 查询有多少满足 j<i 且 a[j]>a[i] 的二元组 (j,a[j])
再进一步：如果对所有逆序对 (a[j], a[i])，求 a[j]+a[i] 的总和呢？

一些典中典

Description

求最长上升子序列。
二维偏序转化：
考虑优化 dp，对于二元组 (i, a[i])，所有满足 j < i 且 a[j] < a[i] 的 j 都可以转移到 dp[i]。因此当我们想转移得到 dp[i] 的时候，我们实际上想得到的信息为：
所有满足 j < i 且 a[j] < a[i] 的 j 中，dp[j] 的最大值。
再进一步：如果我们要求最长上升子序列相邻的两个数字差值不能超过某个给定的常数呢？

例题

Description

$n(\leq 1e5)$ 个数的排列 $a_1..a_n$, q 次查询 $[L, R]$ 问这个区间里有多少对 (i, j) 满足 $a_j\%a_i == 0$

例题

Description

给两个排列 $a_1 \ldots a_n, b_1, \ldots b_n$，$q$ 次询问 $[L1, R1], [L2, R2]$，问 $\{a_{L1}, \ldots a_{R1}\}$ 和 $\{b_{L2}, \ldots b_{R2}\}$ 的交集大小。