# C++算法

# 线段树

Segment Tree

# 快快编程667

| 动态 | 段增加+点查询 |
| --- | --- |

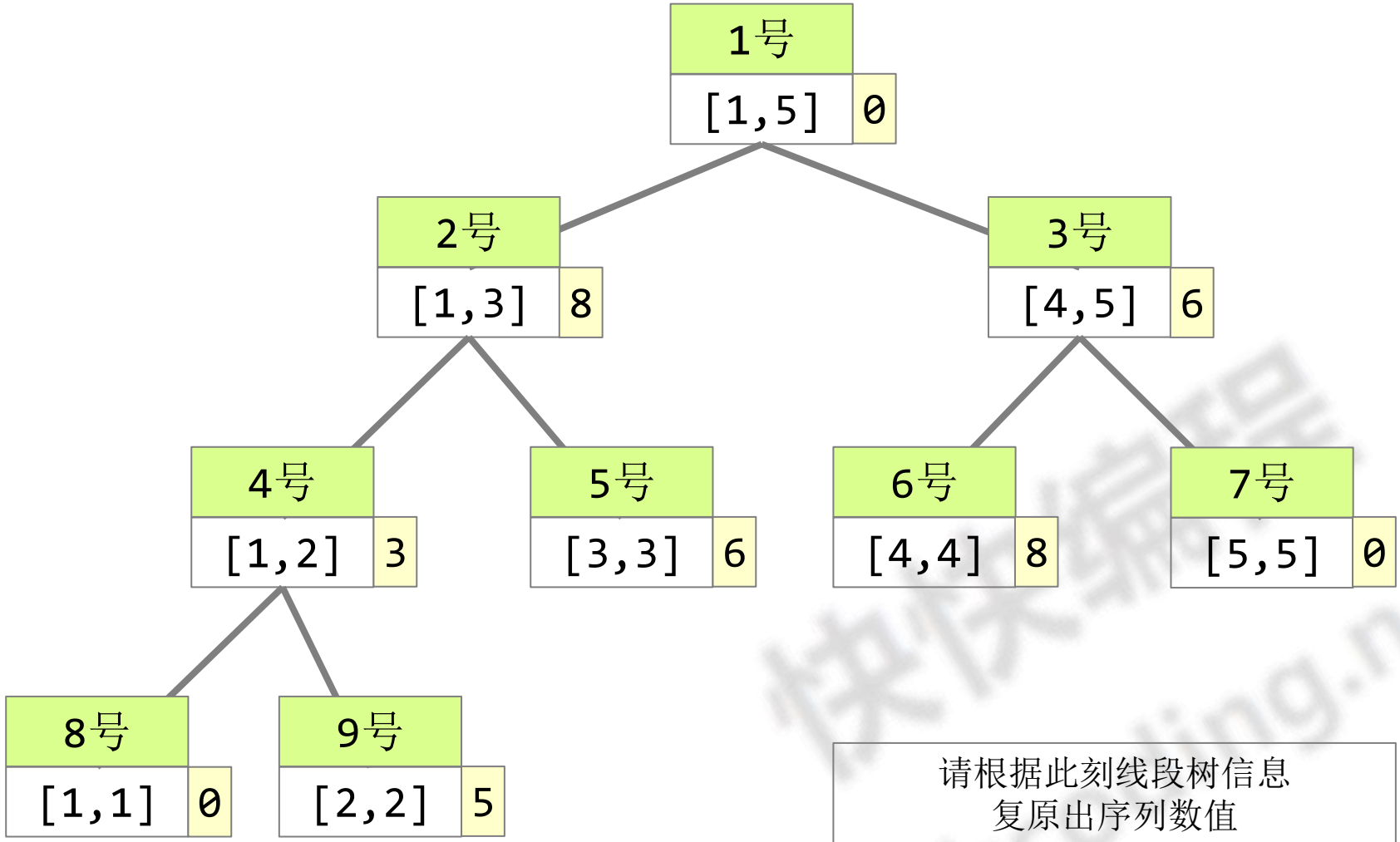
```
6  struct Segment{
7      ll l;
8      ll r;
9      ll toAdd;
10 };
11 Segment tr[N*4];
```

tr[u].toAdd表示u为根子树的
延时增量
lazy tag

为什么不需要.sum?

延时标记
lazy tag

| i=1 | i=2 | i=3 | i=4 | i=5 |
|---|---|---|---|---|
| 11 | 16 | 14 | 14 | 6 |




请根据此刻线段树信息
复原出序列数值

根到叶路径总和

```
19 void add(ll u,ll&l,ll&r,ll&delta){
20     if(r<tr[u].l||tr[u].r<l)return;
21     if(l<=tr[u].l&&tr[u].r<=r){
22
23
24     }
25
26
27 }
```

根到叶路径总和

```
28 ll value(ll u,ll&id){
29     if(tr[u].l==tr[u].r)
30         return tr[u].toAdd;
31     if(id<=tr[u*2].r)
32         return
33     else
34         return
35 }
```

# 讨论

对于序列的区间修改
如果不用延时标记
直接修改线段树整棵子树可以吗?

单次修改复杂度退化为O(n)

单次修改用延时标记复杂度多少?

O(logn)

如何证明这个性质?

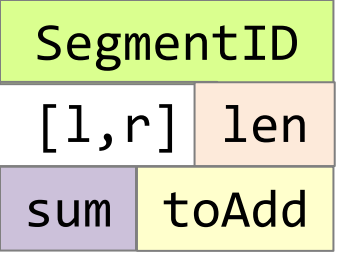
# 快快编程663

| 动态 | 段增加+RSQ |
| --- | --- |

```
6  struct Segment{
7      ll l;
8      ll r;
9      ll len;
10     ll sum;
11     ll toAdd;
12 };
13 Segment tr[N*4];
```

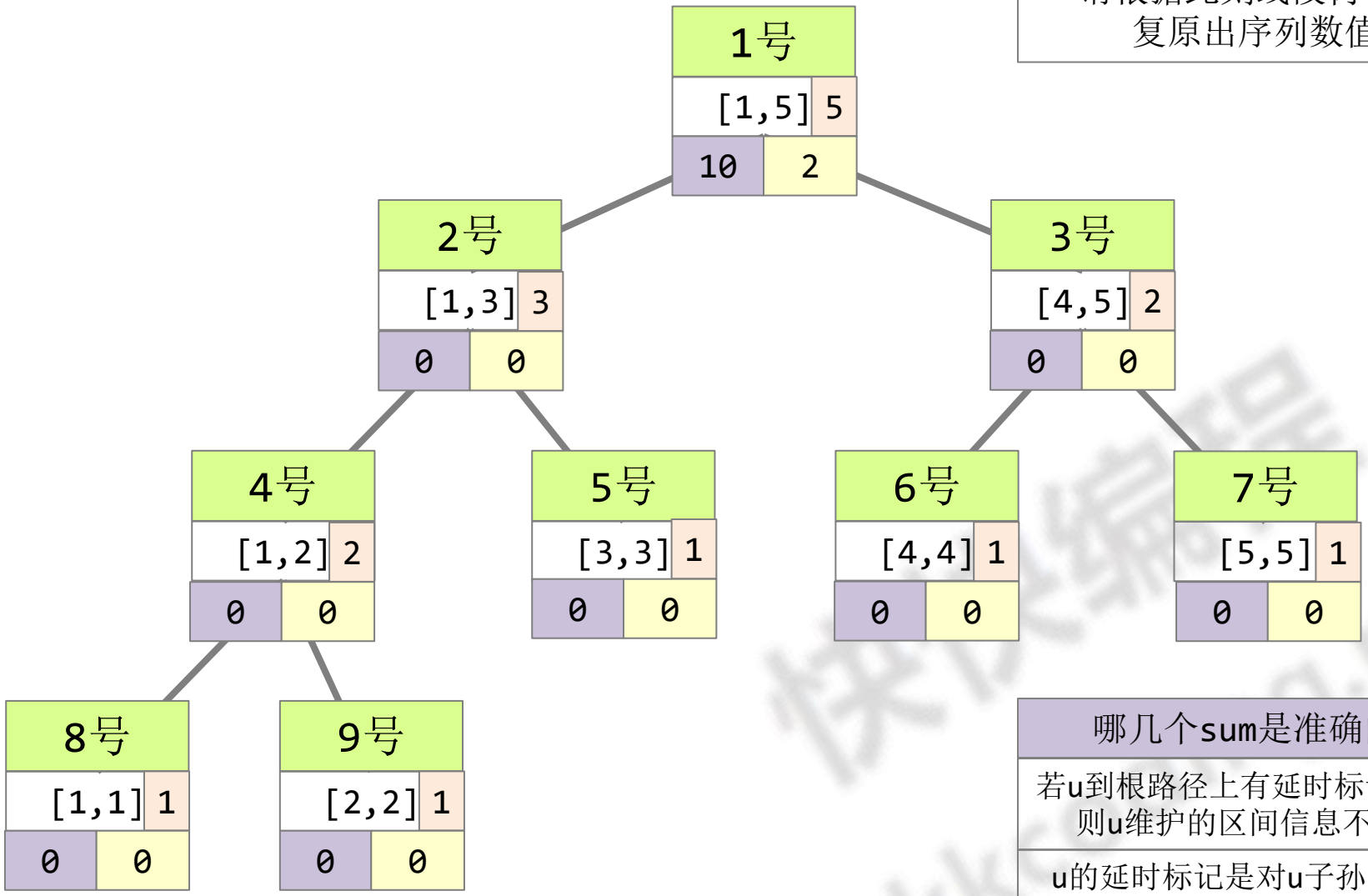
请设计Segment类型

为什么需要.len?




| i=1 | i=2 | i=3 | i=4 | i=5 |
|---|---|---|---|---|
| 2 | 2 | 2 | 2 | 2 |

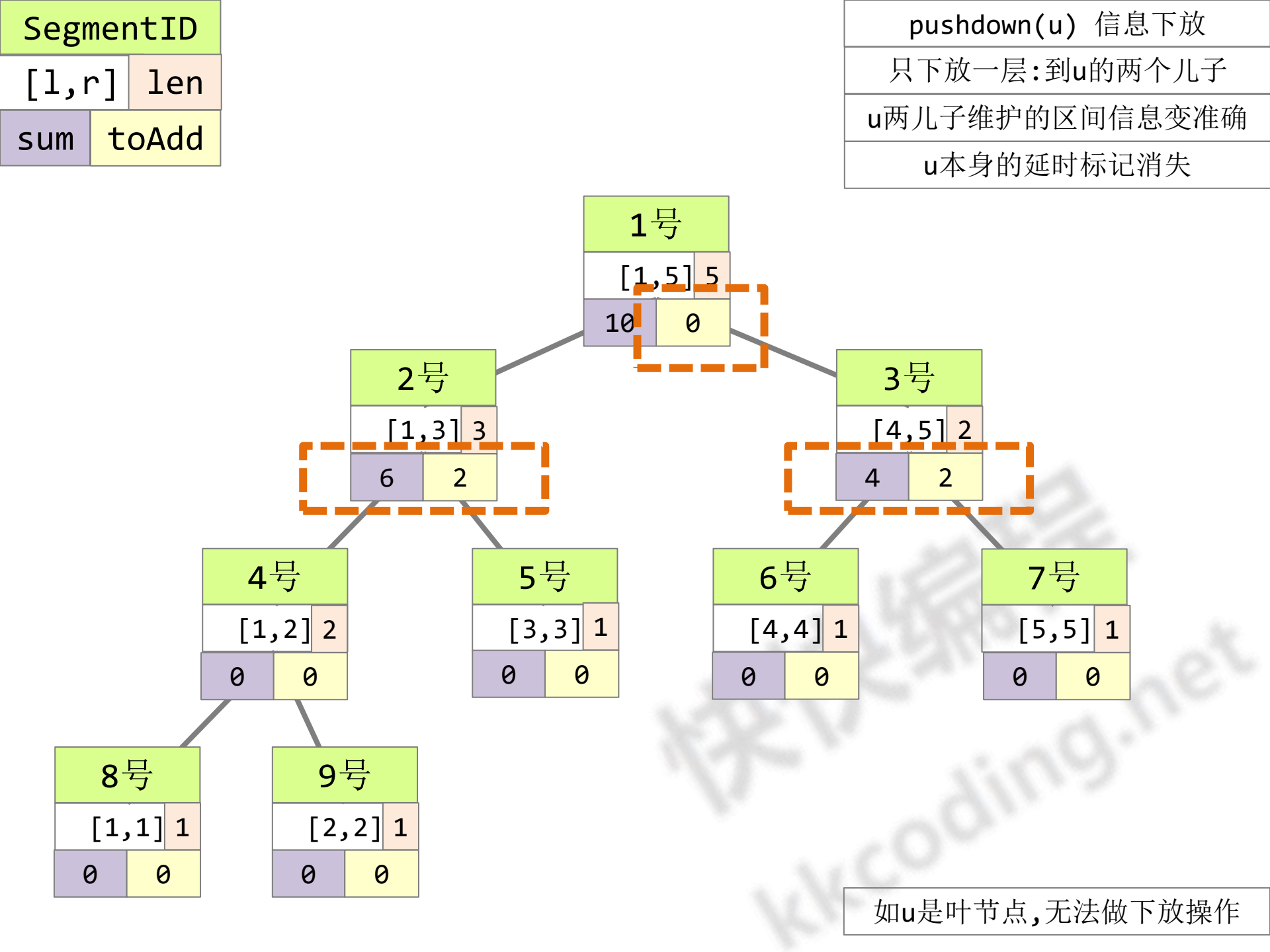
请根据此刻线段树信息
复原出序列数值




哪几个sum是准确的?

若u到根路径上有延时标记不在u
则u维护的区间信息不准确

u的延时标记是对u子孙的延时
对u本身不延时


SegmentID
[l,r]
len
sum
toAdd
pushdown(u) 信息下放
只下放一层:到u的两个儿子
u两儿子维护的区间信息变准确
u本身的延时标记消失
1号
[1,5] 5
10
0
2号
[1,3] 3
6
2
3号
[4,5] 2
4
2
4号
[1,2] 2
0
0
5号
[3,3] 1
0
0
6号
[4,4] 1
0
0
7号
[5,5] 1
0
0
8号
[1,1] 1
0
0
9号
[2,2] 1
0
0
如u是叶节点,无法做下放操作

| pushdown(u) 信息下放 |
| --- |
| 只下放一层:到u的两个儿子 |
| u两儿子维护的区间信息变准确 |
| u本身的延时标记消失 |

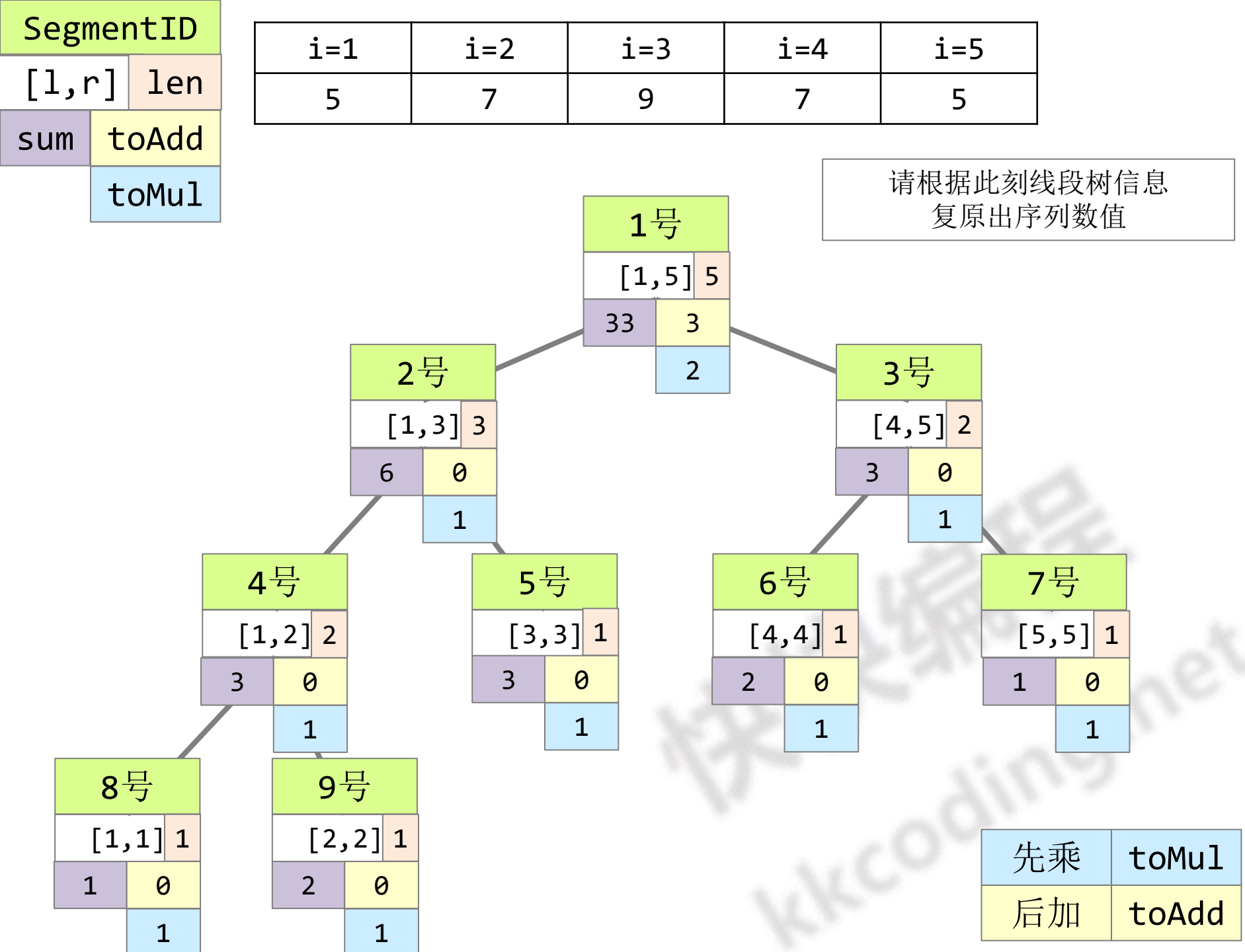
```
21 void pushdown(ll u){
22     if(tr[u].l==tr[u].r)return;
23     ll A=tr[u].toAdd;
24     tr[u].toAdd=0;
25     (tr[u*2].toAdd+=A)%=MOD;
26
27
28
29 }
```

# 快快编程172

| 动态 | 区间整体加/乘+RSQ |
| --- | --- |

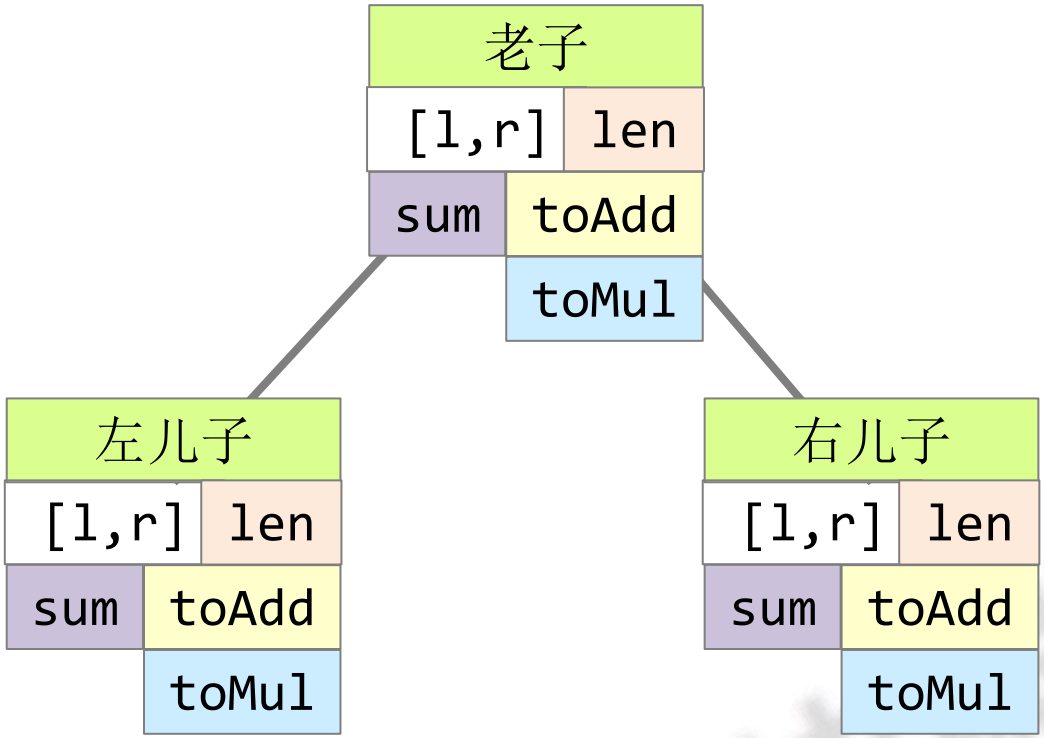
```
 6 struct Segment{
 7     ll l;
 8     ll r;
 9     ll len;
10     ll sum;
11     ll toAdd;
12     ll toMul;
13 };
14 Segment tr[N*4];
```

请设计Segment类型


SegmentID
[l,r]
len
sum
toAdd
toMul
i=1
i=2
i=3
i=4
i=5
5
7
9
7
5
请根据此刻线段树信息
复原出序列数值
1号
[1,5]
5
33
3
2
2号
[1,3]
3
6
0
1
3号
[4,5]
2
3
0
1
4号
[1,2]
2
3
0
1
5号
[3,3]
1
3
0
1
6号
[4,4]
1
2
0
1
7号
[5,5]
1
1
0
1
8号
[1,1]
1
1
0
1
9号
[2,2]
1
2
0
1
先乘
toMul
后加
toAdd

pushdown(u) 信息下放

只下放一层:到u的两个儿子




儿子toMul=儿子toMul*老子toMul

儿子toAdd=儿子toAdd*老子toMul+老子toAdd

儿子sum=儿子sum*老子toMul+老子toAdd*儿子len

| pushdown(u) 信息下放 |
| --- |
| 只下放一层:到u的两个儿子 |
| u两儿子维护的区间信息变准确 |
| u本身的延时标记消失 |

```
32 void pushdown(ll u){
33     if(tr[u].l==tr[u].r)return;
34     ll M=tr[u].toMul;
35     ll A=tr[u].toAdd;
36     tr[u].toMul=1;
37     tr[u].toAdd=0;
38     (tr[u*2].toMul*=M)%=P;
39     ((tr[u*2].toAdd*=M)+=A)%=P;
40
41
42
43
44 }
```

# 讨论

样例不通过时如何查错

打印线段树所有节点信息

样例通过后如何检查正确性

使用暴力算法对拍

# 快快编程

| |
|---|
| 667,663,172 |
| 性质证明发课程群 |

| | |
|---|---|
| 拓展题 | 243 |