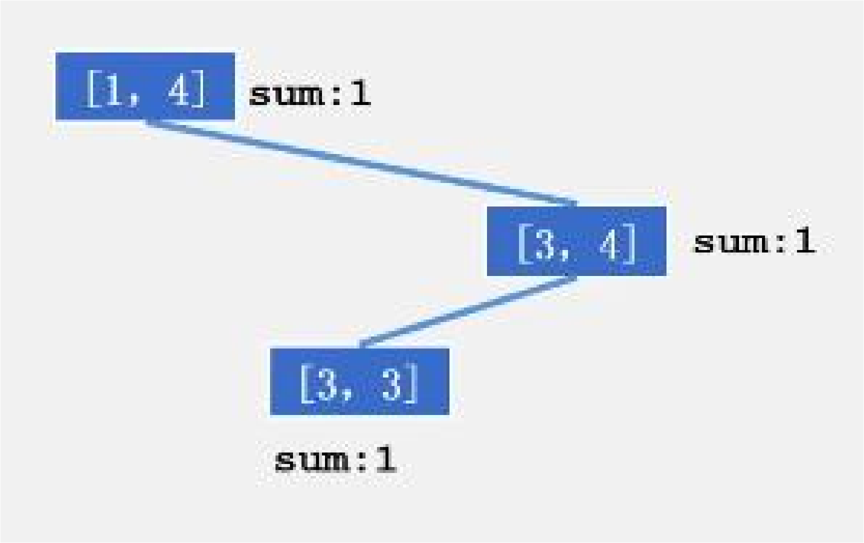
线段树分裂合并与可持久化

动态开节点的线段树

有时候为了节省内存，我们不需要一开始就build整颗线段树，而是每次要修改的时候再去新建。



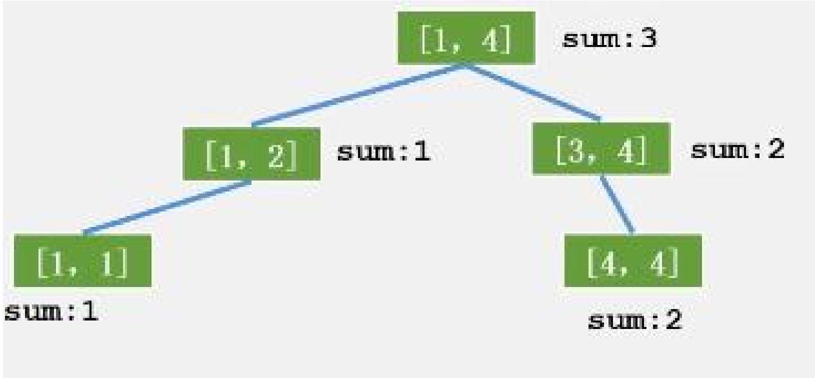
动态开节点的线段树

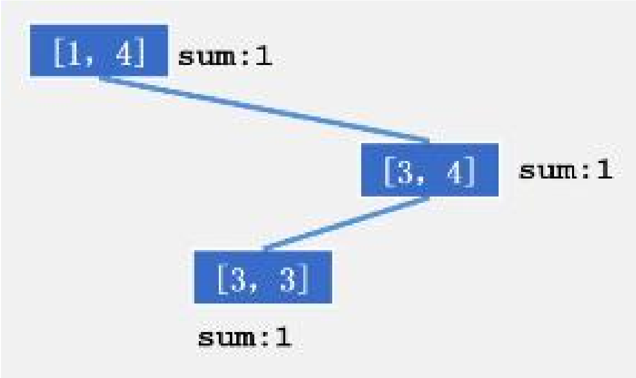
用 lc[x], rc[x] 表示 x 的左右子节点。

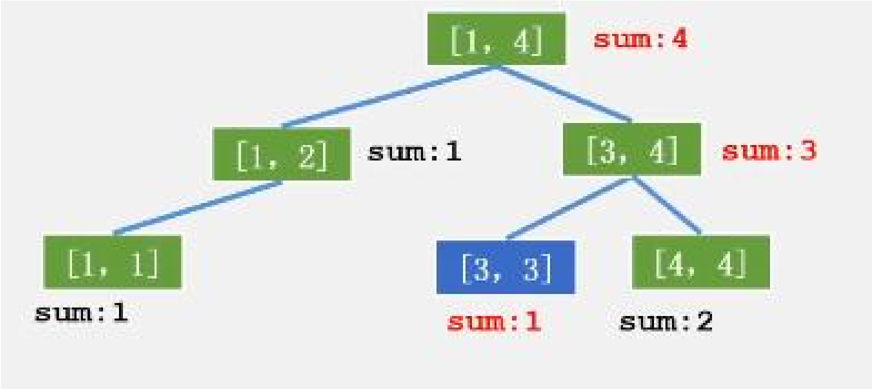
```
void pushup(int x){
    sum[x]=sum[lc[x]]+sum[rc[x]];
}

void modify(int p,int v,int l,int r,int &x){
    if(!x) x=++tot;
    int mid = (l+r)>>1;
    if(p<=mid) modify(p,v,l,mid,lc[x]);
    else modify(p,v,mid+1,r,rc[x]);
    pushup(x);
}
```

线段树的合并

当线段树维护的权值范围 [1,n] 先沟通，则线段树的形态是唯一的，两颗 n 一样的线段树是可以合并的。





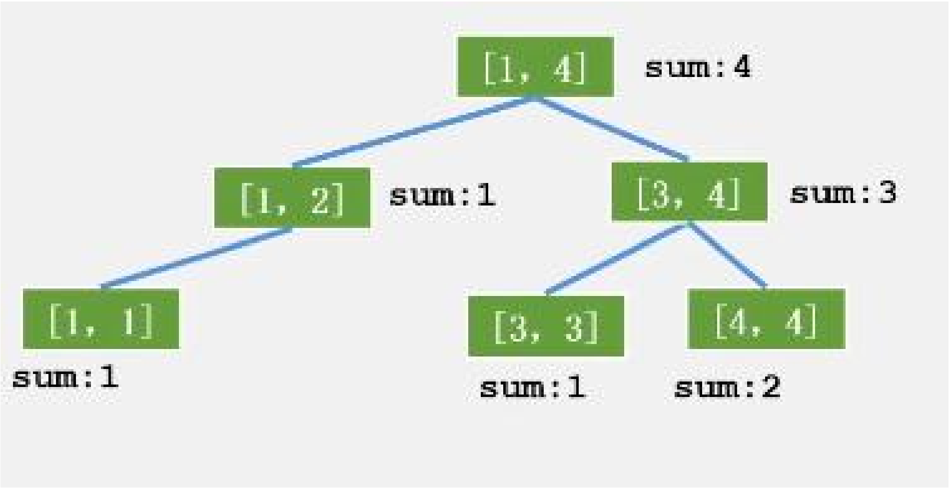


线段树的合并

```
void merge(int &x,int y){
    if(!y) return;
    if(!x){
        x=y;
        return;
    }
    merge(lc[x],lc[y]);
    merge(rc[x],rc[y]);
    pushup(x);
}
```

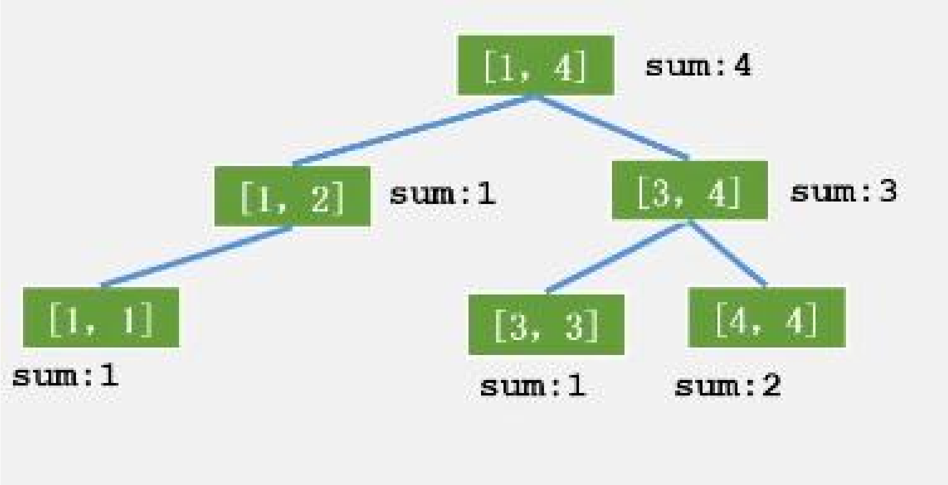
线段树的分裂

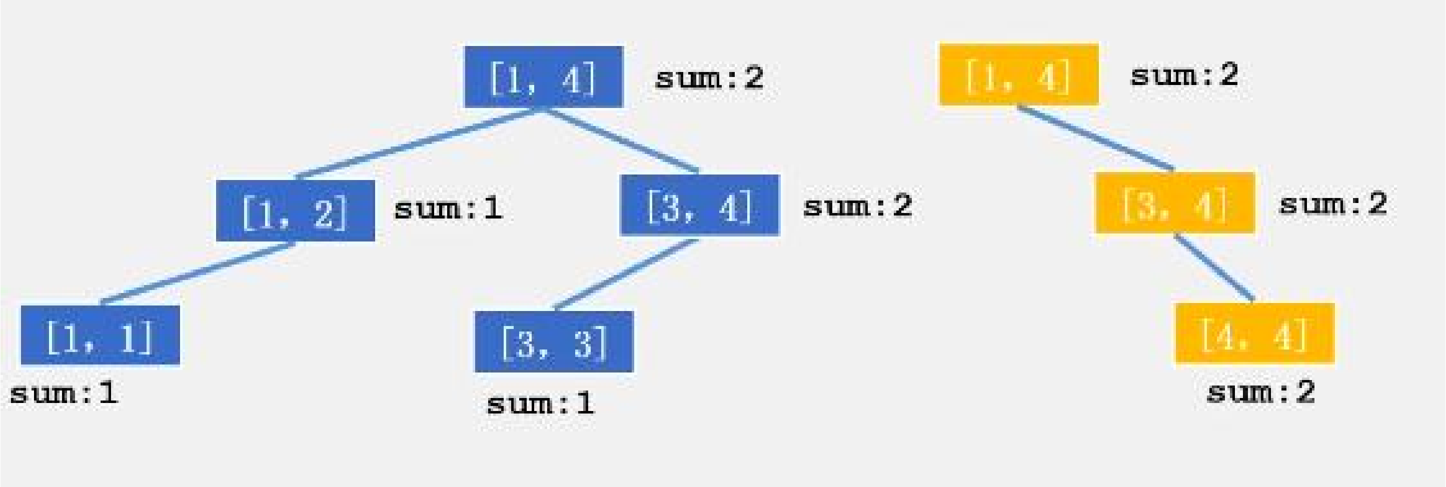
- 对于下图，假设我们想把最小的两个节点分裂出来。



线段树的分裂

- 左边这棵线段树包含了前2小的数。
- 右边是原树减去左边的的树





线段树的分裂

```
pair<int,int> split(int x,int k){
    if(!x) return make_pair(0,0);
    if(k<=sum[lc[x]]){
        pair<int,int> t= split(lc[x],k);
        ++tot;
        lc[tot]=t.first;
        lc[x]=t.second;
        sum[tot]=k;
        sum[x]-=k;
        return make_pair(tot,x);
    }
    else{
        pair<int,int> t=split(rc[x],k-sum[lc[x]]);
        ++tot;
        rc[tot]=t.second;
        rc[x]=t.first;
        sum[tot]=sum[x]-k;
        sum[x]=k;
        return make_pair(x,tot);
    }
}
```

线段树的合并和分裂

- 一次合并/分裂的时间复杂度可大可小。
- 但是均摊复杂度是有保证的。

- n 个单节点线段树合并是 O(nlogn);
- 一个 n 节点线段树分裂 n 次也是 O(nlogn)。

可持久化线段树

有一个长度为 n 的序列,执行 m 次操作:

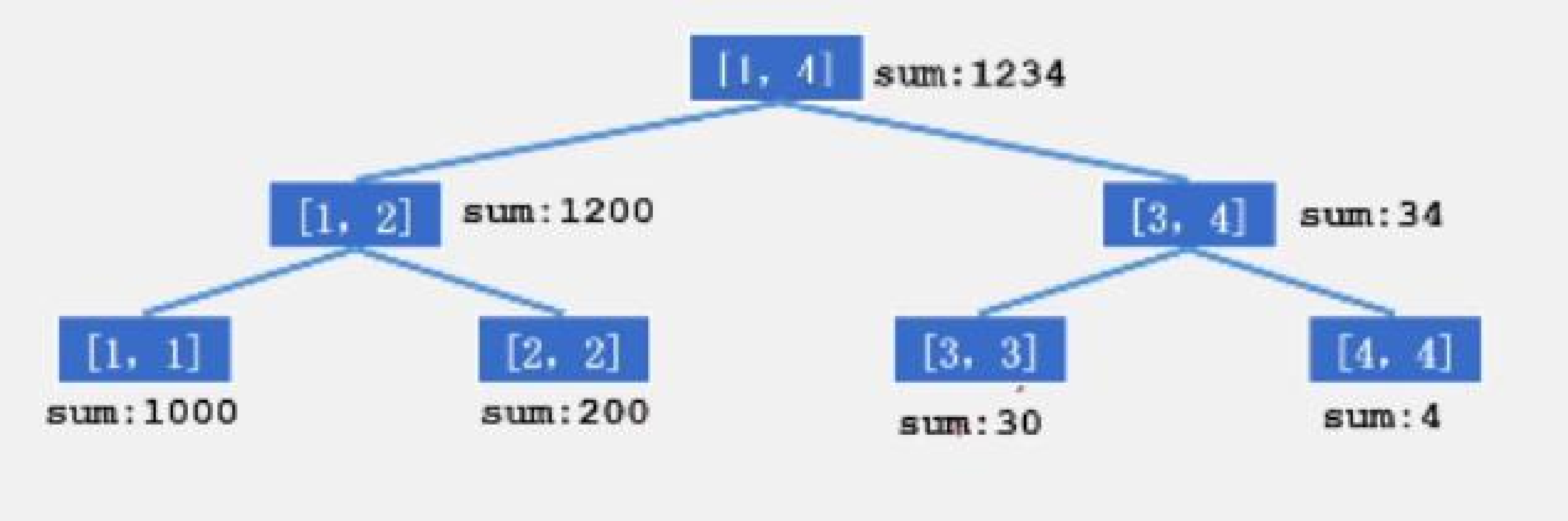
1 i v 将序列中第i个数修改为v

2 k l r 询问第 k 次操作后,[l, r]的区间和。

in:
4
1000 200 30 4
5
1 3 50
1 2 600
2 0 1 4
2 1 1 4
2 2 1 4

out:
1234
1254
1654

可持久化线段树



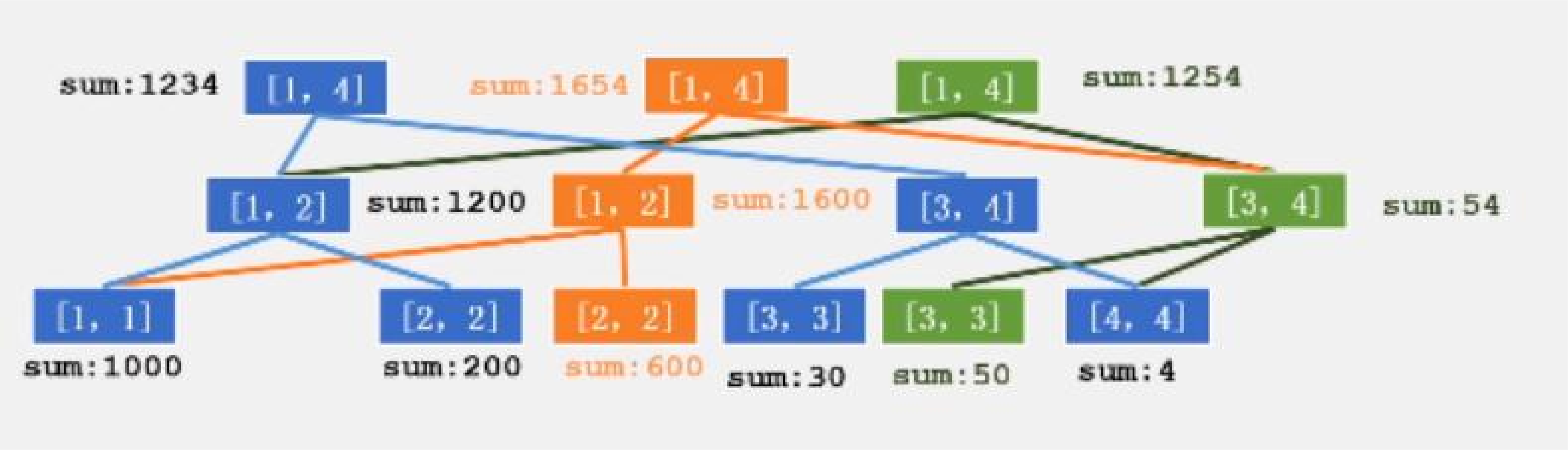
[1, 4]
sum:1234
[1, 2]
sum:1200
[3, 4]
sum:34
[1, 1]
sum:1000
[2, 2]
sum:200
[3, 3]
sum:30
[4, 4]
sum:4

可持久化线段树

可以发现，每次操作与上一颗线段树相比，最多只有 logn 个节点发生变化。也就是从根到对应的叶节点的路径上的所有节点。

因此对于每次修改，不需要存下整颗线段树，只需要存下这新的 logn 个节点。

可持久化线段树



sum:1234
[1, 4]
sum:1654
[1, 4]
[1, 4]
sum:1254
[1, 2]
sum:1200
[1, 2]
sum:1600
[3, 4]
[3, 4]
sum:54
[1, 1]
[2, 2]
[2, 2]
[3, 3]
[3, 3]
[4, 4]
sum:1000
sum:200
sum:600
sum:30
sum:50
sum:4

可持久化线段树

实现时不能再用 2x 和 2x+1 来表示子节点。
每次新加点只需要在每次新开一个节点的时候给它一个新标号就行。

```
void pushup(int x){
    sum[x]=sum[lc[x]]+sum[rc[x]];
}

void build(int l,int r,int &x){
    if(!x) x=++tot;
    if(l==r){
        sum[x]=a[l];
        return;
    }
    int mid = (l+r)>>1'
    build(l,mid,lc[x]);
    build(mid_1,r,rc[x]);
    pushup(x);
}
```

初始化： `build(1,n,root[0]);`

可持久化线段树

单点修改时我们需要新建一个树根，同时有一个指针指向上一个树根，一起向下移动，决定每一层是复制左节点还是右节点。

```
void change(int p,int v,int l,int r,int &x,int pre){
    if(!x) x=++tot;
    if(l==r){
        sum[x]=v;
        return;
    }
    int mid=(l+r)>>1;
    if(p<=mid){
        rc[x]=rc[pre];
        change(p,v,l,mid,lc[x],lc[pre]);
    }else{
        lc[x]=lc[pre];
        change(p,v,mid+1,r,rc[x],rc[pre]);
    }
    pushup(x);
}
```

可持久化线段树

```
int query(int from,int to,int l,int r,int x){
    if(!x) return 0;
    if(from<=l&&r<=to) return sum[x];
    int mid=(l+r)>>1,ret=0;
    if(from<=mid) ret+=query(from,to,l,mid,lc[x]);
    if(mid<to) ret+=query(from,to,mid+1,r,rc[x]);
    return ret;
}
```

```
int main(){
    cin>>n;
    for(int i=1;i<=n;i++) cin>>a[i];
    build(1,n,root[0]);
    cin>>q;
    int op,k,x,y;
    while(q--){
        if(op==1){
            cin>>x>>y;
            change(x,y,1,n,root[i],root[i-1]);
        }else{
            cin>>k>>x>>y;
            cout<<query(x,y,1,n,root[k])<<endl;
        }
    }
}
```

二维线段树

我们可以理解为线段树套线段树。

比如说我们先对 x 坐标建一棵线段树。
这棵线段树的每个节点，又都对应一个线段树的根节点。

修改和查找现在 x 坐标的线段树上找到对应 O(logn) 个点，再在这些点对应的 y 线段树上就行操作即可。

四分树

二维线段树的简化版。

对于区间，我们二分分成两个子区间对应线段树上的子节点。

对于矩形，我们从横、竖两个方向的中间切开，分成四个子矩形，对应四分树上的 4 个子节点。

设 x 的四个子节点编号 4x-2,4x-1,4x,4x+1.

```
void pushup(int x){
    sum[x]=sum[4*x-1]+sum[4*x-1]+sum[4*x]+sum[4*x+1];
}
```

四分树

对于区间，我们二分分成两个子区间对应线段树上的子节点。

对于矩形，我们从横、竖两个方向的中间切开，分成四个子矩形，对应四分树上的 4 个子节点。

设 x 的四个子节点编号 4x-2,4x-1,4x,4x+1.

```
void pushup(int x){
    sum[x]=sum[4*x-1]+sum[4*x-1]+sum[4*x]+sum[4*x+1];
}
```

四分树

对于区间，我们二分分成两个子区间对应线段树上的子节点。

对于矩形，我们从横、竖两个方向的中间切开，分成四个子矩形，对应四分树上的 4 个子节点。

设 x 的四个子节点编号 4x-2,4x-1,4x,4x+1.

```
void pushup(int x){
    sum[x]=sum[4*x-1]+sum[4*x-1]+sum[4*x]+sum[4*x+1];
}
```

四分树

```
void build(int xl,int xr,int yl,int yr,int x){
    if(xl==xr && yl==yr) return;
    int xmid = (xl+xr)>>1;
    int ymid = (yl+yr)>>1;
    build(xl,xmid,yl,yr,4*x-2);
    build(xmid+1,xr,yl,yr,4*x-1);
    build(xl,xr,yl,ymid,4*x);
    build(xl,xr,ymid+1,yr,4*x+1);
    pushup(x);
}
```

四分树

查询操作类似。
检查查询的矩形是否和 4 个子节点有交集。
有交集就需要递归到那个子节点继续查询。

需要注意 时间复杂度不一定是O(logn*logn).
所以还是需要用到二维线段树.