码客行寒假集训 B 层

线段树

安玺儒

天津大学

2025 年 1 月 20 日

线段树

线段树

线段树简介

引入

线段树是算法竞赛中常用的用来维护区间信息的数据结构。
线段树可以在 $O(\log N)$ 的时间复杂度内实现单点修改、区间修改、区间查询（区间求和，求区间最大值，求区间最小值）等操作。

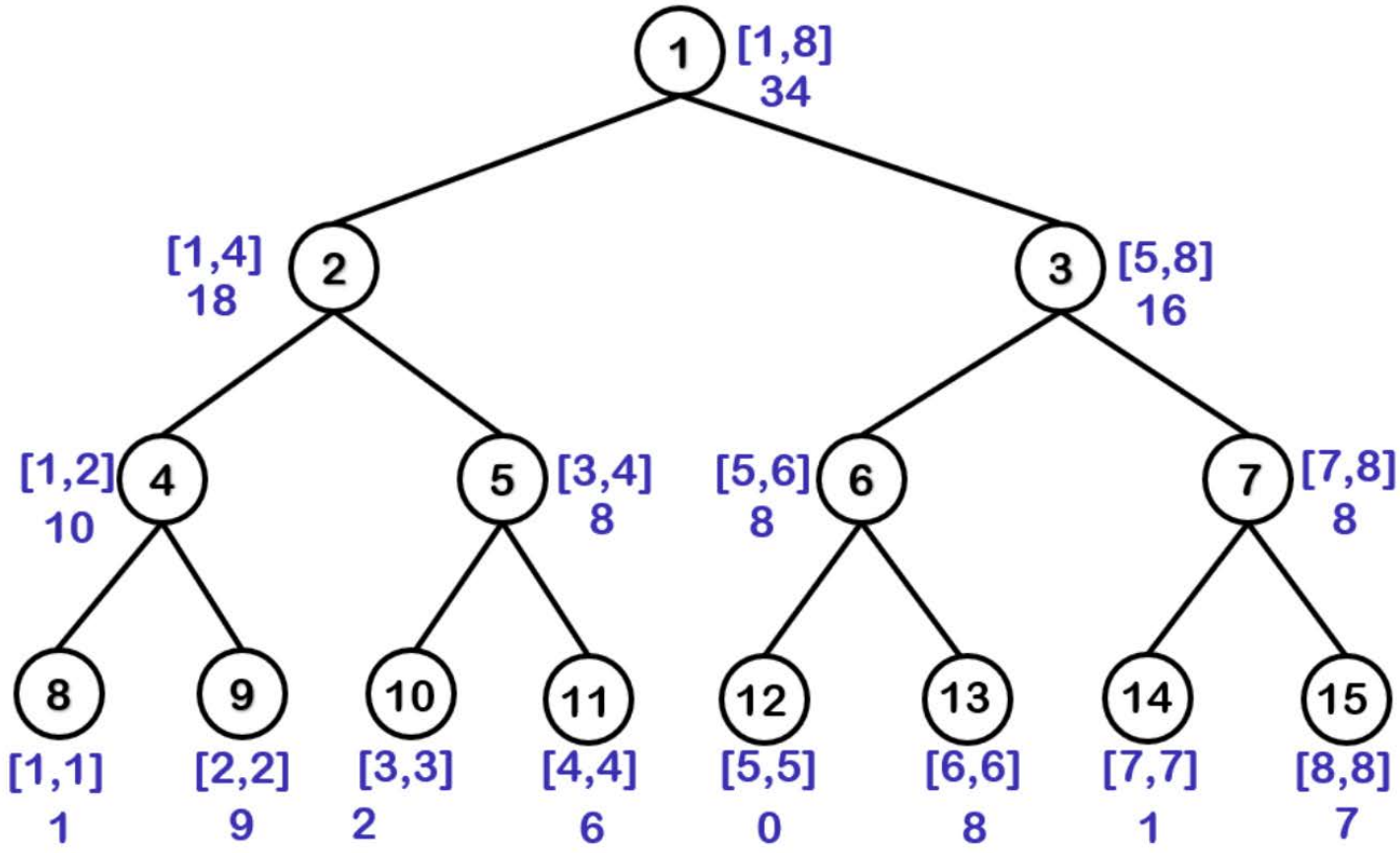
本质

本质上是一颗二叉树，树上每个点都对应了一个区间，区间一分为二为该点对应的左右子节点。每个点可以维护对应区间的信息。

线段树简介

形态

给出 a[] $= \{1, 9, 2, 6, 0, 8, 1, 7\}$，可以构建出如下线段树：



线段树示例

线段树

存储与建树

存储

个人推荐使用结构体进行存储，看起来非常清晰。

```
#define ls k<<1
#define rs k<<1|1
struct node{
    int leftson,rightson,sum,lazytag;//some data
}t[N<<2];//记得开四倍空间
```

建树

我们使用递归的方式，从一个数组构建出一棵线段树。

```
void build(int k, int l, int r){
    if(l==r){t[k].sum=a[l];return ;}
    int mid=l+r>>1;
    build(ls,l,mid),build(rs,mid+1,r);
    t[k].sum=t[ls].sum+t[rs].sum;
}
```

引题

luoguP3374 【模板】树状数组 1

题意: 给定 n 个数的序列 $a_1, a_2, \ldots, a_n$。
对序列进行 m 次操作，每次操作有 2 种类型:
1. 修改第 i 个位置的数 a_i 为 x . (单点修改)
2. 查询区间 $[l, r]$ 的每一个数的和。(区间查询)
$1 \leq n, q, a_i \leq 10^6, 1 \leq l \leq r \leq n$。

区间查询

思路

利用线段树的树状结构，我们可以从根节点向下缩小区间，找到想要查询的区间。
由于查询的区间会被包含在当前线段树区间，因此总共分为三种情况：
1. 查询区间完全被包含在左子树的区间内，
2. 查询区间完全被包含在右子树的区间内，
3. 查询区间横跨左子树的区间和右子树的区间。

分类讨论查询即可。

```
1 int query(int k, int l, int r, int p, int q){
2     //l,r为当前线段树的区间，p,q为想要查询的区间
3     if(p==l&&r==q)return t[k].sum;//找到直接返回
4     int mid=l+r>>1;
5     if(q<=mid)return query(ls,l,mid,p,q);//情况1
6     else if(p>mid)return query(rs,mid+1,r,p,q);//情况2
7     else return query(ls,l,mid,p,mid)
8                 +query(rs,mid+1,r,mid+1,q);//情况3
9 }
```

单点修改

思路

与建树的思路相同，利用递归的思想。
先向下递归子树的信息，子树的信息计算好后将信息传回父节点。
递归同查询思路，分三种情况。递归终止条件是遇到叶子节点。

```
1  int modify(int k, int l, int r, int pos, int val){
2      if(l==r){
3          t[k].sum+=val;
4          return ;
5      }
6      int mid=l+r>>1;
7      if(pos<=mid)modify(ls,l,mid,pos,val);
8      else modify(rs,mid+1,r,pos,val);
9      t[k].sum=t[ls].sum+t[rs].sum;
10 }
```

单点修改泛化

思考

如果我们查询的不是区间和，而是区间最大/小值，该如何维护？

单点修改泛化

思考

如果我们查询的不是区间和，而是区间最大/小值，该如何维护？

区间信息泛化

对于**单点修改**、区间查询的问题，只要满足**区间信息可合并**，我们就可以用线段树来做。

举例

如查询区间和、区间最大/小值、区间和的平方、区间 gcd 等。

线段树

码风比较

建树

其他风格有直接把结构体拆开变成数组的，个人不推荐，代码中表述会非常不清晰，容易写错。

```
int ls[N<<2],rs[N<<2],sum[N<<2],lazy[N<<2];
```

个人推荐结构体，格式清晰。内存方面，如果结构体成员类型相同，则内存大小相同，否则会略大于数组。

```
struct node{
    int ls,rs,sum,lazy;
}tree[N<<2];
```

码风比较

函数变量

其他风格有把线段树节点的左右端点存进结构体中，函数变量中没有线段树节点对应区间的左右端点。

```
struct node{
    int l,r,sum,lazy;
}tree[N<<2]
int query(int k, int p, int q){
    //k代表当前线段树节点编号,p,q代表查询区间左右端点
    ...
}
```

个人推荐将左右端点放进函数变量中，虽然略显繁琐但节省空间。

```
int query(int k, int l, int r, int p, int q){
    //k代表当前线段树节点编号,p,q代表查询区间左右端点
    //l,r代表当前线段树节点对应区间左右端点
    ...
}
```

码风比较

递归判断

以查询举例，有的码风会将三种情况合并为两种情况进行判断。

```
int query(int k, int l, int r, int p, int q){
    ...
    int ans=0;
    if(p<=mid)ans+=query(ls,l,mid,p,q);
    if(q>mid)ans+=query(rs,mid+1,r,p,q);
    return ans;
}
```

个人推荐将三种情况分别列出，便于后续使用懒标记永久化。

```
int query(int k, int l, int r, int p, int q){
    ...
    if(q<=mid)return query(ls,l,mid,p,q);
    else if(p>mid)return query(rs,mid+1,r,p,q);
    else return query(ls,l,mid,p,mid)
                +query(rs,mid+1,r,mid+1,q);
}
```

线段树

引题

luoguP3372 【模板】线段树 1

题意: 给定 n 个数的序列 $a_1, a_2, \ldots, a_n$。
对序列进行 m 次操作，每次操作有 2 种类型:
1. 对于 $i \in [l, r]$ 的数 a_i 将其加上 x .（区间修改）
2. 查询区间 $[l, r]$ 的每一个数的和。（区间查询）

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

懒标记

思考

如果我们像单点修改的思路去做，那么我们每次修改的复杂度高达 $O(n \log n)$。
我们发现一个大区间，我们只需最多 $\log n$ 个节点代表的区间就可以表达，我们可以利用这个性质，给这些区间的节点打上标记。

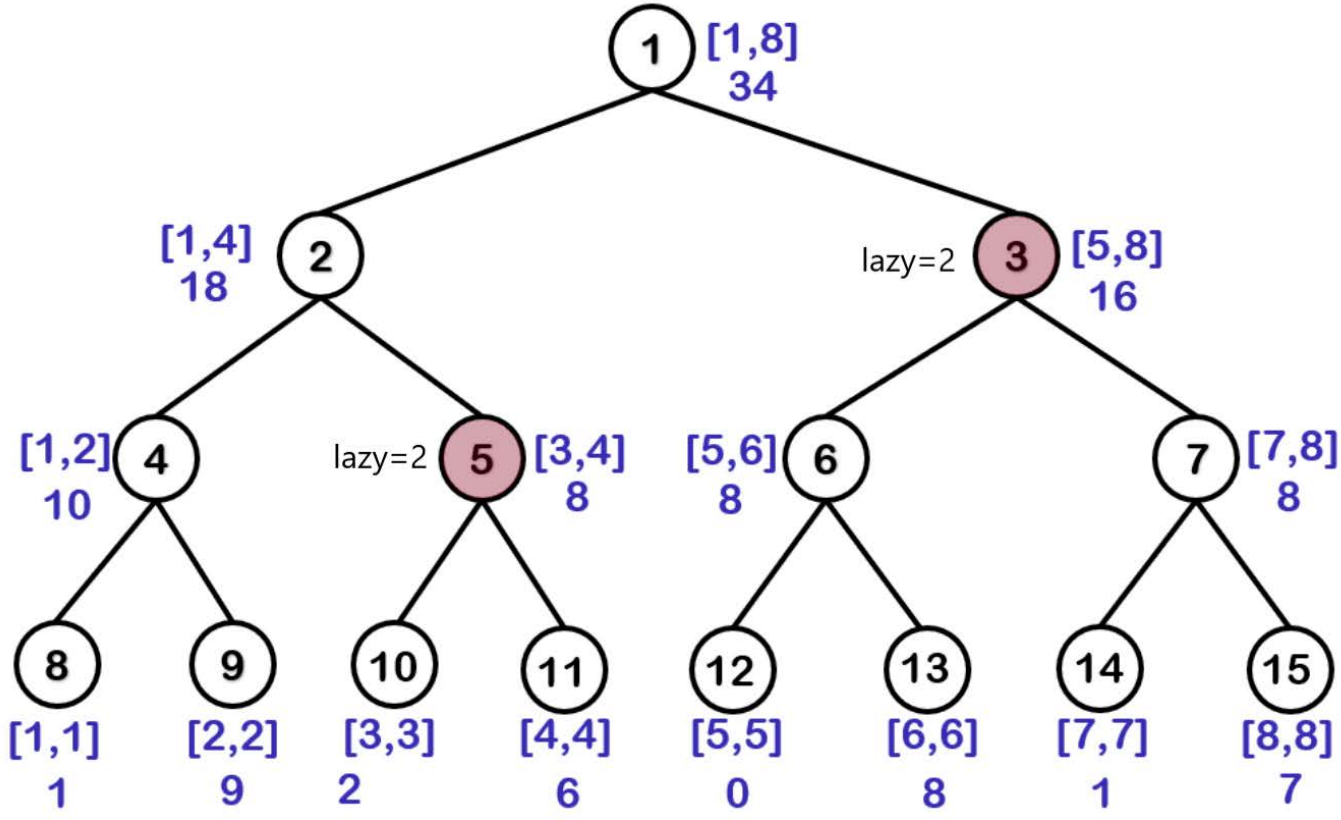
概念

懒标记，简单来说，就是通过延迟对节点信息的更改，从而减少可能不必要的操作次数。每次执行修改时，我们通过打标记的方法表明该节点对应的区间在某一次操作中被更改，但不更新该节点的子节点的信息。实质性的修改则在下一次访问带有标记的节点时才进行。

懒标记

示例

以最开始的数组 a[] $= \{1, 9, 2, 6, 0, 8, 1, 7\}$ 举例: 我们要对区间 $[3, 8]$ 每个数加上 2:



懒标记示例

代码示例

```
1  void modify(int k, int l, int r, int p, int q, ll val){
2      if(p==l&&r==q){put_tag(k,l,r,val);return ;}//new
3      pushdown(k,l,r);//new
4      int mid=l+r>>1;
5      if(q<=mid)modify(ls,l,mid,p,q,val);
6      else if(p>mid)modify(rs,mid+1,r,p,q,val);
7      else modify(ls,l,mid,p,mid,val),
8           modify(rs,mid+1,r,mid+1,q,val);
9      pushup(k);
10 }
11 ll query(int k, int l, int r, int p, int q){
12     if(p==l&&r==q)return t[k].sum;
13     pushdown(k,l,r);//new
14     int mid=l+r>>1;
15     if(q<=mid)return query(ls,l,mid,p,q);
16     else if(p>mid)return query(rs,mid+1,r,p,q);
17     else return query(ls,l,mid,p,mid)
18                 +query(rs,mid+1,r,mid+1,q);
19 }
```

代码示例

```
1  void pushup(int k){
2      t[k].sum=t[ls].sum+t[rs].sum;
3  }
4  void pushdown(int k, int l, int r){
5      if(t[k].lz){
6          int mid=l+r>>1;
7          t[ls].sum+=(mid-l+1)*t[k].lz;
8          t[rs].sum+=(r-mid)*t[k].lz;
9          t[ls].lz+=t[k].lz;
10         t[rs].lz+=t[k].lz;
11         t[k].lz=0;
12     }
13 }
14 void put_tag(int k, int l, int r, int val){
15     t[k].sum+=1ll*(r-l+1)*val;
16     t[k].lz+=val;
17 }
```

区间修改泛化

思考

如果我们修改的不是区间加法，而是区间加、区间乘，该如何维护？

区间修改泛化

思考

如果我们修改的不是区间加法，而是区间加、区间乘，该如何维护？

区间信息泛化

对于**区间修改**、区间查询的问题，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时，同时也可以很方便地更新需要维护的信息（如区间和）

举例

区间加/赋值求区间和，区间加/赋值求区间最大/小值，对 x 取 min/max 求区间和/区间最大值/区间最小值，区间赋值求区间 gcd 等。

线段树

懒标记永久化

为什么需要懒标记永久化

部分特殊的线段树下传懒标记的代价过大，或不支持下传懒标记。例如可持久化 / 动态开点线段树，下传懒标记需要新建结点，空间常数过大；又例如树套树，无法下传懒标记。

区别

未永久化的懒标记：表示暂时未对左右子树的操作，在后续求值或进一步修改时会下放。
永久化的懒标记：表示已经进行对左右子树的操作，在后续求值时不会下放，需要在求值的路径上记录下来，并计入答案中。

懒标记永久化

条件

1. 需要满足区间修改的必要条件，如信息打上标记后可以快速计算。
2. 修改必须与顺序无关，即标记具有交换律。（一个特例是区间赋值，区间赋值虽然与修改顺序相关，但可以通过维护时间戳转化为求时间戳最值）
3. 懒标记在线段树路径上累加不会溢出。

代码示例

```
1  void modify(int k, int l, int r, int p, int q, ll val){
2      t[k].sum+=val*(q-p+1);
3      if(p==l&&q==r){
4          t[k].lz+=val;
5          return ;
6      }
7      int mid=l+r>>1;
8      if(q<=mid)modify(ls,l,mid,p,q,val);
9      else if(p>mid)modify(rs,mid+1,r,p,q,val);
10     else modify(ls,l,mid,p,mid,val),
11          modify(rs,mid+1,r,mid+1,q,val);
12 }
13 ll query(int k, int l, int r, int p, int q, ll lz){
14     if(p==l&&q==r)return t[k].sum+lz*(r-l+1);
15     int mid=l+r>>1;
16     if(q<=mid)return query(ls,l,mid,p,q,lz+t[k].lz);
17     else if(p>mid)return query(rs,mid+1,r,p,q,lz+t[k].lz);
18     else return query(ls,l,mid,p,mid,lz+t[k].lz)
19                 +query(rs,mid+1,r,mid+1,q,lz+t[k].lz);
20 }
```

线段树

空间复杂度优化

前面说过线段树需要提前开出来 $4 \times n$ 的大小的数组，有些节点可能从开始到结束一直没有被用过，造成了空间的浪费。
如何优化？

动态开点

思想

动态开点的核心思想：结点只有在有需要的时候才会创建。
为了节省空间，每次我们可以开始不建好树，最初只建立一个根节点，每次我们需要访问一个子区间时，才建立代表这个区间的子结点。这样可能会导致结点的编号没有父子结点编号二倍的关系，因此我们需要将每个结点的子节点位置记录下来。

复杂度分析

单次操作的复杂度是 $O(\log n)$，m 次单点操作后结点的规模是 $O(m \log n)$。最多也只需要 $2n-1$ 个结点，并且没有浪费。

注意

对于区间修改，有两种方法解决：
1. 通过标记永久化，无需下传标记，解决了子节点可能不存在的问题。
2. 标记不永久化，下传标记时如果子节点不存在则新开一个子节点。

代码示例

```
1  void modify(int &k, int l, int r, int p, int q, ll val){
2      if(!k)k=++cnt; t[k].sum+=val*(q-p+1);
3      if(p==l&&q==r){t[k].lz+=val;return ;}
4      int mid=l+r>>1;
5      if(q<=mid)modify(ls,l,mid,p,q,val);
6      else if(p>mid)modify(rs,mid+1,r,p,q,val);
7      else modify(ls,l,mid,p,mid,val),
8           modify(rs,mid+1,r,mid+1,q,val);
9  }
10 ll query(int k, int l, int r, int p, int q, ll lz){
11     if(!k)return 0;
12     if(p==l&&q==r)return t[k].sum+lz*(r-l+1);
13     int mid=l+r>>1;
14     if(q<=mid)return query(ls,l,mid,p,q,lz+t[k].lz);
15     else if(p>mid)return query(rs,mid+1,r,p,q,lz+t[k].lz);
16     else return query(ls,l,mid,p,mid,lz+t[k].lz)
17                +query(rs,mid+1,r,mid+1,q,lz+t[k].lz);
18 }
```

练习

练习题

使用动态开点技巧完成练习题第一题（洛谷 P3372）。

线段树

线段树上二分

概念

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 $O(\log n)$ 复杂度的查询。

引入

引题 1

给定数组 $a_1, a_2, \ldots, a_n (a_i > 0)$，
求最小的 p，使得 $a_1 + a_2 + \cdots + a_p \geq k$。

引入

引题 1

给定数组 $a_1, a_2, \ldots, a_n (a_i > 0)$，
求最小的 p，使得 $a_1 + a_2 + \cdots + a_p \geq k$。

Solution

求出 a 数组的前缀和数组 s，由于 $a_i > 0$，则 s 数组是递增的，因此在 s 上直接二分，即可得到答案。

例题

例题 1

给定数组 $a_1, a_2, \ldots, a_n (0 < a_i, n \leq 10^6)$，$q (q \leq 10^6)$ 次操作，共两种：

1. 将 a_i 改为 x。
2. 求最小的 p，使得 $a_1 + a_2 + \cdots + a_p \geq k$。

例题

例题 1

给定数组 $a_1, a_2, \ldots, a_n (0 < a_i, n \leq 10^6)$，$q (q \leq 10^6)$ 次操作，共两种：
1. 将 a_i 改为 x。
2. 求最小的 p，使得 $a_1 + a_2 + \cdots + a_p \geq k$。

Solution

有单点修改操作，我们使用线段树存储 a 数组。对于每次询问，如果左子树区间的数字和大于等于 k，则说明答案在左子树，我们在左子树进行进一步查找，否则答案在右子树，我们对右子树进行查找。
注意答案在右子树查找右子树时，应先减去左子树的前缀和。

```
1 int query(int k, int l, int r, int rk){
2     if(l==r)return l;
3     int mid=l+r>>1,x=t[ls].sum;
4     if(x>=rk)return query(ls,l,mid,rk);
5     else return query(rs,mid+1,r,rk-x);
6 }
```

引入

引题 2

给定数组 $a_1, a_2, \ldots, a_n (1 \leq a_i \leq 10^6)$，
求数组中第 k 小元素。

引入

引题 2

给定数组 $a_1, a_2, \ldots, a_n (1 \leq a_i \leq 10^6)$，
求数组中第 k 小元素。

Solution

开一个桶 t，$t[i]$ 代表数组 a 中 i 出现的次数，则转化为引题 1。

例题

例题 2

给定一个集合，一开始为空，有 q 次操作。操作分为两种：
1. 往集合里面加入一个数字 x。
2. 查询集合中第 k 大的数字。

$q, k, x \leq 10^6$。

例题

例题 2

给定一个集合，一开始为空，有 q 次操作。操作分为两种：
1. 往集合里面加入一个数字 x。
2. 查询集合中第 k 大的数字（确保有答案）。

$q, k, x \leq 10^6$。

Solution

将集合转化为桶开一个线段树，例如集合中加入了 10，那么桶对应的变化应为 $t[10] = t[10] + 1$，因此在线段树上将位置 10 的元素加 1。这样可以转化为例题 1，在线段树上二分即可。

权值线段树

概念

我们观察上面的例题 2，将数组或集合转化为桶，再以桶构建一棵线段树，这样的线段树我们称为权值线段树。

性质

可以发现，线段树上结点对应的区间 $[l, r]$ 代表了数组或集合中，元素值域在 $[l, r]$ 中的个数。
由于元素的范围可能较大，因此权值线段树一般搭配离散化使用。

线段树

简介

概念

可持久化数据结构总是可以保留每一个历史版本，并且支持操作的不可变特性。

分类

部分可持久化：所有版本都可以访问，但是只有最新版本可以修改。
完全可持久化：所有版本都既可以访问又可以修改。

实际应用

几何计算、字串处理、版本回溯、函数式编程等。

线段树

引题

luoguP3919 【模板】可持久化线段树 1（可持久化数组）

你需要维护一个长度为 n 的数组，支持以下两种操作：
1. 在某个历史版本上修改某一个位置上的值
2. 访问某个历史版本上的某一位置的值

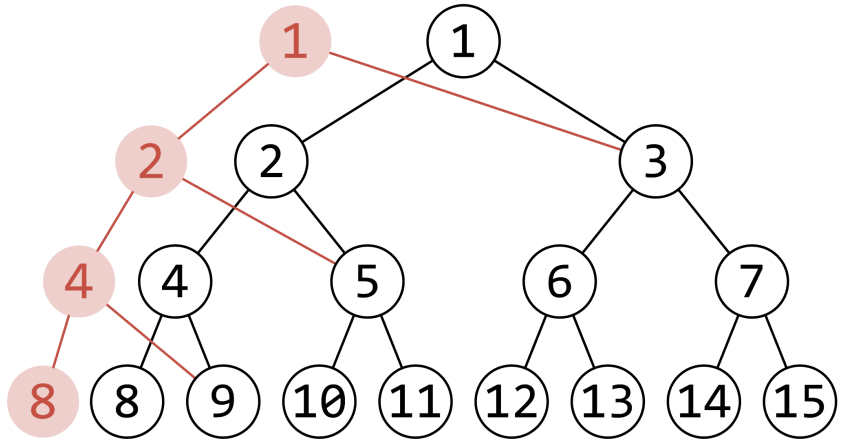
朴素做法

我们维护 m 个线段树（本题只是单点修改单点查询，朴素做法使用数组即可），存下来每一个版本的线段树。
不难发现空间复杂度是 $O(nm)$ 的，如何优化？

可持久化线段树

思考

我们观察两个修改前后的线段树。运用动态开点的思想，单次修改只会改变 $\log n$ 个结点的信息，那么我们直接利用动态开点，将这些点储存到新的结点中，这样既不会破坏原有线段树，又可以存储新的线段树，实现了可持久化。



图中以 15 个点的线段树，并修改了 8 号点为示例。

原理实现

建树

我们用动态开点的方法存储每个点的左右子节点。不过，对于最初版本的那棵树，我们应当一次性建好，而不必一个一个插入。

```
1  void build(int &k, int l, int r){
2      k=++cnt;
3      if(l==r){
4          t[k].val=a[l];
5          return ;
6      }
7      int mid=l+r>>1;
8      build(t[k].ls,l,mid),build(t[k].rs,mid+1,r);
9      ...
10 }
```

使用时直接调用 build(rt[0],1,n) 即可。

原理实现

查询

和普通线段树的查询相同，需要注意查哪个版本就选取哪个根节点。以本题的单点查询为例。

。

```
int query(int k, int l, int r, int pos){
    if(l==r)return t[k].val;
    int mid=l+r>>1;
    if(pos<=mid)return query(t[k].ls,l,mid,pos);
    else return query(t[k].rs,mid+1,r,pos);
}
```

查询第 v 个版本的第 i 个位置调用 query(rt[v],1,n,i) 即可。

原理实现

修改

每进行一次单点修改，都会产生一条新的链。我们当前要对第 v 个版本的某一条链进行更新，假设我们已经更新到点 pre。
首先我们要新建一个点 k，代表当前版本对于点 pre 更新后的版本，我们先将 k 的左右儿子指向 pre 的左右儿子，然后再去判断需要的单点修改会在左子树还是右子树上有影响，并递归相应子树即可。

```
1  void modify(int &k, int l, int r, int pre, int pos, int
       val){
2      k=++cnt,t[k]=t[pre];
3      if(l==r){
4          t[k].val=val;
5          return ;
6      }
7      int mid=l+r>>1;
8      if(pos<=mid)modify(t[k].ls,l,mid,t[pre].ls,pos,val);
9      else modify(t[k].rs,mid+1,r,t[pre].rs,pos,val);
10 }
```

可持久化线段树

注意

1. 可持久化线段树需要动态开点，因此我们需要维护结点的左右子树具体编号。
2. 观察可以发现，可持久化线段树不止有一个根，第 i 个根代表了第 i 个版本对应的线段树的根。
3. 在可持久化线段树进行区间修改时，如果不使用标记永久化，则在下传标记时会新建一个结点，这样会导致空间常数较大，因此应尽可能地使用标记永久化。

空间复杂度分析

首先通过动态开点建树，使用了 $2n-1$ 个结点，每次修改至多增加 $\lfloor \log_2 n \rfloor + 1$ 个点，因此总共需要 $n(\lfloor \log_2 n \rfloor + 3) - 1$ 个结点，对于 $n = 10^5$ 级别，开 30 倍足够使用。

练习

使用可持久化线段树完成练习题第二题（洛谷 P3919）。

线段树

可持久化权值线段树

概念

可持久化权值线段树，又称主席树，顾名思义就是将权值线段树可持久化。
由于与权值相关，因此一般搭配离散化使用。
主席树的一个非常经典的应用是静态区间第 K 小。

luogu P3834 【模板】可持久化线段树 2

题意：给定数列 $a_1, a_2, \ldots, a_n$，q 次询问，每次询问一个区间 $[l, r]$，求 $a_l, a_{l+1}, \ldots, a_r$ 中第 k 小的值是多少。
$0 < l, r, n, m \leq 2 \times 10^5, 0 \leq a_i \leq 10^9$。

实现思路

思考

前面我们在讲线段树上二分时候，已经会在权值线段树上二分求区间 $[1, R]$ 的第 k 小了。那么我们如何求区间 $[L, R]$ 的第 k 小呢？

实现思路

思考

前面我们在讲线段树上二分时候，已经会在权值线段树上二分求区间 $[1, n]$ 的第 k 小了。那么我们如何求区间 $[L, R]$ 的第 k 小呢？

解决

利用前缀和的性质，我们只需要有区间 $[1, R]$ 对应的权值线段树，以及区间 $[1, L-1]$ 对应的权值线段树，在这两棵线段树上一起跑二分，即可实现查询区间 $[L, R]$ 第 k 小。

因此我们需要维护对于每一个 i，区间 $[1, i]$ 的权值线段树。

朴素做法是开 n 个权值线段树，空间复杂度 $O(n^2 \log n)$。

我们观察区间 $[1, i]$ 到区间 $[1, i+1]$ 的变化，相当于在位置 $i+1$ 进行了一次单点修改，和上面提到的可持久化线段树一样，因此相邻区间对应的权值线段树只会有一条链是不同的，因此我们只需要利用可持久化线段树维护这条链即可。空间复杂度 $O(n \log n)$。

练习

使用可持久化权值线段树完成练习题第三题（洛谷 P3834）。

代码

```
1  void modify(int &k, int l, int r, int pre, int pos, int
       val){
2      k=++cnt;
3      t[k]=t[pre];
4      if(l==r){
5          t[k].sum++;
6          return ;
7      }
8      int mid=l+r>>1;
9      if(pos<=mid)modify(t[k].ls,l,mid,t[pre].ls,pos,val);
10     else modify(t[k].rs,mid+1,r,t[pre].rs,pos,val);
11     t[k].sum=t[t[k].ls].sum+t[t[k].rs].sum;
12 }
13 int query(int p, int q, int l, int r, int rk){
14     if(l==r)return l;
15     int mid=l+r>>1,x=t[t[q].ls].sum-t[t[p].ls].sum;
16     if(x>=rk)return query(t[p].ls,t[q].ls,l,mid,rk);
17     else return query(t[p].rs,t[q].rs,mid+1,r,rk-x);
18 }
```

练习题单

https://www.luogu.com.cn/training/524886#information
洛谷 P3372 【模板】线段树 1
洛谷 P3919 【模板】可持久化线段树 1（可持久化数组）
洛谷 P3834 【模板】可持久化线段树 2
洛谷 P1383 高级打字机
洛谷 P3567 [POI2014] KUR-Couriers
洛谷 P7424 [THUPC2017] 天天爱射击

End

Thank You!

Any Questions?