数据结构虐哭空巢老人记

前言

STO f啦sh ORZ

by 去不了冬令营的徐叔叔

搞过的东西就不再写了（数组队列栈链表、线段树动态树替KD树树状数组Splay替罪羊Treap、线段树合并Trie合并、可持久化Trie可持久化线段树、线段树优化DP优化连边）

要写的是

- 李超线段树
- 吉利线段树
- 线段树分裂
- 虚树
- 树链的并

还要进一步学习

- 线段树分治
- 二进制分组
- 珂朵莉树

数据结构题注意点

- 如果更新某点，为了方便访问到其孩子，那么线段树要开8倍空间。否则4倍空间就好了

好题题解链接

- SPOJ Qtree5

李超线段树

用处

维护平面上若干一次函数（直线or线段）的**最值**

（线段树可以维护一次函数的和）

常数小

用法

为了使得单次修改&查询复杂度是$\mathcal{O}(logn)$，我们这样维护这棵线段树：树上每个节点维护一根直线

修改到i号点时

- 如果该点没有维护任何直线，直接赋值，$return$。
- 如果该点维护了一根直线，判断在这个区间内是否其中一根直线始终大于另一根，若是，替换后$return$。

- 否则求出两根直线的交点，把在中点处函数值大(小)的保留，剩下那根往其中一个孩子下放。

如此修改能够保证复杂度不超过$\mathcal{O}(logn)$。

查询横坐标x时，相当于每根直线就是永久化的懒标记，不断计算经过的线段树节点的函数对应的函数值，答案对其取最值，复杂度严格$\mathcal{O}(logn)$

举例

JSOI2008 Blue Mary开公司： 模板题 直接维护直线

SDOI2016 游戏： 强行放在树上让选手写树剖，同时维护的是线段，需要递归到修改区间范围内才能进行修改。需要维护一个子树最小值。注意本题横坐标不连续。

代码

```
double k[N<<2],b[N<<2];
int vis[N<<2];
void Add(int x,int l,int r,double K,double B)
{
    if(!vis[x]) {vis[x]=1;k[x]=K,b[x]=B;return;}
    double l1=k[x]*l+b[x],r1=k[x]*r+b[x];
    double l2=K*l+B,r2=K*r+B;
    int mid=(l+r)>>1;
    if(l1>=l2&&r1>=r2) return;
    if(l1<=l2&&r1<=r2) {k[x]=K,b[x]=B;return;}
    double xx=(B-b[x])/(k[x]-K);
    if(l1>l2)
    {
        if(xx>mid) Add(x<<1|1,mid+1,r,K,B);
        else Add(x<<1,l,mid,k[x],b[x]),k[x]=K,b[x]=B;
    }
    else
    {
        if(xx>mid)
Add(x<<1|1,mid+1,r,k[x],b[x]),k[x]=K,b[x]=B;
        else Add(x<<1,l,mid,K,B);
    }
}
double Query(int x,int l,int r,int p)
{
    if(!vis[x]) return 0;
    if(l==r) return p*k[x]+b[x];
    int mid=(l+r)>>1;
    double res=p*k[x]+b[x];
    if(p<=mid) res=max(res,Query(x<<1,l,mid,p));
    else res=max(res,Query(x<<1|1,mid+1,r,p));
    return res;
}
```

吉利线段树

用处

PS：学艺不精，只完成了与之有关的几道题，所以总结绝对会不到位。等到位之后会更新博客但是不会更新PDF

支持区间覆盖、区间加法、区间求和等操作，**又能同时支持区间取max操作**的线段树（又名吉司机线段树、$segment\ tree\ beats$、势能线段树）

用法

以支持区间取max的操作为例。对于每个节点，维护一个最小值mn的同时，维护最小值出现的次数num以及严格次小值se。当对某点要进行区间对x取max操作时

- $x \leq mn$，直接返回。
- $mn < x < se$，标记tag，要把子树内的最小值更新为x。
- $se \leq x$，继续递归。

毒瘤出题的生成方式就是既区间取min又取max还维护其他乱七八糟的操作，使得码量巨大。

复杂度证明

复杂度被证明在$\mathcal{O}(nlogn)$和$\mathcal{O}(nlog^2n)$之间。jiry_2的2016年国家集训队论文以及冬令营课件《segment tree beats！》中有详细描述，由于一些不可描述的原因jiry在UOJ上说$\mathcal{O}(nlogn)$的证明是假的（~~正好我也没看懂~~），这里简单介绍$\mathcal{O}(nlog^2n)$的证明。同时这种方法在$n \leq 5 * 10^5$是表现较好的。

$\mathcal{STO}$呆呆亮$\mathcal{ORZ}$。

把mn（下放完所有懒标记）和父亲不同的节点定义为关键点，相同的点定义为非关键点。定义势函数ϕ为关键点数量。也就是说，我们假定势函数是下放完所有懒标记之后的线段树上的信息，而实际操作中并不这样做，也不会有影响（因为我们边走边下放，访问到一个点一定是调用的真实值）。

在一次区间操作中会访问到log个点，最坏情况下，全局增加log个关键节点，势函数增加log。（可以是被完整覆盖的点和其父亲的关系、也可以是不被完整覆盖的点和其孩子的关系）

定义三类点：1类点表示满足$x \geq se > mn$的、会暴力往下递归的点；2类点表示$se > x > mn$的，会在此打上标记返回的点；3类点表示$mn \geq x$的直接返回的点。叶子节点的$se = inf$。

由此若从一个1类点暴力递归它的孩子，若它的孩子是2,2或者2,3都会使得势能-1（对应有一个2类点从非关键点变成了关键点），若孩子是有1则会继续递归，直到使势能-1为止（势能不能-1当且仅当两个孩子都是3，而这种情况不会出现。）

那么一共至多会有$nlogn$个关键点，每个关键点至多需要$\mathcal{O}(logn)$的复杂度去变成非关键点。总复杂度至多为$\mathcal{O}(nlog^2n)$

例题

BZOJ4355 Play with sequence

对于一个非负整数序列维护三个操作：($n \leq 5 * 10^5$)

1.区间$[l,r]$覆盖为c
2.区间$[l,r]$都赋值为$max(a[i]+c,0)$
3.查询$[l,r]$内0的个数

把第二个操作看成区间加&区间取max，第三个操作看做查询区间最小值的个数

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#define ll long long
using namespace std;
const ll N=1201000,inf=1e15;
ll n,q,a[N];
struct seg{ll mn,se,tagmin,num,cov,add;}t[N];
void update(seg &A,seg B,seg C)
{
    if(B.mn>C.mn) swap(B,C);
    A.mn=B.mn;A.num=B.num;
    if(C.mn==A.mn) A.num+=C.num,C.mn=C.se;
    A.se=min(B.se,C.mn);
}
void Build(ll x,ll l,ll r)
{
    if(l==r) {t[x].mn=a[l];t[x].num=1;return;}
    ll mid=(l+r)>>1;
    Build(x<<1,l,mid);
    Build(x<<1|1,mid+1,r);
    update(t[x],t[x<<1],t[x<<1|1]);
}
void putcov(ll x,ll l,ll r,ll v)
{
    t[x].mn=v;
    t[x].se=inf;
    t[x].tagmin=-1;
    t[x].add=0;
    t[x].cov=v;
    t[x].num=r-l+1;
}
void putadd(ll x,ll v)
{
    t[x].mn+=v;
    t[x].se+=v;
    t[x].tagmin+=v;
```

```
    t[x].add+=v;
}
void putmax(ll x,ll v)
{
    if(v<=t[x].mn) return;
    t[x].mn=v;
    t[x].tagmin=v;
}
void pushdown(ll x,ll l,ll mid,ll r)
{
    ll &v1=t[x].cov,&v2=t[x].add,&v3=t[x].tagmin;
    if(v1)
putcov(x<<1,l,mid,v1),putcov(x<<1|1,mid+1,r,v1),v1=0;
    if(v2) putadd(x<<1,v2),putadd(x<<1|1,v2),v2=0;
    if(v3!=-1) putmax(x<<1,v3),putmax(x<<1|1,v3),v3=-1;
}
void Cover(ll x,ll l,ll r,ll gl,ll gr,ll v)
{
    if(l>=gl&&r<=gr) {putcov(x,l,r,v);return;}
    ll mid=(l+r)>>1;
    pushdown(x,l,mid,r);
    if(gl<=mid) Cover(x<<1,l,mid,gl,gr,v);
    if(gr>mid) Cover(x<<1|1,mid+1,r,gl,gr,v);
    update(t[x],t[x<<1],t[x<<1|1]);
}
void Add(ll x,ll l,ll r,ll gl,ll gr,ll v)
{
    if(l>=gl&&r<=gr) {putadd(x,v);return;}
    ll mid=(l+r)>>1;
    pushdown(x,l,mid,r);
    if(gl<=mid) Add(x<<1,l,mid,gl,gr,v);
    if(gr>mid) Add(x<<1|1,mid+1,r,gl,gr,v);
    update(t[x],t[x<<1],t[x<<1|1]);
}
void Max(ll x,ll l,ll r,ll gl,ll gr,ll v)
{
    if(v<=t[x].mn) return;
```

```
    if(l>=gl&&r<=gr&&v<t[x].se) {putmax(x,v);return;}
    ll mid=(l+r)>>1;
    pushdown(x,l,mid,r);
    if(gl<=mid) Max(x<<1,l,mid,gl,gr,v);
    if(gr>mid) Max(x<<1|1,mid+1,r,gl,gr,v);
    update(t[x],t[x<<1],t[x<<1|1]);
}
ll Query(ll x,ll l,ll r,ll gl,ll gr)
{
    if(l>=gl&&r<=gr) return (t[x].mn==0)?t[x].num:0;
    ll mid=(l+r)>>1,res=0;
    pushdown(x,l,mid,r);
    if(gl<=mid) res+=Query(x<<1,l,mid,gl,gr);
    if(gr>mid) res+=Query(x<<1|1,mid+1,r,gl,gr);
    return res;
}
int main()
{
    cin>>n>>q;
    for(ll i=1;i<=n;i++) scanf("%lld",&a[i]);
    for(ll i=1;i<=n*4;i++) t[i].se=inf,t[i].tagmin=-1;
    Build(1,1,n);
    for(ll i=1;i<=q;i++)
    {
        ll op,l,r,C;
        scanf("%lld%lld%lld",&op,&l,&r);
        if(op<3) scanf("%lld",&C);
        if(op==1) Cover(1,1,n,l,r,C);
        if(op==2) Add(1,1,n,l,r,C),Max(1,1,n,l,r,0);
        if(op==3) printf("%lld\n",Query(1,1,n,l,r));
    }
    return 0;
}
```

相当于维护了三个标记：$tagmin$、add、cov

下放的优先级是：cov、add、$tagmin$

重点在于$pushdown$

更多题目

- BZOJ5312 冒险
- 等等（可见于FlashHu的博客）

线段树分裂

用处

大概只能解决区间排序问题。。

常能被其他算法艹过。。

用法

和合并一样的写法。合并丢掉哪些点，分裂的时候就加回来

例题

HEOI2016/TJOI2016 排序 ： 被二分+线段树艹翻。当然分裂是$\mathcal{O}(nlogn)$，二分是$\mathcal{O}(nlog^2n)$

代码

例题代码。

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<set>
#define lc t[x].ch[0]
#define rc t[x].ch[1]
using namespace std;
const int N=1e5+10;
int n,m,a[N],lost,tong[N],nod,o[N],rt[N],q,ans;
set<int> S;
struct seg{int ch[2],siz;}t[N*200];
int query(int x,int l,int r)
{
    if(l==r) return l;
    int mid=(l+r)>>1;
    return t[lc].siz?query(lc,l,mid):query(rc,mid+1,r);
}
void add(int &x,int l,int r,int p)
{
    t[++nod]=t[x];x=nod;t[x].siz++;
    if(l==r) return;
    int mid=(l+r)>>1;
    if(p<=mid) add(lc,l,mid,p);
    else add(rc,mid+1,r,p);
}
int merge(int x,int y)
{
    if(!x||!y) return x+y;
    int p=++nod;
    t[p].ch[0]=merge(lc,t[y].ch[0]);
    t[p].ch[1]=merge(rc,t[y].ch[1]);
    t[p].siz=t[t[p].ch[0]].siz+t[t[p].ch[1]].siz;
    return p;
}
void split(int nw,int &x,int &y,int k)
{
    if(t[nw].siz==k) {x=nw,y=0;return;}
```

```
    t[++nod]=t[x];t[x=nod].siz=k;
    t[++nod]=t[y];t[y=nod].siz=t[nw].siz-k;
    if(k<=t[t[nw].ch[0]].siz)

split(t[nw].ch[0],lc,t[y].ch[0],k),t[y].ch[1]=t[nw].ch[1],r
c=0;
    else split(t[nw].ch[1],rc,t[y].ch[1],
               k-
t[t[nw].ch[0]].siz),lc=t[nw].ch[0],t[y].ch[0]=0;
}
void split(int p)
{
    set<int>::iterator it=S.upper_bound(p);it--;
    o[p]=o[*it];if(*it==p) return;
    int tt=rt[*it];rt[*it]=rt[p]=0;
    if(!o[p]) split(tt,rt[*it],rt[p],p-*it);
    else split(tt,rt[p],rt[*it],t[tt].siz-(p-*it));
    S.insert(p);
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    for(int i=1;i<=n;i++) add(rt[i],1,n,a[i]),S.insert(i);
    S.insert(n+1);
    for(int i=1;i<=m;i++)
    {
        int op,l,r;scanf("%d%d%d",&op,&l,&r);
        split(l);split(r+1);
        set<int>::iterator it1,it2,it;
        it1=S.find(l);it2=S.find(r+1);
        for(it=++it1;it!=it2;it++)
rt[l]=merge(rt[l],rt[*it]);
        o[l]=op;S.erase(it1,it2);
    }
    cin>>q;split(q);split(q+1);
    ans=query(rt[q],1,n);
```

```
    cout<<ans<<endl;
    return 0;
}
```

虚树

用处

把有用的点抠出来建树做。

是解决多次询问 $\sum$询问节点数 一定的题的利器。

用法

之前有学过$\mathcal{O}(|\mathcal{S}|)$建虚树的方法（$\mathcal{S}$表示询问到的节点数）：所有点按照dfn排序，把相邻两点的lca也加入询问点集。用单调栈的方式建出虚树。

```
for(int i=1;i<=cc;sta[++tt]=p[i],i++)
{
    while(tt&&out[sta[tt]]<in[p[i]]) tt--;
    if(tt) link(sta[tt],p[i]);
}
//p是涉及到的点（包括lca）按照dfn排序的数组
//in out 是欧拉序中该点 首次被访问到和访问完所有儿子的时间戳
```

但是下面这题为了很方便地维护路径信息，采用了$\mathcal{O}(n)$建虚树的方法

（询问分块+虚树 好像是一个常用套路）

例题

Codeforces1017G The Tree

在黑白树上维护三种操作：（$n, q \leq 10^5$）

daan1.把x点染黑，若其本来就是黑色，则对其所有儿子递归进行这个操作
2.把x点的子树染白
3.单点查颜色

做法：

把询问分块，对某块内涉及到的点和根节点建虚树（不建lca），块内询问暴力做，每做完一块后暴力对整棵树下放所有操作。

具体而言，麻烦的是对整棵树下放操作，需要维护$bl[x]$表示x最多能往下染黑（包括自己）多少个点，$cc[x]$表示x是否打过子树染白标记。下放时对于在虚树内的节点直接看$bl[x]$，对于不在虚树内的节点是黑色当且仅当：父亲有下传染黑标记/自己本身就是黑的且父亲没有下传染白标记

```cpp
#include<iostream>
#include<cmath>
#include<vector>
#define pb push_back
using namespace std;
const int N=2e5+10;
int n,q,M,bl[N],vis[N],cc[N];
struct Que{int op,x;}B[N];
struct TO{int to,d,t;};
vector<TO> V[N];
vector<int> E[N];
void build(int x,int lst,int dep,int wh)
{
    if(vis[x]&&x>1) V[lst].pb((TO){x,dep,wh});
    //边权记录这条边有多少点（记头不计尾）、以及lst需要接受多少个染黑标记才能从这条边扩展出去（也就是这条边除了lst的白点数量+1）
    for(auto y:E[x])
        if(vis[x]) build(y,x,1,1);
        else build(y,lst,dep+1,wh+1-bl[x]);
}
void dfs1(int x) {bl[x]++;for(auto &P:V[x])
if(bl[x]>=P.t+1) dfs1(P.to);}
void dfs2(int x) {bl[x]=0;for(auto &P:V[x])
P.t=P.d,dfs2(P.to);cc[x]=1;}
void Reset(int x,int blkdep,int cov)
{
    if(!vis[x]) bl[x]=blkdep+(bl[x]&&!cov);
    for(auto y:E[x]) Reset(y,max(0,bl[x]-1),cov|cc[x]);
    bl[x]=min(1,bl[x]);
    if(vis[x]) V[x].clear(),vis[x]=cc[x]=0;
}
int main()
{
    scanf("%d%d",&n,&q);
    for(int i=2,x;i<=n;i++) scanf("%d",&x),E[x].pb(i);
    for(int i=1;i<=q;i++) scanf("%d%d",&B[i].op,&B[i].x);
    M=sqrt(q);
```

```
    for(int b=1,l=0,r=0;r<q;b++)
    {
        l=(b-1)*M+1;r=min(l+M-1,q);
        for(int i=l;i<=r;i++) vis[B[i].x]=1;
        vis[1]=1;build(1,1,0,0);
        for(int i=l;i<=r;i++)
        {
            if(B[i].op==1) dfs1(B[i].x);
            else if(B[i].op==2) dfs2(B[i].x);
            else printf("%s\n",bl[B[i].x]?"black":"white");
        }
        Reset(1,0,cc[1]);
    }
}
```

复杂度分析：

根号块询问，每块询问建虚树$\mathcal{O}(n\sqrt{n})$，每块暴力查询$\mathcal{O}(n\sqrt{n})$，块间暴力下放$\mathcal{O}(n\sqrt{n})$，总复杂度$\mathcal{O}(n\sqrt{n})$。完美。

树链的并

这个在这篇博客里有讲。

BZOJ5084 hashit：求支持后端删除的SAM中本质不同的子串个数。

解释一下做法：

维护点集中所有点到根的路径的并

- 加入一个点x：在set中找到dfn序的前驱a和后继b，$+path(x,a),+path(x,b),-path(a,b)$
- 删除一个点x：在set中找到dfn序的前驱a和后继b，$-path(x,a),-path(x,b),+path(a,b)$

上述过程中若不存在a或b，与之有关的式子皆不算进答案。

$path(x,y)=path(x,rt)+path(y,rt)-2*path(lca(x,y),rt)$