

数据结构

我们都喜欢数据结构

众所周知中国选手数据结构水平宇宙第一

数据结构题基本上必考而且在经过大量训练以后数据结构题的得分情况一般是很好的

树状数组BIT



lowbit

Lowbit(x)表示x的二进制位中最低的那位
直接算是log int的

$x \& (x - 1)$

$X \& (-X)$

取负是反码+1

- ▶ $1=(001)$ $C[1]=A[1];$
- ▶ $2=(010)$ $C[2]=A[1]+A[2];$
- ▶ $3=(011)$ $C[3]=A[3];$
- ▶ $4=(100)$ $C[4]=A[1]+A[2]+A[3]+A[4];$
- ▶ $5=(101)$ $C[5]=A[5];$
- ▶ $6=(110)$ $C[6]=A[5]+A[6];$
- ▶ $7=(111)$ $C[7]=A[7];$
- ▶ $8=(1000)$ $C[8]=A[1]+A[2]+A[3]+A[4]+A[5]+A[6]+A[7]+A[8];$

$C[i]=A[i-2^k+1]+A[i-2^k+2]+.....A[i];$ k 为 i 的二进制中从最低位到高位连续零的长度

区间查询

```
int getsum(int x){  
    int res = 0;  
    while(x>0) {  
        res=res+c[x];  
        x=x-lowbit(x);  
    }  
    return res;  
}
```


区间查询

树状数组只支持前缀查询，要求信息可减，才能把区间查询转化为前缀查询

$$\text{sum}[7] = C[4] + C[6] + C[7]$$

很自然的只要每次下标减去lowbit求和

```
int Sum(int x)
{
    int Ans=0;
    for(i=x; i>0; i-= lowbit(i)) Ans+=c[i];
    return Ans;
}
```

单点修改

考虑哪些 $c[i]$ 包含了 $a[i]$

10110100

10111000

11000000

100000000

只要对 x 每次加上 lowbit 的位置修改即可

```
void Modify(int x,int d)
{ for(int i=x;i<=n;i+=lowbit(i))
    c[i]+=d;
}
```




很明显修改和询问代价不超过二进制位数

时间复杂度 $O(n \log n)$

空间复杂度 $O(n)$

非常好背写，长度极短，很实用，方便推广到高维

但是功能很有限

单点查询区间修改

改为维护差分就好了

区间查询区间修改

可以做但没有必要

有兴趣的同学可以去了解一下

求逆序对

离散化以后对权值建bit按下标依次插入查询

二维数点

一种非常常见且重要的应用

二位平面上有一些点，给出坐标，每次询问一个区间里的点的数量或价值和。

因为没有修改可以离线。

每个询问矩阵按x轴拆成2个前缀和询问

把点和询问按x轴排序，只要对y轴建立树状数组就可以维护并回答询问

总复杂度 $O(n\log n)$

校门外有很多树，有苹果树，香蕉树，有会扔石头的，有可以吃掉补充体力的.....

如今学校决定在某个时刻在某一段种上一种树，保证任一时刻不会出现两段相同种类的树，现有两个操作：

K=1，读入 l, r 表示在区间 $[l, r]$ 中种上一种树，每次操作种的树的种类都不同

K=2，读入 l, r 表示询问 $l \sim r$ 之间能见到多少种树

($l, r > 0$)

HNOI2012双十字

```
.....1.... .....1.... .....1.... .....1.... .....1....  
...111... ...111... ...111... ...111... ..11111..  
.....1.... .....1.... .....1.... .....1.... .....1....  
..11111.. ..11111.. .....1.... .....1.... .....1....  
.....1.... .....1.... ..11111.. .11111111 .11111111  
.....1.... .....1.... .....1.... .....1.... .....1....
```

- 两条水平的线段不能在相邻的两行。
- 竖直线段上端必须严格高于两条水平线段，下端必须严格低于两条水平线段。
- 竖直线段必须将两条水平线段严格划分成相等的两半。
- 上方的水平线段必须严格短于下方的水平线段。

现在给定一个 $R \times C$ 的 01 矩阵，要求计算出这个 01 矩阵中有多少个双十字。

数据保证 $R, C, N \leq 10,000, R \times C \leq 1,000,000$ 。

数据保证 $R, C, N \leq 10,000$ 首先预处理出 $L[i, j]$ 、 $R[i, j]$

它们分别表示 (i, j) 格子向左方向延伸、向右方向延伸、向下方向延伸的连续的'1'格子的最大长度。

然后求出 $C[i, j] = \min\{L[i, j], R[i, j]\}$

接着我们遍历棋盘中的每个格子 (i, j) 先遍历列 j ，再遍历行 i ，并时刻维护一个值 top ，表示当前连续的'1'的顶端的行标，并维护一个栈保存的是当前的竖线的点。

$$\sum_{len=1}^{C[i]} len * \min\{len - 1, C[j]\} * (top - j) * down[i]$$

Min拆开讨论用bit维护上方的 c 的和

线段树

二叉搜索树，节点储存区间信息

节点 $[l, r]$ 有两个孩子 $[l, \text{mid}]$ 与 $[\text{mid}+1, r]$

叶子节点是单点

常用堆式储存即点 i 的孩子编号为 $2i, 2i+1$

N 是2的幂时节点数为 $2n-1$

值得注意的是由于 n 不总是2的幂，可以达到 $3n+k$

所以一般开4倍数组


信息合并

线段树维护的信息一定是可加（不要求可减）的
知道左右两个区间的信息就可以得到整个区间的信息
（因此线段树可以维护非常多样的信息）


```
void build(int l,int r,int k)
{
    tree[k].l=l;tree[k].r=r;
    if(l==r)
    {
        init(tree[k].w);
        return ;
    }
    int m=(l+r)/2;
    build(l,m,k*2);
    build(m+1,r,k*2+1);
    tree[k].w=tree[k*2].w+tree[k*2+1].w;
}
```


单点修改

先递归修改对应的子区间
然后合并信息



```
void add(int k,int x)
{
    if(tree[k].l==tree[k].r)
    {
        tree[k].w+=y;
        return;
    }
    int m=(tree[k].l+tree[k].r)/2;
    if(x<=m) add(k*2);
    else add(k*2+1);
    tree[k].w=tree[k*2].w+tree[k*2+1].w;
}
```

区间查询

如果查询区间与节点区间一致直接返回信息

否则把查询区间拆成两份递归给子区间

然后把得到的两个区间的答案合并起来

这个过程等价于把一个区间不重不漏的拆成线段树上的节点

考虑一下这个过程的复杂度

这个区间只会被裂成两份一次

剩下的过程都是向一个子节点递归

```
void sum(int k,int x,int y)
{
    if(tree[k].l>=x&&tree[k].r<=y)
    {
        ans+=tree[k].w;
        return;
    }
    int m=(tree[k].l+tree[k].r)/2;
    if(x<=m) sum(k*2,x,y);
    if(y>m) sum(k*2+1,x,y);
}
```

区间修改


不好直接做因为被修改区间包含的所有区间数量是 $O(n)$ 的

引入lazy-tag，用和区间查询一样的方式遍历节点，更新它们的信息，并给它们打上tag

tag是自上而下传递的

当要访问x的孩子的时候应该用x的tag更新他的孩子并把tag推给它的孩子（对所有查询操作而言）


```
void add(int k,int a,int b)
{
    if(tree[k].l>=a&&tree[k].r<=b) {
        tree[k].w+=(tree[k].r-tree[k].l+1)*x;
        tree[k].f+=x;
        return;
    }
    if(tree[k].f) down(k);
    int m=(tree[k].l+tree[k].r)/2;
    if(a<=m) add(k*2,a,b);
    if(b>m) add(k*2+1,a,b);
    tree[k].w=tree[k*2].w+tree[k*2+1].w;
}
```

```
void down(int k)
{
    tree[k*2].f+=tree[k].f;
    tree[k*2+1].f+=tree[k].f;
    tree[k*2].w+=tree[k].f*(tree[k*2].r-tree[k*2].l+1);
    tree[k*2+1].w+=tree[k].f*(tree[k*2+1].r-tree[k*2+1].l+1);
    tree[k].f=0;
}
```



可以发现进行区间修改的时候由于tag的要求，线段树能解决的问题比单点修改少很多

Tag需要可以叠加否则推标记复杂度会不对

Tag对整个区间的影响可以快速算出

区间+、赋值、加等差数列、去反、求minmaxsum等操作都符合要求

另外需要考虑tag之间的顺序影响

常见操作

区间字符串哈希

区间最长连续0

区间最大连续子段

维护dp

BZOJ2957

小A在平面上 $(0,0)$ 点的位置，第 i 栋楼房可以用一条连接 $(i,0)$ 和 (i,H_i) 的线段表示，其中 H_i 为第 i 栋楼房的高度。如果这栋楼房上任何一个高度大于0的点与 $(0,0)$ 的连线没有与之前的线段相交，那么这栋楼房就被认为是可见的。

施工队的建造总共进行了 M 天。初始时，所有楼房都还没有开始建造，它们的高度均为0。在第 i 天，建筑队将会将横坐标为 X_i 的房屋的高度变为 Y_i (高度可以比原来大---修建，也可以比原来小---拆除，甚至可以保持不变---建筑队这天什么事也没做)。请你帮小A数数每天在建筑队完工之后，他能看到多少栋楼房？

如果一个楼房能被看见的话，那么他楼的最高点到 $(0, 0)$ 直线的斜率一定严格大于他前面所有直线的斜率，也就是 h_i/x_i 的值是递增的。

左孩子的答案一定合法

对右孩子考虑左孩子最大值的影响

当前区间的前方最大值为 mx

如果左孩子的最大值小于 mx 递归右孩子

否则对右孩子而言 mx 没有意义前方的真正的最大值在左孩子内部，这个之前维护好了 $(f[x]-f[2x])$

可以发现查询的复杂度是 \log 的而询问每次更新左孩子后都要重新计算右孩子来更新本节点总复杂度 $O(n\log^2)$

矩形面积并

矩形按x轴离散化排序拆成进入和离开的时刻
只要对y轴建线段树做区间加和

线段树优化建图

比如网络流一个点向区间内点连边可以建树优化

bzoj3638

给出一个长度为 n 的序列， $a_1 \sim a_n$ ，和 m 次操作，每次操作分为
0 x val，将 a_x 变成 val

1 l r k，询问在区间 $l \sim r$ 中，最多 k 个不重合区间的最大和是多少。

$n, m \leq 10^5$ ， $|a_i| \leq 500$ ，1操作 ≤ 10000 ， $|val| \leq 500$ ， $1 \leq k \leq 20$



直接做要维护区间内有 k 个段且头尾有连续段的信息
合并的时候要做背包状更新
更新复杂度 $O(k^2)$



从费用流的角度思考

$N+1$ 点链每个点和ST连边，链上边与原来的点对应

则增广路和不交区间对应

只需增广 k 次

维护区间最大字段和，支持单点修改区间取反

平衡树

二叉排序树

这里介绍treap和splay

用来查k大的平衡树

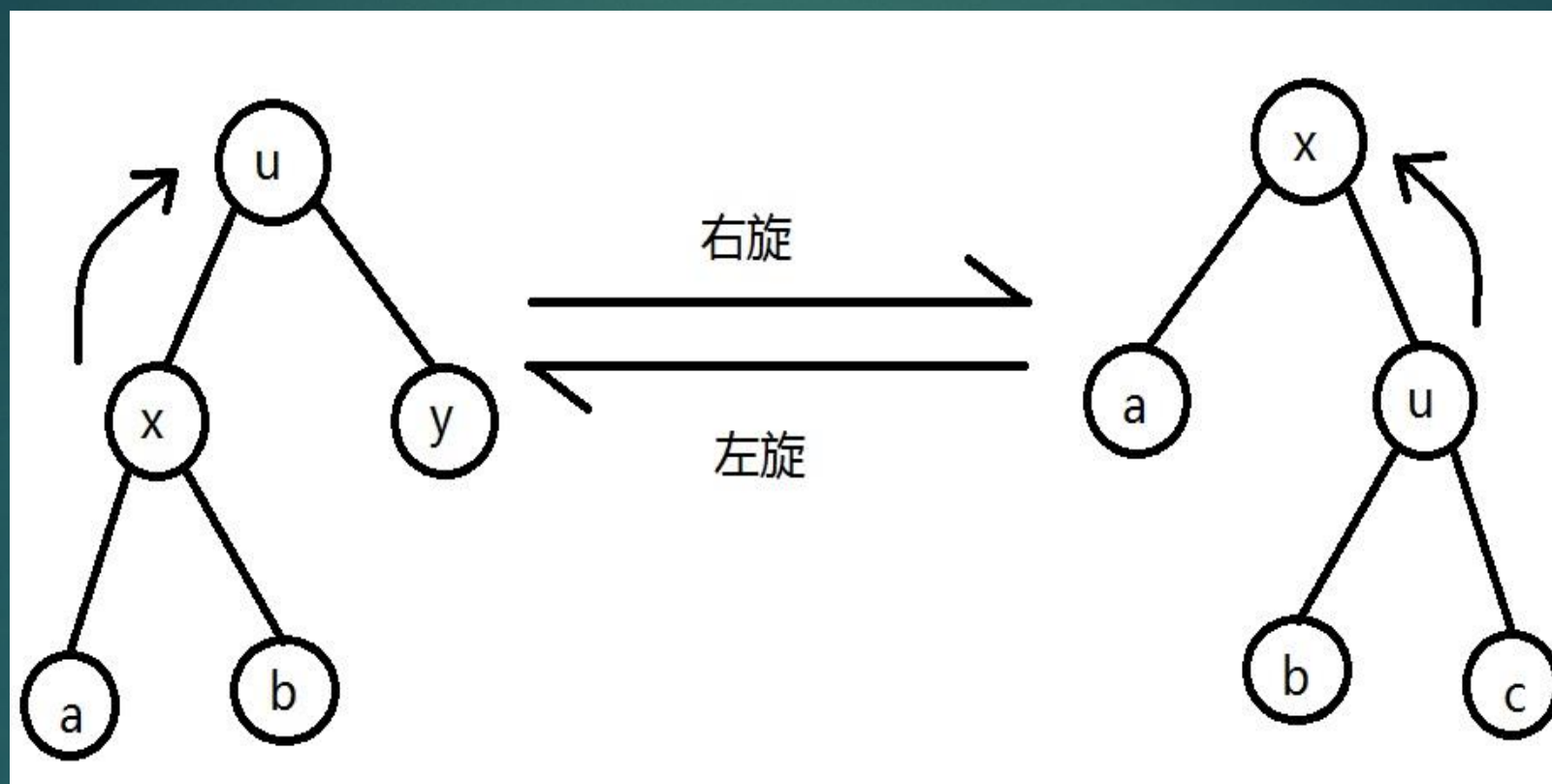
在二叉排序树里进行删除插入找k大数求比x小的数的数量


复杂度都和深度相关

平衡树能保证深度可控，有一些是严格的比如红黑树，avl，sbt等等，而treap的深度是基于随机期望的但是好写

替罪羊树和splay是均摊的

有兴趣可以自己学习一下其他平衡树





Treap给每个节点分配一个随机权并通过旋转保证权值堆式排列
插入操作

只要先插到叶子上然后一步一步向上旋转维持堆性质

删除操作

我们希望把该点转到叶子上删掉

只要先找到该点

把权值较大的孩子转上来直到该点拥有至多一个孩子

替罪羊树

只在子树size不平衡程度大于阈值 α 时暴力重构

重构是把整个子树收集起来然后按线段树状的方式建树

删除为保证复杂度应该把该节点标记为空而不是真的删掉

复杂度保证重构总量和树深度都是 α 相关的对数级别

splay

可以进行区间操作

包含线段树的操作

包括翻转，提取，删除等

区间信息标记和线段树类似的维护

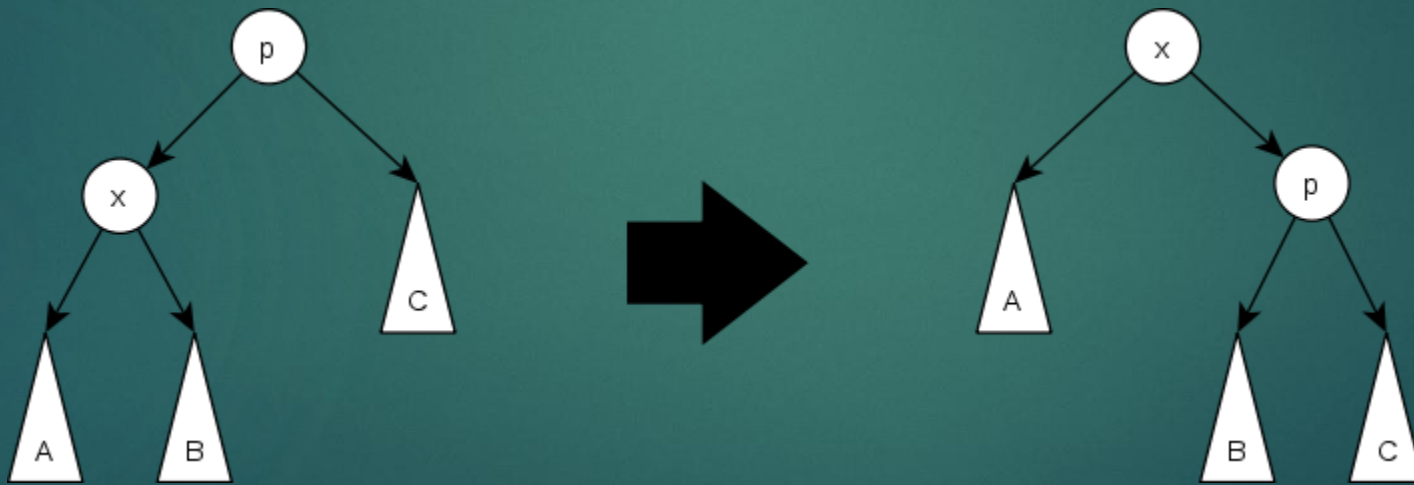
不同的是区间信息变为左孩子 $+x$ +右孩子

Splay不保证树深度但保证 m 个操作的总复杂度

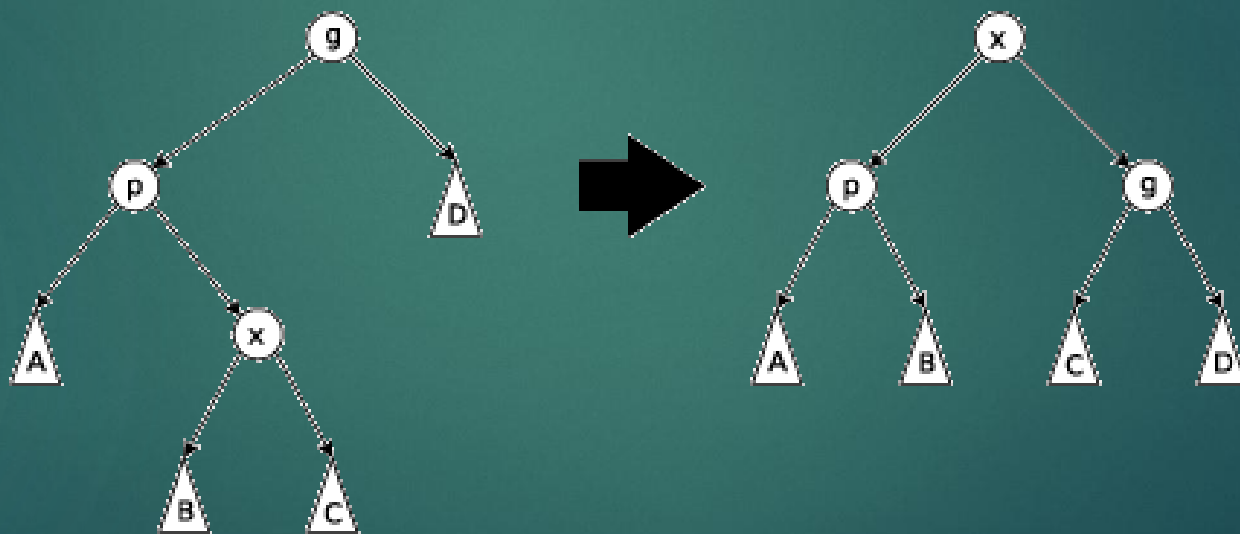


一切操作通过splay操作完成
通过下面的方式把节点转到根上

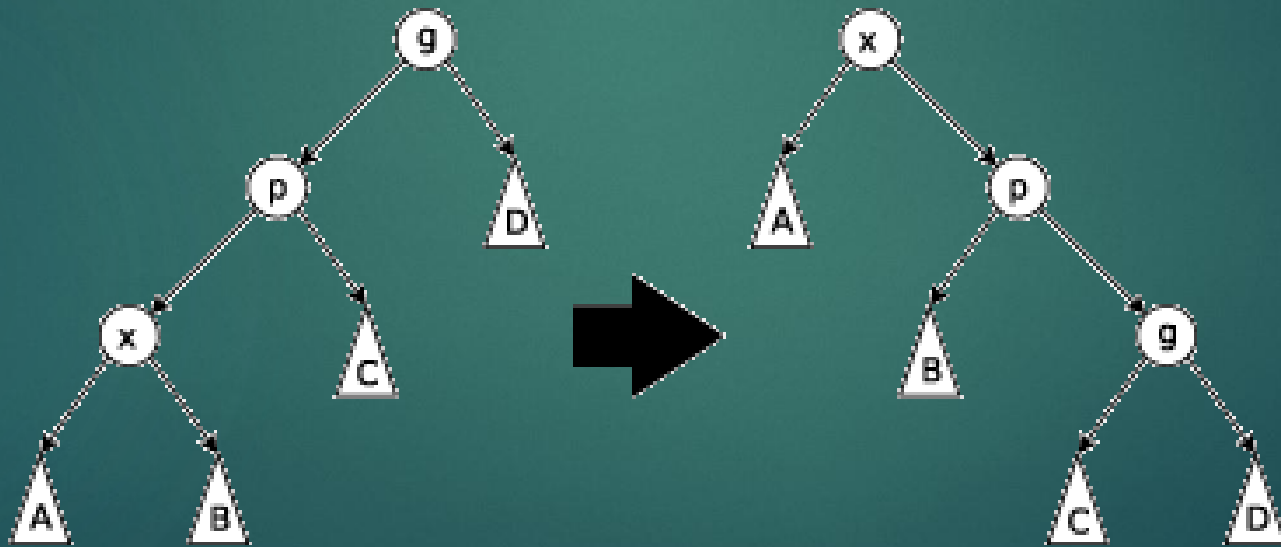
zig: 当目标节点是根节点的左子节点或右子节点时，进行一次单旋转，将目标节点调整到根节点的位置。



zig-zag: 当目标节点、父节点和祖父节点成"zig-zag"构型时，进行一次双旋转，将目标节点调整到祖父节点的位置。



zig-zig: 当目标节点、父节点和祖父节点成"zig-zig"构型时，进行一次zig-zig操作，将目标节点调整到祖父节点的位置。





提取区间可以把左端点-1 splay到根
再把右端点+1 splay到根的右孩子
那么右孩子的左孩子就是所要的区间

翻转标记直接交换左右孩子
这是线段树做不到的

非旋转treap

通过merge操作和split操作维护

Merge根据随机权值决定哪个为根递归合并

Split直接按类似找k大的做法找出分离的链然后回溯的时候更新孩子

复杂度是期望的

可以持久化

```

inline int merge(int a,int b){
    if(!a)return b;if(!b)return a;
    if(rnd[a]<rnd[b]){
        rs[a]=merge(rs[a],b);
        update(a);return a;
    }else{
        ls[b]=merge(a,ls[b]);
        update(b);return b;
    }
}

```

```

inline pa split(int x,int k){
    if(!x)return pa(0,0);pa y;
    if(siz[ls[x]]>=k){
        y=split(ls[x],k);
        ls[x]=y.sec;
        update(x);
        y.sec=x;
    }else{
        y=split(rs[x],k-siz[ls[x]]-1);
        rs[x]=y.fst;
        update(x);
        y.fst=x;
    }
    return y;
}

```

动态开点线段树

权值太大开不下

不用的节点不开用到才建节点

时空复杂度 $O(n \log n)$

可持久化数据结构

可以发现很多数据结构我们一次操作只会修改 \log 个节点
我们改为新建 \log 个节点，将他们的信息孩子进行计算并记录新root

那么老root和新root是不会互相影响的可以同时自由查询
我们就得到了任意时刻历史版本的信息

时空复杂度相同，比较常见的是可持久化线段树（主席树）

查区间k大可以按权值建树按下标一个个插入线段树

查询可以用两颗树相减得到真实值

线段树上根据左孩子size决定走哪边

```
void update(int l,int r,int x,int &y,int v)
{
    y=++sz;
    sum[y]=sum[x]+1;
    if(l==r)return;
    ls[y]=ls[x];rs[y]=rs[x];
    int mid=(l+r)>>1;
    if(v<=mid)update(l,mid,ls[x],ls[y],v);
    else update(mid+1,r,rs[x],rs[y],v);
}
```


区间中位数

区间是否有数出现次数占一半以上
区间内最小没有出现过的自然数


众数能做吗为什么

mex

每个位置存 i 出现的最右
只要查第 r 棵的第一个小于 l 的下标

数区间颜色数

记个 nxt 表示同颜色的下一次出现位置
用主席树数区间内 nxt 在区间外的个数



记个 nxt 表示同颜色的下一次出现位置
用主席树数区间内 nxt 也在区间内的个数

树上主席树

链上k小值

$$X+Y-\text{lca}(X,Y)-\text{fa}[\text{lca}(X,Y)]$$

可持久化trie

可以求出区间中数形成的trie
[l r]内与x异或值最大的数

树套树

多维信息维护

强制在线的时候只能树套树

区间k大带修改

不带修改就用主席树做

带修改要修改 $T[i]..T[n]$ 的主席树

我们可以外层用树状数组，元素表示对应位置的权值线段树的和

查询修改只要在 \log 棵权值线段树里操作

线段树套平衡树

线段树每个节点储存对应区间的元素的平衡树

单点修改修改 \log 棵平衡树

查询在 \log 棵平衡树内统计信息

和bit套主席树相比空间开销小

区间k大带修改带插入

带插入外层只能平衡树

而且不可以旋转，只能替罪羊

三维偏序

二维偏序我们第一维排序第二维树状数组

三维偏序我们第一维排序第二维树套树

可并堆

支持合并操作的堆

~~你也可以用平衡树启发式合并来达到一样的功能~~

左偏树

节点仍满足堆性质

用merge操作实现所有功能

先根据根的大小决定哪个根是新根，然后递归到子树完成子问题

如何保证递归深度

定义 $dis[i]$ 为 i 向下到可插入位置的最小距离

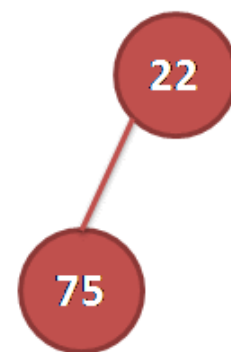
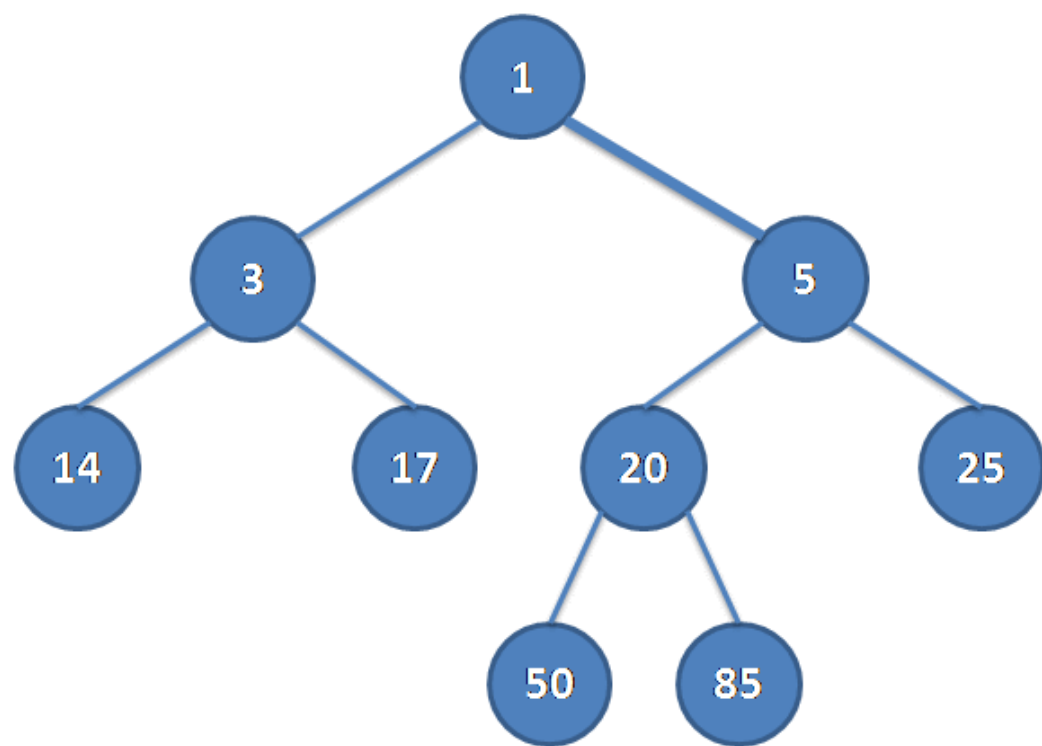
现在要求左孩子 dis 大于右孩子 dis

维护只需要回溯的时候交换左右孩子

每次向右递归那么代价就是两根 dis 之和

在形成满二叉树时 dis 最大由此保证复杂度 $O(n \log n)$

能可持久化



```
struct Ltree{
    int dist[maxn],key[maxn],l[maxn],r[maxn];
    int merge(int x,int y){
        if(x==0 || y==0) return x+y;
        if(key[x]<key[y]) swap(x,y);
        r[x]=merge(r[x],y);
        if(dist[r[x]]>dist[l[x]]) swap(l[x],r[x]);
        dist[x]=dist[r[x]]+1
        return x;
    }
    void pop(int &x){x=merge(l[x],r[x]);}
    int top(int x){return key[x];}
}It;
```


点分治

一般用于处理路径问题



重心：最大孩子size最小，保证最大的孩子siz至多是 $n/2$
在树上统计siz，上方的子树siz为 $n - \text{siz}[x]$

每次统计经过当前根的路径

先找到重心

然后一个一个子树dfs处理当前子树和前面所有子树的点对形成的链的关系

删掉当前根对每个子树递归处理

POJ-1741

一棵有 n 个节点的树，每条边有个权值代表相邻2个点的距离，要求求出所有距离不超过 k 的点 (u,v)

还有一种做法是对一个重心直接dfs出所有深度
直接算所有点对的答案
对每个子树内部算答案减掉

常见操作

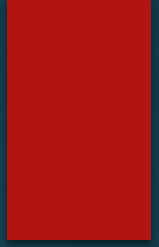
树上路径k大

路径和 $\%3=0$ 的数量

求一条和为k的路径

求所有长度的路径数

bzoj1095



边分治

没什么用

和点分治相比改为选一条边每次计算经过这条边的路径

而且要加虚点转二叉树不然菊花图爆炸

一般都能用点分治实现

动态点分治

点信息修改

bzoj1095

还是一棵不会动的树，每个点有一个颜色（黑色或白色），一开始全是黑色，有一些修改和询问，修改是将某个点的颜色反转，询问黑点间的最长距离。



重心形成树结构

对每个重心开2个堆

A维护这个重心联通块所有黑点的到重心距离

B存所有子树A堆顶


再开全局堆存所有B的最大值次大值

重心的树深度 \log 每次修改只改 \log 个堆

Zjoi2015幻想乡战略游戏

给你一棵树，多次改变一个点的权值或询问所有点到当前带边权点权重心的带权距离是多少。

度 <20



假设当前补给站为 u ，并强制以 u 为根， v 为 u 的一个子节点， $sumd_u$ 和 $sumd_v$ 分别为 u 的子树内的 d 之和以及 v 的子树内的 d 之和， $len(u,v)$ 为边 (u,v) 的长度。

如果将补给站迁移到点 v ，那么 v 的子树内的点到补给站的距离减少了 $len(u,v)$ ，其他的点到补给站的距离增加了 $len(u,v)$ 。也就是说，补给站迁移到点 v 时，代价的增量为：

$$len(u,v) - (sumd_u - sumd_v - sumd_v)$$

只有一个孩子可能是更优的

度数限制保证可以枚举孩子

直接沿着走复杂度不对

在点分树上走到对应分治层就好了

WC2014紫荆花之恋

在一棵树上不断加入一个节点[共 $n \leq 10^5$ 个]

给出 $\{a[\text{在树上的父亲}], c[\text{与父亲的距离}], r\}$

并询问此时共多少对节点满足 $\text{dist}(i, j) \leq r_i + r_j$ ，强制在线。



一次询问点分治

不强制在线只要建出点分树在主席树里查询

强制在线插入节点后要在点分树的根到叶子全进行更新

每个分治中心用平衡树维护

但是插入多了点分树就不平衡了

替罪羊重构

树链剖分

将树分成若干链

链内的边叫重边

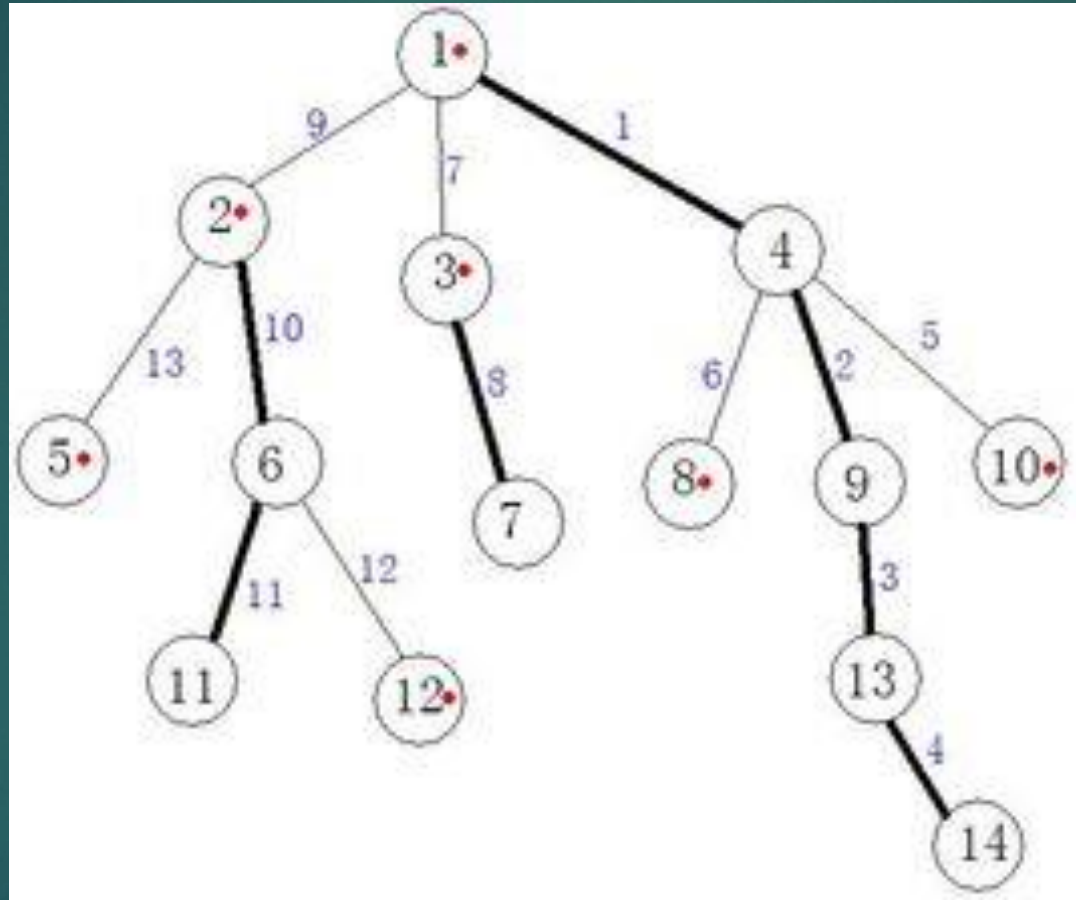
链间的叫轻边

每个节点siz最大的孩子是重孩子连重边其他连轻边

用两个dfs预处理链剖方式

显然到根轻边数量为 \log

可以用来求lca比倍增常数小



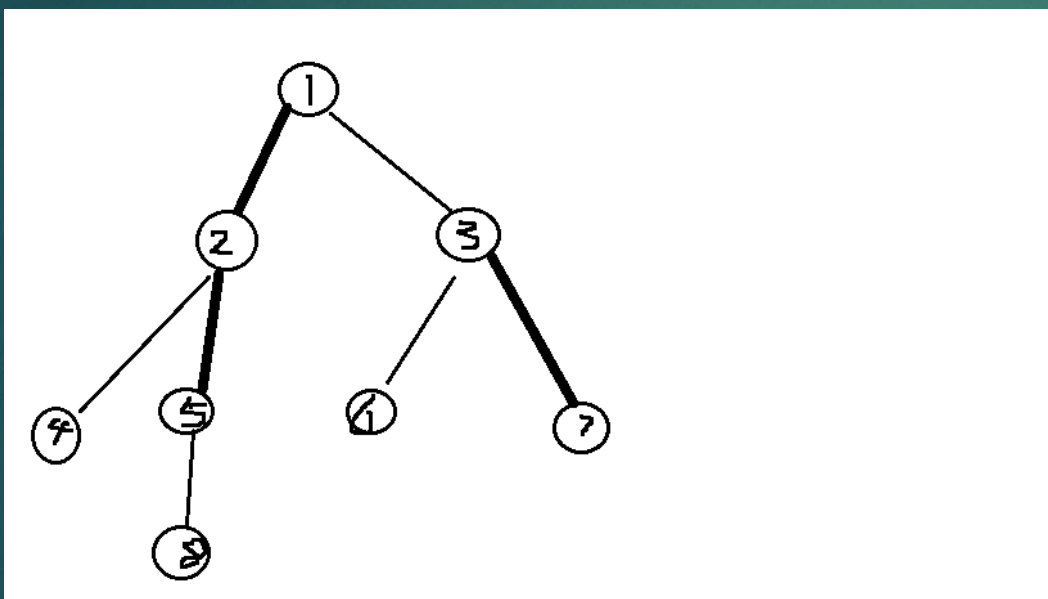



那么一条链的信息可以用线段树甚至平衡树甚至树套树维护

所有序列问题都可以+一个 \log 用连剖搬到静态树上

动态树LCT

相比链剖可以支持删边加边换根等操作





Lct与链剖一样分虚边实边实边连向偏爱儿子
实边的链用splay维护称为辅助树

（实现中辅助树的根的父亲是原来链顶的父亲，但儿子关系不成立，是实现上更方便的trick）

操作access

让x节点不含偏爱儿子且到根均为实边

具体只要把x splay到根然后断右子树

顺着往上走并把之前的辅助树和上方链的辅助树合并



操作makeroot

先access (x) 然后只要翻转x与root的链上的偏爱路径就好了

Link cut 链修改操作只要makeroot access就能简单的完成

判断联通性access 然后splay到根往左孩子走到底就得到了当前根

Tip 一切暴力找的节点要splay才保证复杂度

根据势能分析总复杂度是 $n \log n$ 如果把splay换其他平衡树则会变为 $n \log^2 n$

有兴趣的同学自行了解证明

加边mst

边建成点

每次加入一条边

不成环直接加

成环就查链上最大值如果比新加边打

Cut最大边 link新边

Noi2014 魔法森林

无向图

边有a b两个权

从1到n

代价是路径上的 $\max\{a_i\} + \max\{b_i\}$

枚举答案中的a对
b维护mst



支持加边

询问a b最早什么时候联通

边权为时刻求链max

Top tree

不建议学

思想是同时用数据结构（splay 不加 复杂度但是 更难写）维护虚边信息
支持子树操作

启发式合并

合并2个数据结构可以把size较小的中的所有元素依次插入到大的里

每个元素只会被插入 \log 次，只需要付出额外 \log 的代价

处理一些子树问题对轻重链的合并就是这种思想的

线段树合并

如果一节点在两颗线段树里都有就合并该节点的信息并递归到左右孩子，否则停下

每次代价是两线段树的交的size

可以发现这个过程代价不大于直接插入

所以复杂度是 $O(n \log n)$

简单的字符串科技



hash

用于快速比较字符串是否相同

给出一个映射 f 把大量的信息映射到小的值域里

必须满足 $x_1 = x_2$ 则 $f(x_1) = x_2$

由于存在信息丢失，不能保证

$x_1 \neq x_2$ 则 $f(x_1) \neq x_2$

但是可以构造好的哈希函数使冲突发生概率小

BKDR

$$h[i] = (h[i-1] * b + a[i] + k) \% \text{mod}$$

不取mod就是自然溢出(有的出题人就爱卡这个
相当于用b进制在模意义下映射


好处是信息可减可加, 预处理h以后可以 $O(1)$ 得到任意的子串哈希

带修改可以在线段树里操作

冲突处理

对于每个哈希值开一个链表

在链表里存真实值（或者模longlong的哈希值）



根据生日攻击，映射的值域为 n ，存储的信息达到 $n^{1/2}$ 就有接近一半的概率产生至少一次冲突，因此oi中一般认为 10^7 级别的元素用longlong模是安全的
类似的取多个模数相当于值域空间乘起来

KMP

线性字符串匹配

Hash也行



查找a在b中的出现

维护一个p数组p[i]满足a的前i位形成的串前后p[i]个字符相同

称为border

P的维护只要比对下一位，合法+1，不合法就用p[i]往回跳

匹配ab也一样

P上跳的复杂度均摊线性

```
scanf("%s" , a + 1);scanf("%s" , b + 1);la = len(a) , lb = len(b);
p[1] = 0;
for (j = 0 , i = 2; i <= lb; i ++){
    while (j && b[i] != b[j + 1]) j = p[j];
    if (b[i] == b[j + 1]) j ++;
    p[i] = j;
}
j = 0;
for (i = 1; i <= la; i ++){
    while (j && a[i] != b[j + 1]) j = p[j];
    if (a[i] == b[j + 1]) j ++;
    if (j == lb){
        printf("%d\n" , i - lb + 1);
        j = p[j];
    }
}
```

AC自动机

~~不能帮你自动AC~~

完成多模匹配，字典里多个词，求匹配串里每个词出现情况

思想与kmp相似



先对所有模式串建trie

然后进行bfs求fail指针

Fail[x]表示x下一位如果失配跳到的当前状态的最长后缀

Fail[ch[x][i]]可以通过ch[fail[x]][i]得到

如果不存在i的边就沿着fail跳

另一种写法是直接把x节点下一个字符的所有转移算出来



自动机构造完了就可以直接在自动机上跑着匹配了
统计答案要沿着fail跳到底

由于所有模式串的长度一定，复杂度不会高于线性
还能做一些其他的问题

POI2000病毒

有若干种病毒，每种病毒的特征代码都是一个01串。

每个程序也都是一个01串。

问是否存在不被病毒感染（不包含任何病毒的特征代码）的无限长的程序。



把病毒串的ac自动机建出来

串末端位置标记为危险

Fail会指向危险节点的节点也要标记因为实际上已经不合法了

危险节点不能走

只要自动机上跑出环就可以

NOI2011阿狸的打字机

打字机上只有28个按键，分别印有26个小写英文字母和' B'、' P'两个字母。经阿狸研究发现，这个打字机是这样工作的：·输入小写字母，打字机的一个凹槽中会加入这个字母(这个字母加在凹槽的最后)。·按一下印有' B'的按键，打字机凹槽中最后一个字母会消失。·按一下印有' P'的按键，打字机会在纸上打印出凹槽中现有的所有字母并换行，但凹槽中的字母不会消失。例如，阿狸输入aPaPBbP，纸上被打印的字符如下：a aa ab 我们把纸上打印出来的字符串从1开始顺序编号，一直到n。打字机有一个非常有趣的功能，在打字机中暗藏一个带数字的小键盘，在小键盘上输入两个数(x,y)（其中 $1 \leq x, y \leq n$ ），打字机会显示第x个打印的字符串在第y个打印的字符串中出现了多少次。阿狸发现了这个功能以后很兴奋，他想写个程序完成同样的功能，你能帮助他么？



首先可以按照操作树建trie

建出ac自动机

Fail关系一定是树形的，建出fail树

X在y中出现就是x是y的前缀的后缀

根到y的路径的点fail树上的x出现次数

离线对y进行遍历在fail树里修改

对x进行子树查询（树状数组dfs序）

BZOJ4231 : 回忆树

字符串在树上路径出现次数

1.在LCA处转弯，那么这种情况只有 $O(|S|)$ 次，暴力提取出长度为 $2|S|$ 的链进行KMP即可。

2.不转弯，那么可以拆成两个到根路径的询问。

对所有串的正反串建立AC自动机，求出fail树上每个点的DFS序。

然后DFS原树，记录在AC自动机上走到了哪个点，在那个点+1，回溯的时候-1。

那么一个询问的答案就是fail树上的子树和，树状数组维护即可。

后缀数组

我们记 $\text{suffix}(i)$ 表示从原字符串第 i 个字符开始到字符串结尾的后缀。我们把它所有的后缀拿出来按字典序排序

把排好序的数组记作 $\text{sa } 7 \ 6 \ 4 \ 2 \ 1 \ 5 \ 3$

同时 rank 数组表示后缀 i 表示 i 在 sa 中的排名。

也就是说如果 $\text{sa}[i]=j$ 那么 $\text{rank}[j]=i$

我们现在令 $\text{height}[i]$ 是 $\text{suffix}(\text{sa}[i-1])$ 和 $\text{suffix}(\text{sa}[i])$ 的最长公共前缀长度，即排名相邻的两个后缀的最长公共前缀长度。

比如 $\text{height}[4]$ 就是 anana 和 ana 的最长公共前缀，也就是 ana ，长度为3。

后缀	i
\$	7
a\$	6
ana\$	4
anana\$	2
banana\$	1
na\$	5
nana\$	3



这个height数组有一个神奇的性质：若 $\text{rank}[j] < \text{rank}[k]$ ，则后缀 $S_{j..n}$ 和 $S_{k..n}$ 的最长公共前缀为 $\min(\text{height}[\text{rank}[j]+1], \text{height}[\text{rank}[j]+2] \dots \text{height}[\text{rank}[k]])$ 。这个性质是显然的，因为我们已经后缀按字典序排列。




同时，我们还有一个结论： $\text{height}[\text{rank}[i]] \geq \text{height}[\text{rank}[i-1]] - 1$
证明：

设 $\text{suffix}(k)$ 是排在 $\text{suffix}(i-1)$ 前一名的后缀，则它们的最长公共前缀是 $\text{height}[\text{rank}[i-1]]$

那么 $\text{suffix}(k+1)$ 将排在 $\text{suffix}(i)$ 的前面

并且 $\text{suffix}(k+1)$ 和 $\text{suffix}(i)$ 的最长公共前缀是 $\text{height}[\text{rank}[i-1]] - 1$

所以 $\text{suffix}(i)$ 和在它前一名的后缀的最长公共前缀至少是
 $\text{height}[\text{rank}[i-1]] - 1$



这样我们按照 $\text{height}[\text{rank}[1]], \text{height}[\text{rank}[2]] \dots \text{height}[\text{rank}[n]]$ 的顺序计算，利用height数组的性质，就可以将时间复杂度可以降为 $O(n)$ 。这是因为height数组的值最多不超过 n ，每次计算结束我们只会减1，所以总的运算不会超过 $2n$ 次。

这里我们注意到，所有后缀子串的前缀子串即为原串的所有子串。那么我们就可以利用height数组解决很多问题。

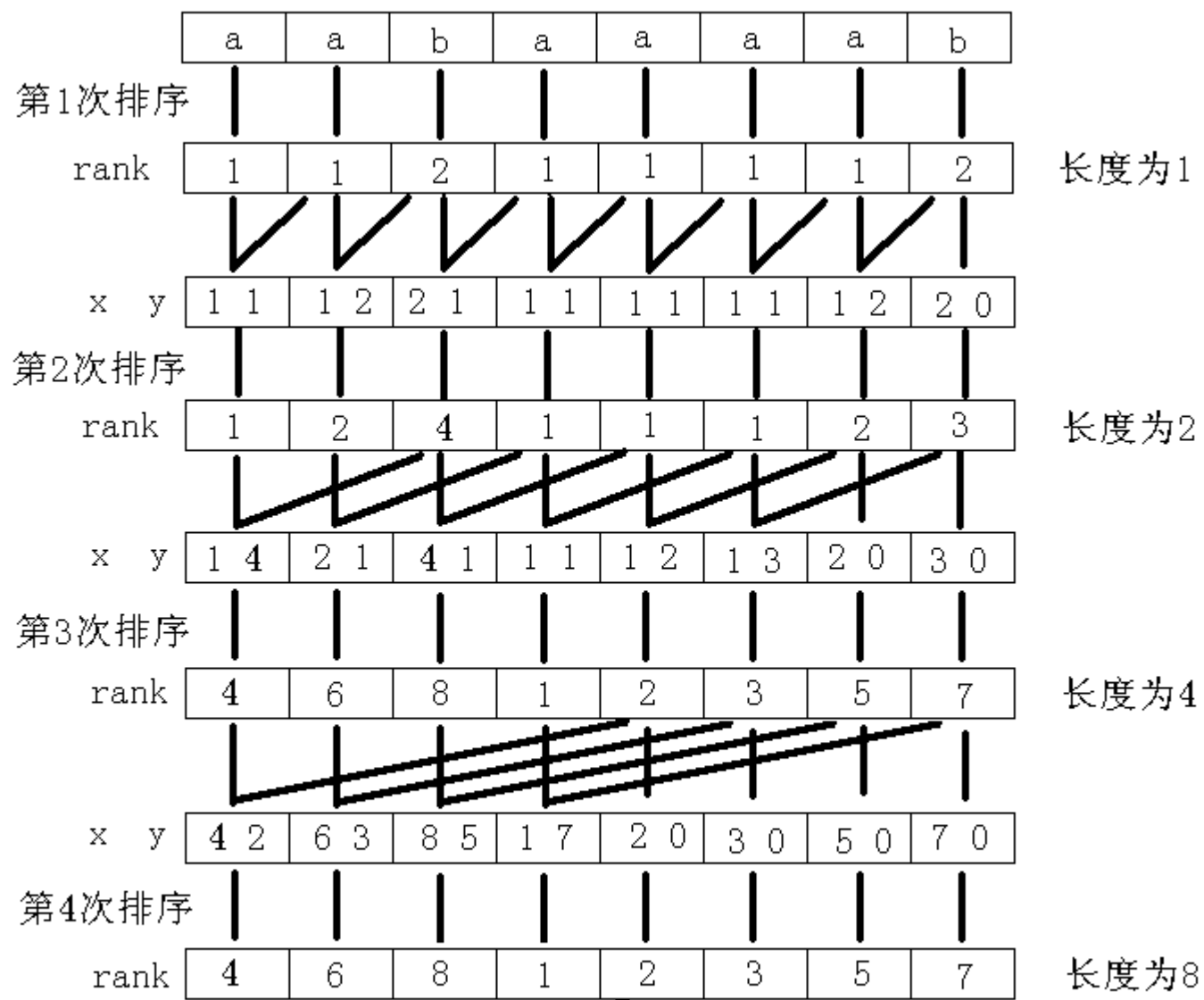


现在我们来考虑后缀数组（rank 与 sa）怎么求

对长度为 2^0 的字符串，也就是所有单字母排序。用长度为 2^0 的字符串，对长度为 2^1 的字符串进行双关键字排序。

考虑到时间效率，我们一般用基数排序。

用长度为 2^{k-1} 的字符串，对长度为 2^k 的字符串进行双关键字排序。直到 $2^k \geq n$ ，或者名次数组 Rank 已经从 1 排到 n ，得到最终的后缀数组



不相同的子串的个数

Height数组是相同子串数，补集一下

Lcp最长公共前缀

用rmq维护height

最长公共子串

把 n 个串用分隔符分开接一起

建sa二分答案

只需要有连续一段 $\text{height} > k$ 且sa来自所有 n 个串就合法

可重叠的k次最长重复子串

二分答案

判断是否 $>mid$ 的height有连续k个

不可重叠最长重复子串

同上还要判断sa的间距是否合法



把所有后缀插进trie得到了后缀树

后缀树的dfs序就是后缀数组

线性构造后缀树可以用sam或者ukk

品酒大会 (NOI2015)

给出一个长度为 n 的字符串，每一位有一个权值 val 。定义两个位字符为 r 相似，是指分别从这两个字符开始，到后面的 r 个字符都相等。两个 r 相似的字符还有一个权值为这两个字符权值的乘积。问对于 $r = 0, 1, 2, \dots, n - 1$ ，统计出有多少种方法可以选出 2 个“ r 相似”的字符，并回答选择 2 个“ r 相似”的字符可以得到的权值的最大值。



把height数组排序

从小到大加入并查集

同时维护最大值最小值（可能有负）

合并时更新答案

NOI 2016 优秀的拆分

如果一个字符串可以被拆分为 $AABB$ 的形式，其中 A 和 B 是任意非空字符串，则我们称该字符串的这种拆分是优秀的。

例如，对于字符串 `aabaabaa`，如果令 $A=aab$ ， $B=a$ ，我们就找到了这个字符串拆分成 $AABB$ 的一种方式。一个字符串可能没有优秀的拆分，也可能存在不止一种优秀的拆分。

比如我们令 $A=a$ ， $B=baa$ ，也可以用 $AABB$ 表示出上述字符串；但是，字符串 `abaabaa` 就没有优秀的拆分。现在给出一个长度为 n ($n < 30000$) 的字符串 S ，我们需要求出，在它所有子串的所有拆分方式中，优秀拆分的总个数。这里的子串是指字符串中连续的一段。

以下事项需要注意：1) 出现在不同位置的相同子串，我们认为不同的子串，它们的优秀拆分均会被记入答案。2) 在一个拆分中，允许出现 $A=B$ 。例如 `cccc` 存在拆分 $A=B=c$ 。3) 字符串本身也是它的一个子串。

形如AA的串为square

设 f_i 表示以 i 结尾的square个数， g_i 表示以 i 开头的square个数，则 $ans = \sum f[i]g[i+1]$ 。

枚举square中长度的一半 L ，每 L 步取一个关键点，那么每个该长度的square的肯定恰好经过两个相邻的关键点，且位置差距为 L 的字符一一匹配。所以相邻关键点之间通过后缀数组求出最长公共前后缀，即可知道存在多少该长度的square。

这相当于 f 和 g 的一段区间+1，用bit维护
时间复杂度 $O(n \log n)$ 。

THANKS