ACM模板

[数据结构 4](#_Toc88313548)

[ST表 4](#_Toc88313549)

[并查集 5](#_Toc88313550)

[树状数组 5](#_Toc88313551)

[二维树状数组 6](#_Toc88313552)

[单点修改，区间查询 6](#_Toc88313553)

[区间修改,单点查询 7](#_Toc88313554)

[区间修改，区间查询 8](#_Toc88313555)

[线段树 9](#_Toc88313556)

[区间赋值区间和 9](#_Toc88313557)

[扫描线 10](#_Toc88313558)

[矩形周长并 10](#_Toc88313559)

[矩形K次以上面积并 12](#_Toc88313560)

[线段树合并 14](#_Toc88313561)

[线段树分裂 15](#_Toc88313562)

[兔队线段树 18](#_Toc88313563)

[SegmentTree beats 21](#_Toc88313564)

[可持久化线段树 30](#_Toc88313565)

[静态区间第k小 30](#_Toc88313566)

[区间不同数个数 31](#_Toc88313567)

[区间修改，标记永久化 33](#_Toc88313568)

[树套树 35](#_Toc88313569)

[树状数组套可持久化线段树 区间k大修改 35](#_Toc88313570)

[分块 37](#_Toc88313571)

[区间众数 强制在线 37](#_Toc88313572)

[带插入区间k大 40](#_Toc88313573)

[莫队 43](#_Toc88313574)

[静态莫队 43](#_Toc88313575)

[回滚莫队 44](#_Toc88313576)

[可删堆 47](#_Toc88313577)

[可并堆 47](#_Toc88313578)

[左偏树 48](#_Toc88313579)

[整体二分 50](#_Toc88313580)

[区间带修第k小 50](#_Toc88313581)

[树论 53](#_Toc88313582)

[dfs序与欧拉序 53](#_Toc88313583)

[LCA 54](#_Toc88313584)

[RMQ 55](#_Toc88313585)

[树剖详见重剖板子 56](#_Toc88313586)

[Kruskal重构树 56](#_Toc88313587)

[重链剖分 57](#_Toc88313588)

[长链剖分 61](#_Toc88313589)

[树上k级祖先 61](#_Toc88313590)

[长链剖分优化DP模板 63](#_Toc88313591)

[长链剖分+后缀和套路 64](#_Toc88313592)

[点分治 66](#_Toc88313593)

[容斥 66](#_Toc88313594)

[枚举子树 68](#_Toc88313595)

[点分+dfs序上dp 70](#_Toc88313596)

[点分+FFT统计树上所有路径条数 72](#_Toc88313597)

[点分树 75](#_Toc88313598)

[Dsu on Tree 82](#_Toc88313599)

[图论 84](#_Toc88313600)

[二分图 84](#_Toc88313601)

[二分图最大匹配 84](#_Toc88313602)

[匈牙利 84](#_Toc88313603)

[HK算法 84](#_Toc88313604)

[二分图最大权匹配 87](#_Toc88313605)

[字符串 89](#_Toc88313606)

[Kmp 89](#_Toc88313607)

[Exkmp 90](#_Toc88313608)

[Trie 91](#_Toc88313609)

[序列自动机 92](#_Toc88313610)

[Ac自动机 92](#_Toc88313611)

[后缀数组 94](#_Toc88313612)

[后缀自动机 98](#_Toc88313613)

[广义后缀自动机 99](#_Toc88313614)

[动态规划 103](#_Toc88313615)

[数位DP 103](#_Toc88313616)

[线性代数 104](#_Toc88313617)

[矩阵快速幂 104](#_Toc88313618)

[线性BM递推 105](#_Toc88313619)

[高斯消元 108](#_Toc88313620)

[矩阵求逆与行列式值 O(n^3) 110](#_Toc88313621)

[线性基 113](#_Toc88313622)

[数论 119](#_Toc88313623)

[质数 119](#_Toc88313624)

[n以内的质数个数 lehmer\_pi O(n^(2/3)) 119](#_Toc88313625)

[反素数 121](#_Toc88313626)

[Miller Robin and pollard\_rho 121](#_Toc88313627)

[筛法 123](#_Toc88313628)

[质数线性筛法及常见函数筛 123](#_Toc88313629)

[Min25筛 127](#_Toc88313630)

[杜教筛 129](#_Toc88313631)

[Dirichlet 前缀和 131](#_Toc88313632)

[常用Dirichlet卷积 132](#_Toc88313633)

[约数 133](#_Toc88313634)

[约数处理 133](#_Toc88313635)

[GCD与LCM 133](#_Toc88313636)

[同余 134](#_Toc88313637)

[快速幂与乘求模 134](#_Toc88313638)

[逆元相关 135](#_Toc88313639)

[Exgcd 135](#_Toc88313640)

[类欧几里德 136](#_Toc88313641)

[CRT 137](#_Toc88313642)

[EXCRT 137](#_Toc88313643)

[阶与原根 139](#_Toc88313644)

[BSGS and EXBSGS 140](#_Toc88313645)

[多项式 144](#_Toc88313646)

[拉格朗日插值 144](#_Toc88313647)

[FFT or NTT 147](#_Toc88313648)

[分治FFT 150](#_Toc88313649)

[Vector全家桶板子 151](#_Toc88313650)

[FWT与子集卷积 157](#_Toc88313651)

[特殊数计算 160](#_Toc88313652)

[两类斯特林数 160](#_Toc88313653)

[贝尔数 163](#_Toc88313654)

[计算几何 163](#_Toc88313655)

[点相关 163](#_Toc88313656)

[线相关 165](#_Toc88313657)

[线段相关 166](#_Toc88313658)

[三角形相关 167](#_Toc88313659)

[多边形相关 168](#_Toc88313660)

[圆相关 170](#_Toc88313661)

[极角排序 177](#_Toc88313662)

[凸包 177](#_Toc88313663)

[旋转卡壳 178](#_Toc88313664)

[半平面交 178](#_Toc88313665)

[三维几何 180](#_Toc88313666)

[球的体积交与并 187](#_Toc88313667)

[其他题 188](#_Toc88313668)

[Hdu6219 最大空凸包 188](#_Toc88313669)

[平面最近点对 190](#_Toc88313670)

[杂项 191](#_Toc88313671)

[快速读入 191](#_Toc88313672)

[\_int128 191](#_Toc88313673)

[常用STL 192](#_Toc88313674)

# 数据结构

## ST表

**静态区间最值查询**

int f[maxn][25];//[i,i+2^j-1]里的数的最大值

int a[maxn],Log[maxn];

void RMQ(){//预处理

for(int i=1;i<=n;++i)f[i][0]=a[i];

for(int j=1;(1<<j)<=n;++j)

for(int i=1;i+(1<<j)-1<=n;++i)

f[i][j]=max(f[i][j-1],f[i+(1<<(j-1))][j-1]);

Log[1]=0;

for(int i=2;i<=n+1;++i)

Log[i]=Log[i/2]+1;

}

int query(int l,int r){

int ans=0;

int k=Log[r-l+1];

ans=max(f[l][k],f[r-(1<<k)+1][k]);

return ans;

}

## 并查集

struct DS{

int fa[maxn],rank[maxn];

void init(){

for(int i=1;i<=n;++i)

fa[i]=i,rank[i]=0;

}

int find(int x){

if(x==fa[x])return x;

return fa[x]=find(fa[x]);

}

void merge(int x,int y){

int fx=find(x),fy=find(y);

if(fx==fy)return;

if(rank[fx]<rank[fy])

fa[fx]=fy;

else{

fa[fy]=fx;

if(rank[fy]==rank[fx])

rank[fx]++;

}

}

}ds;

## 树状数组

**//单点修改区间查询，区间修改，单点查询，区间修改，区间查询**

#define lowbit(x) (x&(-x))

struct BIT{

int c[maxn],N;

void init(int n){

this->N=n;

memset(c,0,sizeof(c));

}

void add(int x,int val){

while(x<=N){

c[x]+=val;

x+=lowbit(x);

}

}

int query(int x){

int ans=0;

while(x){

ans+=c[x];

x-=lowbit(x);

}

return ans;

}

}bit;

### 二维树状数组

#### 单点修改，区间查询

struct TwoDBIT{ 

int c[maxn][maxn],n,m;//n行m列

void init(int n){

this->n=n;this->m=m;

memset(c,0,sizeof(c));

}

void add(int x,int y,int val){

while(x<=n){

int yy=y;

while(yy<=m){

c[x][yy]+=val;yy+=lowbit(yy);

}

x+=lowbit(x);

}

}

int ask(int x,int y){

int ans=0;

while(x){

int yy=y;

while(yy){

ans+=c[x][yy],yy-=lowbit(yy);

}

x-=lowbit(x);

}

return ans;

}

int query(int xL,int yL,int xR,int yR){//二维前缀和区间查询

return ask(xR,yR)-ask(xL-1,yR)-ask(xR,yL-1)+ask(xL-1,yL-1);

}

}bit;

#### 区间修改,单点查询

struct TwoDBIT{//二维差分,c[i][j]=a[i][j]-a[i][j-1]-a[i-1][j]+a[i-1][j-1]

int c[maxn][maxn],n,m;//n行m列

void init(int n){

this->n=n;this->m=m;

memset(c,0,sizeof(c));

}

void add(int x,int y,int val){

while(x<=n){

int yy=y;

while(yy<=m){

c[x][yy]+=val;yy+=lowbit(yy);

}

x+=lowbit(x);

}

}

int ask(int x,int y){

int ans=0;

while(x){

int yy=y;

while(yy){

ans+=c[x][yy],yy-=lowbit(yy);

}

x-=lowbit(x);

}

return ans;

}

void realadd(int xL,int yL,int xR,int yR,int val){

add(xL,yL,val);

add(xR+1,yL,-val);

add(xL,yR+1,-val);

add(xR+1,yR+1,val);

}

int ask(int x,int y){

int ans=0;

while(x){

int yy=y;

while(yy){

ans+=c[x][yy],yy-=lowbit(yy);

}

x-=lowbit(x);

}

return ans;

}

}bit;

#### 区间修改，区间查询

struct TwoDBIT{//二维差分,c[i][j]=a[i][j]-a[i][j-1]-a[i-1][j]+a[i-1][j-1]

//维护c[i][j],c[i][j]\*i,c[i][j]\*j,c[i][j]\*i\*j

int c1[maxn][maxn],c2[maxn][maxn],c3[maxn][maxn],c4[maxn][maxn],n,m;

void init(int n){

this->n=n;this->m=m;

memset(c,0,sizeof(c));

}

void add(int x,int y,int val){

while(x<=n){

int yy=y;

while(yy<=m){

c1[x][yy]+=val,c2[x][yy]+=val\*x;

c3[x][yy]+=val\*y;c4[x][yy]+=val\*x\*y;

yy+=lowbit(yy);

}

x+=lowbit(x);

}

}

void realadd(int xL,int yL,int xR,int yR,int val){

add(xL,yL,val);

add(xR+1,yL,-val);

add(xL,yR+1,-val);

add(xR+1,yR+1,val);

}

int ask(int x,int y){

int ans=0;

while(x){

int yy=y;

while(yy){

ans+=(x+1)\*(y+1)\*c1[x][yy]-(y+1)\*c2[x][yy]-(x+1)\*c3[x][yy]+c4[x][yy];

yy-=lowbit(yy);

}

x-=lowbit(x);

}

return ans;

}

int realask(int xL,int yL,int xR,int yR){

return ask(xR,yR)-ask(xL-1,yR)-ask(xR,yL-1)+ask(xL-1,yL-1);

}

}bit;

## 线段树

### 区间赋值区间和

#define lson p<<1,l,mid

#define rson p<<1|1,mid+1,r

#define ls p<<1

#define rs p<<1|1

struct SegmentTree{

int sum[maxn],add[maxn];

void pushUp(int p){

sum[p]=sum[ls]+sum[rs];

}

void build(int p,int l,int r){

add[p]=-1;

if(l==r){

sum[p]=a[l];return;

}

int mid=l+r>>1;

build(lson);

build(rson);

pushUp(p);

}

void pushDown(int p,int l,int r){

if(add[p]!=-1){

int mid=l+r>>1;

sum[ls]=add[p]\*(mid-l+1);

sum[rs]=add[p]\*(r-mid);

add[ls]=add[rs]=add[p];

add[p]=-1;

}

}

void update(int p,int l,int r,int L,int R,int val){

if(L<=l&&r<=R){

sum[p]=(r-l+1)\*val;

add[p]=val;

return;

}

pushDown(p,l,r);

int mid=l+r>>1;

if(L<=mid)update(lson,L,R,val);

if(R>mid)update(rson,L,R,val);

pushUp(p);

}

int query(int p,int l,int r,int L,int R){

if(R<l||L>r)return 0;

if(L<=l&&r<=R)return sum[p];

int mid=l+r>>1;

int ans=0;

if(L<=mid)ans+=query(lson,L,R);

if(R>mid)ans+=query(rson,L,R);

return ans;

}

}tr;

### 扫描线

#### 矩形周长并

#include<bits/stdc++.h>//自下而上，横线当前长度减上一次，竖线高度\*竖线数量

#define ls p<<1

#define rs p<<1|1

#define lson p<<1,l,mid

#define rson p<<1|1,mid+1,r

using namespace std;

int n;

const int maxn=2e4+5;

struct Line{

int xL,xR,flag,h;

bool operator<(const Line&a){return h<a.h;}

}line[maxn];

int cnt[maxn<<2],num[maxn<<2],len[maxn<<2];//当前线段是否被覆盖，当前有多少竖线

bool Lf[maxn<<2],Rf[maxn<<2];//当前左右端点是否被覆盖

int a,b,c,d;

void pushUp(int p,int l,int r){

if(cnt[p]){

num[p]=2;

Lf[p]=Rf[p]=1;

len[p]=r-l+1;

}else if(l==r){//不被覆盖时候防止越界

num[p]=len[p]=Lf[p]=Rf[p]=0;

}else{

Lf[p]=Lf[ls];Rf[p]=Rf[rs];

len[p]=len[ls]+len[rs];

num[p]=num[ls]+num[rs];

if(Rf[ls]&&Lf[rs])num[p]-=2;//左右都被覆盖就回去

}

}

void update(int p,int l,int r,int L,int R,int val){

if(L<=l&&r<=R){

cnt[p]+=val;

return pushUp(p,l,r);

}

int mid=l+r>>1;

if(L<=mid)update(lson,L,R,val);

if(R>mid)update(rson,L,R,val);

pushUp(p,l,r);

}

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

while(cin>>n){

int lmn=20001,rmx=1;

for(int i=1;i<=n;++i){

cin>>a>>b>>c>>d;

a+=10001,c+=10001;

lmn=min(lmn,a);rmx=max(rmx,c);

line[(i<<1)-1]={a,c,1,b};line[i<<1]={a,c,-1,d};

}

sort(line+1,line+1+2\*n);

int last=0,ans=0;

for(int i=1;i<=2\*n;++i){

if(line[i].xL<line[i].xR)

update(1,1,rmx,line[i].xL,line[i].xR-1,line[i].flag);

ans+=num[1]\*(line[i+1].h-line[i].h);

ans+=abs(len[1]-last);

last=len[1];

}

cout<<ans<<"\n";

}

return 0;

}

#### 矩形K次以上面积并

//矩形k次以上面积并 原题是覆盖整数点右上坐标+1转化

#include<bits/stdc++.h>

#define lson p<<1,l,mid

#define rson p<<1|1,mid+1,r

#define ls p<<1

#define rs p<<1|1

using namespace std;

typedef long long ll;

const int maxn=3e4+5;

struct Line{

int xL,xR,flag,h; //flag-1表示下，1为上

bool operator<(const Line&a){return h<a.h;}

}line[maxn<<1];

int t,n,k,X[maxn<<1];

struct SegmentTree{

int len[maxn<<3][11],cnt[maxn<<3];//被覆盖i次以上的长度以及当前区间被覆盖次数

void pushUp(int p,int l,int r){

for(int i=0;i<=k;++i)len[p][i]=0;//上一次更新后得清空

if(cnt[p]>=k){//当前覆盖次数>=k

len[p][k]=X[r+1]-X[l];

}else if(l==r){//叶子结点只更新对应次数

len[p][cnt[p]]=X[r+1]-X[l];

}else{//覆盖不满的话，得把子结点标记影响上传

for(int i=0;i<=k;++i)

len[p][min(k,cnt[p]+i)]+=len[ls][i]+len[rs][i];

}

}

void build(int p,int l,int r){

cnt[p]=0;

for(int i=1;i<=k;++i)len[p][i]=0;

len[p][0]=X[r+1]-X[l];//初态的时候得赋值 注意扫描线是X[r+1]-X[l]

if(l==r)return;

int mid=l+r>>1;

build(lson);

build(rson);

}

void update(int p,int l,int r,int L,int R,int val){

if(L<=l&&r<=R){

cnt[p]+=val;

return pushUp(p,l,r);

}

int mid=l+r>>1;

if(L<=mid)update(lson,L,R,val);

if(R>mid)update(rson,L,R,val);

pushUp(p,l,r);

}

}tr;

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>t;

int T=0;

while(t--){

cin>>n>>k;

int a,b,c,d;

line[0].h=0;

for(int i=1;i<=n;++i){

cin>>a>>b>>c>>d;

c++;d++;

line[(i<<1)-1]={a,c,1,b};line[i<<1]={a,c,-1,d};

X[(i<<1)-1]=a;X[i<<1]=c;

}

sort(line+1,line+1+2\*n);

sort(X+1,X+1+2\*n);

int m=unique(X+1,X+1+2\*n)-(X+2);//线段数等于端点数-1

tr.build(1,1,m);

ll ans=0;

for(int i=1;i<=2\*n;++i){

ans+=1ll\*tr.len[1][k]\*(line[i].h-line[i-1].h);

int l1=lower\_bound(X+1,X+2+m,line[i].xL)-X;

int r1=lower\_bound(X+1,X+2+m,line[i].xR)-X-1;

tr.update(1,1,m,l1,r1,line[i].flag);

}

cout<<"Case "<<++T<<": "<<ans<<"\n";

}

return 0;

}

### 线段树合并

//子树各众数值之和 线段树合并复杂度取决于总点数 

#include<bits/stdc++.h>

#define pb push\_back

#define ls lc[p]

#define rs rc[p]

#define lson lc[p],l,mid

#define rson rc[p],mid+1,r

#define N maxn\*36

using namespace std;

typedef long long ll;

const int maxn=1e5+5;

int n,a[maxn],s,e,mx,rt[maxn];

ll ans[maxn];

vector<int>G[maxn];

struct SegmentTree{

int lc[N],rc[N],tot,cnt[N];//cnt区间权值线段树中最大值

ll sum[N];//各个众数之和

void pushUp(int p){

cnt[p]=max(cnt[ls],cnt[rs]);

if(cnt[ls]==cnt[rs])sum[p]=sum[ls]+sum[rs];

else sum[p]=cnt[ls]>cnt[rs]?sum[ls]:sum[rs];

}

void update(int &p,int l,int r,int x,int val){

if(!p)p=++tot;

if(l==r){

cnt[p]+=val;sum[p]=l;return;

}

int mid=l+r>>1;

if(x<=mid)update(lson,x,val);

else update(rson,x,val);

pushUp(p);

}

int merge(int p,int q,int l,int r){

if(!p||!q)return p+q;

if(l==r){

cnt[p]+=cnt[q];

sum[p]=l;return p;

}

int mid=l+r>>1;

ls=merge(ls,lc[q],l,mid);

rs=merge(rs,rc[q],mid+1,r);

pushUp(p);

return p;

}

}tr;

void dfs(int x,int fa){

for(auto&v:G[x]){

if(v==fa)continue;

dfs(v,x);

rt[x]=tr.merge(rt[x],rt[v],1,mx);

}

tr.update(rt[x],1,mx,a[x],1);

ans[x]=tr.sum[rt[x]];

}

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>n;

for(int i=1;i<=n;++i)cin>>a[i],mx=max(a[i],mx);

for(int i=1;i<n;++i){

cin>>s>>e;

G[s].pb(e);

G[e].pb(s);

}

dfs(1,-1);

for(int i=1;i<=n;++i)

cout<<ans[i]<<" ";

return 0;

}

### 线段树分裂

//线段树分裂 分裂其实可看作区间查询 每次最多增加O(2logn)结点

//维护多个可重集 合并带内存回收

//0.p集合中值从x到y的放入放入新集合

//1.把t合并到p中，清空t(t以后不出现) 这里用了内存回收 2.p中加入x个q

//3.查询p中x到y数量 4.查询p中第k小

#include<bits/stdc++.h>

#define ls lc[p]

#define rs rc[p]

#define lson lc[p],l,mid

#define rson rc[p],mid+1,r

using namespace std;

typedef long long ll;

#define N maxn\*32

const int maxn=2e5+5;

int n,m,op,cntset=1,a[maxn],rt[maxn];

struct SementTree{

int lc[N],rc[N],tot,rub[N],cnt=0;

ll sum[N];

int New(){//内存回收 分裂合并同时有的时候必须使用

if(cnt)return rub[cnt--];

return ++tot;

}

void del(int&p){

ls=rs=sum[p]=0;

rub[++cnt]=p;

p=0;

}

void pushUp(int p){

sum[p]=sum[ls]+sum[rs];

}

void build(int&p,int l,int r){

if(!p)p=New();

if(l==r){

sum[p]=a[l];return;

}

int mid=l+r>>1;

build(lson);

build(rson);

pushUp(p);

}

void update(int&p,int l,int r,int x,int val){

if(!p)p=New();

if(l==r){

sum[p]+=val;return;

}

int mid=l+r>>1;

if(x<=mid)update(lson,x,val);

else update(rson,x,val);

pushUp(p);

}

int merge(int &p,int &q,int l,int r){

if(!p||!q)return p+q;

if(l==r){

sum[p]+=sum[q];

del(q);

return p;

}

int mid=l+r>>1;

ls=merge(ls,lc[q],l,mid);

rs=merge(rs,rc[q],mid+1,r);

del(q);

pushUp(p);

return p;

}

void split(int&p,int&q,int l,int r,int L,int R){

if(L>r||R<l)return;

if(!p)return;

if(L<=l&&r<=R){

q=p;

p=0;return;//直接断边接上

}

if(!q)q=New();

int mid=l+r>>1;

if(L<=mid)split(ls,lc[q],l,mid,L,R);

if(R>mid)split(rs,rc[q],mid+1,r,L,R);

pushUp(p);//两颗更新

pushUp(q);

}

ll query(int p,int l,int r,int L,int R){

if(!p)return 0;

if(L<=l&&r<=R)return sum[p];

int mid=l+r>>1;

ll ans=0;

if(L<=mid)ans+=query(lson,L,R);

if(R>mid)ans+=query(rson,L,R);

return ans;

}

int kth(int p,int l,int r,int k){

if(l==r)return l;

int mid=l+r>>1;

if(sum[ls]>=k)return kth(lson,k);

else return kth(rson,k-sum[ls]);

}

}tr;

int main(){

int x,y,z;

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>n>>m;

for(int i=1;i<=n;++i)cin>>a[i];

tr.build(rt[1],1,n);

for(int i=1;i<=m;++i){

cin>>op;

if(!op){

cin>>x>>y>>z;

tr.split(rt[x],rt[++cntset],1,n,y,z);

}else if(op==1){

cin>>x>>y;

rt[x]=tr.merge(rt[x],rt[y],1,n);

}else if(op==2){

cin>>x>>y>>z;

tr.update(rt[x],1,n,z,y);

}else if(op==3){

cin>>x>>y>>z;

cout<<tr.query(rt[x],1,n,y,z)<<"\n";

}else if(op==4){

cin>>x>>y;

if(tr.sum[rt[x]]<y)cout<<-1<<"\n";

else cout<<tr.kth(rt[x],1,n,y)<<"\n";

}

}

return 0;

}

### 兔队线段树

1.普通的楼房重建写法

#include<bits/stdc++.h>

#define lson p<<1,l,mid

#define rson p<<1|1,mid+1,r

#define ls p<<1

#define rs p<<1|1

using namespace std;

using ll=long long;

const int maxn=1e5+10;

int n,m,H[maxn];

struct SegmentTree{//求有多少个位置是前缀最大值

int sum[maxn<<2],id[maxn<<2];//sum只考虑当前子树影响的答案

bool isBig(int id1,int id2){ //is id1 > id2

if(!id2)return H[id1];

return (ll)H[id1]\*id2>(ll)H[id2]\*id1;

}

int cal(int p,int l,int r,int mxid){ //p子树内 考虑了前缀最大值mxid后的答案

if(l==r)return isBig(l,mxid);

int mid=l+r>>1;

if(isBig(id[ls],mxid))return cal(lson,mxid)+sum[p]-sum[ls];//信息可减性

else return 0+cal(rson,mxid);

}

void pushUp(int p,int l,int r,int mid){

id[p]=isBig(id[rs],id[ls])?id[rs]:id[ls];

sum[p]=sum[ls]+cal(rson,id[ls]);

}

void build(int p,int l,int r){

id[p]=l;sum[p]=1;

if(l==r)return;

int mid=l+r>>1;

build(lson);

build(rson);

}

void update(int p,int l,int r,int x){

if(l==r) return;

int mid=l+r>>1;

if(x<=mid)update(lson,x);

else update(rson,x);

pushUp(p,l,r,mid);

}

}tr;

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>n>>m;

tr.build(1,1,n);

for(int i=1;i<=m;++i){

int x,y;

cin>>x>>y;

H[x]=y;

tr.update(1,1,n,x);

cout<<tr.cal(1,1,n,0)<<"\n";

}

return 0;

}

2.兔队写法

#include<bits/stdc++.h>

#define lson p<<1,l,mid

#define rson p<<1|1,mid+1,r

#define ls p<<1

#define rs p<<1|1

using namespace std;

using ll=long long;

const int maxn=1e5+10;

int n,m,H[maxn];

struct SegmentTree{//求有多少个位置是前缀最大值

int sum[maxn<<2],id[maxn<<2];//sum只考虑当前子树影响下 右子树的答案

bool isBig(int id1,int id2){ //is id1 > id2

if(!id2)return H[id1];

return (ll)H[id1]\*id2>(ll)H[id2]\*id1;

}

int cal(int p,int l,int r,int mxid){ //p子树内 考虑了前缀最大值mxid后的答案

if(l==r)return isBig(l,mxid);

int mid=l+r>>1;

if(isBig(id[ls],mxid))return cal(lson,mxid)+sum[p];//不用信息可减性

else return 0+cal(rson,mxid);

}

void pushUp(int p,int l,int r,int mid){

id[p]=isBig(id[rs],id[ls])?id[rs]:id[ls];

sum[p]=cal(rson,id[ls]);

}

void build(int p,int l,int r){

id[p]=l;sum[p]=1;//叶子节点下sum是什么随意

if(l==r)return;

int mid=l+r>>1;

build(lson);

build(rson);

}

void update(int p,int l,int r,int x){

if(l==r) return;

int mid=l+r>>1;

if(x<=mid)update(lson,x);

else update(rson,x);

pushUp(p,l,r,mid);

}

}tr;

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>n>>m;

tr.build(1,1,n);

for(int i=1;i<=m;++i){

int x,y;

cin>>x>>y;

H[x]=y;

tr.update(1,1,n,x);

cout<<tr.cal(1,1,n,0)<<"\n";

}

return 0;

}

### SegmentTree beats

区间取min, 区间最大值，区间和

#include<bits/stdc++.h>

#define lson p<<1,l,mid

#define rson p<<1|1,mid+1,r

#define ls p<<1

#define rs p<<1|1

using namespace std;

using ll=long long;

const int maxn=1e6+5;

int n,m,a[maxn];

struct SegmentTreeBeats{

//分最大，次大,最大数量

int mx[maxn<<2],se[maxn<<2],tag[maxn<<2],cnt[maxn<<2];

ll sum[maxn<<2];

void pushUp(int p){

sum[p]=sum[ls]+sum[rs];

mx[p]=max(mx[ls],mx[rs]);

if(mx[ls]==mx[rs]){

se[p]=max(se[ls],se[rs]);

cnt[p]=cnt[ls]+cnt[rs];

}else if(mx[ls]>mx[rs]){

se[p]=max(se[ls],mx[rs]);

cnt[p]=cnt[ls];

}else{

se[p]=max(mx[ls],se[rs]);

cnt[p]=cnt[rs];

}

}

void build(int p,int l,int r){

tag[p]=-1;

if(l==r){

mx[p]=sum[p]=a[l];

se[p]=-1;//初始化严格小于最大值

cnt[p]=1;

return;

}

int mid=l+r>>1;

build(lson);

build(rson);

pushUp(p);

}

void add\_tag(int p,int c){

if(mx[p]<=c)return;//可能最大值不在该子区间

sum[p]-=(ll)cnt[p]\*(mx[p]-c);

mx[p]=tag[p]=c;

}

void pushDown(int p){

add\_tag(ls,tag[p]);

add\_tag(rs,tag[p]);

tag[p]=-1;

}

void update(int p,int l,int r,int L,int R,int val){

if(mx[p]<=val)return;

if(L<=l&&r<=R&&val>se[p]){

add\_tag(p,val);

return;

}

if(tag[p]!=-1)

pushDown(p);

int mid=l+r>>1;

if(L<=mid)update(lson,L,R,val);

if(R>mid)update(rson,L,R,val);

pushUp(p);

}

int queryMx(int p,int l,int r,int L,int R){

if(L<=l&&r<=R)return mx[p];

if(tag[p]!=-1)

pushDown(p);

int mid=l+r>>1,ans=0;

if(L<=mid)ans=max(ans,queryMx(lson,L,R));

if(R>mid)ans=max(ans,queryMx(rson,L,R));

return ans;

}

ll querySum(int p,int l,int r,int L,int R){

if(L<=l&&r<=R)return sum[p];

if(tag[p]!=-1)

pushDown(p);

int mid=l+r>>1;

ll ans=0;

if(L<=mid)ans+=querySum(lson,L,R);

if(R>mid)ans+=querySum(rson,L,R);

return ans;

}

}tr;

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

int t;

cin>>t;

while(t--){

cin>>n>>m;

for(int i=1;i<=n;++i)cin>>a[i];

tr.build(1,1,n);

int op,x,y,z;

for(int i=1;i<=m;++i){

cin>>op>>x>>y;

if(!op){

cin>>z;

tr.update(1,1,n,x,y,z);

}else if(op==1){

cout<<tr.queryMx(1,1,n,x,y)<<"\n";

}else

cout<<tr.querySum(1,1,n,x,y)<<"\n";

}

}

return 0;

}

区间加，区间max,区间min，区间和，区间最大最小查询，划分数域

#include<bits/stdc++.h>

#define lson p<<1,l,mid

#define rson p<<1|1,mid+1,r

#define ls p<<1

#define rs p<<1|1

using namespace std;

using ll=long long;

const int maxn=5e5+5;

const ll INF=1e18;

int n,a[maxn],m;

struct SegmentTree{//数域划分 最大 最小 其他值

ll mx[maxn<<2],mx2[maxn<<2],mxc[maxn<<2];//最大 次大 数量

ll mn[maxn<<2],mn2[maxn<<2],mnc[maxn<<2];

ll aMx[maxn<<2],aMn[maxn<<2],aTag[maxn<<2];//对应数域加法标记

ll sum[maxn<<2];

int len[maxn<<2];

void pushUp(int p){

sum[p]=sum[ls]+sum[rs];

mx[p]=max(mx[ls],mx[rs]);

mn[p]=min(mn[ls],mn[rs]);

if(mx[ls]==mx[rs]){

mx2[p]=max(mx2[ls],mx2[rs]);

mxc[p]=mxc[ls]+mxc[rs];

}else if(mx[ls]>mx[rs]){

mx2[p]=max(mx2[ls],mx[rs]);

mxc[p]=mxc[ls];

}else{

mx2[p]=max(mx[ls],mx2[rs]);

mxc[p]=mxc[rs];

}

if(mn[ls]==mn[rs]){

mn2[p]=min(mn2[ls],mn2[rs]);

mnc[p]=mnc[ls]+mnc[rs];

}else if(mn[ls]<mn[rs]){

mn2[p]=min(mn2[ls],mn[rs]);

mnc[p]=mnc[ls];

}else{

mn2[p]=min(mn[ls],mn2[rs]);

mnc[p]=mnc[rs];

}

}

void build(int p,int l,int r){

aMx[p]=aMx[p]=aTag[p]=0;

len[p]=r-l+1;

if(l==r){

mx[p]=mn[p]=sum[p]=a[l];

mxc[p]=mnc[p]=1;

mx2[p]=-INF;

mn2[p]=INF;

return;

}

int mid=l+r>>1;

build(lson);

build(rson);

pushUp(p);

}

void pushADD(int p,ll amx,ll amn,ll atag){//多值域加法标记下放

if(mx[p]==mn[p]){//特判只有一个值

if(amx==atag)amx=amn;//无最大 就用最少

else amn=amx;//无最小 最小用最大

sum[p]+=amx\*mxc[p];

}

else sum[p]+=amx\*mxc[p]+amn\*mnc[p]+atag\*(len[p]-mxc[p]-mnc[p]);

//两个值时特判 次大等于最小用最小 否则其他

if(mx2[p]==mn[p])mx2[p]+=amn;

else if(mx2[p]!=-INF)mx2[p]+=atag;

if(mn2[p]==mx[p])mn2[p]+=amx;

else if(mn2[p]!=INF)mn2[p]+=atag;

mx[p]+=amx;mn[p]+=amn;

aMx[p]+=amx;aMn[p]+=amn;aTag[p]+=atag;

}

void pushDown(int p){

ll Mx=max(mx[ls],mx[rs]),Mn=min(mn[ls],mn[rs]);

pushADD(ls,(mx[ls]==Mx?aMx[p]:aTag[p]),(mn[ls]==Mn?aMn[p]:aTag[p]),aTag[p]);

pushADD(rs,(mx[rs]==Mx?aMx[p]:aTag[p]),(mn[rs]==Mn?aMn[p]:aTag[p]),aTag[p]);

aMx[p]=aMn[p]=aTag[p]=0;

}

void ADD(int p,int l,int r,int L,int R,ll val){

if(L<=l&&r<=R) return pushADD(p,val,val,val);

pushDown(p);

int mid=l+r>>1;

if(L<=mid)ADD(lson,L,R,val);

if(R>mid)ADD(rson,L,R,val);

pushUp(p);

}

void MAX(int p,int l,int r,int L,int R,ll val){

if(mn[p]>=val)return;

if(L<=l&&r<=R&&mn2[p]>val)return pushADD(p,0,val-mn[p],0);

pushDown(p);

int mid=l+r>>1;

if(L<=mid)MAX(lson,L,R,val);

if(R>mid)MAX(rson,L,R,val);

pushUp(p);

}

void MIN(int p,int l,int r,int L,int R,int val){

if(mx[p]<=val)return;

if(L<=l&&r<=R&&mx2[p]<val)return pushADD(p,val-mx[p],0,0);

pushDown(p);

int mid=l+r>>1;

if(L<=mid)MIN(lson,L,R,val);

if(R>mid)MIN(rson,L,R,val);

pushUp(p);

}

ll queryMin(int p,int l,int r,int L,int R){

if(L<=l&&r<=R)return mn[p];

pushDown(p);

int mid=l+r>>1;

ll ans=INF;

if(L<=mid)ans=min(ans,queryMin(lson,L,R));

if(R>mid)ans=min(ans,queryMin(rson,L,R));

return ans;

}

ll queryMx(int p,int l,int r,int L,int R){

if(L<=l&&r<=R)return mx[p];

pushDown(p);

int mid=l+r>>1;

ll ans=-INF;

if(L<=mid)ans=max(ans,queryMx(lson,L,R));

if(R>mid)ans=max(ans,queryMx(rson,L,R));

return ans;

}

ll querySum(int p,int l,int r,int L,int R){

if(L<=l&&r<=R)return sum[p];

pushDown(p);

int mid=l+r>>1;

ll ans=0;

if(L<=mid)ans+=querySum(lson,L,R);

if(R>mid)ans+=querySum(rson,L,R);

return ans;

}

}tr;

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>n;

for(int i=1;i<=n;++i)cin>>a[i];

cin>>m;

tr.build(1,1,n);

int op,l,r,x;

for(int i=1;i<=m;++i){

cin>>op>>l>>r;

if(op==1){

cin>>x;

tr.ADD(1,1,n,l,r,x);

}else if(op==2){

cin>>x;

tr.MAX(1,1,n,l,r,x);

}else if(op==3){

cin>>x;

tr.MIN(1,1,n,l,r,x);

}else if(op==4){

cout<<tr.querySum(1,1,n,l,r)<<'\n';

}else if(op==5){

cout<<tr.queryMx(1,1,n,l,r)<<"\n";

}else

cout<<tr.queryMin(1,1,n,l,r)<<"\n";

}

return 0;

}

//区间排序后求某点

#include<bits/stdc++.h>

#define ls lc[p]

#define rs rc[p]

#define lson lc[p],l,mid

#define rson rc[p],mid+1,r

#define N maxn\*54

using namespace std;

typedef set<int>::iterator IT;

const int maxn=1e5+5;

int n,m,a,L,R,pos;

int op,Op[maxn],rt[maxn];

set<int>s;//存储有序区间的左端点(根结点)方便分裂与合并

struct SegmentTree{

int lc[N],rc[N],sum[N],rub[N],tot,cnt=0;

int New(){//内存回收 衡量垃圾桶与点数越界的内存差

if(cnt)return rub[cnt--];

return ++tot;

}

void del(int&p){

ls=rs=sum[p]=0;

rub[++cnt]=p;

p=0;

}

int merge(int p,int&q,int l,int r){

if(!p||!q)return p+q;

sum[p]+=sum[q];

if(l==r){del(q);return p;}

int mid=l+r>>1;

ls=merge(ls,lc[q],l,mid);

rs=merge(rs,rc[q],mid+1,r);

del(q);

return p;

}

void pushUp(int p){

sum[p]=sum[ls]+sum[rs];

}

void update(int&p,int l,int r,int x){

if(!p)p=New();

if(l==r){

sum[p]++;return;

}

int mid=l+r>>1;

if(x<=mid)update(lson,x);

else update(rson,x);

pushUp(p);

}

int query(int p,int l,int r){

if(l==r)return l;

int mid=l+r>>1;

if(ls)return query(lson);

else return query(rson);

}

void split(int p,int&q,int k,int op){//把p分裂成左p右q的两个区间 给p留指定的k个

if(!p)return;//有重复元素的时候叶子也可以是k 防空结点

if(sum[p]==k)return;

if(!q)q=New();

sum[q]=sum[p]-k;sum[p]=k;

if(op){//1降序找前k大

if(sum[rs]>=k){

split(rs,rc[q],k,op);

lc[q]=lc[p];lc[p]=0;//左边直接断给q

}else

split(ls,lc[q],k-sum[rs],op);

}else{//0升序找前k小

if(sum[ls]>=k){

split(ls,lc[q],k,op);

rc[q]=rc[p];rc[p]=0;

}else

split(rs,rc[q],k-sum[ls],op);

}

}

}tr;

IT Split(int x){

auto v=s.lower\_bound(x);

if(\*v==x)return v;

--v;

Op[x]=Op[\*v];//拆分的时候保证前k大还是前k小选择正确

tr.split(rt[\*v],rt[x],x-\*v,Op[x]);

return s.insert(x).first;//插入分裂的区间的新的左端点

}

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>n>>m;

s.insert(n+1);

for(int i=1;i<=n;++i){

cin>>a;

tr.update(rt[i],1,n,a);

s.insert(i);

}

for(int i=1;i<=m;++i){

cin>>op>>L>>R;

auto st=Split(L);

auto ed=Split(R+1);

for(auto it=++st;it!=ed;++it){

rt[L]=tr.merge(rt[L],rt[\*it],1,n);

}

Op[L]=op;s.erase(st,ed);//最多2个

}

cin>>pos;

Split(pos);Split(pos+1);//分裂成单点 求整个区间每个i分裂一次就好了

cout<<tr.query(rt[pos],1,n)<<"\n";

return 0;

}

### 可持久化线段树

#### 静态区间第k小

//可持久化线段树静态区间第k大 也可用线段树合并

#include<bits/stdc++.h>

#define ls lc[p]

#define rs rc[p]

#define lson lc[p],l,mid

#define rson rc[p],mid+1,r

#define N maxn\*36

using namespace std;

const int maxn=2e5+5;

int n,m,a[maxn],b[maxn],cnt,L,R,k,rt[maxn];

struct Persistable\_SegmentTree{

int tot,lc[N],rc[N],sum[N];

void pushUp(int p){

sum[p]=sum[ls]+sum[rs];

}

void update(int q,int&p,int l,int r,int x){

p=++tot;

ls=lc[q],rs=rc[q],sum[p]=sum[q];

if(l==r){

sum[p]++;return;

}

int mid=l+r>>1;

if(x<=mid)update(lc[q],lson,x);

else update(rc[q],rson,x);

pushUp(p);

}

int query(int q,int p,int l,int r,int k){

if(l==r)return l;

int mid=l+r>>1,dir=sum[ls]-sum[lc[q]];

if(dir>=k)return query(lc[q],lson,k);

else return query(rc[q],rson,k-dir);

}

}tr;

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>n>>m;

for(int i=1;i<=n;++i)cin>>a[i],b[i]=a[i];

sort(b+1,b+1+n);

cnt=unique(b+1,b+1+n)-(b+1);

for(int i=1;i<=n;++i){

a[i]=lower\_bound(b+1,b+1+cnt,a[i])-b;

}

for(int i=1;i<=n;++i){

tr.update(rt[i-1],rt[i],1,cnt,a[i]);

}

for(int i=1;i<=m;++i){

cin>>L>>R>>k;

cout<<b[tr.query(rt[L-1],rt[R],1,cnt,k)]<<"\n";

}

return 0;

}

#### 区间不同数个数

//在线求区间不同数个数 和对应位置1改为值即可

#include<bits/stdc++.h>

#define N maxn\*36

#define ls lc[p]

#define rs rc[p]

#define lson lc[p],l,mid

#define rson rc[p],mid+1,r

using namespace std;

const int maxn=3e4+5;

const int maxm=1e6+5; //每颗线段树上维护的是pos上有无值

int rt[maxn],a[maxn],pre[maxm],q,L,R,n,b[maxn];

struct Persistable\_SegmentTree{//本质维护[1,i]中数最后一次出现的位置

int lc[N],rc[N],sum[N],tot;

void pushUp(int p){

sum[p]=sum[ls]+sum[rs];

}

void update(int q,int&p,int l,int r,int x,int val){

p=++tot;

ls=lc[q],rs=rc[q],sum[p]=sum[q];

if(l==r){

sum[p]+=val;return;

}

int mid=l+r>>1;

if(x<=mid)update(lc[q],lson,x,val);

else update(rc[q],rson,x,val);

pushUp(p);

}

int query(int p,int l,int r,int L,int R){

if(L<=l&&r<=R)return sum[p];

int mid=l+r>>1;

int ans=0;

if(L<=mid)ans+=query(lson,L,R);

if(R>mid)ans+=query(rson,L,R);

return ans;

}

}tr;

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>n;

for(int i=1;i<=n;++i)cin>>a[i];

cin>>q;

for(int i=1;i<=n;++i){

if(pre[a[i]]){

tr.update(rt[i-1],rt[i],1,n,pre[a[i]],-1);//可离散化x

tr.update(rt[i],rt[i],1,n,i,1);

}

else tr.update(rt[i-1],rt[i],1,n,i,1);

pre[a[i]]=i;

}

for(int i=1;i<=q;++i){

cin>>L>>R;

cout<<tr.query(rt[R],1,n,L,n)<<"\n";

}

return 0;

}

**离线树状数组做法**

#include<bits/stdc++.h>

using namespace std;

const int maxn=1e6+5;

int n,a[maxn],m,ans[maxn],pre[maxn];

struct Q{

int l,r,id;

bool operator<(const Q&a)const{

return r<a.r;

}

}q[maxn];

struct BIT{

int c[maxn];

#define lowb(x) (x&(-x))

void add(int x,int val){

for(int i=x;i<=n;i+=lowb(i))c[i]+=val;

}

int ask(int x){

int ans=0;

for(int i=x;i;i-=lowb(i))ans+=c[i];

return ans;

}

}bit;

int main(){

scanf("%d",&n);

for(int i=1;i<=n;++i)scanf("%d",&a[i]);

scanf("%d",&m);

for(int i=1;i<=m;++i)scanf("%d%d",&q[i].l,&q[i].r),q[i].id=i;

sort(q+1,q+1+m);

int now=1;

for(int i=1;i<=m;++i){

while(now<=q[i].r){

if(pre[a[now]])bit.add(pre[a[now]],-1),pre[a[now]]=now,bit.add(now,1);

else pre[a[now]]=now,bit.add(now,1);

now++;

}

ans[q[i].id]=bit.ask(q[i].r)-bit.ask(q[i].l-1);

}

for(int i=1;i<=m;++i)cout<<ans[i]<<"\n";

return 0;

}

#### 区间修改，标记永久化

//标记永久化 区间加区间和

#include<bits/stdc++.h>

using namespace std;

#define lson lc[p],l,mid

#define rson rc[p],mid+1,r

#define ls lc[p]

#define rs rc[p]

#define N maxn\*32

typedef long long ll;

const int maxn=1e5+5;

int rt[maxn],last[maxn],n,m,L,R,nowtime,d,t;

char s[2];

struct Persistable\_SegmentTree{

int tot,lc[N],rc[N],add[N];

ll sum[N];

void pushUp(int p){

sum[p]=sum[ls]+sum[rs];

}

void update(int q,int&p,int l,int r,int L,int R,int val){

p=++tot;

ls=lc[q];rs=rc[q];sum[p]=sum[q];add[p]=add[q];

sum[p]+=1ll\*(min(R,r)-max(L,l)+1)\*val;//非完全包含带着标记更新/pushUp 看情况

if(L<=l&&r<=R){

add[p]+=val;//完全包含就标记

return;

}

int mid=l+r>>1;

if(L<=mid)update(lc[q],lson,L,R,val);

if(R>mid)update(rc[q],rson,L,R,val);

}

void build(int&p,int l,int r){

p=++tot;

add[p]=0;

if(l==r){

scanf("%lld",&sum[p]);return;

}

int mid=l+r>>1;

build(lson);

build(rson);

pushUp(p);

}

ll query(int p,int l,int r,int L,int R){

if(L<=l&&r<=R)return sum[p];

int mid=l+r>>1;

ll ans=1ll\*add[p]\*(min(R,r)-max(L,l)+1);//查询的时候累积标记值下查

if(L<=mid)ans+=query(lson,L,R);

if(R>mid)ans+=query(rson,L,R);

return ans;

}

}tr;

int main(){

while(~scanf("%d%d",&n,&m)){

tr.tot=nowtime=0;

tr.build(rt[0],1,n);

for(int i=1;i<=m;++i){

scanf("%s",s);

if(s[0]=='C'){//创建新版本

scanf("%d%d%d",&L,&R,&d);

nowtime++;

tr.update(rt[nowtime-1],rt[nowtime],1,n,L,R,d);

}else if(s[0]=='Q'){//当前版本区间和

scanf("%d%d",&L,&R);

cout<<tr.query(rt[nowtime],1,n,L,R)<<"\n";

}else if(s[0]=='H'){//历史版本区间和

scanf("%d%d%d",&L,&R,&t);

cout<<tr.query(rt[t],1,n,L,R)<<"\n";

}else if(s[0]=='B'){//不用清空 自动覆盖

scanf("%d",&nowtime);

}

}

}

return 0;

}

### 树套树

#### 树状数组套可持久化线段树 区间k大修改

//树状数组套权值线段树求动态修改区间第k大 可以在线

#include<bits/stdc++.h>

#define lowbit(x) (x&(-x))

#define N maxn\*300

#define lson lc[p],l,mid

#define rson rc[p],mid+1,r

using namespace std;

const int maxn=2e5+5;

const int maxm=1e5+5;

int n,m,a[maxn],b[maxn],cnt,rt[maxm],jL[22],jR[22],numL,numR;

struct BitAndSegmentTree{

int tot,lc[N],rc[N],sum[N];

void update(int&p,int l,int r,int x,int val){

if(!p)p=++tot;

sum[p]+=val;

if(l==r)return;

int mid=l+r>>1;

if(x<=mid)update(lson,x,val);

else update(rson,x,val);

}

int kth(int l,int r,int k){//log颗一起求k大

if(l==r)return l;

int mid=l+r>>1,ans=0;

for(int i=1;i<=numL;++i)ans-=sum[lc[jL[i]]];

for(int i=1;i<=numR;++i)ans+=sum[lc[jR[i]]];

if(ans>=k){

for(int i=1;i<=numL;++i)//一起换根

jL[i]=lc[jL[i]];

for(int i=1;i<=numR;++i)

jR[i]=lc[jR[i]];

return kth(l,mid,k);

}else{

for(int i=1;i<=numL;++i)

jL[i]=rc[jL[i]];

for(int i=1;i<=numR;++i)

jR[i]=rc[jR[i]];

return kth(mid+1,r,k-ans);

}

}

void add(int x,int y){

int pos=lower\_bound(b+1,b+1+cnt,a[x])-b;

for(int i=x;i<=n;i+=lowbit(i))update(rt[i],1,cnt,pos,y);

}

int query(int l,int r,int k){

numL=numR=0;

for(int i=l-1;i;i-=lowbit(i))jL[++numL]=rt[i];//预处理树状数组上要跳的log颗线段树的根

for(int i=r;i;i-=lowbit(i))jR[++numR]=rt[i];

return kth(1,cnt,k);

}

}tr;

struct Q{

char s[2];

int x,y,z;

}qs[maxm];

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>n>>m;

for(int i=1;i<=n;++i)cin>>a[i],b[i]=a[i];

cnt=n;

for(int i=1;i<=m;++i){

cin>>qs[i].s;

if(qs[i].s[0]=='Q'){

cin>>qs[i].x>>qs[i].y>>qs[i].z;

}else{

cin>>qs[i].x>>qs[i].y;

b[++cnt]=qs[i].y;//修改的值也要离散化 不存的话就暴力在线

}

}

sort(b+1,b+1+cnt);

cnt=unique(b+1,b+1+cnt)-(b+1);

for(int i=1;i<=n;++i)tr.add(i,1);//树状数组上建树

for(int i=1;i<=m;++i){

if(qs[i].s[0]=='Q'){

cout<<b[tr.query(qs[i].x,qs[i].y,qs[i].z)]<<"\n";

}else{

tr.add(qs[i].x,-1);

a[qs[i].x]=qs[i].y;

tr.add(qs[i].x,1);

}

}

return 0;

}

## 分块

### 区间众数 强制在线

#include<bits/stdc++.h>

using namespace std;

const int maxn=40005;

int cnt[250][maxn],mx[250][250],ans[250][250],tmp[maxn];

int a[maxn],n,m,big,pos[maxn],num,L[250],R[250],b[maxn],id[maxn],c;

void init(){

big=sqrt(n);

num=(n-1)/big+1;

for(int i=1;i<=num;++i){

L[i]=(i-1)\*big+1;

R[i]=i\*big;

}

R[num]=n;

for(int i=1;i<=num;++i)

for(int j=L[i];j<=R[i];++j)pos[j]=i;

sort(b+1,b+1+n);

c=unique(b+1,b+1+n)-(b+1);

for(int i=1;i<=n;++i){

int x=lower\_bound(b+1,b+1+c,a[i])-b;

id[x]=a[i];

a[i]=x;

}

for(int i=1;i<=n;++i){

cnt[pos[i]][a[i]]++;

}

for(int i=2;i<=num;++i){

for(int j=1;j<=c;++j)

cnt[i][j]+=cnt[i-1][j];

}

for(int i=1;i<=num;++i){

int x=0,z=0;//mx ans

for(int j=L[i];j<=n;++j){

tmp[a[j]]++;

if(tmp[a[j]]>x)x=tmp[a[j]],z=a[j];

else if(tmp[a[j]]==x&&z!=a[j]){

z=min(z,a[j]);

}

mx[i][pos[j]]=x;ans[i][pos[j]]=z;

}

for(int j=L[i];j<=n;++j)tmp[a[j]]--;

}

}

int query(int l,int r){

int pl=pos[l],pr=pos[r];

if(pl==pr){

int x=0,z=0;

for(int i=l;i<=r;++i){

tmp[a[i]]++;

if(tmp[a[i]]>x)x=tmp[a[i]],z=a[i];

else if(tmp[a[i]]==x&&z!=a[i]){

z=min(z,a[i]);

}

}

for(int i=l;i<=r;++i)tmp[a[i]]--;

return z;

}else{

int x=mx[pl+1][pr-1],z=ans[pl+1][pr-1];

for(int i=l;i<=R[pl];++i)tmp[a[i]]=cnt[pr-1][a[i]]-cnt[pl][a[i]];

for(int i=l;i<=R[pl];++i){

tmp[a[i]]++;

if(tmp[a[i]]>x)x=tmp[a[i]],z=a[i];

else if(tmp[a[i]]==x&&a[i]!=z)z=min(z,a[i]);

}

for(int i=L[pr];i<=r;++i)

tmp[a[i]]=tmp[a[i]]?tmp[a[i]]:cnt[pr-1][a[i]]-cnt[pl][a[i]];

for(int i=L[pr];i<=r;++i){

tmp[a[i]]++;

if(tmp[a[i]]>x)x=tmp[a[i]],z=a[i];

else if(tmp[a[i]]==x&&a[i]!=z)z=min(z,a[i]);

}

for(int i=l;i<=R[pl];++i)tmp[a[i]]=0;

for(int i=L[pr];i<=r;++i)tmp[a[i]]=0;

return z;

}

}

int main(){

scanf("%d%d",&n,&m);

for(int i=1;i<=n;++i)

scanf("%d",&a[i]),b[i]=a[i];

init();

int l,r,anss=0;

for(int i=1;i<=m;++i){

scanf("%d%d",&l,&r);

l=(l+anss-1)%n+1;r=(r+anss-1)%n+1;

if(l>r)swap(l,r);

cout<<(anss=id[query(l,r)])<<"\n";

}

return 0;

}

### 带插入区间k大

//带单点修 插入区间k小值 O(nsqrt(n))

#include<bits/stdc++.h>

using namespace std;

const int maxn=70005;

int n,big,num,pos[maxn],st=1,tmpcnt[305],tmpsum[maxn],m;

struct Block{//序列分块

int l,r,sz;

int cnt[305],sum[maxn];//序列前i块在值域第j块出现次数,前i块值j出现次数

int a[605];

}b[605];

void init(){

big=300;

num=n/big+1;

for(int i=0;i<=70000;++i)pos[i]=i/big+1;//这里将值域和序列块大小设为一样 需要卡常要改

for(int i=1;i<=n;++i){

int x;

scanf("%d",&x);

b[pos[i]].a[++b[pos[i]].sz]=x;

}

for(int i=1;i<=num;++i){

if(i>1)b[i].l=i-1;

if(i<num)b[i].r=i+1;

for(int j=1;j<=b[i].sz;++j){

b[i].cnt[pos[b[i].a[j]]]++;

b[i].sum[b[i].a[j]]++;

}

for(int j=0;j<=70000;++j){

if(!j||pos[j]!=pos[j-1])b[i].cnt[pos[j]]+=b[i-1].cnt[pos[j]];

b[i].sum[j]+=b[i-1].sum[j];

}

}

}

int kth(int l,int r,int k){

int now=st,pl,pr;

while(b[now].sz<l)l-=b[now].sz,now=b[now].r;

pl=now,now=st;

while(b[now].sz<r)r-=b[now].sz,now=b[now].r;

pr=now;//块状链表上找位置

if(pl==pr){//[1,l][l+1,r][r+1,b.sz]

for(int i=l;i<=r;++i)tmpcnt[pos[b[pl].a[i]]]++,tmpsum[b[pl].a[i]]++;//tmpcnt临时 值域第i块出现次数 tmpsum某数出现的次数

now=1;//构建新的值域分块

while(tmpcnt[now]<k)k-=tmpcnt[now],now++;

now=(now-1)\*big;

while(tmpsum[now]<k)k-=tmpsum[now],now++;//进入值域树第三层找

for(int i=l;i<=r;++i)tmpcnt[pos[b[pl].a[i]]]--,tmpsum[b[pl].a[i]]--;

return now;

}else{

for(int i=l;i<=b[pl].sz;++i)tmpcnt[pos[b[pl].a[i]]]++,tmpsum[b[pl].a[i]]++;

for(int i=1;i<=r;++i)tmpcnt[pos[b[pr].a[i]]]++,tmpsum[b[pr].a[i]]++;

now=1;//值域分块中找

while(tmpcnt[now]+b[b[pr].l].cnt[now]-b[pl].cnt[now]<k)k-=tmpcnt[now]+b[b[pr].l].cnt[now]-b[pl].cnt[now],now++;

now=(now-1)\*big;

while(tmpsum[now]+b[b[pr].l].sum[now]-b[pl].sum[now]<k)k-=tmpsum[now]+b[b[pr].l].sum[now]-b[pl].sum[now],now++;

for(int i=l;i<=b[pl].sz;++i)tmpcnt[pos[b[pl].a[i]]]--,tmpsum[b[pl].a[i]]--;

for(int i=1;i<=r;++i)tmpcnt[pos[b[pr].a[i]]]--,tmpsum[b[pr].a[i]]--;

return now;

}

}

void update(int x,int y){

int now=st;

while(b[now].sz<x)x-=b[now].sz,now=b[now].r;

int fi=b[now].a[x];

if(fi==y)return;

b[now].a[x]=y;

while(now){//维护值域分块

b[now].cnt[pos[fi]]--;

b[now].cnt[pos[y]]++;

b[now].sum[fi]--;

b[now].sum[y]++;

now=b[now].r;

}

}

void split(int x){//直接分裂

int newP=++num;

b[newP].r=b[x].r;b[b[x].r].l=newP;b[x].r=newP;b[newP].l=x;

b[newP].sz=b[x].sz/2;

int del=b[x].sz-b[newP].sz;

memcpy(b[newP].cnt,b[x].cnt,sizeof(b[x].cnt));

memcpy(b[newP].sum,b[x].sum,sizeof(b[x].sum));

for(int i=del+1;i<=b[x].sz;++i){

b[newP].a[i-del]=b[x].a[i];

b[x].cnt[pos[b[x].a[i]]]--;b[x].sum[b[x].a[i]]--;

b[x].a[i]=0;

}

b[x].sz=del;

}

void ins(int x,int val){

int now=st;

while(b[now].sz<x){

if(b[now].r)x-=b[now].sz,now=b[now].r;

else break;

}

for(int i=b[now].sz;i>=x;--i)b[now].a[i+1]=b[now].a[i];

b[now].a[x]=val;b[now].sz++;

int on=now;

while(now){//维护值域分块

b[now].cnt[pos[val]]++;

b[now].sum[val]++;

now=b[now].r;

}

if(b[on].sz>=big\*2)split(on);

}

int main(){

scanf("%d",&n);

init();

scanf("%d",&m);

int op,ans=0,x,y,k;

for(int i=1;i<=m;++i){

char s[2];

scanf("%s",s);

if(s[0]=='Q'){

scanf("%d%d%d",&x,&y,&k);

x^=ans;y^=ans;k^=ans;

cout<<(ans=kth(x,y,k))<<"\n";

}

else if(s[0]=='M'){

scanf("%d%d",&x,&y);

x^=ans;y^=ans;

update(x,y);

}

else{

scanf("%d%d",&x,&y);

x^=ans;y^=ans;

ins(x,y);

}

}

return 0;

}

## 莫队

### 静态莫队

#include<bits/stdc++.h>// O(nsqrt(m)) 在块大小等于n/sqrt(m)最优

using namespace std;

const int maxn=1e5+5;

int pos[maxn],big,a[maxn],n,m,cnt[maxn],pl,pr,ans[maxn],nowans;

struct Q{

int L,R,id;

bool operator<(const Q&a){

if(pos[L]!=pos[a.L])return pos[L]<pos[a.L];

if(pos[L]&1)return R>a.R;

return R<a.R;

}

}q[maxn];

void init(){

big=n/sqrt(m);

for(int i=1;i<=n;++i)pos[i]=(i-1)/big+1;

}

void add(int x){

if((++cnt[a[x]])==1)nowans++;

}

void del(int x){

if((--cnt[a[x]])==0)nowans--;

}

int main(){

scanf("%d%d",&n,&m);

for(int i=1;i<=n;++i)scanf("%d",&a[i]);

init();

for(int i=1;i<=m;++i)scanf("%d%d",&q[i].L,&q[i].R),q[i].id=i;

sort(q+1,q+1+m);

for(int i=1;i<=m;++i){

int l=q[i].L,r=q[i].R;

while(pl<l)del(pl++);

while(pl>l)add(--pl);

while(pr<r)add(++pr);

while(pr>r)del(pr--);

if(nowans==(r-l+1))ans[q[i].id]=1;

}

for(int i=1;i<=m;++i)

puts(ans[i]?"Yes":"No");

return 0;

}

### 回滚莫队

应用于删除或插入某种操作难以实现时，只删除或只插入

按左端点所属块排序，枚举块，左右端点同块的时候暴力，否则右端点单调，左端点回滚 O(msqrt(n)+nsqrt(n))

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int maxn=1e5+5;

const int mod=998244353;

int a[maxn],pos[maxn],L[maxn],R[maxn],big,n,m,num,pre[maxn],nxt[maxn],cnt[maxn],ans[maxn],tot,Tong[maxn];

struct Q{

int l,r,k,id;

bool operator<(const Q&x)const{

return (pos[l]^pos[x.l])?(pos[l]<pos[x.l]):r<x.r;

}

}q[maxn];

void init(){

big=sqrt(n);

num=(n-1)/big+1;

for(int i=1;i<=num;++i){

L[i]=R[i-1]+1;

R[i]=i\*big;

}

R[num]=n;

for(int i=1;i<=num;++i)

for(int j=L[i];j<=R[i];++j)

pos[j]=i;

}

void add(int x,int&ans){

cnt[x]++;

if(cnt[x]==1){

pre[x]=nxt[x]=x;

if(cnt[x-1])pre[x]=pre[x-1];

if(cnt[x+1])nxt[x]=nxt[x+1];

if(pre[x])nxt[pre[x]]=nxt[x];

if(nxt[x])pre[nxt[x]]=pre[x];

ans=max(ans,nxt[x]-pre[x]+1);

}

}

void del(int x){

cnt[x]--;

if(!cnt[x]){

pre[x]=nxt[x]=0;//保证链的端点对就好了

if(pre[x-1])

nxt[pre[x-1]]=nxt[x-1]=x-1;

if(nxt[x+1])

pre[nxt[x+1]]=pre[x+1]=x+1;

}

}

void Clear(int x){

cnt[x]=pre[x]=nxt[x]=0;

}

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>n>>m;

init();

for(int i=1;i<=n;++i)cin>>a[i];

for(int i=1;i<=m;++i){

cin>>q[i].l>>q[i].r>>q[i].k;

q[i].id=i;

}

sort(q+1,q+1+m);

for(int i=1,now=1;i<=num;++i){ //枚举块

int pl=R[i]+1,pr=R[i];//

for(int i=1;i<=n;++i)cnt[i]=pre[i]=nxt[i]=0;

int nowans=0;

for(;pos[q[now].l]==i;now++){

if(pos[q[now].l]==pos[q[now].r]){ //同块暴力

int tmpans=0,sum=0;

int k=q[now].k;

for(int j=q[now].l;j<=q[now].r;++j){

add(a[j],tmpans);

Tong[++tot]=a[j];

}

sum+=tmpans;

int tmpL=q[now].l,tmpR=q[now].r;

for(int j=1,f=(n+1);j<=k-1;++j,f=(ll)f\*(n+1)%mod){

add(a[tmpL-j],tmpans);

add(a[tmpR+j],tmpans);

Tong[++tot]=a[tmpL-j],Tong[++tot]=a[tmpR+j];

sum=(sum+(ll)tmpans\*f)%mod;

}

while(tot)del(Tong[tot--]);

// for(int j=tmpL-(k-1);j<=tmpR+(k-1);++j)

// Clear(a[j]);

ans[q[now].id]=sum;

}else{ //不同块右端点单调，左端点回滚

int sum=0,k=q[now].k;

while(pr<q[now].r)

add(a[++pr],nowans);

int tmp=nowans,tmpL=pl;

while(pl>q[now].l){

add(a[--pl],nowans),Tong[++tot]=a[pl];

}

sum+=nowans;

sum%=mod;

for(int j=1,f=(n+1);j<=k-1;++j,f=(ll)f\*(n+1)%mod){

add(a[pl-j],nowans);

add(a[pr+j],nowans);

Tong[++tot]=a[pl-j],Tong[++tot]=a[pr+j];

sum=(sum+(ll)nowans\*f)%mod;

}

ans[q[now].id]=sum;

if(pos[q[now+1].l]==i){ //注意回滚的时候是当前块全部回滚还是只回滚一个询问

while(tot)del(Tong[tot--]);

}else{

for(int j=pl-(k-1);j<=pr+(k-1);++j)Clear(a[j]);

tot=0;

}

pl=tmpL,nowans=tmp;

}

}

}

for(int i=1;i<=m;++i)cout<<ans[i]<<"\n";

return 0;

}

## 可删堆

struct EraseHeap{//两个堆维护可删堆

priority\_queue<int>a,b;//a存所有状态,b存无用状态

void push(int x){

a.push(x);

}

void erase(int x){

b.push(x);

}

void pop(){

while(!b.empty()&&a.top()==b.top())

a.pop(),b.pop();

a.pop();

}

int top(){//最大

while(!b.empty()&&a.top()==b.top())

a.pop(),b.pop();

return a.empty()?0:a.top();

}

int size(){

return a.size()-b.size();

}

int stop(){//次大

if(size()<2)return 0;

int x=top();pop();

int y=top();push(x);

return y;

}

};

## 可并堆

struct EraseHeap{//两个堆维护可删堆

priority\_queue<int>a,b;//a存所有状态,b存无用状态

void push(int x){

a.push(x);

}

void erase(int x){

b.push(x);

}

void pop(){

while(!b.empty()&&a.top()==b.top())

a.pop(),b.pop();

a.pop();

}

int top(){//最大

while(!b.empty()&&a.top()==b.top())

a.pop(),b.pop();

return a.empty()?0:a.top();

}

int size(){

return a.size()-b.size();

}

int stop(){//次大

if(size()<2)return 0;

int x=top();pop();

int y=top();push(x);

return y;

}

};

### 左偏树

//1.pushUp删除任意结点才需，只是根直接merge

//2.假如有find操作，pop删除的时候根要指清楚

#include<bits/stdc++.h>

using namespace std;

const int maxn=100010;

int n,m,op,x,y,tot;

int fa[maxn];

bool tf[maxn];

struct Node{

int val,lc,rc,dis,id;

bool operator<(Node x)const{ return val==x.val?id<x.id:val<x.val;}

}tr[maxn];

int find(int x){

if(x==fa[x])return x;

return fa[x]=find(fa[x]);

}

void pushUp(int x){ //删除任意结点时用 O(logn)

if(!x)return;

if(tr[x].dis!=tr[tr[x].rc].dis+1){

tr[x].dis=tr[tr[x].rc].dis+1;

pushUp(fa[x]);

}

}

int merge(int x,int y){

if(!x||!y)return x+y;

if(tr[x].val>tr[y].val)swap(x,y);

tr[x].rc=merge(tr[x].rc,y);

fa[tr[x].rc]=x;//删除任意结点 若只是根不需要

if(tr[tr[x].lc].dis<tr[tr[x].rc].dis)swap(tr[x].lc,tr[x].rc);

tr[x].dis=tr[tr[x].rc].dis+1;

pushUp(x);//删除任意结点时只需直接merge左右儿子即可 若只是根则不需要

return x;

}

int build() {//O(n)建树

queue<int> q;

for (int i = 1; i <= n; ++i)

q.push(i);

while (!q.empty()) {

if (q.size() == 1) break;

else {

int x = q.front(); q.pop();

int y = q.front(); q.pop();

q.push(merge(x, y));

}

}

int finally = q.front(); q.pop();

return finally;

}

void init(int x){

for(int i=0;i<=x;++i){

tr[i].lc=tr[i].rc=tr[i].dis=0;

fa[i]=i;

}

}

int add(int val,int x){

tr[tot].lc=tr[tot].rc=tr[tot].dis=0;

tr[tot++].val=val;

return merge(tot-1,x);

}

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>n>>m;

init(n);

for(int i=1;i<=n;++i)cin>>tr[i].val;

while(m--){

cin>>op>>x;

if(op==1){ //合并第x个数和第y个数的小根堆

cin>>y;

if(tf[x]||tf[y])continue;

x=find(x);y=find(y);

if(x!=y)fa[x]=fa[y]=merge(x,y);//这一步必须有

}

if(op==2){ //输出第x个数所在堆最小数并删除

if(tf[x]){ cout<<-1<<"\n";continue;}

x=find(x);

cout<<tr[x].val<<"\n";

tf[x]=1;

fa[tr[x].lc]=fa[tr[x].rc]=fa[x]=merge(tr[x].lc,tr[x].rc);//删除根结点 并查集找根必须有fa的维护 否则pushUp就好了

tr[x].lc=tr[x].rc=tr[x].dis=0;//pop

}

}

}

## 整体二分

### 区间带修第k小

#include<bits/stdc++.h>

using namespace std;

const int maxn=1e5+10;

int n,m,a[maxn\*3],val[maxn\*3],b[maxn\*3],tot,cnt;//数组大小注意到底是几倍

struct BIT{

#define lowb(i) (i&(-i))

int c[maxn];

void add(int x,int val){

for(int i=x;i<=n;i+=lowb(i))

c[i]+=val;

}

int ask(int x){

int ans=0;

for(int i=x;i;i-=lowb(i))

ans+=c[i];

return ans;

}

}bit;

struct Opt {

int x, y, k, type, id;

// 对于询问, type = 1, x, y 表示区间左右边界, k 表示询问第 k 小

// 对于修改, type = 0, x 表示修改位置, y 表示修改后的值,

// k 表示当前操作是插入(1)还是擦除(-1), 更新树状数组时使用.

// id 记录每个操作原先的编号, 因二分过程中操作顺序会被打散

}q[maxn\*3],q1[maxn\*3],q2[maxn\*3];

int ans[maxn];

void solve(int l,int r,int ql,int qr){

if(l>r||ql>qr)return;

if(l==r){

for(int i=ql;i<=qr;++i){

if(q[i].type==1)ans[q[i].id]=l;

}

return;

}

int cnt1=0,cnt2=0,mid=l+r>>1;

for(int i=ql;i<=qr;++i){

if(q[i].type==1){

int x=bit.ask(q[i].y)-bit.ask(q[i].x-1);

if(q[i].k<=x)

q1[++cnt1]=q[i];

else{

q[i].k-=x;

q2[++cnt2]=q[i];

}

}else{

if(q[i].y<=mid){

bit.add(q[i].x,q[i].k);//贡献给>mid

q1[++cnt1]=q[i];

}else{

q2[++cnt2]=q[i];

}

}

}

for(int i=1;i<=cnt1;++i){//roll back

if(!q1[i].type)

bit.add(q1[i].x,-q1[i].k);

}

for(int i=1;i<=cnt1;++i)q[i+ql-1]=q1[i];

for(int i=1;i<=cnt2;++i)q[ql+i+cnt1-1]=q2[i];

solve(l,mid,ql,ql+cnt1-1);

solve(mid+1,r,ql+cnt1,qr);

}

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>n>>m;

for(int i=1;i<=n;++i)cin>>a[i],b[++cnt]=a[i];

for(int i=1;i<=n;++i){//初始化看成插入

q[++tot]={i,a[i],1,0,0};

}

char s[2];

int l,r,k;

for(int i=1;i<=m;++i)ans[i]=-1;

for(int i=1;i<=m;++i){

cin>>s;

if(s[0]=='Q'){

cin>>l>>r>>k;

q[++tot]={l,r,k,1,i};

}else{//a\_l改为k

cin>>l>>k;

b[++cnt]=k;

q[++tot]={l,a[l],-1,0,i};

a[l]=k;

q[++tot]={l,k,1,0,i};

}

}

//离散化

sort(b+1,b+1+cnt);

cnt=unique(b+1,b+1+cnt)-(b+1);

for(int i=1;i<=cnt;++i){

val[i]=b[i];

}

for(int i=1;i<=tot;++i){

if(q[i].type==0)

q[i].y=lower\_bound(b+1,b+1+cnt,q[i].y)-b;

}

solve(1,cnt,1,tot);

for(int i=1;i<=m;++i){

if(ans[i]!=-1)cout<<val[ans[i]]<<"\n";

}

return 0;

}

# 树论

## dfs序与欧拉序

**浅谈树序与欧拉序与结合的相关问题**

在本文章中，序指dfs中点的入栈顺序，而出栈顺序表示该点子树的结尾区间

void dfs(int x,int fa){  
in[x]=++ti;  
  ....  
   out[x]=ti;  
}

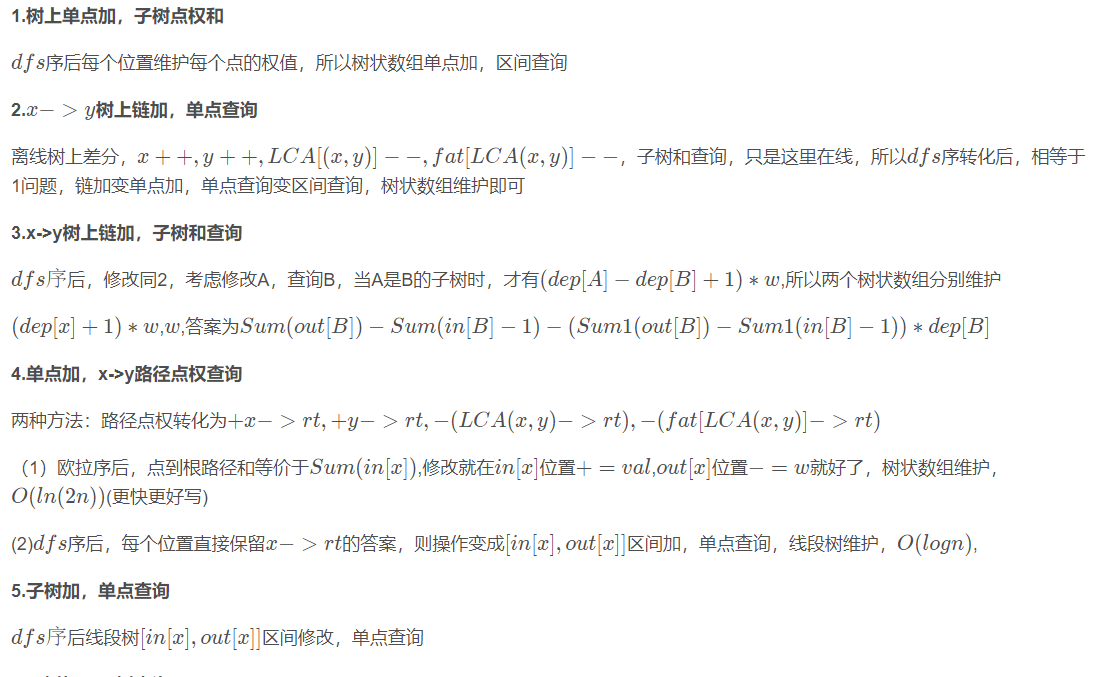
欧拉序长为​,且一般记录对应位置的符号

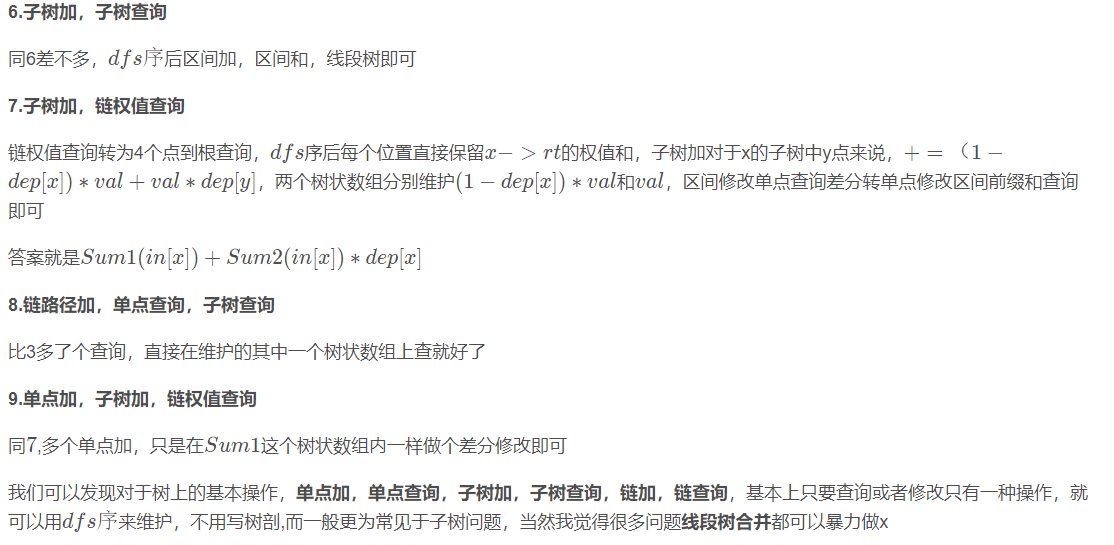
void dfs(int x,int fa){  
   in[x]=++ti;f[ti]=1;  
  ...  
   out[x]=++ti;f[ti]=-1  
}

两者都可将子树线性化，只是面对不同问题，代码复杂度与维护难度稍有不同，基本上dfs序能做的，欧拉序都能做，有些问题我会列举出来，有些不会

诚然，里面很多问题树剖都可以替代dfs序和欧拉序，但是还是希望大家在遇到**子树线性化与某点到根的链计算**维护的时候可以先思考这两者，再考虑树剖，复杂的树链问题当然就使用树剖了

下面先给出关于这两者的7个经典问题及其解法





## LCA

//倍增 nlogn – logn

const int maxn=5e5+5;

int f[maxn][21],t,n,m,d[maxn],tot=0,lg[maxn],s;

int head[maxn],ver[maxn<<1],next1[maxn<<1];

void add(int x,int y){

ver[++tot]=y,next1[tot]=head[x],head[x]=tot;

}

void dfs(int x,int fa){

d[x]=d[fa]+1;//d[1]=1必须

f[x][0]=fa;

for(int i=1;i<=lg[d[x]];++i)

f[x][i]=f[f[x][i-1]][i-1];

for(int i=head[x];i;i=next1[i]){

int v=ver[i];

if(v==fa)continue;

dfs(v,x);

}

}

int lca(int x,int y){

if(d[x]<d[y])swap(x,y);

while(d[x]>d[y])

x=f[x][lg[d[x]-d[y]]-1];

if(x==y)return x;

for(int i=lg[d[y]]-1;i>=0;--i)

if(f[x][i]!=f[y][i])x=f[x][i],y=f[y][i];

return f[x][0];

}

void init(){

for(int i=1;i<=n;++i) //预处理log2(n)+1

lg[i]=lg[i-1]+(1<<lg[i-1]==i);

}

### RMQ

int dfn[maxn],d[maxn],st[maxn<<1][21],lg[maxn<<1],ti; O(nlogn)-O(1)

void dfs(int x,int f){

dfn[x]=++ti;d[x]=d[f]+1;st[ti][0]=x;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f)continue;

dfs(y,x);

st[++ti][0]=x;

}

}

void RMQ(){

for(int i=2;i<=ti;++i)lg[i]=lg[i>>1]+1;

for(int j=1;j<=lg[ti];++j)

for(int i=1;(i+(1<<j)-1)<=ti;++i){

int r=i+(1<<(j-1));

st[i][j]=d[st[i][j-1]]<d[st[r][j-1]]?st[i][j-1]:st[r][j-1];

}

}

int query(int l,int r){

if(l>r)swap(l,r);

int k=lg[r-l+1];

return d[st[l][k]]<d[st[r-(1<<k)+1][k]]?st[l][k]:st[r-(1<<k)+1][k];

}

void init(int rt){

dfs(rt,0);

RMQ();

}

int LCA(int x,int y){

return query(dfn[x],dfn[y]);

}

### 树剖详见重剖板子

O(nlogn)-O(log) 常数小

## Kruskal重构树

//常用于解决从图上某点出发经过边权不超过x能到达的点

struct Edge{

int u,v,w;

bool operator<(const Edge&a){

return w<a.w;//升序最大边权最小 降序最小边权最大 (u,v)上答案=LCA(u,v)

}

}edge[M];

vector<int>G[M];//M大小开两倍 树上2\*n-1个点

int cnt,fa[M],f[M][22],col[M],Q,d[M],rt[M],ans[M],n,m,lg[M],val[M],t;

int find(int x){

if(x==fa[x])return x;

return fa[x]=find(fa[x]);

}

void exKruskal(){

cnt=n;

for(int i=1;i<(n<<1);++i)fa[i]=i;

sort(edge+1,edge+1+m);

for(int i=1;i<=m;++i){

int xx=find(edge[i].u),yy=find(edge[i].v);

if(xx==yy)continue;

val[++cnt]=edge[i].w;

fa[xx]=fa[yy]=cnt;

G[xx].pb(cnt);G[cnt].pb(xx);

G[yy].pb(cnt);G[cnt].pb(yy);

}

dfs(cnt,0);//处理生成树信息

}

## 重链剖分

//树剖模板点权链加、链和查询、子树加、子树和、lca、换根

#include<bits/stdc++.h>

#define ls p<<1

#define rs p<<1|1

#define lson p<<1,l,mid

#define rson p<<1|1,mid+1,r

using namespace std;

typedef long long ll;

const int maxn=1e5+5;

int head[maxn],next1[maxn<<1],ver[maxn<<1],tot,n,m,a[maxn],op,u,v,rt,nval[maxn],ti;

void add(int x,int y){

ver[++tot]=y,next1[tot]=head[x],head[x]=tot;

}

struct SegmentTree{

ll sum[maxn<<2],add[maxn<<2];

void pushUp(int p){

sum[p]=sum[ls]+sum[rs];

}

void pushDown(int p,int l,int r){

int mid=l+r>>1;

sum[ls]+=1ll\*(mid-l+1)\*add[p];

sum[rs]+=1ll\*(r-mid)\*add[p];

add[ls]+=add[p];add[rs]+=add[p];

add[p]=0;

}

void build(int p,int l,int r){

add[p]=0;

if(l==r){

sum[p]=nval[l];

return;

}

int mid=l+r>>1;

build(lson);

build(rson);

pushUp(p);

}

ll query(int p,int l,int r,int L,int R){

if(L<=l&&r<=R)return sum[p];

if(add[p])

pushDown(p,l,r);

int mid=l+r>>1;

ll ans=0;

if(L<=mid)ans+=query(lson,L,R);

if(R>mid)ans+=query(rson,L,R);

return ans;

}

void update(int p,int l,int r,int L,int R,int val){

if(L<=l&&r<=R){

sum[p]+=1ll\*(r-l+1)\*val;

add[p]+=val;return;

}

if(add[p])

pushDown(p,l,r);

int mid=l+r>>1;

if(L<=mid)update(lson,L,R,val);

if(R>mid)update(rson,L,R,val);

pushUp(p);

}

}tr;

struct HCD{

int fa[maxn],son[maxn],dep[maxn],sz[maxn],top[maxn],in[maxn];

//树剖部分

void dfs1(int x,int f){//第一遍处理父亲、深度、重儿子、子树大小

fa[x]=f;sz[x]=1;

dep[x]=dep[f]+1;

int mxson=-1;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f)continue;

dfs1(y,x);

sz[x]+=sz[y];

if(sz[y]>mxson)son[x]=y,mxson=sz[y];//记录重儿子

}

}

void dfs2(int x,int t){//重新标号赋值、记录链顶端

in[x]=++ti;

nval[ti]=a[x];

top[x]=t;

if(!son[x])return;//叶子

dfs2(son[x],t);//先重后轻

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==fa[x]||y==son[x])continue;

dfs2(y,y);//轻边自己开始

}

}

void init(int x){

dfs1(x,0);dfs2(x,x);

}

//求lca

int lca(int x,int y){

while(top[x]!=top[y]){

if(dep[top[x]]<dep[top[y]])swap(x,y);

x=fa[top[x]];

}

if(dep[x]>dep[y])swap(x,y);

return x;

}

//链查询

ll Pquery(int x,int y){//顶端深度深的跳，直到相同，修改同理

ll ans=0;

while(top[x]!=top[y]){

if(dep[top[x]]<dep[top[y]])swap(x,y);

ans+=tr.query(1,1,ti,in[top[x]],in[x]);

x=fa[top[x]];

}

if(dep[x]<dep[y])swap(x,y);

ans+=tr.query(1,1,ti,in[y],in[x]);

return ans;

}

//链修改

void Pupdate(int x,int y,int val){

while(top[x]!=top[y]){

if(dep[top[x]]<dep[top[y]])swap(x,y);

tr.update(1,1,ti,in[top[x]],in[x],val);

x=fa[top[x]];

}

if(dep[x]<dep[y])swap(x,y);

tr.update(1,1,ti,in[y],in[x],val);

}

//找y的祖先中x的儿子

int findson(int x,int y){

while(top[x]!=top[y]){

if(fa[top[y]]==x)

return top[y];

y=fa[top[y]];

}

return son[x];

}

//可判断换根子树修改

void Sonupdate(int x,int val){

if(lca(x,rt)!=x)//新根在x子树外，无影响

tr.update(1,1,ti,in[x],in[x]+sz[x]-1,val);

else if(x==rt)//本身一颗

tr.update(1,1,ti,1,ti,val);

else{//整颗减去新根的祖先中x为根的子树

int s=findson(x,rt);

tr.update(1,1,ti,1,in[s]-1,val);

if(in[s]+sz[s]<=n)

tr.update(1,1,ti,in[s]+sz[s],ti,val);

}

}

//可判断换根子树查询

ll Sonquery(int x){

if(lca(x,rt)!=x)

return tr.query(1,1,ti,in[x],in[x]+sz[x]-1);

else if(x==rt)

return tr.sum[1];

else{

int s=findson(x,rt);

ll ans=0;

ans+=tr.query(1,1,ti,1,in[s]-1);

if(in[s]+sz[s]<=n)

ans+=tr.query(1,1,ti,in[s]+sz[s],ti);

return ans;

}

}

}hcd;

int main(){

rt=1;

scanf("%d",&n);

for(int i=1;i<=n;++i)scanf("%d",&a[i]);

int x;

for(int i=1;i<=n-1;++i){

scanf("%d",&x);

add(i+1,x);

add(x,i+1);

}

hcd.init(1);

tr.build(1,1,ti);

scanf("%d",&m);

for(int i=1;i<=m;++i){

scanf("%d",&op);

if(op==1){//换根

scanf("%d",&rt);

}else if(op==2){//链修改

scanf("%d%d%d",&u,&v,&x);

hcd.Pupdate(u,v,x);

}else if(op==3){//子树修改

scanf("%d%d",&u,&x);

hcd.Sonupdate(u,x);

}else if(op==4){//链查询

scanf("%d%d",&u,&v);

cout<<hcd.Pquery(u,v)<<"\n";

}else{//子树查询

scanf("%d",&u);

cout<<hcd.Sonquery(u)<<"\n";

}

}

return 0;

}

## 长链剖分

### 树上k级祖先

//树上k级祖先 O(nlogn)~O(1)

#include<bits/stdc++.h>

#define pb push\_back

using namespace std;

typedef long long ll;

const int maxn=5e5+5;

int f[maxn][20],n,u,v,len[maxn],head[maxn],next1[maxn<<1],ver[maxn<<1],tot,hbit[maxn],m,lg[maxn];

vector<int>U[maxn],D[maxn];

void add(int x,int y){

ver[++tot]=y,next1[tot]=head[x],head[x]=tot;

}

struct LCD{

int fa[maxn],dep[maxn],md[maxn],hson[maxn],top[maxn];//md[x]表示该点子树的最深深度

void dfs1(int x,int fat){

f[x][0]=fa[x]=fat;

md[x]=dep[x]=dep[fat]+1;

for(int i=1;i<=lg[dep[x]];++i)

f[x][i]=f[f[x][i-1]][i-1];

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==fat)continue;

dfs1(y,x);

if(md[y]>md[hson[x]])hson[x]=y,md[x]=md[y];

}

}

void dfs2(int x,int t){

top[x]=t;

len[x]=md[x]-dep[top[x]];//链长

if(!hson[x])return;

dfs2(hson[x],t);

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==fa[x]||y==hson[x])continue;

dfs2(y,y);

}

}

void init(int x){

dfs1(x,0);dfs2(x,x);

}

int query(int x,int k){//x的k级祖先

if(k>dep[x])return 0;

if(!k)return x;

x=f[x][hbit[k]];k^=(1<<hbit[k]);

if(dep[x]-dep[top[x]]==k)return top[x];

if(dep[x]-dep[top[x]]>k)return D[top[x]][dep[x]-dep[top[x]]-k-1];

return U[top[x]][k-(dep[x]-dep[top[x]])-1];

}

}lcd;

int main(){

for(int i=1;i<=n;++i)

lg[i]=lg[i-1]+(1<<lg[i-1]==i);

lcd.init(rt);

for(int i=1;i<=n;++i){

if(i==lcd.top[i]){

int l=0,x=i;

while(l<len[i]&&x)x=f[x][0],++l,U[i].pb(x);

l=0,x=i;

while(l<len[i])x=lcd.hson[x],++l,D[i].pb(x);

}

}

int mx=1;

for(int i=1;i<=n;++i){

if((i>>mx)&1)++mx;

hbit[i]=mx-1;

}

for(int i=1;i<=m;++i){

cin>>u>>k;

cout<<lcd.query(u,k)<<"\n";

}

return 0;

}

### 长链剖分优化DP模板

//f[i][j]表示i子树中距i为j的方案数，g[i][j]表示i子树中(x,y)->lca(x,y)距离为d,lca(x,y)->i距离为d-j方案数

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int maxn=1e5+5;

ll \*f[maxn],\*g[maxn],tmp[maxn\*3],\*id=tmp,ans;

int md[maxn],dep[maxn],hson[maxn],head[maxn],next1[maxn<<1],ver[maxn<<1],tot,n;

void add(int x,int y){

ver[++tot]=y,next1[tot]=head[x],head[x]=tot;

}

void dfs1(int x,int f){

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f)continue;

dfs1(y,x);

if(md[y]>md[hson[x]])hson[x]=y;

}

md[x]=md[hson[x]]+1;//md表示x的高度 叶子为1

}

void dfs(int x,int fa){

if(hson[x])

f[hson[x]]=f[x]+1,g[hson[x]]=g[x]-1,dfs(hson[x],x);

f[x][0]=1;ans+=g[x][0];//重儿子统计

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==fa||y==hson[x])continue;

f[y]=id;id+=md[y]<<1;g[y]=id;id+=md[y];

dfs(y,x);

for(int j=0;j<md[y];++j){

if(j)

ans+=f[x][j-1]\*g[y][j];//轻儿子在此统计了g[y][1]即对g[x][0]的贡献

ans+=g[x][j+1]\*f[y][j];//最多到md[y]-1

}

for(int j=0;j<md[y];++j){

g[x][j+1]+=f[x][j+1]\*f[y][j];

if(j)g[x][j-1]+=g[y][j];

f[x][j+1]+=f[y][j];//最多可以更新到md[y]

}

}

}

int main(){

scanf("%d",&n);

int a,b;

for(int i=1;i<n;++i){

scanf("%d%d",&a,&b);

add(a,b);add(b,a);

}

dfs1(1,0);

f[1]=id;id+=md[1]<<1;

g[1]=id;id+=md[1];

dfs(1,0);

cout<<ans<<"\n";

return 0;

}

### 长链剖分+后缀和套路

//<=k限制常转>=k

#include<bits/stdc++.h>

#define pb push\_back

#define fi first

#define se second

using namespace std;

typedef long long ll;

const int maxn=3e5+5;

ll \*dp[maxn],tmp[maxn],\*id=tmp,ans[maxn];

int sz[maxn],head[maxn],ver[maxn<<1],next1[maxn<<1],n,q,hson[maxn],md[maxn],dep[maxn],tot;

typedef pair<int,int>P;//dp[i][j]以i为根子树距离i>=j下,ab为c祖先，bc在a子树下的数量

vector<P>Q[maxn];

void dfs1(int x,int f){

dep[x]=dep[f]+1;sz[x]=1;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f)continue;

dfs1(y,x);

sz[x]+=sz[y];

if(md[y]>md[hson[x]])hson[x]=y;

}

md[x]=md[hson[x]]+1;

}

void add(int x,int y){

ver[++tot]=y,next1[tot]=head[x],head[x]=tot;

}

void dfs(int x,int f){

dp[x][0]=sz[x]-1;

if(hson[x]){//指针，长链直接赋值不用+=

dp[hson[x]]=dp[x]+1;dfs(hson[x],x);dp[x][0]+=dp[hson[x]][0];//但>=0没统计

}

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f||y==hson[x])continue;

dp[y]=id;id+=md[y];

dfs(y,x);

dp[x][0]+=dp[y][0];

for(int j=0;j<md[y];++j)//后缀和维护

dp[x][j+1]+=dp[y][j];

}

for(auto&v:Q[x]){//统计答案的时候减法获取需要答案即可

ans[v.fi]+=1ll\*(sz[x]-1)\*min(dep[x]-1,v.se);

if(v.se<md[x]-1)ans[v.fi]+=dp[x][0]-dp[x][v.se+1]-(sz[x]-1);

else ans[v.fi]+=dp[x][0]-(sz[x]-1);

}

}

int main(){

scanf("%d%d",&n,&q);

int u,v;

for(int i=1;i<n;++i){

scanf("%d%d",&u,&v);

add(u,v);add(v,u);

}

dfs1(1,0);

dp[1]=id;id+=md[1];

for(int i=1;i<=q;++i){

scanf("%d%d",&u,&v);

Q[u].pb({i,v});

}

dfs(1,0);

for(int i=1;i<=q;++i)cout<<ans[i]<<"\n";

return 0;

}

## 点分治

### 容斥

#include<iostream>//路径权值和<=k数量

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

const int maxn=1e4+5;

const int INF=2e9;

int head[maxn],ver[maxn<<1],next1[maxn<<1],mxson[maxn],sz[maxn],d[maxn],cnt,q[maxn],rt,S,ans,tot,n,k,edge[maxn<<1];

bool v[maxn];

void add(int x,int y,int z){

ver[++tot]=y,next1[tot]=head[x],edge[tot]=z,head[x]=tot;

}

void getRoot(int x,int f){//求重心

sz[x]=1;mxson[x]=0;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f||v[y])continue;

getRoot(y,x);

sz[x]+=sz[y];

mxson[x]=max(mxson[x],sz[y]);

}

mxson[x]=max(mxson[x],S-sz[x]);

if(mxson[x]<mxson[rt])rt=x;

}

void getQue(int x,int f){

q[++cnt]=d[x];

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f||v[y])continue;

d[y]=d[x]+edge[i];

getQue(y,x);

}

}

int cal(int x,int cost){

d[x]=cost;cnt=0;

getQue(x,0); //获取离当前点的距离

sort(q+1,q+1+cnt);

int l=1,r=cnt,sum=0;

while(l<r){

if(q[l]+q[r]<=k){

sum+=r-l;l++;

}else r--;

}

return sum;

}

void init(int x){

S=sz[x];

rt=0;

getRoot(x,0);

}

void dfz(int x){

ans+=cal(x,0);

v[x]=1;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(v[y])continue;

ans-=cal(y,edge[i]);

init(y);

dfz(rt);

}

}

int main(){

while(~scanf("%d%d",&n,&k)&&n&&k){

int a,b,c;

tot=rt=ans=0;

for(int i=1;i<=n;++i)v[i]=head[i]=0;

for(int i=1;i<n;++i){

scanf("%d%d%d",&a,&b,&c);

add(a,b,c);add(b,a,c);

}

mxson[0]=INF;S=n;

getRoot(1,0);

dfz(rt);

cout<<ans<<"\n";

}

}

### 枚举子树

#include<bits/stdc++.h>//路径权值和恰好为k且边数量最少

#define fi first

#define se second

using namespace std;

typedef long long ll;

const int maxn=2e5+5;

const int M=1e6+5;

int n,k,head[maxn],ver[maxn<<1],next1[maxn<<1],edge[maxn<<1],tot,sz[maxn],mxson[maxn],rt,S,d[maxn],judge[M],cnttmp,a,b,c,ans=maxn;

pair<int,int>tmp[maxn];

int val[maxn];

bool v[maxn];

void add(int x,int y,int z){

ver[++tot]=y,next1[tot]=head[x],edge[tot]=z,head[x]=tot;

}

void getRoot(int x,int f){

sz[x]=1;mxson[x]=0;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f||v[y])continue;

getRoot(y,x);

sz[x]+=sz[y];

mxson[x]=max(mxson[x],sz[y]);

}

mxson[x]=max(mxson[x],S-mxson[x]);

if(mxson[x]<mxson[rt])rt=x;

}

void init(int x){

S=sz[x];

mxson[rt=0]=maxn;

getRoot(x,0);

}

void getQue(int x,int f,int dd){

if(d[x]>k)return;

tmp[++cnttmp]={d[x],dd};

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(v[y]||y==f)continue;

d[y]=d[x]+edge[i];

getQue(y,x,dd+1);

}

}

void cal(int x){

int num=0;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(v[y])continue;

d[y]=edge[i];

cnttmp=0;

getQue(y,x,1);

for(int j=cnttmp;j;--j)

if(judge[k-tmp[j].fi]!=maxn)

ans=min(ans,judge[k-tmp[j].fi]+tmp[j].se);

for(int j=cnttmp;j;--j)

val[++num]=tmp[j].fi,judge[tmp[j].fi]=min(judge[tmp[j].fi],tmp[j].se);

}

for(int i=num;i;--i)

judge[val[i]]=maxn;

}

void dfz(int x){

v[x]=1;judge[0]=0;

cal(x);

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(v[y])continue;

init(y);

dfz(rt);//点分治一定是跳根!!

}

}

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>n>>k;

for(int i=1;i<n;++i){

cin>>a>>b>>c;

a++;b++;

add(a,b,c);add(b,a,c);

}

S=n;

mxson[rt=0]=maxn;

for(int i=1;i<M;++i)judge[i]=maxn;

getRoot(1,0);

dfz(rt);

if(ans==maxn)cout<<"-1"<<"\n";

else cout<<ans<<"\n";

return 0;

}

### 点分+dfs序上dp

#include<bits/stdc++.h>

using namespace std;

const int mod=1e9+7;

const int maxn=1005;

int dp[1005][1026],head[maxn],ver[maxn<<1],next1[maxn<<1],tot,ans[1026],a[maxn],sz1[maxn],sz2[maxn],mxson[maxn],S,rt,ti=0,dfn[maxn],m,n;

bool v[maxn];

void add\_edge(int x,int y){//连通块点权异或和为0~k的方案数

ver[++tot]=y,next1[tot]=head[x],head[x]=tot;

}

void add(int&x,int y){

x+=y;

if(x>=mod)x-=mod;

}

void init1(){

tot=0;

for(int i=1;i<=n;++i)head[i]=0;

for(int i=0;i<m;++i)ans[i]=0;

}

void getRoot(int x,int f){

sz1[x]=1;mxson[x]=0;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f||v[y])continue;

getRoot(y,x);

sz1[x]+=sz1[y];

mxson[x]=max(mxson[x],sz1[y]);

}

mxson[x]=max(mxson[x],S-mxson[x]);

if(mxson[x]<mxson[rt])rt=x;

}

void init(int x){

S=sz1[x];

mxson[rt=0]=maxn;

getRoot(x,0);

}

void dfs(int x,int f){

sz2[x]=1;dfn[++ti]=x;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f||v[y])continue;

dfs(y,x);

sz2[x]+=sz2[y];

}

}

void cal(int x){//重心必选

ti=0;

dfs(x,0);

for(int i=1;i<=ti;++i)

for(int j=0;j<m;++j)dp[i][j]=0;

dp[1][a[x]]=1;

for(int i=1;i<ti;++i)

for(int j=0;j<m;++j){

add(dp[i+1][j^a[dfn[i+1]]],dp[i][j]);

add(dp[i+sz2[dfn[i+1]]][j],dp[i][j]);

}

for(int i=0;i<m;++i)add(ans[i],dp[ti][i]);

}

void dfz(int x){

v[x]=1;

cal(x);

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(v[y])continue;

init(y);

dfz(rt);

}

v[x]=0;

}

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

int t;

cin>>t;

while(t--){

cin>>n>>m;

for(int i=1;i<=n;++i)cin>>a[i];

init1();

int x,y;

for(int i=1;i<n;++i){

cin>>x>>y;

add\_edge(x,y);add\_edge(y,x);

}

mxson[rt=0]=maxn;

S=n;

getRoot(1,0);

dfz(rt);

for(int i=0;i<m-1;++i){

cout<<ans[i]<<" ";

}

cout<<ans[m-1]<<"\n";

}

return 0;

}

### 点分+FFT统计树上所有路径条数

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef double db;

const db Pi=acos(-1.0);

const int maxn=1e5+5;

const int mod=1e9+7;

int head[maxn],ver[maxn<<1],next1[maxn<<1],tot,sz[maxn],mxson[maxn],S,rt,n,tmp[maxn],cnttmp;//cnt[i]表示长度为i的路径条数 边权都为1

ll a[maxn\*2],cnt[maxn\*2];

bool v[maxn];

void add(int x,int y){

ver[++tot]=y,next1[tot]=head[x],head[x]=tot;

}

struct FFT{

struct CP{

CP(db xx=0,db yy=0){x=xx,y=yy;}

db x,y;

CP operator+(CP const&B)const{return CP(x+B.x,y+B.y);}

CP operator-(CP const&B)const{return CP(x-B.x,y-B.y);}

CP operator\*(CP const&B)const{return CP(x\*B.x-y\*B.y,x\*B.y+y\*B.x);}

}f[maxn\*4];

int n,m,tr[maxn\*4];

void fft(CP\*f,bool flag){

for(int i=0;i<n;++i)

if(i<tr[i])swap(f[i],f[tr[i]]);

for(int p=2;p<=n;p<<=1){

int len=p>>1;

CP tG(cos(2\*Pi/p),sin(2\*Pi/p));

if(!flag)tG.y\*=-1;

for(int k=0;k<n;k+=p){

CP buf(1,0);

for(int l=k;l<k+len;l++){

CP tt=buf\*f[len+l];

f[len+l]=f[l]-tt;

f[l]=f[l]+tt;

buf=buf\*tG;

}

}

}

}

void init(int nn,int mm,ll aa[]){

n=nn;m=mm;

int len;

for(len=1;len<=n+m;len<<=1);

for(int i=0;i<len;++i)f[i]={0,0};

for(int i=0;i<=n;++i)f[i]={aa[i],0};

for(m+=n,n=1;n<=m;n<<=1);

for(int i=0;i<n;++i)

tr[i]=(tr[i>>1]>>1)|((i&1)?n>>1:0);

}

void work(ll c[]){

fft(f,1);

for(int i=0;i<n;++i)f[i]=f[i]\*f[i];

fft(f,0);

for(int i=0;i<=m;++i)c[i]=(ll)((f[i].x)/n+0.5);

}

}fft;

void getRoot(int x,int f){

sz[x]=1;mxson[x]=0;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(v[y]||y==f)continue;

getRoot(y,x);

sz[x]+=sz[y];

mxson[x]=max(mxson[x],sz[y]);

}

mxson[x]=max(mxson[x],S-mxson[x]);

if(mxson[x]<mxson[rt])rt=x;

}

void init(int x){

mxson[rt=0]=maxn;

S=sz[x];

getRoot(x,0);

}

void getQue(int x,int f,int d){

tmp[++cnttmp]=d;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f||v[y])continue;

getQue(y,x,d+1);

}

}

void cal(int x,int flag,int d){

cnttmp=0;

getQue(x,0,d);

int mx=\*max\_element(tmp+1,tmp+1+cnttmp);

for(int i=0;i<=mx;++i)a[i]=0;

for(int i=1;i<=cnttmp;++i)a[tmp[i]]++;

fft.init(mx,mx,a);

fft.work(a);

for(int i=0;i<=min(n,2\*mx);++i)cnt[i]+=flag\*a[i];

}

void dfz(int x){

v[x]=1;

cal(x,1,0);

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(v[y])continue;

cal(y,-1,1);

init(y);

dfz(rt);

}

v[x]=0;

}

int main(){

int u,v;

scanf("%d",&n);

for(int i=1;i<n;++i){

scanf("%d%d",&u,&v);

add(u,v);add(v,u);

}

mxson[rt=0]=maxn;

S=n;

getRoot(1,0);

dfz(rt);

return 0;

}

## 点分树

#include<bits/stdc++.h>//1.查询距离x点为k的点的权值和

//2.单点修改第x个点的权值

using namespace std;

const int maxn=1e5+5;

int head[maxn],ver[maxn<<1],next1[maxn<<1],tot,dfn[maxn],d[maxn],st[maxn<<1][21],lg[maxn<<1],ti,sz[maxn],mxson[maxn],rt,S,mxdep,maxdep[maxn],newS;

int n,m,val[maxn],fa[maxn];

bool v[maxn];

void add(int x,int y){

ver[++tot]=y,next1[tot]=head[x],head[x]=tot;

}

struct BIT{

#define lowb(x) (x&(-x))

vector<int>c;

int N;

void init(int x){

N=x-1;

c.resize(x);

}

void add(int x,int val){

for(int i=x;i<=N;i+=lowb(i))c[i]+=val;

}

int ask(int x){

int ans=0;

for(int i=x;i;i-=lowb(i))ans+=c[i];

return ans;

}

}bit1[maxn],bit2[maxn];//bit1维护自己到自己，bit2维护自己到父亲

void dfs(int x,int f){

dfn[x]=++ti;d[x]=d[f]+1;st[ti][0]=x;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f)continue;

dfs(y,x);

st[++ti][0]=x;

}

}

void RMQ(){

for(int i=2;i<=ti;++i)lg[i]=lg[i>>1]+1;

for(int j=1;j<=lg[n];++j)

for(int i=1;(i+(1<<j)-1)<=ti;++i){

int r=i+(1<<(j-1));

st[i][j]=d[st[i][j-1]]<d[st[r][j-1]]?st[i][j-1]:st[r][j-1];

}

}

int query(int l,int r){

if(l>r)swap(l,r);

int k=lg[r-l+1];

return d[st[l][k]]<d[st[r-(1<<k)+1][k]]?st[l][k]:st[r-(1<<k)+1][k];

}

int LCA(int x,int y){

return query(dfn[x],dfn[y]);

}

void getRoot(int x,int f){

sz[x]=1;mxson[x]=0;

newS++;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f||v[y])continue;

getRoot(y,x);

sz[x]+=sz[y];

mxson[x]=max(mxson[x],sz[y]);

}

mxson[x]=max(mxson[x],S-mxson[x]);

if(mxson[x]<mxson[rt])rt=x;

}

void init(int x){

mxson[rt=0]=maxn;

S=sz[x];

getRoot(x,0);

}

void dfz(int x){

v[x]=1;

sz[x]=newS+1;//用上一层分治重心的大小更新当前分治节点的sz用于询问的时候防止越界（或者直接取.size())

bit1[x].init(newS+2);

bit2[x].init(newS+2);

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(v[y])continue;

newS=0;//用于初始化树状数组

init(y);

fa[rt]=x;

dfz(rt);

}

}

int Q(int x,int k,bool flag){

k=min(k+1,sz[x]); //+1因为vector的偏移

int ans=0;

ans+=(flag)?bit1[x].ask(k):bit2[x].ask(k);

return ans;

}

int getDis(int x,int y){

int lca=LCA(x,y);

return d[x]+d[y]-2\*d[lca];

}

void update(int x,int y){

for(int i=x;i;i=fa[i])bit1[i].add(getDis(x,i)+1,y);

for(int i=x;fa[i];i=fa[i])bit2[i].add(getDis(x,fa[i])+1,y);

}

int main(){

scanf("%d%d",&n,&m);

for(int i=1;i<=n;++i)scanf("%d",&val[i]);

int x,y,op;

for(int i=1;i<n;++i){

scanf("%d%d",&x,&y);

add(x,y);add(y,x);

}

dfs(1,0);//预处理LCA

RMQ();

mxson[rt=0]=maxn;

S=n;

getRoot(1,0);

fa[rt]=0;

dfz(rt);

for(int i=1;i<=n;++i)update(i,val[i]);

int ans=0;

for(int i=1;i<=m;++i){

scanf("%d%d%d",&op,&x,&y);

x^=ans;y^=ans;

if(!op){

ans=0;

ans+=Q(x,y,1);//x子树内距离x<=k的贡献

for(int f=x;fa[f];f=fa[f]){

int dd=getDis(x,fa[f]);

if(y-dd>=0){

ans+=Q(fa[f],y-dd,1)-Q(f,y-dd,0);//减去对当前对父亲加上父亲对自己

}

}

cout<<ans<<"\n";

}else

update(x,y-val[x]),val[x]=y;

}

return 0;

}

//树上全为黑点，每次取反，问最远黑点距离

#include<bits/stdc++.h>

using namespace std;

const int maxn=1e5+5;

struct EraseHeap{//两个堆维护可删堆

priority\_queue<int>a,b;//a存所有状态,b存无用状态

void push(int x){

a.push(x);

}

void erase(int x){

b.push(x);

}

void pop(){

while(!b.empty()&&a.top()==b.top())

a.pop(),b.pop();

a.pop();

}

int top(){//最大

while(!b.empty()&&a.top()==b.top())

a.pop(),b.pop();

return a.empty()?0:a.top();

}

int size(){

return a.size()-b.size();

}

int stop(){//次大

if(size()<2)return 0;

int x=top();pop();

int y=top();push(x);

return y;

}

}a[maxn],b[maxn],ans;//a维护子树节点到其点分树父亲的距离,b维护x儿子的a的最大值(堆顶)

int fa[maxn],mxson[maxn],sz[maxn],rt,S,dfn[maxn],d[maxn],st[maxn<<1][21],lg[maxn<<1],ti;

bool v[maxn],col[maxn];

int n,head[maxn],ver[maxn<<1],next1[maxn<<1],tot,num,m;//num存黑点数量

void add(int x,int y){

ver[++tot]=y,next1[tot]=head[x],head[x]=tot;

}

void dfs(int x,int f){

dfn[x]=++ti;d[x]=d[f]+1;st[ti][0]=x;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f)continue;

dfs(y,x);

st[++ti][0]=x;

}

}

void RMQ(){

for(int i=2;i<=ti;++i)lg[i]=lg[i>>1]+1;

for(int j=1;j<=lg[ti];++j)

for(int i=1;(i+(1<<j)-1)<=ti;++i){

int r=i+(1<<(j-1));

st[i][j]=d[st[i][j-1]]<d[st[r][j-1]]?st[i][j-1]:st[r][j-1];

}

}

inline int LCA(int l,int r){

if(l>r)swap(l,r);

int k=lg[r-l+1];

return d[st[l][k]]<d[st[r-(1<<k)+1][k]]?st[l][k]:st[r-(1<<k)+1][k];

}

int dis(int x,int y){

int lca=LCA(dfn[x],dfn[y]);

return d[x]+d[y]-2\*d[lca];

}

void getRoot(int x,int f){

sz[x]=1;mxson[x]=0;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(v[y]||y==f)continue;

getRoot(y,x);

sz[x]+=sz[y];

mxson[x]=max(mxson[x],sz[y]);

}

mxson[x]=max(mxson[x],S-mxson[x]);

if(mxson[x]<mxson[rt])rt=x;

}

void init(int x){

mxson[rt=0]=maxn;

S=sz[x];

getRoot(x,0);

}

void dfz(int x){

v[x]=1;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(v[y])continue;

init(y);

fa[rt]=x;

dfz(rt);

}

}

void OFF(int x){//考虑 x->i->fa[i]讨论

b[x].push(0);//自己当端点

if(b[x].size()==2)ans.push(b[x].top());//自己当端点的情况一条链

for(int i=x;fa[i];i=fa[i]){//更新

int f=fa[i];

int dd=dis(f,x),tmp=a[i].top();

a[i].push(dd);//更新自己到父亲

if(dd>tmp){

int mx=b[f].top()+b[f].stop(),size=b[f].size();

if(tmp)b[f].erase(tmp);//tmp不在堆顶了

b[f].push(dd);

int now=b[f].top()+b[f].stop();

if(now>mx){

if(size>=2)ans.erase(mx);//mx原来在里面

if(b[f].size()>=2)ans.push(now);

}

}

}

}

void ON(int x){

if(b[x].size()==2)ans.erase(b[x].top());//只剩自己当端点的链删掉

b[x].erase(0);

for(int i=x;fa[i];i=fa[i]){

int f=fa[i];

int dd=dis(f,x),tmp=a[i].top();

a[i].erase(dd);

if(dd==tmp){//可能删的就是堆顶

int mx=b[f].top()+b[f].stop(),size=b[f].size();

b[f].erase(dd);

if(a[i].top())b[f].push(a[i].top());//删除后看看有没有必要更新堆顶

int now=b[f].top()+b[f].stop();

if(now<mx){

if(size>=2)ans.erase(mx);

if(b[f].size()>=2)ans.push(now);

}

}

}

}

int main(){

scanf("%d",&n);

int u,v;

for(int i=1;i<n;++i){

scanf("%d%d",&u,&v);

add(u,v);add(v,u);

}

dfs(1,0);

RMQ();

S=n;

mxson[rt=0]=maxn;

getRoot(1,0);

dfz(rt);

scanf("%d",&m);

for(int i=1;i<=n;++i)col[i]=1;//黑点

char s[2];

for(int i=1;i<=n;++i){

OFF(i);

num++;

}

for(int i=1;i<=m;++i){

scanf("%s",s);

if(s[0]=='G'){

if(num<=1)cout<<num-1<<"\n";

else cout<<ans.top()<<"\n";

}else{

scanf("%d",&u);

if(col[u])ON(u),num--;

else OFF(u),num++;

col[u]^=1;

}

}

return 0;

}

## Dsu on Tree

#include<bits/stdc++.h>

using namespace std;.//子树众数和

typedef long long ll;

const int maxn=3e5+5;

int sz[maxn],son[maxn],hs,tot,head[maxn],next1[maxn<<1],ver[maxn<<1],cnt[maxn],a[maxn],n,mx;

ll ans[maxn],nowans;

void add(int x,int y){

ver[++tot]=y,next1[tot]=head[x],head[x]=tot;

}

void dfs(int x,int f){

sz[x]=1;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f)continue;

dfs(y,x);

sz[x]+=sz[y];

if(sz[y]>sz[son[x]])son[x]=y;

}

}

void cal(int x,int f,int val){

if(val==1){

cnt[a[x]]++;

if(cnt[a[x]]>mx)mx=cnt[a[x]],nowans=a[x];

else if(cnt[a[x]]==mx)nowans+=a[x];

}else cnt[a[x]]--;

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f||y==hs)continue;

cal(y,x,val);

}

}

void dsu(int x,int f,bool op){

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(y==f||y==son[x])continue;

dsu(y,x,0);//先统计完轻儿子的问题

}

if(son[x])dsu(son[x],x,1),hs=son[x];//统计重儿子的贡献不消除影响

cal(x,f,1);//加上轻儿子的贡献

ans[x]=nowans;//更新答案

hs=0;

if(!op)cal(x,f,-1),mx=nowans=0;//到轻边时删掉贡献

}

int main(){

scanf("%d",&n);

for(int i=1;i<=n;++i)scanf("%d",&a[i]);

for(int i=1;i<n;++i){

int x,y;

scanf("%d%d",&x,&y);

add(x,y);add(y,x);

}

dfs(1,0);

dsu(1,0,0);

for(int i=1;i<=n;++i)

cout<<ans[i]<<" ";

return 0;

}

# 图论

## 二分图

### 二分图最大匹配

#### 匈牙利

//O(NM) N左M右

bool vis[N];

int match[N];

bool dfs(int x){

for(int i=head[x];i;i=next1[i]){

int y=ver[i];

if(!vis[y]){

vis[y]=1;

if(!match[y]||dfs(match[y])){ //假如没匹配过或者有增广路

match[y]=x;return true; //找增广路

}

}

}

return false;

}

int main(){

int ans=0;

for(i=1;i<=n;++i){

memset(vis,0,sizeof(vis));

if(dfs(i))ans++;

}

}

#### HK算法

#include<bits/stdc++.h> //根号NM

#define INF 0x3f3f3f3f

using namespace std;

const int MAXN = 100+5;

const int MAXM = 300+5;

int p, n;

int a[MAXN][MAXM];

int dis;

int cx[MAXN], cy[MAXM];

int dx[MAXN], dy[MAXM];

bool vis[MAXM];

bool bfs\_findPath() {

queue<int> q;

memset(dx, -1, sizeof(dx));

memset(dy, -1, sizeof(dy));

// 使用BFS遍历对图的点进行分层，从X中找出一个未匹配点v

// (所有v)组成第一层，接下来的层都是这样形成——每次查找

// 匹配点(增广路性质)，直到在Y中找到未匹配点才停止查找，

// 对X其他未匹配点同样进行查找增广路径(BFS只分层不标记

// 是否匹配点)

// 找出X中的所有未匹配点组成BFS的第一层

dis = INF;

for(int i = 1; i <= p; ++i) {

if(cx[i] == -1) {

q.push(i);

dx[i] = 0;

}

}

while(!q.empty()) {

int u = q.front();

q.pop();

if(dx[u] > dis) break;// 该路径长度大于dis，等待下一次BFS扩充

for(int v = 1; v <= n; ++v) {

if(a[u][v] && dy[v] == -1) {// (u,v)之间有边且v还没有分层

dy[v] = dx[u] + 1;

if(cy[v] == -1) dis = dy[v];// v是未匹配点，停止延伸（查找）,得到本次BFS的最大遍历层次

else {// v是已匹配点，继续延伸

dx[cy[v]] = dy[v] + 1;

q.push(cy[v]);

}

}

}

}

return dis != INF;// 若dis为INF说明Y中没有未匹配点，也就是没有增广路径了

}

bool dfs(int u) {

for(int v = 1; v <= n; ++v) {

if(!vis[v] && a[u][v] && dy[v] == dx[u] + 1) {

vis[v] = 1;

// 层次（也就是增广路径的长度）大于本次查找的dis

// 是bfs中被break的情况，也就是还不确定是否是增广路

// 只有等再次调用bfs再判断(每次只找最小增广路集)

if(cy[v] != -1 && dy[v] == dis) continue;

if(cy[v] == -1 || dfs(cy[v])) {// 是增广路径，更新匹配集

cy[v] = u;

cx[u] = v;

return true;

}

}

}

return false;

}

int HK() {

int ans = 0;

memset(cx, -1, sizeof(cx));

memset(cy, -1, sizeof(cy));

while(bfs\_findPath()) {// 有增广路

memset(vis, 0, sizeof(vis));

for(int i = 1; i <= p; ++i) {

// 用DFS查找增广路径，增广路径一定从未匹配点开始

// 如果查找到一个增广路径，匹配数加一

if(cx[i] == -1 && dfs(i)) ++ans;

}

}

return ans;

}

int main() {

int T;

scanf("%d", &T);

while(T--) {

scanf("%d%d", &p, &n);

memset(a, 0, sizeof(a));

for(int i = 1; i <= p; ++i) {

int nm, x;

scanf("%d", &nm);

for(int j = 0; j != nm; ++j) {

scanf("%d", &x);

a[i][x] = 1;

}

}

printf("%s\n", HK() == p ? "YES" : "NO");

}

return 0;

}

### 二分图最大权匹配

#include<bits/stdc++.h>

using namespace std;

#define LL long long

const int inf = 0x3f3f3f3f;

const LL INF = 0x3f3f3f3f3f3f3f3f;

const int N = 210;

int val[N][N];

int lx[N],ly[N];

int linky[N];

int pre[N];

bool vis[N];

bool visx[N],visy[N];

int slack[N];

int n;

void bfs(int k) {

int px, py = 0,yy = 0;

LL d;

memset(pre, 0, sizeof(pre));

memset(slack, inf, sizeof(slack));

linky[py]=k;

do {

px = linky[py],d = INF, vis[py] = 1;

for(int i = 1; i <= n; i++) {

if(!vis[i]) {

if(slack[i] > lx[px] + ly[i] - val[px][i])

slack[i] = lx[px] + ly[i] - val[px][i], pre[i] = py;

if(slack[i] < d)

d = slack[i], yy = i;

}

}

for(int i = 0; i <= n; i++) {

if(vis[i])

lx[linky[i]] -= d, ly[i] += d;

else

slack[i] -= d;

}

py = yy;

} while(linky[py]);

while(py) {

linky[py] = linky[pre[py]], py=pre[py];

}

}

LL KM() {

memset(lx, 0, sizeof(lx));

memset(ly, 0, sizeof(ly));

memset(linky, 0, sizeof(linky));

for(int i = 1; i <= n; i++){

memset(vis, 0, sizeof(vis));

bfs(i);

}

LL ans = 0;

for (int i = 1;i <= n; i++) {

ans += val[linky[i]][i];

}

return ans;

}

int main() {

int T;

scanf("%d", &T);

for(int \_i = 1; \_i <= T; \_i++) {

scanf("%d", &n);

for(int i = 1; i <= n; i++) {

for(int j = 1; j <= n; j++) {

scanf("%d", &val[i][j]);

val[i][j] = -val[i][j];

}

}

printf("Case #%d: %I64d\n", \_i, -KM());

}

return 0;

}

# 字符串

## Kmp

#include<bits/stdc++.h>

using namespace std;

const int maxn=1e6+5;

char s1[maxn],s2[maxn];

int nxt[maxn],len1,len2,f[maxn];

void getKmp(){

nxt[1]=0;

for(int i=2,j=0;i<=len2;++i){

while(j&&s2[i]!=s2[j+1])j=nxt[j];

if(s2[i]==s2[j+1])j++;

nxt[i]=j;

}

for(int i=1,j=0;i<=len1;++i){

while(j&&(j==len2||s1[i]!=s2[j+1]))j=nxt[j];

if(s1[i]==s2[j+1])j++;

f[i]=j;

}

}

int main(){

scanf("%s",s1+1);

scanf("%s",s2+1);

len1=strlen(s1+1);

len2=strlen(s2+1);

getKmp();

for(int i=1;i<=len1;++i){

if(f[i]==len2)cout<<i-len2+1<<"\n";

}

for(int i=1;i<len2;++i){

cout<<nxt[i]<<" ";

}

cout<<nxt[len2]<<"\n";

return 0;

}

## Exkmp

//求与某个后缀的LCP

#include<bits/stdc++.h>

using namespace std;

using ll=long long;

const int maxn=2e7+10;

char s1[maxn],s2[maxn];

int z[maxn],ext[maxn]; //z[i] s2[i,n]与s2[1,n]的LCP ext[i]是s1[i,n]与s2[1,n]的LCP

void getZ(int len){

z[1]=len;

int now=1;

while(now<len&&s2[now]==s2[now+1])now++;

z[2]=now-1;

int p=2;

for(int i=3;i<=len;++i){

if(z[i-p+1]+i-1<p+z[p]-1)z[i]=z[i-p+1];

else{

now=p+z[p]-i;

if(now<0)now=0;

while(s2[now+1]==s2[i+now]&&i+now<=len)now++;

z[i]=now;

p=i;

}

}

}

void exkmp(int len,int len2){

int now=0;

while(now<len&&now<len2&&s2[now+1]==s1[now+1])now++;

ext[1]=now;

int p=1;

for(int i=2;i<=len;++i){

if(z[i-p+1]+i-1<p+ext[p]-1)ext[i]=z[i-p+1];

else{

now=p+ext[p]-i;

if(now<0)now=0;

while(s2[now+1]==s1[i+now]&&i+now<=len&&now<len2)now++;

ext[i]=now;

p=i;

}

}

}

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

cin>>(s1+1)>>(s2+1);

int len1=strlen(s1+1);

int len2=strlen(s2+1);

getZ(len2);

exkmp(len1,len2);

ll ans0=0,ans1=0;

for(int i=1;i<=len2;++i){

ans0^=(ll)i\*(z[i]+1);

}

for(int i=1;i<=len1;++i){

ans1^=(ll)i\*(ext[i]+1);

}

cout<<ans0<<"\n"<<ans1<<"\n";

return 0;

}

## Trie

#include<bits/stdc++.h>

using namespace std;

const int maxn=5e5+5;

const int maxm=1e4+5;

char s[60];

int n,m;

struct Trie{

int nxt[maxn][26],cnt=1,st[maxn];

void insert(char\*s){

int len=strlen(s+1),p=1;

for(int i=1;i<=len;++i){

int c=s[i]-'a';

if(!nxt[p][c])nxt[p][c]=++cnt;

p=nxt[p][c];

}

}

int find(char\*s){

int len=strlen(s+1),p=1;

for(int i=1;i<=len;++i){

int c=s[i]-'a';

if(!nxt[p][c])return 0;

p=nxt[p][c];

}

if(!st[p]){

st[p]=1;return 1;

}

return 2;

}

}trie;

int main(){

scanf("%d",&n);

for(int i=1;i<=n;++i){

scanf("%s",s+1);

trie.insert(s);

}

scanf("%d",&m);

for(int i=1;i<=m;++i){

scanf("%s",s+1);

int b=trie.find(s);

if(!b)puts("WRONG");

else if(b==1)puts("OK");

else puts("REPEAT");

}

return 0;

}

## 序列自动机

Nxt[i][j]表示i后面第一个为j字母的位置

for(int i=n;i>=1;--i){

for(int j=0;j<26;++j){

nxt[i-1][j]=nxt[i][j];

}

nxt[i-1][s1[i]-'a']=i;

}

## Ac自动机

#include<bits/stdc++.h>

#define pb push\_back

using namespace std;

const int maxn=2e5+5;

const int maxm=2e6+5;

char s[maxm];

vector<int>G[maxn];

int ans[maxn];

struct AC{

int tr[maxn][26],fail[maxn],cnt,ed[maxn],sz[maxn];

void insert(char\*s,int pos){

int now=0,len=strlen(s+1);

for(int i=1;i<=len;++i){

int ch=s[i]-'a';

if(!tr[now][ch])tr[now][ch]=++cnt;

now=tr[now][ch];

}

ed[pos]=now;

}

queue<int>q;

void build(){

for(int i=0;i<26;++i){

if(tr[0][i])q.push(tr[0][i]);

}

while(!q.empty()){

int x=q.front();q.pop();

for(int i=0;i<26;++i){

if(tr[x][i])//依情况看tr[x][i]是否受fail[tr[x][i]]影响

fail[tr[x][i]]=tr[fail[x]][i],q.push(tr[x][i]);

else tr[x][i]=tr[fail[x]][i];

}

}

}

void query(char \*s){

int now=0,len=strlen(s+1);

for(int i=1;i<=len;++i){

int ch=s[i]-'a';

now=tr[now][ch];

sz[now]++;

}

for(int i=1;i<=cnt;++i)G[fail[i]].pb(i);

}

void dfs(int x){

for(auto&v:G[x]){

dfs(v);

sz[x]+=sz[v];

}

}

}ac;

int main(){

int n;

scanf("%d",&n);

for(int i=1;i<=n;++i)scanf("%s",s+1),ac.insert(s,i);

ac.build();

scanf("%s",s+1);

ac.query(s);

ac.dfs(0);

for(int i=1;i<=n;++i)cout<<ac.sz[ac.ed[i]]<<"\n";

return 0;

}

## 后缀数组

倍增O(nlogn)

struct SA{

char s[maxn];

int n,m,a[maxn],sa[maxn],rk[maxn],ht[maxn],tax[maxn],tp[maxn],H[maxn];

//rk[i] [i,n]后缀排名 第一关键字 tp[i]第二关键字中,排名为i的数的位置

//sa[i] 排名为i的后缀的位置

//ht[i] 排名为i和i-1的后缀的LCP

//tax[i](基数排序辅助,多少个排名i)

//H[i]=ht[rk[i]] [i,n]和它前一名的LCP

int cmp(int\*f,int x,int y,int w){

return f[x]==f[y]&&f[x+w]==f[y+w];

}

void rsort(){

for(int i=0;i<=m;++i)tax[i]=0;

for(int i=1;i<=n;++i)tax[rk[tp[i]]]++;

for(int i=1;i<=m;++i)tax[i]+=tax[i-1];

for(int i=n;i>=1;--i)sa[tax[rk[tp[i]]]--]=tp[i];

}

void Suffix(){

for(int i=1;i<=n;++i)a[i]=s[i];

for(int i=1;i<=n;++i)rk[i]=a[i],tp[i]=i;

rsort();

for(int w=1,p=1,i;p<n;w+=w,m=p){

for(p=0,i=n-w+1;i<=n;++i)tp[++p]=i;

for(i=1;i<=n;++i)if(sa[i]>w)tp[++p]=sa[i]-w;

rsort(),swap(rk,tp),rk[sa[1]]=p=1;

for(int i=2;i<=n;++i)

rk[sa[i]]=cmp(tp,sa[i],sa[i-1],w)?p:++p;

}

}

void getht(){

int j,k=0;

for(int i=1;i<=n;ht[rk[i++]]=k)

for(k=k?k-1:k,j=sa[rk[i]-1];a[i+k]==a[j+k];++k);

for(int i=1;i<=n;++i)H[i]=ht[rk[i]];

}

void init(){

scanf("%s",s+1);

n=strlen(s+1);//小心多组数据结束符 此时直接赋值

m=(int)'z';//最大字符ASCII码

Suffix();

getht();

}

}sa;

SAIS O(n)

#include <bits/stdc++.h>

using namespace std;

const int maxn=1e6+5;

struct SA{

int s[(maxn<<1)+5],sa[maxn+5],rk[maxn+5],c[maxn+5],p[maxn+5],tmp[maxn+5],n,m,ht[maxn];

char str[maxn+5]; bool t[(maxn<<1)+5];

#define Ar(x,a) sa[p[s[x]]a]=x

void IS(int\*s,int\*s1,int n1,int n,int m,bool\*t){

memset(sa+1,0,n<<2);memset(c+1,0,m<<2);

for(int i=1;i<=n;++c[s[i++]]);

for(int i=2;i<=m;i++) c[i]+=c[i-1];

memcpy(p+1,c+1,m<<2);

for(int i=n1;i;Ar(s1[i],--),i--);

for(int i=1;i<=m;i++) p[i]=c[i-1]+1;

for(int i=1;i<=n;sa[i]>1&&t[sa[i]-1]?Ar(sa[i]-1,++):0,i++);

memcpy(p+1,c+1,m<<2);

for(int i=n;i;sa[i]>1&&!t[sa[i]-1]?Ar(sa[i]-1,--):0,i--);

}

void SAIS(int s[], bool t[], int tmp[], int n, int m){

int n1=0,m1=rk[1]=0,\*s1=s+n;t[n]=0;

for(int i=n-1;i;t[i]=s[i]^s[i+1]?s[i]>s[i+1]:t[i+1],i--);

for(int i=2;i<=n;rk[i]=t[i-1]&&!t[i]?tmp[++n1]=i,n1:0,i++);

IS(s,tmp,n1,n,m,t);

for(int i=1,x,y=0;i<=n;i++) if(x=rk[sa[i]]){

if(m1<=1||tmp[x+1]-tmp[x]!=tmp[y+1]-tmp[y]) ++m1;

else for(int a=tmp[x],b=tmp[y];a<=tmp[x+1];a++,b++)

if((s[a]<<1|t[a])^(s[b]<<1|t[b])){++m1;break;}

s1[y=x]=m1;

}

if(m1<n1) SAIS(s1,t+n,tmp+n1,n1,m1);

else for(int i=1;i<=n1;sa[s1[i]]=i,i++);

for(int i=1;i<=n1;s1[i]=tmp[sa[i]],i++);

IS(s,s1,n1,n,m,t);

}

void MakeSA(){

--n;

for(int i=1;i<=n;i++) sa[i]=sa[i+1],rk[sa[i]]=i;

}

void getHt(){

for(int i=1,k=0;i<=n;ht[rk[i]]=k,i++,k&&--k)

for(int j=sa[rk[i]-1];s[i+k]==s[j+k];k++);

}

void init(){

scanf("%s",str+1); n=strlen(str+1);

for(int i=1;i<=n;i++) s[i]=str[i];

s[++n]=1; //保证1可以当最小字符集 即便加上偏移

SAIS(s,t,tmp,n,'z'+1); //最大字符集+1

MakeSA();

//getHt();

}

}sa;

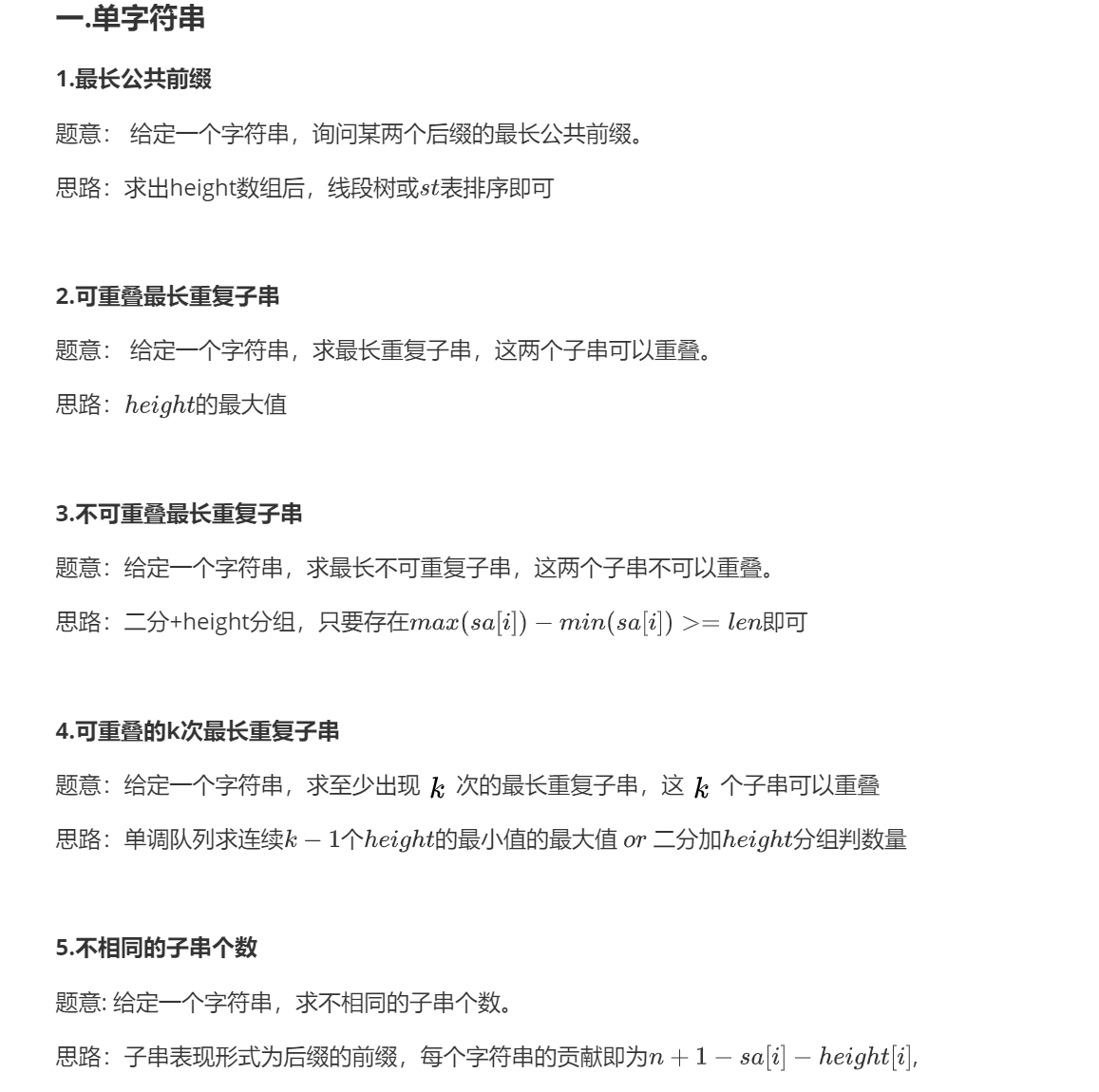
int main(){

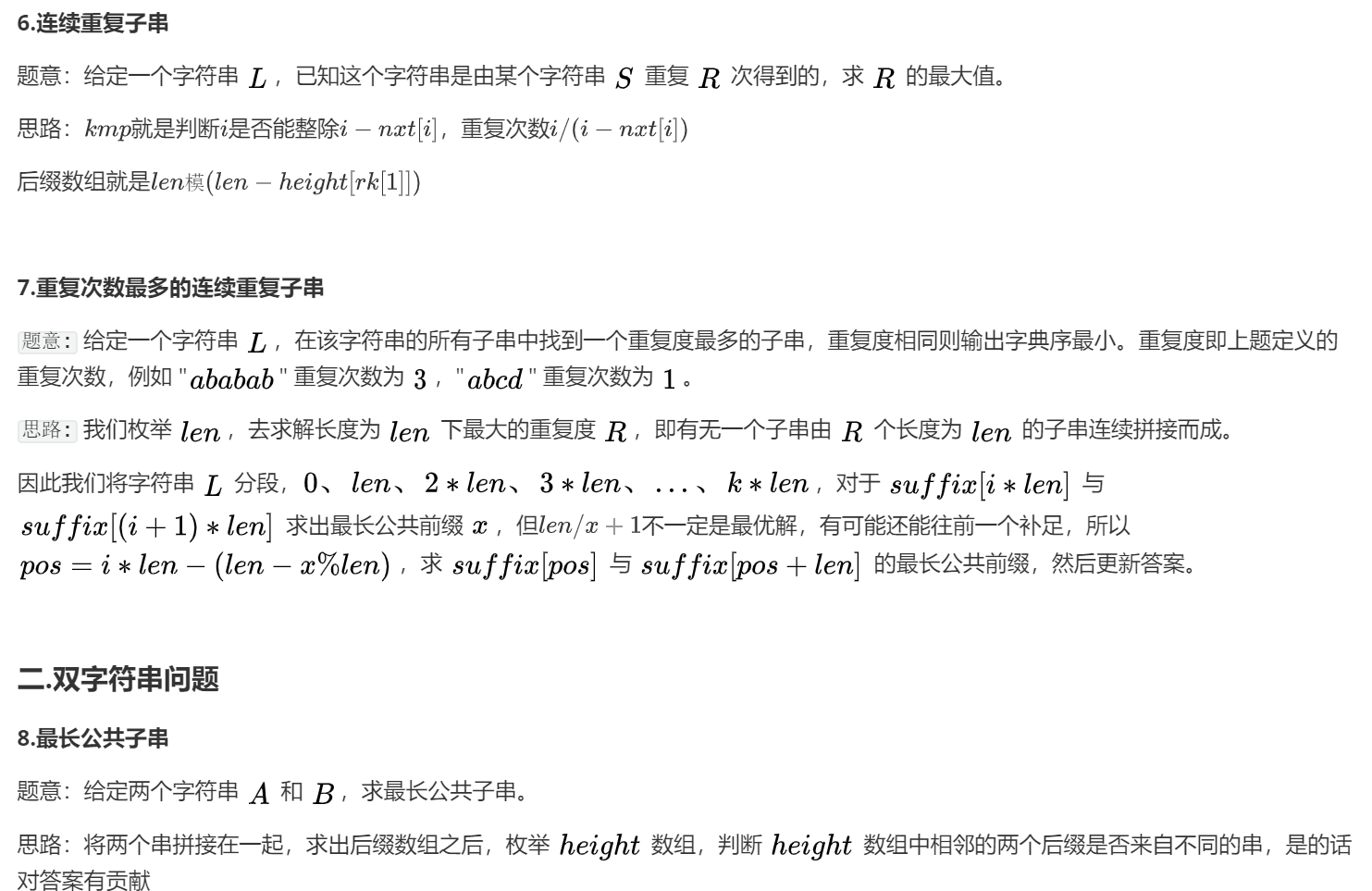
sa.init();

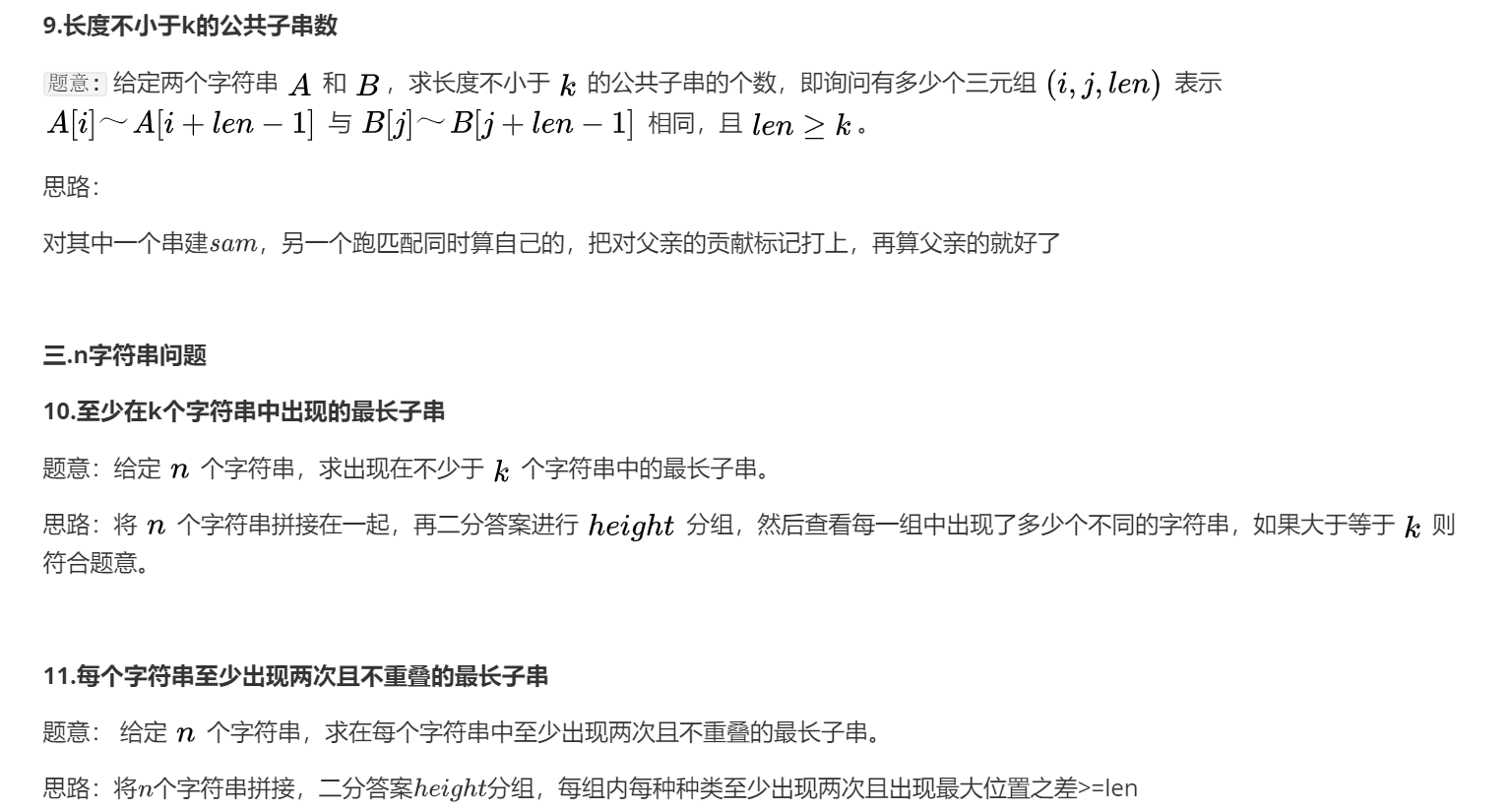
for(int i=1;i<=sa.n;i++) cout<<sa.sa[i]<<" ";

}

**一些套路**







## 后缀自动机

#include<bits/stdc++.h>

using namespace std;

struct SAM{//maxn开2倍字符串

int len[maxn],link[maxn],ch[maxn][26],last,tot,sz[maxn];//len指状态内最长长度

int tax[maxn],rk[maxn];

ll sum[maxn];

SAM(){ //link指向状态内最长字符串的最长的一个在另一个endpos类的后缀

tot=last=1;//sz endpos大小

}

void extend(int c){

int cur=++tot,p=last;last=cur;sz[cur]=1;

len[cur]=len[p]+1;

for(;p&&!ch[p][c];p=link[p])ch[p][c]=cur;

if(!p)link[cur]=1;

else{

int q=ch[p][c];

if(len[p]+1==len[q])link[cur]=q;

else{

int clone=++tot;//==len[p]+1的复制出来

memcpy(ch[clone],ch[q],sizeof(ch[q]));

len[clone]=len[p]+1;

link[clone]=link[q];link[q]=link[cur]=clone;

for(;p&&ch[p][c]==q;p=link[p])ch[p][c]=clone;

}

}

}

int query(char \*s){ //两串匹配 此处求的是最长公共子串

int ans=0,p=1,nowlen=0,L=strlen(s+1);

for(int i=1;i<=L;++i){

int c=s[i]-'a';

if(ch[p][c])nowlen++,p=ch[p][c];

else{

while(p&&!ch[p][c])p=link[p];

if(!p)nowlen=0,p=1;

else{

nowlen=len[p]+1,p=ch[p][c];

}

}

ans=max(ans,nowlen);

}

return ans;

}

void calSZ(int x){

for(int i=1;i<=tot;++i)tax[len[i]]++;

for(int i=1;i<=x;++i)tax[i]+=tax[i-1];

for(int i=1;i<=tot;++i)rk[tax[len[i]]--]=i;

for(int i=tot;i;--i){

int now=rk[i];

sz[link[now]]+=sz[now];

}

sz[1]=0;

}

void calSum(){//sum[i] sam上经过从某字符出发 经过i结点的子串数量

//看题意initsum

for(int i=2;i<=tot;++i)sum[i]=sz[i];

for(int i=tot;i>=1;--i)

for(int j=0;j<26;++j){

int now=rk[i];//记得calSZ先算rk

if(ch[now][j])sum[now]+=sum[ch[now][j]];

}

}

}sam;

## 广义后缀自动机

//离线

struct Trie{

int cnt,tr[N][26],fa[N],ch[N];

Trie(){ cnt=1;}

void insert(char\*s){

int p=1,L=strlen(s+1);

for(int i=1;i<=L;++i){

int c=s[i]-'a';

if(!tr[p][c])tr[p][c]=++cnt,fa[cnt]=p,ch[cnt]=c;

p=tr[p][c];

}

}

}trie;

struct GSAM{

int tot,pos[maxn],link[maxn],len[maxn],ch[maxn][26];

queue<int>q;

GSAM(){ tot=1;}

int insert(int c,int last){

int cur=++tot,p=last;

len[cur]=len[p]+1;

while(p&&!ch[p][c])ch[p][c]=cur,p=link[p];

if(!p)link[cur]=1;

else{

int q=ch[p][c];

if(len[p]+1==len[q])link[cur]=q;

else{

int clone=++tot;

memcpy(ch[clone],ch[q],sizeof(ch[q]));

len[clone]=len[p]+1;

while(p&&ch[p][c]==q)ch[p][c]=clone,p=link[p];

link[clone]=link[q];link[q]=link[cur]=clone;

}

}

return cur;

}

void build(){

for(int i=0;i<26;++i)

if(trie.tr[1][i])q.push(trie.tr[1][i]);

pos[1]=1;

while(!q.empty()){

int x=q.front();q.pop();

pos[x]=insert(trie.ch[x],pos[trie.fa[x]]);

for(int i=0;i<26;++i)

if(trie.tr[x][i])q.push(trie.tr[x][i]);

}

}

}gsam;

//多模式串一个个插入

struct GSAM{

int link[maxn],ch[maxn][26],tot,len[maxn];

GSAM(){ tot=1;}

int extend(int c,int last){

if(ch[last][c]){

int p=last,q=ch[p][c];

if(len[p]+1==len[q])return q;

else{

int clone=++tot;

len[clone]=len[p]+1;

memcpy(ch[clone],ch[q],sizeof(ch[q]));

while(p&&ch[p][c]==q)ch[p][c]=clone,p=link[p];

link[clone]=link[q];link[q]=clone;

return clone;

}

}

int cur=++tot,p=last;

len[cur]=len[p]+1;

while(p&&!ch[p][c])ch[p][c]=cur,p=link[p];

if(!p)link[cur]=1;

else{

int q=ch[p][c];

if(len[q]==len[p]+1)link[cur]=q;

else{

int clone=++tot;

len[clone]=len[p]+1;

memcpy(ch[clone],ch[q],sizeof(ch[q]));

while(p&&ch[p][c]==q)ch[p][c]=clone,p=link[p];

link[clone]=link[q];link[q]=link[cur]=clone;

}

}

return cur;

}

void dfs(int x,int fa,int fap){//在线 与dfs或bfs无关

int xfa=extend(col[x],fap);

for(auto&v:G[x]){

if(v==fa)continue;

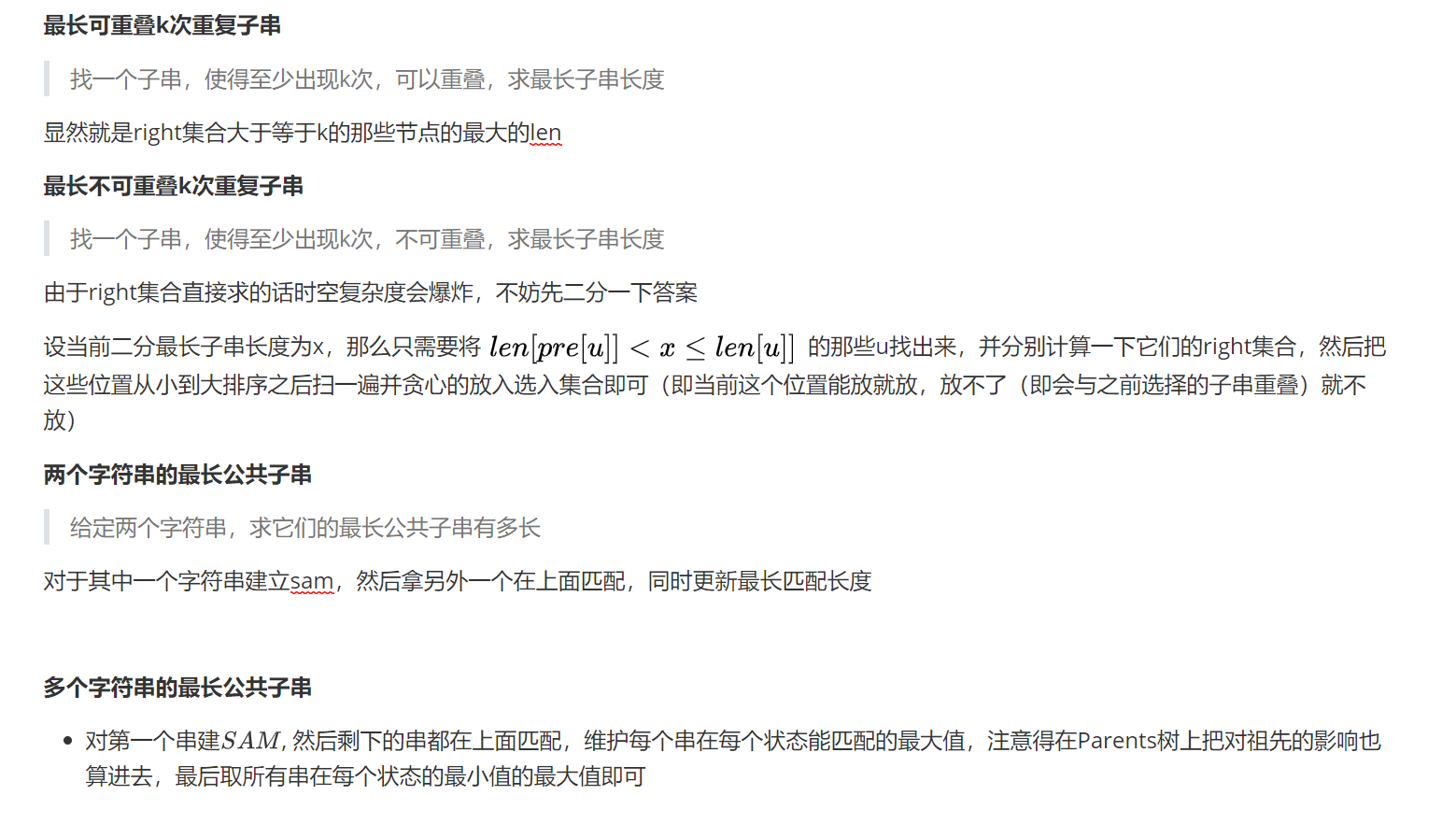
dfs(v,x,xfa);

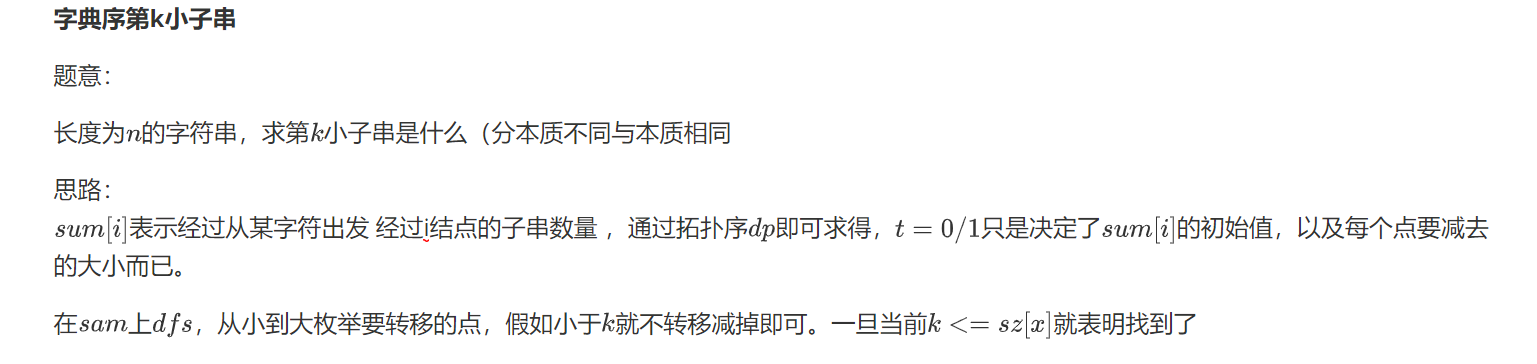
}

}

}gsam;







# 动态规划

## 数位DP

typedef long long ll;

int a[20];

ll dp[20][state];//不同题目状态不同

ll dfs(int pos,/\*state变量\*/,bool lead/\*前导零\*/,bool limit/\*数位上界变量\*/){

if(pos==-1) return 1

if(!limit && !lead && dp[pos][state]!=-1) return dp[pos][state];

int up=limit?a[pos]:9;//根据limit判断枚举的上界up;这个的例子前面用213讲过了

ll ans=0;

for(int i=0;i<=up;i++)

{

if() ...

else if()...

ans+=dfs(pos-1,/\*状态转移\*/,lead && i==0,limit && i==up);//保证合法

}

if(!limit && !lead) dp[pos][state]=ans;

return ans;

}

ll solve(ll x){

memset(dp,-1,sizeof(dp));

int pos=0;

while(x)//把数位都分解出来

{

a[pos++]=x%10;//个人老是喜欢编号为[0,pos),看不惯的就按自己习惯来，反正注意数位边界就行

x/=10;

}

return dfs(pos-1/\*从最高位开始枚举\*/,/\*一系列状态 \*/,true,true);//刚开始最高位都是有限制并且有前导零的，显然比最高位还要高的一位视为0嘛

}

# 线性代数

## 矩阵快速幂

//m阶递推式子的转移矩阵一般是m\*m

//复杂度O(m^3 logn)

struct Matrix{

int mat[maxn][maxn];

Matrix(){memset(mat,0,sizeof(mat));}

Matrix operator\*(Matrix const &b)const{

Matrix res;

memset(res.mat,0,sizeof(res.mat));

for(int i=1;i<maxn;++i)

for(int j=1;j<maxn;++j)

for(int k=1;k<maxn;++k)

res.mat[i][j]=(res.mat[i][j]+this->mat[i][k]\*b.mat[k][j])%mod;

return res;

}

}

Matrix mypow(Matrix a,int b){

Matrix res;

memset(res.mat,0,sizeof(res.mat));

for(int i=1;i<maxn;++i)

res.mat[i][i]=1;

while(b){

if(b&1)res=res\*a;

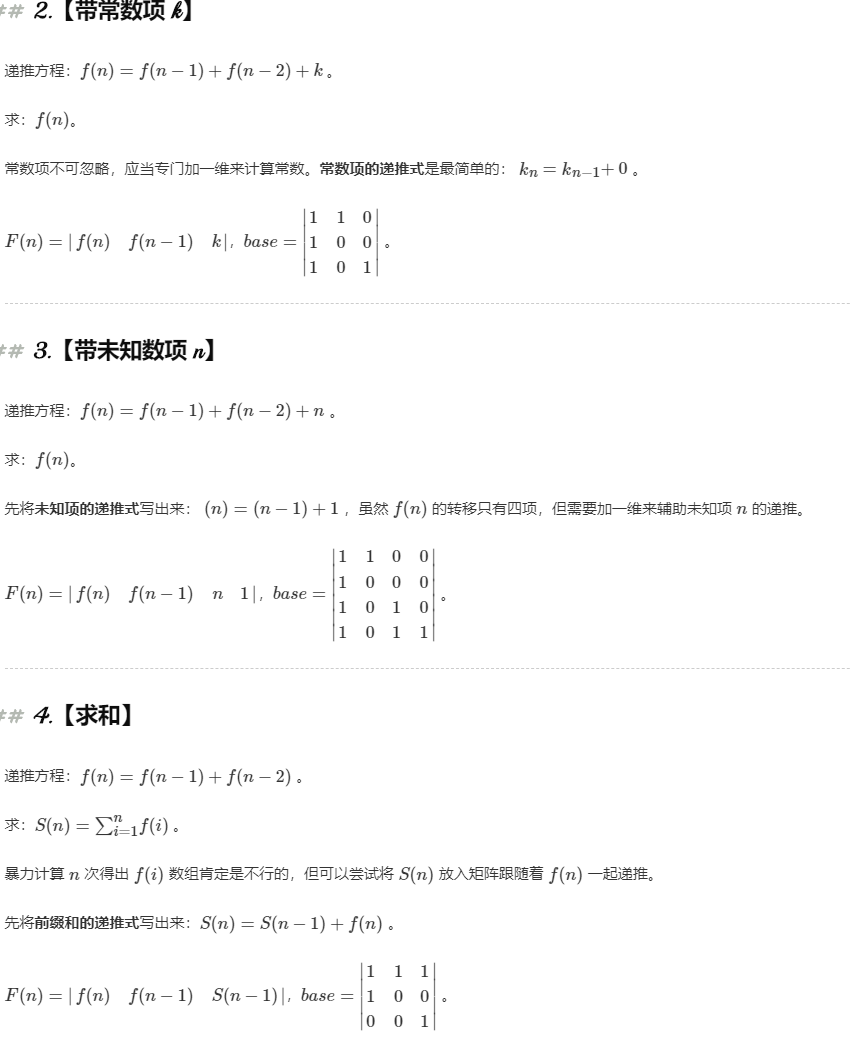
a=a\*a;

b>>=1;

}

return res;

}



## 线性BM递推

#include<bits/stdc++.h>

using namespace std;

#define PB push\_back

typedef long long ll;

using namespace std;

const int mod=1000000007; //按实际改

ll powmod(ll a,ll b){ //快速幂

ll res=1;a%=mod;

assert(b>=0);

while(b){

if(b&1) res=res\*a%mod; a=a\*a%mod; b>>=1;

}

return res;

}

namespace linear\_seq {

const int N=10010; //不需改

ll res[N],base[N],\_c[N],\_md[N];

vector<ll> Md;

void mul(ll \*a,ll \*b,int k) {

for(int i=0;i<k+k;i++) \_c[i]=0;

for(int i=0;i<k;i++) if (a[i])

for(int j=0;j<k;j++) \_c[i+j]=(\_c[i+j]+a[i]\*b[j])%mod;

for (ll i=k+k-1;i>=k;i--) if (\_c[i])

for(int j=0;j<Md.size();j++)

\_c[i-k+Md[j]]=(\_c[i-k+Md[j]]-\_c[i]\*\_md[Md[j]])%mod;

for(int i=0;i<k;i++) a[i]=\_c[i];

}

int solve(ll n,vector<ll> a,vector<ll> b) {

// a 系数 b 初值 b[n+1]=a[0]\*b[n]+...

//求出的是第n+1项

ll ans=0,pnt=0;

ll k=a.size();

assert(a.size()==b.size());

for(int i=0;i<k;i++) \_md[k-1-i]=-a[i];\_md[k]=1;

Md.clear();

for(int i=0;i<k;i++) if (\_md[i]!=0) Md.push\_back(i);

for(int i=0;i<k;i++) res[i]=base[i]=0;

res[0]=1;

while ((1ll<<pnt)<=n) pnt++;

for (ll p=pnt;p>=0;p--) {

mul(res,res,k);

if ((n>>p)&1) {

for (ll i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;

for(int j=0;j<Md.size();j++) res[Md[j]]=(res[Md[j]]-res[k]\*\_md[Md[j]])%mod;

}

}

for(int i=0;i<k;i++) ans=(ans+res[i]\*b[i])%mod;

if (ans<0) ans+=mod;

return ans;

}

vector<ll> BM(vector<ll> s) {

vector<ll> C(1,1),B(1,1);

int L=0,m=1,b=1;

for(int n=0;n<s.size();n++) {

ll d=0;

for(int i=0;i<L+1;i++) d=(d+(ll)C[i]\*s[n-i])%mod;

if (d==0) ++m;

else if (2\*L<=n) {

vector<ll> T=C;

ll c=mod-d\*powmod(b,mod-2)%mod;

while (C.size()<B.size()+m) C.PB(0);

for(int i=0;i<B.size();i++) C[i+m]=(C[i+m]+c\*B[i])%mod;

L=n+1-L; B=T; b=d; m=1;

} else {

ll c=mod-d\*powmod(b,mod-2)%mod;

while (C.size()<B.size()+m) C.PB(0);

for(int i=0;i<B.size();i++) C[i+m]=(C[i+m]+c\*B[i])%mod;

++m;

}

}

return C;

}

int gao(vector<ll> a,ll n) {

vector<ll> c=BM(a);

c.erase(c.begin());

for(int i=0;i<c.size();i++) c[i]=(mod-c[i])%mod;

return solve(n,c,vector<ll>(a.begin(),a.begin()+c.size()));

}

};

//用的时候只用改mod的值和前几项的数值

int main(){

ll n;

while(~scanf ("%lld", &n)){ //求第n项 //一般放入前8项

printf("%lld\n",linear\_seq::gao(vector<ll>{1,5,11,36,95,281,781,2245},n-1));

}

}

## 高斯消元

struct Guass{

int solve(){//n\*n正常消元 枚举列初等行变消阶梯型,再回代得出答案

int r,c;//当前最左列最上行

for(c=1,r=1;c<=n;++c){

int t=r;

for(int i=r+1;i<=n;++i){

if(fabs(a[i][c])>fabs(a[t][c]))

t=i;//找最大值的行

}

if(fabs(a[t][c])<eps)continue;

for(int i=c;i<=n+1;++i)swap(a[t][i],a[r][i]);//丢到最上面的行

for(int i=n+1;i>=c;--i)a[r][i]/=a[r][c];//第一个数变1

for(int i=r+1;i<=n;++i)

if(fabs(a[i][c])>eps)

for(int j=n+1;j>=c;--j)//下面行清零

a[i][j]-=a[r][j]\*a[i][c];

r++;

}

if(r<=n){

for(int i=r;i<=n;++i)

if(fabs(a[i][n+1])>eps)return -1;//非0 无解

return 1;//无穷解

}//回代求解

for(int i=n;i>=1;--i)

for(int j=i+1;j<=n+1;++j)

a[i][n+1]-=a[j][n+1]\*a[i][j];

return 0;//唯一解 看情况return row统计个数

}

int xorSolve(bitset<maxn>a[],int n,int m){//n\*m a11x1^a12x2...a1m-1^xm-1=b1

int r=0,c=0,mx;//注意bitset读进来的顺序 bitset最右方为0

for(;c<m;++c){//行列从0读入

for(mx=r;mx<n;++mx){

if(a[mx][c])break;//每列找按一个不为0的数丢最上面

}

if(mx==n)continue;

if(!a[mx][c])continue;

swap(a[mx],a[r]);

for(int i=r+1;i<n;++i)

if(a[i][c])

a[i]^=a[r];//下面全都消成0

r++;

}

if(r<n){

for(int i=r;i<n;++i)

if(a[i][m])

return -1;//无解

return 1;//无穷解

}

for(int i=n-1;i>=0;--i){//回代

for(int j=i+1;j<m;++j)

a[i][m]=a[i][m]^(a[j][m]\*a[i][j]);

}

return 0;//唯一解a[i][m]

}

//解线性同余方程组

int a[N][N],x[N];//解集

bool freeX[N];//标记是否为自由变元

int gcd(int a, int b) {return !b ? a : gcd(b, a % b);}

int LCM(int a, int b) {return a / gcd(a, b) \* b;}

int solveMod(int equ, int var) { //返回自由变元个数

/\*初始化\*/

for (int i = 0; i <= var; i++) {

x[i] = 0;

freeX[i] = true;

}

/\*转换为阶梯阵\*/

int r,c = 0; //当前处理的行、列

for (r = 0; r < equ && c < var; r++, c++) { //枚举当前处理的行

int mxr = r; //当前列绝对值最大的行

for (int i = r + 1; i < equ; i++) //寻找当前列绝对值最大的行

if (abs(a[i][c]) > abs(a[mxr][c]))

mxr = i;

if (mxr != r) //与第r行交换

for (int j = r; j < var + 1; j++)

swap(a[r][j], a[mxr][j]);

if (a[r][c] == 0) { //c列第r行以下全是0，处理当前行的下一列

r--;

continue;

}

for (int i = r + 1; i < equ; i++) //枚举要删去的行

if (a[i][c] != 0) {

int lcm = LCM(abs(a[i][c]), abs(a[r][c]));

int ta = lcm / abs(a[i][c]);

int tb = lcm / abs(a[r][c]);

if (a[i][c]\*a[r][c] < 0) //异号情况相加

tb = -tb;

for (int j = c; j < var + 1; j++)

a[i][j] = ((a[i][j] \* ta - a[r][j] \* tb) % MOD + MOD) % MOD;

}

}

//无解：化简的增广阵中存在(0,0,...,a)这样的行，且a!=0

for (int i = r; i < equ; i++)

if (a[i][c] != 0)

return -1;

//无穷解: 在var\*(var+1)的增广阵中出现(0,0,...,0)这样的行

int temp = var - r; //自由变元有var-r个

if (r < var) //返回自由变元数

return temp;

//唯一解: 在var\*(var+1)的增广阵中形成严格的上三角阵

for (int i = var - 1; i >= 0; i--) { //计算解集

int temp = a[i][var];

for (int j = i + 1; j < var; j++) {

if (a[i][j] != 0)

temp -= a[i][j] \* x[j];

temp = (temp % MOD + MOD) % MOD; //取模

}

while (temp % a[i][i] != 0)

//外层每次循环都是求a[i][i]，它是每个方程中唯一一个未知的变量

temp += MOD; //a[i][i]必须为整数，加上周期MOD

x[i] = (temp / a[i][i]) % MOD; //取模

}

return 0;

}

}gauss;

## 矩阵求逆与行列式值 O(n^3)

#include <bits/stdc++.h>

#define fp(i, a, b) for(int i = a, i##\_ = (b) + 1; i < i##\_; ++i)

using namespace std;

const int P = 1e9 + 7;

using ll = uint64\_t;

/\*---------------------------------------------------------------------------\*/

//int P;

int POW(ll a, int b = P - 2, ll x = 1) { for (; b >= 1; b >>= 1, a = a \* a % P)if (b & 1)x = x \* a % P; return x; }

namespace Matrix{

using Mat = vector<vector<int>>;

Mat matrix(int n, int m) { return Mat(n, vector<int>(m)); }

Mat I(int n) {

Mat a = matrix(n, n);

fp(i, 0, n - 1) a[i][i] = 1;

return a;

}

Mat operator\*(const Mat &a, const Mat &b) {

// assert(a[0].size() == b.size());

const int n = a.size(), o = a[0].size(), m = b[0].size();

Mat c = matrix(n, m);

fp(k, 0, o - 1) fp(i, 0, n - 1) if (a[i][k]) fp(j, 0, m - 1)

c[i][j] = (c[i][j] + 1ll \* a[i][k] \* b[k][j]) % P;

return c;

}

Mat operator^(Mat a, int b) {

// assert(a.size() == a[0].size());

Mat x = I(a.size());

for (; b; b >>= 1, a = a \* a)

if (b & 1) x = x \* a;

return x;

}

#ifndef prime

int det(Mat a){ // P no limit

// assert(a.size() == a[0].size());

int d = 1, n = a.size();

fp(i, 0, n - 1) {

auto x = a[0].end(), y = x;

fp(j, i, n - 1) if (a[j][i] && (x == y || a[j][i] < x[i])) x = a[j].begin();

if (x == y) return 0;

fp(j, i, n - 1) if (a[j].begin() != x && a[j][i])

for (y = a[j].begin();; swap(x, y)) {

int w = P - y[i] / x[i];

fp(k, i, n - 1) y[k] = (y[k] + (ll) x[k] \* w) % P;

if (!y[i]) break;

}

if (x != a[i].begin()){

d ^= 1;

fp(k, i, n - 1) swap(x[k], a[i][k]);

}

}

if (!d) d = P - 1;

fp(i, 0, n - 1) d = (ll) d \* a[i][i] % P;

return d;

}

#else

int det(Mat a){

// assert(a.size() == a[0].size());

int d = 1, n = a.size(), inv, w;

fp(i, 0, n - 1) {

if (!a[i][i]) fp(j, i, n - 1) if (a[j][i]) { swap(a[i], a[j]), d ^= 1; break; }

if (!a[i][i]) return 0;

inv = P - POW(a[i][i]);

fp(j, i + 1, n - 1){

w = (ll) a[j][i] \* inv % P;

fp(k, i, n - 1) a[j][k] = (a[j][k] + (ll) a[i][k] \* w) % P;

}

}

if (!d) d = P - 1;

fp(i, 0, n - 1) d = (ll) d \* a[i][i] % P;

return d;

}

#endif

int Inv(Mat &a) { // P is a prime number

// assert(a.size() == a[0].size());

const int n = a.size();

vector<int> R(n);

fp(k, 0, n - 1) {

fp(i, k, n - 1) if (a[i][k]) { swap(a[k], a[R[k] = i]); break; }

if (!a[k][k]) return 0;

a[k][k] = POW(a[k][k]);

for (auto &r: a) if (&r != &a[k]) {

r[k] = P - (ll) r[k] \* a[k][k] % P;

fp(j, 0, n - 1) if(j != k) r[j] = (r[j] + (ll) r[k] \* a[k][j]) % P;

}

for (auto &x: a[k]) if (&x != &a[k][k]) x = (ll) x \* a[k][k] % P;

}

for (int k = n - 1; ~k; --k) swap(a[k], a[R[k]]);

return 1;

}

}

/\*---------------------------------------------------------------------------\*/

using namespace Matrix;

int n;

void P4783() {

scanf("%d", &n);

Mat a = matrix(n, n);

fp(i, 0, n - 1) fp(j, 0, n - 1) scanf("%d", &a[i][j]);

if (!Inv(a)) return puts("No Solution"), void();

for (auto &r: a) fp(j, 0, n - 1) printf("%d%c", r[j], " \n"[j == n - 1]);

}

void P7112(){

scanf("%d%d", &n, &P);

Mat a = matrix(n, n);

fp(i, 0, n - 1) fp(j, 0, n - 1) scanf("%d", &a[i][j]), a[i][j] %= P;

printf("%d\n", det(a));

}

int main() {

#ifdef LOCAL

freopen("s.in", "r", stdin);

// freopen("s.out", "w", stdout);

#endif

P4783();

return 0;

}

## 线性基

1.原序列里面的任意一个数都可以由线性基里面的一些数异或得到

2.线性基里面的任意一些数异或起来都不能得到 0

3.线性基里面的数的个数唯一，并且在保持性质一的前提下，数的个数是最少的

**一.线性基构造**

int d[i]中的第i位必定为1且最高位为1(0开始)

d[i]不为0说明高斯消元对应的第i位为主元

void add(ll x){

for(int i=60;i>=0;--i){

if(x&(1ll<<i)){//i>31必须ll

if(d[i])x^=d[i];

else{

d[i]=x;

break;

}

}

}

}

//判断某个数能否插入线性基

bool check(ll x){

for(int i=60;i>=0;--i){

if(x&(1ll<<i)){

if(!d[i])return false;

else x^=d[i];

}

}

return true;

}

二.求异或最大值 贪心

ll get\_max(){

ll ans=0;

for(int i=60;i>=0;--i)

if((ans^d[i])>ans)ans^=d[i];

return ans;

}

**三.求最小值(线性基能异或的最小值)就是最小的d[i]**

**若求原序列异或的最小值,看看有没有元素无法插入，假如有，最小值0，否则仍为min(d[i])**

ll get\_min(){

if(zero)return 0;

for(int i=0;i<=64;++i)

if(d[i])return d[i];

}

**四、查询第k小值 n个元素的线性基最多只能构建出2n-1的元素**

//重构线性基

void rebuild(){

cnt=0;top=0;

for(int i=64;i>=0;--i)

for(int j=i-1;j>=0;--j)

if(d[i]&(1ll<<j))d[i]^=d[j];

for(int i=0;i<=64;++i)

if(d[i])p[cnt++]=d[i];

else zero=1;

}

ll get\_kth(int k){

if(k>=(1ll<<cnt))return -1;

ll ans=0;

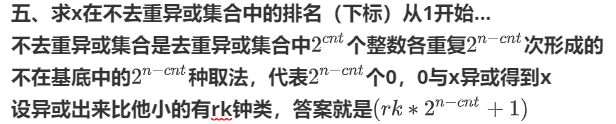
for(int i=64;i>=0;--i)

if(k&(1ll<<i))ans^=p[i];//多组数组初始化d[i],p[i],cnt

return ans;

}

if(cnt<n)k--;//特判0才能求第k小



for(int i=0;i<=30;++i){

if(d[i])p[top++]=i;

}

ll rk=0;

for(int i=0;i<top;++i){//枚举最高的被删去的位 前面的位随便选

if((1<<p[i])&ask)rk+=(1<<i);

}

cout<<(rk\*mypow(2,n-top)+1)%mod;

**六.查询连通图上1到n的最大异或和**

void dfs(int now,ll x){

val[now]=x;v[now]=1;

for(int i=head[now];i;i=next1[i]){

int y=ver[i];

if(!v[y])dfs(y,x^edge[i]);

else insert(x^edge[i]^val[y]);

}

}

dfs(1,0);

cout<<get\_max(val[n]);

**六.线性空间下的线性基** 本质其实只是一个高斯消元 下面是洛谷P3265贪心下的线性基

for(int i=1;i<=n;++i){

for(int j=1;j<=m;++j){

if(fabs(node[i].a[j])>esp){

if(!d[j]){//枚举行，每一行直接找第一个变元

d[j]=i;++cnt;sum+=node[i].cost;

break;

}else{

double t=node[i].a[j]/node[d[j]].a[j];

for(int k=j;k<=m;++k)

node[i].a[k]-=t\*node[d[j]].a[k];

}

}

}

}

**七.线性基求交**

**//封装成线性基结构体的时候适当改写**

int\*merge(int \*a,int \*b){//合并基a和b

int A[64],tmp[64],ans[64];

memset(A,0,sizeof(A));

memset(tmp,0,sizeof(tmp));

memset(ans,0,sizeof(ans));

for(int i=0;i<=63;++i)

A[i]=tmp[i]=a[i]; //tmp不断构建A+(B\ans)

ll cur,d;

for(int i=0;i<=32;++i){//从低到高，使得不存在一个基底可以仅由(tmp\A)表示

if(b[i]){//b中有这个基底

cur=0;d=b[i];

for(int j=i;j>=0;--j){

if((d>>j)&1){

if(tmp[j]){

d^=tmp[j],cur^=A[j];

if(d)continue;

ans[i]=cur;//cur的第i位不为0

}

else tmp[j]=d,A[j]=cur;

break;//如果不能被表示, A的赋值是为了让高位中含有j这位的基底下放到A中j的位置

}

}

}

}

return ans;

}

**八、线段树维护线性基合并**

//线段树维护线性基交 O(qlogn(log^2max))

//本题求区间异或后|k的最大值

#include<bits/stdc++.h>

#define lson p<<1,l,mid

#define rson p<<1|1,mid+1,r

#define ls p<<1

#define rs p<<1|1

using namespace std;

const int maxn=1e4+5;

int T,a[maxn],q,k,L,R,n;

struct LB{

int cnt;

int d[33],p[33];

LB(){}

void init(){

memset(d,0,sizeof(d));

}

void insert(int x){

for(int i=29;i>=0;--i){

if(x&(1<<i)){

if(d[i])x^=d[i];

else{

d[i]=x;break;

}

}

}

}

int getMax(){

int ans=0;

for(int i=29;i>=0;--i){

if((ans^d[i])>ans)ans^=d[i];

}

return ans;

}

LB merge(LB m){

LB ret;

for(int i=0;i<=29;i++){ret.d[i]=d[i];}

for(int i=0;i<=29;i++){

for(int j=i;j>=0;j--){

if(m.d[i]&(1<<j)){

if(ret.d[j]) m.d[i]^=ret.d[j];

else {ret.d[j]=m.d[i];break;}

}

}

}

return ret;

}

}tr[maxn<<2],A,tmp;

void pushUp(int p){

tr[p]=tr[ls].merge(tr[rs]);

}

void build(int p,int l,int r){

if(l==r){

tr[p].init();

tr[p].insert(a[l]);

return;

}

int mid=l+r>>1;

build(lson);

build(rson);

pushUp(p);

}

LB query(int p,int l,int r,int L,int R){

if(L<=l&&r<=R)return tr[p];

int mid=l+r>>1;

if(R<=mid)return query(lson,L,R);

else if(L>mid)return query(rson,L,R);

else return query(lson,L,R).merge(query(rson,L,R));

}

int main(){

cin>>T;

while(T--){

cin>>n>>q>>k;

k=~k;

for(int i=1;i<=n;++i)cin>>a[i],a[i]&=k;

k=~k;

build(1,1,n);

for(int i=1;i<=q;++i){

cin>>L>>R;

cout<<(query(1,1,n,L,R).getMax()|k)<<"\n";

}

}

return 0;

}

# 数论

## 质数

### n以内的质数个数 lehmer\_pi O(n^(2/3))

//一组1e11 150ms

//一组1e12 830ms

#include<bits/stdc++.h>

using namespace std;

#define LL long long

const int N = 5e6 + 2;

bool np[N];

int prime[N], pi[N];

int getprime() {

int cnt = 0;

np[0] = np[1] = true;

pi[0] = pi[1] = 0;

for(int i = 2; i < N; ++i)

{

if(!np[i]) prime[++cnt] = i;

pi[i] = cnt;

for(int j = 1; j <= cnt && i \* prime[j] < N; ++j)

{

np[i \* prime[j]] = true;

if(i % prime[j] == 0) break;

}

}

return cnt;

}

const int M = 7;

const int PM = 2 \* 3 \* 5 \* 7 \* 11 \* 13 \* 17;

int phi[PM + 1][M + 1], sz[M + 1];

void init() {

getprime();

sz[0] = 1;

for(int i = 0; i <= PM; ++i) phi[i][0] = i;

for(int i = 1; i <= M; ++i)

{

sz[i] = prime[i] \* sz[i - 1];

for(int j = 1; j <= PM; ++j) phi[j][i] = phi[j][i - 1] - phi[j / prime[i]][i - 1];

}

}

int sqrt2(LL x) {

LL r = (LL)sqrt(x - 0.1);

while(r \* r <= x) ++r;

return int(r - 1);

}

int sqrt3(LL x) {

LL r = (LL)cbrt(x - 0.1);

while(r \* r \* r <= x) ++r;

return int(r - 1);

}

LL getphi(LL x, int s) {

if(s == 0) return x;

if(s <= M) return phi[x % sz[s]][s] + (x / sz[s]) \* phi[sz[s]][s];

if(x <= prime[s]\*prime[s]) return pi[x] - s + 1;

if(x <= prime[s]\*prime[s]\*prime[s] && x < N)

{

int s2x = pi[sqrt2(x)];

LL ans = pi[x] - (s2x + s - 2) \* (s2x - s + 1) / 2;

for(int i = s + 1; i <= s2x; ++i) ans += pi[x / prime[i]];

return ans;

}

return getphi(x, s - 1) - getphi(x / prime[s], s - 1);

}

LL getpi(LL x) {

if(x < N) return pi[x];

LL ans = getphi(x, pi[sqrt3(x)]) + pi[sqrt3(x)] - 1;

for(int i = pi[sqrt3(x)] + 1, ed = pi[sqrt2(x)]; i <= ed; ++i) ans -= getpi(x / prime[i]) - i + 1;

return ans;

}

LL lehmer\_pi(LL x) {

if(x < N) return pi[x];

int a = (int)lehmer\_pi(sqrt2(sqrt2(x)));

int b = (int)lehmer\_pi(sqrt2(x));

int c = (int)lehmer\_pi(sqrt3(x));

LL sum = getphi(x, a) +(LL)(b + a - 2) \* (b - a + 1) / 2;

for (int i = a + 1; i <= b; i++)

{

LL w = x / prime[i];

sum -= lehmer\_pi(w);

if (i > c) continue;

LL lim = lehmer\_pi(sqrt2(w));

for (int j = i; j <= lim; j++) sum -= lehmer\_pi(w / prime[j]) - (j - 1);

}

return sum;

}

int main() {

init();

LL n;

while(~scanf("%lld",&n)) {

printf("%lld\n",lehmer\_pi(n));

}

return 0;

}

### 反素数

### Miller Robin and pollard\_rho

#include<bits/stdc++.h>

using namespace std;

#define fi first

#define se second

typedef long long ll;

typedef pair<int,int> pii;

typedef pair<ll,ll> pll;

namespace pollard\_rho {

const int C=2307;

const int S=10;

typedef pair<ll,int> pli;

mt19937 rd(time(0));

vector<ll>ve;

ll gcd(ll a,ll b){return b?gcd(b,a%b):a;}

ll mul(ll a,ll b,ll mod){return (\_\_int128)a\*b%mod;}

//miller\_rabin klogn

//考虑O(1)快速乘

ll power(ll a,ll b,ll mod){

ll res=1;a%=mod;

while(b){

if(b&1)res=mul(res,a,mod);

a=mul(a,a,mod);

b>>=1;

}

return res;

}

bool check(ll a,ll n){

ll m=n-1,x,y;

int j=0;

while(!(m&1))m>>=1,j++;

x=power(a,m,n);

for(int i=1;i<=j;x=y,i++){

y=mul(x,x,n);

if(y==1&&x!=1&&x!=n-1)return 1;

}

return y!=1;

}

bool miller\_rabin(ll n){O(klogn)

ll a;

if(n==1)return 0;

if(n==2)return 1;

if(!(n&1))return 0;

for(int i=0;i<S;i++)if(check(rd()%(n-1)+1,n))return 0;

return 1;

}

ll pollard\_rho(ll n,int c){

ll i=1,k=2,x=rd()%n,y=x,d;

while(1){

i++;x=(mul(x,x,n)+c)%n,d=gcd(y-x,n);

if(d>1&&d<n)return d;

if(y==x)return n;

if(i==k)y=x,k<<=1;

}

}

void findfac(ll n,int c){

if(n==1)return ;

if(miller\_rabin(n)){

ve.push\_back(n);

return ;

}

ll m=n;

while(m==n)m=pollard\_rho(n,c--);

findfac(m,c);

findfac(n/m,c);

}

vector<pli> solve(ll n){//返回大数分解的质因子及其次数

vector<pli>res;//n^(1/4) 实际上是最小的p的sqrt(p)

ve.clear();

findfac(n,C);

sort(ve.begin(),ve.end());

for(auto x:ve){

if(res.empty()||res.back().fi!=x)res.push\_back({x,1});

else res.back().se++;

}

return res;

}

}

using namespace pollard\_rho;

int main(){

return 0;

}

## 筛法

### 质数线性筛法及常见函数筛

bool is\_prime(int n) {

if(n<2)return false; //0,1非质数非合数

for(int i=2;i\*i<=n;++i)

if(n%i==0)return false;

return true;

}

**//埃筛**

const int maxn=1e6;

int vis[maxn];//是否被筛

void is\_prime(){

memset(vis,0,sizeof(v));//先都标记为质数

vis[1]=vis[0]=1;//如果有需要特判1、0不是质数

for(int i=2;i<=maxn;++i){

if(!vis[i]){

prime[++cnt]=i;

for(int j=2\*i;j<=maxn;j+=i)

vis[j]=1;//合数标记

}

}

}

**//O(N)线性筛**

const int maxn=1e6;

bool v[MAX\_N];int prime[MAX\_N],cnt=0;//prime存储质数，v为0表示质数

void init(){

for(int i=2;i<=maxn;++i){

if(!v[i])prime[++cnt]=i;

for(int j=1;j<=cnt&&prime[j]\*i<=maxn;+ +j){

v[prime[j]\*i]=1;// prime里面纪录的素数，升序来当做要消去合数的最小素因子。

if(i%prime[j]==0)break;

}

}

}

//数组 v存储最小质因子

void init(){

memset(v,0,sizeof(v));

int cnt=0;

for(int i=2;i<=n;++i){

if(!v[i])v[i]=i,prime[++cnt]=i;

for(int j=1;j<=cnt&&prime[j]\*i<=n;++j){

if(prime[j]>v[i])break;

v[i\*prime[j]]=prime[j];

}

}

}

**//线性筛欧拉函数**

void get\_phi(){

v[1]=v[0]=1;phi[1]=1;

for(int i=2;i<maxn;++i){

if(!v[i])prime[++cnt]=i,phi[i]=i-1;

for(int j=1;j<=cnt&&prime[j]\*i<maxn;++j){

v[prime[j]\*i]=1;

if(i%prime[j]==0){

phi[prime[j]\*i]=phi[i]\*prime[j];

break;

}else

phi[i\*prime[j]]=phi[prime[j]]\*phi[i];

}

}

}

**//线性筛莫比乌斯**

void get\_mu(){

mu[1]=1;

for(int i=2;i<maxn;++i){

if(!v[i]){

prime[++cnt]=i;

mu[i]=-1;

}

for(int j=1;j<=cnt&&prime[j]\*i<maxn;++j){

v[prime[j]\*i]=1;

if(i%prime[j]==0){

mu[i\*prime[j]]=0;break;

}

else mu[i\*prime[j]]=-mu[i];

}

}

}

int mu(int n)//求单个数的莫比乌斯{

if(n==1)return 1;

int sq=0;

for(int i=2;i\*i<maxn;++i){

if(n%i==0){

sq++;

int cnt=0;

while(n%i==0)n/=i,cnt++;

if(cnt>=2)return 0;

}

}

if(n>1)sq++;

return (sq&1)?-1:1;

}

**//线性筛约数个数**

**//a[i]表示i的最小质因子的指数**

int N, prime[MAXN], vis[MAXN], D[MAXN], a[MAXN], tot;

void GetD(int N) {

vis[1] = D[1] = a[1] = 1;

for(int i = 2; i <= N; i++) {

if(!vis[i]) prime[++cnt] = i, D[i] = 2, a[i] = 1;

for(int j = 1; j <= cnt && i \* prime[j] <= N; j++) {

vis[i \* prime[j]] = 1;

if(i % prime[j]==0) {

D[i \* prime[j]] = D[i] / (a[i] + 1) \* (a[i] + 2);

a[i \* prime[j]] = a[i] + 1;

break;

}

D[i \* prime[j]] = D[i] \* D[prime[j]];

a[i \* prime[j]] = 1;

}

}

}

**//线性筛约数和**

int N, prime[MAXN], vis[MAXN], SD[MAXN], sum[MAXN], low[MAXN], tot;

void GetSumD(int N) {//low[i]表示i最小质因子p1^a1 sum[i]=最小质因子p的∑(i=0到ai)p^i求和

vis[1] = SD[1] = low[1] = sum[1] = 1;

for(int i = 2; i <= N; i++) {

if(!vis[i]) prime[++tot] = i, sum[i] = SD[i] = i + 1, low[i] = i;

for(int j = 1; j <= tot && i \* prime[j] <= N; j++) {

vis[i \* prime[j]] = 1;

if(i % prime[j]==0) {

low[i \* prime[j]] = low[i] \* prime[j];

sum[i \* prime[j]] = sum[i] + low[i \* prime[j]];

SD[i \* prime[j]] = SD[i] / sum[i] \* sum[i \* prime[j]];

break;

}

low[i \* prime[j]] = prime[j];

sum[i \* prime[j]] = prime[j] + 1;

//这里low和sum不是积性函数

SD[i \* prime[j]] = SD[i] \* SD[prime[j]];

}

}

}

**//利用最小质因子线性筛筛一般函数**

vis[1] = low[1] = 1; H[1] = 初始化

for(int i = 2; i <= N; i++) {

if(!vis[i]) prime[++tot] = i, mu[i] = -1, H[i] = 质数的情况, low[i] = i;

for(int j = 1; j <= tot && i \* prime[j] <= N; j++) {

vis[i \* prime[j]] = 1;

if(!(i % prime[j])) {

low[i \* prime[j]] = (low[i] \* prime[j]);

if(low[i] == i) H[i \* prime[j]] = 特殊判断;//由f(p^k)到f(p^(k+1))

else H[i \* prime[j]] = H[i / low[i]] \* H[prime[j] \* low[i]];

break;

}

H[i \* prime[j]] = H[i] \* H[prime[j]];

low[i \* prime[j]] = prime[j];

}

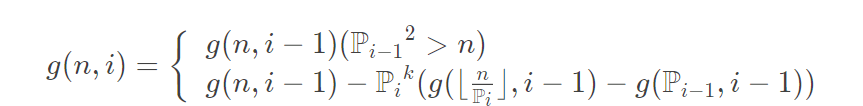
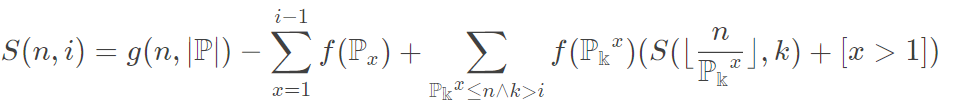
}

### Min25筛

前置筛：g(n)表示p的k次方的单项式前缀和，通过多个单项式组成f(p)

g(n,j)：所有质数+最小质因子>pj 的所有数，做了j次埃筛

S(n,j)表示最小质因子>pk的f的前缀和

**步骤：**

1. **筛根号n内质数，以及对应质数次方前缀和**
2. **记录和初始化g分块里面的第k个数，用两个id数组记录x，n/x相应下标，初始化的值是对应次方前缀和-1（枚举从2开始，不算1）**
3. **滚动dp处理出g(n)，多个g’组成**
4. **枚举pk及其次数递归求S(n,0)**
5. **递推pk^3的时候，一定开LL**
6. **Id1,id2数组为根号n以内，但g1,g2和w数组是由分块得来，很可能大于根号n**

下面板子处理的是

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int maxn=1e5+5;//根号n

const int maxm=2e6+5;

const int mod=1e9+7;

int cnt,id1[maxn],id2[maxn],sqr;

ll sum1[maxn],sum2[maxn],n,w[maxm],g1[maxm],g2[maxm],inv6;//sum1表示p的前缀和,sum2表示p^2的前缀和

ll prime[maxn],inv2,m=0;//注意g1 g2 w等根号分块的数组大小很可能大于根号N！！

bool v[maxn];

void init(){

v[1]=1;

for(int i=2;i<maxn;++i){

if(!v[i])prime[++cnt]=i;

for(int j=1;j<=cnt&&prime[j]\*i<maxn;++j){

v[i\*prime[j]]=1;

if(i%prime[j]==0)break;

}

}

for(int i=1;i<=cnt;++i){//预处理f(p)中几个单项式的和

sum1[i]=(sum1[i-1]+prime[i])%mod;

sum2[i]=(sum2[i-1]+prime[i]\*prime[i]%mod)%mod;

}

}

ll mypow(ll a,ll b){

ll ans=1;

while(b){

if(b&1)ans=ans\*a%mod;

a=a\*a%mod;

b>>=1;

}

return ans;

}

ll S(ll x,int j){//递归求S 从2开始 不算S(1)

if(prime[j]>=x)return 0;

int t=((x<=sqr)?id1[x]:id2[n/x]);

ll ans=((g2[t]-g1[t])+mod-(sum2[j]-sum1[j])+mod)%mod;//质数部分

for(int i=j+1;i<=cnt&&prime[i]\*prime[i]<=x;++i){

ll sp=prime[i];

for(int e=1;sp<=x;sp\*=prime[i],++e){

int tmp=sp%mod;

ans=(ans+1ll\*tmp\*(tmp-1)%mod\*(S(x/sp,i)+(e>1)))%mod;

}

}

return ans;

}

int main(){

init();

inv2=mypow(2,mod-2);

inv6=mypow(6,mod-2);

scanf("%lld",&n);

sqr=sqrt(n+0.5);

for(ll i=1,j;i<=n;i=j+1){//初始化g以及记录分块下要处理的数及其对应下标

j=(n/(n/i));

w[++m]=n/i;

g1[m]=w[m]%mod;//g1 g2分别为对应要处理的p^k的前缀和 g1先当中间变量 w[m]为传进去处理的前缀和

g2[m]=(g1[m]\*(g1[m]+1)%mod\*(2\*g1[m]+1)%mod\*inv6)%mod;

g2[m]--;//前缀和得减去1

g1[m]=(g1[m]\*(g1[m]+1)%mod\*inv2)%mod;

g1[m]--;

if(w[m]<=sqr)id1[w[m]]=m;

else id2[n/w[m]]=m;

}

//dp处理出g(n)，质数处f(p)前缀和

for(int i=1;i<=cnt;++i){

ll pr=prime[i]\*prime[i];

for(int j=1;j<=m&&pr<=w[j];++j){

ll x=w[j]/prime[i];

x=((x<=sqr)?id1[x]:id2[n/x]);

g1[j]=(g1[j]-prime[i]\*(g1[x]-sum1[i-1])%mod+mod)%mod;//如果要卡常必要可以删去此处所有模数，因子数\*根号n\*K会不会炸 但用g的时候必须保证加回正数,最好只在有一个g的时候用

g2[j]=(g2[j]-prime[i]\*prime[i]%mod\*(g2[x]-sum2[i-1]+mod)%mod+mod)%mod;

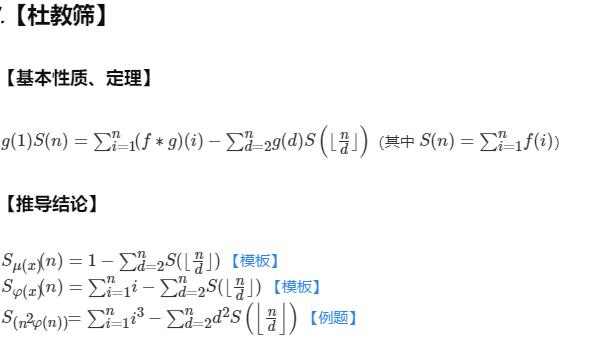
}

}

printf("%d\n",(S(n,0)+1)%mod);

}

### 杜教筛



//杜教筛 O(n^(2/3))

//g(1)S(n)=∑(f\*g)(i)(i属于1到n)-∑g(d)S(n/d) (d属于2到n)

#include<iostream>

#include<algorithm>

#include<cstdio>

#include<cstring>

#include<cmath>

#include<unordered\_map>

using namespace std;

typedef long long ll;

const int maxn=2e6+5;

int prime[maxn],cnt=0,N,t;

bool visphi[1300],v[maxn],vismu[1300];

ll phi[maxn],mu[maxn];

ll phiS2[1300],muS2[maxn];

unordered\_map<int,ll>ansphi;

unordered\_map<int,int>ansmu;

void init(){

mu[1]=phi[1]=1;

for(int i=2;i<=maxn-5;++i){

if(!v[i])prime[++cnt]=i,phi[i]=i-1,mu[i]=-1;

for(int j=1;j<=cnt&&prime[j]\*i<=maxn-5;++j){

v[i\*prime[j]]=1;

if(i%prime[j]==0){

phi[i\*prime[j]]=phi[i]\*prime[j];

break;

}

mu[i\*prime[j]]=-mu[i];

phi[i\*prime[j]]=phi[i]\*(prime[j]-1);

}

}

for(int i=2;i<=maxn-5;++i)

phi[i]+=phi[i-1],mu[i]+=mu[i-1];

}

ll phiS(int n){

if(n<=maxn-5)return phi[n];

if(ansphi[n])return ansphi[n];

/\*int x=N/n;// 卡常用 原理易得 注意清空

if(visphi[x])return phiS2[x];

visphi[x]=1;

ll&ans=phiS2[x];\*/

ll ans=1ll\*(n+1)\*n/2;

for(int i=2,j;i<=n;i=j+1){

j=n/(n/i);

ans-=1ll\*(j-i+1)\*phiS(n/i);

}

return ansphi[n]=ans;

}

ll muS(int n){

if(n<=maxn-5)return mu[n];

if(ansmu[n])return ansmu[n];

/\*int x=N/n;

if(vismu[x])return muS2[x];

vismu[x]=1;

ll&ans=muS2[x];\*/

ll ans=1ll;

for(int i=2,j;i<=n;i=j+1){

j=n/(n/i);

ans-=1ll\*(j-i+1)\*muS(n/i);

}

return ansmu[n]=ans;

}

int main(){

init();

scanf("%d",&t);

while(t--){

scanf("%d",&N);

/\*for(int i=1;i<=1295;++i)vismu[i]=visphi[i]=0;\*/

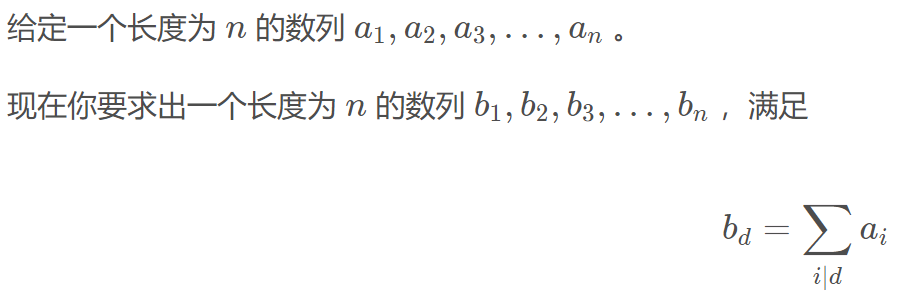
printf("%lld %lld\n",phiS(N),muS(N));

}

return 0;

}

## Dirichlet 前缀和

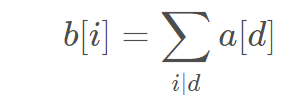


for(int i = 1; i <= cnt && primes[i] <= n; ++ i)

for(int j = 1; j \* primes[i] <= n; ++ j)

a[j \* primes[i]] += a[j];

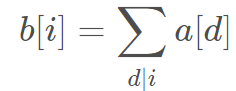
//Dirichlet 后缀和



for(int i = 1; i <= cnt && primes[i] <= n; ++ i)

for(int j = n / primes[i]; j ; -- j)

a[j] += a[j \* primes[i]];

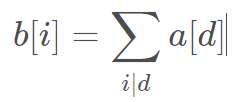


倒推 Dirichlet 前缀和

for(int i = cnt; i ; -- i)

for(int j = n / primes[i]; j ; -- j)

a[j \* primes[i]] -= a[j];



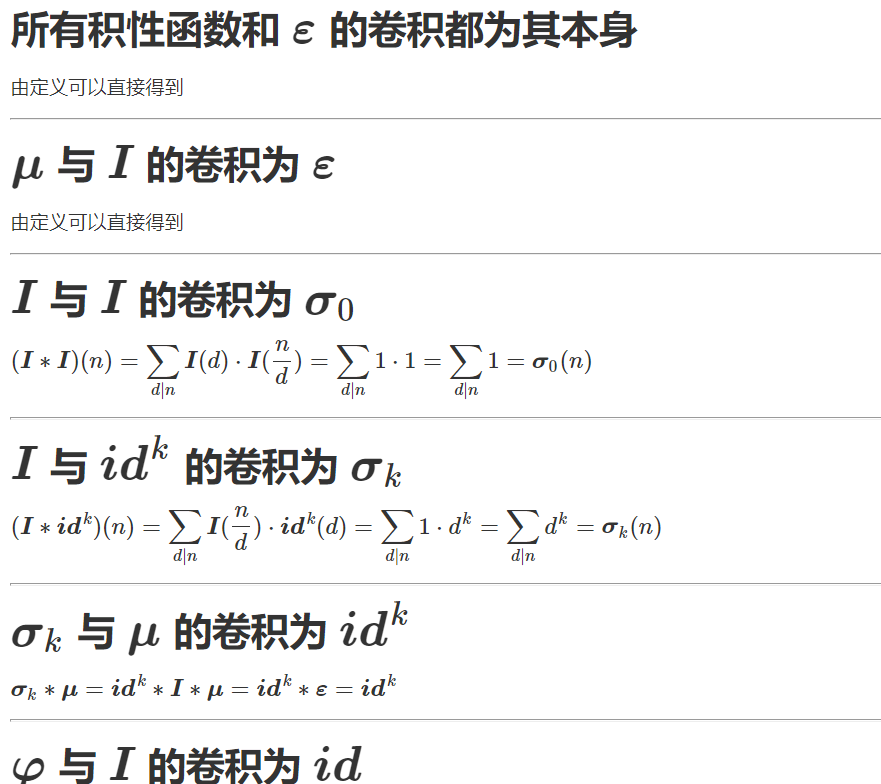
倒推 Dirichlet 后缀和

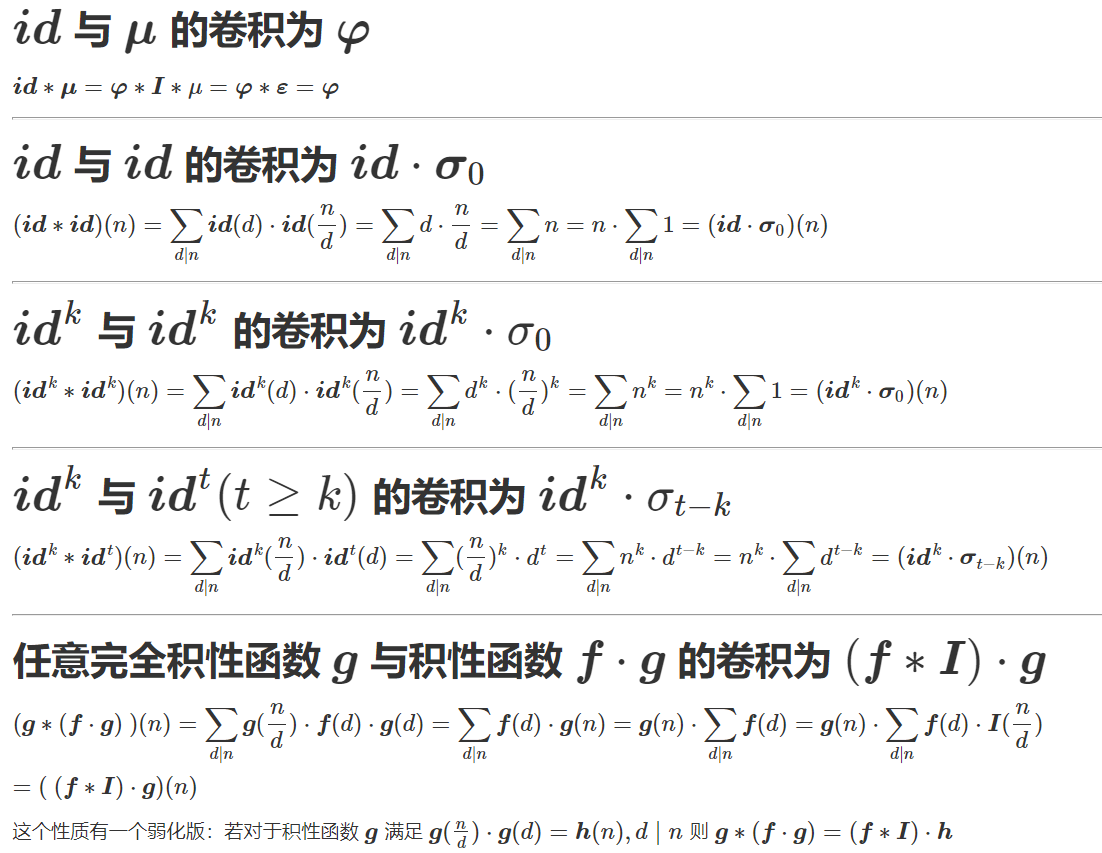
for(int i = cnt; i ; -- i)

for(int j = 1; j \* primes[i] <= n; ++ j)

a[j] -= a[j \* primes[i]];

## 常用Dirichlet卷积





## 约数

### 约数处理

试除法 

int fac[2000],m=0;

for(int i=1;i\*i<=n;++i){

if(n%i==0){

fac[++m]=i;

if(i!=n/i)fac[++m]n/i;

}

}

1~N每个数正约数集合-倍数法

vector<int>fac[500010];

for(int i=1;i<=n;++i)

for(int j=1;j<=n/i;++j)

fac[i\*j].pb(i);

### GCD与LCM

int gcd(int a,int b){ return b?gcd(b,a%b):a;}

int lcm(int a,int b){ return a/gcd(a,b)\*b;}

## 同余

### 快速幂与乘求模

//快速幂与乘求模

ll mypow(ll a,ll b,ll mod){

ll ans=1;

while(b){

if(b&1)ans=ans\*a%mod;

a=a\*a%mod;

b>>=1;

}

return ans;

}

//64位整数乘法 a\*b%p 1e18 O(logb)

ll mul(ll a,ll b,ll mod){

ll ans=0;

while(b){

if(b&1)ans=(ans+a)%mod;

a=(a+a)%mod;

b>>=1;

}

return ans;

}

//O(1)快速乘

ll mul(ll a,ll b,ll p){

if(p<=1000000000)return a\*b%p;

else if(p<=1000000000000ll)return (((a\*(b>>20)%p)<<20)+(a\*(b&((1<<20)-1))))p;//小于1e12加法分配率

else{

ll d=(ll)floor(a\*(ld)b/p+0.5);

ll ret=(a\*b-d\*p)%p;

if(ret<0)ret+=p;

return ret;

}

}

### 逆元相关

//费马小定理 O(logmod)

//a,p互质下

ll mypow(ll a,ll b,ll mod){

ll ans=1;

while(b){

if(b&1)ans=ans\*a%mod;

a=a\*a%mod;

b>>=1;

}

return ans;

}

ll getInv(int a,ll mod){ return mypow(a,mod-2,mod);}

//线性递推求逆元

//调用前预处理 调用时候对除数取模 适用Lucas

ll inv[maxn];

void init(){

inv[1]=1;

for(int i=2;i<maxn;++i)

inv[i]=(mod-mod/i)\*inv[mod%i]%mod;

}

//线性递推求阶乘逆元

void init(){

fac[1]=fac[0]=1;

for(int i=2;i<maxn;++i)fac[i]=1ll\*fac[i-1]\*i%mod;

facinv[maxn]=mypow(fac[maxn],mod-2);

for(int i=maxn-1;i>=0;--i)

facinv[i]=1ll\*facinv[i+1]\*(i+1)%mod;

}

ll inv(ll i){ //i和mod互质 i<mod

if(i==1)return 1;

return (mod-mod/i)\*inv(mod%i)%mod;

}

### Exgcd

//ax+by=c (gcd(a,b)|c)才有解

int exgcd(int a,int b,int&x,int&y){ //gcd(a,b)

if(!b){ x=1,y=0;return a;}

int d=exgcd(b,a%b,x,y);

int z=x;x=y;y=z-y\*(a/b);

return d;

}

//xy乘(c/d)得一组特解

//最小非负整数解

//b=b/gcd(a,b) a=a/gcd(a,b)

//x=(x%b+b)%b y=(y%a+a)%a

### 类欧几里德

int f(int a,int b,int c,int n){//只求f

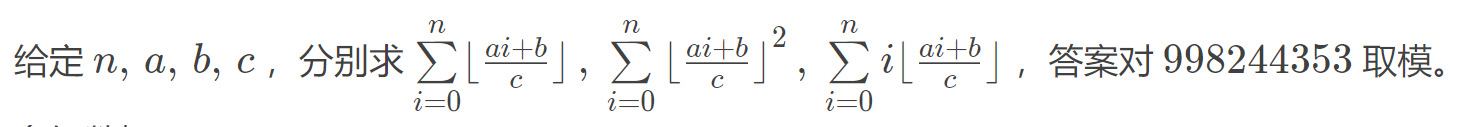
if(a==0)return((b/c)\*(n+1));

if(a>=c||b>=c)return f(a%c,b%c,c,n)+(a/c)\*n\*(n+1)/2+(b/c)\*(n+1);

int m=(a\*n+b)/c;

return n\*m-f(c,c-b-1,a,m-1);

}



typedef long long ll;

const ll mod = 998244353, inv2 = 499122177, inv6 = 166374059;

struct query {

ll f, g, h;

};

il ll S1(ll n) {return n \* (n + 1) / 2 % mod;}

il ll S2(ll n) {return n \* (n + 1) % mod \* (2 \* n + 1) % mod \* inv6 % mod;}

query calc(ll a, ll b, ll c, ll n) {

query ret;

ll s1, s2, ac, bc;

ll m;

ac = a / c, bc = b / c, s1 = S1(n), s2 = S2(n), m = (a \* n + b) / c;

if (a == 0) {

ret.f = bc \* (n + 1) % mod;

ret.g = bc \* s1 % mod;

ret.h = bc \* bc % mod \* (n + 1) % mod;

} else if (a >= c || b >= c) {

auto pre = calc(a % c, b % c, c, n);

ret.f = (s1 \* ac % mod + (n + 1) \* bc % mod + pre.f) % mod;

ret.g = (ac \* s2 % mod + bc \* s1 % mod + pre.g) % mod;

ret.h = (ac \* ac % mod \* s2 + 2 \* s1 \* ac % mod \* bc % mod + (n + 1) \* bc % mod \* bc % mod + 2 \* ac \* pre.g % mod + 2 \* bc \* pre.f % mod + pre.h) % mod;

} else {

auto pre = calc(c, c - b - 1, a, m - 1);

ret.f = (n \* m % mod + mod - pre.f) % mod;

ret.g = (m \* n % mod \* (n + 1) % mod + mod - pre.h + mod - pre.f) % mod \* inv2 % mod;

ret.h = (n \* m % mod \* (m + 1) % mod + mod - 2 \* pre.g % mod + mod - 2 \* pre.f % mod + mod - ret.f) % mod;

}

return ret;

}

### CRT

//CRT，用于求解x同余多个ai(mod mi) mi互质

//M=m1\*m2..mn Mi=M/mi Miti同余1(mod mi)

//则可以构造x=∑(aiMiti)

//任意解为 x+kM 下面给出的是求最小正整数解 (x+kM)%M、

ll n,m[maxn],a[maxn],M,Mi[maxn];

void exgcd(ll a,ll b,ll &x,ll &y){

if(b==0){x=1,y=0;return;}

exgcd(b,a%b,x,y);

int z=x;x=y;y=z-y\*(a/b);

}

ll CRT(){

ll M=1,d,ans=0;

for(int i=1;i<=n;++i)M\*=m[i];

for(int i=1;i<=n;++i){

ll x=0,y=0;

Mi[i]=M/m[i];

exgcd(Mi[i],m[i],x,y);

ans=(ans+a[i]\*Mi[i]\*x)%M;

}

return (ans+M)%M;

}

### EXCRT

//EXCRT求解x同余ai(mod mi) mi不互质

ll mi[maxn],ai[maxn];

ll mul(ll a,ll b){//快速乘防爆ll

ll ans=0;

while(b){

if(b&1)ans=(ans+a)%mod;

a=(a+a)%mod;

b>>=1;

}

return ans;

}

ll exgcd(ll a,ll b,ll &x,ll &y){

if(!b){x=1,y=0;return a;}

ll gcd=exgcd(b,a%b,x,y);

ll z=x;x=y;y=z-y\*(a/b);

return gcd;

}

ll excrt(){

ll x,y,k;

ll M=mi[1],ans=ai[1];//特判只有一个方程的解

for(int i=2;i<=n;++i){

ll a=M,b=mi[i],c=((ai[i]-ans)%b+b)%b;//ax同余c(mod b)

ll gcd=exgcd(a,b,x,y),bg=b/gcd;

if(c%gcd!=0)return -1;//判断是否有解

x=mul(x,c/gcd,bg);//exgcd求出x的解

ans+=x\*M; //加法更新前k个的答案

M\*=bg; //前k个的lcm

ans=(ans%M+M)%M;

}

return ans;

}

### 阶与原根



const int N = 1000005;

int cnt, tot, p;

int vis[N], prime[N], fac[N];

//质数筛

void Factor(int x) {

tot = 0;

int t = (int) sqrt(x + 0.5);

for(int i = 1; prime[i] <= t; i++) {

if(x % prime[i] == 0) {

fac[tot++] = prime[i];

while(x % prime[i] == 0) x /= prime[i];

}

}

if(x > 1) fac[tot++] = x;

}

int mypow(int a, int b, int mod) {

int ans = 1;

while(b) {

if(b & 1) ans = 1ll\*ans\*a% mod;

a = 1ll\*a \* a % mod;

b>>= 1;

}

return ans;

}

init();

int solve(int \_p) {

p=\_p;

Factor(p - 1);

for(int g = 2; g < p; g++) {

bool flag = true;

for(int i = 0; i < tot; i++) {

int t = (p - 1) / fac[i];

if(quick\_pow(g, t, p) == 1) {

flag = false;

break;

}

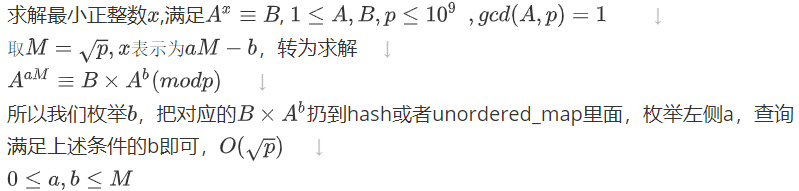
}

if(flag) {return g;break;}

}

}

### BSGS and EXBSGS



**bsgs**

//手写hash

const int HashMod=sqrt(1e9+10);

struct HashTable

{

struct Line{int u,v,next;}e[1000000];

int h[HashMod],cnt;

void Add(int u,int v,int w){e[++cnt]=(Line){w,v,h[u]};h[u]=cnt;}

void Clear(){memset(h,0,sizeof(h));cnt=0;}

void Hash(int x,int k){

int s=x%HashMod;

Add(s,k,x);

}

int Query(int x){

int s=x%HashMod;

for(int i=h[s];i;i=e[i].next)

if(e[i].u==x)return e[i].v;

return -1;

}

}Hash;

ll mypow(ll a,ll b,ll p){

ll ans=1;

while(b){

if(b&1)ans=ans\*a%p;

a=a\*a%p;

b>>=1;

}

return ans;

}

int BSGS(int A,int B,int p)

{

if(A%p==0){return -1;//无解}

A%=p;B%=p;

if(B==1){return 0;}

int m=sqrt(p)+1;

Hash.Clear();

for(int i=0,t=B;i<m;++i,t=1ll\*t\*A%p)Hash.Hash(t,i);

for(int i=1,tt=mypow(A,m,p),t=tt;i<=m+1;++i,t=1ll\*t\*tt%p){

int k=Hash.Query(t);if(k==-1){continue; };

return i\*m-k;

}

return -1;//无解

}

//unorder\_map版本

unordered\_map<ll,ll>Map;

ll BSGS(ll A,ll B){

Map.clear();

ll m=sqrt(p)+1,tmp=0;

if(A%p==0&&B==0)return 1;

if(A%p==0&&B!=0)return -1;

for(int i=0;i<=m;++i){

if(!i){ tmp=B%p;Map[tmp]=i;continue;}

tmp=(tmp\*A)%p;

Map[tmp]=i;

}

tmp=1;ll t=mypow(A,m);

for(int i=1;i\*i<=p;++i){

tmp=(tmp\*t)%p;

if(Map[tmp]){

ll ans=i\*m-Map[tmp];

return ans;

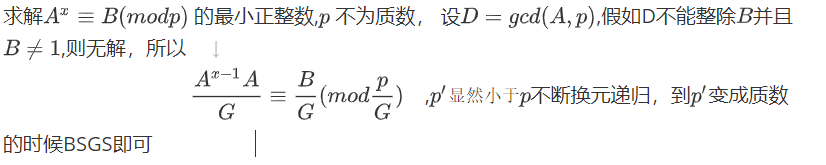
}

}

return -1;//-1无解

}

**Exbsgs**



int gcd(int a,int b){ return b?gcd(b,a%b):a;}

const int HashMod=123456;//质数

struct HashTable

{

struct Line{int u,v,next;}e[1000000];

int h[HashMod],cnt;

void Add(int u,int v,int w){e[++cnt]=(Line){w,v,h[u]};h[u]=cnt;}

void clear(){memset(h,0,sizeof(h));cnt=0;}

void Hash(int x,int k){

int s=x%HashMod;

Add(s,k,x);

}

int Query(int x){

int s=x%HashMod;

for(int i=h[s];i;i=e[i].next)

if(e[i].u==x)return e[i].v;

return -1;

}

}Hash;

int mypow(int a,int b,int mod){

int ans=1;

while(b){

if(b&1)ans=1ll\*ans\*a%mod;

a=1ll\*a\*a%mod;b>>=1;

}

return ans;

}

int exbsgs(int A,int B,int p){

if(B==1){return 0; }

int k=0,a=1;

while(1){

int d=gcd(A,p);

if(d==1)break;

if(B%d){ return -1;}//无解

B/=d;p/=d;++k;

a=1ll\*a\*A/d%p;

if(B==a)return k;

}

Hash.clear();

int m=sqrt(p)+1;

for(int i=0,t=B;i<m;++i,t=1ll\*t\*A%p)Hash.Hash(t,i);

for(int i=1,tt=mypow(A,m,p),t=1ll\*a\*tt%p;i<=m;++i,t=1ll\*t\*tt%p){

int x=Hash.Query(t);

if(x==-1){continue; };

return i\*m-x+k;

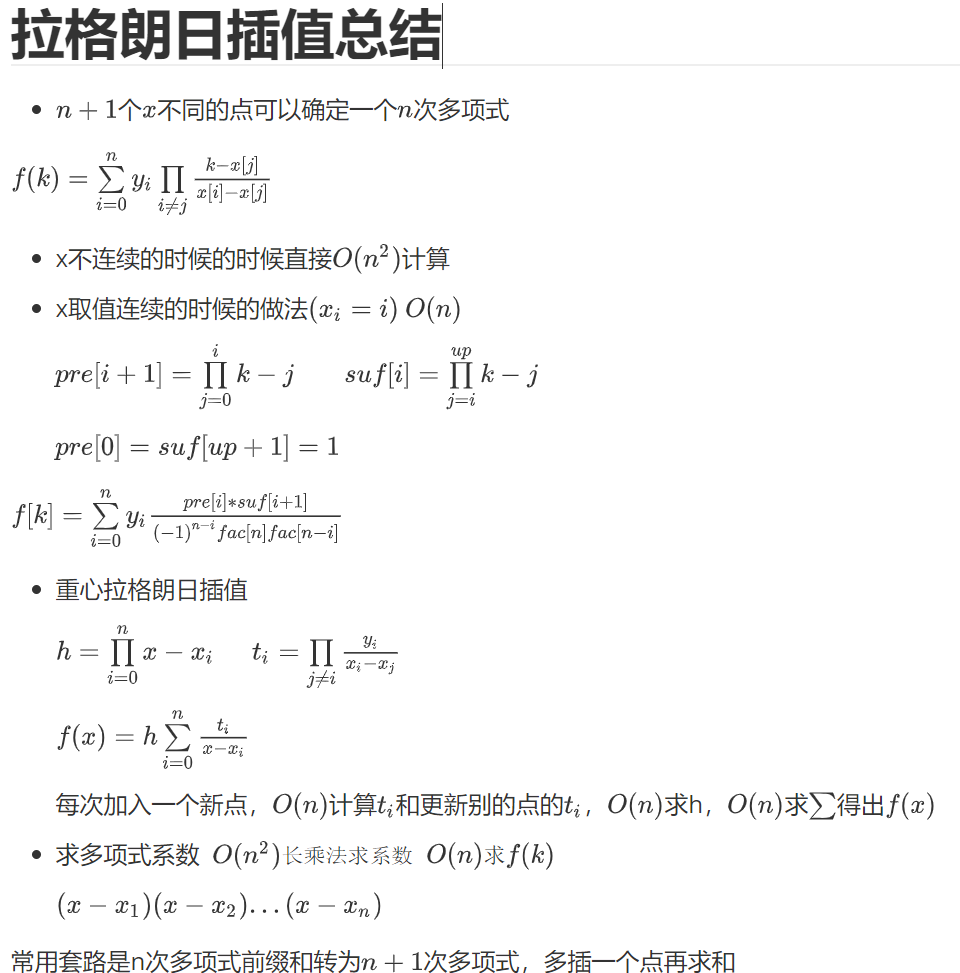
}

return -1;//无解

}

# 多项式

## 拉格朗日插值



struct Lglr{

void init(){

fac[1]=fac[0]=1;

for(int i=2;i<maxn;++i)fac[i]=fac[i-1]\*i%mod;

facinv[maxn-1]=mypow(fac[maxn-1],mod-2);

for(int i=maxn-2;i>=0;--i)

facinv[i]=facinv[i+1]\*(i+1)%mod;

}

//给定(0,f0)...(up,fup)求 up次多项式 f(x)的值 O(n)

ll cal(ll x,ll\*a,int up){//pre往偏移一位

if(x<=up)return a[x];

pre[0]=1;suf[up+1]=1;

for(int i=0;i<=up;++i)pre[i+1]=(x-i)%mod\*pre[i]%mod;

for(int i=up;i>=0;--i)suf[i]=(x-i)%mod\*suf[i+1]%mod;

ll ans=0;

for(int i=0;i<=up;++i){

ll f=a[i]\*pre[i]%mod\*suf[i+1]%mod\*facinv[i]%mod\*facinv[up-i]%mod;

if((up-i)&1)ans=(ans-f+mod)%mod;

else ans=(ans+f)%mod;

}

return ans;

}

//给定n+1个点，求n次多项式f(k)的值 O(n^2)

ll inv(int x){return mypow(x,mod-2);}

ll cal2(ll k,ll\*x,ll\*y){

for(int i=0;i<=n;++i)cin>>x[i]>>y[i];

ll ans=0;

for(int i=0;i<=n;++i){

ll zi=y[i],mu=1;

for(int j=0;j<=n;++j){

if(i!=j){

zi=zi\*(k-x[j])%mod;

mu=mu\*(x[i]-x[j])%mod;

}

}

ans=(ans+zi\*inv(mu)%mod)%mod;

}

return (ans+mod)%mod;

}

//O(n^2)求多项式 O(n)单点求f(k)

ll a[maxn], b[maxn], c[maxn], temp[maxn];

ll x[maxn], y[maxn];

int n;

void mul(ll \*f, int len, ll t) { //len为多项式的次数+1，函数让多项式f变成f\*(x+t)

for (int i = len; i > 0; --i)

temp[i] = f[i], f[i] = f[i - 1];

temp[0] = f[0], f[0] = 0;

for (int i = 0; i <= len; ++i)

f[i] = (f[i] + t \* temp[i]) % mod;

}

void dev(ll \*f, ll \*r, ll t) { //f是被除多项式的系数，r保存f除以x+t的结果

for (int i = 0; i <= n; ++i)

temp[i] = f[i];

for (int i = n; i > 0; --i) {

r[i - 1] = temp[i];

temp[i - 1] = (temp[i - 1] - t \* temp[i]) % mod;

}

}

void lglr() {

memset(a, 0, sizeof a);

b[1] = 1, b[0] = -x[1];

for (int i = 2; i <= n; ++i) {

mul(b, i, -x[i]);

}//预处理(x-x1)\*(x-x2)...\*(x-xn)

for (int i = 1; i <= n; ++i) {

ll fz = 1;

for (int j = 1; j <= n; ++j) {

if (j == i)

continue;

fz = fz \* (x[i] - x[j]) % mod;

}

fz = qpow(fz, mod - 2);

fz = fz \* y[i] % mod; //得到多项式系数

dev(b, c, -x[i]);//得到多项式，保存在c数组

for (int j = 0; j < n; ++j)

a[j] = (a[j] + fz \* c[j]) % mod;

}

}

ll sum(){//n点n-1次多项式

cin >> n >> k;

for (int i = 1; i <= n; ++i)

scanf("%lld%lld", &x[i], &y[i]);

lglr();

ll ans = 0,res=1;

for (int i = 0; i < n; ++i) {

ans = (ans + res \* a[i]) % mod;

res = res \* k % mod;

}

return (ans+mod)%mod;

}

}lglr;

## FFT or NTT

#include<bits/stdc++.h>

using namespace std;

using ll=long long;

using db=double;

const int maxn=4e6+5;

namespace Poly{

const db PI=acos(-1.0);

int rev[maxn];

struct cp{

db x,y;

friend cp operator+(const cp&a,const cp&b){

return cp{ a.x+b.x,a.y+b.y};

}

friend cp operator-(const cp&a,const cp&b){

return cp{ a.x-b.x,a.y-b.y};

}

friend cp operator\*(const cp&a,const cp&b){

return cp{ a.x\*b.x-a.y\*b.y,a.x\*b.y+a.y\*b.x};

}

}A[maxn],B[maxn],C[maxn];//大小(n+m)<<1

void fft(cp A[],int lim,int op){

for(int i=0;i<lim;++i)if(i<rev[i])swap(A[i],A[rev[i]]);

for(int i=2;i<=lim;i<<=1){

cp wn={ cos(2\*PI/i),sin(2\*PI/i)};

int d=i>>1;

for(int j=0;j<lim;j+=i){

cp wk={ 1,0};

for(int k=j;k<j+d;++k,wk=wk\*wn){

cp x=A[k],y=wk\*A[k+d];

A[k]=x+y,A[k+d]=x-y;

}

}

}

if(op==-1){

reverse(A+1,A+lim);

for(int i=0;i<lim;++i)A[i].x/=lim;

}

}

void workFFT(int a[],int n,int b[],int m,int c[]){

int lim=1;

while(lim<=n+m)lim<<=1;

for(int i=0;i<lim;++i){

rev[i]=(rev[i>>1]>>1)|((i&1)\*(lim>>1));

}

for(int i=0;i<lim;++i){

A[i]=i<=n?cp{ a[i],0}:cp{ 0,0};//输入a 0~n次系数

B[i]=i<=m?cp{ b[i],0}:cp{ 0,0};//输入b 0~n次系数

}

fft(A,lim,1),fft(B,lim,1);

for(int i=0;i<lim;++i)C[i]=A[i]\*B[i];

fft(C,lim,-1);

for(int i=0;i<=n+m;++i)c[i]=(int)(C[i].x+0.5);

}

//NTT

int f[maxn],g[maxn],lim,pre[maxn],c[maxn],n,m,rev[maxn];

const int mod=998244353,G=3,invG=332748118;

int mypow(int a,int b){

int ans=1;

while(b){

if(b&1)ans=(ll)ans\*a%mod;

a=(ll)a\*a%mod;

b>>=1;

}

return ans;

}

int Add(int x,int y){

x+=y;

if(x>=mod)x-=mod;

return x;

}

int Sub(int x,int y){

x-=y;

if(x<0)x+=mod;

return x;

}

int mul(int x,int y){ return (ll)x\*y%mod;}

int inv(int x){ return mypow(x,mod-2);}

void getlim(int x){

lim=1;

while(lim<x)lim<<=1;

}

void initrev(){

for(int i=0;i<lim;++i){

rev[i]=(rev[i>>1]>>1)|((i&1)\*(lim>>1));

}

}

void initpre(){

for(int i=1;i<lim;i<<=1){ //一半长度

int w = mypow(3, (mod - 1) / (i << 1));//g^((p-1)/(n))

pre[i] = 1;

for (int j = 1; j < i; j++) pre[i + j] = mul(pre[i + j - 1], w);

}

}

void ntt(int\*a,bool tp){

for (int i = 0; i < lim; i++) if (i < rev[i]) swap(a[i], a[rev[i]]);

for (int mid = 1, cnt = 0; mid < lim; mid <<= 1, cnt++)

for (int j = 0, len = mid << 1; j < lim; j += len)

for (int k = 0; k < mid; k++) {

int x = a[j + k], y = mul(pre[mid + k], a[j + k + mid]);

a[j + k] = Add(x, y);

a[j + k + mid] = Sub(x, y);

}

if (tp) return;

int invlim = inv(lim);

for (int i = 0; i < lim; i++) a[i] = mul(a[i], invlim);

reverse(a + 1, a + lim);

}

void getmul(int n,int m,int\*a,int\*b){

getlim(n+m+2);

initpre();

initrev();

ntt(a,1);ntt(b,1);

for(int i=0;i<lim;++i)c[i]=mul(a[i],b[i]);

ntt(c,0);

}

void solve(){

cin>>n>>m;

for(int i=0;i<=n;++i)cin>>f[i];

for(int i=0;i<=m;++i)cin>>g[i];

getmul(n,m,f,g);

//[0,n+m]对应系数

}

}

using namespace Poly;

int main(){

ios::sync\_with\_stdio(false);cin.tie(0);

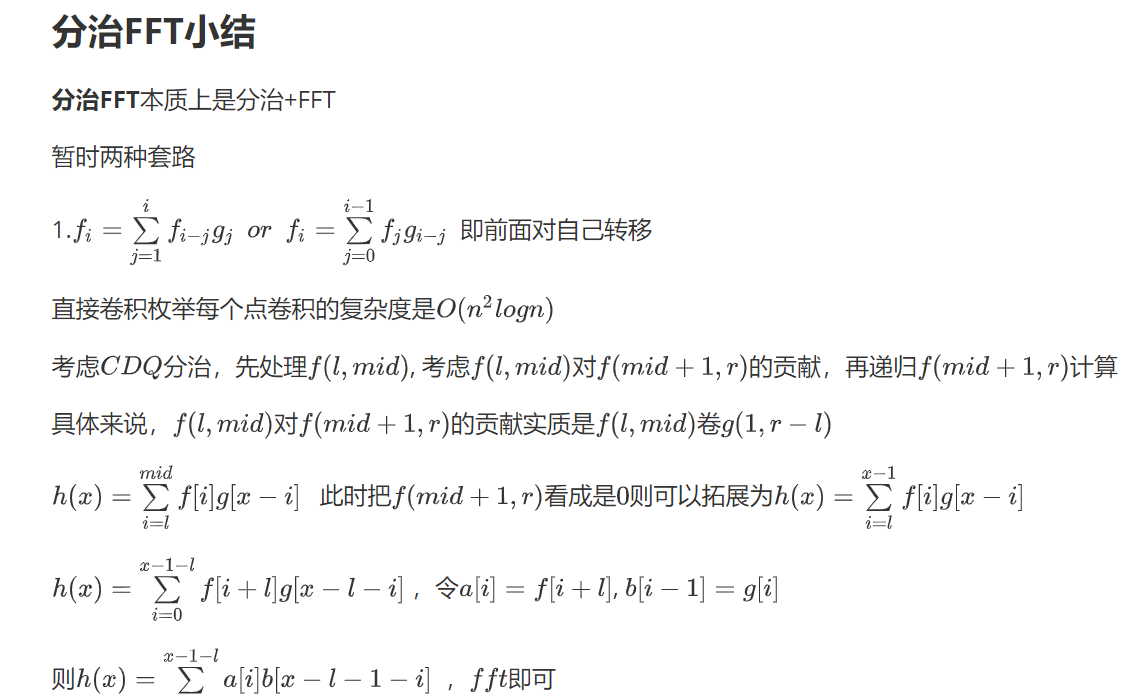
solve();

for(int i=0;i<=n+m;++i)cout<<c[i]<<" ";

return 0;

}

## 分治FFT



void cdq(int l,int r){ //一般ntt模板 给f[0] g[1~n-1]算f[1~n-1]

if(l==r)return;

int mid=l+r>>1,length=r-l+1;

cdq(l,mid);//先做完左边

getlim(length+1);//r-l+2

initrev();

initpre();

for(int i=0;i<lim;++i)a[i]=b[i]=0;

for(int i=l;i<=mid;++i)a[i-l]=f[i];//考虑贡献到[mid+1,r]即为

for(int i=1;i<length;++i)b[i-1]=g[i];//f[l,mid]卷g[1,r-l]

ntt(a,1);ntt(b,1);

for(int i=0;i<lim;++i)a[i]=mul(a[i],b[i]);

ntt(a,0);

for(int i=mid+1;i<=r;++i){

f[i]+=a[i-1-l];

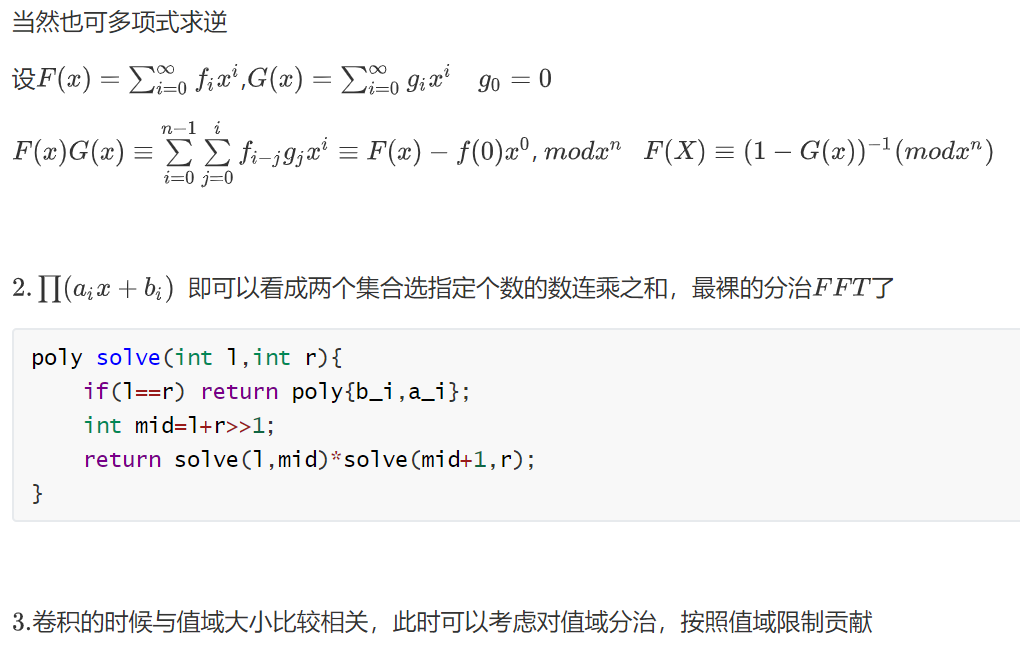
if(f[i]>=mod)f[i]-=mod;

if(f[i]<0)f[i]+=mod;

}

cdq(mid+1,r);

}



## Vector全家桶板子

#include<bits/stdc++.h>

using namespace std;

using ull=unsigned long long;

using ll=long long;

const int mod=998244353,G=3;//原根

const int maxn=2e5+5;

template<typename T>

void read(T &f){

f=0;T fu=1;char c=getchar();

while(c<'0'||c>'9'){ if(c=='-')fu=-1;c=getchar();}

while(c>='0'&&c<='9'){ f=(f<<3)+(f<<1)+(c&15);c=getchar();}

f\*=fu;

}

template<typename T>

void print(T x){

if(x<0)putchar('-'),x=-x;

if(x<10)putchar(x+48);

else print(x/10),putchar(x%10+48);

}

template <typename T>

void print(T x, char t) {

print(x); putchar(t);

}

inline int Add(int x,int y){

x+=y;

if(x>=mod)x-=mod;

return x;

}

inline int Sub(int x,int y){

x-=y;

if(x<0)x+=mod;

return x;

}

inline int mul(int x,int y){ return 1ll\*x\*y%mod;}

int mypow(int a,int b){

int ans=1;

while(b){

if(b&1)ans=mul(ans,a);

a=mul(a,a);

b>>=1;

}

return ans;

}

namespace Poly{

typedef vector<int>poly;

poly roots,rev;

void getRevRoot(int base){

int lim=1<<base;

if((int)roots.size()==lim)return;

roots.resize(lim);rev.resize(lim);

for(int i=1;i<lim;++i)rev[i]=(rev[i>>1]>>1)|((i&1)<<(base-1));

for(int mid=1;mid<lim;mid<<=1){

int wn=mypow(G,(mod-1)/(mid<<1));

roots[mid]=1;

for(int i=1;i<mid;++i)roots[mid+i]=mul(roots[mid+i-1],wn);

}

}

void ntt(poly&a,int base){

int lim=1<<base;

for(int i=0;i<lim;++i)if(i<rev[i])swap(a[i],a[rev[i]]);

for(int mid=1;mid<lim;mid<<=1){

for(int i=0;i<lim;i+=(mid<<1)){

for(int j=0;j<mid;++j){

int x=a[i+j],y=mul(a[i+j+mid],roots[j+mid]);

a[i+j]=Add(x,y);

a[i+j+mid]=Sub(x,y);

}

}

}

}

poly operator\*(poly a,poly b){

int lim=(int)a.size()+(int)b.size()-1,base=0;

while((1<<base)<lim)++base;

a.resize(1<<base);b.resize(1<<base);

getRevRoot(base);

ntt(a,base);ntt(b,base);

for(int i=0;i<(1<<base);++i)a[i]=mul(a[i],b[i]);

ntt(a,base);

reverse(a.begin()+1,a.end());

a.resize(lim);

int inv=mypow(1<<base,mod-2);

for(int i=0;i<lim;++i)a[i]=mul(a[i],inv);

return a;

}

poly polyInv(poly f,int base){

int lim=1<<base;

f.resize(lim);

if(lim==1){

poly ans(1,mypow(f[0],mod-2));

return ans;

}

poly g(1<<base,0),g0=polyInv(f,base-1),tmp=g0\*g0\*f;

for(int i=0;i<(1<<(base-1));++i)g[i]=Add(g0[i],g0[i]);

for(int i=0;i<(1<<base);++i)g[i]=Sub(g[i],tmp[i]);

return g;

}

poly polyInv(poly f){

int lim=(int)f.size(),base=0;

while((1<<base)<lim)++base;

f=polyInv(f,base);f.resize(lim);

return f;

}

poly operator+(const poly&a,const poly&b){

poly c=a;

c.resize(max(a.size(),b.size()));

int lim=(int)b.size();

for(int i=0;i<lim;++i)c[i]=Add(c[i],b[i]);

return c;

}

poly operator-(const poly&a,const poly&b){

poly c=a;

c.resize(max(a.size(),b.size()));

int lim=(int)b.size();

for(int i=0;i<lim;++i)c[i]=Sub(c[i],b[i]);

return c;

}

poly operator\*(const int&b,const poly&a){

poly c=a;

int lim=(int)a.size();

for(int i=0;i<lim;++i)c[i]=mul(b,c[i]);

return c;

}

int inv[maxn<<2];//未知

void initInv(){

inv[1]=1;

for(int i=2;i<maxn;++i)inv[i]=mul(inv[mod%i],(mod-mod/i));//跑2倍

}

//多项式求导

poly deri(poly a){

int lim=(int)a.size()-1;

for(int i=0;i<lim;++i)a[i]=mul(a[i+1],i+1);

a.resize(lim);

return a;

}

//多项式积分

poly inte(poly a){ //!!记得init inv

int lim=(int)a.size()+1;

a.resize(lim);

for(int i=lim-1;i>=1;--i)a[i]=mul(a[i-1],inv[i]);

a[0]=0;//原a\_0=1才可以用于求ln ln(a\_0)=0

return a;

}

poly polyLn(poly a){

int lim=(int)a.size();//

// a=inte(deri(a)\*polyInv(a));//n+m大小 逆元开第一个二次幂

a=deri(a)\*polyInv(a);//mod (x^n)下 [0,n-1]用

a.resize(lim-1);

a=inte(a);

// a.resize(lim);

return a;

}

poly polySqr(poly a,int base){

int lim=1<<base;

a.resize(lim);

if(lim==1){

poly ans(1,1);//a[0]=1

return ans;

}

poly g0=polySqr(a,base-1);g0.resize(lim);

a=a\*polyInv(g0)+g0;

a.resize(lim);

for(int i=0;i<lim;++i){

if(a[i]&1)a[i]=(a[i]+mod)>>1;

else a[i]=a[i]>>1;

}

return a;

}

poly polySqr(poly f){

int lim=(int)f.size(),base=0;

while((1<<base)<lim)++base;

f=polySqr(f,base);f.resize(lim);

return f;

}

poly polyExp(poly f,int base){

int lim=1<<base;f.resize(lim);

if(lim==1){

poly ans(1,1);

return ans;

}

poly g0=polyExp(f,base-1),g(1,1);

g0.resize(lim);//同样%x^n下,所以要两倍

g=g0\*(g-polyLn(g0)+f);

return g;

}

poly polyExp(poly f){

int lim=(int)f.size(),base=0;

while((1<<base)<lim)++base;

f=polyExp(f,base);f.resize(lim);

return f;

}

poly polyPow(poly a,int k){ //a\_0=1 多项式系数mod 998244353下算

return polyExp(k\*polyLn(a));

}

poly polyPow(poly a){

int n,c,k=0;

scanf("%d",&n);

a.resize(n);

c=getchar();while(isspace(c))c=getchar();//k是int直接读

for(;isdigit(c);c=getchar())k=(k\*10ll+(c^'0'))%mod;

for(int i=0;i<n;++i)scanf("%d",&a[i]);

return polyPow(a,k);

}

int norm(int n){return 1<<(32-\_\_builtin\_clz(n-1));}

void norm(poly &a){

if(!a.empty()){

a.resize(norm(a.size()),0);

}else

a={0};

}

poly polyPow2(poly a,int b1,int b2){//b1=b%P,b2=b%phi(P) and b>=n iff a[0]>0

int n=a.size(),d=0,k;

while(d<n&&!a[d])++d;

if((ll)d\*b1>=n)return poly(n,0);

a.erase(a.begin(),a.begin()+d);

k=mypow(a[0],mod-2);

a=k\*a;

norm(a);

a=mypow(k,mod-1-b2)\*polyPow(a,b1);

a.resize(n);d\*=b1;

for(int i=n-1;i>=0;--i)a[i]=i>=d?a[i-d]:0;

return a;

}

void P5273(){//A(x)^s次方 s是大数

int n;

int k1 = 0, k2 = 0, big = 0; string s;

cin >> n >> s;

poly a(n,0);

for(auto c: s){

k1 = (10ll \* k1 + c - '0') % mod,

k2 = (10ll \* k2 + c - '0') % (mod - 1), big = big || k1 >= n;

}

for (auto &x: a)

cin >> x;

a = big && !a[0] ? poly(n,0) : polyPow2(a, k1, k2);

for (auto x : a)

cout<<x<<" ";

}

}

### FWT与子集卷积

//FWT

void FWT\_OR(poly&a,bool f){

int n=a.size();

for(int l=2,m=1;l<=n;l<<=1,m<<=1){

for(int j=0;j<n;j+=l)

for(int i=0;i<m;++i){

if(!f)

a[i+j+m]=Add(a[i+j+m],a[i+j]);

else

a[i+j+m]=Sub(a[i+j+m],a[i+j]);

}

}

}

void FWT\_AND(poly&a,bool f){

int n=a.size();

for(int l=2,m=1;l<=n;l<<=1,m<<=1){

for(int j=0;j<n;j+=l)

for(int i=0;i<m;++i){

if(!f)

a[i+j]=Add(a[i+j],a[i+j+m]);

else

a[i+j]=Sub(a[i+j],a[i+j+m]);

}

}

}

void FWT\_XOR(poly&a,bool f){

int n=a.size(),inv2=(mod+1)>>1;

for(int l=2,m=1;l<=n;l<<=1,m<<=1){

for(int j=0;j<n;j+=l)

for(int i=0;i<m;++i){

int x=a[i+j],y=a[i+j+m];//换LL时候也要换

if(!f){

a[i+j]=Add(x,y);

a[i+j+m]=Sub(x,y);

}else{

a[i+j]=mul(Add(x,y),inv2);

a[i+j+m]=mul(Sub(x,y),inv2);

}

}

}

}

int getlim(int x){ //传的是(1<<max) 不是bit

int lim=1;

while(lim<x)lim<<=1;

return lim;

}

poly OR(poly a,poly b){

int n=max(a.size(),b.size());

int lim=getlim(n);

a.resize(lim);b.resize(lim);

FWT\_OR(a,0);

FWT\_OR(b,0);

for(int i=0;i<lim;++i)a[i]=mul(a[i],b[i]);

FWT\_OR(a,1);

return a;

}

poly AND(poly a,poly b){

int n=max(a.size(),b.size());

int lim=getlim(n);

a.resize(lim);b.resize(lim);

FWT\_AND(a,0);

FWT\_AND(b,0);

for(int i=0;i<lim;++i)a[i]=mul(a[i],b[i]);

FWT\_AND(a,1);

return a;

}

poly XOR(poly a,poly b){

int n=max(a.size(),b.size());

int lim=getlim(n);

a.resize(lim);b.resize(lim);

FWT\_XOR(a,0);

FWT\_XOR(b,0);

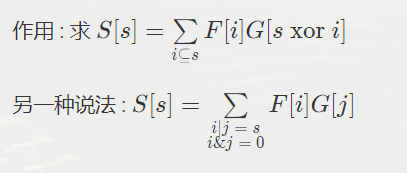
for(int i=0;i<lim;++i)a[i]=mul(a[i],b[i]);

FWT\_XOR(a,1);

return a;

}

**子集卷积**



多开一位，f(i,s)表示集合s的子集中所有位数为i的值的和

#include<bits/stdc++.h>

#define cbit(x) \_\_builtin\_popcount(x)

using namespace std;

using ll=long long;

const int maxn=(1<<20)+100;

const int mod=1e9+9;

int a[22][maxn],b[22][maxn],c[22][maxn];

inline int mul(int x,int y){ return (ll)x\*y%mod;}

inline int Add(int x,int y){

x+=y;

if(x>=mod)x-=mod;

return x;

}

inline int Sub(int x,int y){

x-=y;

if(x<0)x+=mod;

return x;

}

void fwt(int a[],bool f,int lim){

for(int l=2,m=1;l<=lim;l<<=1,m<<=1){

for(int j=0;j<lim;j+=l){

for(int i=0;i<m;++i){

if(!f)a[i+j+m]=Add(a[i+j+m],a[i+j]);

else a[i+j+m]=Sub(a[i+j+m],a[i+j]);

}

}

}

}

int getlim(int x){

int lim=1;

while(lim<x)lim<<=1;

return lim;

}

int main(){

ios::sync\_with\_stdio(false);

cin.tie(0);

int n;

cin>>n;

int lim=(1<<n);

for(int i=0;i<lim;++i)cin>>a[cbit(i)][i];

for(int i=0;i<lim;++i)cin>>b[cbit(i)][i];

for(int i=0;i<=n;++i)fwt(a[i],0,lim);

for(int i=0;i<=n;++i)fwt(b[i],0,lim);

for(int i=0;i<=n;++i){

for(int j=0;i+j<=n;++j){

for(int k=0;k<lim;++k)

c[i+j][k]=Add(c[i+j][k],mul(a[i][k],b[j][k]));

}

}

for(int i=0;i<=n;++i)fwt(c[i],1,lim);

for(int i=0;i<((1<<n)-1);++i)cout<<c[cbit(i)][i]<<" ";

cout<<c[n][(1<<n)-1];

return 0;

}

### 特殊数计算

#### 两类斯特林数

const int N=3e5+5;

int fac[N],facinv[N];

void init(int n){

fac[0]=1;

for(int i=1;i<=n;++i)fac[i]=mul(fac[i-1],i);

facinv[n]=mypow(fac[n],mod-2);

for(int i=n-1;i>=0;--i)facinv[i]=mul(facinv[i+1],i+1);

}

using namespace Poly;

namespace Stiring{

namespace Stiring1{

poly Offset(poly f,int c){ //f(x)->f(x+c)

poly g(c+1),h(c+1);

for(int i=0,j=1;i<=c;++i,j=(ll)j\*c%mod){

h[i]=mul(j,facinv[i]);

g[i]=mul(f[c-i],fac[c-i]);

}

g=g\*h;

for(int i=0;i<=c;++i)h[i]=mul(g[c-i],facinv[i]);

return h;

}

poly Solve(int n){//x^(up n)

if(!n)return { 1};

int m=n/2;

poly f=Solve(m),g=Offset(f,m);

f=f\*g;

f.resize(n+1);

if(n&1){//乘上(x+n-1)

for(int i=n;i>=0;--i){

f[i]=mul(f[i],n-1);

if(i>0)

f[i]=Add(f[i],f[i-1]);

}

}

return f;

}

poly Stiring1\_row(int n){

init(n+1);

return Solve(n);

}

poly Stiring1\_line(int n,int k){

poly f(n+1);

int fack=1;

for(int i=1;i<=k;++i)fack=(ll)fack\*i%mod;

fack=mypow(fack,mod-2);

for(int i=0;i<=n;++i)f[i]=inv[i];//记得initInv()

f=polyPow2(f,k,k);

for(int i=0,fac=1;i<=n;++i,fac=(ll)fac\*i%mod)f[i]=(ll)f[i]\*fack%mod\*fac%mod;

f.resize(n+1);

return f;

}

vector<vector<int>> Stiring1All(int n,int m){//n行,m列

vector<vector<int>>dp(n+1,vector<int>(m+1,0));

dp[0][0]=1;

for(int i=1;i<=n;++i)

for(int j=1;j<=m;++j)

dp[i][j]=(dp[i-1][j-1]+(ll)(i-1)\*dp[i-1][j])%mod;

return dp;

}

}

namespace Stiring2{

poly Stiring2\_row(int n,int m){//第n行的[0,m]列

poly f(m+1),g(m+1);

init(m+1);

for(int i=0;i<=m;++i)f[i]=mul(mypow(-1,i),facinv[i]);

for(int i=0;i<=m;++i)g[i]=mul(mypow(i,n),facinv[i]);

f=f\*g;

f.resize(m+1);

return f;

}

poly Stiring2\_line(int n,int k){//第k列 [0,n]

poly f(n+1,0);

init(n+1);

for(int i=1;i<=n;++i)f[i]=facinv[i];//f[0]=0;

f=polyPow2(f,k,k);

for(int i=0;i<=n;++i)f[i]=(ll)f[i]\*facinv[k]%mod\*fac[i]%mod;

f.resize(n+1);

return f;

}

vector<vector<int>> Stiring2All(int n,int m){//n行,m列

vector<vector<int>>dp(n+1,vector<int>(m+1,0));

dp[0][0]=1;

for(int i=1;i<=n;++i)

for(int j=1;j<=m;++j)

dp[i][j]=(dp[i-1][j-1]+(ll)j\*dp[i-1][j])%mod;

return dp;

}

}

}

using namespace Stiring;

using namespace Stiring1;

using namespace Stiring2;

#### 贝尔数

namespace Bell{

//cal B[1,n]

poly BellAll(int n){

init(n+1);

poly ans(n+1,0);

for(int i=1;i<=n;++i)ans[i]=facinv[i];

return polyExp(ans);

}

} using namespace Bell;

# 计算几何

## 点相关

#include<bits/stdc++.h>

#define cha(p1,p2,p3) (sign(det(p2-p1,p3-p1)))//p2-p1叉p3-p1

using namespace std;

typedef double db;

const db eps=1e-8;

int sign(db a){ //符号正负

return a<-eps?-1:a>eps;

}

int dcmp(db a,db b){ //两数大小

return sign(a-b);

}

db dAbs(db a){return a\*sign(a);}

struct P{ //点向量类

db x,y;

P(db \_x=0,db \_y=0):x(\_x),y(\_y){ }

P operator+(P p){return P(x+p.x,y+p.y);}//左操作数

P operator-(P p){ return P(x-p.x,y-p.y);}

P operator\*(db d){return P(x\*d,y\*d);}

P operator/(db d){ return P(x/d,y/d);}

bool operator<(P p)const{//点按x排序后按y排序

int c=dcmp(x,p.x);

if(c)return c==-1;

return dcmp(y,p.y)==-1;

}

db len2(){ return x\*x+y\*y;}//能用ll就ll，精度

db len(){ return sqrt(len2());}

};

bool operator==(P a,P b){return !dcmp(a.x,b.x)&&!dcmp(a.y,b.y);}

int dis(P a,P b){

return (a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y);

}

db dot(P a,P b){//内积

return a.x\*b.x+a.y\*b.y;

}

db det(P a,P b){//叉积

return a.x\*b.y-a.y\*b.x;

}

//点积运用

db Angle(P a,P b){//(2)两向量夹角

return acos(dot(a,b)/len(a)/len(b));

}

db AngleSign(P a,P b){//(3)两夹角钝锐直

return sign(dot(a,b));

}

P proj(P p1,P p2,P p){//(4)求p投影到p1,p2直线的点

P a=p2-p1;

return p1+a\*(dot(p-p1,a)/a.len2());

}

P reflect(P p1,P p2,P p){//(5) p关于p1,p2直线对称的点

return proj(p1,p2,p)\*2-p;

}

P Unit(const P&a){//(4)求单位向量

double L=len(a);

return P(a.x/L,a.y/L);

}

db Area2(P a,P b,P c){//两向量平行四边形有向面积

return det(b-a,c-a);

}

P RotateL(const P&a,const double&rad){//逆时针旋转

return P(a.x\*cos(rad)-a.y\*sin(rad),a.y\*cos(rad)+a.x\*sin(rad));

}

P RotateR(const P&a,const double&rad){//顺时针旋转

return P(a.x\*cos(rad)+a.y\*sin(rad),a.y\*cos(rad)-a.x\*sin(rad));

}

P RotateL90(const P&a){//向量逆时针旋转90度

return P(-a.y,a.x);

}

P RotateR90(const P&a){//顺时针旋转90度

return P(a.y,-a.x);

}

## 线相关

struct L{//ps[0]->ps[1]

P ps[2];

P &operator[](int i){return ps[i];}

P dir(){return ps[1]-ps[0];}//直线方向向量

L(P X=P(),P Y=P()){ps[0]=X,ps[1]=Y;}

};

bool thesameLine(L x1,L x2){//两直线是否重合

P a=x2.ps[1]-x1.ps[1];

return (sign(det(x1.dir(),x2.dir()))==0&&sign(det(x2.dir(),a))==0);

}

bool parallel(L x1,L x2){//判两直线是否平行

P a=x2.ps[1]-x1.ps[1];

return sign(det(x1.dir(),x2.dir())==0&&sign(det(x2.dir(),a))!=0);

}

bool Orthogonal(L x1,L x2){//判两直线是否垂直

return sign(dot(x1.dir(),x2.dir()))==0;

}

bool onLine(L l1,P a){//点在直线上

return sign(det(l1.dir(),a-l1.ps[0]))==0;

}

bool LineInterset(L l1,L l2){//直线是否相交

return !parallel(l1,l2);

}

db PLinedis(L l1,P a){//平行四边形面积求点到直线距离

return dAbs(det(a-l1.ps[0],l1.dir())/l1.dir().len());

}

P LineInter(L l1,L l2){ //两直线交点 不平行的时候用

P a=l1.ps[0],b=l1.ps[1],c=l2.ps[0],d=l2.ps[1];

db t=det(a-c,c-d)/det(a-b,c-d);

return P(a.x+(b.x-a.x)\*t,a.y+(b.y-a.y)\*t);

}

## 线段相关

bool onSeg(P c,P a,P b){//判断c是否在线段ab上

return sign(det(c-a,b-a))==0&&sign(dot(c-a,c-b))<=0;

}

db PdisSeg(P c,P a,P b){//点c到线段ab的距离

if(a==b)return (c-a).len();

P x=c-a,y=c-b,z=b-a;

if(AngleSign(x,z)<0)return x.len();//c离a更近

if(AngleSign(y,z)>0)return y.len();//c离b更近

return dAbs(det(x,z)/z.len());

}

bool LineInSeg(L c,P a,P b){//直线C与线段ab是否交

if(c.ps[0]==c.ps[1])return false;//看情况特判直线是否是重点

return sign(det(c.dir(),a-c.ps[0]))\*sign(det(c.dir(),b-c.ps[0]))<=0;

}

bool recInter(db l1,db r1,db l2,db r2){//矩形相交 快速排斥

if(l1>r1)swap(l1,r1);if(l2>r2)swap(l2,r2);

return !(dcmp(r1,l2)==-1||dcmp(r2,l1)==-1);

}

bool isSegInter(P p1,P p2,P q1,P q2){//判 线段p1p2与q1q2是否相交 端点算交 重点无关

return recInter(p1.x,p2.x,q1.x,q2.x)&&recInter(p1.y,p2.y,q1.y,q2.y)&&

cha(p1,p2,q1)\*cha(p1,p2,q2)<=0&&cha(q1,q2,p1)\*cha(q1,q2,p2)<=0;

}

bool isSegInter(P p1,P p2,P q1,P q2){//判 线段p1p2与q1q2是否相交 端点不算交

return cha(p1,p2,q1)\*cha(p1,p2,q2)<0&&cha(q1,q2,p1)\*cha(q1,q2,p2)<0;

}

db SSdis(P p1,P p2,P q1,P q2){//线段p1p2到线段q1q2的最短路径

if(isSegInter(p1,p2,q1,q2))return 0;

return min(min(PdisSeg(p1,q1,q2),PdisSeg(p2,q1,q2)),min(PdisSeg(q1,p1,p2),PdisSeg(q2,p1,p2)));

}

## 三角形相关

//三角形

//重心 到三角形三顶点距离平方和最小的点

//到三角形三边距离之积最大的点

P gravity(P a,P b,P c){

db x=(a.x+b.x+c.x)/3;

db y=(a.y+b.y+c.y)/3;

return P(x,y);

}

P Incenter(P a,P b,P c){//内心 到三边距离相等

#define diss(a,b) sqrt(dis(a,b))

db A=diss(b,c),B=diss(a,c),C=diss(a,b);

db S=A+B+C;

db x=(A\*a.x+B\*b.x+C\*c.x)/S;

db y=(A\*a.y+B\*b.y+C\*c.y)/S;

return P(x,y);

}

P circumcenter(P a,P b,P c){//外心 到三顶点距离相等

db X1=a.x,Y1=a.y,X2=b.x,Y2=b.y,X3=c.x,Y3=c.y;

db a1=2\*(X2-X1),b1=2\*(Y2-Y1);

db c1=X2\*X2+Y2\*Y2-X1\*X1-Y1\*Y1;

db a2=2\*(X3-X2),b2=2\*(Y3-Y2);

db c2=X3\*X3+Y3\*Y3-X2\*X2-Y2\*Y2;

db x=(c1\*b2-c2\*b1)/(a1\*b2-a2\*b1);

db y=(a1\*c2-a2\*c1)/(a1\*b2-a2\*b1);

return P(x,y);

}

P perpencenter(P a,P b,P c){//垂心

db A1=b.x-c.x,B1=b.y-c.y;

db C1=A1\*a.y-B1\*a.x;

db A2=a.x-c.x,B2=a.y-c.y;

db C2=A2\*b.y-B2\*b.x;

db x=(A1\*C2-A2\*C1)/(A2\*B1-A1\*B2);

db y=(B1\*C2-B2\*C1)/(A2\*B1-A1\*B2);

return P(x,y);

}

P feiPoint(P a,P b,P c){//三角形费马点 到三顶点距离之和最小的点

#define diss(a,b) sqrt(dis(a,b))

P u=gravity(a,b,c),v;

db step=fabs(a.x)+fabs(b.x)+fabs(c.x)+fabs(a.y)+fabs(b.y)+fabs(c.y);

int i,j,k;

while(step>1e-10){

for(k=0;k<10;step/=2,k++)

for(i=-1;i<=1;++i)

for(j=-1;j<=1;++j){

v.x=u.x+step\*i;

v.y=u.y+step\*j;

if(dcmp(diss(u,a)+diss(u,b)+diss(u,c),diss(v,a)+diss(v,b)+diss(v,c))>0)

u=v;

}

}

return u;

}

## 多边形相关

db area(vector<P>p){//任意多边形面积 顺逆都可

db ans=0;

int sz=p.size();

for(int i=0;i<sz-1;++i)ans+=det(p[i],p[i+1]);

ans+=det(p[sz-1],p[0]);

return dAbs(ans/2);

}

//O(n)点在任意多边形内 点数为1或2也可

int contain(P ps[],P p,int n){//2:onSeg 1:inside 0：outside

int flag=0,cnt=0;

db tmp;

for(int i=1;i<=n;++i){

int j=i<n?i+1:1;

if(onSeg(p,ps[i],ps[j]))return 2;//点在多边形上

if(p.y>=min(ps[i].y,ps[j].y)&&p.y<max(ps[i].y,ps[j].y))

tmp=ps[i].x+(p.y-ps[i].y)/(ps[j].y-ps[i].y)\*(ps[j].x-ps[i].x),cnt+=(sign(tmp-p.x)>0);

}

return cnt&1;

}

int TuContain(P p[],int n,P a){//二分法判断a是否在凸多边形以内 必须是凸多边形

if(n==1)if(p[1]==a)return 2;else return 0; //特判非凸包 看情况加

else if(n==2)if(onSeg(a,p[1],p[2]))return 2;else return 0;

else{

//点必须按逆时针给 顺时针>改<即可

if(cha(p[1],a,p[2])>0||cha(p[1],p[n],a)>0)return 0;//P[1,2]或[1,n]外

if(onSeg(a,p[1],p[2])||onSeg(a,p[1],p[n]))return 2;//[1,2]/[1,n]上

int l=2,r=n-1;

while(l<r){

int mid=l+r+1>>1;

if(cha(p[1],p[mid],a)>0)l=mid;//使得a被mid,mid+1到1夹住

else r=mid-1;

}

if(cha(p[l],a,p[l+1])>0)return 0;

if(onSeg(a,p[l],p[l+1]))return 2;

return 1;

}

}

//线与多边形

//(1). 判断线段 AB 是否在任意多边形 Poly 以内：不相交且两端点 A,B 均在多边形以内。

//(2). 判断线段 AB 是否在凸多边形 Poly 以内：两端点 A,B 均在多边形以内。

//判两多边形是否相离 O(n^2logn)

int PPLI(P a[],P b[],int num1,int num2){//先求两凸包

for(int i1=1;i1<=num1;++i1){

int j1=i1<num1?i1+1:1;

for(int i2=1;i2<=num2;++i2){

int j2=i2<num2?i2+1:1;

if(isSegInter(a[i1],a[j1],b[i2],b[j2]))return 0;//线段相交不相离

if(TuContain(b,num2,a[i1])||TuContain(a,num1,b[i2]))return 0;//点在多边形内不相离

}

}

return 1;

}

## 圆相关

#include<bits/stdc++.h>

#define diss(a,b) sqrt(dis(a,b))

#define pb push\_back

using namespace std;

typedef long double db;

const db eps=1e-8;

const db PI=acos(-1.0);

int q;

int sign(db a){ //符号正负

return a<-eps?-1:a>eps;

}

int dcmp(db a,db b){ //两数大小

return sign(a-b);

}

struct P{ //点向量类

db x,y;

P(db \_x=0,db \_y=0):x(\_x),y(\_y){ }

P operator+(P p){return P(x+p.x,y+p.y);}//左操作数

P operator-(P p){ return P(x-p.x,y-p.y);}

P operator\*(db d){return P(x\*d,y\*d);}

P operator/(db d){ return P(x/d,y/d);}

db len2(){ return x\*x+y\*y;}//能用ll就ll，精度

db len(){ return sqrt(len2());}

bool operator<(P p)const{//点按x排序后按y排序

int c=dcmp(x,p.x);

if(c)return c==-1;

return dcmp(y,p.y)==-1;

}

};

db dis(P a,P b){

return (a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y);

}

struct L{//ps[0]->ps[1]

P ps[2];

P &operator[](int i){return ps[i];}

P dir(){return ps[1]-ps[0];}//直线方向向量

L(P X=P(),P Y=P()){ps[0]=X,ps[1]=Y;}

};

struct Cir{

db r;

P c;

Cir(P \_c=P(),db \_r=0):c(\_c),r(\_r){}

P GP(db b){//圆上过一点知道角度和距离

return P(c.x+cos(b)\*r,c.y+sin(b)\*r);

}

};

db dot(P a,P b){

return a.x\*b.x+a.y\*b.y;

}

db det(P a,P b){//叉积

return a.x\*b.y-a.y\*b.x;

}

db PLinedis(L l1,P a){//平行四边形面积求点到直线距离

return fabs(det(a-l1.ps[0],l1.dir())/l1.dir().len());

}

P LineInter(L l1,L l2){ //两直线交点 不平行的时候用

P a=l1.ps[0],b=l1.ps[1],c=l2.ps[0],d=l2.ps[1];

db t=det(a-c,c-d)/det(a-b,c-d);

return P(a.x+(b.x-a.x)\*t,a.y+(b.y-a.y)\*t);

}

//圆与直线交点

//线段就直接求判点是否在线段上

vector<P> getCircleLineSec(P c,db r,P p1,P p2){//返回空 or 1 or 2交点

vector<P>ans;

db dd=PLinedis(L(p1,p2),c);

if(dcmp(dd,r)>0)return ans;//离

P p=c;

db t;

p.x+=p1.y-p2.y;

p.y+=p2.x-p1.x;

p=LineInter(L(p,c),L(p1,p2));

t=sqrt(r\*r-diss(p,c)\*diss(p,c))/diss(p1,p2);

P x1,x2;

x1={p.x+(p2.x-p1.x)\*t,p.y+(p2.y-p1.y)\*t};

x2={p.x-(p2.x-p1.x)\*t,p.y-(p2.y-p1.y)\*t};

ans.pb(x1);

if(dcmp(dd,r)==0)//切

return ans;

else

ans.pb(x2);

return ans;

}

Cir read\_cir(){

Cir C;

cin>>C.c.x>>C.c.y>>C.r;

return C;

}

P RotateL90(const P&a){

return P(-a.y,a.x);

}

vector<P>getCircelIntersec(P a,db r,P b,db R){ //圆与圆交点

vector<P>ret;

db d=diss(a,b);

if (dcmp(d,r+R)>0||dcmp(d+min(r,R),max(r,R))<0) return ret;

db x = (d\*d-R\*R+r\*r)/(2\*d);

db y = sqrt(r\*r-x\*x);

P v = (b-a)/d;

ret.push\_back(a+v\*x + RotateL90(v)\*y);

if (sign(y)>0) ret.push\_back(a+v\*x - RotateL90(v)\*y);

return ret;

}

P RotateL(const P&a,const double&rad){//逆时针旋转

return P(a.x\*cos(rad)-a.y\*sin(rad),a.y\*cos(rad)+a.x\*sin(rad));

}

//点p1到圆c的两条切线对应的切点

int getPCir\_tangents(P c,db r,P p1,vector<P>&ans){

P u=c-p1;

db dist=diss(p1,c);

if(dcmp(dist,r)<0)return 0;//内含

else if(dcmp(dist,r)==0){//1条

P x=RotateL(u,PI/2);

ans.pb(p1);

ans.pb(p1);

return 1;

}else{//2条切线

db ang=asin(r/dist);

u=u\*sqrt(dis(p1,c)-r\*r)/dist;

P x1=RotateL(u,ang),x2=RotateL(u,-ang);

ans.pb(x1+p1);

ans.pb(x2+p1);

return 2;

}

}

//4相离 3外切 2两点交 1内切 0内含

int relationCir(Cir x,Cir y){

db d=sqrt(dis(x.c,y.c));

int f1=sign(d-x.r-y.r);

if(f1>0)return 4;

if(f1==0)return 3;

db l=fabs(x.r-y.r);

int f2=sign(d-l);

if(f1<0&&f2>0)return 2;

if(f2==0)return 1;

if(f2<0)return 0;

}

// 圆的公切线条数 a,b分别表示第i条切线在圆c1,c2上切点

// 重合-无数条 内含-无 内切-一条外公切线 相交-两条外公切线

// 外切 3条公切线 一条内公切线,两条外公切线

// 相离 4条公切线 内公切线2条 外公切线2条

int getPCir\_tangents(Cir c1,Cir c2,P\*a,P\*b){//点集在[0,cnt-1]上

int cnt=0;

if(dcmp(c1.r,c2.r)<0)swap(c1,c2),swap(a,b);//对应点集仍不变

int f=relationCir(c1,c2);

if(!f)return 0;//内含

P p1=c1.c,p2=c2.c;

db r1=c1.r,r2=c2.r;

if(dcmp(dis(p1,p2),0)==0&&dcmp(r1,r2)==0)return -1;//无数条切线

db base=atan2(p2.y-p1.y,p2.x-p1.x);

if(f==1){//内切一条

a[cnt]=c1.GP(base);

b[cnt]=c2.GP(base);cnt++;

return 1;

}

//2条外公切线

db ang=acos((r1-r2)/diss(p1,p2));

a[cnt]=c1.GP(base+ang);b[cnt]=c2.GP(base+ang);cnt++;

a[cnt]=c1.GP(base-ang);b[cnt]=c2.GP(base-ang);cnt++;

if(f==3){//一条内

a[cnt]=c1.GP(base);b[cnt]=c2.GP(PI+base);cnt++;

}else if(f==4){//两条内

db ang=acos((r1+r2)/diss(p1,p2));

a[cnt]=c1.GP(base+ang);b[cnt]=c2.GP(PI+base+ang);cnt++;

a[cnt]=c1.GP(base-ang);b[cnt]=c2.GP(PI+base-ang);cnt++;

}

return cnt;

}

db Cirarea(P a,db r1,P b,db r2){//两圆面积交 很可能要开long double 精度不够

db d = sqrt((a.x-b.x)\*(a.x-b.x) + (a.y-b.y)\*(a.y-b.y));

if (dcmp(d,r1+r2)>=0)return 0;//相离相切

if (dcmp(r1,r2)>0)swap(r1,r2);

if(dcmp(r2-r1,d)>=0)//内含

return PI\*r1\*r1;

db ang1=acos((r1\*r1+d\*d-r2\*r2)/(2\*r1\*d));

db ang2=acos((r2\*r2+d\*d-r1\*r1)/(2\*r2\*d));

return ang1\*r1\*r1 + ang2\*r2\*r2 - r1\*d\*sin(ang1);

}

//判某条线段能否被若干圆完全覆盖

bool direct(L x){

vector<pair<P,int>>sec;

P a=x.ps[0],b=x.ps[1];

if(a<b)swap(a,b);

sec.push\_back({a,-2});

sec.push\_back({b,2});

for(auto&u:cir){

auto tmpjiao=getCircleLineSec(u.c,u.r,x.ps[0],x.ps[1]);

if(tmpjiao.size()==2){

if(tmpjiao[0]<tmpjiao[1])swap(tmpjiao[0],tmpjiao[1]);

sec.push\_back({tmpjiao[0],0});

sec.push\_back({tmpjiao[1],1});

}

}

sort(sec.begin(),sec.end());

bool ret = true, counting = false;

int inside = 0;

for(int i=0; i < sec.size(); i++) {

if(abs(sec[i].se) == 2) counting = !counting;

else sec[i].se ? inside-- : inside++;

if(counting and !inside) ret = false;

}

return ret;

}

//求多个圆与线段的交的长度

using PDD = pair<double, double>;

#define sqr(x) ((x) \* (x))

db coverLen(L a, Cir c[], int n) {//[1,n]是圆

db Cos = ((a.ps[1].x - a.ps[0].x) / a.len()), Sin = ((a.ps[1].y - a.ps[0].y) / a.dir().len());

db dx = a.ps[0].x, dy = a.ps[0].y;

vector<PDD> mat;

for (int i = 1; i <= n; i++) {

db A = 1;

db B = -2 \* (Cos \* (c[i].c.x - dx) + Sin \* (c[i].c.y - dy));

db C = pow2(c[i].c.x - dx) + pow2(c[i].c.y - dy) - c[i].r \* c[i].r;

db D = B \* B - 4 \* A \* C;

if (sign(D) < 0) continue;

D = sign(D) ? sqrt(D) : 0; // 清误差

db L = max(0.0, min(a.dir().len(), (-B - D) / (2 \* A)));

db R = max(0.0, min(a.dir().len(), (-B + D) / (2 \* A)));

mat.eb(mp(L, R));

}

sort(all(mat));

db las = 0, tot = 0;

for (int i = 0; i < mat.size(); i++) {

if (dcmp(las,mat[i].fi)<0) las = mat[i].fi;

if (dcmp(las,mat[i].se)<0) tot += mat[i].se - las, las = mat[i].se;

}

return tot;

}

db TriCirInterction(Cir c,P a,P b){ //三角剖分求圆与多边形面积交

P oa=a-c.c,ob=b-c.c;//逆时针

P ba=a-b,bc=c.c-b;

P ab=b-a,ac=c.c-a;

db doa=oa.len(),dob=ob.len(),dab=ab.len(),r=c.r;

if(dcmp(det(oa,ob),0)==0)return 0;//三点一线 无三角形

if(dcmp(doa,r)<0&&dcmp(dob,r)<0)return 0.5\*det(oa,ob);//三点内部

else if(dcmp(dob,r)<0&&dcmp(doa,r)>=0){

db x=(dot(ba,bc)+sqrt(r\*r\*dab\*dab-det(ba,bc)\*det(ba,bc)))/dab;

db TS=det(oa,ob)\*0.5;

return asin(TS\*(1-x/dab)\*2/r/doa)\*r\*r\*0.5+TS\*x/dab;

}else if(dcmp(dob,r)>=0&&dcmp(doa,r)<0){

db y=(dot(ab,ac)+sqrt(r\*r\*dab\*dab-det(ab,ac)\*det(ab,ac)))/dab;

db TS=det(oa,ob)\*0.5;

return asin(TS\*(1-y/dab)\*2/r/dob)\*r\*r\*0.5+TS\*y/dab;

}else if(fabs(det(oa,ob))>=r\*dab||dcmp(dot(ab,ac),0)<=0||(dcmp(dot(ba,bc),0)<=0)){//两点外 单扇形

db ang=acos(dot(oa,ob)/doa/dob);

return sign(det(oa,ob))\*ang\*r\*r\*0.5;

}else{

db x=(dot(ba,bc)+sqrt(r\*r\*dab\*dab-det(ba,bc)\*det(ba,bc)))/dab;

db y=(dot(ab,ac)+sqrt(r\*r\*dab\*dab-det(ab,ac)\*det(ab,ac)))/dab;

db TS=det(oa,ob)\*0.5;

return (asin(TS\*(1-x/dab)\*2/r/doa)+asin(TS\*(1-y/dab)\*2/r/dob))\*r\*r\*0.5+TS\*((x+y)/dab-1);

}

}

int n;

db PolyonCirArea(Cir c,vector<P>&p){ //圆与多边形面积交

db ans=0;

for(int i=0;i<n;++i){

ans+=TriCirInterction(c,p[i],p[i+1]);

}

return fabs(ans);

}

vector<P>p(100005);

int main(){

db x,y,h,x1,y1,t,r,ans=0;

cin>>n>>r;

Cir c;

c.c=P(0,0);c.r=r;

for(int i=0;i<n;++i)cin>>p[i].x>>p[i].y;

p[n]=p[0];

printf("%.12f\n",PolyonCirArea(c,p));

return 0;

}

## 极角排序

bool cmp1(P a,P b){

if(dcmp(atan2(a.y,a.x),atan2(b.y,b.x))!=0)

return dcmp(atan2(a.y,a.x),atan2(b.y,b.x))<0;

else return dcmp(a.x,b.x)<0;

}

//极角排序，按某点排序

int Qua(P a){

if(a.x>0&&a.y>=0)return 1;

if(a.x<=0&&a.y>0)return 2;

if(a.x<0&&a.y<=0)return 3;

if(a.x>=0&&a.y<0)return 4;

}

bool cmp2(P a,P b){ //一般是对a-c,b-c排

int f1=Qua(a),f2=Qua(b);

if(f1==f2){

ll tmp=det(a,b);

if(tmp>0)return 1;

else if(!tmp&&a.len2()<b.len2())return 1;

return 0;

}

return f1<f2;

}

## 凸包

int Andrew(P p[],P ch[]){//求凸包

sort(p+1,p+n+1);//重载的排序

int m = 0;

for(int i = 1; i <=n; i++){

while(m > 1 && cha(ch[m-2],ch[m-1],p[i]) <= 0) m--;//<=不需要共线

ch[m++] = p[i];

}

int k = m;

for(int i = n-1; i >= 1; i--){

while(m > k && cha(ch[m-2],ch[m-1],p[i]) <= 0) m--;

ch[m++] = p[i];

}

if(n > 1) m--;

return m;

}

## 旋转卡壳

int rotate\_calipers(){//旋转卡壳凸包直径平方 直径相应改db和cha

Andrew();

int j=1,ans=0;//j上一次做完的点

for(int i=0;i<m;++i){

while(cha(ch[i],ch[i+1],ch[j])<cha(ch[i],ch[i+1],ch[j+1]))j=(j+1)%m;

ans=max(ans,max(dis(ch[i],ch[j]),dis(ch[i+1],ch[j])));

}

return ans;

}

## 半平面交

逆时针向量，极角排序，双端队列维护半平面，弹出所有在当前右侧的点，最后判头尾合法

//求多边形核的面积，用到点类，线类模板

bool cmp(L l1,L l2){

P a=l1.dir(),b=l2.dir();

int f1=Qua(a),f2=Qua(b);

if(f1==f2){

db tmp=det(a,b);

return tmp>=0;

}

return f1<f2;

}

db area(vector<P>p){//任意多边形面积 顺逆都可

db ans=0;

int sz=p.size();

if(!sz)return ans;

for(int i=0;i<sz-1;++i)ans+=det(p[i],p[i+1]);

ans+=det(p[sz-1],p[0]);

return fabs(ans/2);

}

P LineInter(L l1,L l2){ //两直线交点 不平行的时候用

P a=l1.ps[0],b=l1.ps[1],c=l2.ps[0],d=l2.ps[1];

db t=det(a-c,c-d)/det(a-b,c-d);

return P(a.x+(b.x-a.x)\*t,a.y+(b.y-a.y)\*t);

}

bool onRight(P a,L x){

return sign(det(x.dir(),a-x.ps[0]))<=0;

}

int HalfPlaneInter(vector<L>line,vector<P>&sol){

int n=line.size(),h,t;

sort(line.begin(),line.end(),cmp);

q[h=t=0]=line[0];

for(int i=1;i<=n-1;++i){

while(h<t&&onRight(p[t-1],line[i]))t--;

while(h<t&&onRight(p[h],line[i]))h++;

q[++t]=line[i];

if(sign(det(q[t].dir(),q[t-1].dir()))==0){

t--;//共线保留靠左

if(!onRight(line[i].ps[0],q[t]))q[t]=line[i];

}

if(h<t)p[t-1]=LineInter(q[t],q[t-1]);

}

while(h<t&&onRight(p[t-1],q[h]))t--;

if(t-h<=1)return 0;

sol.clear();p[t]=LineInter(q[h],q[t]);

for(int i=h;i<=t;++i)sol.eb(p[i]);

return 1;

}

vector<P>a,hpi;//多边形、半平面

vector<L>line;

int main(){

int t,n;

cin>>t;

while(t--){

cin>>n;

a=vector<P>(n+1);

for(int i=0;i<n;++i)cin>>a[i].x>>a[i].y;

a[n]=a[0];

for(int i=0;i<n;++i)line.eb(L(a[i],a[i+1]));

}

HalfPlaneInter(line,hpi);

printf("%.3lf",area(hpi));

return 0;

}

## 三维几何

const db EPS=1e-8;

typedef struct P3D {

db x, y, z;

P3D(db xx = 0, db yy = 0, db zz = 0): x(xx), y(yy), z(zz) {}

bool operator == (const P3D& A) const {

return x==A.x && y==A.y && z==A.z;

}

}V3D;

P3D read\_Point\_3D() {

db x,y,z;

scanf("%lf%lf%lf",&x,&y,&z);

return P3D(x,y,z);

}

V3D operator + (const V3D & A, const V3D & B) {

return V3D(A.x + B.x, A.y + B.y, A.z + B.z);

}

V3D operator - (const P3D & A, const P3D & B) {

return V3D(A.x - B.x, A.y - B.y, A.z - B.z);

}

V3D operator \* (const V3D & A, db p) {

return V3D(A.x \* p, A.y \* p, A.z \* p);

}

V3D operator / (const V3D & A, db p) {

return V3D(A.x / p, A.y / p, A.z / p);

}

db pow2(db a){

return a\*a;

}

db dis(const P3D&A,const P3D&B){

return sqrt(pow2(A.x-B.x)+pow2(A.y-B.y)+pow2(A.z-B.z));

}

db Dot(const V3D & A, const V3D & B) {

return A.x \* B.x + A.y \* B.y + A.z \* B.z;

}

db Length(const V3D & A) {

return sqrt(Dot(A, A));

}

db Angle(const V3D & A, const V3D & B) {

return acos(Dot(A, B) / Length(A) / Length(B));

}

V3D Cross(const V3D & A, const V3D & B) {

return V3D(A.y \* B.z - A.z \* B.y, A.z \* B.x - A.x \* B.z, A.x \* B.y - A.y \* B.x);

}

db Area2(const P3D & A, const P3D & B, const P3D & C) {

return Length(Cross(B - A, C - A));

}

db Volume6(const P3D & A, const P3D & B, const P3D & C, const P3D & D) {

return Dot(D - A, Cross(B - A, C - A));

}

// 四面体的重心

P3D Centroid(const P3D & A, const P3D & B, const P3D & C, const P3D & D) {

return (A + B + C + D) / 4.0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*点线面\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 点p到平面p0-n的距离。n必须为单位向量

db DistanceToPlane(const P3D & p, const P3D & p0, const V3D & n){

return fabs(Dot(p - p0, n)); // 如果不取绝对值，得到的是有向距离

}

// 点p在平面p0-n上的投影。n必须为单位向量

P3D GetPlaneProjection(const P3D & p, const P3D & p0, const V3D & n){

return p - n \* Dot(p - p0, n);

}

//直线p1-p2 与平面p0-n的交点

P3D LinePlaneIntersection(P3D p1, P3D p2, P3D p0, V3D n){

V3D v= p2 - p1;

db t = (Dot(n, p0 - p1) / Dot(n, p2 - p1)); //分母为0，直线与平面平行或在平面上

return p1 + v \* t; //如果是线段 判断t是否在0~1之间

}

// 点P到直线AB的距离

db DistanceToLine(const P3D & P, const P3D & A, const P3D & B){

V3D v1 = B - A, v2 = P - A;

return Length(Cross(v1, v2)) / Length(v1);

}

//点到线段的距离

db DistanceToSeg(P3D p, P3D a, P3D b){

if(a == b){

return Length(p - a);

}

V3D v1 = b - a, v2 = p - a, v3 = p - b;

if(Dot(v1, v2) + EPS < 0){

return Length(v2);

}

else{

if(Dot(v1, v3) - EPS > 0){

return Length(v3);

}

else{

return Length(Cross(v1, v2)) / Length(v1);

}

}

}

//求异面直线 p1+s\*u与p2+t\*v的公垂线对应的s 如果平行|重合，返回false

bool LineDistance3D(P3D p1, V3D u, P3D p2, V3D v, db & s){

db b = Dot(u, u) \* Dot(v, v) - Dot(u, v) \* Dot(u, v);

if(abs(b) <= EPS){return false;}

db a = Dot(u, v) \* Dot(v, p1 - p2) - Dot(v, v) \* Dot(u, p1 - p2);

s = a / b;

return true;

}

// p1和p2是否在线段a-b的同侧

bool SameSide(const P3D & p1, const P3D & p2, const P3D & a, const P3D & b){

return Dot(Cross(b - a, p1 - a), Cross(b - a, p2 - a)) - EPS >= 0;

}

// 点P在三角形P0, P1, P2中

bool PointInTri(const P3D & P, const P3D & P0, const P3D & P1, const P3D & P2){

return SameSide(P, P0, P1, P2) && SameSide(P, P1, P0, P2) && SameSide(P, P2, P0, P1);

}

// 三角形P0P1P2是否和线段AB相交

bool TriSegIntersection(const P3D & P0, const P3D & P1, const P3D & P2, const P3D & A, const P3D & B, P3D & P){

V3D n = Cross(P1 - P0, P2 - P0);

if(abs(Dot(n, B - A)) <= EPS){

return false; // 线段A-B和平面P0P1P2平行或共面

}

else{ // 平面A和直线P1-P2有惟一交点

db t = Dot(n, P0 - A) / Dot(n, B - A);

if(t + EPS < 0 || t - 1 - EPS > 0){

return false; // 不在线段AB上

}

P = A + (B - A) \* t; // 交点

return PointInTri(P, P0, P1, P2);

}

}

//三维旋转 old绕vx逆时针旋转theta

P get(double oldx, double oldy, double oldz, double vx, double vy, double vz, double theta){

double r = theta \* PI / 180;

double c = cos(r);

double s = sin(r);

double nx = (vx\*vx\*(1 - c) + c) \* oldx + (vx\*vy\*(1 - c) - vz\*s) \* oldy + (vx\*vz\*(1 - c) + vy\*s) \* oldz;

double ny = (vy\*vx\*(1 - c) + vz\*s) \* oldx + (vy\*vy\*(1 - c) + c) \* oldy + (vy\*vz\*(1 - c) - vx\*s) \* oldz;

double nz = (vx\*vz\*(1 - c) - vy\*s) \* oldx + (vy\*vz\*(1 - c) + vx\*s) \* oldy + (vz\*vz\*(1 - c) + c) \* oldz;

return P(nx, ny, nz);

}

//空间两三角形是否相交

bool TriTriIntersection(P3D \* T1, P3D \* T2){

P3D P;

for(int i = 0; i < 3; i++){

if(TriSegIntersection(T1[0], T1[1], T1[2], T2[i], T2[(i + 1) % 3], P)){return true;}

if(TriSegIntersection(T2[0], T2[1], T2[2], T1[i], T1[(i + 1) % 3], P)){return true;}

}

return false;

}

//空间两直线上最近点对 返回最近距离 点对保存在ans1 ans2中

db SegSegDistance(P3D a1, P3D b1, P3D a2, P3D b2, P3D& ans1, P3D& ans2){

V3D v1 = (a1 - b1), v2 = (a2 - b2);

V3D N = Cross(v1, v2);

V3D ab = (a1 - a2);

db ans = Dot(N, ab) / Length(N);

P3D p1 = a1, p2 = a2;

V3D d1 = b1 - a1, d2 = b2 - a2;

db t1, t2;

t1 = Dot((Cross(p2 - p1, d2)), Cross(d1, d2));

t2 = Dot((Cross(p2 - p1, d1)), Cross(d1, d2));

db dd = Length((Cross(d1, d2)));

t1 /= dd \* dd;

t2 /= dd \* dd;

ans1 = (a1 + (b1 - a1) \* t1);

ans2 = (a2 + (b2 - a2) \* t2);

return fabs(ans);

}

// 判断P是否在三角形A, B, C中，并且到三条边的距离都至少为mindist。保证P, A, B, C共面

bool InsideWithMinDistance(const P3D & P, const P3D & A, const P3D & B, const P3D & C, db mindist){

if(!PointInTri(P, A, B, C))return 0;

if(DistanceToLine(P, A, B) < mindist)return 0;

if(DistanceToLine(P, B, C) < mindist)return 0;

if(DistanceToLine(P, C, A) < mindist)return 0;

return 1;

}

// 判断P是否在凸四边形ABCD（顺时针或逆时针）中，并且到四条边的距离都至少为mindist。保证P, A, B, C, D共面

bool InsideWithMinDistance(const P3D & P, const P3D & A, const P3D & B, const P3D & C, const P3D & D, db mindist){

if(!PointInTri(P, A, B, C))return 0;

if(!PointInTri(P, C, D, A))return 0;

if(DistanceToLine(P, A, B) < mindist)return 0;

if(DistanceToLine(P, B, C) < mindist)return 0;

if(DistanceToLine(P, C, D) < mindist)return 0;

if(DistanceToLine(P, D, A) < mindist)return 0;

return 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*凸包相关问题\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//加干扰

db rand01(){

return rand() / (db)RAND\_MAX;

}

db randeps(){

return (rand01() - 0.5) \* EPS;

}

P3D add\_noise(const P3D & p){

return P3D(p.x + randeps(), p.y + randeps(), p.z + randeps());

}

struct Face{

int v[3];//顶点下标

Face(int a, int b, int c){ v[0]=a;v[1]=b;v[2]=c;}

V3D Normal(const vector<P3D> & P) const{//面的法线

return Cross(P[v[1]] - P[v[0]], P[v[2]] - P[v[0]]);

}

// P[i]是否能看到面

int CanSee(const vector<P3D> & P, int i) const{

return Dot(P[i] - P[v[0]], Normal(P)) > 0;

}

};

// 增量法求三维凸包

// 注意：没有考虑各种特殊情况（如四点共面）。实践中，请在调用前对输入点进行微小扰动

vector<Face> CH3D(const vector<P3D> & P){

int n = P.size();

vector<vector<int> > vis(n);

for(int i = 0; i < n; i++){

vis[i].resize(n);

}

vector<Face> cur;

cur.push\_back(Face(0, 1, 2)); // 由于已经进行扰动，前三个点不共线

cur.push\_back(Face(2, 1, 0));

for(int i = 3; i < n; i++){

vector<Face> next;

// 计算每条边的“左面”的可见性

for(int j = 0; j < cur.size(); j++) {

Face & f = cur[j];

int res = f.CanSee(P, i);

if(!res){

next.push\_back(f);

}

for(int k = 0; k < 3; k++){

vis[f.v[k]][f.v[(k + 1) % 3]] = res;

}

}

for(int j = 0; j < cur.size(); j++)

for(int k = 0; k < 3; k++){

int a = cur[j].v[k], b = cur[j].v[(k + 1) % 3];

if(vis[a][b] != vis[b][a] && vis[a][b]){ // (a,b)是分界线，左边对P[i]可见

next.push\_back(Face(a, b, i));

}

}

cur = next;

}

return cur;

}

struct ConvexPolyhedron{

int n;

vector<P3D> P, P2;

vector<Face> faces;

bool read(){

if(scanf("%d", &n) != 1){

return false;

}

P.resize(n);

P2.resize(n);

for(int i = 0; i < n; i++){

P[i] = read\_Point\_3D();

P2[i] = add\_noise(P[i]);

}

faces = CH3D(P2);

return true;

}

//三维凸包重心

P3D centroid(){

P3D C = P[0];

db totv = 0;

P3D tot(0, 0, 0);

for(int i = 0; i < faces.size(); i++){

P3D p1 = P[faces[i].v[0]], p2 = P[faces[i].v[1]], p3 = P[faces[i].v[2]];

db v = -Volume6(p1, p2, p3, C);

totv += v;

tot = tot + Centroid(p1, p2, p3, C) \* v;

}

return tot / totv;

}

//凸包重心到表面最近距离

db mindist(P3D C){

db ans = 1e30;

for(int i = 0; i < faces.size(); i++){

P3D p1 = P[faces[i].v[0]], p2 = P[faces[i].v[1]], p3 = P[faces[i].v[2]];

ans = min(ans, fabs(-Volume6(p1, p2, p3, C) / Area2(p1, p2, p3)));

}

return ans;

}

};

//两球体积交

### 球的体积交与并

db cap(db r,db h){return pi\*(r\*3-h)\*h\*h/3;}

//2球体积交

db sphere\_intersect(db x1,db y1,db z1,db r1,db x2,db y2,db z2,db r2){

db d=dis(x1,y1,z1,x2,y2,z2);

//相离

if(d>=pow2(r1+r2))return 0;

//包含

if(d<=pow2(r1-r2))return pow3(min(r1,r2))\*4\*pi/3;

//相交

db h1=r1-r1\*cos(r2,r1,sqrt(d)),h2=r2-r2\*cos(r1,r2,sqrt(d));

return cap(r1,h1)+cap(r2,h2);

}

//2球体积并

db sphere\_union(db x1,db y1,db z1,db r1,db x2,db y2,db z2,db r2){

db d=dis(x1,y1,z1,x2,y2,z2);

//相离

if(d>=pow2(r1+r2))return (pow3(r1)+pow3(r2))\*4\*pi/3;

//包含

if(d<=pow2(r1-r2))return pow3(max(r1,r2))\*4\*pi/3;

//相交

db h1=r1+r1\*cos(r2,r1,sqrt(d)),h2=r2+r2\*cos(r1,r2,sqrt(d));

return cap(r1,h1)+cap(r2,h2);

}

## 其他题

### Hdu6219 最大空凸包

//O(n^3)最大空凸包面积模板

#include<bits/stdc++.h>

#define cha(p1,p2,p3) (sign(det(p2-p1,p3-p1)))//p2-p1叉p3-p1

#define cha1(p1,p2,p3) det(p2-p1,p3-p1)

using namespace std;

typedef double db;

const int maxn=55;

const db PI=acos(-1.0);

const db eps=1e-8;

struct P{

db x,y;

P():x(0),y(0){}

P(db x,db y):x(x),y(y){}

P operator-(P p){ return P(x-p.x,y-p.y);}

}a[maxn],p[maxn],O;

int T,n,cnt;

db ans,dp[maxn][maxn];//以三角形Oij为凸包最后一块三角形的面积

int sign(db a){ //符号正负

return a<-eps?-1:a>eps;

}

int dcmp(db a,db b){ //两数大小

return sign(a-b);

}

db det(P a,P b){//叉积

return a.x\*b.y-a.y\*b.x;

}

db dis(P p1,P p2){

return (p2.x-p1.x)\*(p2.x-p1.x)+(p2.y-p1.y)\*(p2.y-p1.y);

}

//极角排序函数，角度相同则距离小的在前面

bool cmp(P p1,P p2){

db tmp=cha(O,p1,p2);

if(tmp>0) return 1;

else if(tmp==0&&dis(O,p1)<dis(O,p2)) return 1;

return 0;

}

void solve(){

for(int i=0;i<n;++i)

for(int j=0;j<n;++j)

dp[i][j]=0.0;

for(int i=1;i<=cnt;++i){

int j=i-1;

while(j&&!cha(O,p[i],p[j])) --j;//找到第一个不共线的

int flag=(j==i-1);//oi上有点

while(j){

int k=j-1;

while(k&&cha(p[i],p[j],p[k])>0) --k;//

db area=fabs(cha1(O,p[i],p[j]))/2.0;

if(k) area+=dp[j][k];

if(flag) dp[i][j]=area;//Oi上没点才能更新dp,否则下个点回算进内点

ans=max(ans,area);

j=k;

}

if(flag) for(int j=1;j<i;++j) dp[i][j]=max(dp[i][j],dp[i][j-1]);//前缀最大值优化

}

}

int main(){

scanf("%d",&T);

while(T--){

scanf("%d",&n);

ans=0.0;

for(int i=0;i<n;++i)

scanf("%lf%lf",&a[i].x,&a[i].y);

for(int i=0;i<n;++i){

O=a[i];

cnt=0;

for(int j=0;j<n;++j)

if(dcmp(a[j].y,a[i].y)>0||dcmp(a[j].y,a[i].y)==0&&dcmp(a[j].x,a[i].x)>0) p[++cnt]=a[j];

sort(p+1,p+cnt+1,cmp);

solve();

}

printf("%.1lf\n",ans);

}

return 0;

}

### 平面最近点对

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

struct P{int x,y;}a[2000000],tmp[2000000];

int n,b[2000000];

const ll inf=5e18;

ll sqr(int x){return (ll)x\*x;}

ll dis(int l,int r){return sqr(a[l].x-a[r].x)+sqr(a[l].y-a[r].y);}

int cmp(const P &a,const P &b){return a.x==b.x?a.y<b.y:a.x<b.x;}

void merge(int l,int r){

int mid=(l+r)>>1,i=l,j=mid+1;//归并

for(int k=l;k<=r;k++){

if(i<=mid&&(j>r||a[i].y<a[j].y))tmp[k]=a[i++];

else tmp[k]=a[j++];

}

for(int i=l;i<=r;i++)a[i]=tmp[i];

}

ll solve(int l,int r){

if(l>=r)return inf;

if(l+1==r){if(a[l].y>a[r].y)swap(a[l],a[r]);return dis(l,r);}

int mid=(l+r)>>1,t=a[mid].x,cnt=0;//重新排序后中位数就乱了，需要记下来

ll d=min(solve(l,mid),solve(mid+1,r));

merge(l,r);

for(int i=l;i<=r;i++)

if(sqr(a[i].x-t)<d)//两边平方的技巧

b[++cnt]=i;//区间内的拉出来处理

for(int i=1;i<=cnt;i++)

for(int j=i+1;j<=cnt&&sqr(a[b[j]].y-a[b[i]].y)<d;j++)

d=min(d,dis(b[j],b[i]));

return d;

}

int main(){

scanf("%d",&n);

for(int i=1;i<=n;i++)scanf("%d%d",&a[i].x,&a[i].y);

sort(a+1,a+n+1,cmp);

printf("%.4lf",sqrt(solve(1,n)));

}

# 杂项

## 快速读入

inline int read(){

char c = getchar();int x = 0,s = 1;

while(c < '0' || c > '9') {if(c == '-') s = -1;c = getchar();}//是符号

while(c >= '0' && c <= '9') {x = x\*10 + c -'0';c = getchar();}//是数字

return x\*s;

}

inline void in(int&x){

x=read();

}

## \_int128

#include <bits/stdc++.h>

using namespace std;

inline \_\_int128 read(){

\_\_int128 x=0,f=1;

char ch=getchar();

while(ch<'0'||ch>'9'){

if(ch=='-')

f=-1;

ch=getchar();

}

while(ch>='0'&&ch<='9'){

x=x\*10+ch-'0';

ch=getchar();

}

return x\*f;

}

inline void print(\_\_int128 x){

if(x<0){

putchar('-');

x=-x;

}

if(x>9)

print(x/10);

putchar(x%10+'0');

}

int main(void){

\_\_int128 a = read();

\_\_int128 b = read();

\_\_int128 m =read();

print((a + b)%m);

cout<<endl;

return 0;

}

## 常用STL

map的Upperbound

map<int,int>::iterator se = mp.upper\_bound(mid);

返回迭代器

**优先队列**

priority\_queue<int>Q;//采用默认优先级构造队列

priority\_queue<int,vector<int>,cmp2>que2;//最大值优先

Q.push(x);

int x = Q.top(); Q.pop();

**multiset**

begin() 返回指向第一个元素的迭代器

clear() 清除所有元素

count() 返回某个值元素的个数

empty() 如果集合为空，返回 true

end() 返回指向最后一个元素的迭代器

erase() 删除集合中的元素 ( 参数是一个元素值，或者迭代器)

find() 返回一个指向被查找到元素的迭代器

insert() 在集合中插入元素

size() 集合中元素的数目

lower\_bound() 返回指向大于（或等于）某值的第一个元素的迭代器

upper\_bound() 返回大于某个值元素的迭代器

equal\_range() 返回集合中与给定值相等的上下限的两个迭代器

multiset <point> po;

multiset <point>::iterator L, R, it;

离散化lower\_lound

lower\_bound 第一个出现的值大于等于val的位置

upper\_bound 最后一个大于等于val的位置，也是有一个新元素val进来时的插入位置

sort(a + 1, a + A+1);

A = unique(a + 1, a + A+1) - (a + 1);

for (int i = 1; i <= n; i++){ // segtree

int L = lower\_bound(a+1, a+A+1, l[i]) - a;

int R = lower\_bound(a+1, a+A+1, r[i]) - a;

T.update(L, R, i, 1, T.M+1,1);

}

------------use in ChairTree-------------------

for (int i = 1; i <= n; i++) {

scanf("%d", &arr[i]);

Rank[i] = arr[i];

}

sort(Rank + 1, Rank + n+1);//Rank存储原值

int m = unique(Rank + 1, Rank + n +1) - (Rank + 1);//这个m很重要，WA一天系列

for (int i = 1; i <= n; i++) {//离散化后的数组，仅仅用来更新

arr[i] = lower\_bound(Rank + 1, Rank + m+1, arr[i]) - Rank;

}

**vector 的插入删除**

// erase the 6th element

myvector.erase (myvector.begin()+5);

// erase the first 3 elements:

myvector.erase (myvector.begin(),myvector.begin()+3);

// ------------inserting into a vector-----------

#include <iostream>

#include <vector>

int main (){

std::vector<int> myvector (3,100);

std::vector<int>::iterator it;

it = myvector.begin();

it = myvector.insert ( it , 200 );

myvector.insert (it,2,300);

// "it" no longer valid, get a new one:

it = myvector.begin();

std::vector<int> anothervector (2,400);

myvector.insert (it+2,anothervector.begin(),anothervector.end());

int myarray [] = { 501,502,503 };

myvector.insert (myvector.begin(), myarray, myarray+3);

std::cout << "myvector contains:";

for (it=myvector.begin(); it<myvector.end(); it++) std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

**bitset**

构造函数

bitset<n> b;

b有n位，每位都为0.参数n可以为一个表达式.

如bitset<5> b0;则"b0"为"00000";

bitset<n> b(unsigned long u);

b有n位,并用u赋值;如果u超过n位,则顶端被截除

如:bitset<5>b0(5);则"b0"为"00101";

bitset<n> b(string s);

b是string对象s中含有的位串的副本

string bitval ( "10011" );

bitset<5> b0 ( bitval4 );

则"b0"为"10011";

bitset<n> b(s, pos);

b是s中从位置pos开始位的副本,前面的多余位自动填充0;

string bitval ("01011010");

bitset<10> b0 ( bitval5, 3 );

则"b0" 为 "0000011010";

bitset<n> b(s, pos, num);

b是s中从位置pos开始的num个位的副本,如果num<n,则前面的空位自动填充0;

string bitval ("11110011011");

bitset<6> b0 ( bitval5, 3, 6 );

则"b0" 为 "100110";

bool any() 是否存在置为1的二进制位？和none()相反

bool none() 是否不存在置为1的二进制位,即全部为0？和any()相反.

size\_t count() 二进制位为1的个数.

size\_t size() 二进制位的个数

flip() 把所有二进制位逐位取反

flip(size\_t pos)把在pos处的二进制位取反

bool operator[]( size\_type \_Pos )获取在pos处的二进制位

set() 把所有二进制位都置为1

set(pos) 把在pos处的二进制位置为1

reset() 把所有二进制位都置为0

reset(pos) 把在pos处的二进制位置为0

test(size\_t pos)在pos处的二进制位是否为1？

unsigned long to\_ulong( )用同样的二进制位返回一个unsigned long值

string to\_string ()返回对应的字符串.