目录

Part I
基础模版

1 常用模板

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
inline ll rd(){
    ll x=0;char o,f=1;
    while(o=getchar(),o<48)if(o==45)f=-f;
    do x=(x<<3)+(x<<1)+(o^48);
    while(o=getchar(),o>47);
    return x*f;
}
const int maxn=1e3+5;
ll ksm(ll a,ll b){ll ans=1;a%=mod;while(b){if(b&1)ans=(ans*a)%mod;a=(
   a*a)%mod;b>>=1;}return ans;}
ll gcd(ll a, ll b){ll t;while(b){t=b;b=a%b;a=t;}return a;}
int main() {

    return 0;
}
```

2 存图（链式前向星）

```
const int maxn=1e3+5;
int head[maxn],tot;
struct nn{int v,nxt;}g[maxn<<1];
void add_edge(int u,int v){g[++tot]={v,head[u]};head[u]=tot;}
```

3 fread 快读

使用时讲所有需要读入的 int/long long 变量均替换成 io»x; 的形式即可

```
struct ios {
    inline char read(){
        static const int IN_LEN=1<<18|1;
        static char buf[IN_LEN],*s,*t;
        return (s==t)&&(t=(s=buf)+fread(buf,1,IN_LEN,stdin)),s==t
           ?-1:*s++;
    }

    template <typename _Tp> inline ios & operator >> (_Tp&x){
        static char c11,boo;
        for(c11=read(),boo=0;!isdigit(c11);c11=read()){
            if(c11==-1)return *this;
            boo|=c11=='-';
        }
        for(x=0;isdigit(c11);c11=read())x=x*10+(c11^'0');
        boo&&(x=-x);
```

```
        return *this;
    }
} io;
```

4 int128

```
void scan(__int128 &x){
        x=0;int f=1;char ch;
        if ((ch=getchar())=='-')f=-f;
        else x=(x<<3)+(x<<1)+ch-'0';
        while ((ch=getchar())>='0'&&ch<='9')x=(x<<3)+(x<<1)+ch-'0';
        x*=f;
}
void print(__int128 x){
        if (x<0)x=-x,putchar('-');
        if (x>9)print(x/10);
        putchar(x%10+'0');
}
```

Part II
数据结构

5 ST 表

```
//静态区间求最值
const int maxn=1e6+5;
int n,a[maxn];
int Max[22][maxn],lg[maxn],Min[22][maxn];
void init_ST(){
    for (int i=2;i<=n;i++)lg[i]=lg[i>>1]+1;
    for (int i=1;i<=n;i++)
        Max[0][i]=a[i];
    for (int j=1;j<=21;j++)
        for (int i=1;i+(1<<j)-1<=n;i++)
            Max[j][i]=max(Max[j-1][i],Max[j-1][i+(1<<(j-1))]);
    for (int i=1;i<=n;i++)
        Min[0][i]=a[i];
    for (int j=1;j<=21;j++)
        for (int i=1;i+(1<<j)-1<=n;i++)
            Min[j][i]=min(Min[j-1][i],Min[j-1][i+(1<<(j-1))]);
}
int querymx(int l,int r){
    if (l>r)return -1e9;//注意考虑非法状态是否出现和出现后的取值
    int k=lg[r-l+1];
    return max(Max[k][l],Max[k][r-(1<<k)+1]);
}
int querymn(int l,int r){
    if (l>r)return 1e9;//注意考虑非法状态是否出现和出现后的取值
    int k=lg[r-l+1];
    return min(Min[k][l],Min[k][r-(1<<k)+1]);
```

```
}
```

6 Trie 树

求对 trie 树中的数异或最值

```
struct trie{
        static const int B=30;
        struct node{
                int to[2];
                int &operator[](int n){return to[n];}
        }a[maxn];
        int t;
        void init(){t=0;a[t++]=node();}
        void insert(ll s){
                int k=0;
                for (int i=B-1;i>=0;i--){
                        int c=(s>>i)&1;
                        if (!a[k][c])a[k][c]=t,a[t++]=node();
                        k=a[k][c];
                }
        }
        ll query(ll s){//求异或最小值
                int k=0; ll ans=0;
                for (int i=B-1;i>=0;i--){
                        int c=(s>>i)&1;
                        if (!a[k][c])c^=1,ans^=1ll<<i;
                        k=a[k][c];
                }
                return ans;
        }
}t;
```

7 树状数组

7.1 二维树状数组

用法一：** 单点修改，区间查询 **
单点修改：add(x,y,v)
区间查询：query(x2,y2)+query(x1-1,y1-1)-query(x1-1,y2)-query(x2,y1-1)
用法二：** 区间修改，单点查询 **
区间修改：add(x1,y1,v),add(x2+1,y2+1,v),add(x1,y2+1,-v),add(x2+1,y1,-v)
单点查询：query(x,y)

```
const int SIZE=5005;
int limit1=SIZE-1,limit2=SIZE-1;//两个维度的下标的上界
ll bit[SIZE][SIZE];

void add(int x,int y,ll v){
        for(int i=x;i<=limit1;i+=i&-i)
                for(int j=y;j<=limit2;j+=j&-j)
                        bit[i][j]+=v;
}
ll query(int x,int y){
```

```
        ll res=0;
        for(int i=x;i;i-=i&-i)
                for(int j=y;j;j-=j&-j)
                        res+=bit[i][j];
        return res;
}
```

用法三：** 区间修改，区间查询 **

令数组 a 的二维差分数组为 d ，即 $a[n][m]=\sum_{i=1}^{n}\sum_{j=1}^{m}d[i][j]$

则 $\sum_{i=1}^{x}\sum_{j=1}^{y}a[i][j]=\sum_{i=1}^{x}\sum_{j=1}^{y}\sum_{k=1}^{i}\sum_{t=1}^{j}d[k][t]=\sum_{i=1}^{x}\sum_{j=1}^{y}(x+1-i)(y+1-j)d[i][j]$

$=(x+1)(y+1)\sum_{i=1}^{x}\sum_{j=1}^{y}d[i][j]-(y+1)\sum_{i=1}^{x}\sum_{j=1}^{y}d[i][j]*i-(x+1)\sum_{i=1}^{x}\sum_{j=1}^{y}d[i][j]*j+\sum_{i=1}^{x}\sum_{j=1}^{y}d[i][j]*ij$

用四个树状数组分别维护 $d[i][j]$ ， $d[i][j]*i$ ， $d[i][j]*j$ ， $d[i][j]*ij$ 的前缀和

```
const int SIZE=5005;
int limit1=SIZE-1,limit2=SIZE-1;//两个维度的下标的上界
ll bit1[SIZE][SIZE],bit2[SIZE][SIZE],bit3[SIZE][SIZE],bit4[SIZE][SIZE
   ];

void add(int x,int y,ll v){
        for(int i=x;i<=limit1;i+=i&-i)for(int j=y;j<=limit2;j+=j&-j){
                bit1[i][j]+=v;
                bit2[i][j]+=v*x;
                bit3[i][j]+=v*y;
                bit4[i][j]+=v*x*y;
        }
}
ll query(int x,int y){
        ll res=0;
        for(int i=x;i;i-=i&-i)
                for(int j=y;j;j-=j&-j)
                        res+=(x+1)*(y+1)*bit1[i][j]-(y+1)*bit2[i][j
                           ]-(x+1)*bit3[i][j]+bit4[i][j];
        return res;
}
void real_add(int x1,int y1,int x2,int y2,ll v){
        add(x1,y1,v);
        add(x1,y2+1,-v);
        add(x2+1,y1,-v);
        add(x2+1,y2+1,v);
}
ll real_query(int x1,int y1,int x2,int y2){
        return query(x2,y2)-query(x1-1,y2)-query(x2,y1-1)+query(x1-1,
           y1-1);
}
```

8 线段树

8.1 log^2

例：楼房重建
单点更新询问最长序列长度

序列定义：区间第一个必选，往后遍历的过程中，若遇到一个数大于上一个值则选。
信息：
l：区间序列的最左端值大小
r：区间序列的最右端值大小
num：区间序列长度
$pushup$ 时，若左区间的 $r <$ 右区间的 l，则 num 可以直接加
反之需要计算当前值为左区间的 r 时，在右区间的序列长度
同样用一个分治函数计算
因线段树将一个区间分为 log 段，而每段区间又会继续分下去，最多分 log 次
故复杂度为 log^2

```
struct node{
    double l,r;
    int num;
}tr[maxn<<2];
int n,m;
int q(int p,int l,int r,double mx){
    if (l==r)return mx<tr[p].r;
    if (tr[p].l>mx)return tr[p].num;
    if (mx>=tr[p].r)return 0;
    int mid=(l+r)>>1;
    if (mx<tr[p<<1].r)return q(p<<1,l,mid,mx)+tr[p].num-tr[p<<1].num;
    else return q(p<<1|1,mid+1,r,mx);
}
void pushup(int p,int l,int r){
    tr[p].l=tr[p<<1].l;
    tr[p].r=max(tr[p<<1].r,tr[p<<1|1].r);
    int mid=(l+r)>>1;
    if (tr[p<<1].r<tr[p<<1|1].l)tr[p].num=tr[p<<1].num+tr[p<<1|1].num
        ;
    else tr[p].num=tr[p<<1].num+q(p<<1|1,mid+1,r,tr[p<<1].r);
}
void update(int x,int y,int l=1,int r=n,int p=1){
    if (x>r||x<l)return;
    if (l==r&&l==x){
        tr[p].l=tr[p].r=1.0*y/x;
        tr[p].num=1;
        return;
    }
    int mid=(l+r)>>1;
    update(x,y,l,mid,p<<1);
    update(x,y,mid+1,r,p<<1|1);
    pushup(p,l,r);
}
```

8.2 猫树

预处理 $O(nlog)$ 查询 $O(1)$
具体来讲我们建树的时候对于线段树树上的一个节点，设它代表的区间为 $(l,r]$。
不同于传统线段树在这个节点里只保留 $[l,r]$ 的和，我们在这个节点里面额外保存 $(l,mid]$ 的后缀和数组和 $(mid,r]$ 的前缀和数组。
如果我们询问的区间是 $[l,r]$ 那么我们把代表 $[l,l]$ 的节点和代表 $[r,r]$ 的节点的 lca 求出来，记为 p。
根据刚才的两个性质，l,r 在 p 所包含的区间之内并且一定跨越了 p 的中点。

这意味这一个非常关键的事实是我们可以使用 p 里面的前缀和数组和后缀和数组，将 $[l,r]$ 拆成 $[l,mid]+(mid,r]$ 从而拼出来 $[l,r]$ 这个区间。

而这个过程仅仅需要 $O(1)$ 次合并操作！

求 lca 的过程如下

将这个序列补成 2 的整次幂，然后建线段树。

此时我们发现线段树上两个节点的 lca 编号，就是两个节点二进制编号的 lcp。

稍作思考即可发现发现在 x 和 y 的二进制下 lcp(x,y)=x»log[xŷ]。

所以我们预处理一个 ‘log‘ 数组即可轻松完成求 lca 的工作。

9 线段树合并/分裂

洛谷 P5494

合并/分裂两个动态开点线段树。

** 如用数组来代替结构体，一定要记得开 32 倍空间!!! **

```
const int maxn=1e5+5;
int n,m,trnum,rub,rt[maxn],bin[maxn<<5],all;
struct sgt{
  int ls,rs;
  ll cnt;
}tr[maxn<<5];
int new_node(){//新建节点
  if (rub)return bin[rub--];
  else return ++trnum;
}
void del_node(int p){//删点
  tr[p].ls=tr[p].rs=tr[p].cnt=0;
  bin[++rub]=p;
}
void update(int &now,int x,int v,int l=1,int r=n){
  if (now==0)now=new_node();
  tr[now].cnt+=v;
  if (l==r)return;
  int mid=(l+r)>>1;
  if (x<=mid)update(tr[now].ls,x,v,l,mid);
  else update(tr[now].rs,x,v,mid+1,r);
}
int merge(int x,int y){//合并
  if (!x||!y)return x^y;
  tr[x].cnt+=tr[y].cnt;
  tr[x].ls=merge(tr[x].ls,tr[y].ls);
  tr[x].rs=merge(tr[x].rs,tr[y].rs);
  del_node(y);
  return x;
}
void spilt(int x,int &y,ll k){//分裂
  if (x==0)return ;
  y=new_node();
  ll v=tr[tr[x].ls].cnt;
  if (k<=v)swap(tr[x].rs,tr[y].rs);
  if (k<v)spilt(tr[x].ls,tr[y].ls,k);
  if (k>v)spilt(tr[x].rs,tr[y].rs,k-v);
  tr[y].cnt=tr[x].cnt-k;
  tr[x].cnt=k;
}
```

```
int query_kmin(int now,int k,int l=1,int r=n){//查询第k大
  if (l==r)return l;
  int mid=(l+r)>>1;
  if (k<=tr[tr[now].ls].cnt)return query_kmin(tr[now].ls,k,l,mid);
  else return query_kmin(tr[now].rs,k-tr[tr[now].ls].cnt,mid+1,r);
}
ll query_cnt(int now,int a,int b,int l=1,int r=n){//查询[a,b]范围内的数量
  if (!now||l>b||r<a)return 0;
  if (l>=a&&b>=r)return tr[now].cnt;
  int mid=(l+r)>>1;
  return query_cnt(tr[now].ls,a,b,l,mid)+query_cnt(tr[now].rs,a,b,mid+1,r);
}
int p,x,y,tmp;
int main() {
    n=rd();m=rd();
    all=1;
    for (int i=1,x;i<=n;i++){
      x=rd();
      update(rt[1],i,x);
    }
    while (m--){
      int op=rd();p=rd();
      if (op==0){
        x=rd();y=rd();
        ll k1=query_cnt(rt[p],1,y);
        ll k2=query_cnt(rt[p],x,y);
        spilt(rt[p],rt[++all],k1-k2);
        spilt(rt[all],tmp,k2);
        rt[p]=merge(rt[p],tmp);
      }
      else if (op==1){
        y=rd();
        rt[p]=merge(rt[p],rt[y]);
      }
      else if (op==2){
        x=rd();y=rd();
        update(rt[p],y,x);
      }
      else if (op==3){
        x=rd();y=rd();
        printf("%lld\n",query_cnt(rt[p],x,y));
      }
      else{
        x=rd();
        if (x<=0||x>query_cnt(rt[p],1,n))puts("-1");
        else printf("%d\n",query_kmin(rt[p],x));
      }
    }
    return 0;
}
```

10 主席树

求区间第 k 小的数
洛谷 P3834
开 n 个对应前缀 [1,i] 的权值线段树，因为每次相当于只更新一个位置，故每次只开 log 个节点，其他节点可以沿用上一个版本的节点。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
inline ll rd(){
    ll x=0;char o,f=1;
    while(o=getchar(),o<48)if(o==45)f=-f;
    do x=(x<<3)+(x<<1)+(o^48);
    while(o=getchar(),o>47);
    return x*f;
}
const int maxn=2e5+5;
int n,q,tot,l,r,k,a[maxn],b[maxn],trnum,rt[maxn];
struct PST{
    int ls,rs,num;
}tree[maxn<<5];
void update(int &now,int lst,int x,int l=1,int r=tot){
    now=++trnum;
    tree[now]=tree[lst];tree[now].num++;
    if (l==r)return;
    int mid=(l+r)>>1;
    if (x<=mid)update(tree[now].ls,tree[lst].ls,x,l,mid);
    else update(tree[now].rs,tree[lst].rs,x,mid+1,r);
}
int query(int L,int R,int k,int l=1,int r=tot){
    if (l==r)return l;
    int mid=(l+r)>>1;
    int res=tree[tree[R].ls].num-tree[tree[L].ls].num;
    if (k<=res)return query(tree[L].ls,tree[R].ls,k,l,mid);
    else return query(tree[L].rs,tree[R].rs,k-res,mid+1,r);
}
int main() {
    tree[0].ls=tree[0].rs=tree[0].num=0;
    n=rd();q=rd();
    for (int i=1;i<=n;i++)a[i]=rd(),b[i]=a[i];
    sort(b+1,b+n+1);
    tot=unique(b+1,b+n+1)-b-1;
    for (int i=1;i<=n;i++)a[i]=lower_bound(b+1,b+tot+1,a[i])-b;
    for (int i=1;i<=n;i++)update(rt[i],rt[i-1],a[i]);
    while (q--){
        l=rd();r=rd();k=rd();
        printf("%d\n",b[query(rt[l-1],rt[r],k)]);
    }
    return 0;
}
```

带修主席树
支持单点修改的查询区间第 k 小的主席树
实现思路是树状数组套主席树
裸的主席树上若想实现改功能，需要单次 $O(nlogn)$ 的复杂度更新当前点及之后点的权值线段树

于是考虑减少这个过程，
发现树状数组的思想可以优化这一过程，具体为：i 这点更新了 $i+lowbit(i)$ 也更新，以此类推
查询时也需要查询 log 棵权值线段树
故单次更新/查询复杂度均为 log^2
总复杂度为 $O(nlog^2n)$

```
const int maxn=5e5+5;
struct query{
    int op,l,r,k;
}q[maxn];
int n,m,a[maxn],b[maxn],tot,trnum,rt[maxn];
struct PST{
    int ls,rs,num;
}tr[maxn<<5];
void update(int &now,int x,int v,int l=1,int r=tot){
    if (!now)now=++trnum;
    tr[now].num+=v;
    if (l==r)return;
    int mid=(l+r)>>1;
    if (x<=mid)update(tr[now].ls,x,v,l,mid);
    else update(tr[now].rs,x,v,mid+1,r);
}
void add(int x,int v){
    int k=lower_bound(b+1,b+tot+1,a[x])-b;
    for (int i=x;i<=n;i+=i&-i)update(rt[i],k,v);
}
int tmp[2][20],cnt[2];
int query(int k,int l=1,int r=tot){
    if (l==r)return l;
    int mid=(l+r)>>1,sum=0;
    for (int i=1;i<=cnt[1];i++)sum+=tr[tr[tmp[1][i]].ls].num;
    for (int i=1;i<=cnt[0];i++)sum-=tr[tr[tmp[0][i]].ls].num;
    if (k<=sum){
        for (int i=1;i<=cnt[1];i++)tmp[1][i]=tr[tmp[1][i]].ls;
        for (int i=1;i<=cnt[0];i++)tmp[0][i]=tr[tmp[0][i]].ls;
        return query(k,l,mid);
    }
    else{
        for (int i=1;i<=cnt[1];i++)tmp[1][i]=tr[tmp[1][i]].rs;
        for (int i=1;i<=cnt[0];i++)tmp[0][i]=tr[tmp[0][i]].rs;
        return query(k-sum,mid+1,r);
    }
}
int ask(int l,int r,int k){
    memset(tmp,0,sizeof(tmp));
    cnt[1]=cnt[0]=0;
    for (int i=r;i;i-=i&-i)tmp[1][++cnt[1]]=rt[i];
    for (int i=l-1;i;i-=i&-i)tmp[0][++cnt[0]]=rt[i];
    return query(k);
}
int main() {
    n=rd();m=rd();
    for (int i=1;i<=n;i++)a[i]=rd(),b[++tot]=a[i];
    for (int i=1;i<=m;i++){
        char c;
        while (c=getchar(),c!='Q'&&c!='C');
```

```
        if (c=='Q'){
            q[i].op=2;
            q[i].l=rd();q[i].r=rd();q[i].k=rd();
        }
        else{
            q[i].op=1;q[i].l=rd();q[i].k=rd();
            b[++tot]=q[i].k;
        }
    }//修改的操作全部读入离散化
    sort(b+1,b+tot+1);
    tot=unique(b+1,b+tot+1)-b-1;
    for (int i=1;i<=n;i++)add(i,1);
    for (int i=1;i<=m;i++){
        if (q[i].op==1){
            add(q[i].l,-1);
            a[q[i].l]=q[i].k;
            add(q[i].l,1);
        }
        else{
            printf("%d\n",b[ask(q[i].l,q[i].r,q[i].k)]);
        }
    }
    return 0;
}
```

11 李超线段树

在二维平面上，支持插入一条线段，查询横坐标为某个值时最上面的线段。插入 $O(log^2 n)$，查询 $O(log\ n)$。

运用标记永久化的思想，保存每个区间的最优线段

查询单点时，递归过程中对 log 个区间取最大值即是答案

```
const int maxn=1e5+5;
const int mod1=39989;
const int mod2=1e9+1;
const int N=39989;
struct seg{
    double k,b;
    seg(){}
    seg(int X0,int Y0,int X1,int Y1){
        if (X0==X1)k=0,b=max(Y0,Y1);
        else k=1.0*(Y0-Y1)/(X0-X1),b=-k*X0+Y0;
    }
    double gety(int x){return k*x+b;}
}s[maxn];
int cnt,ans,v[maxn<<2];
int trans(int x,int mod){return (x+ans-1+mod)%mod+1;}
inline int sgn(double x){return fabs(x)<1e-8?0:(x>0?1:-1);}
void update(int a,int b,int now,int l=1,int r=N,int p=1){
    if (a>r||b<l)return;
    if (a<=l&&b>=r){
        if (sgn(s[now].gety(l)-s[v[p]].gety(l))>0
            &&sgn(s[now].gety(r)-s[v[p]].gety(r))>0){v[p]=now;return
                ;}
        if (sgn(s[now].gety(l)-s[v[p]].gety(l))<=0
```

```
            &&sgn(s[now].gety(r)-s[v[p]].gety(r))<=0)return;
        if (l==r)return;
    }
    int mid=(l+r)>>1;
    update(a,b,now,l,mid,p<<1);update(a,b,now,mid+1,r,p<<1|1);
}
void ask(int a,int l=1,int r=N,int p=1){
    if (sgn(s[ans].gety(a)-s[v[p]].gety(a))<0)ans=v[p];
    else if (!sgn(s[ans].gety(a)-s[v[p]].gety(a))&&ans>v[p])ans=v[p];
    if (l==r)return;
    int mid=(l+r)>>1;
    if (a<=mid)ask(a,l,mid,p<<1);
    else ask(a,mid+1,r,p<<1|1);
}
int main() {
    int q=rd();
    while (q--){
        int op=rd();
        if (op==1){
            int X0=trans(rd(),mod1),Y0=trans(rd(),mod2);
            int X1=trans(rd(),mod1),Y1=trans(rd(),mod2);
            if (X0>X1)swap(X0,X1),swap(Y0,Y1);
            s[++cnt]=seg(X0,Y0,X1,Y1);
            update(X0,X1,cnt);
        }
        else{
            int k=trans(rd(),mod1);
            ans=0;ask(k);
            printf("%d\n",ans);
        }
    }
    return 0;
}
```

12 吉老师线段树

即 $Segment\ Tree\ Beats$
给出一个长度为 n 的数列 A
维护 5 种操作
操作 1：区间 $+k$
操作 2：区间内值更新为 $min(A_i, v)$
操作 3：询问区间和
操作 4：询问区间最大值
操作 5：询问区间历史最大值
线段树变量含义：
sum：区间和
$maxa$：区间最大值
se：区间严格次大值
cnt：区间最大值的数量
$maxb$：区间历史最大值
$lazy$ 标记：
add_a：区间最大值加法 $lazy$ 标记
add_a1：区间非最大值加法 $lazy$ 标记
add_b：区间最大的历史最大值的加法 $lazy$ 标记
add_b1：区间非最大的历史最大值的加法 $lazy$ 标记

区间取 min，意味着只对那些大于 t 的数有更改。因此这个操作的对象不再是整个区间，而是“这个区间中大于 t 的数”。于是我们可以有这样的思路：每个结点维护该区间的最大值 Max、次大值 Se、区间和 Sum 以及最大值的个数 Cnt。接下来我们考虑区间对 t 取 min 的操作。

1. 如果 $Max \le t$，显然这个 t 是没有意义的，直接返回；2. 如果 $Se < t \le Max$，那么这个 t 就能更新当前区间中的最大值。于是我们让区间和加上 $Cnt(t - Max)$，然后更新 Max 为 t，并打一个标记。3. 如果 $t \le Se$，那么这时你发现你不知道有多少个数涉及到更新的问题。于是我们的策略就是，暴力递归向下操作。然后上传信息。

这个算法的复杂度如何？使用势能分析法可以得到复杂度是 $O(m \log n)$ 的。具体分析过程见论文。

```
const int maxn=5e5+5;
struct sgt{
    ll sum;
    int add_a,add_a1,add_b,add_b1;
    int maxa,se,maxb,cnt;
}tr[maxn<<2];
int n,m,a[maxn];
void pushup(int p){
    tr[p].maxa=max(tr[p<<1].maxa,tr[p<<1|1].maxa);
    tr[p].maxb=max(tr[p<<1].maxb,tr[p<<1|1].maxb);
    tr[p].sum=tr[p<<1].sum+tr[p<<1|1].sum;
    if (tr[p<<1].maxa==tr[p<<1|1].maxa){
        tr[p].se=max(tr[p<<1].se,tr[p<<1|1].se);
        tr[p].cnt=tr[p<<1].cnt+tr[p<<1|1].cnt;
    }
    if (tr[p<<1].maxa>tr[p<<1|1].maxa){
        tr[p].se=max(tr[p<<1].se,tr[p<<1|1].maxa);
        tr[p].cnt=tr[p<<1].cnt;
    }
    if (tr[p<<1].maxa<tr[p<<1|1].maxa){
        tr[p].se=max(tr[p<<1|1].se,tr[p<<1].maxa);
        tr[p].cnt=tr[p<<1|1].cnt;
    }
}
void build(int l=1,int r=n,int p=1){
    if (l==r){
        tr[p].sum=tr[p].maxa=tr[p].maxb=a[l];
        tr[p].se=-1e9;tr[p].cnt=1;
        return;
    }
    int mid=(l+r)>>1;
    build(l,mid,p<<1);build(mid+1,r,p<<1|1);
    pushup(p);
}
void update(int k1,int k2,int k3,int k4,int p,int len){
    tr[p].sum+=1ll*k1*tr[p].cnt+1ll*k3*(len-tr[p].cnt);
    tr[p].maxb=max(tr[p].maxb,tr[p].maxa+k2);
    tr[p].add_b=max(tr[p].add_b,tr[p].add_a+k2);
    tr[p].add_b1=max(tr[p].add_b1,tr[p].add_a1+k4);
    tr[p].maxa+=k1;tr[p].add_a+=k1;
    tr[p].add_a1+=k3;
    if (tr[p].se!=-1e9)tr[p].se+=k3;
}
void pushdown(int p,int l,int r){
    int maxn=max(tr[p<<1].maxa,tr[p<<1|1].maxa);
    int mid=(l+r)>>1;
```

```
47      if (maxn==tr[p<<1].maxa)
48          update(tr[p].add_a,tr[p].add_b,tr[p].add_a1,tr[p].add_b1,p
              <<1,mid-l+1);
49      else update(tr[p].add_a1,tr[p].add_b1,tr[p].add_a1,tr[p].add_b1,p
          <<1,mid-l+1);
50      if (maxn==tr[p<<1|1].maxa)
51          update(tr[p].add_a,tr[p].add_b,tr[p].add_a1,tr[p].add_b1,p
              <<1|1,r-mid);
52      else update(tr[p].add_a1,tr[p].add_b1,tr[p].add_a1,tr[p].add_b1,p
          <<1|1,r-mid);
53      tr[p].add_a=tr[p].add_a1=tr[p].add_b=tr[p].add_b1=0;
54  }
55  void update_add(int a,int b,int v,int l=1,int r=n,int p=1){
56      if (a>r||b<l)return;
57      if (a<=l&&b>=r){
58          update(v,v,v,v,p,r-l+1);
59          return;
60      }
61      int mid=(l+r)>>1;
62      pushdown(p,l,r);
63      update_add(a,b,v,l,mid,p<<1);
64      update_add(a,b,v,mid+1,r,p<<1|1);
65      pushup(p);
66  }
67  void update_min(int a,int b,int v,int l=1,int r=n,int p=1){
68      if (a>r||b<l||v>=tr[p].maxa)return;
69      if (a<=l&&b>=r&v>tr[p].se){
70          update(v-tr[p].maxa,v-tr[p].maxa,0,0,p,r-l+1);
71          return;
72      }
73      int mid=(l+r)>>1;
74      pushdown(p,l,r);
75      update_min(a,b,v,l,mid,p<<1);
76      update_min(a,b,v,mid+1,r,p<<1|1);
77      pushup(p);
78  }
79  ll query_sum(int a,int b,int l=1,int r=n,int p=1){
80      if (a>r||b<l)return 0;
81      if (a<=l&&b>=r)return tr[p].sum;
82      int mid=(l+r)>>1;
83      pushdown(p,l,r);
84      return query_sum(a,b,l,mid,p<<1)+query_sum(a,b,mid+1,r,p<<1|1);
85  }
86  ll query_maxa(int a,int b,int l=1,int r=n,int p=1){
87      if (a>r||b<l)return -1e9;
88      if (a<=l&&b>=r)return tr[p].maxa;
89      int mid=(l+r)>>1;
90      pushdown(p,l,r);
91      return max(query_maxa(a,b,l,mid,p<<1),query_maxa(a,b,mid+1,r,p
          <<1|1));
92  }
93  ll query_maxb(int a,int b,int l=1,int r=n,int p=1){
94      if (a>r||b<l)return -1e9;
95      if (a<=l&&b>=r)return tr[p].maxb;
96      int mid=(l+r)>>1;
97      pushdown(p,l,r);
```

```
98      return max(query_maxb(a,b,l,mid,p<<1),query_maxb(a,b,mid+1,r,p
            <<1|1));
99  }
100 int main() {
101     n=rd();m=rd();
102     for (int i=1;i<=n;i++)a[i]=rd();
103     build();
104     while (m--){
105         int op=rd(),l=rd(),r=rd(),k;
106         if (op==1)k=rd(),update_add(l,r,k);
107         else if (op==2)k=rd(),update_min(l,r,k);
108         else if (op==3)printf("%lld\n",query_sum(l,r));
109         else if (op==4)printf("%lld\n",query_maxa(l,r));
110         else if (op==5)printf("%lld\n",query_maxb(l,r));
111     }
112     return 0;
113 }
```

13 可持久化字典树

注意考虑查询的区间会不会包含隐含的 0
如果需要，在 $rt[1]$ 处插入，$rt[1]=1$，$query$ 函数中 L 传入 0 即可

```
1   struct Trie{
2       int t,rt[maxn];
3       struct node{
4           int to[2],val;
5           int &operator[](int n){return to[n];}
6           node(){to[0]=to[1]=val=0;}
7       }a[maxn<<5];
8       void init(){
9           t=0;a[t++]=node();
10      }
11      void insert(int &no,int lst,int v){
12          no=t;a[t++]=node();
13          int now=no;
14          for (int i=30;i>=0;i--){
15              a[now].val=a[lst].val+1;
16              if ((v&(1<<i))==0){
17                  if (!a[now][0])a[now][0]=t,a[t++]=node();
18                  a[now][1]=a[lst][1];
19                  now=a[now][0];
20                  lst=a[lst][0];
21              }
22              else{
23                  if (!a[now][1])a[now][1]=t,a[t++]=node();
24                  a[now][0]=a[lst][0];
25                  now=a[now][1];
26                  lst=a[lst][1];
27              }
28          }
29          a[now].val=a[lst].val+1;
30      }
31      int query(int L,int R,int v){
32          int ans=0;
33          for (int i=30;i>=0;i--){
```

```
            int tmp=(v&(1<<i))?1:0;
            if (a[a[R][!tmp]].val-a[a[L][!tmp]].val){
                ans+=(1<<i);
                L=a[L][!tmp];R=a[R][!tmp];
            }
            else{
                L=a[L][tmp];R=a[R][tmp];
            }
        }
        return ans;
    }
}t;
```

14 可撤销并查集

14.1 基础版

```
struct revoke_dsu{
    static const int SIZE=1e6+5;
    int top,fa[SIZE],sz[SIZE],st[SIZE];
    void init(int n){
        top=0;
        for (int i=1;i<=n;i++)fa[i]=i,sz[i]=1;
    }
    int find(int x){
        while (x^fa[x])x=fa[x];
        return x;
    }
    bool merge(int x,int y){
        x=find(x),y=find(y);
        if (x==y)return 0;
        if (sz[x]>sz[y])swap(x,y);
        fa[x]=y;
        sz[y]+=sz[x];
        st[++top]=x;
        return 1;
    }
    void pop_to(int tar){
        while (top>tar){
            sz[fa[st[top]]]-=sz[st[top]];
            fa[st[top]]=st[top];
            top--;
        }
    }
}U;
```

14.2 支持查询点到根距离的奇偶性

```
struct revoke_dsu{
    static const int SIZE=1e6+5;
    int top,fa[SIZE],sz[SIZE],st[SIZE];
    int len[SIZE],dis[SIZE];//dis代表点到根距离奇偶性，len代表点到fa[
       x]的距离奇偶性
```

```
    void init(int n){
        top=0;
        for (int i=1;i<=n;i++)fa[i]=i,sz[i]=1;
    }
    int find(int x){
        if (fa[x]==x){dis[x]=0;return x;}
        int f=find(fa[x]);
        dis[x]=dis[fa[x]]+len[x];
        return f;
    }
    bool merge(int x,int y){
        int fx=find(x),fy=find(y);
        if (fx==fy)return 0;
        if (sz[fx]>sz[fy])swap(fx,fy);
        fa[fx]=fy;
        sz[fy]+=sz[fx];
        st[++top]=fx;
        len[fx]=(dis[x]+dis[y]+1)%2;
        return 1;
    }
    void pop_to(int tar){
        while (top>tar){
            sz[fa[st[top]]]-=sz[st[top]];
            fa[st[top]]=st[top];
            top--;
        }
    }
}U;
```

15 线段树分治

题目：https://codeforces.com/gym/102968/problem/D

题意：

给定一个 n 个点 m 条边的图，你可以给每条边赋值 [0,val] (保证 val=2^k-1)

Q 次操作，每次加入一条边或删除一条边。

要求：图上每一个环的异或和为 0

求图的种类数

思路：

每条边可以通过 map 或 sort 等方法得到生命周期。然后用线段树分治的方法，每条边的生命周期最多被划分成 log 段，然后将其“挂”在线段树的节点上。

查询的时候每次更新当前节点上有的边，然后递归下去。

我们使用可撤销并查集来维护连通性，这样回溯的时候可以直接撤销操作来达到删除边的目的。

```
struct revoke_dsu{
    static const int SIZE=1e6+5;
    int top,fa[SIZE],sz[SIZE],st[SIZE];
    void init(int n){
        top=0;
        for (int i=1;i<=n;i++)fa[i]=i,sz[i]=1;
    }
    int find(int x){
        while (x^fa[x])x=fa[x];
        return x;
    }
    bool merge(int x,int y){
```

```
13          x=find(x),y=find(y);
14          if (x==y)return 0;
15          if (sz[x]>sz[y])swap(x,y);
16          fa[x]=y;
17          sz[y]+=sz[x];
18          st[++top]=x;
19          return 1;
20      }
21      void pop_to(int tar){
22          while (top>tar){
23              sz[fa[st[top]]]-=sz[st[top]];
24              fa[st[top]]=st[top];
25              top--;
26          }
27      }
28  }U;
29  const int maxn=1e6+5;
30  const ll mod=1e9+7;
31  ll ksm(ll a,ll b){ll ans=1;a%=mod;while(b){if(b&1)ans=(ans*a)%mod;a=(
        a*a)%mod;b>>=1;}return ans;}
32  ll n,m,q,val,tot,ans=1,ni;
33  ll s[maxn],e[maxn],from[maxn],to[maxn];
34  map<pair<int,int>,int>mp;
35  vector<int>tree[maxn<<2];
36  void insert(int a,int b,int pos,int l=1,int r=q+1,int p=1){
37          if (a>r||b<l)return;
38          if (a<=l&&b>=r){tree[p].push_back(pos);return;}
39          int mid=(l+r)>>1;
40          insert(a,b,pos,l,mid,p<<1);insert(a,b,pos,mid+1,r,p<<1|1);
41  }
42  void dfs(int l,int r,int p=1){
43          int last_top=U.top;
44      ll last_ans=ans;
45          for (int u:tree[p]){
46                  if (U.merge(from[u],to[u])){ans=ans*val%mod;}
47          }
48          if (l==r)printf("%lld\n",ans);
49          else{
50                  int mid=(l+r)>>1;
51                  dfs(l,mid,p<<1);dfs(mid+1,r,p<<1|1);
52          }
53          U.pop_to(last_top);
54      ans=last_ans;
55  }
56  int main() {
57      n=rd();m=rd();q=rd();val=rd();val=(val+1)%mod;
58      U.init(n);ni=ksm(val,mod-2);
59      for (int i=1;i<=m;i++){
60          from[i]=rd();to[i]=rd();
61          if (from[i]>to[i])swap(from[i],to[i]);
62          mp[make_pair(from[i],to[i])]=i;
63          s[i]=1;e[i]=q+1;
64      }
65      tot=m;
66      for (int i=1,u,v;i<=q;i++){
67          u=rd();v=rd();
68          if (u>v)swap(u,v);
```

```
        if (mp[make_pair(u,v)]){
                e[mp[make_pair(u,v)]]=i;
                mp[make_pair(u,v)]=0;
        }
        else{
                tot++;
                from[tot]=u;to[tot]=v;
                s[tot]=i+1;e[tot]=q+1;
                mp[make_pair(u,v)]=tot;
        }
    }
    for (int i=1;i<=tot;i++)insert(s[i],e[i],i);
    dfs(1,q+1);
    return 0;
}
```

16 CDQ 分治

把点对 (i,j) 分为三类
$1 <= i <= mid, 1 <= j <= mid$
$1 <= i <= mid, mid+1 <= j <= r$
$mid+1 <= i <= r, mid+1 <= j <= r$
一与三都可以递归完成，主要问题集中在处理第二种。
洛谷 P3810
题意：三维偏序
有 n 个元素，第 i 个元素有 $a[i], b[i], c[i]$ 三个属性，设 $f[i]$ 表示满足 $a[j] <= a[i]$ 且 $b[j] <= b[i]$ 且 $c[j] <= c[i]$ 且 $j \neq i$ 的 j 的数量。对于 d[0,n)，求 $f[i] = d$ 的数量。
思路：
主函数在 CDQ 分治之前先对第一维升序排序（若相同其他维也升序）。
在函数中用归并排序的方法使得每次处理完都是对第二维升序。
因此每次处理贡献时，可以保证第二维的单调性，对第一维再与 mid 比较，第三维可用树状数组维护

```
const int maxn=1e5+5;
ll n,k;
int a[maxn],b[maxn],c[maxn],rk[maxn],cnt[maxn],tot,res[maxn],ans[maxn
    ];
bool cmp(int x,int y){
    if (a[x]!=a[y])return a[x]<a[y];
    if (b[x]!=b[y])return b[x]<b[y];
    return c[x]<c[y];
}
struct node{
    int y,z,id;
    bool operator<(const node A)const{
        if (y!=A.y)return y<A.y;
        return id<A.id;
    }
}Q[maxn],ji[maxn];
int bit[maxn<<1],st[maxn],num[maxn],top;
void add(int x,int v){
    while (x<=k){
        bit[x]+=v;
        x+=x&-x;
    }
}
```

```
int query(int x){
    int ans=0;
    while (x){
        ans+=bit[x];x-=x&-x;
    }
    return ans;
}

void CDQ(int l,int r){
    if (l==r)return;
    int mid=(l+r)>>1;
    CDQ(l,mid);CDQ(mid+1,r);
    int p1=l,p2=mid+1,o=l-1;
    while (p1<=mid&&p2<=r){
        if (Q[p1].y<=Q[p2].y){
            add(Q[p1].z,cnt[Q[p1].id]);
            st[++top]=Q[p1].z;num[top]=cnt[Q[p1].id];
            ji[++o]=Q[p1++];
        }
        else{
            res[Q[p2].id]+=query(Q[p2].z);
            ji[++o]=Q[p2++];
        }
    }
    while (p1<=mid)ji[++o]=Q[p1++];
    while (p2<=r)res[Q[p2].id]+=query(Q[p2].z),ji[++o]=Q[p2++];
    for (int i=l;i<=r;i++)Q[i]=ji[i];
    while (top){
        add(st[top],-num[top]);
        top--;
    }
}
int main() {
    n=rd();k=rd();
    for (int i=1;i<=n;i++)a[i]=rd(),b[i]=rd(),c[i]=rd(),rk[i]=i;
    sort(rk+1,rk+n+1,cmp);
    for (int i=1;i<=n;i++){
        if (a[rk[i]]!=a[rk[i-1]]||b[rk[i]]!=b[rk[i-1]]||c[rk[i]]!=c[
           rk[i-1]])
            tot++,Q[tot]={b[rk[i]],c[rk[i]],tot};
        cnt[tot]++;
    }
    CDQ(1,tot);
    for (int i=1;i<=tot;i++)ans[res[i]+cnt[i]-1]+=cnt[i];
    for (int i=0;i<n;i++)cout<<ans[i]<<endl;
    return 0;
}
```

17 KD Tree

(插入 + 查询)
luoguP4148 二维空间单点插入 + 矩阵求和
操作 1：在 (x,y) 加 v
操作 2：查询（x1,y1) 到（x2,y2) 矩阵内数字和
强制在线且空间小不能树套树

这题为 2-D Tree

每个节点都类似于线段树的一个节点。不同的是，线段树每一个节点表示一段区间的总体信息，KD Tree 中一个节点表示了 K 维空间中一个区域的总体信息，例如本题中一个节点表示了一个矩阵的总体信息。

每个节点本身是一个 K 维空间的一个点，以这个点为分界点，按照某一维（x 或 y）对这个区域进行划分。

显然划分的时候，如果点的坐标是中位数是最优的。这样可以保证树的高度是 log 层

所以就会用到一个 KD-Tree 常用的函数 nth_element(); 函数介绍在第 3 点。

查询类似于线段树查询，复杂度比较玄学，主要看剪枝。

本题每个节点维护信息有：

结构体 node：当前节点的坐标与值

ls,rs：左儿子右儿子

sum：矩阵内所有点的权值和

sz：当前子树节点数量

mn[i],mx[i]：当前子树在第 i 维的最值

nth_element() 介绍

函数介绍 nth_element(a+1,a+x+1,a+n+1);

将 a[1]-a[n] 这段数组中的数，第 x 大的数放置在 a[x]。

且将比 a[x] 小的数放置在 a[1]-a[x-1]，把比 a[x] 大的数放置在 a[x+1]-a[n]

时间复杂度：$O(n)$

在 KD Tree 中的运用：

nth_element(tmp+l,tmp+mid,tmp+r+1);

以中位数来划分区间。

```
const int maxn=2e5+5;
int n,op,x,y,v,X1,Y1,X2,Y2;
ll lastans;
int cnt,TYPE,root,treeid,rub,bin[maxn],dfsid;
struct kd_node{
    int  dim[2],val;
    bool operator<(const kd_node a)const{
        return dim[TYPE]<a.dim[TYPE];
    }
}tree[maxn],tmp[maxn];
int type[maxn],ls[maxn],rs[maxn],mx[maxn][2],mn[maxn][2],sz[maxn];
ll sum[maxn];
void up(int x){
    for (int i=0;i<2;i++){
        mx[x][i]=mn[x][i]=tree[x].dim[i];
        if (ls[x]){
            mx[x][i]=max(mx[x][i],mx[ls[x]][i]);
            mn[x][i]=min(mn[x][i],mn[ls[x]][i]);
        }
        if (rs[x]){
            mx[x][i]=max(mx[x][i],mx[rs[x]][i]);
            mn[x][i]=min(mn[x][i],mn[rs[x]][i]);
        }
    }
    sum[x]=sum[ls[x]]+sum[rs[x]]+tree[x].val;
    sz[x]=sz[ls[x]]+sz[rs[x]]+1;
}
void new_node(int &x,int ty,const kd_node &rhs){
    if (rub)x=bin[rub--];
    else x=++treeid;
    type[x]=ty;
    tree[x]=rhs;
    ls[x]=rs[x]=0;
```

```
34      up(x);
35  }
36  void clean_node(int x){
37      bin[++rub]=x;
38      ls[x]=rs[x]=sz[x]=sum[x]=0;
39  }
40  void rebuild(int &x,int l,int r){//重建子树
41      int mid=(l+r)>>1;
42      TYPE=(++cnt)&1;
43      nth_element(tmp+l,tmp+mid,tmp+r+1);
44      new_node(x,TYPE,tmp[mid]);
45      if (l<mid)rebuild(ls[x],l,mid-1);
46      if (mid<r)rebuild(rs[x],mid+1,r);
47      up(x);
48  }
49  void to_array(int x){//子树全部压入临时数组
50      if (ls[x])to_array(ls[x]);
51      if (rs[x])to_array(rs[x]);
52      tmp[++dfsid]=tree[x];
53      clean_node(x);
54  }
55  void insert(int &x,const kd_node &rhs){//插入一个点
56      if (x==0){
57          new_node(x,(++cnt)&1,rhs);
58          return;
59      }
60      if (rhs.dim[type[x]]<=tree[x].dim[type[x]])insert(ls[x],rhs);
61      else insert(rs[x],rhs);
62      up(x);
63      if (sz[ls[x]]>sz[x]*0.75||sz[rs[x]]>sz[x]*0.75){
64          dfsid=0;
65          to_array(x);
66          rebuild(x,1,dfsid);
67      }
68  }
69  bool inside(int a1,int a2,int b1,int b2,int c1,int c2,int d1,int d2){
70      return c1>=a1&&c2<=a2&&d1>=b1&&d2<=b2;
71  }
72  bool outside(int a1,int a2,int b1,int b2,int c1,int c2,int d1,int d2)
        {
73      return c1>a2||c2<a1||d1>b2||d2<b1;
74  }
75  ll query(int x,int a1,int a2,int b1,int b2){
76      if (x==0)return 0;
77      if (outside(a1,a2,b1,b2,mn[x][0],mx[x][0],mn[x][1],mx[x][1]))
            return 0;
78      if (inside(a1,a2,b1,b2,mn[x][0],mx[x][0],mn[x][1],mx[x][1]))
            return sum[x];
79      ll ans=query(ls[x],a1,a2,b1,b2)+query(rs[x],a1,a2,b1,b2);
80      if (inside(a1,a2,b1,b2,tree[x].dim[0],tree[x].dim[0],tree[x].dim
            [1],tree[x].dim[1]))ans+=tree[x].val;
81      return ans;
82  }
83  int main() {
84      n=rd();
85      while (1){
86          op=rd();
```

```
        if (op==1){
            x=rd();y=rd();v=rd();
            x^=lastans;y^=lastans;v^=lastans;
            insert(root,(kd_node){x,y,v});
        }
        else if (op==2){
            X1=rd();Y1=rd();X2=rd();Y2=rd();
            X1^=lastans;Y1^=lastans;X2^=lastans;Y2^=lastans;
            printf("%lld\n",lastans=query(root,X1,X2,Y1,Y2));
        }
        else break;
    }
    return 0;
}
```

(建树 + 查询)

luoguP4357 二维空间给定 n 个点,求平面内第 k 远点对欧几里得距离平方(k<=min(100,n*(n-1)/2)

建 2-D Tree，把 n 个点都丢进树里

然后在小根堆中插入 2k 个 0

对于每一个点，都在 KD Tree 中进行查询。如果出现比当前小根堆堆顶元素大的就 pop 出来然后 push 当前距离进去。这样每一个点对会被 push 两次，最后堆顶即答案。

这样子听起来是很暴力，但是 KD Tree 做法关键在于剪枝

剪枝 1:

记录当前子树每一维度的最值，来计算出可能出现的最大值。若最大值也比堆顶要小就无需进入当前子树，就可以直接 return

剪枝 2:

计算左右子树的可能出现的最大值，先跑大的那一边，这样子小的那个被剪枝 1 剪掉的可能就更大了，故时间上会更优。

算不来复杂度貌似是 $O(n\sqrt{n})$

```
const int maxn=1e5+5;
int n,k,cnt,root,TYPE,treeid;
int ls[maxn],rs[maxn],type[maxn],mx[maxn][2],mn[maxn][2];
struct kd_node{
    int dim[2];
    bool operator<(const kd_node a)const{
        return dim[TYPE]<a.dim[TYPE];
    }
}tree[maxn],tmp[maxn];
priority_queue<ll,vector<ll>,greater<ll> >q;
void up(int x){
    for (int i=0;i<2;i++){
        mx[x][i]=mn[x][i]=tree[x].dim[i];
        if (ls[x]){
            mx[x][i]=max(mx[x][i],mx[ls[x]][i]);
            mn[x][i]=min(mn[x][i],mn[ls[x]][i]);
        }
        if (rs[x]){
            mx[x][i]=max(mx[x][i],mx[rs[x]][i]);
            mn[x][i]=min(mn[x][i],mn[rs[x]][i]);
        }
    }
}
void new_node(int &x,int ty,const kd_node rhs){
    x=++treeid;
    type[x]=ty;
    tree[x]=rhs;
```

```
    ls[x]=rs[x]=0;
    up(x);
}
void build(int &x,int l,int r){
    int mid=(l+r)>>1;
    TYPE=(++cnt)&1;
    nth_element(tmp+l,tmp+mid,tmp+r+1);
    new_node(x,TYPE,tmp[mid]);
    if (l<mid)build(ls[x],l,mid-1);
    if (mid<r)build(rs[x],mid+1,r);
    up(x);
}
bool check(int x,kd_node rhs){
    ll lenx=max(abs(mx[x][0]-rhs.dim[0]),abs(mn[x][0]-rhs.dim[0]));
    ll leny=max(abs(mx[x][1]-rhs.dim[1]),abs(mn[x][1]-rhs.dim[1]));
    if (lenx*lenx+leny*leny<=q.top())return 1;
    return 0;
}
ll calc(int x,kd_node rhs){
    ll lenx=max(abs(mx[x][0]-rhs.dim[0]),abs(mn[x][0]-rhs.dim[0]));
    ll leny=max(abs(mx[x][1]-rhs.dim[1]),abs(mn[x][1]-rhs.dim[1]));
    return lenx*lenx+leny*leny;
}
void query(int x,kd_node rhs){
    if (check(x,rhs))return;
    ll lenx=abs(tree[x].dim[0]-rhs.dim[0]);
    ll leny=abs(tree[x].dim[1]-rhs.dim[1]);
    ll res=lenx*lenx+leny*leny;
    if (res>q.top()){
        q.pop();q.push(res);
    }
    if (ls[x]&&!rs[x]){
        query(ls[x],rhs);
    }
    else if (!ls[x]&&rs[x]){
        query(rs[x],rhs);
    }
    else if (ls[x]&&rs[x]){
        if (calc(ls[x],rhs)>calc(rs[x],rhs)){
            query(ls[x],rhs);
            query(rs[x],rhs);
        }
        else{
            query(rs[x],rhs);
            query(ls[x],rhs);
        }
    }
}
int main() {
    n=rd();k=rd();
    for (int i=1;i<=n;i++)tmp[i].dim[0]=rd(),tmp[i].dim[1]=rd();
    build(root,1,n);
    for (int i=1;i<=k+k;i++)q.push(0);
    for (int i=1;i<=n;i++)query(root,tree[i]);
    printf("%lld\n",q.top());
    return 0;
}
```

18 Splay

例题普通平衡树-luoguP3369
操作 1：插入 x
操作 2：删除 x
操作 3：查询 x 的排名
操作 4：查询排名为 x 的数
操作 5：查询 x 的前驱
操作 6：查询 y 的前驱
查询 x 的前驱后继时，先插入 x 然后在左/右子树中查询最大/小的值即可
查询完再删除 x

```
const int maxn=1e3+5;
struct Splay{
    static const int maxn=1e6+5;
    int tot,rt,fa[maxn],ch[maxn][2],val[maxn],cnt[maxn],sz[maxn];
    void clear(int x){
        sz[x]=ch[x][0]=ch[x][1]=val[x]=cnt[x]=fa[x]=0;
    }
    void up(int x){
        sz[x]=sz[ch[x][0]]+sz[ch[x][1]]+cnt[x];
    }
    bool get(int x){return x==ch[fa[x]][1];}
    void rotate(int x){
        int y=fa[x],z=fa[y],chk=get(x);
        ch[y][chk]=ch[x][chk^1];
        if (ch[x][chk^1])fa[ch[x][chk^1]]=y;
        ch[x][chk^1]=y;
        fa[y]=x;fa[x]=z;
        if (z)ch[z][y==ch[z][1]]=x;
        up(y);up(x);
    }
    void splay(int x,int to){//把x转到to的位置
        to=fa[to];
        for (int f=fa[x];fa[x]!=to;rotate(x)){
            f=fa[x];
            if (fa[f]!=to)rotate(get(x)==get(f)?f:x);
        }
        if (!fa[x])rt=x;
    }
    void insert(int k){
        if (!rt){
            val[++tot]=k;
            cnt[tot]++;
            rt=tot;
            up(rt);
            return;
        }
        int cur=rt,f=0;
        while (1){
            if (val[cur]==k){
                cnt[cur]++;
                up(cur);up(f);
                splay(cur,rt);break;
            }
            f=cur;cur=ch[cur][val[cur]<k];
            if (!cur){
```

```
46                  val[++tot]=k;cnt[tot]++;
47                  fa[tot]=f;
48                  ch[f][val[f]<k]=tot;
49                  up(tot);up(f);
50                  splay(tot,rt);
51                  break;
52              }
53          }
54      }
55      int rk(int k){
56          int res=0,cur=rt;
57          while (1){
58              if (k<val[cur]){
59                  cur=ch[cur][0];
60              }
61              else{
62                  res+=sz[ch[cur][0]];
63                  if (k==val[cur]){
64                      splay(cur,rt);return res+1;
65                  }
66                  res+=cnt[cur];
67                  cur=ch[cur][1];
68              }
69          }
70      }
71      int kth(int k){
72          int cur=rt;
73          while (1){
74              if (ch[cur][0]&&k<=sz[ch[cur][0]]){
75                  cur=ch[cur][0];
76              }
77              else{
78                  k-=cnt[cur]+sz[ch[cur][0]];
79                  if (k<=0){
80                      splay(cur,rt);return val[cur];
81                  }
82                  cur=ch[cur][1];
83              }
84          }
85      }
86      int pre(){
87          int cur=ch[rt][0];
88          if (!cur)return cur;
89          while (ch[cur][1])cur=ch[cur][1];
90          splay(cur,rt);
91          return cur;
92      }
93      int nxt(){
94          int cur=ch[rt][1];
95          if (!cur)return cur;
96          while (ch[cur][0])cur=ch[cur][0];
97          splay(cur,rt);
98          return cur;
99      }
100     void del(int k){
101         rk(k);
102         if (cnt[rt]>1){
```

```
            cnt[rt]--;up(rt);return;
        }
        if (!ch[rt][0]&&!ch[rt][1]){
            clear(rt);rt=0;return;
        }
        if (!ch[rt][0]){
            int cur=rt;
            rt=ch[rt][1];
            fa[rt]=0;clear(cur);
            return;
        }
        if (!ch[rt][1]){
            int cur=rt;
            rt=ch[rt][0];
            fa[rt]=0;clear(cur);
            return;
        }
        int cur=rt,x=pre();
        fa[ch[cur][1]]=x;
        ch[x][1]=ch[cur][1];
        clear(cur);
        up(rt);
    }
}splay;
int q;
int main() {
    q=rd();
    while (q--){
        int op=rd(),x=rd();
        if (op==1)splay.insert(x);
        else if (op==2)splay.del(x);
        else if (op==3)printf("%d\n",splay.rk(x));
        else if (op==4)printf("%d\n",splay.kth(x));
        else if (op==5)splay.insert(x),printf("%d\n",splay.val[splay.
           pre()]),splay.del(x);
        else if (op==6)splay.insert(x),printf("%d\n",splay.val[splay.
           nxt()]),splay.del(x);
    }
    return 0;
}
```

例题文艺平衡树-luoguP3391
区间翻转
每次将 $r+1$ 提到根，再将 $l-1$ 提到根的左儿子
这样 $l-1$ 节点的右儿子即是要翻转的区间，打上反转标记，查询时标记下传即可

```
struct Splay{
    static const int maxn=1e6+5;
    int tot,rt,fa[maxn],ch[maxn][2],val[maxn],sz[maxn],flip[maxn],cnt
       [maxn];
    void clear(int x){
        sz[x]=ch[x][0]=ch[x][1]=val[x]=cnt[x]=fa[x]=0;
    }
    void up(int x){
        sz[x]=sz[ch[x][0]]+sz[ch[x][1]]+cnt[x];
    }
    void down(int x){
        if (!flip[x])return;
```

```
12          swap(ch[x][0],ch[x][1]);
13          if (ch[x][0])flip[ch[x][0]]^=1;
14          if (ch[x][1])flip[ch[x][1]]^=1;
15          flip[x]=0;
16      }
17      bool get(int x){return x==ch[fa[x]][1];}
18      void rotate(int x){
19          int y=fa[x],z=fa[y],chk=get(x);
20          ch[y][chk]=ch[x][chk^1];
21          if (ch[x][chk^1])fa[ch[x][chk^1]]=y;
22          ch[x][chk^1]=y;
23          fa[y]=x;fa[x]=z;
24          if (z)ch[z][y==ch[z][1]]=x;
25          up(y);up(x);
26      }
27      void splay(int x,int to){//把x转到to的位置
28          to=fa[to];
29          for (int f=fa[x];fa[x]!=to;rotate(x)){
30              f=fa[x];
31              if (fa[f]!=to)rotate(get(x)==get(f)?f:x);
32          }
33          if (!fa[x])rt=x;
34      }
35      void build(int &x,int f,int l,int r){
36          x=++tot;
37          int mid=(l+r)>>1;
38          fa[x]=f;val[x]=mid;cnt[x]++;
39          if (l<mid)build(ch[x][0],x,l,mid-1);
40          if (mid<r)build(ch[x][1],x,mid+1,r);
41          up(x);
42      }
43      int kth(int k){
44          int cur=rt;
45          while (1){
46              down(cur);
47              if (ch[cur][0]&&k<=sz[ch[cur][0]]){
48                  cur=ch[cur][0];
49              }
50              else{
51                  k-=cnt[cur]+sz[ch[cur][0]];
52                  if (k<=0)return cur;
53                  cur=ch[cur][1];
54              }
55          }
56      }
57      void reverse(int l,int r){
58          int x=kth(l-1),y=kth(r+1);
59          splay(y,rt);splay(x,ch[rt][0]);
60          flip[ch[x][1]]^=1;
61      }
62  }splay;
63  int n,m;
64  int main() {
65      n=rd();m=rd();
66      splay.build(splay.rt,0,0,n+1);
67      for (int i=1;i<=m;i++){
68          int l=rd(),r=rd();
```

```
69          splay.reverse(l+1,r+1);
70      }
71      for (int i=1;i<=n;i++)printf("%d ",splay.val[splay.kth(i+1)]);
72      return 0;
73  }
```

19 Link Cut Tree

```
1   const int maxn=1e5+5;
2   const int mod=51061;
3   #define int long long
4   int n,q,u,v,c;
5   char op;
6   struct LinkCutTree {
7     int ch[maxn][2], fa[maxn], siz[maxn], val[maxn], sum[maxn], rev[
        maxn],
8       add[maxn], mul[maxn];
9     void clear(int x) {
10      ch[x][0] = ch[x][1] = fa[x] = siz[x] = val[x] = sum[x] = rev[x] =
          add[x] =
11          0;
12      mul[x] = 1;
13    }
14    int get(int x) { return (ch[fa[x]][1] == x); }
15    int isroot(int x) {
16      clear(0);
17      return ch[fa[x]][0] != x && ch[fa[x]][1] != x;
18    }
19    void up(int x) {
20      clear(0);
21      siz[x] = (siz[ch[x][0]] + 1 + siz[ch[x][1]]) % mod;
22      sum[x] = (sum[ch[x][0]] + val[x] + sum[ch[x][1]]) % mod;
23    }
24    void pushdown(int x) {
25      clear(0);
26      if (mul[x] != 1) {
27        if (ch[x][0])
28          mul[ch[x][0]] = (mul[x] * mul[ch[x][0]]) % mod,
29          val[ch[x][0]] = (val[ch[x][0]] * mul[x]) % mod,
30          sum[ch[x][0]] = (sum[ch[x][0]] * mul[x]) % mod,
31          add[ch[x][0]] = (add[ch[x][0]] * mul[x]) % mod;
32        if (ch[x][1])
33          mul[ch[x][1]] = (mul[x] * mul[ch[x][1]]) % mod,
34          val[ch[x][1]] = (val[ch[x][1]] * mul[x]) % mod,
35          sum[ch[x][1]] = (sum[ch[x][1]] * mul[x]) % mod,
36          add[ch[x][1]] = (add[ch[x][1]] * mul[x]) % mod;
37        mul[x] = 1;
38      }
39      if (add[x]) {
40        if (ch[x][0])
41          add[ch[x][0]] = (add[ch[x][0]] + add[x]) % mod,
42          val[ch[x][0]] = (val[ch[x][0]] + add[x]) % mod,
43          sum[ch[x][0]] = (sum[ch[x][0]] + add[x] * siz[ch[x][0]] % mod
              ) % mod;
44        if (ch[x][1])
```

```
45          add[ch[x][1]] = (add[ch[x][1]] + add[x]) % mod,
46          val[ch[x][1]] = (val[ch[x][1]] + add[x]) % mod,
47          sum[ch[x][1]] = (sum[ch[x][1]] + add[x] * siz[ch[x][1]] % mod
              ) % mod;
48        add[x] = 0;
49      }
50      if (rev[x]) {
51        if (ch[x][0]) rev[ch[x][0]] ^= 1, swap(ch[ch[x][0]][0], ch[ch[x
              ][0]][1]);
52        if (ch[x][1]) rev[ch[x][1]] ^= 1, swap(ch[ch[x][1]][0], ch[ch[x
              ][1]][1]);
53        rev[x] = 0;
54      }
55    }
56    void update(int x) {
57      if (!isroot(x)) update(fa[x]);
58      pushdown(x);
59    }
60    void rotate(int x) {
61      int y = fa[x], z = fa[y], chx = get(x), chy = get(y);
62      fa[x] = z;
63      if (!isroot(y)) ch[z][chy] = x;
64      ch[y][chx] = ch[x][chx ^ 1];
65      fa[ch[x][chx ^ 1]] = y;
66      ch[x][chx ^ 1] = y;
67      fa[y] = x;
68      up(y);
69      up(x);
70      up(z);
71    }
72    void splay(int x) {
73      update(x);
74      for (int f = fa[x]; f = fa[x], !isroot(x); rotate(x))
75        if (!isroot(f)) rotate(get(x) == get(f) ? f : x);
76    }
77    void access(int x) {
78      for (int f = 0; x; f = x, x = fa[x]) splay(x), ch[x][1] = f, up(x
            );
79    }
80    void makeroot(int x) {
81      access(x);
82      splay(x);
83      swap(ch[x][0], ch[x][1]);
84      rev[x] ^= 1;
85    }
86    int find(int x) {
87      access(x);
88      splay(x);
89      while (ch[x][0]) x = ch[x][0];
90      splay(x);
91      return x;
92    }
93  }lct;
94  signed main() {
95    scanf("%lld%lld", &n, &q);
96    for (int i = 1; i <= n; i++) lct.val[i] = 1, lct.up(i);
97    for (int i = 1; i < n; i++) {
```

```
98      scanf("%lld%lld", &u, &v);
99      if (lct.find(u) != lct.find(v)) lct.makeroot(u), lct.fa[u] = v;
100   }
101   while (q--) {
102     scanf(" %c%lld%lld", &op, &u, &v);
103     if (op == '+') {//u-v路径上的点+c
104       scanf("%lld", &c);
105       lct.makeroot(u), lct.access(v), lct.splay(v);
106       lct.val[v] = (lct.val[v] + c) % mod;
107       lct.sum[v] = (lct.sum[v] + lct.siz[v] * c % mod) % mod;
108       lct.add[v] = (lct.add[v] + c) % mod;
109     }
110     if (op == '-') {//删边 并且加入一条新边
111       lct.makeroot(u);
112       lct.access(v);
113       lct.splay(v);
114       if (lct.ch[v][0] == u && !lct.ch[u][1]) lct.ch[v][0] = lct.fa[u
            ] = 0;
115       scanf("%lld%lld", &u, &v);
116       if (lct.find(u) != lct.find(v)) lct.makeroot(u), lct.fa[u] = v;
117     }
118     if (op == '*') {//u-v路径上的点*c
119       scanf("%lld", &c);
120       lct.makeroot(u), lct.access(v), lct.splay(v);
121       lct.val[v] = lct.val[v] * c % mod;
122       lct.sum[v] = lct.sum[v] * c % mod;
123       lct.mul[v] = lct.mul[v] * c % mod;
124     }
125     if (op == '/')//查询u-v路径和
126       lct.makeroot(u), lct.access(v), lct.splay(v), printf("%lld\n",
            lct.sum[v]);
127   }
128   return 0;
129 }
```

Part III
树上问题

20 树上倍增 +LCA

```
1  const int maxn=500010;
2  int head[maxn],tot,pre[maxn][22],dep[maxn],vis[maxn],dis[maxn];
3  struct nn{int v,w,nxt;} g[maxn<<1];
4  void add_edge(int u,int v,int w){g[++tot]={v,w,head[u]};head[u]=tot;}
5  void clear()
6  {
7      memset(head,0,sizeof(head));
8      memset(vis,0,sizeof(vis));
9      memset(dis,0,sizeof(dis));
10 }
11 void init(int u,int fa)
12 {
```

```
    dep[u]=dep[fa]+1;
    pre[u][0]=fa;
    for(int i=1;i<=20;i++){
        if ((1<<i)<=dep[u])pre[u][i]=pre[pre[u][i-1]][i-1];
        else pre[u][i]=0;
    }
    for(int i=head[u];i;i=g[i].nxt)
    {
        int v=g[i].v,w=g[i].w;
        if(v!=fa) {dis[v]=dis[u]+w;init(v,u);}
    }
}
int get_lca(int u,int v)
{
    if(dep[u]<dep[v]) swap(u,v);
    for(int i=20;i>=0;i--)
        if(dep[u]-(1<<i)>=dep[v]) u=pre[u][i];
    if(u==v) return u;
    for(int i=20;i>=0;i--)
        if(pre[u][i]!=pre[v][i]) {u=pre[u][i];v=pre[v][i];}
    return pre[u][0];
}
```

21 O(1) LCA

ST 表预处理 $O(nlogn)$
树需建双向遍
查询原理为 RMQ 故复杂度为 $O(1)$

```
void init(int u,int fa){
    dfn[++cnt]=u;
    dep[u]=dep[fa]+1;
    id[u]=cnt;
    for (int i=head[u];i;i=g[i].nxt){
        int v=g[i].v;
        if (v==fa)continue;
        init(v,u);
        dfn[++cnt]=u;
    }
}
void presolve(){
    for (int i=1;i<=cnt;i++)lg[i]=lg[i-1]+(1<<lg[i-1]==i);
    for (int i=1;i<=cnt;i++)f[i][0]=dfn[i];
    for (int j=1;(1<<j)<=cnt;j++){
        for (int i=1;i+(1<<j)-1<=tot;i++){
            int a=f[i][j-1],b=f[i+(1<<(j-1))][j-1];
            if (dep[a]<=dep[b])f[i][j]=a;
            else f[i][j]=b;
        }
    }
}
int LCA(int x,int y){
    if (id[x]<id[y])swap(x,y);
    int l=id[y],r=id[x];
    int k=lg[r-l+1]-1;
```

```
    int a=f[l][k],b=f[r-(1<<k)+1][k];
    if (dep[a]<=dep[b])return a;
    else return b;
}
///建双向边保持正确性
```

22 树链剖分 +LCA

轻重链剖分后，可以在 dfs 序上建维护区间信息的数据结构。
树剖自身复杂度 $O(logn)$

```
int sz[maxn],son[maxn],f[maxn],dep[maxn],dfn[maxn],top[maxn],dfsid,id
   [maxn];
void init(int u,int fa){//预处理size, 重儿子等信息
    sz[u]=1;son[u]=0;f[u]=fa;dep[u]=dep[fa]+1;
    for (int i=head[u];i;i=g[i].nxt){
        int v=g[i].v;
        if (v==fa)continue;
        init(v,u);
        sz[u]+=sz[v];
        if (sz[v]>sz[son[u]])son[u]=v;
    }
}
void chain(int u,int k){//预处理dfs序 chain(1,1)
    top[u]=k;
    dfn[++dfsid]=u;
    id[u]=dfsid;
    if (son[u])chain(son[u],k);
    for (int i=head[u];i;i=g[i].nxt){
        int v=g[i].v;
        if (v==f[u]||v==son[u])continue;
        chain(v,v);
    }
}
int LCA(int x,int y){
    while (top[x]!=top[y]){
        if (dep[top[x]]<dep[top[y]])swap(x,y);
        //若需维护链信息，可以在数据结构上对区间[id[topx],id[x]]进行
           修改
        x=f[top[x]];
    }
    if (dep[x]>dep[y])swap(x,y);
    //维护链信息还需要最后一段[id[x],id[y]]的区间更新/查询
    return x;
}
```

23 点分治

树重心的定义：一个点若所有的子树中最大的子树节点数最少，则该点为重心
每次选择重心 u 作为当前子树的根。
对于这颗子树来说，路径分为经过 u 和不经过 u。我们先处理经过 u 的路径，然后递归到每个子树中进行处理不经过 u 的路径。
洛谷 P3806

题意：给定一棵有 n 个点的树，询问树上距离为 k 的点对是否存在。
思路：点分治。每次对于当前树都找到重心，然后对于每个重心都查询一次更新一次。每次更新到 cnt 数组上。
cnt[i]: 到重心长为 i 的路径是否存在。

```
const int maxn=1e4+5;
const ll mod=998244353;
int head[maxn],tot,n,m,all;
struct nn{int v,w,nxt;}g[maxn<<1];
void add_edge(int u,int v,int w){g[++tot]={v,w,head[u]};head[u]=tot;}
int root,vis[maxn],sz[maxn],mx[maxn];
int cnt[10000005],ans[maxn],Q[maxn];
void getroot(int u,int fa){//求重心
        sz[u]=1;mx[u]=0;
        for (int i=head[u];i;i=g[i].nxt){
                int v=g[i].v,w=g[i].w;
                if (v==fa||vis[v])continue;
                getroot(v,u);
                sz[u]+=sz[v];
                if (sz[v]>mx[u])mx[u]=sz[v];
        }
        mx[u]=max(mx[u],all-sz[u]);
        if (mx[u]<mx[root]){
                root=u;
        }
}
void update(int u,int fa,int d,int va){//去除vis[v]
        if (d<=10000000)cnt[d]+=va;
        for (int i=head[u];i;i=g[i].nxt){
                int v=g[i].v,w=g[i].w;
                if (v==fa||vis[v])continue;
                update(v,u,d+w,va);
        }
}
void query(int u,int fa,int d){//去除vis[v]
        for (int i=1;i<=m;i++)
                if (Q[i]-d>=0&&Q[i]-d<=10000000&&cnt[Q[i]-d])ans[i
                   ]=1;
        for (int i=head[u];i;i=g[i].nxt){
                int v=g[i].v,w=g[i].w;
                if (vis[v]||v==fa)continue;
                query(v,u,d+w);
        }
}
void solve(int u){
        cnt[0]=1;
        for (int i=head[u];i;i=g[i].nxt){
                int v=g[i].v,w=g[i].w;
                if (vis[v])continue;
                query(v,u,w);
                update(v,u,w,1);//动态更新
        }
        update(u,0,0,-1);//清空cnt数组
}
void divide(int u){
        solve(u);//关键在于solve的使用
        vis[u]=1;
```

```
        for (int i=head[u];i;i=g[i].nxt){
                int v=g[i].v,w=g[i].w;
                if (vis[v])continue;
                all=sz[v];root=0;
                getroot(v,u);
                divide(root);
        }
}
int main() {
    n=rd();m=rd();
    for (int i=1,u,v,w;i<n;i++){
        u=rd();v=rd();w=rd();
        add_edge(u,v,w);
        add_edge(v,u,w);
    }
    for (int i=1;i<=m;i++)Q[i]=rd();
    all=n;root=0;mx[root]=n;
    getroot(1,0);
    divide(root);
    for (int i=1;i<=m;i++)printf("%s\n",ans[i]?"AYE":"NAY");
    return 0;
}
```

24 树上启发式合并

预处理出整个树每个节点的重儿子。
dfs 递归的时候，每次保留重儿子信息，轻儿子则要更新后删除。
一个点到根的路径上，最多会经过 log 条轻边，故单点被增加删除的次数为 log
故裸的 Dsu On Tree 自身复杂度 $O(nlogn)$

```
ll n,son[maxn],sz[maxn];
void init(int u,int fa){
        son[u]=0;sz[u]=1;
        for (int i=head[u];i;i=g[i].nxt){
                int v=g[i].v;
                if (v==fa)continue;
                init(v,u);
                sz[u]+=sz[v];
                if (sz[v]>sz[son[u]])son[u]=v;
        }
}
void dfs(int u,int fa,int va,int rt){
        //更新当前点
        for (int i=head[u];i;i=g[i].nxt){
                int v=g[i].v;
                if (v==fa)continue;
                dfs(v,u,va,rt);
        }
}
void dsu(int u,int fa,int op){
        for (int i=head[u];i;i=g[i].nxt){
                int v=g[i].v;
                if (v==son[u]||v==fa)continue;
                dsu(v,u,1);
        }
```

```
        if (son[u]!=0){
                dsu(son[u],u,0);
                //递归完成重儿子子树且保留
        }
        for (int i=head[u];i;i=g[i].nxt){
                int v=g[i].v;
                if (v==son[u]||v==fa)continue;
                dfs(v,u,1,u);
        }//合并所有轻儿子子树
        //合并当前根节点
        if (op){
                //去除当前点贡献
                for (int i=head[u];i;i=g[i].nxt){
                        int v=g[i].v;
                        if (v==fa)continue;
                        dfs(v,u,-1,0);
                }//去除所有子树贡献
        }
}
```

25 虚树

对每次询问的点建虚树，虚树是包含查询点，根节点和查询点间的 lca 的最小树

```
while (m--){
                for (int i=1;i<=k;i++)a[i]=rd();
            sort(a+1,a+k+1,cmp);//按dfs序排序
            s[tp=1]=1;
            for (int i=1;i<=k;i++){
                int lca=get_lca(s[tp],a[i]);
                while (tp>1&&d[lca]<=d[s[tp-1]]){
                    add_edge1(s[tp-1],s[tp]);tp--;
                }
                if (s[tp]!=lca)add_edge1(lca,s[tp]),s[tp]=lca;
                if (s[tp]!=a[i])s[++tp]=a[i];
            }
            while (tp>1)add_edge1(s[tp-1],s[tp]),tp--;
            ans=0;
            dfs(1,0);
            printf("%d\n",ans);
}
```

Part IV 数学

26 组合数

版本 1 $O(n)$ 预处理,$O(1)$ 求 10^7 内组合数

```
namespace Com{
    static const int SIZE=1e6+5;
```

```
    static const int mod=1e9+7;//注意模数修改
    ll fac[SIZE],inv[SIZE];
    void init(){
        fac[0]=1;
        for(int i=1;i<SIZE;i++)fac[i]=fac[i-1]*i%mod;
        inv[SIZE-1]=ksm(fac[SIZE-1],mod-2);
        for(int i=SIZE-2;i>=0;i--)inv[i]=inv[i+1]*(i+1)%mod;
    }
    ll comb(int n,int m){
        if(n<0||m<0||n<m)return 0;
        return 1ll*fac[n]*inv[m]%mod*inv[n-m]%mod;
    }
}
```

版本 2 $O(n^2)$ 预处理组合数（小范围时常数较好）

```
ll c[maxn][maxn];
void init()
{
    c[0][0]=c[1][0]=c[1][1]=1;
    for(int i=2;i<maxn;++i){
        c[i][0]=1;
        for(int j=1;j<maxn;++j){
            c[i][j]=(c[i-1][j-1]+c[i-1][j])%mod;
        }
    }
}
```

27 数论分块

复杂度 $O(\sqrt{n})$

```
int ans = 0;
for(int l = 1, r = 0; l <= n; l=r+1) {
        r = n / (n / l);
        // do something
}
```

28 线性筛

$O(n)$ 复杂度预处理出每个数的最小质因子和范围内所有的质数

```
const int M=1e7+5;
int pre[M],prime[M],ptot;
void init(){
    for (int i=2;i<M;i++){
        if (pre[i]==0)
            prime[++ptot]=pre[i]=i;
        for (int j=1;j<=ptot;j++){
            int t =i*prime[j];
            if (t>=M)
                break;
            pre[t]=prime[j];
            if (i%prime[j]==0)
```

```
                break;
            }
        }
    }
```

29 线性基

29.1 线性基插入

```
int p[60];
void insert(int x){
    for (int i=50;i+1;i--){
        if (!(x>>i))continue;
        if (!p[i]){
            p[i]=x;break;
        }
        x^=p[i];
    }
}
```

合并线性基：把某个线性基的全部元素插入到另一个线性基里即可，复杂度 $O(log^2)$

29.2 求最值

最大值

```
int querymx(){
                int x=0;
    for (int i=30;i+1;i--){
        if (!p[i])continue;
        if ((x^p[i])>x)x^=p[i];
    }
    return x;
}
```

最小值
若在建线性基过程中出现不能插入的数，则答案为 0。否则取最小的线性基

29.3 一些性质

设：
mx(x) 是 x 与线性基异或的最大值
mn(x) 是 x 与线性基异或的最小值
mx(x1)⨁mx(x2)=mn(x1⨁x2)
mx(x1)⨁mx(x2)⨁mx(x3)=mx(x1⨁x2⨁x3)
mn(x1)⨁mn(x2)=mn(x1⨁x2)

30 欧拉降幂

$a^b \equiv a^{b\%\phi(p)}, gcd(a,p)=1 \quad (mod\ \ p)$

31 质因数分解 + 素数判断

质因数分解复杂度 $O(\sqrt[4]{n})$
素数判断复杂度 $O(8logn)$
此素数判断模板仅能保证 long long 范围内的素数判断不会出错
质因数找到的顺序按照从小到大

```
ll ksm(ll a,ll b,ll mod){ll ans=1;a%=mod;while(b){if(b&1)ans=(ans*a)%mod;a=(a*a)%mod;b>>=1;}return ans;}
inline ll mul(ll a,ll b,ll mod){
    return (__int128)a*b%mod;
}
bool miller_rabin(ll a,ll n){
    ll d=n-1,r=0;
    while (d%2==0)d/=2,r++;
    ll x=ksm(a,d,n);
    if (x==1)return true;
    for (int i=0;i<r;i++){
        if (x==n-1)return true;
        x=(ll)x*x%n;
    }
    return false;
}
bool is_prime(ll n){
    if (n<=1)return false;
    ll num[10]={0,2,3,5,7,13,29,37,89};
    for (int a=1;a<=8;a++)
        if (n==num[a])return true;
    for (int a=1;a<=8;a++)
        if (!miller_rabin(num[a],n))return false;
    return true;
}
//Miller_Rabin素数判断部分
ll fun(ll x,ll c,ll mod){
    return (mul(x,x,mod)+c)%mod;
}
ll gcd(ll n,ll m){
    if(m==0)return n;
    return gcd(m,n%m);
}
ll pollard_rho(ll x){
    ll c=rand()%(x-1)+1;
    ll a=0,b=c;
    while(a!=b){
        ll d=gcd(abs(a-b),x);
        if(d>1&&d<x)return d;
        a=fun(a,c,x),b=fun(fun(b,c,x),c,x);
    }
    return x;
}
int tot,a[maxn],b[maxn];
void find_fac(ll x){
    if(x==1)return;
    if(is_prime(x)){
        //找到了质因子x
        return;
    }
```

```
    ll y=x;
    while(y==x)y=pollard_rho(x);
    find_fac(y),find_fac(x/y);
}
```

32 矩阵类

32.1 矩阵乘法 + 矩阵快速幂

（已并行优化版本）

```
struct Matrix{
        static const int N=105,M=105;
        ll a[N][M];
        Matrix(ll e=0){
                for (int i=1;i<=n;i++)for (int j=1;j<=n;j++)a[i][j]=e
                   *(i==j);
        }
        Matrix mul(Matrix A,Matrix B){
                Matrix ans(0);
                for (int i=1;i<=n;i++){
                        for (int k=1;k<=n;k++){
                                for (int j=1;j<=n;j++){
                                        ans.a[i][j]=(ans.a[i][j]+A.a[
                                           i][k]*B.a[k][j])%mod;
                                }
                        }
                }
                return ans;
        }
        Matrix ksm(Matrix A,ll b){
                Matrix ans(1);
                while (b){
                        if (b&1)ans=mul(ans,A);
                        A=mul(A,A);b>>=1;
                }
                return ans;
        }
}tmp;
//用法参见 tmp=tmp.ksm(tmp,b);
```

32.2 高斯消元 & 矩阵求逆

取模版

```
struct Matrix{
        static const int N=405,M=805;
        ll a[N][M];
        void r_div(int x,int m,ll k){// a[x][]/=k
                ll inv=ksm(k,mod-2);
                for (int i=1;i<=m;i++)a[x][i]=a[x][i]*inv%mod;
        }
        void r_plus(int x,int y,int m,ll k){// a[x][]+=a[y][]*k
                for (int i=1;i<=m;i++)a[x][i]=(a[x][i]+a[y][i]*k)%mod
                   ;
```

```
        }
        bool gauss(int n,int m){//返回矩阵是否可消元
                for (int i=1;i<=n;i++){
                        int t=-1;
                        for (int j=i;j<=n;j++)if (a[j][i]){t=j;break;}
                        if (t==-1)return 0;
                        if (t!=i)for (int j=1;j<=m;j++)swap(a[i][j],a[t][j]);
                        r_div(i,m,a[i][i]);
                        for (int j=1;j<=n;j++)if (j!=i&&a[j][i])r_plus(j,i,m,mod-a[j][i]);
                }
                return 1;
        }
        bool get_inv(int n){//同高斯消元
                for (int i=1;i<=n;i++)for (int j=1;j<=n;j++)a[i][j+n]=(i==j);
                bool ans=gauss(n,n<<1);
                for (int i=1;i<=n;i++)for (int j=1;j<=n;j++)a[i][j]=a[i][j+n];
                return ans;
        }
}tmp;
```

32.3 行列式计算

```
struct Matrix{
    static const int N=605;
    ll a[N][N];
    ll det(int n){
        ll ans=1;
        for (int i=1;i<=n;i++){
            for (int j=i+1;j<=n;j++){
                while (a[j][i]){
                    ll t=a[i][i]/a[j][i];
                    for (int k=i;k<=n;k++)a[i][k]=(a[i][k]-a[j][k]*t)%mod;
                    for (int k=1;k<=n;k++)swap(a[i][k],a[j][k]);
                    ans=-ans;
                }
            }
            ans=ans*a[i][i]%mod;
            if (!ans)return 0;
        }
        return (ans+mod)%mod;
    }
}tmp;
```

33 自适应辛普森积分

利用二次函数的积分来近似需要的积分，表达式为：$\int_l^r f(x)dx = \frac{(r-l)*(f(l)+f(r)+4f(\frac{l+r}{2}))}{6}$
利用递归计算积分，根据误差大小判断是否继续划分

调用时：“double res=Simpson(l,r,simpson(l,r),1e-6);“

```
//调用方式 Simpson(l,r,simpson(l,r),eps);
inline double fun(double x){
        return (0.0);//需要积分的函数
}
inline double simpson(double l,double r){
        return (r-l)*(fun(l)+fun(r)+4*fun((l+r)/2))/6;
}
double Simpson(double l,double r,double res,double eps){
        double mid=(l+r)/2;
        double L=simpson(l,mid);
        double R=simpson(mid,r);
        if(fabs(L+R-res)<=15*eps)return L+R+(L+R-res)/15;
        return Simpson(l,mid,L,eps/2)+Simpson(mid,r,R,eps/2);
}
```

34 快速傅立叶变换（FFT）

例题：给定 n 次多项式 $F(x)$，m 次多项式 $G(x)$，求 $F(x)$ 与 $G(x)$ 的卷积
问题可转化为给定两个长度分别为 n,m 的序列 A,B
求 $C_i = \sum_{j+k=i} a_j * b_k$

```
//  从低到高输入F(x) G(x)系数，同样方式输出卷积的全部系数
const int maxn=4e6+5;
const double pi=acos(-1.0);
int n,m;
int limit,bit;
int r[maxn<<1];
struct comp{
    double x,y;
    comp (double xx=0,double yy=0){x=xx;y=yy;}
}f[maxn],g[maxn];
comp operator + (comp a,comp b){return comp(a.x+b.x,a.y+b.y);}
comp operator - (comp a,comp b){return comp(a.x-b.x,a.y-b.y);}
comp operator * (comp a,comp b){return comp(a.x*b.x-a.y*b.y,a.x*b.y+a
   .y*b.x);}
void FFT_init(int x){
    limit=1;bit=0;
    while (limit<=x)limit<<=1,bit++;
    for (int i=0;i<limit;i++)
    r[i]=(r[i>>1]>>1)|((i&1)<<(bit-1));
}
void FFT(comp a[],int type){
    for (int i=0;i<limit;i++)
        if (i<r[i])swap(a[i],a[r[i]]);
    for (int mid=1;mid<limit;mid<<=1){
        comp wn(cos(pi/mid),type*sin(pi/mid));
        for (int r=mid<<1,j=0;j<limit;j+=r){
            comp w(1,0);
            for (int k=0;k<mid;k++,w=w*wn){
                comp x=a[j+k],y=w*a[j+mid+k];
                a[j+k]=x+y;
                a[j+mid+k]=x-y;
            }
```

```
        }
    }
}
int main() {
    n=rd();m=rd();
    for (int i=0;i<=n;i++)f[i]=rd();
    for (int i=0;i<=m;i++)g[i]=rd();
    FFT_init(n+m);
    FFT(f,1);FFT(g,1);
    for (int i=0;i<=limit;i++)f[i]=f[i]*g[i];
    FFT(f,-1);
    for (int i=0;i<=n+m;i++)printf("%d ",(int)(f[i].x/limit+0.5));
    return 0;
}
```

34.1 分治 FFT

给定 $g_1 \ldots g_{n-1}$，求 $f_0 \ldots f_{n-1}$
$f_i = \sum_{j=1}^{i} f_{i-j} g_j$，边界为 $f_0 = 1$
答案取模 998244353
考虑 CDQ 分治
当前区间为 $[l,r]$ 时，处理 $[l,mid]$ 对 $[mid+1,r]$ 的贡献。
对前半段区间做 FFT，再将贡献计算到后半段区间上。
复杂度 $O(nlog^2)$

```
void CDQ(int l,int r){
    if (l==r)return;
    int mid=(l+r)>>1;
    CDQ(l,mid);
    NTT::work(f+l,mid-l+1,g+1,r-l);
    for (int i=mid+1;i<=r;i++)f[i]=(f[i]+NTT::f[i-l-1])%mod;
    CDQ(mid+1,r);
}
```

35 快速沃尔什变换（FWT）

问题可转化为给定两个长度分别为 n,m 的序列 A,B
求 $C_i = \sum_{j \oplus k = i} a_j * b_k$ ，其中 $\oplus$ 为位运算符

```
const int maxn=1<<17|1;
const ll mod=998244353;
const ll inv2=499122177;//不同模数记得取不同的逆元
ll n,a[maxn],b[maxn],f[maxn],g[maxn];
void in(){
        for (int i=0;i<n;i++)f[i]=a[i],g[i]=b[i];
}
void out(){
        for (int i=0;i<n;i++)printf("%lld%c",f[i],i==n-1?'\n':' ');
}
void get(){
        for (int i=0;i<n;i++)f[i]=f[i]*g[i]%mod;
}
void FWT(ll *f, int op,int n) {
    for(int len=2; len<=n; len<<=1) {
```

```
        for(int l=0, hf=len>>1; l<n; l+=len) {
            for(int i=l; i<l+hf; ++i) {
                ll x=f[i], y=f[i+hf];
                if(op>0) {
                    if(op==1) f[i]=(x+y)%mod, f[i+hf]=(x-y+mod)%mod; //xor
                    else if(op==2) f[i]=(x+y)%mod; //and
                    else f[i+hf]=(x+y)%mod; //or
                }
                else {
                    if(op==-1) f[i]=(x+y)*inv2%mod, f[i+hf]=(x-y+mod)*inv2%mod; //xor
                    else if(op==-2) f[i]=(x-y+mod)%mod; //and
                    else f[i+hf]=(y-x+mod)%mod; //or
                }
            }
        }
    }
}
int main() {
    n=rd();
    n=1ll<<n;
    for (int i=0;i<n;i++)a[i]=rd();
    for (int i=0;i<n;i++)b[i]=rd();
    in();FWT(f,3);FWT(g,3);get();FWT(f,-3);out();
    in();FWT(f,2);FWT(g,2);get();FWT(f,-2);out();
    in();FWT(f,1);FWT(g,1);get();FWT(f,-1);out();
    return 0;
}
```

35.1 分治 FWT

某类 DP 方程转移式可能有以下形式：
$dp[i]=\sum_{j>i} dp[j]\oplus a[i\oplus j]$
转移项 $j<i$ *or* $j>i$ 可以运用 CDQ 分治套 FWT 的思路
以下假设 $j>i$
处理到当前区间 [l,r] 时，先处理 [mid+1,r]，对 [mid+1,r] 的元素和对应 a 数组区间的元素进行 FWT 卷积
时间复杂度 $O(nlogn^2)$

```
void CDQ(int l,int r){
    if (l==r){
        dp[l]=dp[l]*ksm(f[l],mod-2)%mod;//若式子有分母则在此处除尽
        return;
    }
    int mid=(l+r)>>1;
    CDQ(mid+1,r);
    ll tmp=0;
    for (int i=1;i<=r-mid;i++)c[i-1]=dp[mid+i],d[i-1]=g[mid+i-l],tmp=(tmp+g[mid+i-l])%mod;//用新数组跑FWT变换
    FWT(c,1,r-mid);FWT(d,1,r-mid);
    for (int i=1;i<=r-mid;i++)c[i-1]=c[i-1]*d[i-1]%mod;
    FWT(c,-1,r-mid);
    for (int i=l;i<=mid;i++)dp[i]=(dp[i]+c[i-l])%mod,f[i]=(f[i]+tmp)%mod;//计算贡献
    CDQ(l,mid);
```

```
}
```

36 快速数论变换 (NTT)

36.1 普通 NTT

可以跑例如 998244353 模数

```
// array [0, n)
namespace NTT {
  static const int SIZE = (1<<21)+3;
  const int G = 3;
  int MOD=mod;
  int len, bit;
  int rev[SIZE];
  long long f[SIZE], g[SIZE];
  template <class T>
  void ntt(T a[], int flag = 1) {
    for (int i = 0; i < len; ++i)
      if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int base = 1; base < len; base <<= 1) {
      long long wn = ksm(G, (MOD-1)/(base*2)), w;
      if (flag == -1) wn = ksm(wn, MOD-2);
      for (int i = 0; i < len; i += base*2) {
        w = 1;
        for (int j = 0; j < base; ++j) {
          long long x = a[i+j], y = w*a[i+j+base]%MOD;
          a[i+j] = (x+y)%MOD;
          a[i+j+base] = (x-y+MOD)%MOD;
          w = w*wn%MOD;
        }
      }
    }
  }
  template <class T>
  void work(T a[], const int &n, T b[], const int &m) {
    len = 1; bit = 0;
    while (len < n+m) len <<= 1, ++bit;
    for (int i = 0; i < n; ++i) f[i] = a[i];
    for (int i = n; i < len; ++i) f[i] = 0;
    for (int i = 0; i < m; ++i) g[i] = b[i];
    for (int i = m; i < len; ++i) g[i] = 0;
    for (int i = 0; i < len; ++i)
      rev[i] = (rev[i>>1]>>1)|((i&1)<<(bit-1));
    ntt(f, 1); ntt(g, 1);
    for (int i = 0; i < len; ++i) f[i] = f[i]*g[i]%MOD;
    ntt(f, -1);
    long long inv = ksm(len, MOD-2);
    for (int i = 0; i < n+m-1; ++i) f[i] = f[i]*inv%MOD;
  }
}
```

36.2 任意模数 NTT

```
const double PI=acos(-1.0);
struct comp {
  typedef double T; // maybe long double
  T real, imag;
  comp (const double &_real = 0, const double &_imag = 0) : real(_real), imag(_imag) {}
  friend comp operator + (const comp &c1, const comp &c2) { return comp(c1.real+c2.real, c1.imag+c2.imag); }
  friend comp operator - (const comp &c1, const comp &c2) { return comp(c1.real-c2.real, c1.imag-c2.imag); }
  friend comp operator * (const comp &c1, const comp &c2) { return comp(c1.real*c2.real-c1.imag*c2.imag, c1.real*c2.imag+c1.imag*c2.real); }
  comp& operator += (const comp &c) { return *this = *this+c; }
  comp& operator -= (const comp &c) { return *this = *this-c; }
  comp& operator *= (const comp &c) { return *this = *this*c; }
  friend istream& operator >> (istream &is, comp &c) { return is >> c.real >> c.imag; }
  friend ostream& operator << (ostream &os, comp &c) { return os << c.real << setiosflags(ios::showpos) << c.imag << "i";}
  comp conjugate() { return comp(real, -imag); }
  friend comp conjugate(const comp &c) { return comp(c.real, -c.imag); }
};

namespace MTT {
  static const int SIZE = (1<<21)+7;//多项式最大次数
  int Mod = mod;
  comp w[SIZE];
  int bitrev[SIZE];
  long long f[SIZE];
  void fft(comp *a, const int &n) {
    for (int i = 0; i < n; ++i) if (i < bitrev[i]) swap(a[i], a[bitrev[i]]);
    for (int i = 2, lyc = n >> 1; i <= n; i <<= 1, lyc >>= 1)
      for (int j = 0; j < n; j += i) {
        comp *l = a + j, *r = a + j + (i >> 1), *p = w;
        for (int k = 0; k < i>>1; ++k) {
          comp tmp = *r * *p;
          *r = *l - tmp, *l = *l + tmp;
          ++l, ++r, p += lyc;
        }
      }
  }
  template <class T>
  inline void work(T *x, const int &n, T *y, const int &m) {
    static int bit, L;
    static comp a[SIZE], b[SIZE];
    static comp dfta[SIZE], dftb[SIZE];

    for (L = 1, bit = 0; L < n+m-1; ++bit, L <<= 1);
    for (int i = 0; i < L; ++i) bitrev[i] = bitrev[i >> 1] >> 1 | ((i & 1) << (bit - 1));
    for (int i = 0; i < L; ++i) w[i] = comp(cos(2 * PI * i / L), sin(2 * PI * i / L));

```

```
46      for (int i = 0; i < n; ++i) (x[i] += Mod) %= Mod, a[i] = comp(x[i
            ] & 32767, x[i] >> 15);
47      for (int i = n; i < L; ++i) a[i] = 0;
48      for (int i = 0; i < m; ++i) (y[i] += Mod) %= Mod, b[i] = comp(y[i
            ] & 32767, y[i] >> 15);
49      for (int i = m; i < L; ++i) b[i] = 0;
50      fft(a, L), fft(b, L);
51      for (int i = 0; i < L; ++i) {
52        int j = (L - i) & (L - 1);
53        static comp da, db, dc, dd;
54        da = (a[i] + conjugate(a[j])) * comp(.5, 0);
55        db = (a[i] - conjugate(a[j])) * comp(0, -.5);
56        dc = (b[i] + conjugate(b[j])) * comp(.5, 0);
57        dd = (b[i] - conjugate(b[j])) * comp(0, -.5);
58        dfta[j] = da*dc + da*dd*comp(0, 1);
59        dftb[j] = db*dc + db*dd*comp(0, 1);
60      }
61      for (int i = 0; i < L; ++i) a[i] = dfta[i];
62      for (int i = 0; i < L; ++i) b[i] = dftb[i];
63      fft(a, L), fft(b, L);
64      for (int i = 0; i < L; ++i) {
65        int da = (long long)(a[i].real / L + 0.5) % Mod;
66        int db = (long long)(a[i].imag / L + 0.5) % Mod;
67        int dc = (long long)(b[i].real / L + 0.5) % Mod;
68        int dd = (long long)(b[i].imag / L + 0.5) % Mod;
69        f[i] = (da + ((long long)(db + dc) << 15) + ((long long)dd <<
              30)) % Mod;
70      }
71      for (int i = 0; i < n+m-1; ++i) (f[i] += Mod) %= Mod;
72    }
73  }
```

37 博弈论

37.1 SG 函数计算方法

一个局面的 SG 为 mex SG，mex 运算为集合中未出现的最小自然数。
整体 SG 为所有的 SG 的异或和。
先手必胜：$SG \neq 0$

37.2 Nim Game

n 堆石子，两个人轮流从某一堆中取走任意个石子，不能取者败
先手必败：$a_1 \oplus a_2 \oplus ...a_n = 0$

37.3 Nim-k Game

每次最多从 k 堆石子中取任意个石子，这 k 堆取的石子可不同。
每一堆石子的 SG 函数为石子数。对每一个二进制位单独考虑，若每一个二进制位 1 的个数 $mod(k+1) = 0$ 则必败，反之必胜。

37.4 Staircase Nim

n 个阶梯，限制每次从上一个阶梯移到下一个阶梯

仅考虑奇数位置进行 Nim 博弈即可，因为对偶数位置操作存在平衡操作使得状态不变。

37.5 Anti-SG

先手必胜：
$SG \neq 0, \quad SG > 1$
$SG = 0, \quad SG > 1$

37.6 Multi-SG(lasker's Game)

n 堆石子，两个人轮流操作，每次操作可以选择某一堆中取走任意个石子或者选择一堆至少为 2 的石子分成两堆非空石子。

$$SG(x) = \begin{cases} x-1 \ x \ \ mod \ 4 = 0 \\ x \ x \ \ mod \ 4 = 1 \cup 2 \\ x+1, x \ \ mod \ 4 = 3 \end{cases}$$

37.7 Every-SG

游戏规定任意未结束的单一游戏，游戏者必须对它进行决策，游戏结束时间取决于最后一个结束的单一游戏，胜负取决于最后一个结束的单一游戏的输赢。
贪心来看：先手必胜的尽量长，先手必败的尽量短。
所以需要知道 SG 为 0 的局势最少多少步结束，SG 不为 0 的局势最多多少步结束。
用 $step$ 函数来记录

$$step(u) = \begin{cases} 0, u \\ max\{step(v)\} + 1 \ sg(u) \neq 0 \cap v \ u \quad \cap sg(v) = 0 \\ min\{step(v)\} + 1, sg(u) = 0 \cap v \ u \end{cases}$$

显然对于必胜的局势 u,$step(u)$ 为奇数
定理：对于 Every-SG 游戏先手必胜当且仅当单一游戏中最大的 $step$ 为奇数。

37.8 Bash Game

每次最多取 m 个石子，其他同 Nim。一堆石子的 SG 函数为石子数 $mod(m+1)$.
必胜：$SG \neq 0$；必败：$SG = 0$

37.9 Wythoff Game

两堆石子，每人每次可以拿走任意一堆中任意数量石子或在两堆石子中拿走相同数量的石子，不能拿的人输。
定义先手必败的局势为奇异局势。
在平面直角坐标系上，两堆石子的数量对应一个点的横纵坐标。
游戏变成每次操作可以向左或向下或向左下走任意格。
显然 $(0,0)$ 为必败点
每一次从必败态点出发将能到达的点染色，然后找到横纵坐标和最小的点，即下一个必败态点。
对于第 k 个奇异局势 (x,y),x 为前 0 k-1 个奇异局势中没有出现过的最小自然数,y=x+k;
第 k 个奇异局势是 $(\lfloor \frac{1+\sqrt{5}}{2}k \rfloor, \lfloor \frac{3+\sqrt{5}}{2}k \rfloor)$
**EX. 扩展威佐夫博弈 **
每次可以拿走任意一堆中任意数量石子或从两堆石子中拿走数量之差不超过 d 的石子
游戏中向左下移动变成向左移动 dx, 向右移动 dy, 要求 $|dx - dy| < d$
对于第 k 个奇异局势 (x,y),x 为前 $0 \ k-1$ 个奇异局势中没有出现过的最小自然数,$y = x + d * k$;
第 k 个奇异局势是 $(\lfloor \frac{2-d+\sqrt{d^2+4}}{2}k \rfloor, \lfloor \frac{2+d+\sqrt{d^2+4}}{2}k \rfloor)$

$Betty$ 定理：a,b 为无理数且 $\frac{1}{a}+\frac{1}{b}=1$，则数列 $\{\lfloor an\rfloor\}\{\lfloor bn\rfloor\}$ 无交集且覆盖正整数集合
威佐夫博弈推导方法：
运用 $Betty$ 定理，因为 $\frac{1}{a}+\frac{1}{b}=1$，再令式子 $y=x+k\to\lfloor bn\rfloor=\lfloor an\rfloor+n$，最后求解 a,b 即可。

37.10 斐波那契博弈

一堆石子，第一次操作可以拿任意个但不能全拿走且不能不拿，之后每次操作最少拿一个，最多拿前一次操作两倍。谁不能取谁输。
石子数为斐波那契数时先手必败，反之必胜。

37.11 k 倍动态减法博弈

37.12 树上删边游戏

给出 n 个点的树，两个人轮流删除边，删除后，不与根相连部分将被移走。
叶子节点 SG 值为 0，中间节点 SG 值为所有孩子 $SG+1$ 的异或和

37.13 无向图删边游戏 *

37.14 翻硬币游戏 *

Part V
图论

38 Dijkstra 堆优化

```
ll n,m,s,d[maxn],vis[maxn];
struct node{
    ll id,dis;
    bool operator<(const node a)const{
        return dis>a.dis;
    }
};
void dijkstra(int s){
    priority_queue<node>q;
    q.push({s,0});
    for (int i=1;i<=n;i++)d[i]=1e18;
    d[s]=0;
    while (!q.empty()){
        node e1=q.top();
        q.pop();
        if (vis[e1.id])continue;
        vis[e1.id]=1;
        for (int i=head[e1.id];i;i=g[i].nxt){
            int v=g[i].v,w=g[i].w;
            if (d[v]>d[e1.id]+w){
                d[v]=d[e1.id]+w;
                q.push({v,d[v]});
            }
        }
```

```
    }
}
```

39 网络流

39.1 最大流 Dinic

```
const int max_v=100005;
#define INF 0x3f3f3f3f
struct edge{
    int to,cap,rev;//终点 容量 反向边
};
vector<edge> G[max_v];//图的邻接表表示
int level[max_v];//顶点到源点的距离标号
int iter[max_v];//当前弧，在其之前的边已经没有用了
//给图中增加一条从from到tod容量为cap的边
void add_edge(int from,int to,int cap){
    G[from].push_back((edge){to,cap,(int)G[to].size()});
    G[to].push_back((edge){from,0,(int)G[from].size()-1});
}
//通过BFS计算从源点出发的距离标号
void bfs(int s){
    memset(level,-1,sizeof(level));
    queue<int>q;
    level[s]=0;
    q.push(s);
    while (!q.empty()){
        int v=q.front();
        q.pop();
        for (int i=0;i<G[v].size();i++){
            edge &e=G[v][i];
            if (e.cap>0&&level[e.to]<0){
                level[e.to]=level[v]+1;
                q.push(e.to);
            }
        }
    }
}
//通过DFS寻找增广路
int dfs(int v,int t,int f){
    if (v==t)
        return f;
    for (int &i=iter[v];i<G[v].size();i++){
        edge &e=G[v][i];
        if (e.cap>0&&level[v]<level[e.to]){
            int d=dfs(e.to,t,min(f,e.cap));
            if (d>0)
            {
                e.cap-=d;
                G[e.to][e.rev].cap+=d;
                return d;
            }
        }
    }
```

```
48      return 0;
49  }
50  int max_flow(int s,int t){
51      int flow=0;
52      for (;;){
53          bfs(s);
54          if (level[t]<0)
55              return flow;
56          memset(iter,0,sizeof(iter));
57          int f;
58          while ((f=dfs(s,t,INF))>0){
59              flow+=f;
60          }
61      }
62  }
63  int main() {
64      int n,m,s,t;
65      cin>>n>>m>>s>>t;
66      for (int i=0;i<max_v;i++)
67      G[i].clear();
68      for (int i=1;i<=m;i++){
69          int f,t,c;
70          cin>>f>>t>>c;
71          add_edge(f,t,c);
72      }
73      cout<<max_flow(s,t)<<endl;
74      return 0;
75  }
```

39.2 最小费用最大流

```
1   const int maxn=100010;
2   bool vis[maxn];
3   int n,m,s,t,x,y,z,f,dis[maxn],pre[maxn],last[maxn],flow[maxn],maxflow
        ,mincost;
4   //dis最小花费;pre每个点的前驱; last每个点的所连的前一条边; flow源点到
        此处的流量
5   struct nn{
6       int to,next,flow,dis;//flow流量 dis花费
7   }g[maxn];
8   int head[maxn],tot;
9   queue <int> q;
10  void init(){//每次均要初始化
11      maxflow=mincost=0;
12      memset(head,-1,sizeof(head));
13      tot=-1;
14  }
15  void add_edge(int from,int to,int flow,int dis)
16  {
17      g[++tot].next=head[from];
18      g[tot].to=to;
19      g[tot].flow=flow;
20      g[tot].dis=dis;
21      head[from]=tot;
22      g[++tot].next=head[to];
```

```
23      g[tot].to=from;
24      g[tot].flow=0;
25      g[tot].dis=-dis;
26      head[to]=tot;
27  }//反边已经建好
28  //正常情况建边是add_edge(x,y,z,f); add_edge(y,x,0,-f);
29  bool spfa(int s,int t)
30  {
31      memset(dis,0x7f,sizeof(dis));
32      memset(flow,0x7f,sizeof(flow));
33      memset(vis,0,sizeof(vis));
34      q.push(s); vis[s]=1; dis[s]=0; pre[t]=-1;
35
36      while (!q.empty())
37      {
38          int now=q.front();
39          q.pop();
40          vis[now]=0;
41          for (int i=head[now]; i!=-1; i=g[i].next)
42          {
43              if (g[i].flow>0 && dis[g[i].to]>dis[now]+g[i].dis)//正边
44              {
45                  dis[g[i].to]=dis[now]+g[i].dis;
46                  pre[g[i].to]=now;
47                  last[g[i].to]=i;
48                  flow[g[i].to]=min(flow[now],g[i].flow);//
49                  if (!vis[g[i].to])
50                  {
51                      vis[g[i].to]=1;
52                      q.push(g[i].to);
53                  }
54              }
55          }
56      }
57      return pre[t]!=-1;
58  }
59
60  void MCMF()
61  {
62      while (spfa(s,t))
63      {
64          int now=t;
65          maxflow+=flow[t];
66          mincost+=flow[t]*dis[t];
67          while (now!=s)
68          {//从源点一直回溯到汇点
69              g[last[now]].flow-=flow[t];//flow和dis容易搞混
70              g[last[now]^1].flow+=flow[t];
71              now=pre[now];
72          }
73      }
74  }
```

40 图的匹配

40.1 二分图最大匹配

最大匹配：二分图中边集的数目最大的匹配；
最小顶点覆盖：用最少的点，让每条边都至少和其中一个点关联；
最小边覆盖：用尽量少的不相交简单路径覆盖 DAG 图上所有顶点
最大独立集：在 N 个点的图中选出 m 个两两之间没有边的点，m 的最大取值
二分图最小点覆盖 = 二分图最大匹配
二分图最少边覆盖 = 点数-二分图最大匹配
二分图的最大独立集 = 点数-二分图最大匹配
无向图的最大团 = 无向图补图的最大独立集
Hopcroft-Karp 算法求二分图最大匹配复杂度 $O(n\sqrt{m})$

```
/**dx[i]表示左集合i顶点的距离编号，dy[i]表示右集合i顶点的距离编号**/
/**mx[i]表示左集合顶点所匹配的右集合顶点序号，my[i]表示右集合i顶点匹配到的左集合顶点序号。**/
const int maxn=505;
const int inf=0x3f3f3f3f;
struct nn {int v,nxt;}g[maxn];
int tot,head[maxn];
void add_edge(int u,int v){g[++tot]={v,head[u]};head[u]=tot;}
int mx[maxn],my[maxn],vis[maxn];
int dis;
int dx[maxn],dy[maxn];
int n,m;
bool searchp() {
    queue<int>q;
    dis=inf;
    memset(dx,-1,sizeof(dx));
    memset(dy,-1,sizeof(dy));
    for(int i=1;i<=n;i++) {
        if(mx[i]==-1) {
            q.push(i);
            dx[i]=0;
        }
    }
    while(!q.empty()) {
        int u=q.front();
        q.pop();
        if(dx[u]>dis) break;
        for(int i=head[u];i;i=g[i].nxt) {
            int v=g[i].v;
            if(dy[v]==-1) {
                dy[v]=dx[u]+1;
                if(my[v]==-1) dis=dy[v];
                else {
                    dx[my[v]]=dy[v]+1;
                    q.push(my[v]);
                }
            }
        }
    }
    return dis!=inf;
}
bool dfs(int u) {
    for(int i=head[u];i;i=g[i].nxt) {
        int v=g[i].v;
```

```
        if(vis[v]||(dy[v]!=dx[u]+1)) continue;
        vis[v]=1;
        if(my[v]!=-1&&dy[v]==dis) continue;
        if(my[v]==-1||dfs(my[v])) {
            my[v]=u;
            mx[u]=v;
            return true;
        }
    }
    return false;
}
int maxMatch() {
    int res=0;
    memset(mx,-1,sizeof(mx));
    memset(my,-1,sizeof(my));
    while(searchp()) {
        memset(vis,0,sizeof(vis));
        for(int i=1;i<=n; i++)
            if(mx[i]==-1 && dfs(i))
                res++;
    }
    return res;
}
void init() {
    tot=0;
    memset(head,0,sizeof(head));
}
int n1,n2;
int main() {
    n1=rd();n2=rd();m=rd();
    n=max(n1,n2);
    for (int i=1,u,v;i<=m;i++){
        u=rd();v=rd();
        add_edge(u,v);
    }
    cout<<maxMatch()<<endl;
    return 0;
}
```

40.2 二分图最大权匹配 *

KM 算法

40.3 一般图最大匹配 *

带花树
复杂度 $O(n^3)$

41 强联通分量（Tarjan）

```
// bcnt为强连通分量个数，belong数组代表每个点所属的强连通分量编号
int n,m,u,v,inx,bcnt;
```

```
const int maxn=1e5+5;
struct node{int v,nxt;}g[maxn<<1];
int head[maxn],tot,dfn[maxn],low[maxn],vis[maxn],belong[maxn];
void add_edge(int u,int v){g[++tot]={v,head[u]};head[u]=tot;}
stack<int>s;
void tarjan(int u){
         int v;
     dfn[u]=low[u]=++inx;
     s.push(u);
     vis[u]=1;
     for (int i=head[u];i;i=g[i].nxt){
          v=g[i].v;
          if (!dfn[v]){
                  tarjan(v);
                  low[u]=min(low[u],low[v]);
          }
          else{
                   if (vis[v])
                           low[u]=min(low[u],dfn[v]);
          }
     }
     if (dfn[u]==low[u]){
         bcnt++;
         do{
                v=s.top();
                s.pop();
                vis[v]=0;
           belong[v]=bcnt;
         }while(u!=v);
     }
}
int main() {
    scanf("%d%d",&n,&m);
    for (int i=1;i<=m;i++){
         scanf("%d%d",&u,&v);
         add_edge(u,v);
    }
    for (int i=1;i<=n;i++)
        if (!dfn[i])
                tarjan(i);
    return 0;
}
```

42 Kruskal 重构树

在 Kruskal 算法执行过程中，将需要的边提取成一个点，并将边权赋给这个点，然后用这个点连接两个集合。

这样执行完后，会得到一个二叉树。

两点间路径上最大边权是 LCA 的点权。

```
int n,m,q,fa[maxn<<1],val[maxn<<1];
struct Edge{
    int a,b,w;
    bool operator<(const Edge c)const{
        return w<c.w;
    }
```

```
}e[maxn<<1];
int find(int x){
    return x==fa[x]?x:fa[x]=find(fa[x]);
}
void Ex_Kruskal() {
    int ind=n,lim=n<<1; sort(e+1,e+1+m);
    for(int i=1;i<=lim;++i) fa[i]=i;
    for(int i=1;i<=m;++i) {
        int fx=find(e[i].a),fy=find(e[i].b);
        if(fx!=fy) {
            fa[fx]=fa[fy]=++ind;
            val[ind]=e[i].w;
            add_edge(ind,fx); add_edge(ind,fy);
            if(ind==lim-1) break;
        }
    } return ;
}
```

43 最小斯坦纳树

给定连通图 G 中的 n 个点与 k 个关键点，连接 k 个关键点，使得生成树的所有边的权值和最小。

我们使用状态压缩动态规划来求解。用 $f(i,S)$ 表示以 i 为根的一棵树，包含集合 S 中所有点的最小边权值和。

考虑状态转移：

- 对于根的度数为 1，考虑枚举所有连接根的点来转移，$f(i,S) \leftarrow \min(f(i,S), f(j,S)+w(j,i))$（类似于最短路松弛操作）。
- 对于根的度数不为 1 的情况下，$f(i,S) \leftarrow \min(f(i,S), f(i,T)+f(i,S-T))$（用枚举子集的方式来实现）。

时间复杂度 $O\ n*3^k+mlogm*2^k$

```
const int maxn=1e3+5;
const ll mod=998244353;
int head[maxn],tot;
struct nn{int v,w,nxt;}g[maxn<<1];
void add_edge(int u,int v,int w){g[++tot]={v,w,head[u]};head[u]=tot;}
int n,m,k,p[maxn],dp[maxn][1100],vis[maxn];
priority_queue<pair<int,int> >q;
void dij(int s){
    memset(vis,0,sizeof(vis));
    while (!q.empty()){
        auto e=q.top();
        int u=e.second;
        q.pop();
        if (vis[u])continue;
        vis[u]=1;
        for (int i=head[u];i;i=g[i].nxt){
            int v=g[i].v,w=g[i].w;
            if (dp[v][s]>dp[u][s]+w){
                dp[v][s]=dp[u][s]+w;
                q.push({-dp[v][s],v});
            }
        }
    }
}
```

```cpp
int main() {
    memset(dp,0x3f,sizeof(dp));
    n=rd();m=rd();k=rd();
    for (int i=1;i<=m;i++){
        int u=rd(),v=rd(),w=rd();
        add_edge(u,v,w);
        add_edge(v,u,w);
    }
    for (int i=1;i<=k;i++){
        p[i]=rd();dp[p[i]][1<<(i-1)]=0;
    }
    for (int s=1;s<(1<<k);s++){
        for (int i=1;i<=n;i++){
            for (int j=s&(s-1);j;j=s&(j-1)){
                dp[i][s]=min(dp[i][s],dp[i][j]+dp[i][s^j]);
            }
            if (dp[i][s]!=0x3f3f3f3f)q.push(make_pair(-dp[i][s],i));
               //方便pair类型优先队列故值取负数
        }
        dij(s);
    }
    printf("%d\n",dp[p[1]][(1<<k)-1]);
    return 0;
}
```

44 仙人掌 + 圆方树

仙人掌图是指一种无向连通图且满足每条边最多属于一个环
边数最大为 $2n-2$
bzoj2125:
给一个 N 个点 M 条边的连通无向图，满足每条边最多属于一个环，有 Q 组询问，每次询问两点之间的最短路径。
数据范围都是 1e5
建圆方树，初始情况 $1-n$ 为圆点。
在原图中，若一条边 (u,v,w) 不作为某一个环的一部分，则直接在圆方树上建边 (u,v,w)
对于每一个环，新建一个节点作为方点，环上所有节点与方点连接，权值为点到圆方树上深度最小的点的最短路

```cpp
//仙人掌遍历+圆方树建图
void dfs(int u,int fa){
    f[u]=fa;dfn[u]=++dfsid;
    for (int i=head[u];i;i=g[i].nxt){
        int v=g[i].v,w=g[i].w;
        if (v==fa)continue;
        if (!dfn[v]){
            dis1[v]=dis1[u]+w;
            dfs(v,u);
            if (!from[v]){
                vv[u].push_back(make_pair(v,w));
                vv[v].push_back(make_pair(u,w));
            }
        }
        else if (dfn[v]<dfn[u]){
            ++cnt;
            int j=u;
```

```
18              sum[cnt]=dis1[u]-dis1[v]+w;
19              while (j!=v){
20                  from[j]=cnt;
21                  vv[j].push_back(make_pair(n+cnt,min(dis1[j]-dis1[v],
                        dis1[u]-dis1[j]+w)));
22                  vv[n+cnt].push_back(make_pair(j,min(dis1[j]-dis1[v],
                        dis1[u]-dis1[j]+w)));
23                  j=f[j];
24              }
25              vv[j].push_back(make_pair(n+cnt,0));
26              vv[n+cnt].push_back(make_pair(j,0));
27          }
28      }
29  }
```

在圆方树上某两点 LCA 若为原点，答案为两点距离
若为方点，答案为两点跑到离 LCA 只差一步后再加上环上最短距离

45 LGV 引理

用于处理有向无环图上不相交路径计数问题

45.1 定义

$\omega(P)$ 表示 P 这条路径上所有边的边权之积。（路径计数时，可以将边权都设为 1）（事实上，边权可以为生成函数）
$e(u,v)$ 表示 u 到 v 的 ** 每一条 ** 路径 P 的 $\omega(P)$ 之和，即 $e(u,v)=\sum\limits_{P:u\to v}\omega(P)$。
起点集合 A，是有向无环图点集的一个子集，大小为 n。
终点集合 B，也是有向无环图点集的一个子集，大小也为 n。
一组 $A\to B$ 的不相交路径 S：S_i 是一条从 A_i 到 $B_{\sigma(S)_i}$ 的路径（$\sigma(S)$ 是一个排列），对于任何 $i\neq j$，S_i 和 S_j 没有公共顶点。
$N(\sigma)$ 表示排列 σ 的逆序对个数。

45.2 引理

$$M=\begin{bmatrix} e(A_1,B_1) & e(A_1,B_2) & \cdots & e(A_1,B_n)\\ e(A_2,B_1) & e(A_2,B_2) & \cdots & e(A_2,B_n)\\ \vdots & \vdots & \ddots & \vdots\\ e(A_n,B_1) & e(A_n,B_2) & \cdots & e(A_n,B_n)\end{bmatrix}$$

$$\det(M)=\sum_{S:A\to B}(-1)^{N(\sigma(S))}\prod_{i=1}^{n}\omega(S_i)$$

其中 $\sum\limits_{S:A\to B}$ 表示满足上文要求的 $A\to B$ 的每一组不相交路径 S。

Part VI
动态规划

46 单调栈/单调队列优化

形如 $dp[i]=max(dp[j])$，根据 j 的取值范围来选择单调栈/单调队列进行优化
时间复杂度 $O(n)$

47 斜率优化

dp 方程式如 $dp[i]=Min/Max(a[i]*b[j]+c[j]+d[i])$,$b$ 严格单调递增
因为 $a[i]*b[j]$ 这一项，既有 i 又有 j，单调队列优化不再适用，故使用斜率优化的方式处理
将其转化为二维平面上的斜率后，只需找到截距最大/最小的点，一般需要通过单调队列的方式维护一个半凸包来解决

48 四边形不等式优化

又可以理解为决策单调性 DP
使用分治做法复杂度为 $O(nlogn)$

```
void solve(int l,int r,int L,int R){//[l,r]的最优决策点在[L,R]
          int mid=(l+r)>>1;
    int pos=find(mid,L,R);
          //处理mid位置
    solve(l,mid-1,L,pos);solve(mid+1,r,pos,R);
}
```

49 DP 的凸优化（wqs 二分）

wqs 二分用于解决这样一类问题：
若干个物品，选出若干个，选的时候有限制，求最优方案。
并且问题转化为平面函数是一个凸函数
例：
bzoj2654
给一张图，每条边要么是白边要么是黑边，找到一棵恰好包含 k 条白边的最小生成树。
若 $g(i)$ 代表选 i 条白边的最小生成树代价，$g(i)$ 是一个凸函数
二分一个权值让所有的白边加上这个权值然后求最小生成树
若权值给的过大，则白边数量不够，反之白边数量过多。
由此可以得到一个正确的权值下的最小生成树代价，答案里在减去多加的权值即可。

```
const int maxn=1e5+5;
int n,m,k,sum,cnt,fa[maxn];
struct Edge{
    int u,v,w,c;
    bool operator<(const Edge a)const{//考虑二分条件来判断选出的白边
       数量至多或至少
        if (w==a.w)return c<a.c;//此处尽可能选白边，使得求出的白边数
           量至多
        return w<a.w;
```

```
    }
}e[maxn];
int find(int x){return x==fa[x]?x:fa[x]=find(fa[x]);}
bool check(int x){
    sum=cnt=0;
    for (int i=1;i<=m;i++)if (!e[i].c)e[i].w+=x;
    sort(e+1,e+m+1);
    for (int i=1;i<=n;i++)fa[i]=i;
    for (int i=1;i<=m;i++){
        if (find(e[i].u)==find(e[i].v))continue;
        fa[find(e[i].u)]=find(e[i].v);
        sum+=e[i].w;cnt+=e[i].c^1;
    }
    for (int i=1;i<=m;i++)if (!e[i].c)e[i].w-=x;
    return cnt>=k;
}
int main() {
    n=rd();m=rd();k=rd();
    for (int i=1;i<=m;i++){
        e[i].u=rd();e[i].v=rd();e[i].w=rd();e[i].c=rd();
        e[i].u++;e[i].v++;
    }
    int l=-100,r=100;
    while (l<=r){
        int mid=(l+r)>>1;
        if (check(mid))l=mid+1;
        else r=mid-1;
    }
    check(r);
    printf("%d\n",sum-k*r);
    return 0;
}
```

50 CDQ 分治优化

洛谷 P2487
二维 LIS
思路：对 dp 方程进行 CDQ 分治优化，套区间单点取最大值更新，区间求最大值线段树
使用 CDQ 分治优化 DP 时，CDQ 函数中需先 CDQ(l,mid) 再处理左区间对右区间贡献，最后 CDQ(mid+1,r)

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
inline ll rd(){
        ll x=0;char o,f=1;
        while(o=getchar(),o<48)if(o==45)f=-f;
        do x=(x<<3)+(x<<1)+(o^48);
        while(o=getchar(),o>47);
        return x*f;
}
const int maxn=5e4+5;
int n,totb,totc;
struct node{
        int a,b,c;
}p[maxn],t[maxn];
```

```
struct segment_tree{
        int f;
        double g;
}tree[maxn<<2];
void build(int l=1,int r=totc,int p=1){
        tree[p].f=0;tree[p].g=0;
        if (l==r)return;
        int mid=(l+r)>>1;
        build(l,mid,p<<1);build(mid+1,r,p<<1|1);
}
void update(int a,int f,double g,int l=1,int r=totc,int p=1){
        if (a>r||a<l)return;
        if (l==r){
                if (g==0){tree[p].f=0;tree[p].g=0;return;}
                if (tree[p].f==f)tree[p].g+=g;
                else if (tree[p].f<f){tree[p].f=f,tree[p].g=g;}
                return;
        }
        int mid=(l+r)>>1;
        update(a,f,g,l,mid,p<<1);update(a,f,g,mid+1,r,p<<1|1);
        if (tree[p<<1].f>tree[p<<1|1].f){
                tree[p].f=tree[p<<1].f;
                tree[p].g=tree[p<<1].g;
        }
        else if (tree[p<<1].f<tree[p<<1|1].f){
                tree[p].f=tree[p<<1|1].f;
                tree[p].g=tree[p<<1|1].g;
        }
        else{
                tree[p].f=tree[p<<1].f;
                tree[p].g=tree[p<<1].g+tree[p<<1|1].g;
        }
}
segment_tree query(int a,int b,int l=1,int r=totc,int p=1){
        if (a>r||b<l){
                segment_tree ans;
                ans.f=0;
                ans.g=0;
                return ans;
        }
        if (a<=l&&b>=r){
                return tree[p];
        }
        int mid=(l+r)>>1;
        segment_tree cnt1=query(a,b,l,mid,p<<1),cnt2=query(a,b,mid+1,
           r,p<<1|1);
        if (cnt1.f>cnt2.f)
                return cnt1;
        else if (cnt1.f<cnt2.f)
                return cnt2;
        else{
                segment_tree ans;
                ans.f=cnt1.f;
                ans.g=cnt1.g+cnt2.g;
                return ans;
        }
}
```

```
72 bool cmp1(node x,node y){
73         if (x.b!=y.b)return x.b>y.b;
74         return x.a<y.a;
75 }
76 bool cmp2(node x,node y){
77         return x.a<y.a;
78 }
79 int f1[maxn],f2[maxn];
80 double g1[maxn],g2[maxn];
81 void CDQ(int l,int r,int f[],double g[]){
82         if (l==r)return;
83         int mid=(l+r)>>1;
84         CDQ(l,mid,f,g);
85         sort(p+l,p+r+1,cmp1);
86         for (int i=l;i<=r;i++){
87                 if (p[i].a<=mid)update(p[i].c,f[p[i].a],g[p[i].a]);
88                 else{
89                         segment_tree cnt=query(p[i].c,totc);
90                         if (f[p[i].a]==cnt.f+1){
91                                 g[p[i].a]+=cnt.g;
92                         }
93                         else if (f[p[i].a]<cnt.f+1){
94                                 f[p[i].a]=cnt.f+1;
95                                 g[p[i].a]=cnt.g;
96                         }
97                 }
98         }
99         for (int i=l;i<=r;i++){
100                 if (p[i].a<=mid)update(p[i].c,0,0.0);
101         }
102         sort(p+l,p+r+1,cmp2);
103         CDQ(mid+1,r,f,g);
104 }
105 int tmp[maxn];
106 void solve(int op){
107         for (int i=1;i<=n;i++)p[i].a=i;
108         for (int i=1;i<=n;i++)tmp[i]=t[i].b;
109     sort(tmp+1,tmp+n+1);
110         totb=unique(tmp+1,tmp+n+1)-tmp-1;
111         for (int i=1;i<=n;i++)p[i].b=lower_bound(tmp+1,tmp+totb+1,t[i
              ].b)-tmp;
112         for (int i=1;i<=n;i++)tmp[i]=t[i].c;
113     sort(tmp+1,tmp+n+1);
114         totc=unique(tmp+1,tmp+n+1)-tmp-1;
115         for (int i=1;i<=n;i++)p[i].c=lower_bound(tmp+1,tmp+totc+1,t[i
              ].c)-tmp;
116         build();
117         if (op==0)CDQ(1,n,f1,g1);
118         else CDQ(1,n,f2,g2);
119 }
120 int main() {
121     n=rd();
122     for (int i=1;i<=n;i++){
123         t[i].a=i;t[i].b=rd();t[i].c=rd();
124     }
125     for (int i=1;i<=n;i++)f1[i]=f2[i]=1,g1[i]=g2[i]=1.0;
126     solve(0);
```

```
        for (int i=1;i<=n;i++)t[i].a=n-t[i].a+1,t[i].b=-t[i].b,t[i].c
           =-t[i].c;
        reverse(t+1,t+n+1);
        solve(1);
        int ans=0;
        double sum=0;
        for (int i=1;i<=n;i++)ans=max(ans,f1[i]);
        for (int i=1;i<=n;i++)if (f1[i]==ans)sum+=g1[i];
        printf("%d\n",ans);
        for (int i=1;i<=n;i++){
                if (f1[i]+f2[n-i+1]-1==ans)printf("%.5f%c",1.0*(g1[i
                   ]*g2[n-i+1])/sum,i==n?'\n':' ');
                else printf("0.0000%c",i==n?'\n':' ');
        }
    return 0;
}
```

51 仙人掌 DP

考虑树边与非树边，先思考出问题在树形 DP 下解法，再考虑加入非树边后 DP 的方程变化

52 DP 套 DP

数据范围很小，统计某一个通过 dp 算出的值为固定值的方案数
例 1
DNA 序列是一个非空的仅包含大写字母 A、C、G、T 的字符串。
给定一个 DNA 序列 S*S*，请对于每个 $i = 0, 1, 2, ..., S$ 统计有多少长度为 $m(m <= 1000)$ 的 DNA 序列 T 满足 S 与 T 的最长公共子序列长度刚好为 i。
注意到最长公共子序列是一个单调递增的过程
可以用状压的方式保存每一位与前一位的差值（0，1）
设 dp 数组为 $dp[maxn][1 << 15]$
代表当前串与去掉最后一个字母的串和 S 串的 LCS 的差值
此题第二维状态在加入了一个字母后的新状态可以被预处理，可以加速 dp 转移过程

```
const int maxn=1e3+5;
const ll mod=1e9+7;
int _,n,m;
int dp[maxn][1<<15],ans[16],a[maxn];
char s[20];
int calc(int x){
    int ans=0;
    while (x){
        x-=x&-x;ans++;
    }
    return ans;
}
int tmp[20]={0},nxt[20]={0},pre[1<<15][5];
int main() {
    for(scanf("%d",&_);_;_--){
        memset(dp,0,sizeof(dp));
        memset(ans,0,sizeof(ans));
        dp[0][0]=1;
        scanf("%s",s+1);
```

```
        scanf("%d",&m);
        n=strlen(s+1);
        for (int i=1;i<=n;i++){
            if (s[i]=='A')a[i-1]=1;
            if (s[i]=='C')a[i-1]=2;
            if (s[i]=='G')a[i-1]=3;
            if (s[i]=='T')a[i-1]=4;
        }
        for (int j=0;j<(1<<n);j++){
            for (int k=1;k<=4;k++){
                tmp[0]=nxt[0]=0;
                int vj=0;
                for (int x=0;x<n;x++){
                    if (((j>>x)&1)==1)tmp[x+1]=tmp[x]+1;
                    else tmp[x+1]=tmp[x];
                    if (a[x]==k){
                        nxt[x+1]=tmp[x]+1;
                    }
                    else nxt[x+1]=max(nxt[x],tmp[x+1]);
                    if ((nxt[x+1]-nxt[x])==1)vj|=(1<<x);
                }
                pre[j][k]=vj;
            }
        }
        for (int i=1;i<=m;i++){
            for (int j=0;j<(1<<n);j++){
                for (int k=1;k<=4;k++){
                    int vj=pre[j][k];
                    dp[i][vj]=(dp[i][vj]+dp[i-1][j])%mod;
                }
            }
        }
        for (int i=0;i<(1<<n);i++){
            ans[calc(i)]=(ans[calc(i)]+dp[m][i])%mod;
        }
        for (int i=0;i<=n;i++)printf("%d\n",ans[i]);
    }
    return 0;
}
```

例 2

对于两个字符串 A 和 B，定义它们之间的编辑距离为最少的对 A 的编辑次数使得 A 和 B 相等，每次编辑可以是插入/删除/修改一位字符。

给定一个仅由 26 个大写字母构成的字符串 S，统计有多少个仅由 26 个大写字母构成字符串与 S 的编辑距离恰好为 d。

$|S| <= 10, 0 <= d <= 10$

更为普遍的方法是用 map 来保存当前串跟 $S[1...i]$ 的编辑距离 vector，vector 相同的计为相同。

主要考虑当前 vector 在加入新字符后 vector 状态的改变

```
const int maxn=1e3+5;
const int mod=998244353;
char s[12];
int d,ans;
map<vector<int>,int>dp[2];
int main() {
    scanf("%s",s+1);scanf("%d",&d);
    int n=strlen(s+1);
```

```
    if (n==d)ans++;
    vector<int>v;
    for (int i=1;i<=n;i++)v.push_back(i);
    dp[0][v]=1;
    for (int i=1;i<=n+d;i++){
        dp[i&1].clear();
        for (auto j:dp[~i&1]){
            if (j.second==0)continue;
            for (int k=0;k<26;k++){
                vector<int>tmp=j.first;
                vector<int>res(n);
                if(s[1]==k+'A')res[0]=i-1;
                else res[0]=min(tmp[0]+1,i);
                for (int x=2;x<=n;x++){
                    if (s[x]==k+'A')res[x-1]=tmp[x-2];
                    else res[x-1]=min(tmp[x-1],min(res[x-2],tmp[x-2])
                       )+1;
                }
                int flag=1;
                for(int x=0;x<n;x++)if(res[x]<=d)flag=0;
                if(flag)continue;
                dp[i&1][res]=(dp[i&1][res]+j.second)%mod;
            }
        }
        for (auto j:dp[i&1]){
            if (j.first[n-1]==d){
                ans=(ans+j.second)%mod;
            }
        }
    }
    printf("%d\n",ans);
    return 0;
}
```

Part VII
字符串

多串问题时，若需要连接两个字符串时，记得在中间加入一个保证不会出现的字符以免对结果有影响。

53 KMP

单模串匹配
先对模式串跑出 nxt 数组，再用 nxt 数组对文本串跑字符串匹配。两步骤算法流程类似
时间复杂度：$O(n)$

```
const int maxn=1e3+5;
char s[maxn],t[maxn];
int nxt[maxn],n,m,ans;
int main() {
  scanf("%s",s+1);
  scanf("%s",t+1);
  n=strlen(s+1);m=strlen(t+1);
```

```
  for (int i=0;i<=m;i++)nxt[i]=0;
  int j=0;
  for (int i=2;i<=m;i++){
    while (j&&t[i]!=t[j+1])j=nxt[j];
    if (t[i]==t[j+1])j++;
    nxt[i]=j;
  }
  j=0;
  for (int i=1;i<=n;i++){
    while (j&&t[j+1]!=s[i])j=nxt[j];
    if (t[j+1]==s[i])j++;
    if (j==m){
      //匹配成功
      j=nxt[j];
    }
  }
    return 0;
}
```

54 EXKMP

nxt 数组为字符串 t 的 Z 函数数组 extend 数组为 t 和 s 的每一个后缀的 LCP(最长公共前缀) 时间复杂度 $O(n)$

```
void getnxt(){
    nxt[1]=0;
    int now=1;
    while (now+1<=m&&t[now]==t[1+now])now++;
    nxt[2]=now-1;
    int j=2;
    for (int i=3;i<=m;i++){
        if (i+nxt[i-j+1]<nxt[j]+j)nxt[i]=nxt[i-j+1];
        else{
            int now=max(nxt[j]+j-i,1);
            while (i+now-1<=m&&t[now]==t[i+now-1])now++;
            nxt[i]=now-1;
            j=i;
        }
    }
}
void exkmp(){
    getnxt();
    int now=1;
    while (s[now]==t[now]&&now<=min(n,m))now++;
    extend[1]=now-1;
    int j=1;
    for (int i=2;i<=n;i++){
        if (i+nxt[i-j+1]<extend[j]+j)extend[i]=nxt[i-j+1];
        else{
            int now=max(extend[j]+j-i,1);
            while (now<=m&&now+i-1<=n&&t[now]==s[now+i-1])now++;
            extend[i]=now-1;
            j=i;
        }
    }
}
```

55 AC 自动机

Trie+KMP

若处理每个串匹配成功次数，对 $query$ 函数中暴力跳 $fail$ 链的过程修改为在 $fail$ 树上差分即可

```
struct ACTrie{
    static const int N=1e6+5;
    struct node{
        int to[26];
        int &operator[](int n){return to[n];}
    }a[N];
    int num[N];
    int t,fail[N];
    void init(){
        t=0;a[t++]=node();
        memset(num,0,sizeof(num));
    }
    void insert(char *s){
        int len=strlen(s),k=0;
        for (int i=0;i<len;i++){
            if (!a[k][s[i]-'a'])a[k][s[i]-'a']=t,a[t++]=node();
            k=a[k][s[i]-'a'];
        }
        num[k]++;
    }//正常建trie树即可
    void build(){
        queue<int>q;
        for (int i=0;i<26;i++)if (a[0][i])q.push(a[0][i]);//fail指针1所有点指向根（0）
        while (!q.empty()){
            int u=q.front();
            q.pop();
            for (int i=0;i<26;i++){
                if (a[u][i])fail[a[u][i]]=a[fail[u]][i],q.push(a[u][i]);//fail指针2
                else a[u][i]=a[fail[u]][i];
            }
        }
    }//建立自动机，处理fail指针
    int query(char *s){
        int len=strlen(s);
        int u=0,ans=0;
        for (int i=0;i<len;i++){
            u=a[u][s[i]-'a'];
            for (int j=u;j&&num[j]!=-1;j=fail[j]){
                ans+=num[j],num[j]=-1;
            }
        }
        return ans;
    }
}ac;
//统计文本串成功匹配的模式串个数
//使用前需init（来自trie树预处理）
```

56 后缀数组

$sa[i]$ 表示将所有后缀排序后第 i 小的后缀的编号，$rk[i]$ 表示后缀 i 的排名
倍增求后缀数组，时间复杂度 $O(nlogn)$
$height[i] = lcp(sa[i], sa[i-1])$，即第 i 名的后缀与它前一名的后缀的最长公共前缀。
$height[1]$ 可视作 0
每次运行完会自动清空，不需要额外操作
求 $height$ 数组的效率为 $O(n)$
通过 $height$ 数组可以解决本质不同子串数量等问题
对于多串问题时常可以直接连接并用分隔符分隔再做后缀数组来求解

```
const int maxn=1e6+5;
char s[maxn];
int n,sa[maxn],rk[maxn],oldrk[maxn<<1],id[maxn],cnt[maxn],px[maxn],ht
   [maxn];
bool cmp(int x, int y, int w) {
        return oldrk[x]== oldrk[y]&&oldrk[x + w]==oldrk[y + w];
}
void get_sa(){
        int m=300,p;
        for (int i=1;i<=n;i++)++cnt[rk[i]=s[i]];
        for (int i=1;i<=m;i++)cnt[i]+=cnt[i-1];
        for (int i=n;i>=1;i--)sa[cnt[rk[i]]--]=i;
        for (int w=1;;w<<=1,m=p){
                p=0;
                for (int i=n;i>n-w;i--)id[++p]=i;
                for (int i=1;i<=n;i++)if (sa[i]>w)id[++p]=sa[i]-w;
                memset(cnt,0,sizeof(cnt));
                for (int i=1;i<=n;i++)++cnt[px[i]=rk[id[i]]];
                for (int i=1;i<=m;i++)cnt[i]+=cnt[i-1];
                for (int i=n;i>=1;--i)sa[cnt[px[i]]--]=id[i];
                memcpy(oldrk,rk,sizeof(rk));
                p=0;
                for (int i=1;i<=n;i++)
                        rk[sa[i]]=cmp(sa[i],sa[i-1],w)?p:++p;
                if (p==n){
                        for (int i=1;i<=n;i++)sa[rk[i]]=i;
                        break;
                }
        }
  memset(cnt,0,sizeof(cnt));
}
void get_height(){
        for (int i=1,k=0;i<=n;++i) {
          if (k)--k;
          while (s[i+k]==s[sa[rk[i]-1]+k])++k;
          ht[rk[i]]=k;
        }
}
void clear(){
  for (int i=0;i<=n;i++){
    s[i]=sa[i]=rk[i]=oldrk[i]=id[i]=cnt[i]=px[i]=ht[i]=0;
  }
}//线性清空
```

若要查询两个后缀的最长公共前（后）缀
可对 $height$ 数组建 ST 表，通过 rk 数组找到对应的 $height$ 区间

ST 表上查询即可

57 （广义）后缀自动机

len 代表此状态的最长串，fa 代表当前点 $parent$ 树上的父亲
cnt 代表自动机状态数（1 节点为空）,$last$ 为每次插入后的当前节点
普通 SAM 和广义 SAM 均可直接使用 $insert$ 函数直接构造后缀自动机
普通 SAM 时，每次 $last$ 是上一次插入的位置
广义 SAM 时，一个新串插入时需要将 $last$ 置 1
SAM 的状态对应一个 $endpos$ 等价类，每一个状态对应一些字符串，所有字符串都是最长字符串的后缀。
每一个状态的父亲节点对应的字符串为当前节点代表字符串的后缀。
若当前节点对应字符串长度为 $[MaxLen, MinLen]$，则父亲节点对应字符串最长为 $MinLen-1$
不同子串数：每次加入一个串后，数量增加 $len[i]-len[fa[i]]$
构造 SAM 的时空复杂度均为 $O(n)$
常用技巧：
用线段树合并维护 $endpos$ 集合，每个节点的线段树可以代表当前状态的子串在整串中的出现位置
广义后缀自动机线段树合并可以维护当前子串在哪些串中出现
最后根据题目要求要维护对应的线段树上信息
自动机 DAG 上跑另一个串：

```
int now=1,l=0;//now为自动机上节点位置， l为当前前缀匹配成功的后缀长度
for (int i=1;i<=n;i++){
  while (now&&!nxt[now][s[i]-'a'])now=fa[now],l=len[now];
  if (!now)now=1;
  if (nxt[now][s[i]-'a'])now=nxt[now][s[i]-'a'],l++;
}
```

建 SAM：

```
const int maxlen=2e6+5;//SAM节点数要开两倍
const int CH=26;
int nxt[maxlen][CH],len[maxlen],cnt,fa[maxlen],last;
inline int insert(int x,int y){
    if (nxt[x][y]){
        int q=nxt[x][y];
        if (len[q]==len[x]+1)return q;
        else{
            int nq=++cnt;len[nq]=len[x]+1;
            memcpy(nxt[nq],nxt[q],sizeof(nxt[nq]));
            fa[nq]=fa[q];fa[q]=nq;
            for (;nxt[x][y]==q;x=fa[x])nxt[x][y]=nq;
            return nq;
        }
    }
    else{
        int p=x,np=++cnt;
        len[np]=len[p]+1;
        for (;p&&!nxt[p][y];p=fa[p])nxt[p][y]=np;
        if (!p)fa[np]=1;
        else{
            int q=nxt[p][y];
            if (len[q]==len[p]+1)fa[np]=q;
            else{
                int nq=++cnt;len[nq]=len[p]+1;
```

```
                memcpy(nxt[nq],nxt[q],sizeof(nxt[nq]));
                fa[nq]=fa[q];fa[q]=fa[np]=nq;
                for (;nxt[p][y]==q;p=fa[p])nxt[p][y]=nq;
            }
        }
        return np;
    }
}
void init(){
    cnt=1;last=1;
}
void clear(){
    for (int i=0;i<=cnt;i++){
        memset(nxt[i],0,sizeof(nxt[i]));
        len[i]=fa[i]=0;
    }
}//线性清空
```

广义 SAM 时若要用如下 update 方式，需在整个自动机建好后 update

```
void update(int x,int now){
    while (x&&lst[x]!=now){
        val[x]++;lst[x]=now;x=fa[x];
    }
}
```

对于第 k 大子串类问题（本质相同子串出现位置不同算不同）
首先需要统计一个子串的出现次数只需要看它所在节点的 endpos 集合大小
具体求法为：每次 $insert$ 完，$val[last]$++; 最后 dfs 一次 $parent$ 树合并子树（val[u]+=val[v]）即可
若遇到求第 k 大本质不同子串类题目
通过

```
for (int i=2;i<=cnt;i++)val[i]=1;
```

直接控制当前串的出现次数为 1，因此显然也不需要进行 dfs 合并子树

58 Manacher

先将 s 串变换成如 $\#s[1]\#s[2]\#...\#s[n]\#@$ 的形式
这样原串中奇数长度的回文串即变换后串以字母为中心的回文串
偶数长度的回文串即变换后以 # 为中心的回文串
算法在 $O(m)$ 的时间复杂度内将 f 数组求出
具体含义见注释

```
const int maxn=11000005;
char s[maxn],t[maxn<<1];
int f[maxn<<1];//f[i]=t[i]为中心的最长回文串半径
void manacher(){
    t[0]='$';t[1]='#';
    int n=strlen(s+1),m;
    for (int i=1;i<=n;i++){
        t[i<<1]=s[i];
        t[i<<1|1]='#';
    }
    t[m=(n+1)<<1]='@';
    f[1]=f[m]=1;int r=0,p=0;
```

```
    for (int i=2;i<m;i++){
        f[i]=r>i?min(r-i,f[(p<<1)-i]):1;
        while (t[i-f[i]]==t[i+f[i]])f[i]++;
        if (i+f[i]>r)r=i+f[i],p=i;
    }
}
```

59 回文自动机

因一个串的回文子串数量最大为 $O(n)$，故后续 fail 等指针暴力求总复杂度也为 $O(n)$

每个节点代表唯一的回文串，fail 指针指向这个节点所代表的回文串的最长回文后缀所对应的节点

转移边并非代表在原节点代表的回文串后加一个字符，而是表示在原节点代表的回文串前后各加一个相同的字符

0 节点为偶根，1 节点为奇根

若求本质不同的回文串数量，输出 $cnt-2$ 即可（除去奇根和偶根）

多串问题时可直接连接并用两个不同的且不在题目给定字符集的字符间隔再做 PAM 求解

若求回文子串出现次数，用类似 SAM 求次数的方法即可，但由于插入时是安排拓扑序的方式插入

故统计出现次数可使用如下方法：

```
//cnt[last]++; 每次insert完,使当前节点cnt++;
for (int i=cnt;i>=0;i--){
    cnt[fail[i]]+=cnt[i];
}
```

求 trans 指针（指向小于等于当前节点长度的一半最长回文后缀）

```
if (len[cnt]<=2)trans[cnt]=fail[cnt];
else{
    int now=trans[x];
    while (s[i-len[now]-1]!=s[i]||(len[now]+2)*2>len[cnt])now=fail[
       now];
    trans[cnt]=nxt[now][s[i]-'a'];
}
```

增量构造：

```
const int maxn=5e5+5;
int nxt[maxn][26],fail[maxn],last,cnt,f[maxn],len[maxn];
char s[maxn];
int n;
int getfail(int x,int i){
    while (s[i-len[x]-1]!=s[i])x=fail[x];
    return x;
}
int insert(int x,int i){
    x=getfail(x,i);
    if (!nxt[x][s[i]-'a']){
        len[++cnt]=len[x]+2;
        fail[cnt]=nxt[getfail(fail[x],i)][s[i]-'a'];
        //若需要求trans指针, 则讲上一段代码加入此处
        f[cnt]=f[fail[cnt]]+1;//f数组代表深度, 可表示为当前点结尾的回
           文串数量
        nxt[x][s[i]-'a']=cnt;
    }
```

```
    return nxt[x][s[i]-'a'];
}//return last
//使用方法last=insert(last,i) i为在s串中位置
void init(){
    last=cnt=1;fail[0]=1;len[1]=-1;
}
void clear(){
    for (int i=0;i<=cnt;i++){
        memset(nxt[i],0,sizeof(nxt[i]));
        len[i]=fail[i]=f[i]=0;
    }
}
int main(){
    scanf("%s",s+1);
    n=strlen(s+1);
    int lastans=0;
    for(int i=1;i<=n;i++){
        s[i]=(s[i]-97+lastans)%26+97;
        last=insert(last,i);
        printf("%d ",lastans=f[last]);
    }
    return 0;
}
```

60 序列自动机

```
char s[maxn];
int nxt[maxn][26];
void build(){
    for (int i=strlen(s+1);i;--i){
        memcpy(nxt[i-1],nxt[i],104);
        nxt[i-1][s[i]-'a']=i;
    }
}
```

61 最小表示法

字符串的最小表示：字符串 S 的最小表示为与 S 循环同构的所有字符串中字典序最小的字符串

复杂度:$O(n)$

函数范围从 1 开始的数组 a 最小表示的第一个元素位置

```
int find_min(int a[]){
    int k=0,i=1,j=2;
    while (k<n&&i<=n&&j<=n){
        if (a[(i+k-1)%n+1]==a[(j+k-1)%n+1])k++;
        else{
            a[(i+k-1)%n+1]>a[(j+k-1)%n+1]?i+=k+1:j+=k+1;
            if (i==j)i++;k=0;
        }
    }
    return min(i,j);
}
```

```
//return min_position
```

也可使用后缀数组解决
具体实现可将 S 复制一份变成 SS
对 SS 做出后缀数组，找到最小的 $sa[i]$ 满足 $sa[i] <= n$ 即为 S 的最小表示

62 Lyndon 分解

Lyndon 串：字符串 s 的字典序严格小于 s 的所有后缀的字典序
上述条件等价于 s 的字典序严格小于它的所有非平凡的循环同构串。
Lyndon 分解：串 s 的 Lyndon 分解记为 $s = w_1 w_2 \cdots w_k$，其中所有 w_i 为 Lyndon 串，并且他们的字典序按照非严格单减排序，即 $w_1 \geq w_2 \geq \cdots \geq w_k$。可以发现，这样的分解存在且唯一。
Duval 可以在 $O(n)$ 的时间内求出一个串的 Lyndon 分解。

```
vector<int> lyndon(){
    int i=1;
    vector<int>ans;
    while (i<=n){
        int j=i+1,k=i;
        while (j<=n&&s[k]<=s[j]){
            if (s[k]<s[j])k=i;
            else k++;
            j++;
        }
        while (i<=k){
            ans.push_back(i+j-k-1);
            i+=j-k;
        }
    }
    return ans;
}
//return right pos
```

Part VIII
杂项

63 莫队

63.1 普通莫队

洛谷 P1494
询问区间选两个数，数相同概率。

```
const int maxn=5e4+5;
int n,m,a[maxn],block_size;
ll ans1[maxn],ans2[maxn],cnt[maxn];
ll sum;
void add(int x){
    sum+=cnt[x];
    cnt[x]++;
}
void del(int x){
```

```
    cnt[x]--;
    sum-=cnt[x];
}
struct node{
    int l,r,id;
    bool operator<(const node a)const{
        if (l/block_size!=a.l/block_size)return l<a.l;
        return (l/block_size)&1?r<a.r:r>a.r;
    }//排序基本固定 先按块排 后按r交替升降序
}q[maxn];
int main() {
    n=rd();m=rd();
    block_size=int(ceil(pow(n,0.5)));//块的大小
    for (int i=1;i<=n;i++)a[i]=rd();
    for (int i=1;i<=m;i++){
        q[i].l=rd();q[i].r=rd();
        q[i].id=i;
    }
    sort(q+1,q+m+1);
    for (int i=1,l=1,r=0;i<=m;i++){
        if (q[i].l==q[i].r){
            ans1[q[i].id]=0;ans2[q[i].id]=1;
            continue;
        }
        while (l>q[i].l)add(a[--l]);
        while (r<q[i].r)add(a[++r]);
        while (l<q[i].l)del(a[l++]);
        while (r>q[i].r)del(a[r--]);//区间暴力转移 先扩大后缩小
        ans1[q[i].id]=sum;
        ans2[q[i].id]=1ll*(r-l+1)*(r-l)/2;
    }
    for (int i=1;i<=m;i++){
        if (ans1[i]){
            ll tmp=gcd(ans1[i],ans2[i]);
            ans1[i]/=tmp;
            ans2[i]/=tmp;
        }
        else
            ans2[i]=1;
        printf("%lld/%lld\n",ans1[i],ans2[i]);
    }
    return 0;
}
```

63.2 带修莫队

n 个整数，m 次操作
操作 1：修改单点值
操作 2：询问区间种类数
也可用带修主席树（树套树）的方式做到两个 log
取块大小为 $S=n^{\frac{2}{3}}$ 可得到最佳时间复杂度 $O(n^{\frac{5}{3}})$

```
const int maxn=140005;
const int S=2500;
const int K=1e6+5;
struct query{
```

```
    int l,r,t,id;
    bool operator <(const query &A)const{
        if(l/S!=A.l/S)return l/S<A.l/S;
        if(r/S!=A.r/S){
            if(l/S&1)return r/S<A.r/S;
            else return r/S>A.r/S;
        }
        if(r/S&1)return t<A.t;
        else return t>A.t;
    }
}q[maxn];
char s[maxn][5];
int n,m,now,tot,a[maxn],pos[maxn],X[maxn],Y[maxn],ans[maxn];
int res,cnt[K];
void add(int x){if (++cnt[x]==1)res++;}
void del(int x){if (--cnt[x]==0)res--;}
int main() {
    scanf("%d%d",&n,&m);
    for (int i=1;i<=n;i++)scanf("%d",&a[i]);
    for (int i=1;i<=m;i++){
        scanf("%s",s[i]+1);
        int x,y;
        scanf("%d%d",&x,&y);
        if (s[i][1]=='R'){
            pos[++now]=x;
            X[now]=a[x];
            Y[now]=y;
            a[x]=y;
        }
        else q[++tot]=(query){x,y,now,i};
    }
    sort(q+1,q+tot+1);
    int l=1,r=0;
    for (int i=1;i<=tot;i++){
        while (now<q[i].t){
            now++;
            a[pos[now]]=Y[now];
            if (pos[now]>=l&&pos[now]<=r)del(X[now]),add(Y[now]);
        }
        while (now>q[i].t){
            a[pos[now]]=X[now];
            if (pos[now]>=l&&pos[now]<=r)del(Y[now]),add(X[now]);
            now--;
        }
        while(r<q[i].r)add(a[++r]);
        while(l>q[i].l)add(a[--l]);
        while(r>q[i].r)del(a[r--]);
        while(l<q[i].l)del(a[l++]);
        ans[q[i].id]=res;
    }
    for (int i=1;i<=m;i++)if (s[i][1]=='Q')printf("%d\n",ans[i]);
    return 0;
}
```

63.3 树上（带修）莫队

询问关于一个树上路径
处理出树的括号序，在括号序上进行莫队。
块大小设为 S
复杂度同普通（带修）莫队

```
const int maxn=2e5+5;
const int S=2000;
int head[maxn],tot;
struct nn{int v,nxt;}g[maxn<<1];
void add_edge(int u,int v){g[++tot]={v,head[u]};head[u]=tot;}
int n,m,qy,v[maxn],w[maxn],c[maxn],now,totq;
int son[maxn],sz[maxn],f[maxn],dep[maxn],dfsid,top[maxn],dfn[maxn],
   idL[maxn],idR[maxn];
void init(int u,int fa){
    sz[u]=1;son[u]=0;f[u]=fa;dep[u]=dep[fa]+1;
    for (int i=head[u];i;i=g[i].nxt){
        int v=g[i].v;
        if (v==fa)continue;
        init(v,u);
        sz[u]+=sz[v];
        if (sz[v]>sz[son[u]])son[u]=v;
    }
}
void chain(int u,int k){
    top[u]=k;
    dfn[++dfsid]=u;
    idL[u]=dfsid;
    if (son[u])chain(son[u],k);
    for (int i=head[u];i;i=g[i].nxt){
        int v=g[i].v;
        if (v==f[u]||v==son[u])continue;
        chain(v,v);
    }
    dfn[++dfsid]=u;
    idR[u]=dfsid;
}
int LCA(int x,int y){
    while (top[x]!=top[y]){
        if (dep[top[x]]<dep[top[y]])swap(x,y);
        x=f[top[x]];
    }
    if (dep[x]>dep[y])swap(x,y);
    return x;
}
struct query{
    int l,r,t,lca,id;
    bool operator <(const query &A)const{
        if(l/S!=A.l/S)return l/S<A.l/S;
        if(r/S!=A.r/S){
            if(l/S%2)return r/S<A.r/S;
            else return r/S>A.r/S;
        }
        if(r/S%2)return t<A.t;
        else return t>A.t;
    }
}q[maxn];
```

```
51 int pos[maxn],X[maxn],Y[maxn],mark[maxn],cnt[maxn];
52 ll res,ans[maxn];
53 void add(int x){//更新一个点的时候因为要去重，故需要一个mark数组来判
       断是否已经加入该点
54     mark[x]^=1;
55     if (mark[x])res+=1ll*v[c[x]]*w[++cnt[c[x]]];
56     else res-=1ll*v[c[x]]*w[cnt[c[x]]--];
57 }
58 int main() {
59     n=rd();m=rd();qy=rd();
60     for (int i=1;i<=m;i++)v[i]=rd();
61     for (int i=1;i<=n;i++)w[i]=rd();
62     for (int i=1;i<n;i++){
63         int u=rd(),v=rd();
64         add_edge(u,v);
65         add_edge(v,u);
66     }
67     init(1,0);
68     chain(1,1);
69     for (int i=1;i<=n;i++)c[i]=rd();
70     for (int i=1;i<=qy;i++){
71         int op=rd(),x=rd(),y=rd();
72         if (op==0){//修改操作
73             pos[++now]=x;
74             X[now]=c[x];Y[now]=y;
75             c[x]=y;
76         }
77         else{
78             int lca=LCA(x,y);
79             if (idL[x]>idL[y])swap(x,y);
80             if (x==lca||y==lca)q[++totq]={idL[x],idL[y],now,0,i};
81             else q[++totq]={idR[x],idL[y],now,lca,i};
82         }
83     }
84     sort(q+1,q+totq+1);
85     int l=1,r=0;
86     for (int i=1;i<=totq;i++){
87         while (l>q[i].l)add(dfn[--l]);
88         while (r<q[i].r)add(dfn[++r]);
89         while (l<q[i].l)add(dfn[l++]);
90         while (r>q[i].r)add(dfn[r--]);
91         while (now<q[i].t){
92             now++;
93             if (mark[pos[now]]){
94                 add(pos[now]);
95                 c[pos[now]]=Y[now];
96                 add(pos[now]);
97             }
98             else c[pos[now]]=Y[now];
99         }
100        while (now>q[i].t){
101            if (mark[pos[now]]){
102                add(pos[now]);
103                c[pos[now]]=X[now];
104                add(pos[now]);
105            }
106            else c[pos[now]]=X[now];
```

```
107             now--;
108         }
109         if (q[i].lca)add(q[i].lca);
110         ans[q[i].id]=res;
111         if (q[i].lca)add(q[i].lca);
112     }
113     for (int i=1;i<=qy;i++)if (ans[i])printf("%lld\n",ans[i]);
114     return 0;
115 }
```

63.4 回滚莫队

对于询问，容易维护加入而不容易维护删除/容易维护删除而不容易维护加入
可以使用回滚莫队。
对于所有询问，第一关键字为升序 $\frac{l}{block\_size}$，第二关键字为升序 r 排序
对于左右端点在同一块的询问直接暴力处理
对于不在同一块的询问，r 单调递增，l 每次只会变动根号次，处理完询问撤销左端点的移动即可
** 撤销必须按照加入顺序倒序撤销，不然维护信息可能出现问题 **
时间复杂度 $O(n\sqrt{n}+m\sqrt{n})$

```
1  const int maxn=1e5+5;
2  int block_size;
3  struct query{
4      int l,r,id;
5      bool operator<(const query a)const{
6          if (l/block_size==a.l/block_size)return r<a.r;
7          return l/block_size<a.l/block_size;
8      }
9  }q[maxn];
10 int n,a[maxn],qy,l,r,b[maxn],tot,cnt[maxn],L,R;
11 ll res,ans[maxn];
12 int main() {
13     n=rd();qy=rd();
14     block_size=int(ceil(pow(n,0.5)));
15     for (int i=1;i<=n;i++)a[i]=rd(),b[i]=a[i];
16     sort(b+1,b+n+1);
17     tot=unique(b+1,b+n+1)-b-1;
18     for (int i=1;i<=n;i++)a[i]=lower_bound(b+1,b+tot+1,a[i])-b;
19     for (int i=1;i<=qy;i++){
20         q[i].l=rd();q[i].r=rd();q[i].id=i;
21     }
22     sort(q+1,q+qy+1);
23     for (int i=1;i<=qy;i++){
24         if (i==1||q[i].l/block_size!=q[i-1].l/block_size){
25             L=min(n+1,q[i].l/block_size*block_size+block_size);
26             R=L-1;
27             res=0;
28             for (int i=1;i<=tot;i++)cnt[i]=0;
29         }//l是新的一块
30         int l=q[i].l,r=q[i].r,id=q[i].id;
31         if (l/block_size==r/block_size){//暴力处理l,r在同一块的情况
32             ll tmp=0;
33             for (int j=l;j<=r;j++)tmp=max(tmp,1ll*b[a[j]]*(++cnt[a[j
                   ]]));
34             for (int j=l;j<=r;j++)cnt[a[j]]=0;
```

```
            ans[id]=tmp;
        }
        else{//处理l,r不在同一块情况，r单调递增，l每次处理完回撤
            while (R<r)++R,res=max(res,1ll*b[a[R]]*(++cnt[a[R]]));
            ll tmp=res;
            for (int j=l;j<L;j++)tmp=max(tmp,1ll*b[a[j]]*(++cnt[a[j
               ]]));
            for (int j=l;j<L;j++)--cnt[a[j]];
            ans[id]=tmp;
        }
    }
    for (int i=1;i<=qy;i++)printf("%lld\n",ans[i]);
    return 0;
}
```

64 计算一个数二进制位上 1 的个数

```
__builtin_popcount(n);
```

65 手写 Muitiset

比 *STL* 的 *multiset* 常数上较好，但没有 *lower_bound* 等函数，适合在需要一个简单的 *multiset* 时调用

```
struct Multiset{
        priority_queue<int>Q,T;
        void insert(int x){
                Q.push(x);
        }
        void erase(int x){
                T.push(x);
                while(!Q.empty()&&!T.empty()&&Q.top()==T.top())Q.pop
                   (),T.pop();
        }
        int top(){
                if(Q.empty())return -1e9;
                else return Q.top();
        }
};
```