目录

1 杂项

1.1 STL

1.1.1 bitset

```
bitset<maxn> S;

S.count(); // count 1
S.any(); // if exist 1
S.none(); // if not exist 1
S.all(); // if not exist 0
S.flip(i); // S[i]^=1

S._Find_first(); // first S[j]=1
S._Find_next(i); // first S[j]=1(j>i)
for (int i=S._Find_first();i<S.size();i=S._Find_next(i));
```

1.1.2 配对堆（dijkstra）

```
#include<ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
typedef __gnu_pbds::priority_queue<pair<int,int>,
                                   greater< pair<int,int> >,pairing_heap_tag> Heap;

Heap que;Heap::point_iterator it[maxn+5];bool vis[maxn+5];

inline void Dij(int S){ // n point, start from s
    while (!que.empty()) que.pop();
    for (int i=1;i<=n;i++) it[i]=0,vis[i]=false;
    for (int i=1;i<=n;i++) dis[i]=2e9;
    dis[S]=0;it[S]=que.push(mp(0,S));vis[S]=true;
    while (!que.empty()){
        int x=que.top().sc;que.pop();vis[x]=false;
        for (int j=lnk[0][x];j;j=nxt[j])
            if (dis[x]+w[j]<dis[son[j]]){
                dis[son[j]]=dis[x]+w[j];
                if (vis[son[j]]) que.modify(it[son[j]],mp(dis[son[j]],son[j]));
                else it[son[j]]=que.push(mp(dis[son[j]],son[j])),vis[son[j]]=true;
            }
    }
}
```

1.2 套路

1.2.1 等差数列乘积转阶乘

$$a(a+d)(a+2d)\cdots(a+nd)=\frac{a}{d}(\frac{a}{d}+1)(\frac{a}{d}+2)\cdots(\frac{a}{d}+n)\cdot d^{n+1}$$

质模数小时可以转为阶乘。

1.2.2 最小二乘法

拟合后直线为 $y=kx+b$ ： $k=\dfrac{n\sum_{i=1}^{n}x_iy_i-\left(\sum_{i=1}^{n}x_i\right)\left(\sum_{i=1}^{n}y_i\right)}{n\sum_{i=1}^{n}x_i^2-\left(\sum_{i=1}^{n}x_i\right)^2},b=\dfrac{\sum_{i=1}^{n}y_i-k\sum_{i=1}^{n}x_i}{n}$

1.2.3 小范围不重复数点

给 n 组点 (A,B,C) ，每个点覆盖 $\geq A,\geq B,\geq C$ 的部分，每组点覆盖的部分取并。
求 (i,j,k) 被几组覆盖。

```
const int maxa=...; // the range of (A,B,C)

int f[maxa+5][maxa+5][maxa+5];
int m;pair<int, pair<int,int> > a[maxm+5]; // a is array of points
set< pair<int,int> > S;set< pair<int,int> >::iterator it;

for (int i=1;i<=n;i++){
    readi(m);
    for (int i=1;i<=m;i++) readi(a[i].A),readi(a[i].B),readi(a[i].C);
    sort(a+1,a+1+m);
    S.clear();S.insert(mp(0,maxa+1));S.insert(mp(maxa+1,0));
    for (int i=1;i<=m;i++){
        int A=a[i].A;pair<int,int> p=a[i].sc;
        it=prev(S.upper_bound(p));
        if (it->sc<=p.sc) continue;
        Update(A,it,-1);
        for (it++;it->sc>=p.sc;it=S.erase(it)) Update(A,it,-1);
        it=S.insert(p).fr;
        Update(A,it,1);Update(A,--it,1);
    }
}
for (int i=1;i<=maxa;i++)
    for (int j=1;j<=maxa;j++)
        for (int k=1;k<=maxa;k++)
            f[i][j][k]+=f[i-1][j][k];
for (int i=1;i<=maxa;i++)
    for (int j=1;j<=maxa;j++)
        for (int k=1;k<=maxa;k++)
            f[i][j][k]+=f[i][j-1][k];
for (int i=1;i<=maxa;i++)
    for (int j=1;j<=maxa;j++)
        for (int k=1;k<=maxa;k++)
            f[i][j][k]+=f[i][j][k-1];
// f[i][j][k] is the answer of [1..i][1..j][1..k]
```

1.2.4 gcd,and,or 分块

如果对每个右端点 i ，记录 suf_j 表示 $[j,i]$ 的 gcd ，将 suf_j 相邻的相同值合并，最多只有 log 块，因为 gcd 变化时至少整除 2 。
对于已经维护好的右端点 i ，可以 $O(\log)$ 添加 $i+1$ 。
由于 and, or 也只会变化 log 次，因此也可以如上述处理。

1.2.5 子集分块

动态维护 $f_s=\sum_{t\subseteq S}a_t$ ，需要支持 a_x 修改操作（超集同理）。
如果全集 S 比较大，可以拆成两半，t 是 s 子集等价于两半都是 s 子集。
定义 $f_{a,b}=\sum_{i=a,j\subseteq b}a_{i\cup j}$ ，从 $O(S)$ 修改 $O(1)$ 查询变为 $O(\sqrt{S})$ 修改查询。
动态维护因子也可以这么处理，将 n 拆为 $\prod_k p_k^{w_k}$ ，并分为 $\prod(w_i+1)$ 尽可能接近的两半，建出因子关系后同上述处理。

1.2.6 二进制分组

假设动态加入 n 个可以合并的数据结构 D，并且查询时不需要在合并后的数据结构中查询，可以采用二进制分组。
每次加入一个 D 时，先在右端加入一个大小为 1 的组，然后查询 $Tail-1$ 和 $Tail$ 组大小是否一致，如果一致则暴力合并。
每个 D 只会被合并 $\log_2 n$ 次。

1.2.7 启发式分裂

如果一个块 $[L,R]$ 能按规则分为两个块 $[L,m],[m+1,R]$ 且支持继续拆分，则可以从两侧同时往内部枚举，一侧遇到合法位置即停止枚举，这样复杂度是 $O(n\log_2 n)$ 的，因为可以看作启发式合并的逆过程，每个点只会拆分 $O(\log_2 n)$ 次。

1.2.8 超级钢琴贪心模型

选 k 个长度在 $[L,R]$ 范围内的子段，求 k 个子段的和的最大值。
类似选出前 k 小的 a_i+a_j ，定义 $Max(i,L,R)$ 表示 i 作左端点，右端点在 $[L,R]$ 范围内的最大值，可以用 ST 表查询出 $Max(i,L,R)$ 中最优秀的右端点 t ，然后将 $Max(i,L,R)$ 拆为 $Max(i,L,t-1),Max(i,t+1,R)$ ，用堆取 k 次即可。

1.2.9 平方计数

求所有方案中 $\sum_S cnt(S,A)^2$ ，$cnt(S,A)$ 表示 A 在方案 S 中的出现次数。
可以转化为在 S 中可重复的选择两次 A 的方案数。

1.2.10 循环矩阵乘法

$$A_{i,j}=A_{0,(j-i)\bmod n}$$

$$(AB)_{0,k}=\sum_{i+j\equiv k\pmod n}A_{0,i}B_{0,j}$$

将 $A_{i,j}$ 简记为 A_i ，则 AB 就是 A 和 B 的循环卷积。

1.2.11 树上解方程

$$f(x)=\frac{1}{d_x}\sum_{(x,y)\in E} f(y)+w_x$$

可以设 $f(x)=f(fa_x)+a_x$ ，如果 $w_x=1$ ，则：

$$f(x)=f(fa_x)+sum_x$$

其中 sum_x 为 x 子树中节点度数的和，即 $2si_x-1$ 。

1.2.12 四连通解方程

在 $x\in[-R,R],y\in[-R,R]$ 的四连通网格中解方程：

$$f_{x,y}=p_{x,y,0}f_{x-1,y}+p_{x,y,1}f_{x,y-1}+p_{x,y,2}f_{x+1,y}+p_{x,y,3}f_{x,y+1}+a_{x,y}$$

将可用格子从左上角到右下角编号，则每行方程只涉及 $2R$ 距离内的变元，因此每次高斯消元不需要 $O(R^2)$ ，只需要 $O(R)$ 。

```
#define ID(x,y) (id[(x)+maxr][(y)+maxr])
DB Solve(){
    for (int i=1;i<=n;i++)
        for (int j=i+1,d=1;j<=n && d<=(R<<1);j++,d++){
            DB now=M[j][i]/M[i][i];
            for (int k=i,d=0;k<=n && d<=(R<<1);k++,d++) M[j][k]-=M[i][k]*now;
            M[j][0]-=M[i][0]*now;
        }
    for (int i=n;i;i--){
        ans[i]=M[i][0];
        for (int j=i+1;j<=n;j++) ans[i]-=M[i][j]*ans[j];
        ans[i]/=M[i][i];
    }
    return ans[ID(0,0)];
}

for (int x=-R;x<=R;x++) for (int y=-R;y<=R;y++) if (ok(x,y)) ID(x,y)=++n;
for (int x=-R;x<=R;x++)
    for (int y=-R;y<=R;y++){
        if (!ID(x,y)) continue;
        M[ID(x,y)][ID(x,y)]= ... ;
        M[ID(x,y)][0]= ... ;
        for (int f=0,xx,yy;f<4;f++)
            if (ID(xx=x+df[f][0],yy=y+df[f][1]))
                M[ID(x,y)][ID(xx,yy)]= ... ;
    }
```

1.2.13 去绝对值

由于 $|x-y|=\max\{x-y,y-x\}$ ，如果要求最大值，则不合法的情况（负数）一定不会选。
枚举 $|x-y|$ 的正负情况后即可分别求 $\max\{\sum x\},\max\{\sum y\}$ 。

1.2.14 随机赋权判奇偶性

利用随机赋权 + 异或，可以判整体奇偶性。

例：有 $n \times m$ 矩阵，每一行都有一个区间 $[L_i, R_i]$ 被覆盖，问多少个 $[A, B]$ 满足任意一行 $[A, B]$ 被覆盖的次数都是奇数/偶数。

给每行的 $[L_i, R_i]$ 随机一个权值，利用差分求出 a_i 表示经过 i 的线段的权值异或和，d_i 表示 $L \le i$ 的权值异或和，记录 sum_i 为 a_i 的前缀异或和。

均出现偶数次：sum_B xor $sum_{A-1} = 0$ 。

均出现奇数次：sum_B xor $sum_{A-1} = d_B$ xor d_A xor a_A ，即 sum_B xor $sum_A = d_B$ xor d_A 。

1.2.15 FFT 字符串匹配

一般匹配：$\sum_{i=0}^{len-1}(A_i - B_i)^2 = 0$ 则匹配成功。

通配符匹配：如果串 s 第 i 位是通配符，则令 $S_i = 0$ ，$\sum_{i=0}^{len-1}(A_i - B_i)^2 A_i B_i = 0$ 则匹配成功。

K 次容错匹配：对于每个字符 ch ，$S_i = [s_i = ch]$ ，$\sum_{i=0}^{len-1} A_i B_i$ 即 ch 字符为 A, B 提供的匹配数量。最后查询总匹配数量。

以上三个式子均可以对每个位置展开为卷积形式，然后用 FFT 快速求解。

1.3 奇技淫巧

1.3.1 读入输出优化

```
#define EOLN(x) ((x)==10 || (x)==13 || (x)==EOF)
inline char readc(){
    static char buf[1<<16],*l=buf,*r=buf;
    return l==r && (r=(l=buf)+fread(buf,1,1<<16,stdin),l==r)?EOF:*l++;
}
template<typename T> int readi(T &x){
    T tot=0;char ch=readc(),lst='+';
    while (!isdigit(ch)) {if (ch==EOF) return EOF;lst=ch;ch=readc();}
    while (isdigit(ch)) tot=(tot<<3)+(tot<<1)+(ch^48),ch=readc();
    lst=='-'?x=-tot:x=tot;return EOLN(ch);
}
struct fastO{
    int si;char buf[1<<16];
    fastO() {si=0;}
    void putc(char ch){
        if (si==(1<<16)) fwrite(buf,1,si,stdout),si=0;
        buf[si++]=ch;
    }
    ~fastO() {fwrite(buf,1,si,stdout);}
}fo;
#define putc fo.putc
template<typename T> void writei(T x,char ch='\n'){
    static int len=0,buf[100];
    if (x<0) putc('-'),x=-x;
    do buf[len++]=x%10,x/=10; while (x);
    while (len) putc(buf[--len]+48);
    if (ch) putc(ch);
}
```

1.3.2 模数非素数无法求逆元

如果模数为非素数 MOD，但是整数运算最后要除以 n（且保证结果也是整数）。
可以运算时将模数改为 $MOD \cdot n$，就可以除以 n 了。

1.3.3 模拟退火

```
#define Rand() ((DB)rand()/RAND_MAX)
inline void Anneal(DB T){
    Clear(State);
    for (DB res;T>1e-14;T*=0.99){
        nextState=Change(State); // Change range is influence by T
        res=getres(State);
        if (res<ans) ans=res,ansState=State=nextState; else
        if (exp((ans-res)/T)>Rand()) State=nextState;
    }
}
```

1.3.4 中位数最小转权值最小

n 个元素，选出最少的个数满足条件 A，并且最少的前提下，权值中位数最小。
二分中位数，小于等于中位数的权值给 $3n-1$，大于中位数的权值给 $3n+1$，这样权值和最小就等价于选取最少（因为这样少选一个一定比多选一个权值和小）。
求出最小权值 ans，二分验证条件是 $3n-1$ 的个数大于等于 $3n+1$ 的个数。易知选取个数为 $cnt = \lfloor \frac{ans+n}{3n} \rfloor$，因此验证条件即 $ans \le 3n \cdot cnt$。

1.3.5 线性求逆元

求出 $s_i = \prod_{j=1}^{i} a_j, sv_n = (s_n)^{-1}, sv_i = sv_{i+1}a_{i+1}$。
则 $(a_i)^{-1} = s_{i-1}sv_i$。

1.3.6 线性快速幂

要求 a^n，令 $S = \lceil \sqrt{n} \rceil$，预处理 a^{iS} 和 a^j，则 $a^n = a^{iS} \cdot a^j$，预处理复杂度 $O(\sqrt{n})$。

1.3.7 vscode 配置

快捷键：ctrl+shift+b 改为 ctrl+f9。
配置：Auto Closing Brackets 和 Auto Closing Delete 改为 always，Insert Spaces 取消勾选。
终端-配置默认生成任务，运行-添加配置。

1.3.8 Linux 对拍

调用方法：sh check.sh

```
while true; do
  ./maker
  ./Correct
  ./Wrong
  if diff Correct.out Wrong.out; then
    printf "AC\n";
  else
    printf "WA\n";
    exit 0;
  fi
done
```

2 动态规划

2.1 背包回退

```
for (int i=a[x];i<=maxv;i++) g[i]=ADD(g[i],MOD-g[i-a[x]]); // limited
for (int i=maxv;i>=a[i];i--) g[i]=ADD(g[i],MOD-g[i-a[x]]); // unlimited
```

2.2 树形 DP

2.2.1 树形背包优化

```
void DP(int x,int pre=0){
    si[x]=1;
    for (int j=lnk[x],u;j;j=nxt[j])
        if ((u=to[j])!=pre){
            DP(u,x);
            for (int i=min(si[x]+si[u],K);~i;i--)
                for (int k=max(i-si[x],0);k<=si[u] && k<=i;k++)
                    f[x][i]=max(f[x][i],f[x][i-k]+f[u][k]+val(k));
            si[x]+=si[u];
        }
}
```

2.2.2 路径有方向

求一条最优路径，但如果一条边行走方向不同，产生的权值不同。
定义 f_i 表示向上走到 i 的状态，g_i 表示 i 向下走的状态。
DFS 时正着枚举儿子做一遍，倒着枚举儿子做一遍，即可考虑到两个儿子 $u \to v, v \to u$ 两种方向。

2.2.3 DFS 序上 DP

树形背包，每个节点有 s_x 个体积 w_x 价值 p_x 的物品。
但需要满足儿子选了至少一个的话，则父亲也必须至少选一个。
定义 $f_{i,j}$ 表示 DFS 序 $[1,i]$ 体积为 j 的最优解，这样转移就只有两种：
1. i 点选，那么转移到 $f[i+1][k]$ 。
2. i 点不选，那么子树也没有选，转移到 $f[out[i]+1][k]$ 。

```
for (int i=1,k=ID[i];i<=n;k=ID[++i]){
    for (int j=0;j<=m;j++) f[rt[k]+1][j]=max(f[rt[k]+1][j],f[i][j]);
    for (int j=m;~j;j--)
        if (j>=w[k] && f[i][j-w[k]]>=0)
            f[i][j]=f[i][j-w[k]]+p[k]; else f[i][j]=INF;
    s[k]--;
    for (int t=1;t<s[k];s[k]-=t,t<<=1)
        for (int j=m,W=t*w[k],P=t*p[k];j>=W;j--)
            f[i][j]=max(f[i][j],f[i][j-W]+P);
    if (s[k]) for (int j=m,W=s[k]*w[k],P=s[k]*p[k];j>=W;j--)
                  f[i][j]=max(f[i][j],f[i][j-W]+P);
    for (int j=0;j<=m;j++) f[i+1][j]=max(f[i+1][j],f[i][j]);
}
```

2.3 状压 DP

2.3.1 斯坦纳树

一般问题

在 $n \times m$ 的网格上有 k 个景点，选取每个网格都有代价，求最小代价使得景点之间全部连通。

定义 $f_{i,j,s}$ 表示当 i,j 与 s 这些景点连通时的最优解，那么有两种方法转移：

1. 通过子集转移：$f_{i,j,t} + f_{i,j,s-t} - cost_{i,j} \to f_{i,j,s}$ ，减去 $cost_{i,j}$ 是因为算重复了。
2. 通过相邻点转移：$f_{x,y,s} + cost_{i,j} \to f_{i,j,s}$ 。

第二种转移是最短路形式，用 SPFA 进行，每次先用第一种转移处理出 $f_{i,j,s}$ ，然后把 $f_{i,j,s}$ 加进队列，对每个 s 都做一遍 SPFA。

```
inline void Spfa(int s){
    while (Head!=Tail){
        pair<int,int> now=que[Head=AM(Head)];
        int x=now.fr,y=now.sc;vis[x][y]=false;
        for (int t=0;t<4;t++){
            int xx=x+dif[t][0],yy=y+dif[t][1];
            if (xx<1 || xx>n || yy<1 || yy>m) continue;
            if (f[x][y][s]+cst[xx][yy]<f[xx][yy][s]){
                f[xx][yy][s]=f[x][y][s]+cst[xx][yy];
                if (!vis[xx][yy]) que[Tail=AM(Tail)]=mp(xx,yy),vis[xx][yy]=true;
            }
        }
    }
}

for (int s=1;s<(1<<K);s++){
    Head=0;Tail=0;
    for (int i=1;i<=n;i++)
        for (int j=1;j<=m;j++){
            for (int t=(s-1)&s;t;t=(t-1)&s)
                f[i][j][s]=min(f[i][j][s],f[i][j][t]+f[i][j][s^t]-cst[i][j]);
            if (f[i][j][s]<INF) que[Tail=AM(Tail)]=mp(i,j),vis[i][j]=true;
        }
    Spfa(s);
}
```

随机映射

$n \times m$ 网格，每个格点颜色为 $c_{i,j}$ ，权值 $cost_{i,j}$ 。选取一个最小权值连通块使得至少选了 K 种颜色。

若 K 小，颜色集大，可以将颜色集随机映射到 $[1,K]$ 内，然后做斯坦纳树，这样单次正确概率为 $\frac{K!}{K^K}$ 。

2.3.2 DP 套 DP

给定一个状态 S ，可用 DP f ，求其最优解为 $f(S)$ 。问多少个 S 满足 $f(S) = F$ 。

将 f 的 DP 状态记录下来，可以得到一个 DAG，外层 DP 储存在 DAG 的位置。

第 a 步，当状态 A 添加元素时，即从 $f(A) \to f(B)$ 。

将 $f(A)$ 的 DP 状态压缩为 s_A ，令外层 DP g_{a,s_A} ，则 $g_{a,s_A} \to g_{a+1,s_B}$ 进行转移。

2.3.3 插头 DP

记录轮廓线上的插头状态，处理当前格子时将左上角轮廓更新为右下角轮廓。

哈密顿回路

任意条哈密顿回路铺满网格：记录 $0,1$ 表示是否有插头。

一条哈密顿回路铺满网格：记录 $0,1,2$ 表示没有插头，左括号插头，右括号插头。

2.4 斜率优化

若 DP 在 j 比 k 优秀时满足以下式子（$j<i,k<i,X_k<X_j$ ）：

$$f_j+val(j)-[f_k+val(k)]\le A_i(X_j-X_k)\Rightarrow\frac{Y_j-Y_k}{X_j-X_k}\le A_i$$

记 $K(a,b)=\frac{Y_b-Y_a}{X_b-X_a}(X_a<X_b)$ ，则 $K(k,j)\le A_i$ 说明 j 比 k 优秀，$K(k,j)>A_i$ 说明 k 比 j 优秀。

推论：若 $K(k,j)\ge K(j,t)$ ，则 j 对任何 i 都必然不优秀。

证明：若 $K(j,t)\le A_i$ ，则 t 比 j 优秀。若 $K(j,t)>A_i$ ，则 $K(k,j)>A_i$ ，k 比 j 优秀。

因此有用的点必须满足 $K(que_{i-1},que_i)<K(que_i,que_{i+1})$ ，即 que 构成一个下凸壳。

2.4.1 队列维护

若 X_i,A_i 均递增，可以用队列维护凸壳。

更新点 i 时，判断队首，如果 $K(que_{head},que_{head+1})\le A_i$ 则 que_{head} 失效（因为 A_i 只会增大）。

更新完成后 que_{head} 即最优解。

加入点 i 时，判断队尾，如果 $K(que_{tail-1},que_{tail})\ge K(que_{tail},i)$ 则队尾失效。

2.4.2 cdq 分治

若 X_i,A_i 有非递增，可以用 cdq 分治。

求出 $[L,mid]$ 的答案后，构造出左边的凸壳，然后在凸壳上二分斜率，更新 $[mid+1,R]$ 的答案。

当然也可以平衡树动态维护凸壳。

```
inline bool cmp(const int &i,const int &j) {return mp(X(i),Y(i))<mp(X(j),Y(j));}
void Solve(int L,int R){
    if (L==R) return;
    int mid=L+(R-L>>1);
    Solve(L,mid);
    int m=0;for (int i=L;i<=mid;i++) ID[++m]=i;
    sort(ID+1,ID+m+1,cmp);
    /* Make convex hull */
    if (!top) {Solve(mid+1,R);return;}
    for (int i=mid+1;i<=R;i++){
        int l=1,r=top-1;
        for (int mid=l+(r-l>>1);l<=r;mid=l+(r-l>>1))
            K(stk[mid],stk[mid+1])>=A(i)?r=mid-1:l=mid+1;
            // stk[mid] is better than stk[mid+1]
        f[i]=min(f[i],f[stk[l]]+val(stk[l]));
    }
    Solve(mid+1,R);
}
```

2.5 决策单调性

若 DP 对于任意的 $i<j<k<t$ 且对于 k 来说 j 比 i 优秀，都满足对于 t 来说 j 比 i 优秀，则该 DP 满足决策单调性。

令 pos_i 表示转移到 i 的最优位置，则 pos 是单调不降的（因为取到 pos_i 说明前面都没有 pos_i 优秀，那么对于 i 后面的肯定也是不优秀的）。

维护队列，队列元素 p_i,l_i,r_i 表示 $[l_i,r_i]$ 的 pos 都为 p_i ，每次考虑加入 i 这个决策点，由于 pos 单调，因此可以二分找到从哪个位置开始 pos 变为 i 。

```
#define val(j,i) (f[j]+Sum((j)+1,(i))) // DP transfer

Head=1;Tail=0;
p[++Tail]=0;l[Tail]=1;r[Tail]=n;
for (int i=1;i<=n;i++){
    f[i]=val(p[Head],i);
    int lst=-1;
    while (Head<=Tail)
        if (val(p[Tail],l[Tail])>val(i,l[Tail])) lst=l[Tail--]; else {
            int L=l[Tail],R=r[Tail];
            for (int mid=L+(R-L>>1);L<=R;mid=L+(R-L>>1))
                val(p[Tail],mid)>val(i,mid)?R=mid-1:L=mid+1;
            if (L<=r[Tail]) lst=L,r[Tail]=L-1;
            break;
        }
    if (~lst) p[++Tail]=i,l[Tail]=lst,r[Tail]=n;
    if (Head<=Tail) {l[Head]++;if (l[Head]>r[Head]) Head++;}
}
```

3 数据结构

3.1 线段树

3.1.1 求上升序列

初始 $val=-INF$ ，遍历 $\{a_n\}$ ，如果 $a_i \geq val$ ，更新 val 为 a_i ，问更新次数。
定义 $Find(p,k)$ 表示在 p 节点前 $val=k$ 得到的 val 变大的次数。
如果 $k > MAX_{ls_p}$ ，可以直接查询 $Find(rs_p,k)$ 。否则查询 $Find(ls_p,k)+Find(rs_p,MAX_{ls_p})$ 。
$Find(rs_p,MAX_{ls_p})$ 是一个固定值，记录下来之后每次就只会查询一边了。

```
inline int Maxer(int x,int y) {return a[x]>a[y] || a[x]==a[y] && x>y?x:y;}
pair<int,int> Find(int L,int R,int p,int who){
    if (L==R) return a[L]>=a[who]?mp(1,L):mp(0,who);
    int mid=L+(R-L>>1);
    if (a[who]>a[MAX[p<<1]]) return Find(mid+1,R,p<<1|1,who); else {
        pair<int,int> ls=Find(L,mid,p<<1,who);
        return mp(ls.fr+res[p].fr,res[p].sc);
    }
}
void Build(int L,int R,int p=1){
    if (L==R) {MAX[p]=L;return;}
    int mid=L+(R-L>>1);
    Build(L,mid,p<<1);Build(mid+1,R,p<<1|1);
    MAX[p]=Maxer(MAX[p<<1],MAX[p<<1|1]);
    res[p]=Find(mid+1,R,p<<1|1,MAX[p<<1]);
}
void Update(int pos,int l=1,int r=n,int p=1){
    if (l==r) return;
    int mid=l+(r-l>>1);
    pos<=mid?Update(pos,l,mid,p<<1):Update(pos,mid+1,r,p<<1|1);
    MAX[p]=Maxer(MAX[p<<1],MAX[p<<1|1]);
    res[p]=Find(mid+1,r,p<<1|1,MAX[p<<1]);
}
```

3.1.2 线段树上二分

```
void Find(int L,int R,int l=1,int r=n,int p=1){
    if (answer is already found) return;
    if (L==l && r==R){
        if ([l,r] is impossible) return;
        if (l==r) {answer is found; return;} // single point
        int mid=l+(r-l>>1); Pushdown(p);
        if (...) Find(L,mid,l,mid,p<<1); else Find(mid+1,R,mid+1,r,p<<1|1);
        return;
    }
    int mid=l+(r-l>>1); Pushdown(p);
    if (R<=mid) Find(L,R,l,mid,p<<1); else if (L>mid) Find(L,R,mid+1,r,p<<1|1); else{
        Find(L,mid,l,mid,p<<1);
        if (answer is already found) return;
        Find(mid+1,R,mid+1,r,p<<1|1);
    }
}
```

3.1.3 分裂合并

```
int ro[maxn+5];bool vis[maxn+5]; // 0 inc 1 dec
map<int,int> S;map<int,int>::iterator l,r,it; // S[x] is the right endpoint of x

void Update(int p,int k){
    // Global update
}
inline int newnode() {} // Clear
inline void Pushup(int p){
    si[p]=si[ls[p]]+si[rs[p]];
    // Update
}
inline void Addflip(int p) {if (p) swap(ls[p],rs[p]),flip[p]^=1;} // fliptag
void Pushdown(int p) {if (flip[p]) Addflip(ls[p]),Addflip(rs[p]),flip[p]=0;}
void Insert(int &p,int pos,int l=1,int r=n){
    if (!p) p=newnode();
    if (l==r) {si[p]++;return;}
    int mid=l+(r-l>>1);
    Pushdown(p);
    pos<=mid?Insert(ls[p],pos,l,mid):Insert(rs[p],pos,mid+1,r);
    Pushup(p);
}
void SplitR(int &p,int q,int k){ // Split q's last k elements
    if (!p) p=newnode();
    Pushdown(q);
    int now=si[rs[q]];
    if (k>=now){
        rs[p]=rs[q];rs[q]=0;
        if (k-=now) SplitR(ls[p],ls[q],k);
    } else SplitR(rs[p],rs[q],k);
    Pushup(p);Pushup(q);
}
// Split [L,R] from [fr,sc], and sort [L,R] to tp
void Split(int fr,int sc,int L,int R,int tp){
    if (R<sc){
        ro[R+1]=0;SplitR(ro[R+1],ro[fr],sc-R);
        vis[R+1]=vis[fr];
        S[fr]=R;Update(fr,ro[fr]);
        S[R+1]=sc;Update(R+1,ro[R+1]);
    }
    if (L>fr){
        ro[L]=0;SplitR(ro[L],ro[fr],R-L+1);
        vis[L]=vis[fr];
        S[fr]=L-1;Update(fr,ro[fr]);
        S[L]=R;Update(L,ro[L]);
    }
    if (vis[L]!=tp) Addflip(ro[L]),vis[L]=tp,Update(L,ro[L]);
}
void Merge(int &x,int y,int l=1,int r=n){
    if (!y) return;if (!x) {x=y;return;}
    int mid=l+(r-l>>1);
```

```
    Pushdown(x);Pushdown(y);
    Merge(ls[x],ls[y],l,mid);Merge(rs[x],rs[y],mid+1,r);
    Pushup(x);
}
void AskLR(int p,int L,int R){ // Ask from L-th to R-th
    if (!p) return;
    if (L==1 && R==si[p]) return AskAnswer(); // Ask answer
    Pushdown(p);
    int now=si[ls[p]];
    if (R<=now) AskLR(ls[p],L,R); else if (L>now) AskLR(rs[p],L-now,R-now);
    else AskLR(ls[p],L,now),AskLR(rs[p],1,R-now);
}

// Sort [L,R] to tp
l=prev(S.upper_bound(L));r=prev(S.upper_bound(R));
int A=l->fr,B=l->sc,C=r->fr,D=r->sc;
if (A==C) Split(A,B,L,R,tp); else{
    Split(A,B,L,B,tp);
    Split(C,D,C,R,tp);
    for (it=S.lower_bound(L+1);it!=S.end() && it->sc<=R;it=S.erase(it)){
        if (vis[it->fr]!=tp) Addflip(ro[it->fr]);
        Merge(ro[L],ro[it->fr]);Update(it->fr,0);
    }
    S[L]=R;Update(L,ro[L]);
}

// Ask [L,R]
l=prev(S.upper_bound(L));r=prev(S.upper_bound(R));
int A=l->fr,B=l->sc,C=r->fr,D=r->sc;
if (l==r) AskLR(ro[A],L-A+1,R-A+1); else {
    AskLR(ro[A],L-A+1,B-A+1);
    if (B+1<=C-1) Ask(B+1,C-1); // Global Ask
    AskLR(ro[C],1,R-C+1);
}
```

3.1.4 维护平方和以及历史平方和

记 len 表示区间长度，sum 表示区间和，val 表示区间平方和，pre 表示到目前为止的前缀区间平方和，然后我们考虑区间 $+k$ 操作：$x^2 \to (x+k)^2 = x^2 + 2kx + k^2$

$$
\begin{aligned}
len &= len' \\
sum &= sum' + k \cdot len \\
val &= val' + 2k \cdot sum' + k^2 \cdot len \\
pre &= pre' + val' + 2k \cdot sum' + k^2 \cdot len
\end{aligned}
$$

如果仅记录数值 tag 我们不难发现没有办法进行标记合并，观察上面的式子发现这是一次线性变换，因此我们可以用矩阵来表示：

$$
\begin{bmatrix} len' & sum' & val' & pre' \end{bmatrix}
\begin{bmatrix} 1 & k & k^2 & k^2 \\ 0 & 1 & 2k & 2k \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
= \begin{bmatrix} len & sum & val & pre \end{bmatrix}
$$

所以我们可以将这个矩阵记为 tag ，这样就能够标记合并了。

这样常数很大，由于这个转移矩阵是上三角的，因此矩阵乘法时我们只需要枚举上三角，常数小很多。

实现时需要注意，为了对齐时间（求 pre ），不仅需要对 $[L,R]$ 进行 $+k$ 操作，还需要对 $[1,L-1],[R+1,n]$ 进行 $+0$ 操作。

可把 len 定义改为 si 表示区间中存在点的个数，从而解决某些问题。

```
struct Matrix{
    int s[4][4];
    void zero() {for (int i=0;i<4;i++) for (int j=0;j<4;j++) s[i][j]=0;}
    void unit() {zero();for (int i=0;i<4;i++) s[i][i]=1;}
};
int M[maxt+5][4],cnt[maxt+5];Matrix tag[maxt+5],tem;
#define ADD(x,y) (((x)+(y))%MOD)
#define MUL(x,y) ((LL)(x)*(y)%MOD)
Matrix operator * (const Matrix &a,const Matrix &b){
    static Matrix c;c.zero();
    for (int i=0;i<4;i++)
        for (int j=i;j<4;j++)
            for (int k=i;k<=j;k++)
                c.s[i][j]=ADD(c.s[i][j],MUL(a.s[i][k],b.s[k][j]));
    return c;
}
void Make(int k){ // the matrix of tag k
    tem.unit();
    tem.s[0][1]=k;tem.s[0][2]=MUL(k,k);tem.s[0][3]=MUL(k,k);
    tem.s[1][2]=ADD(k,k);tem.s[1][3]=ADD(k,k);
    tem.s[2][3]=1;
}
void Mul(int *a,Matrix &A){
    static int b[4];
    b[0]=b[1]=b[2]=b[3]=0;
    for (int i=0;i<4;i++)
        for (int j=0;j<4;j++)
            b[i]=ADD(b[i],MUL(a[j],A.s[j][i]));
    a[0]=b[0];a[1]=b[1];a[2]=b[2];a[3]=b[3];
}
void Build(int L,int R,int p=1){
    M[p][0]=R-L+1;tag[p].unit();
    if (L==R) {M[p][1]=a[L];M[p][2]=MUL(a[L],a[L]);M[p][3]=M[p][2];return;}
    int mid=L+(R-L>>1);
    Build(L,mid,p<<1);Build(mid+1,R,p<<1|1);
    for (int i=1;i<4;i++) M[p][i]=ADD(M[p<<1][i],M[p<<1|1][i]);
}
inline void Addtag(int p,Matrix &A) {Mul(M[p],A);tag[p]=tag[p]*A;cnt[p]++;}
void Pushdown(int p) {
    if (cnt[p]) Addtag(p<<1,tag[p]),Addtag(p<<1|1,tag[p]),cnt[p]=0,tag[p].unit();
}
void Insert(int L,int R,int l=1,int r=n,int p=1){
    if (L==l && r==R) return Addtag(p,tem);
    int mid=l+(r-l>>1);Pushdown(p);
    if (R<=mid) Insert(L,R,l,mid,p<<1); else if (L>mid) Insert(L,R,mid+1,r,p<<1|1);
    else Insert(L,mid,l,mid,p<<1),Insert(mid+1,R,mid+1,r,p<<1|1);
    for (int i=1;i<4;i++) M[p][i]=ADD(M[p<<1][i],M[p<<1|1][i]);
```

```
}
int Ask(int L,int R,int l=1,int r=n,int p=1){
    if (L==l && r==R) return M[p][3];
    int mid=l+(r-l>>1);Pushdown(p);
    if (R<=mid) return Ask(L,R,l,mid,p<<1);
    else if (L>mid) return Ask(L,R,mid+1,r,p<<1|1);
    else return ADD(Ask(L,mid,l,mid,p<<1),Ask(mid+1,R,mid+1,r,p<<1|1));
}
// use hint
if (l[i]>1) Make(0),Insert(1,l[i]-1);
Make(X[i]);Insert(l[i],r[i]);
if (r[i]<n) Make(0),Insert(r[i]+1,n);
```

3.1.5 李超线段树

对于每个线段树节点，记录“覆盖最广线段”(k,b)（顾名思义就是作为最高线段的区间最长的线段，不难发现询问 $x=k$ 的答案一定在包含 k 的节点的“覆盖最广线段”中）。然后对于一个节点 old，当有新的线段 new 出现的时候，我们需要修改标记：

交点在左边的时候，说明 new 是“覆盖最广线段”，先把 old 当做新的线段去修改左边，然后将 old 修改为 new。

交点在右边的时候，说明 old 是“覆盖最广线段”，把 new 当做新的线段去修改右边。

```
inline Line Add(int a,int b,int c,int d){
    if (a==c) return Line(0,max(b,d),++tot);
    DB k=(DB)(d-b)/(c-a);return Line(k,b-k*a,++tot);
}
inline int fcmp(DB a,DB b) {if (fabs(a-b)<1e-8) return 0;if (a<b) return -1;return 1;}
#define Y(l,x) (l.k*(x)+l.b)
inline bool cmp(Line a,Line b,int pos){
    if (!fcmp(Y(a,pos),Y(b,pos))) return a.ID<b.ID;
    return fcmp(Y(a,pos),Y(b,pos))>0;
}
void Pushdown(Line x,int l,int r,int p){
    if (!tr[p].ID) {tr[p]=x;return;}if (cmp(x,tr[p],l)) swap(x,tr[p]);
    if (l==r||!fcmp(x.k,tr[p].k)) return;DB pos=(x.b-tr[p].b)/(tr[p].k-x.k);
    if (r<pos||pos<l) return;int mid=l+(r-l>>1);
    if (pos<=mid) Pushdown(tr[p],l,mid,LS),tr[p]=x; else Pushdown(x,mid+1,r,RS);
}
void Insert(int L,int R,Line x,int l=1,int r=maxn,int p=1){
    if (R<l||r<L) return;if (L<=l&&r<=R) return Pushdown(x,l,r,p);
    int mid=l+(r-l>>1);Insert(L,R,x,l,mid,LS);Insert(L,R,x,mid+1,r,RS);
}
```

3.1.6 主席树求 mex 以及所有子区间 mex 的 mex

开 n 个权值主席树，第 i 棵树记录 $1 \sim i$ 中 x 最后一次出现的位置 pos_x 。询问 $[L,R]$ 中的 mex 时，在第 R 棵树中查询，如果左儿子最小的 $pos < L$ ，就说明左儿子中有数没有出现，因此往左儿子查找，否则往右儿子查找。

对于 x ，我们以 $a_i = x$ 的点为分割点，把数组分成好多块，如果任意一个块的 mex 等于 x ，就说明 x 合法。

最后求所有合法 x 的 mex 就行了。

（示例代码中 mex 定义为最小正整数，定义为自然数时需要更改权值线段树边界）

```cpp
#include<cstdio>
#include<algorithm>
using namespace std;
const int maxn=100000,maxt=1e7;

int n,a[maxn+5],ID[maxn+5],ans;bool ok[maxn+5];
int si,ro[maxn+5],ls[maxt+5],rs[maxt+5],MIN[maxt+5];

inline bool cmp(const int &i,const int &j) {return a[i]<a[j] || a[i]==a[j] && i<j;}
int Build(int L,int R){
    int now=++si;if (L==R) return now; int mid=L+(R-L>>1);
    ls[now]=Build(L,mid);rs[now]=Build(mid+1,R);return now;
}
int Insert(int p,int pos,int k,int l=1,int r=n+1){ // notice the range [1,n+1]
    int now=++si;ls[now]=ls[p];rs[now]=rs[p];MIN[now]=MIN[p];
    if (l==r) {MIN[now]=k;return now;} int mid=l+(r-l>>1);
    if (pos<=mid) ls[now]=Insert(ls[p],pos,k,l,mid);
    else rs[now]=Insert(rs[p],pos,k,mid+1,r);
    MIN[now]=min(MIN[ls[now]],MIN[rs[now]]); return now;
}
int Mex(int p,int k,int l=1,int r=n+1){
    if (l==r) return l; int mid=l+(r-l>>1);
    if (MIN[ls[p]]<k) return Mex(ls[p],k,l,mid); else return Mex(rs[p],k,mid+1,r);
}
int main(){
    scanf("%d",&n);ro[0]=Build(1,n+1);
    for (int i=1;i<=n;i++) scanf("%d",&a[i]),ID[i]=i,ro[i]=Insert(ro[i-1],a[i],i);
    sort(ID+1,ID+1+n,cmp);
    for (int L=1,R;L<=n;L=R+1){
        int now=a[ID[L]];for (R=L+1;R<=n && a[ID[R]]==now;R++);R--;
        if (ID[L]>1 && Mex(ro[ID[L]-1],1)==now) goto OK;
        for (int i=L;i<R;i++)
            if (ID[i]+1<=ID[i+1]-1 && Mex(ro[ID[i+1]-1],ID[i]+1)==now) goto OK;
        if (ID[R]<n && Mex(ro[n],ID[R]+1)==now) goto OK;
        continue;OK:ok[a[ID[L]]]=true;
    }
    ok[Mex(ro[n],1)]=true;ans=1;while (ok[ans]) ans++;
    printf("%d\n",ans);return 0;
}
```

3.2 LCT

3.2.1 正常版

```
#define is_ro(p) (son[fa[p]][0]!=(p) && son[fa[p]][1]!=(p))
#define Son(p) ((p)==son[fa[p]][1])
void Pushup(int p) {si[p]=si[son[p][0]]+1+si[son[p][1]];}
void Rotate(int t){
    int p=fa[t],d=Son(t);son[p][d]=son[t][d^1];son[t][d^1]=p;
    if (!is_ro(p)) son[fa[p]][Son(p)]=t;Pushup(p);Pushup(t);
    if (son[p][d]) fa[son[p][d]]=p;fa[t]=fa[p];fa[p]=t;
}
void Addflip(int p) {swap(son[p][0],son[p][1]);flip[p]^=1;}
void Pushdown(int p) {if (flip[p]) Addflip(son[p][0]),Addflip(son[p][1]),flip[p]^=1;}
void Splay(int p){
    static int top,stk[maxn+5];stk[top=1]=p;
    for (int i=p;!is_ro(i);i=fa[i]) stk[++top]=fa[i];
    while (top) Pushdown(stk[top--]);
    for (int pre=fa[p];!is_ro(p);Rotate(p),pre=fa[p])
        if (!is_ro(pre)) Rotate(Son(p)==Son(pre)?pre:p);
}
void Access(int p) {for (int lst=0;p;lst=p,p=fa[p]) Splay(p),son[p][1]=lst,Pushup(p);}
void Makero(int x) {Access(x);Splay(x);Addflip(x);}
void Link(int x,int y) {Makero(x);Makero(y);fa[x]=y;}
void Cut(int x,int y) {Makero(x);Access(y);Splay(x);son[x][1]=fa[y]=0;Pushup(x);}
int Ask(int x) {Makero(n);Access(x);Splay(x);return si[x];}
int LCA(int x,int p){
    int ans;Access(x);
    for (int lst=0;p;lst=p,p=fa[p]) Splay(p),son[p][1]=lst,Pushup(p),ans=p;
    return ans;
}
```

3.2.2 缩边双版

```
#define fa(p) (B.getfa(fa[p]))
#define is_ro(p) ((p)!=son[fa(p)][0] && (p)!=son[fa(p)][1])
#define Son(p) ((p)==son[fa(p)][1])
inline void Pushup(int p) {sum[p]=sum[son[p][0]]+val[p]+sum[son[p][1]];}
inline void Rotate(int t){
    int p=fa(t),d=Son(t);son[p][d]=son[t][d^1];son[t][d^1]=p;
    Pushup(p);Pushup(t);if (!is_ro(p)) son[fa(p)][Son(p)]=t;
    if (son[p][d]) fa[son[p][d]]=p;fa[t]=fa(p);fa[p]=t;
}
inline void Addflip(int p) {swap(son[p][0],son[p][1]);flip[p]^=1;}
inline void Pushdown(int p) {
    if (flip[p]) flip[p]^=1,Addflip(son[p][0]),Addflip(son[p][1]);
}
inline void Splay(int p){
    static int top,stk[maxn+5];stk[top=1]=p;
    for (int i=p;!is_ro(i);i=fa(i)) stk[++top]=fa(i);
    while (top) Pushdown(stk[top--]);
    for (int pre=fa(p);!is_ro(p);Rotate(p),pre=fa(p))
        if (!is_ro(pre)) Rotate(Son(p)==Son(pre)?pre:p);
```

```
}
inline void Access(int p) {
    for (int lst=0;p;lst=p,p=fa(p)) Splay(p),son[p][1]=lst,Pushup(p);
}
inline void Makero(int x) {Access(x);Splay(x);Addflip(x);}
void DFS(int x,int y){
    if (son[x][0]) val[y]+=val[son[x][0]],DFS(son[x][0],y);
    if (son[x][1]) val[y]+=val[son[x][1]],DFS(son[x][1],y);
    B.fat[x]=y;son[x][0]=son[x][1]=0; // B is LCT dsu
}
inline void Link(int x,int y){
    int fx=A.getfa(x),fy=A.getfa(y); // A is global dsu
    if (fx!=fy) {Makero(x);fa[x]=y;A.fat[fx]=fy;return;}
    Makero(x);Access(y);Splay(y);DFS(y,y);Pushup(y);
}
```

3.2.3 维护子树信息

LCT 实链上的信息可以方便维护，所以我们关注虚边：虚边只在 Access 和 Link 的时候改变。那么我们只需要记录两个信息 $si[x][0], si[x][1]$ 表示虚边节点数以及虚边节点数加上 Splay 中子树节点数，则 $si[x][1] = si[L[x]][1] + si[R[x]][1] + si[x][0] + 1$ 。然后 Access 和 Link 的时候顺便维护一下 $si[x][0]$ 就行了。注意 Link 的时候一定要让被更新节点成为根，否则无法更新信息（将引起大量信息改动）。

树合并，维护重心：维护出子树信息之后，根据重心的性质，新的树的重心在原来两棵树重心的路径上，将这条路径取出之后在 Splay 上二分，每次看左右两边子树大小是否 $\leq \frac{si}{2}$ ，如果满足则是重心。每次往节点多的那边走。

```
#define is_ro(p) ((p)!=son[fa[p]][0] && (p)!=son[fa[p]][1])
#define Son(p) ((p)==son[fa[p]][1])
inline void Pushup(int p) {si[p][1]=si[son[p][0]][1]+si[son[p][1]][1]+si[p][0]+1;}
inline void Rotate(int t){
    int p=fa[t],d=Son(t);son[p][d]=son[t][d^1];son[t][d^1]=p;
    Pushup(p);Pushup(t);if (!is_ro(p)) son[fa[p]][Son(p)]=t;
    if (son[p][d]) fa[son[p][d]]=p;fa[t]=fa[p];fa[p]=t;
}
inline void Addflip(int p) {swap(son[p][0],son[p][1]);flip[p]^=1;}
inline void Pushdown(int p) {
    if (flip[p]) flip[p]^=1,Addflip(son[p][0]),Addflip(son[p][1]);
}
inline void Splay(int p){
    static int top,stk[maxn+5];stk[top=1]=p;
    for (int i=p;!is_ro(i);i=fa[i]) stk[++top]=fa[i];
    while (top) Pushdown(stk[top--]);
    for (int pre=fa[p];!is_ro(p);Rotate(p),pre=fa[p])
        if (!is_ro(pre)) Rotate(Son(p)==Son(pre)?pre:p);
}
inline void Access(int p){
    for (int lst=0;p;Pushup(p),lst=p,p=fa[p]) // change virtual son
        Splay(p),si[p][0]+=si[son[p][1]][1]-si[lst][1],son[p][1]=lst;
}
inline void Makero(int x) {Access(x);Splay(x);Addflip(x);}

```

```cpp
// add virtual son
inline void Link(int x,int y) {
    Makero(x);Makero(y);fa[x]=y;si[y][0]+=si[x][1];Pushup(y);
}
void Split(int x,int y) {Makero(x);Access(y);Splay(x);}
int Askg(int x){
    int lim=si[x][1]>>1,suml=0,sumr=0,g=1e9;
    for (int ls=son[x][0],rs=son[x][1];x;ls=son[x][0],rs=son[x][1]){
        Pushdown(x);
        int nowl=suml+si[ls][1],nowr=sumr+si[rs][1];
        if (nowl<=lim && nowr<=lim) g=min(g,x);
        if (nowl<nowr) suml+=si[x][0]+si[ls][1]+1,x=rs;
        else sumr+=si[x][0]+si[rs][1]+1,x=ls;
    }
    Splay(g);
    return g;
}
/*use hint
initial: si[x][1]=1,si[x][0]=0
update center of gravity
fat[x] is the center of gravity in the x's tree
*/
int gx=getfa(x),gy=getfa(y);
Split(gx,gy);
fat[gx]=fat[gy]=fat[g]=g;
```

3.2.4 维护置换群

给出置换 p_i ，每次置换执行 $a_i = a_{p_i}$ 。环断链，LCT 维护每个置换群的头尾。支持交换 p_x, p_y 。

```cpp
#define is_ro(p) (son[fa[p]][0]!=(p) && son[fa[p]][1]!=(p))
#define Son(p) ((p)==son[fa[p]][1])
inline void Pushup(int p){
    MIN[p]=min({MIN[son[p][0]],MIN[son[p][1]],p});
    si[p]=si[son[p][0]]+1+si[son[p][1]];
}
void Rotate(int t){
    int p=fa[t],d=Son(t);son[p][d]=son[t][d^1];son[t][d^1]=p;
    if (!is_ro(p)) son[fa[p]][Son(p)]=t;Pushup(p);Pushup(t);
    if (son[p][d]) fa[son[p][d]]=p;fa[t]=fa[p];fa[p]=t;
}
inline void Addflip(int p) {swap(son[p][0],son[p][1]);flip[p]^=1;}
inline void Addtag(int p,pair<int,int> x) {L[p]=x.fr;R[p]=x.sc;tag[p]=x;}
void Pushdown(int p){
    if (flip[p]) Addflip(son[p][0]),Addflip(son[p][1]),flip[p]^=1;
    if (tag[p].fr) Addtag(son[p][0],tag[p]),Addtag(son[p][1],tag[p]),tag[p]=mp(0,0);
}
void Splay(int p){
    static int top,stk[maxn+5];stk[top=1]=p;
    for (int i=p;!is_ro(i);i=fa[i]) stk[++top]=fa[i];
    while (top) Pushdown(stk[top--]);
    for (int pre=fa[p];!is_ro(p);Rotate(p),pre=fa[p])
        if (!is_ro(pre)) Rotate(Son(p)==Son(pre)?pre:p);
```

```
}
void Access(int p) {
    for (int lst=0;p;lst=p,p=fa[p]) Splay(p),son[p][1]=lst,Pushup(p);
}
inline void Makero(int x) {Access(x);Splay(x);Addflip(x);}
inline void Link(int x,int y) {Makero(x);Makero(y);fa[x]=y;}
inline void Cut(int x,int y) {
    Makero(x);Access(y);Splay(x);son[x][1]=fa[y]=0;Pushup(x);
}
//Cover L R of the link between x and y
inline void Cover(int x,int y) {Makero(x);Access(y);Splay(y);Addtag(y,mp(x,y));}
inline pair<int,int> Ask(int x,int y) {
    Makero(x);Access(y);Splay(y);return mp(si[y],MIN[y]);
}
//Lx Rx is the head and tail of circle[x]
void SwapP(int x,int y){ // Swap Px Py
    if (x==y) return;
    Splay(x);Splay(y);
    if (L[x]!=L[y]){
        pair<int,int> X=Ask(L[x],R[x]),Y=Ask(L[y],R[y]);
        //Update Here
        if (x!=R[x] && y!=R[y]){
            Cut(x,p[x]);Cut(y,p[y]);
            Link(x,p[y]);Link(y,p[x]);
            Link(R[y],L[y]);Cover(L[x],R[x]);
        } else if (x==R[x] && y==R[y]){Link(R[x],L[y]); Cover(L[x],R[y]);}
        else if (x==R[x]){Cut(y,p[y]);Link(y,L[x]);Link(R[x],p[y]);Cover(L[y],R[y]);}
        else {Cut(x,p[x]);Link(x,L[y]);Link(R[y],p[x]);Cover(L[x],R[x]);}
        //Update Here
    } else {
        if (Ask(L[x],x).fr>Ask(L[y],y).fr) swap(x,y);
        pair<int,int> X=Ask(L[x],R[x]);
        //Update Here
        int lx=L[x],rx=R[x];
        if (y!=rx){
            Cut(x,p[x]);Cut(y,p[y]);Link(x,p[y]);
            Cover(lx,rx);X=Ask(lx,rx);
            //Update Here
            Cover(p[x],y);X=Ask(p[x],y);
            //Update Here
        } else {
            Cut(x,p[x]);
            Cover(lx,x);X=Ask(lx,x);
            //Update Here
            Cover(p[x],rx);X=Ask(p[x],rx);
            //Update Here
        }
    }
    swap(p[x],p[y]);
}
```

3.3 Splay 维护区间

```
struct SplayTree{
    int pl,ro,son[maxt+5][2],fa[maxt+5],si[maxt+5];bool flip[maxt+5];
    int cmp(int p,int &k){
        if (k<=si[son[p][0]]) return 0;
        if (k==si[son[p][0]]+1) return -1;
        k-=si[son[p][0]]+1;return 1;
    }
    void Pushup(int p) {si[p]=si[son[p][0]]+1+si[son[p][1]];}
    int newnode() {pl++;son[pl][0]=son[pl][1]=fa[pl]=flip[pl]=0;si[pl]=1;return pl;}
    int Build(int L,int R,int pre=0){
        if (L>R) return 0;
        int mid=L+(R-L>>1),now=newnode();
        son[now][0]=BuildA(L,mid-1,now);son[now][1]=BuildA(mid+1,R,now);
        fa[now]=pre;Pushup(now);return now;
    }
    void Clear(int n) {pl=0;ro=Build(0,n+1);} // Build 0~n+1
    void Rotate(int &p,int d){
        int t=son[p][d];son[p][d]=son[t][d^1];son[t][d^1]=p;Pushup(p);Pushup(t);
        if (son[p][d]) fa[son[p][d]]=p;fa[t]=fa[p];fa[p]=t;p=t;
    }
    void Addflip(int p) {if (!p) return;swap(son[p][0],son[p][1]);flip[p]^=1;}
    void Pushdown(int p) {if(flip[p])Addflip(son[p][0]),Addflip(son[p][1]),flip[p]^=1;}
    void Splay(int &p,int k){
        int d=cmp(p,k);Pushdown(p);if (d<0) return;
        int &t=son[p][d],f=cmp(t,k);Pushdown(t);
        if (~f) Splay(son[t][f],k),d==f?Rotate(p,d):Rotate(t,f);Rotate(p,d);
    }
    int Askkth(int p,int k){
        int d=cmp(p,k);Pushdown(p);
        return ~d?Askkth(son[p][d],k):p;
    }
    int Askrk(int p){
        static int top,stk[maxt+5];
        top=0;for (int i=p,f=fa[i];i!=ro;i=f,f=fa[i]) stk[++top]=fa[i];
        while (top) Pushdown(stk[top]),top--;
        int rk=si[son[p][0]]+1;
        for (int f=fa[p];p!=ro;p=f,f=fa[p])
            if (p==son[f][1]) rk+=si[son[f][0]]+1;
        Splay(ro,rk);return rk;
    }
    int getseg(int L,int R){
        L--;R++;Splay(ro,L);cmp(ro,R);Splay(son[ro][1],R);
        return son[son[ro][1]][0];
    }
    int Insert(int x){
        int now=newnode();getseg(x,x-1);
        son[son[ro][1]][0]=now;fa[now]=son[ro][1];
        Pushup(son[ro][1]);Pushup(ro);return now;
    }
}tr;
```

```
/*use hint
build [0,n+1]: tr.Clear(n);
segment [L,R]: L++;R++;tr.getseg(L,R);
*/
```

3.4 KDT

各种估价函数看 Claris 板子。

```
struct Point{
    int x[2],w;
    bool operator < (const Point &a) {return x[D]<a.x[D];}
}a[maxn+5];
int que[maxn+5];
int ro,son[maxn+5][2],MN[maxn+5][2],MX[maxn+5][2],val[maxn+5];
inline bool cmp(const int &i,const int &j) {return a[i]<a[j];}
void Pushup(int p){
    for (int i=0;i<2;i++) MN[p][i]=MX[p][i]=a[p].x[i];
    val[p]=max(a[p].w,max(val[son[p][0]],val[son[p][1]]));
    sum[p]=sum[son[p][0]]+a[p].w+sum[son[p][1]];
    for (int j=0;j<2;j++)
        if (son[p][j])
            for (int i=0;i<2;i++)
                MN[p][i]=min(MN[p][i],MN[son[p][j]][i]),
                MX[p][i]=max(MX[p][i],MX[son[p][j]][i]);
}
int Build(int L,int R,int d=0){
    int m=L+(R-L>>1);
    D=d;nth_element(que+L,que+m,que+R+1,cmp);
    int p=que[m];son[p][0]=son[p][1]=0;
    if (L<m) son[p][0]=Build(L,m-1,d^1);
    if (m<R) son[p][1]=Build(m+1,R,d^1);
    Pushup(p);return p;
}
#define absi(x) ((x)<0?-(x):(x))
#define TOP(p,x,y) (max(absi((x)-MX[p][0]),absi((x)-MN[p][0]))
                        +max(absi((y)-MX[p][1]),absi((y)-MN[p][1])))
int VAL(int p,int x,int y){
    if (!p) return 0;
    return min(TOP(p,x,y),val[p]);
}
void Ask(int p,int x,int y){ // optimize
    if (val[p]<=ans || TOP(p,x,y)<=ans) return;
    ans=max(ans,min(absi(x-a[p].x[0])+absi(y-a[p].x[1]),a[p].w));
    int d=VAL(son[p][1],x,y)>VAL(son[p][0],x,y);
    if (son[p][d]) Ask(son[p][d],x,y);if (son[p][d^1]) Ask(son[p][d^1],x,y);
}
int Ask(int p,int A,int B,int C,int D){ // matrix, x in [A,B], y in [C,D]
    if (!p || B<MN[p][0] || MX[p][0]<A || D<MN[p][1] || MX[p][1]<C) return 0;
    if (A<=MN[p][0] && MX[p][0]<=B && C<=MN[p][1] && MX[p][1]<=D) return sum[p];
    int val=(A<=a[p].x[0] && a[p].x[0]<=B && C<=a[p].x[1] && a[p].x[1]<=D?a[p].w:0);
    return Ask(son[p][0],A,B,C,D)+val+Ask(son[p][1],A,B,C,D);
}
```

3.5 猫树

定义 $A_{i,j}=\prod_{k=\lfloor \frac{i}{2^j} \rfloor 2^j}^{i} a_k, B_{i,j}=\prod_{k=i}^{\lceil \frac{i}{2^j} \rceil 2^j-1} a_k$ ，可以 $O(n\log_2 n)$ 处理。

然后询问区间 $[L,R]$ ，如果 $L=R$ 那么就是 a_L ，否则我们需要找到一个 k 使得 $[L,R]$ 内只存在一个 2^k 的倍数，这样的话就可以直接 $B_{L,k}\cdot A_{R,k}$ 得出结果。

找规律可知 $highbit(L\ xor\ R)$ 就是一个合法的 k 。

```
#define L(i,k) ((i)>>(k)<<(k))
#define R(i,k) (((i)+(1<<(k)))>>(k)<<(k))-1)

// maxs=(1<<LOG)
for (int j=1;j<LOG;j++) lg[1<<j]=j;
for (int i=1;i<maxs;i++) if (!lg[i]) lg[i]=lg[i-1];
for (int i=0;i<n;i++) A[0][i]=B[0][i]=a[i]; // a is the initial array
for (int j=1;(1<<j)<=n;j++){
    A[j][0]=0;B[j][n]=0; // initial value
    // calc(a,A) and calc(A,a) may be different
    for (int i=0;i<=n-1;i++) A[j][i]=(L(i,j)==i?a[i]:calc(a[i],A[j][i-1]));
    for (int i=n-1;i>=0;i--) B[j][i]=(R(i,j)==i?a[i]:calc(B[j][i+1],a[i]));
}
```

3.6 ODT

用 set 维护相同信息的线段。若数据含有 Assign 并且随机，则块数是 $O(\log\log n)$ 的。

区间操作时暴力遍历 $[L,R]$ 中所有线段。

```
struct node{
    int L,R,c;
    node(int l=0,int r=0,int w=0) {L=l;R=r;c=w;}
    bool operator < (const node &a) const {return L<a.L;}
};
typedef set<node>::iterator SI;
set<node> S;SI it,l,r;

inline SI Node(int x) {return prev(S.upper_bound(node(x,1e9)));}
SI Split(int x){ // split [L,R] to [L,x-1] and [x,R]
    if (x>n) return S.end();
    it=Node(x);if (it->L==x) return it;
    int L=it->L,R=it->R,c=it->c;
    S.erase(it);S.insert(node(L,x-1,c));
    return S.insert(node(x,R,c)).first;
}
void Assign(int L,int R,int c){ // merge [L,R] and change value to c
    r=Split(R+1);l=Split(L);
    for (it=l;it!=S.end() && it->R<=R;it=S.erase(it))
        Insert(ro,it->L,it->R,tag[it->c]-tag[c]);
    S.insert(node(L,R,c));
}
```

3.7 随机堆

```
int ro[maxn+5],son[maxn+5][2],fa[maxn+5],tag[maxn+5];
int fat[maxn+5];

int getfa(int x) {return x==fat[x]?x:fat[x]=getfa(fat[x]);}
inline void Addtag(int p,int k) {if (!p) return;a[p]+=k;tag[p]+=k;}
inline void Pushdown(int p) {
    if (tag[p]) Addtag(son[p][0],tag[p]),Addtag(son[p][1],tag[p]),tag[p]=0;
}
int Merge(int A,int B){
    if (!A || !B) return A+B;
    if (a[A]<a[B]) swap(A,B);Pushdown(A);
    int d=rand()&1;son[A][d]=Merge(son[A][d],B);
    if (son[A][0]) fa[son[A][0]]=A;if (son[A][1]) fa[son[A][1]]=A;
    return A;
}
#define Son(x) ((x)==son[fa[x]][1])
inline void Swap(int x,int f){
    int d=Son(x);swap(son[x],son[f]);son[x][d]=f;
    if (son[f][0]) fa[son[f][0]]=f;if (son[f][1]) fa[son[f][1]]=f;
    if (son[x][d^1]) fa[son[x][d^1]]=x;
    if (fa[f]) son[fa[f]][Son(f)]=x; else ro[getfa(x)]=x;
    fa[x]=fa[f];fa[f]=x;
}
inline void Add(int x,int v){
    static int top=0,stk[maxn+5];
    for (int i=x;i;i=fa[i]) stk[++top]=i;while (top) Pushdown(stk[top--]);a[x]+=v;
    for (int f=fa[x];f&&a[x]>a[f];f=fa[x]) Swap(x,f);
    int l,r;
    for (l=son[x][0],r=son[x][1];l&&a[x]<a[l]||r&&a[x]<a[r];l=son[x][0],r=son[x][1]){
        if (!l||r&&a[r]>a[l]) l=r;
        Pushdown(l);Swap(l,x);
    }
}
```

4 卷积与多项式

4.1 FFT

一个神奇的板子，但是要注意这个板子 DFT 之后并没有蝴蝶变换。
因此涉及到对点值进行运算时需要注意做蝴蝶变换。

```
inline CN operator ! (const CN &a) {return mp(a.fr,-a.sc);}
inline CN operator + (const CN &a,const CN &b) {return mp(a.fr+b.fr,a.sc+b.sc);}
inline CN operator - (const CN &a,const CN &b) {return mp(a.fr-b.fr,a.sc-b.sc);}
inline CN operator * (const CN &a,const CN &b) {
    return mp(a.fr*b.fr-a.sc*b.sc,a.fr*b.sc+a.sc*b.fr);
}
__attribute__((constructor)) void FFTPre(){
    for (int i=0;i<(maxt>>1);i++)
        wn[i+(maxt>>1)]=mp(cos(2*pi*i/maxt),sin(2*pi*i/maxt));
    for (int i=(maxt>>1)-1;i;i--) wn[i]=wn[i<<1];
}
void FFT(CN *a,int n,int f){
    CN x,y;
    if (f>0){
        for (int k=n>>1;k;k>>=1)
            for (int i=0;i<n;i+=k<<1)
                for (int j=0;j<k;j++){
                    x=a[i+j];y=a[i+j+k];
                    a[i+j+k]=(x-y)*wn[k+j];
                    a[i+j]=x+y;
                }
    } else {
        for (int k=1;k<n;k<<=1)
            for (int i=0;i<n;i+=k<<1)
                for (int j=0;j<k;j++){
                    x=a[i+j];y=a[i+j+k]*!wn[k+j];
                    a[i+j+k]=x-y;
                    a[i+j]=x+y;
                }
        for (int i=0;i<n;i++) a[i].fr/=n,a[i].sc/=n;
    }
}
```

4.2 分治 FFT

4.2.1 非自身卷积

$f_i=\sum_{k=0}^{i-1} f_k g_{i-k}$ ，f_0 已知，给出 $g_{1..n}$ 。（$k>i$ 同理）
cdq 分治，先处理 $[L,mid]$ 的 f ，然后用 $[L,mid]$ 的 f 更新 $[mid+1,R]$ 的 f 。
$p_i=f_{i+L}(0\le i\le mid-L)$ ，$q_i=g_i(1\le i\le R-L)$ 。
$f_i(mid+1\le i\le R)$ 加上 $[x^{i-L}](p*q)(x)$ 。
卡常技巧：虽然我们要做 $mid-L$ 和 $R-L$ 的卷积，但由于我们只需要用到 $(p*q)(x)$ 从 $mid+1-L$ 开始的部分，因此我们可以将 FFT 长度设成 $R-L$ ，FFT 会循环溢出，但是只会溢出 $mid-L$ ，不会影响到 $mid+1-L$ 后面的部分。式子简单时可以用多项式求逆求解，复杂度更低。

4.2.2 自身卷积

$f_i=\sum_{k=1}^{i-1} f_k f_{i-k}(i\geq 2)$ ，f_0,f_1 已知。

还是考虑 cdq 分治，但是会出现一个问题，在统计 $[L,mid]$ 对 $[mid+1,R]$ 的贡献时，f_{R-L} 可能是未知的 $(R-L>mid)$ 。先只统计 $[L,mid]$ 中的贡献，即 $[L,mid]$ 卷 $[L,mid]$ ，剩下的贡献在后面补上。

当 $R-L<L$ 时，$[1,R-L]$ 的信息和 $[L,mid]$ 没有重叠，并且 $[1,L-1]$ 的 f 已经求出，因此将 $[1,R-L]$ 和 $[L,mid]$ 卷积，加到 $[mid+1,R]$ 上，从而补上没算的权值。

注意 $k\leq R-L,i-k\leq R-L$ 在 $[L,mid]$ 卷 $[L,mid]$ 均没有出现，因此需要算两次。下面是例子：

$$f_0=1,f_1=2,f_i=(i-1)f_{i-1}+\sum_{k=2}^{i-2}(k-1)f_k f_{i-k}$$

```
void Solve(int L,int R){ // f*g
    if (L==R) {/*Update here*/ return;}
    int mid=L+(R-L>>1);
    Solve(L,mid);
    int t;for (t=1;t<=R-L;t<<=1);
    for (int i=L;i<=mid;i++) A[i-L]=f[i];for (int i=mid-L+1;i<t;i++) A[i]=0;
    B[0]=0;for (int i=1;i<=R-L;i++) B[i]=g[i];for (int i=R-L+1;i<t;i++) B[i]=0;
    NTT(A,t,1);NTT(B,t,1);
    for (int i=0;i<t;i++) A[i]=MUL(A[i],B[i]);
    NTT(A,t,-1);for (int i=mid+1;i<=R;i++) f[i]=ADD(f[i],A[i-L]);
    Solve(mid+1,R);
}
void Solve(int L,int R){ // f*f
    if (L==R) {f[L]=ADD(f[L],MUL(L-1,f[L-1]));return;}
    int mid=L+(R-L>>1); Solve(L,mid);
    if (R-L<L){
        int t;for (t=1;t<=R-L;t<<=1);
        for (int i=L;i<=mid;i++) A[i-L]=f[i];
        for (int i=mid-L+1;i<t;i++) A[i]=0;
        for (int i=2;i<=R-L;i++) B[i]=MUL(i-1,f[i]);
        B[0]=B[1]=0;for (int i=R-L+1;i<t;i++) B[i]=0;
        NTT(A,t,1);NTT(B,t,1);
        for (int i=0;i<t;i++) A[i]=MUL(A[i],B[i]);NTT(A,t,-1);
        for (int i=mid+1;i<=R;i++) f[i]=ADD(f[i],A[i-L]);
        for (int i=L;i<=mid;i++) A[i-L]=MUL(i-1,f[i]);
        for (int i=mid-L+1;i<t;i++) A[i]=0;
        for (int i=2;i<=R-L;i++) B[i]=f[i];
        B[0]=B[1]=0;for (int i=R-L+1;i<t;i++) B[i]=0;
        NTT(A,t,1);NTT(B,t,1);
        for (int i=0;i<t;i++) A[i]=MUL(A[i],B[i]);NTT(A,t,-1);
        for (int i=mid+1;i<=R;i++) f[i]=ADD(f[i],A[i-L]);
    } else {
        int t;for (t=1;t<=(mid-L<<1);t<<=1);
        for (int i=L;i<=mid;i++) A[i-L]=f[i],B[i-L]=MUL(i-1,f[i]);
        for (int i=mid-L+1;i<t;i++) A[i]=B[i]=0;
        NTT(A,t,1);NTT(B,t,1);
        for (int i=0;i<t;i++) A[i]=MUL(A[i],B[i]);NTT(A,t,-1);
        for (int i=mid+1;i<=R;i++) if (i>=(L<<1)) f[i]=ADD(f[i],A[i-(L<<1)]);
    }
    Solve(mid+1,R);
}
```

4.3 Bluestein

循环卷积：$f_i g_j \to h_{(i+j) \bmod n}$
任意长度 DFT：

$$F_k = f(w_n^k) = \sum_{i=0}^{n-1} f_i w_n^{ik}$$

$$ik = \binom{i+k}{2} - \binom{i}{2} - \binom{k}{2}$$

$$F_k = w_n^{-\binom{k}{2}} \sum_{i=0}^{n-1} f_i w_n^{-\binom{i}{2}} \cdot w_n^{\binom{i+k}{2}}$$

令 $g_i = f_{n-i} w_n^{-\binom{n-i}{2}}, h_i = w_n^{\binom{i}{2}}$ ，则：

$$F_k = w_n^{-\binom{k}{2}} [x^{n-k}](g * h)(x)$$

同理，逆变换时代入 w_n^{-k} ，最后除以 n 。

4.4 MTT

$P(x) = A_0(x) + iA_1(x), Q(x) = A_0(x) - iA_1(x)$
点值满足 $Q_i = conj(P_{2t-i})$ ，因此只需要算 $P(x)$ 的点值就可以得到 $Q(x)$ 的点值，从而得到：
$A_0(x) = \frac{P(x)+Q(x)}{2}, A_1 = \frac{P(x)-Q(x)}{2i}$
然后通过点值 $P = A_0B_0 + iA_0B_1, Q = A_1B_0 + iA_1B_1$ ，逆变换得到 $A_0B_0, A_0B_1, A_1B_0, A_1B_1$ 。
由于涉及到点值共轭，使用上面的 FFT 板子时需要注意蝴蝶变换。

```
void MTT(int *c,int *a,int n,int *b,int m){
    static int lg=15,BA=(1<<lg)-1;
    static CN A[maxt+5],P[maxt+5],Q[maxt+5];
    int t;for (t=1;t<=n+m;t<<=1);
    for (int i=0;i<=n;i++) P[i]=mp(a[i]&BA,a[i]>>lg);
    for (int i=n+1;i<t;i++) P[i]=mp(0,0);
    for (int i=0;i<=m;i++) Q[i]=mp(b[i]&BA,b[i]>>lg);
    for (int i=m+1;i<t;i++) Q[i]=mp(0,0);
    FFT(P,t,1);FFT(Q,t,1);
    for (int i=0;i<t;i++) A[i]=P[i];
    P[0]=mp((A[0].fr+A[0].fr)/2,(A[0].sc-A[0].sc)/2)*Q[0];
    Q[0]=mp((A[0].sc+A[0].sc)/2,(A[0].fr-A[0].fr)/2)*Q[0];
    for (int k=1,i=1,j;k<t;k<<=1)
        for (j=i^k-1;i<(k<<1);i++,j=i^k-1)
            P[i]=mp((A[i].fr+A[j].fr)/2,(A[i].sc-A[j].sc)/2)*Q[i],
            Q[i]=mp((A[i].sc+A[j].sc)/2,(A[j].fr-A[i].fr)/2)*Q[i];
    FFT(P,t,-1);FFT(Q,t,-1);
    for (int i=0;i<=n+m;i++){
        LL A0B0=LL(P[i].fr+0.5)%MOD,A0B1=LL(P[i].sc+0.5)%MOD;
        LL A1B0=LL(Q[i].fr+0.5)%MOD,A1B1=LL(Q[i].sc+0.5)%MOD;
        c[i]=(A0B0+(A0B1+A1B0<<lg)%MOD+(A1B1<<lg+lg)%MOD)%MOD;
    }
}
```

4.5 巨大 NTT 模数

取模数为 $29 \times 2^{57} + 1$ ，原根为 3 。可以解决大答案不取模的 FFT 精度问题。

4.6 多项式全家桶

```
inline int ADD(int x,int y) {return x+y>=MOD?x+y-MOD:x+y;}
inline int MUL(int x,int y) {return (LL)x*y%MOD;}
int Pow(int w,int b) {int s;for (s=1;b;b>>=1,w=MUL(w,w)) if (b&1) s=MUL(s,w);return s;}
__attribute__((constructor)) void NTTPre(){
    int x=Pow(3,(MOD-1)/maxt);
    wn[maxt>>1]=1;
    for (int i=(maxt>>1)+1;i<maxt;i++) wn[i]=MUL(wn[i-1],x);
    for (int i=(maxt>>1)-1;i;i--) wn[i]=wn[i<<1];
}
void NTT(int *a,int n,int f){
    if (f>0){
        for (int k=n>>1;k;k>>=1)
            for (int i=0;i<n;i+=k<<1)
                for (int j=0;j<k;j++){
                    int x=a[i+j],y=a[i+j+k];
                    a[i+j+k]=MUL(x+MOD-y,wn[k+j]);
                    a[i+j]=ADD(x,y);
                }
    } else {
        for (int k=1;k<n;k<<=1)
            for (int i=0;i<n;i+=k<<1)
                for (int j=0;j<k;j++){
                    int x=a[i+j],y=MUL(a[i+j+k],wn[k+j]);
                    a[i+j+k]=ADD(x,MOD-y);
                    a[i+j]=ADD(x,y);
                }
        for (int i=0,INV=MOD-(MOD-1)/n;i<n;i++) a[i]=MUL(a[i],INV);
        reverse(a+1,a+n);
    }
}
void Make(int n){
    INV[0]=INV[1]=1;for (int i=2;i<=n;i++) INV[i]=MUL(MOD-MOD/i,INV[MOD%i]);
    fac[0]=1;for (int i=1;i<=n;i++) fac[i]=MUL(fac[i-1],i),INV[i]=MUL(INV[i-1],INV[i]);
}
void Inte(int *F,int *a,int n,int K){ // F=integral(a)
    for (int i=n+K;i>=K;i--) F[i]=MUL(a[i-K],MUL(INV[i],fac[i-K]));
    for (int i=0;i<K;i++) F[i]=0;
}
void Deri(int *F,int *a,int n,int K){ // F=a'
    for (int i=0;i<=n-K;i++) F[i]=MUL(a[i+K],MUL(fac[i+K],INV[i]));
    for (int i=n-K+1;i<=n;i++) F[i]=0;
}
void Inv(int *F,int *a,int n){ // F=1/a
    if (n==1) {F[0]=Pow(a[0],MOD-2);return;}
    Inv(F,a,n>>1);
    for (int i=0;i<n;i++) temA[i]=a[i],temA[i+n]=F[i+n]=0;
```

```
    NTT(temA,n<<1,1);NTT(F,n<<1,1);
    for (int i=0;i<(n<<1);i++) temA[i]=MUL(F[i],2+MOD-MUL(temA[i],F[i]));
    NTT(temA,n<<1,-1);for (int i=0;i<n;i++) F[i]=temA[i],F[i+n]=0;
}
void Ln(int *F,int *a,int n){ // F=ln(a)
    Inv(temB,a,n);
    Deri(temA,a,n-1,1);for (int i=0;i<n;i++) temA[i+n]=0;
    NTT(temA,n<<1,1);NTT(temB,n<<1,1);
    for (int i=0;i<(n<<1);i++) temA[i]=MUL(temA[i],temB[i]);
    NTT(temA,n<<1,-1);Inte(F,temA,n-2,1);
}
void Exp(int *F,int *a,int n){ // F=exp(a)
    if (n==1) {F[0]=1;return;} // only when a[0]=0
    Exp(F,a,n>>1);Ln(temA,F,n);
    for (int i=0;i<n;i++) temA[i]=ADD(a[i],MOD-temA[i]),temA[i+n]=F[i+n]=0;
    temA[0]=ADD(temA[0],1);
    NTT(temA,n<<1,1);NTT(F,n<<1,1);
    for (int i=0;i<(n<<1);i++) temA[i]=MUL(temA[i],F[i]);
    NTT(temA,n<<1,-1);for (int i=0;i<n;i++) F[i]=temA[i],F[i+n]=0;
}
void Sqrt(int *F,int *a,int n){ // F=sqrt(a)
    if (n==1) {F[0]=1;return;} // only when a[0]=1
    Sqrt(F,a,n>>1);Inv(temB,F,n);
    for (int i=0;i<n;i++) temA[i]=a[i],temA[i+n]=0;
    NTT(temA,n<<1,1);NTT(temB,n<<1,1);
    for (int i=0;i<(n<<1);i++) temA[i]=MUL(temA[i],temB[i]);
    NTT(temA,n<<1,-1);for (int i=0;i<n;i++) F[i]=MUL(F[i]+temA[i],INV2),F[i+n]=0;
}
inline PN operator + (const PN &a,const PN &b){
    static PN c;c.resize(max(a.size(),b.size()));
    for (int i=0;i<c.size();i++) c[i]=ADD(i<a.size()?a[i]:0,i<b.size()?b[i]:0);
    return c;
}
inline PN operator - (const PN &a,const PN &b){
    static PN c;c.resize(max(a.size(),b.size()));
    for (int i=0;i<c.size();i++) c[i]=ADD(i<a.size()?a[i]:0,i<b.size()?MOD-b[i]:0);
    return c;
}
PN operator * (const PN &a,const PN &b){
    static PN c;
    int n=a.size(),m=b.size(),t;
    for (t=1;t<n+m-1;t<<=1);
    for (int i=0;i<n;i++) temA[i]=a[i];for (int i=n;i<t;i++) temA[i]=0;
    for (int i=0;i<m;i++) temB[i]=b[i];for (int i=m;i<t;i++) temB[i]=0;
    NTT(temA,t,1);NTT(temB,t,1);
    for (int i=0;i<t;i++) temA[i]=MUL(temA[i],temB[i]);
    NTT(temA,t,-1);
    c.resize(n+m-1);for (int i=0;i<n+m-1;i++) c[i]=temA[i];
    return c;
}
inline PN operator / (const PN &a,const PN &b){
    static int C[maxt+5];static PN c;
```

```
    int n=a.size()-1,m=b.size()-1;
    if (n<m) return {0};
    int t;for (t=1;t<n-m+1;t<<=1);
    for (int i=0;i<n-m+1;i++) temB[i]=m-i<0?0:b[m-i];
    for (int i=n-m+1;i<t;i++) temB[i]=0;
    Inv(C,temB,t);
    for (int i=0;i<n-m+1;i++) temA[i]=a[n-i];
    for (int i=n-m+1;i<(t<<1);i++) temA[i]=0;
    NTT(temA,t<<1,1);NTT(C,t<<1,1);
    for (int i=0;i<(t<<1);i++) C[i]=MUL(temA[i],C[i]);
    NTT(C,t<<1,-1);
    c.resize(n-m+1);for (int i=0;i<n-m+1;i++) c[i]=C[n-m-i];
    return c;
}
inline PN operator % (const PN &a,const PN &b) {
    static PN c;c=a-a/b*b;c.resize(b.size()-1);return c;
}
void Build(int L,int R,int p=1){
    if (L==R) {P[p]={(MOD-X[L])%MOD,1};return;}
    int mid=L+(R-L>>1);
    Build(L,mid,p<<1);Build(mid+1,R,p<<1|1);
    P[p]=P[p<<1]*P[p<<1|1];
}
void getv(int L,int R,int p=1){
    if (L==R) {v[L]=F[0];return;}
    int mid=L+(R-L>>1);PN tem=F;
    F=tem%P[p<<1];getv(L,mid,p<<1);
    F=tem%P[p<<1|1];getv(mid+1,R,p<<1|1);
}
PN Lag(int L,int R,int p=1){
    if (L==R) return {v[L]};
    int mid=L+(R-L>>1);
    return P[p<<1|1]*Lag(L,mid,p<<1)+P[p<<1]*Lag(mid+1,R,p<<1|1);
}
```

```
use hint
f(x) -> f(ai):
    input F[0..n-1],X[1..m]
    Build(1,m);getv(1,m);
f(ai) -> f(x):
    input X[1..n],Y[1..n]
    Build(1,n);
    for (int i=0;i<=n;i++) DP[i]=P[1][i];
    Deri(DP,DP,n,1);
    F.resize(n);for (int i=0;i<n;i++) F[i]=DP[i];
    getv(1,n);
    for (int i=1;i<=n;i++) v[i]=MUL(Y[i],Pow(v[i],MOD-2));
    F=Lag(1,n);
```

4.7 常系数线性齐次递推

给出 $f_{0..K-1}, a_{1..K}$ ，$f_n = \sum_{i=1}^{K} a_i f_{n-i}$ ，求 f_n 。

```
int Linear(int *f,int *a,int K,int n){
    if (n<K) return f[n];
    static PN p,w,s;p.resize(K+1);
    for (int i=0;i<K;i++) p[i]=(MOD-a[K-i])%MOD;p[K]=1;
    w={0,1};w=w%p;
    for (s={1};n;n>>=1,w=w*w%p) if (n&1) s=s*w%p;
    int ans=0;
    for (int i=0,si=s.size();i<K && i<si;i++) ans=ADD(ans,MUL(s[i],f[i]));
    return ans;
}
```

4.8 多项式除法求系数

$G(x) = \frac{U(x)}{D(x)}$ ，$U(x), D(x)$ 是 m 次多项式，求第 n 项系数（n 很大）。

$$
\begin{aligned}
& U(x) = G(x)D(x) \\
\Rightarrow & \forall n > m, \sum_{i+j=n} g_i d_j = u_n = 0 \\
\Leftrightarrow & \forall n > m, \sum_{i=0}^{m} d_i g_{n-i} = 0 \\
\Leftrightarrow & \forall n > m, d_0 g_n + \sum_{i=1}^{m} d_i g_{n-i} = 0 \\
\Leftrightarrow & \forall n > m, g_n = \sum_{i=1}^{m} \frac{-d_i}{d_0} g_{n-i}
\end{aligned}
$$

先用多项式求逆求出 $[0, m]$ 的系数，然后常系数线性齐次递推。

```
int Asknth(int *U,int *D,int m,int n){
    static int A[maxt+5];
    int t;for (t=1;t<=(m<<1);t<<=1);
    Inv(A,D,t>>1);NTT(A,t,1);NTT(U,t,1);
    for (int i=0;i<t;i++) A[i]=MUL(A[i],U[i]);
    NTT(A,t,-1);
    if (n<=m) return A[n];
    for (int i=0;i<m;i++) A[i]=A[i+1];
    for (int i=1,INV=Pow(D[0],MOD-2);i<=m;i++) D[i]=MUL(MOD-D[i],INV);
    return Linear(A,D,m,n-1);
}
```

4.9 多项式复杂度优化

1. 树上背包，可以利用链分治优化，轻链分治 NTT，重链分治矩乘 NTT。
2. 给出 $A_i(x), B_i(x)$ ，求 $\sum_{i=1}^{n} val_i \prod_{j=1}^{i-1} A_j(x) \prod_{j=i+1}^{n} B_j(x)$ ，可以利用分治矩乘 NTT:

$$
\begin{bmatrix} F_{i-1,0}(x) & F_{i-1,1}(x) \end{bmatrix} \begin{bmatrix} A_i(x) & val_i \\ 0 & B_i(x) \end{bmatrix} = \begin{bmatrix} F_{i,0}(x) & F_{i,1}(x) \end{bmatrix}
$$

3. 给出 $\{a_n\}$ ，求 $F(x) = \sum_{i=0}^{t} (\sum_{j=1}^{n} a_j^i) x^i$ ，即每个 k 次幂之和:

$$
\begin{aligned}
f_i(x) = \sum_{k=0}^{t} a_i^k x^k = \frac{1}{1 - a_i x}, F(x) = \sum_{i=1}^{n} f_i(x) \\
g_i(x) = -[\ln(1 - a_i x)]' = \frac{a_i}{1 - a_i x}, f_i(x) = x g_i(x) + 1 \\
F(x) = n - x \sum_{i=1}^{n} [\ln(1 - a_i x)]' = n - x\{\ln[\prod_{i=1}^{n} (1 - a_i x)]\}'
\end{aligned}
$$

因此，$\sum_{i=1}^{n} P(a_i x) = \sum_{i=1}^{n} \sum_j p_j a_i^j x^j = \sum_j p_j (\sum_{i=1}^{n} a_i^j) x^j$ ，可以求出 $F(x)$ 后得出。

同理，$\prod_{i=1}^{n} P(a_i x) = \exp[\sum_{i=1}^{n} (\ln P)(a_i x)]$ 也可以得出。

4.10 一些凑多项式的技巧

1. $g_j = (j+1)f_{j+1}$: $G(x) = F'(x)$

2. $F_i(x) = F'_{i-1}(x) + A(x)F_{i-1}(x)$
   令 $G'(x) = A(x), G(x) = \int A(x)dx, P_i(x) = e^{G(x)}F_i(x)$
   $P'_i(x) = e^{G(x)}[F'_i(x) + A(x)F_i(x)] = e^{G(x)}F_{i+1}(x) = P_{i+1}(x)$
   $P_k(x) = P_0^{(k)}(x)$ ， $F_k(x) = \dfrac{P_k(x)}{e^{G(x)}}$

3. $f_i g_j w^{ij} \to h_{i+j}$: $\dfrac{f_i}{w^{\binom{i}{2}}}\dfrac{g_j}{w^{\binom{j}{2}}} \to \dfrac{h_{i+j}}{w^{\binom{i+j}{2}}}$

4. $f_{i,j} = \sum_{k=0}^{j} a^k f_{i-1,k} \cdot g_{j-k}$
   $F_n(x) = F_{n-1}(ax)G(x) = F_0(a^n x)\prod_{i=0}^{n-1} G(a^i x)$
   如果 $F_0(x) = 1$（常数同理），则 $F_n(x) = \prod_{i=0}^{n-1} G(a^i x)$ ，可以分治：
   (1) n 偶数：$F_n(x) = F_{\frac{n}{2}}(x)F_{\frac{n}{2}}(a^{\frac{n}{2}}x)$
   (2) n 奇数：$F_n(x) = F_{\frac{n}{2}}(x)F_{\frac{n}{2}}(a^{\frac{n}{2}}x)G(a^{n-1}x)$

4.11 伯努利数

$$B_i = i! \cdot [x^i](\sum_{i=0}^{+\infty} \frac{x^i}{(i+1)!})^{-1}$$

$$\sum_{i=0}^{n-1} i^K = \frac{1}{K+1}\sum_{i=0}^{K}\binom{K+1}{i}B_i n^{K+1-i}$$

4.12 单位根反演

可用于求多项式（可推广至矩阵等形式）$f(x)$ 所有 k 倍数位置上的系数之和。
模意义下单位根用原根代替，即 $g_n = g^{\frac{MOD-1}{n}}, g_n^n = g^{MOD-1} = 1$ 。

$$\frac{1}{k}\sum_{i=0}^{k-1}\omega_k^{ni} = [k \mid n]$$

$$f(x) = \sum_i a_i x^i$$

$$\sum_{k|i} a_i = \sum_i a_i[k \mid i] = \frac{1}{k}\sum_i a_i \sum_{j=0}^{k-1}\omega_k^{ij} = \frac{1}{k}\sum_{j=0}^{k-1}\sum_i a_i(\omega_k^j)^i = \frac{1}{k}\sum_{j=0}^{k-1} f(\omega_k^j)$$

4.12.1 扩展形式

$$\sum_{i \bmod k = r} a_i = \sum_{k|i-r} a_i = \frac{1}{k}\sum_{j=0}^{k-1}\frac{f(\omega_k^j)}{\omega_k^{jr}}$$

原理：$f(x)$ 向左平移 r 个，即 $\frac{f(x)}{x^r}$ 。

4.13 拉格朗日插值

用 n 个点 (x_i, y_i) 求出 $n-1$ 次多项式 $f(x)=\sum_{i=0}^{n-1} a_i x^i$ 。

$$f(x)=\sum_{i=1}^{n} y_i \prod_{j\neq i} \frac{x-x_j}{x_i-x_j}$$

令 $A(k)=\prod_{i=1}^{n}(k-x_i), w_i=\prod_{j\neq i}(x_i-x_j)$ ，则 $f(x)=A(k)\sum_{i=1}^{n}\frac{y_i}{(k-x_i)w_i}$ 。

支持 $O(n)$ 动态加点（维护 w_i ），$O(n)$ 查询 $f(k)$ 。

4.13.1 求 k 次幂前缀和

易知 $f(n,k)=\sum_{i=1}^{n} i^k$ 是 $k+1$ 次多项式，需要 $k+2$ 个点插值。

$$x_i=i, y_i=\sum_{j=1}^{i} j^k$$
$$f(n,k)=\sum_{i=1}^{k+2} y_i \prod_{j\neq i}\frac{n-x_j}{x_i-x_j}=\sum_{i=1}^{k+2} y_i \prod_{j\neq i}\frac{n-i}{i-j}$$
$$pre_i=\prod_{j=1}^{i} n-i, suf_i=\sum_{j=i}^{k+2} n-i$$
$$f(n,k)=\sum_{i=1}^{k+2} y_i \frac{pre_{i-1}\cdot suf_{i+1}}{(i-1)!\cdot(-1)^{k+2-i}(k+2-i)!}$$

```
int Sumpow(LL n,int K){
    static int sum[maxk+5],pre[maxk+5],suf[maxk+5];
    for (int i=1;i<=K+2;i++) sum[i]=ADD(sum[i-1],Pow(i,K));
    if (n<=(K+=2)) return sum[n];
    pre[0]=1;for (int i=1;i<=K;i++) pre[i]=MUL(pre[i-1],(n-i)%MOD);
    suf[K+1]=1;for (int i=K;i;i--) suf[i]=MUL(suf[i+1],(n-i)%MOD);
    int ans=0;
    for (int i=1;i<=K;i++){
        int U=MUL(pre[i-1],suf[i+1]);
        int D=MUL(INV[i-1],INV[K-i]);
        ans=ADD(ans,MUL(sum[i],MUL(U,K-i&1?MOD-D:D)));
    }
    return ans;
}
```

4.14 FWT

```
void FWT(int *a,int n,int f){
    for (int k=1;k<n;k<<=1)
        for (int i=0;i<n;i+=(k<<1))
            for (int j=0;j<k;j++)
                if (f==1){
                    int x=a[i+j],y=a[i+j+k];
                    //and:a[i+j]+=a[i+j+k];
                    //or :a[i+j+k]+=a[i+j];
                    //xor:a[i+j]=x+y;a[i+j+k]=x-y;
                } else{
                    int x=a[i+j],y=a[i+j+k];
                    //and:a[i+j]-=a[i+j+k];
                    //or :a[i+j+k]-=a[i+j];
                    //xor:a[i+j]=(x+y)/2;a[i+j+k]=(x-y)/2;
                }
}
```

4.15 分治 FWT

$f_i = \sum_{j \oplus k = i (j>i)} f_j p_k$ ，f_n 已知，给出 p 。（$j < i$ 同理）

当分治到 $[L, R]$ ，对应二进制位为 d 的时候，不难发现 $[L, mid]$ 第 d 位是 0 ，$[mid+1, R]$ 第 d 位是 1 ，且 d 更高位相同。同时，由于 d 更高位相同，所以 $p(k)$ 的 k 中 d 的更高位一定是 0 ，而 d 位一定是 1 。因此只需要考虑 d 低位的 2^d 个数即可。

```
void cdqFWT(int L,int R){
    if (L==R) {/*Update here*/ return;}
    int mid=L+(R-L>>1),t=R-L+1>>1;
    cdqFWT(mid+1,R);
    for (int i=mid+1;i<=R;i++) A[i-mid-1]=f[i];
    for (int i=0;i<t;i++) B[i]=p[i+t];
    FWT(A,t,1);FWT(B,t,1);
    for (int i=0;i<t;i++) A[i]=MUL(A[i],B[i]);
    FWT(A,t,-1);
    for (int i=L;i<=mid;i++) f[i]=ADD(f[i],A[i-L]);
    cdqFWT(L,mid);
}
```

4.16 FMT

并集卷积：$h_s = \sum_{A \cup B = s} f_A g_B$

$$\hat{f}_s = \sum_{t \subseteq s} f_t$$

$$\hat{h}_s = \sum_{t \subseteq s} h_t = \sum_{t \subseteq s} \sum_{A \cup B = t} f_A g_B = \sum_A \sum_B [A \cup B \subseteq s] f_A g_B = \sum_{A \subseteq s} f_A \sum_{B \subseteq s} g_B = \hat{f}_s \hat{g}_s$$

交集卷积：$h_s = \sum_{A \cap B = s} f_A g_B$

$$\hat{f}_s = \sum_{t \supseteq s} f_t$$

$$\hat{h}_s = \sum_{t \supseteq s} h_t = \sum_{t \supseteq s} \sum_{A \cap B = t} f_A g_B = \sum_A \sum_B [A \cap B \supseteq s] f_A g_B = \sum_{A \supseteq s} f_A \sum_{B \supseteq s} g_B = \hat{f}_s \hat{g}_s$$

快速莫比乌斯变换就是高维前缀和，逆变换就是把加号改为减号（实现容斥过程）。

```
void FMT(int *a,int n,int f){ // or
    for (int i=0;i<n;i++)
        for (int s=1<<i;s<(1<<n);s=(s+1)|(1<<i))
            a[s]=ADD(a[s],f>0?a[s^(1<<i)]:MOD-a[s^(1<<i)]);
}
void FMT(int *a,int n,int f){ // and
    for (int i=0;i<n;i++)
        for (int s=1<<i;s<(1<<n);s=(s+1)|(1<<i))
            a[s^(1<<i)]=ADD(a[s^(1<<i)],f>0?a[s]:MOD-a[s]);
}
```

4.17 子集卷积

子集卷积：$h_s = \sum_A \sum_B [A \cup B = s][A \cap B = \emptyset] f_A g_B$
等价于 $h_s = \sum_A \sum_B [A \cup B = s][|A| + |B| = |s|] f_A g_B$
因此加上一维 i 表示集合大小，然后 $h_{i,s} = \sum_{j=0}^{i} \sum_{A \cup B = s} f_{j,A} g_{i-j,B}$
可以用 FMT 实现。

```
for (int i=0;i<=n;i++){
    FMT(f[i],n,1);FMT(g[i],n,1);
    for (int j=0;j<=i;j++)
        for (int s=0;s<(1<<n);s++)
            h[i][s]=ADD(h[i][s],MUL(f[j][s],g[i-j][s]));
    FMT(h[i],n,-1);
}
```

5 代数

可以用矩阵乘法处理的代数结构为半环 $(A, +, \cdot)$，即满足：

1. $(A, +)$ 为带有单位元 0 的交换幺半群（单位元，结合律，交换律）；
2. $(A, \cdot)$ 为带有单位元 1 的幺半群（单位元，结合律）；
3. 乘法对加法同时有左、右分配律；加法单位元 0 抵消乘法。

ID	A	+	0	·	1
1	$\mathbb{R}$	min	$+\infty$	max	$-\infty$
2	$\mathbb{R}$	max	$-\infty$	min	$+\infty$
3	$\mathbb{R} \cup \{-\infty\}$	max	$-\infty$	ordinary addition $+$	0
4	$\mathbb{R} \cup \{+\infty\}$	min	$+\infty$	ordinary addition $+$	0
5	$\forall n \in \mathbb{N}^+,\ [0, 2^n) \cap \mathbb{N}$	bitwise OR	0	bitwise AND	$2^n - 1$
6	$\forall n \in \mathbb{N}^+,\ [0, 2^n) \cap \mathbb{N}$	bitwise AND	$2^n - 1$	bitwise OR	0
7	$\forall n \in \mathbb{N}^+,\ [0, 2^n) \cap \mathbb{N}$	bitwise XOR	0	bitwise AND	$2^n - 1$

```cpp
int n, m, a[N][N];
Matrix(int _n = N - 1, int _m = N - 1, bool I = false) : n(_n), m(_m) {
    memset(a, 0, sizeof(a)); if (I) rep(i, 1, n) a[i][i] = 1;
}
inline Matrix operator * (const Matrix &obj) const { // assert(m == obj.n);
    Matrix res(n, obj.m);
    rep(i, 1, n) rep(k, 1, m)
        if (a[i][k] != 0) rep(j, 1, obj.m)
            res.a[i][j] = (res.a[i][j] + 1ll * a[i][k] * obj.a[k][j]) % mod;
    return res;
}
inline Matrix fpow(ll t) const {
    Matrix res(n, n, true), tmp = *this;
    for (; t; t >>= 1, tmp = tmp * tmp) if (t & 1) res = res * tmp;
    return res;
}
inline pair<bool, Matrix> Inv() { // assert(n == m);
    Matrix res(n, n, true);
    rep(i, 1, n) {
        int nw = i;
        rep(j, i, n) if (a[j][i]) {nw = j; break;}
        if (a[nw][i] == 0) return make_pair(false, res);
        if (nw != i) {swap(a[nw], a[i]); swap(res.a[nw], res.a[i]);}
        int inv = ::fpow(a[i][i], mod - 2);
        rep (k, 1, n) {
            a[i][k] = 1ll * a[i][k] * inv % mod;
            res.a[i][k] = 1ll * res.a[i][k] * inv % mod;
        }
        rep(j, 1, n) if (j != i && a[j][i]) {
            ll cof = mod - a[j][i];
            rep(k, i, n) a[j][k] = (a[j][k] + cof * a[i][k]) % mod;
            rep(k, 1, n) res.a[j][k] = (res.a[j][k] + cof * res.a[i][k]) % mod;
        }
    }
    return make_pair(true, res);
}
```

5.1 行列式

$$\det A \triangleq \sum_{i_1 i_2 \cdots i_n} (-1)^{\tau(i_1 i_2 \cdots i_n)} a_{1,i_1} a_{2,i_2} \cdots a_{n,i_n}$$

1. 矩阵转置行列式不变。下述所有对行成立的性质对列都成立。
2. 行倍加行列式不变。行交换行列式符号取反。行倍乘系数可以提到外面（注意 $\det kA = k^n \det A$）。
3. 行列式具有分行可加性。若两行成比例，则行列式值为 0 。
4. 方阵乘法可取行列式：若 A, B 为 n 阶方阵，则 $\det AB = \det A \det B$ 。推论 $\det A^{-1} = \dfrac{1}{\det A}$ 。
4. 行列式可按行展开，展开定理：$\sum_{k=1}^{n} a_{i,k} A_{j,k} = \begin{cases} \det A, & (i = j) \\ 0, & (i \neq j) \end{cases}$
5. 范德蒙德行列式：$D_n = \begin{vmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{vmatrix} = \prod_{1 \leq i < j \leq n} (x_j - x_i)$

```cpp
inline int Det() {
    assert(n == m);
    bool fl = 0;
    rep(i, 1, n) { // Mod a Prime!!
        int nw = i;
        rep(j, i, n) if (a[j][i]) {nw = j; break;}
        if (a[nw][i] == 0) return 0;
        if (nw != i) {swap(a[nw], a[i]); fl ^= 1;}
        int inv = ::fpow(a[i][i], mod - 2);
        rep(j, i + 1, n)
            if (a[j][i]) {
                ll cof = mod - 1ll * inv * a[j][i] % mod;
                rep(k, i, n) a[j][k] = (a[j][k] + cof * a[i][k]) % mod;
            }
    }
    rep(i, 1, n) { // Mod a simple number!!
        int nw = i;
        rep(j, i, n) if (a[j][i] && (!a[nw][i] || a[nw][i] > a[j][i])) nw = j;
        if (a[nw][i] == 0) return 0;
        if (nw != i) {swap(a[nw], a[i]); fl ^= 1;}
        rep(j, i + 1, n)
            while (a[j][i]) { // 辗转相除，复杂度还是 O(n^3) 的
                if (a[i][i] > a[j][i]) {swap(a[j], a[i]); fl ^= 1;}
                int cof = mod - a[j][i] / a[i][i];
                rep(k, i, n) a[j][k] = (a[j][k] + 1ll * cof * a[i][k]) % mod;
            }
    }
    int res = 1;
    for (int i = 1; i <= n; ++i) res = 1ll * res * a[i][i] % mod;
    return fl ? (mod - res) % mod : res;
}
```

5.2 伴随矩阵

$$A^* \triangleq \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{pmatrix}^T$$

其中 A_{ij} 是 A 中 (i,j) 元的代数余子式。

1. $A^*A = AA^* = |A|I$ ， $|A^*| = |A|^{n-1}$ 。
2. A 可逆时：(a) $A^* = |A|A^{-1}$ ，复杂度 $\mathcal{O}(n^3)$ ；(b) A^* 可逆，此时 $(A^*)^{-1} = \dfrac{A}{|A|}$ 。
3. $R(A^*) = \begin{cases} n, & \text{若} R(A) = n \\ 1, & \text{若} R(A) = n-1 \\ 0, & \text{若} R(A) < n-1 \end{cases}$ （设 $A_{m\times n}B_{n\times t} = 0$ ，则 $R(A_{m\times n}) + R(B_{n\times t}) \le n$ ）

克拉默法则

设 A 是 n 阶可逆矩阵，$\forall b \in \mathbb{R}^n$，方程 $Ax = b$ 有唯一解：

解为 $x_i = \dfrac{\det A_i(b)}{\det A}$ ，其中 $A_i(b)$ 表示 A 中第 i 列替换成 b 得到的矩阵。

计算伴随矩阵后可以快速查询代数余子式，复杂度 $\mathcal{O}(n^3 + nq)$。

5.3 矩阵树定理

对于图 G ，求 $\sum_{T\in\text{图 G 的生成树}} \prod_{e\in T} w_e$ 的值。

无向图生成树：$K_{i,j} = \begin{cases} -\sum_{(i,j,w)\in E} w, & i \ne j \\ \sum_k \sum_{(i,k,w)\in E} w, & i = j \end{cases}$ ，任意余子式都是所有生成树权值的和。

有向图外向树：$K_{i,j} = \begin{cases} -\sum_{(i,j,w)\in E} w, & i \ne j \\ \sum_k \sum_{(k,i,w)\in E} w, & i = j \end{cases}$ ，余子式 $A_{i,i}$ 是所有以 i 为根外向生成树权值的和。

有向图内向树：$K_{i,j} = \begin{cases} -\sum_{(i,j,w)\in E} w, & i \ne j \\ \sum_k \sum_{(i,k,w)\in E} w, & i = j \end{cases}$ ，余子式 $A_{i,i}$ 是所有以 i 为根内向生成树权值的和。

即对于主对角线，内向树只考虑出边，外向树只考虑入边。

乘积变求和

对于图 G ，求 $\sum_{T\in\text{图 G 的生成树}} \sum_{e\in T} w_e$

把边的贡献写成 $w_e x + 1$ ，答案就是行列式函数的一次项的系数，乘法变成了模 x^2 的多项式乘法。

BEST 定理

对于有向图 $G = (V, E)$ ：记 d_i 为点 i 的出度，$T_u(G)$ 是 G 的以点 u 为根的内向生成树的个数。

则从点 s 出发的欧拉回路数量为:

$$F(s) = T_s(G) \times d_s \times \prod_{u\in V} (d_u - 1)!$$

整个有向图 G 的欧拉回路数量为：随意指定一点 s，$F(G) = T_s(G) \times \prod_{u\in V} (d_u - 1)!$

存在欧拉回路的有向图 G 满足：对于任意点 $s, T_s(G)$ 相等。

5.4 LGV 引理

G 是一个有限的带权有向无环图。每个顶点的度是有限的, 不存在有向环 (所以路径数量是有限的)。
起点 $A=\{a_1,\cdots,a_n\}$, 终点 $B=\{b_1,\cdots,b_n\}$ 。
每条边 e 有权 w_e, 并假定值属于某交换环。对于一个有向路径 P, 定义 $\omega(P)$ 为路径上所有边权的积。
对任意顶点 a,b, 定义 $e(a,b)=\sum_{P:a\to b}\omega(P)$ 。
设矩阵

$$M=\begin{pmatrix} e(a_1,b_1) & e(a_1,b_2) & \cdots & e(a_1,b_n)\\ e(a_2,b_1) & e(a_2,b_2) & \cdots & e(a_2,b_n)\\ \vdots & \vdots & \ddots & \vdots\\ e(a_n,b_1) & e(a_n,b_2) & \cdots & e(a_n,b_n)\end{pmatrix}$$

从 A 到 B 的不相交路径 $P=(P_1,P_2,\cdots,P_n),P_i$ 表示从 a_i 到 $b_{\sigma(i)}$ 的一条路径, 其中 σ 是一个排列（置换）, 并且满足对任意 $i\neq j,P_i$ 与 P_j 没有公共点。记 $\sigma(P)$ 表示 P 对应的排列。

定理 M 的行列式是所有从 A 到 B 的不相交路径 $P=(P_1,\cdots,P_n)$ 的带符号和。
其中符号指 $\sigma(P)$ 的逆序数的奇偶性：$(-1)^{\text{逆序数}}$ ，记为 $\operatorname{sign}(\sigma(P))$。

$$\det(M)=\sum_{P:A\to B}\operatorname{sign}(\sigma(P))\prod_{i=1}^{n}\omega(P_i)$$

5.5 特征值

设 A 是 n 阶方阵, 若存在数 λ 和非零向量 ξ, 使得 $A\xi=\lambda\xi$ 则称 λ 是 A 的特征值。

对角化

设 $A_{n\times n}$ 的互异的特征值为 $\lambda_1,\lambda_2,\cdots,\lambda_s$
则 A 可对角化 $\Leftrightarrow$ 对每一个 λ_i, 它的几何重数 = 代数重数（特殊情况 n 个特征值），其中：

1. λ_i 的代数重数, 指在特征方程中 λ_i 的重根数
2. λ_i 的几何重数, 指特征空间 V_{λ_i} 的维数 $=\dim Nul(A-\lambda_iI)=n-R(A-\lambda_iI)$

此时, 可令

$$\begin{aligned}P&=(\xi_{11},\xi_{12},\cdots,\xi_{1r_1},\xi_{21},\xi_{22},\cdots,\xi_{2r_2},\cdots,\xi_{s1},\xi_{s2},\cdots,\xi_{sr_s})\\ \Lambda&=\operatorname{diag}(\underbrace{\lambda_1,\lambda_1,\cdots,\lambda_1}_{r_1\text{个}},\underbrace{\lambda_2,\lambda_2,\cdots,\lambda_2}_{r_2\text{个}},\cdots,\underbrace{\lambda_s,\lambda_s,\cdots,\lambda_s}_{r_s\text{个}})\end{aligned}$$

则 $P^{-1}AP=\Lambda$ ，有多项式 $\varphi(A)=P\varphi(\Lambda)P^{-1}=P\operatorname{diag}(\varphi(\lambda_1),\varphi(\lambda_2),\ldots,\varphi(\lambda_n))P^{-1}$

特征多项式

A 的特征多项式 $f(\lambda)=|A-\lambda I|$ ，全体特征值为 $f(\lambda)$ 的根。可 $\mathcal{O}(n^4)$ 拉格朗日插值。
因为相似矩阵 $B=PAP^{-1}$ 特征多项式相同，于是考虑通过初等变换来构造 B :
1. 行交换: $E_{i,j}AE_{i,j}^{-1}$ ，互换第 i,j 行、第 i,j 列。
2. 行倍乘: $E_i(k)AE_i^{-1}(k)$, 第 i 行乘以 k, 第 i 列乘以 $\frac{1}{k}$
3. 行倍加：$E_{i,j}(k)AE_{i,j}^{-1}(k)$, 第 i 行加上 k 倍的第 j 行, 第 j 列减去 k 倍的第 i 列。

由于每次对第 i 行的操作都会影响到第 i 列，所以考虑将 A 化简为上海森堡矩阵，分离主元的行列。

令第 i 行到第 n 行, 第 i 列到第 n 列相交形成的矩阵的特征多项式为 $D_i(x)$ （令 $D_{n+1}(x)=1$ ）。
则按照第一列展开，如果选择的是第二行，再按照第一行展开可以得到：

$$D_i(x)=(a_{i,i}-x)D_{i+1}(x)+\sum_{j=i+1}^{n}a_{i,j}(-1)^{j-i}D_{j+1}(x)\prod_{k=i+1}^{j}a_{k,k+1}$$

这样直接后往前计算 $D_i(x)$, 复杂度也是 $O(n^3)$ 的。加上前面化简矩阵, 整体的复杂度就是 $O(n^3)$ 了。

5.6 置换群

置换

n 元集合 Ω 到它自身的一个一一映射，称为 Ω 上的一个置换或 n 元置换。
Ω 上的置换为，上面为 1 到 n $(\alpha_1 \sim \alpha_n)$ ，下面是 n 阶排列 $(\alpha_{1j} \sim \alpha_{nj})$，表示 α_i 由 α_{ij} 取代。
置换的运算是连接。设置换 A, B ，A 下元素 i 由 $a[i]$ 取代，则 $A \times B = C$，其中 $c[i] = b[a[i]]$。

置换群

若一组置换在连接运算下构成一个群，则称其为置换群。
单位元：$1, 2, 3, ..., n$
A 的逆元 B ： $b[a[i]] = i$
若 G 有限，则存在最小正整数 r 使得 $A^r = E$，A^{r-1} 为逆元，r 为置换 A 轮换的最小公倍数。

Burnside 引理

集合 X 在有限群 G 作用下本质不同的元素个数为

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$

其中 X^g 表示集合 X 在 g 作用下的不动点。
即：一个置换群的等价类数目等于这个置换群中所有置换的不动点数目的平均值。

Pólya 定理

Pólya 定理是 Burnside 引理在染色情况下的一个特例。
如果两种染色方案，经过变换可以变成相同的，称之为在一个**轨道**上。
我们把排列写成若干个轮换的形式，其中轮换的个数叫做**循环指标**。
设有限群 G 有 m 个置换，第 i 个置换有 a_i 个循环，现在要将所有的点染成 c 种颜色。

$$\text{染色后群 } G \text{ 的等价类数目} L = \frac{\sum_{i=1}^{m} c^{a_i}}{m}$$

不动点要求所有循环的颜色相同，每个循环有 c 种颜色选择，所以不动点数目为 c^{a_i}。

一个例题

n 个点的环，每个点染 m 种颜色之一，旋转和反转都视为同构，问不同的染色方案数。
先只考虑旋转，假如旋转 k 个位置，考虑此时循环的长度 x ，是 $kx \equiv 0 \pmod{n}$ 的最小正整数解（转了 x 次后回到原处）。

$$x = \frac{\operatorname{lcm}(k, n)}{k} = \frac{n}{\gcd(k, n)} \text{ ，因此循环个数为} \frac{n}{\frac{n}{\gcd(k,n)}} = \gcd(k, n) \text{ ，方案数为} m^{\gcd(k,n)}$$

再考虑翻转，**对于任意的旋转-翻转-旋转操作都等同于一次翻转操作**，因此只需要统计所有本质不同的翻转操作的答案。此时点数的奇偶性不同会导致对称轴产生方式不同。
若 n 为奇数，则只会产生点-边对称轴，此时个数为 n 个，循环的组数为 $\frac{n}{2} + 1$，方案数为 $nm^{\frac{n}{2}+1}$。
若 n 为偶数，则会产生点-点对称轴和边-边对称轴两种，各 $\frac{n}{2}$ 个，其中点-点对称轴循环的组数为 $\frac{n}{2} + 1$，边-边对称轴循环的组数为 $\frac{n}{2}$，方案数为 $\frac{n}{2}(m^{\frac{n}{2}} + m^{\frac{n}{2}+1})$ 。

6 组合容斥

6.1 常用公式

$2n$ 个元素两两组合的方案数为 $\prod_{i=1}^{n}(2i-1)=(2n-1)!!$ 。

卡特兰数：$f(n)=\frac{1}{n+1}\binom{2n}{n}$ 。

错排：$D(n)=(n-1)[D(n-2)+D(n-1)], D(0)=1, D(1)=0$ 。

Lucas 定理：$\binom{x}{y}=\binom{x \bmod p}{y \bmod p}\binom{\lfloor \frac{x}{p} \rfloor}{\lfloor \frac{y}{p} \rfloor}$ 。

6.2 组合背包

n 个物品，$f_{t,i,j}$ 表示前 t 个物品选 i 个，容量为 j 方案数，给定 K ，需要对每个 j 求：

$$ans_j=\sum_{i=K}^{n}(-1)^{i-K}\binom{i-1}{K-1}f_{n,i,j}$$

令 $F_{t,j,k}=\sum_{i=k}^{n}(-1)^{i-k}\binom{i-1}{k-1}f_{t,i,j}$ ，则：

$$\begin{aligned}
f_{t,i,j}&=f_{t-1,i,j}+f_{t-1,i-1,j-p_t}\\
\binom{i-1}{k-1}&=\binom{i-2}{k-2}+\binom{i-2}{k-1}\\
F_{t-1,j,k}&=\sum_{i=k}^{n}(-1)^{i-k}\binom{i-1}{k-1}f_{t-1,i,j}\\
F_{t-1,j-p_t,k-1}&=\sum_{i=k-1}^{n}(-1)^{i-k+1}\binom{i-1}{k-2}f_{t-1,i,j-p_t}\\
&=\sum_{i=k}^{n}(-1)^{i-k}\binom{i-2}{k-2}f_{t-1,i-1,j-p_t}\\
F_{t-1,j-p_t,k}&=\sum_{i=k}^{n}(-1)^{i-k}\binom{i-1}{k-1}f_{t-1,i,j-p_t}\\
&=\sum_{i=k+1}^{n}(-1)^{i-1-k}\binom{i-2}{k-1}f_{t-1,i-1,j-p_t}\\
&=-\sum_{i=k}^{n}(-1)^{i-k}\binom{i-2}{k-1}f_{t-1,i-1,j-p_t}\\
F_{t,j,k}&=F_{t-1,j,k}+F_{t-1,j-p_t,k-1}-F_{t-1,j-p_t,k}
\end{aligned}$$

6.3 二项式反演

$$f_n=\sum_{i=0}^{n}(-1)^i\binom{n}{i}g_i \Leftrightarrow g_n=\sum_{i=0}^{n}(-1)^i\binom{n}{i}f_i$$

$$f_n=\sum_{i=0}^{n}\binom{n}{i}g_i \Leftrightarrow g_n=\sum_{i=0}^{n}(-1)^{n-i}\binom{n}{i}f_i$$

证明考虑直接代入。

6.4 Min-Max 容斥

对于一个集合 S ，定义 $\max(S)$ 为 S 中最大值，$\min(S)$ 为 S 中最小值，设：

$$\max(S)=\sum_{T\subseteq S}f(|T|)\min(T)$$

设 a_i 为 S 中第 i 大的数，则：

$$\max(S)=\sum_{i=0}^{|S|-1}a_{i+1}\sum_{j=0}^{i}f(j+1)\binom{i}{j}=a_1$$

$$\sum_{j=0}^{i}f(j+1)\binom{i}{j}=[i=0]$$

$$f(i+1)=\sum_{j=0}^{i}(-1)^{i-j}\binom{i}{j}[j=0]=(-1)^i,f(i)=(-1)^{i-1}$$

$$\max(S)=\sum_{T\subseteq S}(-1)^{|T|-1}\min(T)$$

同理，$\max_k(S)$ 为 S 中第 k 大值，则：

$$f(i+1)=\sum_{j=0}^{i}(-1)^{i-j}\binom{i}{j}[j=k-1]=(-1)^{i-k+1}\binom{i}{k-1},f(i)=(-1)^{i-k}\binom{i-1}{k-1}$$

$$\max_k(S)=\sum_{T\subseteq S}(-1)^{|T|-k}\binom{|T|-1}{k-1}\min(T)$$

$\min,\max$ 互换也成立（显然对称），期望意义下也成立。

6.5 广义容斥

求选出恰好 K 个时的方案数（集合答案仅与大小有关时下面的推导才成立）。
设 ans_S 表示选出集合为 S 时的方案数，ans_i 表示选出 i 个的方案数。a_i 表示至少选了 i 个的方案数。

$$ans_K=\sum_S f(|S|)a_S=\sum_S f(|S|)\sum_{S\subseteq T}ans_T=\sum_T ans_T\sum_{S\subseteq T}f(|S|)=\sum_{i=0}^{n}ans_i\sum_{j=0}^{i}\binom{i}{j}f(j)$$

$$\sum_{j=0}^{i}\binom{i}{j}f(j)=[i=K]$$

二项式反演：

$$f(n)=\sum_{i=0}^{n}(-1)^{n-i}\binom{n}{i}[i=K]=(-1)^{n-K}\binom{n}{K}$$

$$ans_K=\sum_{i=0}^{n}f(i)a_i=\sum_{i=K}^{n}(-1)^{i-K}\binom{i}{K}a_i$$

6.6 牛顿二项式定理

$$(x+y)^\alpha=\sum_{i=0}^{+\infty}\binom{\alpha}{i}x^iy^{\alpha-i}\ ,\quad \binom{\alpha}{i}=\frac{\prod_{j=0}^{i-1}(\alpha-j)}{i!}$$

其中 α 是实数。

6.7 序列容斥

从一个位置走到下一个位置有若干种走法（合法或非法）。定义 f_i 表示走了 i 格均合法的方案数，g_i 表示走了 i 格均非法的方案数。特殊的，g_1 也认为非法，f_0 为初值。

先考虑前 $n-1$ 行都合法，则权值为 $f_{n-1}g_1$ ，这样有个问题，就是 $n-1$ 行到 n 行时不一定合法，因此我们强制 $n-1$ 到 n 非法，即减去 $f_{n-2}g_2$ ，但是类似的会发现 $f_{n-2}g_2$ 里包含了 $n-2$ 到 $n-1$ 非法的情况是我们多减的，所以加上 $f_{n-3}g_3$ 。以此类推会发现这是个容斥：

$$f_n=\sum_{i=1}^{n}(-1)^{i-1}f_{n-i}g_i=\sum_{i=1}^{n}f_{n-i}(-1)^{i-1}g_i$$

$$h_i=(-1)^{i-1}g_i,f_n=\sum_{i=1}^{n}f_{n-i}h_i$$

即 $f=f*h+f_0,f=\frac{f_0}{1-h}$ ，如果能求出 g 即可求出 f 。

6.8 q-binomial

$$[n]_q=\sum_{i=0}^{n-1}q^i=\lim_{x\to q}\frac{1-x^n}{1-x},[n]!_q=\prod_{i=1}^{n}[i]_q,\begin{bmatrix}n\\m\end{bmatrix}_q=\frac{[n]!_q}{[m]!_q[n-m]!_q}$$

$$\begin{bmatrix}n\\m\end{bmatrix}_q=\lim_{x\to q}\frac{\prod_{1\le i\le n}\frac{1-x^i}{1-x}}{\prod_{1\le i\le m}\frac{1-x^m}{1-x}\prod_{1\le i\le n-m}\frac{1-x^m}{1-x}}=\lim_{x\to q}\frac{\prod_{n-m+1\le i\le n}(1-x^i)}{\prod_{1\le i\le m}(1-x^i)}$$

结论 1：$\begin{bmatrix}n\\m\end{bmatrix}_q=\begin{bmatrix}n-1\\m-1\end{bmatrix}_q+q^m\begin{bmatrix}n-1\\m\end{bmatrix}_q=q^{n-m}\begin{bmatrix}n-1\\m-1\end{bmatrix}_q+\begin{bmatrix}n-1\\m\end{bmatrix}_q$

(1) 组合意义：$\begin{bmatrix}n\\m\end{bmatrix}_q$ 表示 $(0,0)$ 向上向右走到 $(n-m,m)$ 的所有方案中 q^{count} 的和，$count$ 表示路径右下角/左上角点的个数。

(2) 说明：考虑坐标变换 $(n,m)\to(n-m,m)$ ，这样式子中就变成了向上向右走，然后考虑 q^m 为每次向右走时下面点的个数，同理 q^{n-m} 为每次向上走时左边点的个数。

结论 2：$\prod_{i=0}^{n-1}(1+q^iy)=\sum_{i=0}^{n}q^{\binom{i}{2}}\begin{bmatrix}n\\i\end{bmatrix}_q y^i$

(1) 组合意义：$\begin{bmatrix}n\\i\end{bmatrix}_q$ 表示在长度为 n 的序列里选 i 个位置的所有方案中 $\prod_{j=1}^{i}q^{pre_j}$ 的和，pre_j 表示第 j 个选的位置前面有多少个没有被选。

(2) 说明：式子前面的生成函数表示第 i 个位置选的贡献为 q^i ，y^i 的系数表示选了 i 个位置的贡献乘积和。然后考虑 $q^{\binom{i}{2}}$ ，表示每个选了的位置往前和其他选了的位置的贡献，因此考虑从 y^i 的系数中去掉 $q^{\binom{i}{2}}$ 就是 $\begin{bmatrix}n\\i\end{bmatrix}_q$ 的组合意义。

结论 3：$\begin{bmatrix}n+m\\k\end{bmatrix}_q=\sum_{i=0}^{k}q^{(n-i)(k-i)}\begin{bmatrix}n\\i\end{bmatrix}_q\begin{bmatrix}m\\k-i\end{bmatrix}_q$ ，证明考虑结论 2 组合意义，两边合并。

结论 4：$\begin{bmatrix}n+m+1\\n+1\end{bmatrix}_q=\sum_{i=0}^{m}q^i\begin{bmatrix}n+i\\n\end{bmatrix}_q$ ，证明 $\begin{bmatrix}n\\m\end{bmatrix}_q=q^{n-m}\begin{bmatrix}n-1\\m-1\end{bmatrix}_q+\begin{bmatrix}n-1\\m\end{bmatrix}_q$ 展开 m 次。

结论 5：$\frac{1}{\prod_{i=0}^{n}(1-q^ix)}=\sum_{i\ge0}x^i\begin{bmatrix}i+n\\n\end{bmatrix}_q$

6.9 第一类斯特林数

第一类斯特林数：n 个数分成 m 个圆排列，且没有空排列的方案数。

$$\begin{bmatrix} n \\ m \end{bmatrix} = (n-1)\begin{bmatrix} n-1 \\ m \end{bmatrix} + \begin{bmatrix} n-1 \\ m-1 \end{bmatrix}$$

$$x^{\overline{n}} = \prod_{i=0}^{n-1}(x+i) = \sum_{i=0}^{n}\begin{bmatrix} n \\ i \end{bmatrix} x^i$$

第二个式子：观察递推式可知另一个意义，对于 $[1,n]$ 的每个数都有 1 种方案选和 $i-1$ 种方案不选，求选出 m 个数的方案数，定义上升幂 $x^{\overline{n}} = \prod_{i=0}^{n-1}(x+i)$ ，则 $x^{\overline{n}}$ 第 m 项系数即为 $\begin{bmatrix} n \\ m \end{bmatrix}$ 。

6.9.1 求一行

求 $\begin{bmatrix} n \\ 0 \end{bmatrix}, \begin{bmatrix} n \\ 1 \end{bmatrix}, \cdots, \begin{bmatrix} n \\ n \end{bmatrix}$ 。

$$x^{\overline{n}} = \prod_{i=0}^{n-1}(x+i) = \sum_{i=0}^{n}\begin{bmatrix} n \\ i \end{bmatrix} x^i$$

$$x^{\overline{2n}} = x^{\overline{n}}(x+n)^{\overline{n}}$$

$$\begin{aligned}(x+n)^{\overline{n}} &= \sum_{i=0}^{n} a_i (x+n)^i \\ &= \sum_{i=0}^{n} a_i \sum_{j=0}^{i}\binom{i}{j} x^j n^{i-j} \\ &= \sum_{i=0}^{n} x^i \frac{1}{i!}\sum_{j=i}^{n} a_j j! \frac{n^{j-i}}{(j-i)!}\end{aligned}$$

利用倍增求出 $x^{\overline{n}}$ 即可得出。

```
void Stir1Row(int *S,int n){
    static int F[maxt+5],G[maxt+5];
    if (n==0) {S[0]=1;return;}
    if (n==1) {S[0]=0;S[1]=1;return;}
    int m=n>>1;Stir1Row(S,m);
    for (int i=0,pw=1;i<=m;i++,pw=MUL(pw,m))
        F[i]=MUL(S[m-i],fac[m-i]),G[i]=MUL(pw,INV[i]);
    int t;for (t=1;t<=(m<<1);t<<=1);
    for (int i=m+1;i<t;i++) F[i]=G[i]=0;
    NTT(F,t,1);NTT(G,t,1);
    for (int i=0;i<t;i++) F[i]=MUL(F[i],G[i]);
    NTT(F,t,-1);reverse(F,F+m+1);
    for (int i=0;i<=m;i++) F[i]=MUL(F[i],INV[i]),G[i]=S[i];
    for (int i=m+1;i<t;i++) F[i]=G[i]=0;
    NTT(F,t,1);NTT(G,t,1);
    for (int i=0;i<t;i++) F[i]=MUL(F[i],G[i]);
    NTT(F,t,-1);
    for (int i=0;i<=n;i++) S[i]=F[i];
    if (n&1) for (int i=n;~i;i--) S[i]=ADD(MUL(S[i],n-1),i?S[i-1]:0);
}
```

6.9.2 求一列

求 $\begin{bmatrix}0\\m\end{bmatrix},\begin{bmatrix}1\\m\end{bmatrix},\cdots,\begin{bmatrix}n\\m\end{bmatrix}$ 。

考虑指数型生成函数，令 $F(x)=\sum_{i=1}^{+\infty}(i-1)!\frac{x^i}{i!}$

则 $F(x)$ 的 $[\frac{x^i}{i!}]$ 表示 i 个数分成 1 个圆排列的方案数。

那么 $F(x)^m$ 的 $[\frac{x^i}{i!}]$ 表示 i 个数分成 m 个有顺序的圆排列的方案数。

所以 $\begin{bmatrix}i\\m\end{bmatrix}=[\frac{x^i}{i!}]\frac{F(x)^m}{m!}$ ，可以通过多项式快速幂求出（分治，或 ln+exp）。

```
void Pow(int *F,int *a,int n,int b){
    if (!b) return;
    Pow(F,a,n,b>>1);
    NTT(F,n,1);for (int i=0;i<n;i++) F[i]=MUL(F[i],F[i]);
    NTT(F,n,-1);for (int i=n>>1;i<n;i++) F[i]=0;
    if (b&1){
        NTT(F,n,1);for (int i=0;i<n;i++) F[i]=MUL(F[i],a[i]);
        NTT(F,n,-1);for (int i=n>>1;i<n;i++) F[i]=0;
    }
}
void Stir1Col(int *S,int n,int m){
    static int F[maxt+5],G[maxt+5];
    int t;for (t=1;t<=(n<<1);t<<=1);
    G[0]=0;for (int i=1;i<=n;i++) G[i]=MUL(fac[i-1],INV[i]);
    for (int i=n+1;i<t;i++) G[i]=0;
    NTT(G,t,1);
    F[0]=1;for (int i=1;i<t;i++) F[i]=0;
    Pow(F,G,t,m);
    for (int i=0;i<=n;i++) S[i]=MUL(MUL(F[i],fac[i]),INV[m]);
}
```

6.10 第二类斯特林数

第二类斯特林数：n 个不同元素分成 m 个相同集合，且没有空集的方案数。

$$\begin{Bmatrix}n\\m\end{Bmatrix}=m\begin{Bmatrix}n-1\\m\end{Bmatrix}+\begin{Bmatrix}n-1\\m-1\end{Bmatrix}$$

$$\begin{Bmatrix}n\\m\end{Bmatrix}=\frac{1}{m!}\sum_{i=0}^{m}(-1)^i\binom{m}{i}(m-i)^n$$

$$n^k=\sum_{i=0}^{min\{n,k\}}\binom{n}{i}\begin{Bmatrix}k\\i\end{Bmatrix}i!$$

第二个式子是个容斥，i 枚举的是空盒子的个数，那么 n 个球可以在其他盒子乱放，但由于第二类斯特林数表示的是相同盒子，所以最后要除以 $m!$ 。

第三个式子左边表示 k 个不同的球放进 n 个不同的盒子的方案数，右边枚举了非空盒子的个数 i ，那么把 k 个球放入这 i 个盒子，由于第二类斯特林数表示的是相同盒子，所以还要乘上 $i!$ 。

6.10.1 求一行

求 $\left\{{n \atop 0}\right\},\left\{{n \atop 1}\right\},\cdots,\left\{{n \atop n}\right\}$ 。

$$\left\{{n \atop m}\right\}=\frac{1}{m!}\sum_{i=0}^{m}(-1)^i\binom{m}{i}(m-i)^n=\sum_{i=0}^{m}\frac{(-1)^i}{i!}\frac{(m-i)^n}{(m-i)!}$$

显然是一个卷积。

```
void Stir2Row(int *S,int n){
    static int A[maxt+5],B[maxt+5];
    int t;for (t=1;t<=(n<<1);t<<=1);
    for (int i=0;i<=n;i++) A[i]=(i&1?MOD-INV[i]:INV[i]),B[i]=MUL(Pow(i,n),INV[i]);
    for (int i=n+1;i<t;i++) A[i]=B[i]=0;
    NTT(A,t,1);NTT(B,t,1);
    for (int i=0;i<t;i++) A[i]=MUL(A[i],B[i]);
    NTT(A,t,-1);
    for (int i=0;i<=n;i++) S[i]=A[i];
}
```

6.10.2 求一列

求 $\left\{{0 \atop m}\right\},\left\{{1 \atop m}\right\},\cdots,\left\{{n \atop m}\right\}$ 。

考虑指数型生成函数，令 $F(x)=\sum_{i=1}^{+\infty}\frac{x^i}{i!}=e^x-1$

则 $F(x)$ 的 $[\frac{x^i}{i!}]$ 表示 i 个不同元素放进 1 个盒子的方案数。

那么 $F(x)^m$ 的 $[\frac{x^i}{i!}]$ 表示 i 个不同元素放进 m 个不同盒子的方案数。

所以 $\left\{{i \atop m}\right\}=[\frac{x^i}{i!}]\frac{F(x)^m}{m!}$ ，可以通过多项式快速幂求出（分治，或 ln+exp）。

```
void Pow(int *F,int *a,int n,int b){
    if (!b) return;
    Pow(F,a,n,b>>1);
    NTT(F,n,1);for (int i=0;i<n;i++) F[i]=MUL(F[i],F[i]);
    NTT(F,n,-1);for (int i=n>>1;i<n;i++) F[i]=0;
    if (b&1){
        NTT(F,n,1);for (int i=0;i<n;i++) F[i]=MUL(F[i],a[i]);
        NTT(F,n,-1);for (int i=n>>1;i<n;i++) F[i]=0;
    }
}
void Stir2Col(int *S,int n,int m){
    static int F[maxt+5],G[maxt+5];
    int t;for (t=1;t<=(n<<1);t<<=1);
    G[0]=0;for (int i=1;i<=n;i++) G[i]=INV[i];
    for (int i=n+1;i<t;i++) G[i]=0;
    NTT(G,t,1);
    F[0]=1;for (int i=1;i<t;i++) F[i]=0;
    Pow(F,G,t,m);
    for (int i=0;i<=n;i++) S[i]=MUL(MUL(F[i],fac[i]),INV[m]);
}
```

6.11 斯特林科技

6.11.1 上升幂下降幂

$$
\begin{aligned}
x^{\overline{n}} &= \prod_{i=0}^{n-1}(x+i) = \binom{x+n-1}{n} n! \\
x^{\underline{n}} &= \prod_{i=0}^{n-1}(x-i) = \binom{x}{n} n! \\
x^{\overline{n}} &= (-1)^n (-x)^{\underline{n}}, x^{\underline{n}} = (-1)^n (-x)^{\overline{n}} \\
x^{\overline{n}} &= \sum_{i=0}^{n} \begin{bmatrix} n \\ i \end{bmatrix} x^i, x^{\underline{n}} = \sum_{i=0}^{n} (-1)^{n-i} \begin{bmatrix} n \\ i \end{bmatrix} x^i \\
x^n &= \sum_{i=0}^{n} \binom{x}{i} i! \begin{Bmatrix} n \\ i \end{Bmatrix} = \sum_{i=0}^{n} \begin{Bmatrix} n \\ i \end{Bmatrix} x^{\underline{i}} = \sum_{i=0}^{n} (-1)^{n-i} \begin{Bmatrix} n \\ i \end{Bmatrix} x^{\overline{i}}
\end{aligned}
$$

6.11.2 斯特林反演

$$
\begin{aligned}
f_n &= \sum_{i=0}^{n} \begin{Bmatrix} n \\ i \end{Bmatrix} g_i \Leftrightarrow g_n = \sum_{i=0}^{n} (-1)^{n-i} \begin{bmatrix} n \\ i \end{bmatrix} f_i \\
f_n &= \sum_{i=0}^{n} \begin{bmatrix} n \\ i \end{bmatrix} g_i \Leftrightarrow g_n = \sum_{i=0}^{n} (-1)^{n-i} \begin{Bmatrix} n \\ i \end{Bmatrix} f_i
\end{aligned}
$$

6.12 贝尔数

B_n 表示 n 个有标号物品分成若干个非空无标号集合的方案数。

$$
\begin{aligned}
B_n &= \sum_{i=0}^{n} \begin{Bmatrix} n \\ i \end{Bmatrix} = \sum_{i=0}^{n-1} \binom{n-1}{i} B_i \\
\frac{B_n}{(n-1)!} &= \sum_{i=0}^{n-1} \frac{1}{(n-1-i)!} \frac{B_i}{i!} \\
\hat{B}(x) e^x &= \sum_{i=0}^{\infty} B_n \frac{x^{n-1}}{(n-1)!} = \hat{B}'(x), \hat{B}(x) = \exp(e^x - 1)
\end{aligned}
$$

7 数论

7.1 BSGS

```
LL gcd(LL a,LL b) {return b?gcd(b,a%b):a;}
LL exgcd(LL a,LL b,LL &x,LL &y){
    if (!b) {x=1;y=0;return a;}
    LL r=exgcd(b,a%b,y,x);y-=a/b*x;
    return r;
}
LL Inv(LL A,LL C){
    LL x,y;exgcd(A,C,x,y);
    return (x%C+C)%C;
}
struct Hashmap{
    static const int HA=99991,maxe=46341;
    int E,lnk[HA],nxt[maxe+5];LL to[maxe+5],w[maxe+5];
    int top,stk[maxe+5];
    inline void Add(int x,LL y) {stk[++top]=x;to[++E]=y;w[E]=0;nxt[E]=lnk[x];lnk[x]=E;}
    void clear() {E=0;while (top) lnk[stk[top--]]=0;}
    bool count(LL x){
        int ha=x%HA;
        for (int j=lnk[ha];j;j=nxt[j])
            if (to[j]==x) return true;
        return false;
    }
    LL& operator [] (LL x){
        int ha=x%HA;
        for (int j=lnk[ha];j;j=nxt[j])
            if (to[j]==x) return w[j];
        Add(ha,x);return w[E];
    }
};
LL BSGS(LL A,LL B,LL C){ // A^x=B(mod C)
    static Hashmap f;
    A%=C;B%=C;
    if (!A && B) return B==1?0:-1; // if 0^0=1
    LL m=sqrt(C)+1,pw=1,INV=Inv(A,C),ipw=1,w=1;
    f.clear();
    for (int i=0;i<m;i++){
        LL now=ipw*B%C;
        if (!f.count(now)) f[now]=i;
        pw=pw*A%C;ipw=ipw*INV%C;
    }
    for (int i=0;i<m;i++,w=w*pw%C) if (f.count(w)) return i*m+f[w];
    return -1;
}
LL exBSGS(LL A,LL B,LL C){ // extend A^x=B(mod C)
    static Hashmap f;
    A%=C;B%=C;
    if (!A && B) return B==1?0:-1; // if 0^0=1
    LL pw=1%C;
```

```
    for (int i=0;i<30;i++,pw=pw*A%C) if (pw==B) return i;
    LL r,D=1,num=0;
    while ((r=gcd(A,C))>1){
        if (B%r) return -1;num++;
        B/=r;C/=r;D=(D*A/r)%C;
    }
    A%=C;
    LL m=sqrt(C)+1,INV=Inv(A,C),ipw=1;pw=1;
    f.clear();
    for (int i=0;i<m;i++){
        LL now=ipw*B%C;
        if (!f.count(now)) f[now]=i;
        pw=pw*A%C;ipw=ipw*INV%C;
    }
    for (int i=0;i<m;i++,D=D*pw%C) if (f.count(D)) return i*m+f[D]+num;
    return -1;
}
```

7.2 CRT

我们观察两个模方程：$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \end{cases}$ 。

转化：$x = a_1 + x_1 m_1 = a_2 + x_2 m_2 \Leftrightarrow x_2 m_2 \equiv a_1 - a_2 \pmod{m1}$ 。

一般题目里都要我们求最小非负整数解，所以我们可以用扩展欧几里得求出最小的 x_2 ，从而得出满足两个模方程的最小 $x = a_2 + x_2 m_2$ ，记为 x_0 。

因为要同时满足两个模方程，所以得出新的模方程：$x \equiv x_0 \pmod{[m_1, m_2]}$ 。

不断合并模方程，最后的 x_0 就是答案。

```
inline LL MUL(LL x,LL y,LL MOD) {return ((x*y-(LL)((DB)x/MOD*y)*MOD)%MOD+MOD)%MOD;}
LL gcd(LL a,LL b) {return b?gcd(b,a%b):a;}
LL exgcd(LL a,LL b,LL &x,LL &y){
    if (!b) return x=1,y=0,a;
    LL r=exgcd(b,a%b,y,x);y-=a/b*x;
    return r;
}
LL Solve(LL A,LL B,LL C){ // Ax=B(mod C)
    if (B<0) B+=C;LL x,y,r=exgcd(A,C,x,y);if (B%r) return -1;
    C/=r;x%=C;if (x<0) x+=C;x=MUL(B/r%C,x,C);return x;
}
#define lcm(A,B) ((A)/gcd((A),(B))*(B))
LL China(int n,LL *a,LL *m){ // x=a(mod m)
    LL A=a[1],M=m[1];
    for (int i=2;i<=n;i++){
        LL now=Solve(m[i],A-a[i],M);if (now<0) return -1;
        M=lcm(M,m[i]);A=MUL(now,m[i],M)+a[i];if (A>=M) A-=M;
    }
    return A;
}
```

7.3 二次剩余

求解 $x^2 \equiv a \pmod P$ 。

两边开根 $2\log x = \log a \Leftrightarrow \log x = \frac{\log a}{2}$ 。

求出 P 的原根 g ，BSGS 算出 $g^{\log a} \equiv a \pmod P$ ，则 $x \equiv g^{\frac{\log a}{2}} \pmod P$ 。

注意特判 $a = 0$ 。

虽然但是，建议直接看 Claris 的二次剩余板子。

```
int getg(int MOD){
    for (int i=2;i<MOD;i++){
        for (int j=1;j<=p[0] && (LL)p[j]*p[j]<=MOD;j++)
            if (!((MOD-1)%p[j])) if (Pow(i,(MOD-1)/p[j])==1) goto GG;
        return i;GG:continue;
    }
    return -1;
}
int Sqrt(int a){
    if (!a) return 0;
    int k=BSGS(g,a,MOD);
    return k<0 || (k&1)?-1:Pow(g,k>>1);
}
```

7.4 Pollard-Rho

```
const int maxp=10000,prime[]={2,3,5,7,11,13,17,19,23,29,31,37};

int P;LL p[maxp+5];
mt19937_64 mrand(19260817);

inline ULL ADD(ULL x,ULL y,ULL MOD) {return x+y>=MOD?x+y-MOD:x+y;}
inline ULL MUL(ULL x,ULL y,ULL MOD){
    LL s=x*y-(ULL)((DB)1/MOD*x*y)*MOD;
    return s<0?s+=MOD:(s>=MOD?s-=MOD:s);
}
LL Pow(LL w,LL b,LL MOD) {
    LL s;for (s=1;b;b>>=1,w=MUL(w,w,MOD)) if (b&1) s=MUL(s,w,MOD);return s;
}
bool check(LL p,LL n){
    int k=0;LL d=n-1,x,s;
    while (d&1^1) d>>=1,k++;
    x=Pow(p,d,n);
    for (;k;k--,x=s){
        s=MUL(x,x,n);
        if (s==1) return x==1 || x==n-1;
    }
    return false;
}
bool MR(LL n){
    for (int t=0;t<12;t++){
        if (n==prime[t]) return true;
        if (!(n%prime[t])) return false;
```

```
        if (!check(prime[t],n)) return false;
    }
    return true;
}
#define absi(x) ((x)<0?-(x):(x))
#define ctz __builtin_ctzll
LL gcd(LL a,LL b){
    if (!a) return b;if (!b) return a;
    int t=ctz(a|b);a>>=ctz(a);
    do {b>>=ctz(b);if (a>b) swap(a,b);b-=a;} while (b);
    return a<<t;
}
#define stp(x) (ADD(MUL(x,x,n),r,n))
LL FindD(LL n){
    if (n&1^1) return 2;
    static const int S=128;
    LL x=mrand()%n,r=mrand()%(n-1)+1;
    for (int l=1;;l<<=1){
        LL y=x,p=1;
        for (int i=0;i<l;i++) x=stp(x);
        for (int i=0;i<l;i+=S){
            LL z=x;
            for (int j=0;j<S && j<l-i;j++) x=stp(x),p=MUL(p,x+n-y,n);
            p=gcd(p,n);
            if (p==1) continue;if (p<n) return p;
            for (p=1,x=z;p==1;) x=stp(x),p=gcd(x+n-y,n);
            return p;
        }
    }
}
void PR(LL n){
    if (n==1) return;if (MR(n)) {p[++P]=n;return;}
    LL D;for (D=FindD(n);D==n;D=FindD(n));PR(D);PR(n/D);
}
// use hint: P=0;PR(n);
```

7.5 莫比乌斯反演

7.5.1 因数形式

$$f=g*1, f(n)=\sum_{d|n} g(d) \iff g=f*\mu, g(n)=\sum_{d|n} f(d)\mu(\frac{n}{d})$$

7.5.2 倍数形式

$$F(d)=\sum_{d|n} f(n)=\sum_{k=1}^{\infty} f(kd) \iff f(d)=\sum_{d|n} \mu(\frac{n}{d})F(n)=\sum_{k=1}^{\infty} \mu(k)F(kd)$$

证明：$f(d)=\sum_{i=1}^{\infty}\mu(i)\sum_{j=1}^{\infty}f(ijd)=\sum_{k=1}^{\infty}f(kd)\sum_{i|k}\mu(i)=\sum_{k=1}^{\infty}f(kd)e(k)=f(d)$ 。

7.6 杜教筛

$$\sum_{i=1}^{n}(f*g)(i)=\sum_{i=1}^{n}\sum_{d|i}f(\frac{i}{d})g(d)=\sum_{d=1}^{n}g(d)\sum_{i=1}^{\lfloor\frac{n}{d}\rfloor}f(i)=\sum_{i=1}^{n}g(1)S(\lfloor\frac{n}{i}\rfloor)$$

$$S(n)=\frac{\sum_{i=1}^{n}(f*g)(i)-\sum_{i=2}^{n}g(i)S(\lfloor\frac{n}{i}\rfloor)}{g(1)}$$

莫比乌斯函数 $\mu*1=e$：

$$S(n)=1-\sum_{i=2}^{n}S(\lfloor\frac{n}{i}\rfloor)$$

欧拉函数 $\varphi*1=id$：

$$S(n)=\frac{n(n+1)}{2}-\sum_{i=2}^{n}S(\lfloor\frac{n}{i}\rfloor)$$

欧拉函数乘上 $i^k, f(i)=i^k\varphi(i)$，$(f*id_k)(n)=n^k\sum_{d|n}\varphi(d)=n^{k+1}$：

$$S(n)=\sum_{i=1}^{n}i^{k+1}-\sum_{i=2}^{n}i^kS(\lfloor\frac{n}{i}\rfloor)$$

预处理前 $O(n^{\frac{2}{3}})$ 可加速。记忆化后，杜教筛外面套除法分块复杂度不变。

```
map<int,int> f;
int Sum(int n){
    if (n<=maxs) return sum[n];
    if (f.count(n)) return f[n];
    int ans= ... ;
    for (int l=2,r;l<=n;l=r+1)
        r=n/(n/l),ans-= ... ;
    ans/= ... ;
    return f[n]=ans;
}
```

7.7 Min25 筛

已知积性函数 $f(n)$，其中 $f(p)$ 形式友好，且 $f(p^k)$ 可以快速计算（p 是素数），求其前缀和。定义 $h(n)$ 表示把 n 强行当成素数代入 $f(n)$ 中的值，比如 $f(p)=p$ 那么 $h(n)=n$，且 $h(n)$ 是完全积性函数。

定义 $g_{i,j}$ 表示 $[1,i]$ 中是素数或者最小质因数 $>p_j$ 的 $h(n)$ 的和，初值 $g_{i,0}=\sum_{i=2}^{i}h(i)$，考虑递推：

1. $p_j^2>i$：$g_{i,j}=g_{i,j-1}$，因为最小质因数 $>p_j$，所以 $g_{i,j}=g_{i,j-1}$。
2. $p_j^2\le i$：$g_{i,j}=g_{i,j-1}-h(p_j)[g_{\lfloor\frac{i}{p_j}\rfloor,j-1}-\sum_{k=1}^{j-1}h(p_k)]$，$g_{i,j-1}$ 比 $g_{i,j}$ 多出来一部分即最小质因数为 p_j 的部分，又因为 $h(n)$ 是完全积性函数，所以先把 $h(p_j)$ 提出来，乘上最小质因数 $\ge p_j$ 的部分。

这样就可以得到 $\sum_{i=1}^{n}[i\in P]f(i)$ 即 $g_{n,|P|}$，实际只需要开一维数组，且只需要 $2\sqrt{n}$ 个（除法分块）。
定义 $S(a,b)$ 表示 $[1,a]$ 中最小质因数 $\ge p_b$ 的 $f(n)$ 的和，那么 $\sum_{i=1}^{n}f(i)$ 就是 $S(n,1)+f(1)$。
由于我们已经知道了素数的答案，只需要加上合数，枚举合数的最小质因子以及次数，则：

$$S(a,b)=g_a-\sum_{i=1}^{b-1}f(p_i)+\sum_{i=b}\sum_{k=1}S(\lfloor\frac{a}{p_i^k}\rfloor,i+1)f(p_i^k)+f(p_i^{k+1})$$

7.7.1 技巧

如果 $h(n)$ 不是积性函数，可以考虑拆分等方法，拆成若干个积性函数的和差。

例 1：$f(p)=c$ 常数，可以先求出 $f(p)=1$ 的 g 数组，然后全部乘上 c 。

例 2：$f(p)=p-1$ ，拆成 $f_1(p)=p, f_2(p)=1$ 。

如果 $f(n)$ 能通过枚举一个素数进行转移，则可能也可以 Min25 筛（因为 Min25 求和部分是通过枚举素数实现的）。

例 1：$f(n)$ 表示 n 的次大质因子（相同质数记多次），如果没有次大质因子记 $f(p)=0$ 。算 $S(a,b)$ 时，素数贡献均为 0 ，只需要考虑合数贡献，合数分为两种：$p_b^k p_c(c>b)$ 和 $p_b^k x$（x 是合数），因此还是可以枚举 p_b^k 然后计算。

```
const int maxs=2*sqrt(maxn);
int S,N,a[maxs+5],g[maxs+5],ID[2][maxs+5];
int p[maxs+5];bool pri[maxs+5];

void Make(){
    for (int i=2;i<=maxs;i++){
        if (!pri[i]) p[++p[0]]=i;
        for (int j=1,t;j<=p[0] && (t=i*p[j])<=maxs;j++)
            {pri[t]=true;if (!(i%p[j])) break;}
    }
    // Make sum of f(p) and f(p^k)
}
#define id(x) ((x)<=S?ID[0][x]:ID[1][N/(x)])
int Sum(int a,int b){
    if (a<1 || p[b]>a) return 0;
    int ans=g[id(a)]-(sum of f[1,b-1]);
    for (int i=b;i<=p[0] && (LL)p[i]*p[i]<=a;i++){
        LL A=p[i],B=p[i]*p[i];
        for (int k=1;B<=a;k++,A=B,B*=p[i])
            ans=ans+Sum(a/A,i+1)*f(p[i]^k)+f(p[i]^(k+1));
    }
    return ans;
}
int Min25(int n){
    int m=0;S=sqrt(n);N=n;
    for (int l=1,r;l<=n;l=r+1){
        r=n/(n/l);a[++m]=n/l;g[m]=(sum of h[2,a[m]]);
        a[m]<=S?ID[0][a[m]]=m:ID[1][n/a[m]]=m;
    }
    for (int j=1;j<=p[0] && p[j]<=S;j++)
        for (int i=1;i<=m && (LL)p[j]*p[j]<=a[i];i++)
            g[i]=g[i]+h(p[j])*(sum of h[p[1],p[j-1]])-h(p[j])*g[id(a[i]/p[j])];
    // Update g
    return Sum(n,1)+f(1);
}
```

7.8 线性筛

假设求积性函数 $f(x)$ 的质数幂 $f\left(p^k\right)$ 复杂度为 t，则最终复杂度为 $\mathcal{O}\left(n+\frac{n}{\ln n}\times t\right)$。
求的过程：p 直接走定义，p^k 利用公式复杂度为 t，其他合数直接用积性。

```
int mnd[N], mncnt[N], mnpw[N], prm[N], phi[N] = {0, 1};
for (int i = 2; i < N; ++i) {
    if (!mind[i]) {
        prm[++prm[0]] = mnd[i] = mnpw[i] = i; mncnt[i] = 1;
        phi[i] = i - 1; // cntd[i] = 2; // sumd[i] = 1 + i;
    }
    for (int j = 1, p = prm[1], n; j <= prm[0]; p = prm[++j]) {
        if ((n = i * p) > N) break;
        if (p == mind[i]) {
            mind[n] = p;
            pw[n] = pw[i] * p;
            cnt[n] = cnt[i] + 1;
            if (i == pw[i]) phi[n] = phi[i] * p;
                            // cntd[n] = cnt[n] + 1;
                            // sumd[n] = (pw[n] * p - 1) / (p - 1);
            else phi[n] = phi[i / pw[i]] * phi[pw[i] * p];
            break;
        }
        mnd[n] = pw[n] = p; cnt[n] = 1;
        phi[n] = phi[i] * phi[p];
    }
}
```

7.9 常用卷积恒等式

(1) $d=1*1$
(2) $\sigma_1=id\times 1$
(3) $id=\varphi*1$
(4) $\mu*1=e$
(5) $\mu*id=\varphi$
(6) $\sum_{i=1}^{n}[(i,n)=1]i=\dfrac{e(n)+n\varphi(n)}{2}$

7.10 除法分块

下取整分块

$x=\lfloor\frac{n}{l}\rfloor, r=\lfloor\frac{n}{x}\rfloor$

上取整分块

$x=\lceil\frac{n}{l}\rceil, r=\lfloor\frac{n-1}{x-1}\rfloor$
注意，当 $x=1$ 时，$r=+\infty$，而不是 $r=n$，查询信息时需要特别注意。

8 树论

8.1 Prufer 序列

编码

找到编号最小的度为 1 的节点，删除该点及连接该点的边，将与该点相邻的点加入序列。
重复删除/加入，直到剩下两个节点，最后的序列就是 Prufer 编码（长度为 $n-2$ ）。

解码

记录一个集合 S ， S 刚开始是全集。
令 x 是 S 中在当前 Prufer 编码中没出现的最小编号，将 x 与序列第一项连边，删除第一项以及 x 。
重复连边/删除，直到序列为空，此时 S 中还有两个节点，进行连边。

性质

如果一个节点的度数为 D ，那么该节点将会在 Prufer 编码中出现 $D-1$ 次。

8.2 树链合并

在线可以树剖 + 打覆盖标记线段树两个 log 。下面讲离线。
树上给 n 条到根的链，每条链将路径上点标记，求有几个标记点。
将 pos 按入栈序排序，把 pos_j 到根路径上的点都 $+1$ ，把 $LCA(pos_{j-1},pos_j)$ 到根路径上的点都 -1 。
多次树链合并，用差分 + 树状数组就可以单点查询。

```
sort(pos+1,pos+1+pos[0],cmp);Add(lt[pos[1]],1);
for (int i=2;i<=pos[0];i++) Add(lt[pos[i]],1),Add(lt[LCA(pos[i-1],pos[i])],-1);
```

8.3 直径合并

两棵树合并的时候，新树直径的两个端点只会在两棵树直径的四个端点中产生。

8.4 虚树

```
int vt[maxt+5],K,vn[maxt+5],top,stk[maxt+5];

#define Son(fa,x) (lt[fa]<=lt[x] && rt[x]<=rt[fa])
inline bool cmp(const int &i,const int &j) {return lt[i]<lt[j];}
void VT(){
    vn[++K]=ro;sort(vn+1,vn+1+K,cmp);
    int m=K;for (int i=2;i<=K;i++) vn[++m]=LCA(vn[i-1],vn[i]);
    sort(vn+1,vn+1+m,cmp);m=unique(vn+1,vn+1+m)-(vn+1);
    for (int i=1;i<=m;i++) /* Clear vt node */;
    top=0;
    for (int i=1;i<=m;i++){
        while (top && !Son(stk[top],vn[i])) top--;
        if (top) Add(vt,stk[top],vn[i]);
        stk[++top]=vn[i];
    }
}
```

8.5 树重心的性质

1. 以重心为根，所有的子树的大小都不超过整个树大小的一半。
2. 一棵树最多只有两个重心。
3. 树中所有点到某个点的距离和中，到重心的距离和是最小的，如果有多个重心，他们的距离和一样。
4. 把两棵树通过某一点相连得到一颗新的树，新的树的重心必然在连接原来两棵树重心的路径上，且这条路径上的点的最大子树在左右两个相邻的点上。
5. 一棵树添加或者删除一个节点，树的重心最多只移动一条边的位置。

8.6 点分治

模板：求树上距离为 d 的点对数 sum_d 。

```
int n,now[maxn+5],MAX;
int gr,S,si[maxn+5],ms[maxn+5];bool vis[maxn+5];

void getgr(int x,int pre=0){
    si[x]=1;ms[x]=0;
    for (int j=lnk[x],u;j;j=nxt[j])
        if ((u=to[j])!=pre && !vis[u]){
            getgr(u,x);
            si[x]+=si[u];ms[x]=max(ms[x],si[u]);
        }
    ms[x]=max(ms[x],S-si[x]);
    if (!gr || ms[x]<ms[gr]) gr=x;
}
void getdep(int x,int dep=0,int pre=0){
    now[++now[0]]=dep;if (dep>MAX) MAX=dep;
    for (int j=lnk[x],u;j;j=nxt[j])
        if ((u=to[j])!=pre && !vis[u]) getdep(u,dep+1,x);
}
void Count(int f){
    int t;for (t=1;t<=(MAX<<1);t<<=1);
    for (int i=0;i<t;i++) F[i]=0;
    for (int i=1;i<=now[0];i++) F[now[i]]++;
    NTT(F,t,1);
    for (int i=0;i<t;i++) F[i]=MUL(F[i],F[i]);
    NTT(F,t,-1);
    for (int i=0;i<t;i++) sum[i]+=f*F[i];
}
void Divide(int x){
    vis[x]=true;
    now[0]=0;MAX=0;getdep(x);Count(1);
    for (int j=lnk[x],u;j;j=nxt[j])
        if (!vis[u=to[j]]){
            now[0]=0;MAX=0;getdep(u,1);Count(-1);
            gr=0;S=si[u];getgr(u);Divide(gr);
        }
}

gr=0;S=n;getgr(1);Divide(gr);
```

8.7 点分树

将点分治 DFS 的顺序建成一棵新树，新树高是 $\log_2 n$ 的。假设 x 节点在点分树上子树大小为 $size_x$ ，则对所有 x 都可以记录 $O(size_x)$ 的信息（vector，线段树动态开点等），从而支持暴力信息修改。

点分树中通常会记录 A_x 表示 x 点分树子树中，与 x 的原树信息。以及 B_x 表示 x 点分树子树中，与点分树父亲 fa_x 的原树信息。以便进行容斥（和点分治同理）。

点分树中各种子树信息、距离信息基本和原树无关，如 x 到 fa_x 的距离不一定小于 x 到 fa_{fa_x} 的距离。

模板：求 x 附近距离小于等于 K 的点的权值和，带单点修改。

解法：维护 A_x 表示 x 点分树子树中，到 x 距离的权值线段树。B_x 表示 x 点分树子树中，到 fa_x 距离的权值线段树。首先查询 A_x 线段树中距离 $[0,K]$ 的权值，然后暴力枚举 x 的点分树祖先 y ，查询 A_{fa_y} 中 $[0,K-dis(x,fa_y)]$ 权值减去 B_y 中 $[0,K-dis(x,fa_y)]$ 权值。修改也是暴力枚举 x 的点分树祖先，修改 A_y 和 B_y 。

```
void Divide(int x,int pre=0){
    vis[x]=true;fa[x]=pre;
    for (int j=lnk[x],u;j;j=nxt[j])
        if (!vis[u=to[j]])
            gr=0,S=si[u],getgr(u),Divide(gr,x);
}
void Insert(int &p,int pos,int k,int l=0,int r=n-1){
    if (!p) p=++pl;sum[p]+=k;
    if (l==r) return;
    int mid=l+(r-l>>1);
    pos<=mid?Insert(ls[p],pos,k,l,mid):Insert(rs[p],pos,k,mid+1,r);
}
int Ask(int p,int L,int R,int l=0,int r=n-1){
    if (!p) return 0;
    if (L==l && r==R) return sum[p];
    int mid=l+(r-l>>1);
    if (R<=mid) return Ask(ls[p],L,R,l,mid);
    else if (L>mid) return Ask(rs[p],L,R,mid+1,r);
    else return Ask(ls[p],L,mid,l,mid)+Ask(rs[p],mid+1,R,mid+1,r);
}
void Update(int x,int y){
    for (int i=x;i;i=fa[i]){
        Insert(ro[i][0],Dis(x,i),y);
        if (fa[i]) Insert(ro[i][1],Dis(x,fa[i]),y);
    }
}
int Count(int x,int K){
    int ans=Ask(ro[x][0],0,K);
    for (int i=x;fa[i];i=fa[i]){
        int D=Dis(x,fa[i]);
        if (D<=K) ans+=Ask(ro[fa[i]][0],0,K-D)-Ask(ro[i][1],0,K-D);
    }
    return ans;
}
```

8.8 树链剖分

8.8.1 重链剖分

求 k 级祖先：跳到 k 级祖先所在长链，然后通过深度和 DFS 序确定 k 级祖先。

8.8.2 长链剖分

性质：x 的 k 级祖先所在长链长度一定 $\geq k$（否则显然有更长的链）。

求 k 级祖先：倍增预处理出 2^i 级祖先。先跳到 $2^{\lfloor \log_2 k \rfloor}$ 级祖先所在长链的链顶 y ，y 所在长链长度大于等于 $2^{\lfloor \log_2 k \rfloor}$ ，因此 y 沿着长链往下或者往上 len_y 个就能找到 k 级祖先，对于每个链顶可以预处理这 $2len_y$ 个节点。

有关深度的 DP：类似 $f_{i,j}$ 表示 i 子树中与 i 距离为 j ，距离为 $d-j$ 等 DP。
长链继承信息，短链暴力合并，复杂度 $O(n)$ 。

贪心：选 k 个叶子到根的路径，求覆盖的边权和的最大值。
带权长链剖分之后，显然分成了若干个不相交的，到叶子的路径，选前 k 大即可。

```cpp
int tem[maxt+5],*pl=tem,*f[maxn+5],*g[maxn+5],ans;

void DFS(int x,int pre=0){
    dep[x]=dep[pre]+1;md[x]=dep[x];
    for (int j=lnk[x];j;j=nxt[j])
        if (to[j]!=pre){
            DFS(to[j],x);
            if (md[to[j]]>md[x]) md[x]=md[to[j]],SH[x]=to[j];
        }
    len[x]=len[SH[x]]+1;
}
void LSD(int x,int lst){
    top[x]=lst;
    if (SH[x]) LSD(SH[x],lst);
    for (int j=lnk[x];j;j=nxt[j])
        if (to[j]!=SH[x]) LSD(to[j],to[j]);
}
int getanc(int x,int k){
    if (!k) return x;
    int t=lg[k];
    x=ST[t][x];k-=1<<t;k-=dep[x]-dep[top[x]];x=top[x];
    if (!k) return x;
    return k>0?U[x][k-1]:D[x][-k-1];
}
void DP(int x,int pre=0,bool fl=true){
    if (fl) f[x]=pl,pl+=len[x],g[x]=pl+len[x]-1,pl+=len[x]<<1;
    if (SH[x]) f[SH[x]]=f[x]+1,g[SH[x]]=g[x]-1,DP(SH[x],x,false);
    f[x][0]++;
    for (int j=lnk[x],u;j;j=nxt[j])
        if ((u=to[j])!=pre && u!=SH[x]){
            DP(u,x,true);
            for (int j=0;j<len[u];j++){
                ans+=f[u][j]*g[x][j+1];
                if (j>1) ans+=f[x][j-1]*g[u][j];
```

```
            }
            for (int j=0;j<len[u];j++){
                g[x][j+1]+=f[x][j+1]*f[u][j];
                f[x][j+1]+=f[u][j];
                if (j) g[x][j-1]+=g[u][j];
            }
        }
    ans+=g[x][0];
}

DFS(1);LSD(1,1);
for (int i=1;i<=n;i++)
    if (i==top[i]){
        U[i]=pl;pl+=len[i];D[i]=pl;pl+=len[i];
        for (int j=0,x=i,y=i;j<len[i];j++)
            x=fa[x],U[i][j]=x,y=SH[y],D[i][j]=y;
    }
```

8.9 链分治

8.9.1 重链分治

将根节点所在重链视为第一层，第一层点连出去的轻链一定是另一个重链的顶点，作为第二层重链。以此类推，由于轻儿子 $size$ 至少减半，因此这样分层只有 $\log_2 n$ 层。

常用于处理子树 DP 的优化，例如动态最大独立集，独立集背包。

先求出轻链的信息，挂在重链节点上，然后对重链进行分治等方法快速合并信息。

```
void DFS(int x,int pre=0){
    si[x]=1;fa[x]=pre;
    for (int j=lnk[x];j;j=nxt[j])
        if (to[j]!=pre){
            DFS(to[j],x);si[x]+=si[to[j]];
            if (si[to[j]]>si[SH[x]]) SH[x]=to[j];
        }
}
void HLD(int x,int lst){
    lt[x]=++lt[0];top[x]=lst;btm[x]=x;
    if (SH[x]) HLD(SH[x],lst),btm[x]=btm[SH[x]];
    for (int j=lnk[x];j;j=nxt[j])
        if (to[j]!=fa[x] && to[j]!=SH[x]) HLD(to[j],to[j]);
}
Matrix operator * (const Matrix &a,const Matrix &b){
    static Matrix c;c.zero();
    for (int i=0;i<2;i++)
        for (int j=0;j<2;j++)
            for (int k=0;k<2;k++)
                c.s[i][j]=max(c.s[i][j],a.s[i][k]+b.s[k][j]);
    return c;
}
void Build(int x,bool fl=true){
    if (SH[x]) Build(SH[x],false);
    sum[x][1]=a[x];
```

```
    for (int j=lnk[x],u;j;j=nxt[j])
        if ((u=to[j])!=fa[x] && u!=SH[x]){
            Build(u,true);
            sum[x][0]+=max(f[u][0],f[u][1]);
            sum[x][1]+=f[u][0];
        }
    Update(lt[x],sum[x][0],sum[x][1]);
    if (fl){
        tem=Ask(lt[x],lt[btm[x]]);
        f[x][0]=tem.s[0][0];f[x][1]=tem.s[0][1];
    }
}
void Modify(int x,int y){ // Modify a[x] to y
    LL lst[2]={0,y-a[x]};a[x]=y;
    for (;x;x=fa[x]){
        Update(lt[x],lst[0],lst[1]);
        sum[x][0]+=lst[0];sum[x][1]+=lst[1];
        x=top[x];tem=Ask(lt[x],lt[btm[x]]);
        lst[0]=max(tem.s[0][0],tem.s[0][1])-max(f[x][0],f[x][1]);
        lst[1]=tem.s[0][0]-f[x][0];
        f[x][0]=tem.s[0][0];f[x][1]=tem.s[0][1];
    }
}

DFS(1);HLD(1,1);Build(1);
```

8.9.2 长链分治

同重链分治分层方式，由于上层比下层至少长 1 ，因此只有 $2\sqrt{n}$ 层。

问题：每个节点给出多项式 $a_i x + b_i$ ，求所有叶子到根多项式乘积的和。

每个节点记录长链上多项式，以及拐到短链之后的多项式的和。

类似重链分治，先求出短链的多项式，加到长链上，然后对长链进行分治 NTT，得到长链多项式，以及长链拐出去的多项式之和。

8.10 DSU on Tree

按子树 size 轻重链剖分，然后使用某个数据结构统计每个点的子树信息：

1. 先让轻儿子统计子树信息，并撤销对数据结构的影响；
2. 如果有重儿子，统计重儿子子树信息，保留对数据结构的影响（不撤销）；
3. 向数据结构中添加轻儿子子树信息和当前点信息，统计当前点信息。

复杂度分析：每个点只会在到根路径上遇到轻边时被添加 / 撤销，由轻重连剖分结论，每个点到根的路径上至多 $\log n$ 条轻边。所以总复杂度是 $\mathcal{O}(n \log n)$ ，由于明显跑不满所以常数会很小。

如果需要多次运行 DSU on Tree，可以把所有轻儿子的子树点集预处理出来，省掉递归的常数。

```
int sz[N], mxs[N];
vector<int> e[N];
void dfs(int u, int fa) {
    sz[u] = 1;
    for (auto v : e[u])
        if (v != fa) {
            dfs(v, u); sz[u] += sz[v];
            if (sz[v] > sz[mxs[u]]) mxs[u] = v;
        }
}
inline void upd(int u) {  /*adding information of u into data structures*/}
inline void del(int u) {/*deleting information of u from data structures*/}
void upd(int u, int fa) {upd(u); for (auto v : e[u]) if (v != fa) upd(v, u);}
void del(int u, int fa) {del(u); for (auto v : e[u]) if (v != fa) del(v, u);}
void dsu(int u, int fa) {
    for (auto v : e[u]) if (v != fa && v != mxs[u]) {dsu(v, u); del(v, u);}
    if (mxs[u]) dsu(mxs[u], u);
    for (auto v : e[u]) if (v != fa && v != mxs[u]) upd(v, u);
    upd(u);
}
// 预处理轻儿子子树部分
vector<int> subtree[N];
void dsu(int u, int fa) {
    for (auto v : e[u])
        if (v != fa && v != mxs[u]) {dsu(v, u); for (auto k : subtree[v]) del(k);}
    if (mxs[u]) dsu(mxs[u], u);
    for (auto v : e[u])
        if (v != fa && v != mxs[u]) for (auto k : subtree[v]) add(k);
    add(u);
}
void get_subtree(int u, int fa, int cur) {
    subtree[cur].pb(u);
    for (auto v : e[u]) if (v != fa) get_subtree(v, u, cur);
}
void dsu_tree(int u, int fa) {
    for (auto v : e[u]) if (v != fa && v != mxs[u]) get_subtree(v, u, v);
    for (auto v : e[u]) if (v != fa) dsu_tree(v, u);
}
dfs(1, 1); dsu_tree(1, 1); dsu(1, 1);
```

8.11 仙人掌

8.11.1 建图

DFS 树：在找到返祖边 $u \to x$ 时，建立一个方点，将 x 连向方点，然后从 u 暴力向上跳，将方点连向途中的点，并且记录环的总距离。求最短路：圆方边的边权是点到祖先点的最小距离（环上有两个方向），用环的总距离以及 DFS 树上的距离就可以算出。

询问 x, y 时，在圆方树上求出 x, y 的 lca ，然后讨论：

1. lca 是方点，先将 x, y 跳到 lca 的前一个节点 fx, fy ， x, y 走到 fx, fy 之后再求环上的最小距离
2. lca 是圆点，那么答案就是 x, y 在圆方树上的距离

```
void Cactus(int x,int pre=-1){
    dfn[x]=++dfn[0];
    for (int j=lnk[0][x],u;~j;j=nxt[j])
        if ((j^1)!=pre)
            if (!dfn[u=son[j]]) fa[u]=x,lst[u]=j,dis[u]=dis[x]+w[j],Cactus(u,j); else
            if (dfn[u]<dfn[x]){
                N++;Add(lnk[1],u,N,0);vis[j]=vis[j^1]=true;
                sum[N]=dis[x]-dis[u]+w[j];
                for (int i=x;i!=u;i=fa[i]){
                    Add(lnk[1],N,i,min(dis[i]-dis[u],sum[N]-(dis[i]-dis[u])));
                    vis[lst[i]]=vis[lst[i]^1]=true;
                }
            }
}
void DFS(int x,int fa=0){
    ST[x][0]=fa;dep[x]=dep[fa]+1;
    for (int j=1;j<=LOG;j++) ST[x][j]=ST[ST[x][j-1]][j-1];
    for (int j=lnk[1][x],u;~j;j=nxt[j])
        u=son[j],pre[u]=pre[x]+w[j],DFS(u,x);
}
int LCA(int &x,int &y){
    if (dep[x]<dep[y]) swap(x,y);
    for (int j=LOG;~j && dep[x]>dep[y];j--) if (dep[ST[x][j]]>=dep[y]) x=ST[x][j];
    if (x==y) return x;
    for (int j=LOG;~j;j--) if (ST[x][j]!=ST[y][j]) x=ST[x][j],y=ST[y][j];
    return ST[x][0];
}
int Dis(int x,int y){
    if (x==y) return 0;
    int fx=x,fy=y,lca=LCA(fx,fy);
    if (lca<=n) return pre[x]+pre[y]-(pre[lca]<<1);
    int ans=pre[x]-pre[fx]+pre[y]-pre[fy];
    if (dis[fx]>dis[fy]) swap(fx,fy);
    return ans+min(dis[fy]-dis[fx],sum[lca]-(dis[fy]-dis[fx]));
}
/* Build lnk[0] graph */
N=n;Cactus(1);
for (int i=1;i<=n;i++)
    for (int j=lnk[0][i];~j;j=nxt[j])
        if (!vis[j] && dfn[son[j]]>dfn[i]) Add(lnk[1],i,son[j],w[j]);
/* Make lnk[1] ST */
```

8.11.2 最大独立集

建出圆方树，定义 $f_{i,0/1}$ 表示 i 点是否选择的最大独立集，并且对于方点，将 $f_{i,0/1}$ 定义为与方点 i 的顶点相邻的节点是否选择的最大独立集。

那么圆点就正常进行最大独立集 DP，而方点需要进行进一步 DP。

由于 DFS 树建圆方树时是按照深度顺序建边的，因此对于方点，按照加边顺序获取出所有儿子即为环上的顺序。

定义 $F_{i,0/1,0/1}$ 表示前 i 个节点中，第一个节点是否选择，且第 i 个节点是否选择的最大独立集。

最后 $\max\{f_{1,0}, f_{1,1}\}$ 就是答案。

```
void DP(int x){
    int si=e[x].size();
    for (int j=0;j<si;j++) DP(e[x][j]);
    if (x>n){
        for (int i=0;i<2;i++)
            for (int j=0;j<2;j++)
                F[0][i][j]=(i==j?f[e[x][0]][i]:-1e9);
        for (int i=1;i<si;i++)
            for (int j=0;j<2;j++){
                F[i][j][0]=max(F[i-1][j][0],F[i-1][j][1])+f[e[x][i]][0];
                F[i][j][1]=F[i-1][j][0]+f[e[x][i]][1];
            }
        f[x][0]=F[si-1][0][0];
        f[x][1]=-1e9;
        for (int i=0;i<2;i++)
            for (int j=0;j<2;j++)
                if (i+j) f[x][1]=max(f[x][1],F[si-1][i][j]);
    } else {
        for (int j=0;j<si;j++) f[x][1]+=f[e[x][j]][0];f[x][1]++;
        for (int j=0;j<si;j++) f[x][0]+=max(f[e[x][j]][0],f[e[x][j]][1]);
    }
}
```

9 图论

9.1 差分约束系统

$a_y - a_x \le b$ 转化为 $x \to y$ 边长为 b 的边，然后刷最短路，如果从 S 开始，则 $dis_i - S$ 都是最大的（最短路顶到限制）。
同理 $a_y - a_x \ge b$ 转化为 $x \to y$ 边长为 b 的边，刷最长路，如果从 S 开始，则 $dis_i - S$ 都是最小的。

9.2 K 短路

每个点的代价函数为 $f(x) + h(x)$ ，其中 $f(x)$ 为 S 到 x 距离，$h(x)$ 为 x 到 T 距离（完美估价函数）。
使用 A*，优先队列存 $f(x) + h(x)$ 最小的 x ，每一次 x 抵达 T 就说明是比上一次劣的最优最短路。

9.3 二分图

9.3.1 生成树个数

左 n 个点，右 m 个点的完全二分图，生成树个数为 $m^{n-1} \cdot n^{m-1}$ 。

9.3.2 霍尔定理

二分图 X, Y 存在完美匹配等价于：
对于 X 的任意子集 S ，定义 $M(S)$ 表示 S 连向的 Y 的点集，均满足 $|S| \le |M(S)|$ 。

9.3.3 稳定婚姻

有 n 对 CP，和 m 对前男女友关系。一对 CP 解散之后双方可能会和前男女友旧情复燃，导致很多 CP 都解散。问第 i 对 CP 解散之后还能否使得所有人都找到新 CP。
如果第 i 对 CP 除了直接相连的边之外还能找到另外的增广路，说明就可以 CP 重组。那么把 CP 边建成 (x, y) ，而前男女友关系建成 (y, x) ，有多条增广路其实就是有 SCC，Tarjan 就好了。
不能双向边刷边双，这样可能会把前男女友与 CP 搞混。

9.3.4 König 定理

二分图最大匹配大小 = 最小点覆盖大小。
构造最小点覆盖：假设此前求最大匹配是每次从左侧开始尝试增广，那么增广结束之后，图上没有增广路了。接下来，从左侧每一个未匹配点开始走交错路（匹配边，未匹配边间隔放置），经过的点都打上标记，复杂度 $\mathcal{O}(m)$。最小点覆盖就是**所有左侧未标记点和右侧标记点**。
任何图最大独立集和最小匹配为对偶问题（答案互补）。

9.3.5 Dilworth 定理

对于偏序关系，最长反链（最大不可比较集）大小 = 最小链（可比较集）覆盖大小（链数）。
对偶定理成立：最小反链覆盖大小 = 最长链大小。

求最小链覆盖：先将偏序的 DAG 传递闭包，然后问题变成求解最小不可重路径覆盖。只需要将每个点拆成出点和入点，然后对于原图的边连 $u \to v$ ，答案即 $|V| -$ 最大匹配。

求最长反链：按上述构造二分图并求最大独立集，如果一个点的出入点都在其中，将其加入最长反链。

9.3.6 Hopcroft-Karp

Dinic 在二分图上的特殊化，求最大匹配复杂度 $\mathcal{O}(m\sqrt{n})$ ，实际很难跑满，常数非常小。
使用时记得设置左右点数 nl 和 nr ，以及增广路长度上限 inf 。

```cpp
template <const int V, const int inf = 1000000000>
struct Hopcroft_Karp {
    bool vis[V];
    vector<int> e[V];
    int nl, nr, ml[V], mr[V], dl[V], dr[V]; // m for match, d for distance
    inline bool bfs() {
        static int q[V], hd, tl, dT; hd = 1; tl = 0; dT = inf;
        memset(dl, -1, sizeof(int) * (nl + 1));
        memset(dr, -1, sizeof(int) * (nr + 1));
        for (int i = 1; i <= nl; ++i) if (!ml[i]) {dl[i] = 0; q[++tl] = i;}
        for (int u; hd <= tl;) {
            if (dl[u = q[hd++]] >= dT) break;
            for (auto v : e[u])
                if (dr[v] == -1) {
                    dr[v] = dl[u] + 1;
                    if (!mr[v]) getmin(dT, dr[v] + 1);
                    else {dl[mr[v]] = dr[v] + 1; q[++tl] = mr[v];}
                }
        }
        return dT != inf;
    }
    bool dfs(int u) {
        for (auto v : e[u]) {
            if (vis[v] || dl[u] + 1 != dr[v]) continue;
            vis[v] = true;
            if (!mr[v] || dfs(mr[v])) {mr[v] = u; ml[u] = v; return true;}
        }
        return false;
    }
    inline void add(int u, int v) {e[u].push_back(v);}
    inline int max_matching() {
        int ans = 0;
        while(bfs()) {
            memset(vis, 0, sizeof(bool) * (nr + 1));
            for (int i = 1; i <= nl; ++i) if (!ml[i]) ans += dfs(i);
        }
        return ans;
    }
    bool visl[V], visr[V], anti[V];
    void addtag(int u) {
        visl[u] = true;
        for (auto v : e[u]) if (!visr[v]){visr[v] = true; addtag(mr[v]);}
    }
    void construct() {
        for (int i = 1; i <= nl; ++i) if (!ml[i]) addtag(i);
        // vertex cover:    visl[i] = false || visr[i] = true
        // independent set: visl[i] = true  || visr[i] = false
        // antichain:       visl[i] = true  && visr[i] = false
    }
};
```

9.4 强连通分量

9.4.1 Kosaraju's Algorithm

DFS1：DFS **原图**记录后序 post order，post number 最大的点一定在反图的某个汇中。
DFS2：按 DFS1 中 post order 的**逆序** DFS **反图**，每次找到一个反图的汇 SCC，从小到大编号。

```cpp
bool vis[N];
int scc, bl[N];
vector<int> e[N], re[N], nodes[N], order;
void add(int u, int v) {e[u].push_back(v); re[v].push_back(u);}
void dfs1(int u) {
    vis[u] = 1;
    for (auto v : e[u]) if (!vis[v]) dfs1(v);
    order.push_back(u);
}
void dfs2(int u) {
    nodes[bl[u] = scc].push_back(u);
    for (auto v : re[u]) if (!bl[v]) dfs2(v);
}
void kosaraju(int n) {
    for (int u = 1; u <= n; ++u) if (!vis[u]) dfs1(u);
    reverse(order.begin(), order.end());
    for (auto u : order) if (!bl[u]) {++scc; dfs2(u);}
}
```

9.4.2 Tarjan's SCC Algorithm

$low[u]$：节点 u 能到达的，且能走回来的最小的 idx（强制在栈中）。
在**反图**上求 SCC 并缩点，则编号 $bl[u]$ 同原图拓扑序[1]，即：若原图中 $u \to v$，则 $bl[u] \le bl[v]$。

```cpp
bool vis[N], instack[N];
int idxcnt, idx[N], low[N], top, stk[N], scc, bl[N];
void dfs(int u) {
    low[u] = idx[u] = ++idxcnt;
    instack[stk[++top] = u] = true;
    for (auto v : re[u]) // on reversed graph
        if (!idx[v]) {dfs(v); low[u] = min(low[u], low[v]);}
        else if (instack[v])  low[u] = min(low[u], idx[v]);
    if (low[u] == idx[u]) { // find a scc with root u
        ++scc;
        do {
            instack[stk[top]] = false;
            nodes[bl[stk[top]] = scc].push_back(stk[top]);
        } while (stk[top--] != u);
    }
}
void tarjan(int n) {
    for (int i = 1; i <= n; ++i) if (!idx[i]) dfs(i);
}
```

[1]证明：考虑原图中两个强连通分量 S_1 和 S_2，若 S_1 中某个点有一条指向 S_2 中某个点的边，讨论反图中 S_1 和 S_2 的 DFS 顺序即可证明

9.4.3 缩点（去重边）

任何有向图都是其强连通分量（SCC ）的有向无环图（DAG ）。

按上述代码编号，则编号 $bl[u]$ 从小到大即原图拓扑序：若原图中 $u \to v$ ，则 $bl[u] \le bl[v]$ 。

```
int indeg[N], outdeg[N];
vector<int> source, sink;
unordered_map<ll, bool> valid;
inline bool has_edge(int u, int v) {
    ll w = 10000000000ll * u + v;
    return valid[w] ? true : (valid[w] = true, 0);
}
inline void shrink(int n) {
    valid.clear();
    for (int u = 1; u <= n; ++u) e[u].clear();
    for (int v = 1; v <= n; ++v)
        for (auto u : re[v])
            if (bl[u] != bl[v] && !has_edge(bl[u], bl[v])) {
                ++indeg[bl[v]]; ++outdeg[bl[u]]; e[bl[u]].push_back(bl[v]);
            }
    for (int u = 1; u <= scc; ++u) if (!indeg[u]) source.push_back(u);
    for (int u = 1; u <= scc; ++u) if (!outdeg[u]) sink.push_back(u);
    for (int u = 1; u <= n; ++u) re[u].clear();
    for (int u = 1; u <= scc; ++u) for (auto v : e[u]) re[v].push_back(u);
}
```

9.4.4 构造强连通图方法

首先，收缩 SCC. 假设有 s 个源 (入度为 0 的点) $v_0, \ldots, v_{s-1}$ 和 t 个汇（出度为 0 的点) $w_0, \ldots, w_{t-1}$. 不失一般性，假设 $s \le t$. 我们之后可以证明，$v_0, \ldots, v_{s-1}$ 和 $w_0, \ldots, w_{t-1}$ 可以重新排列，使得存在一个 $1 \le p \le s$ ，满足以下三个条件：

1. 对于 $i \in [0, p), v_i \rightsquigarrow w_i$.
2. 对于 $i \in [p, s)$, 存在 $j \in [0, p), v_i \rightsquigarrow w_j$.
3. 对于 $j \in [p, t)$, 存在 $i \in [0, p), v_i \rightsquigarrow w_j$.

其中，$u \rightsquigarrow v$ 代表存在一条从 u 到 v 的**路径**。

现在先假设满足上述性质的 p 存在。连下列 t 条边:

1. 对于 $i \in [0, p)$ ，连 $w_i \to v_{(i+1) \bmod p}$. 这样可以把 $v_0, \ldots, v_{p-1}, w_0, \ldots, w_{p-1}$ 变成一个环。
2. 对于 $i \in [p, s)$ ，连 $w_i \to v_i$. 因为后两个性质，这样可以把 v_i 和 w_i 加入前 p 对点组成的环。
3. 对于 $i \in [s, t)$ ，连 $w_i \to v_0$. 因为最后一个性质，这样可以把 w_i 加入环。

下面证明如何构造出一组满足条件的排列和对应的 p.

我们考虑收缩完 SCC 的图，建立一个超级源 S ，向 $v_0, \ldots, v_{s-1}$ 连边. 建立一个超级汇 T ，从 $w_0, \ldots, w_{t-1}$ 连边。

接下来，从 S 到 T 跑 Dinic 算法的**一次迭代**，换言之，求一个阻塞流。

假设这个阻塞流大小是 f, 分别是从 $a_0 \rightsquigarrow b_0, \ldots, a_{f-1} \rightsquigarrow b_{f-1}$. 那么我们令 $p = f$. 那么第一个条件已经满足了。我们同时可以注意到，对于一条阻塞的边 $p \to q$. 那么存在一个阻塞的源点到 q, 从 p 也可以到达一个阻塞的汇点。

对于一个非阻塞的源点，它肯定能到达一条阻塞的边，从而可以到达一个阻塞的汇点。非阻塞的汇点情况也是一样。于是就满足了后两个条件。

当然需要额外注意的是孤立的点，但是我们可以强行把这个点拆成两个点，转化为前面的问题。

9.5 2-SAT

9.5.1 建图

n 个节点，每个节点非 0 即 1 ，i_0 表示取 0 ，i_1 表示取 1 。

选了 i_x 必须选 j_y ： $i_x \to j_y, j_{1-y} \to i_{1-x}$ 。

选了 i_x 不能选 j_y ： $i_x \to j_{1-y}, j_y \to i_{1-x}$ 。

i 必须为 x ： $i_x \to i_{1-x}$ 。

```
// Brute force
bool DFS(int x){
    if (vis[x]) return true;
    if (vis[x^1]) return false;
    vis[x]=true;stk[++top]=x;
    for (int j=lnk[x];j;j=nxt[j])
        if (!DFS(to[j])) return false;
    return true;
}
bool Solve(){
    for (int i=0;i<(n<<1);i++) vis[i]=false;
    for (int i=0;i<(n<<1);i+=2)
        if (!vis[i] && !vis[i^1]){
            top=0;if (DFS(i)) continue;
            while (top) vis[stk[top--]]=false;
            if (!DFS(i^1)) return false;
        }
    return true;
}
// Tarjan
void Tarjan(int x){
    dfn[x]=low[x]=++ti;stk[++top]=x;instk[x]=true;
    for (int j=lnk[x];j;j=nxt[j])
        if (!dfn[to[j]]) Tarjan(to[j]),low[x]=min(low[x],low[to[j]]); else
        if (instk[to[j]]) low[x]=min(low[x],dfn[to[j]]);
    if (dfn[x]==low[x]){
        SCC[x]=++cnt;instk[x]=false;
        for (int y=stk[top--];y!=x;y=stk[top--]) SCC[y]=cnt,instk[y]=false;
    }
}
/* Build graph */
for (int i=0;i<(n<<1);i++) if (!dfn[i]) Tarjan(i);
for (int i=0;i<(n<<1);i+=2)
    if (SCC[i]==SCC[i^1]) goto GG;
    else ans[i>>1]=(SCC[i]<SCC[i^1]?0:1);
```

9.5.2 前后缀优化建图

n 个组，第 i 组 K_i 个点，每个组只能选一个。

对于单组，每个前缀新加一个点 pre_i ，表示前 i 个点是否选了：

1. pre_{i-1} 选了必须选 pre_i ， pre_i 没选不能选 pre_{i-1}
2. i 选了必须选 pre_i ， pre_i 没选不能选 i
3. i 选了不能选 pre_{i-1} ， pre_{i-1} 选了不能选 i

9.6 圆方树（点双）

```
void Tarjan(int x,int pre=-1){
    dfn[x]=low[x]=++ti;stk[++top]=x;
    for (int j=lnk[0][x],u;~j;j=nxt[j])
        if ((j^1)!=pre)
            if (!dfn[u=son[j]]){
                Tarjan(u,j);low[x]=min(low[x],low[u]);
                if (low[u]>=dfn[x]){
                    N++;Add(1,x,N);
                    for (int y=stk[top--];;y=stk[top--]) {Add(1,N,y);if (y==u) break;}
                }
            } else low[x]=min(low[x],dfn[u]);
}
```

9.7 带花树

求解一般无向图的最大匹配，复杂度 $O(n^3+nm)$ ，稀疏图跑得飞快。

```
int LCA(int x,int y){
    x=fa[x];y=fa[y];
    for (ti++;vis[x]<ti;swap(x,y)) if (x) vis[x]=ti,x=fa[pre[who[x]]];
    return x;
}
void Blossom(int x,int y,int lca){
    while (fa[x]!=lca){
        pre[x]=y;y=who[x];
        if (tp[y]) tp[y]^=1,que[++Tail]=y;
        fa[x]=fa[y]=lca;x=pre[y];
    }
}
bool Find(int x){
    for (int i=1;i<=n;i++) fa[i]=i,tp[i]=-1,pre[i]=0;
    Head=0;Tail=0;que[++Tail]=x;tp[x]=0;
    while (Head!=Tail){
        int x=que[++Head];
        for (int j=lnk[x],u;j;j=nxt[j])
            if (tp[u=son[j]]<0){
                tp[u]=1;pre[u]=x;
                if (!who[u]){
                    for (int p=u,lst;p;p=lst)
                        lst=who[pre[p]],who[p]=pre[p],who[pre[p]]=p;
                    return true;
                }
                tp[who[u]]=0;que[++Tail]=who[u];
            } else if (tp[u]==0){
                int lca=LCA(x,u);Blossom(x,u,lca);Blossom(u,x,lca);
            }
    }
    return false;
}
for (int i=1,x,y;i<=m;i++) scanf("%d%d",&x,&y),Add(x,y),Add(y,x);
for (int i=1;i<=n;i++) if (!who[i]) ans+=Find(i);
```

9.8 三元环

无向图，度数小的节点连向度数大的节点（度数相同比编号），然后暴力枚举。
复杂度 $O(m\sqrt{m})$ ，三元环最多 $O(m\sqrt{m})$ 个。

```
inline bool cmp(const int &x,const int &y) {return d[x]<d[y] || d[x]==d[y] && x<y;}

for (int i=1;i<=m;i++) scanf("%d%d",&X[i],&Y[i]),d[X[i]]++,d[Y[i]]++,num[i]=0;
for (int i=1;i<=m;i++) cmp(X[i],Y[i])?Add(X[i],Y[i],i):Add(Y[i],X[i],i);
for (int i=1;i<=m;i++){
    ti++;for (int j=lnk[X[i]];j;j=nxt[j]) ID[to[j]]=w[j],vis[to[j]]=ti;
    for (int j=lnk[Y[i]];j;j=nxt[j])
        if (vis[to[j]]==ti) cnt[w[j]]++,cnt[ID[to[j]]]++,cnt[i]++,ans++;
}
/*use hint
ans is the number of three-circle
cnt[i] is the number of three-circle using edge i
*/
```

9.9 四元环

无向图，度数小的节点连向度数大的节点，枚举 u 双向边连向的节点 v ，枚举 v 单向边连向的节点 z ，如果 u 度数小于 z 则统计 z 之前出现次数，并累加 z 出现次数（z 是度数最大点，避免计数重复）。
复杂度 $O(m\sqrt{m})$ ，四元环可能很多。

```
inline bool cmp(const int &x,const int &y) {return d[x]<d[y] || d[x]==d[y] && x<y;}

for (int i=1;i<=m;i++){
    scanf("%d%d",&X[i],&Y[i]);
    d[X[i]]++;d[Y[i]]++;
    e[X[i]].push_back(Y[i]);
    e[Y[i]].push_back(X[i]);
}
for (int i=1;i<=m;i++) cmp(X[i],Y[i])?Add(X[i],Y[i]):Add(Y[i],X[i]);
for (int x=1;x<=n;x++){
    for (auto y:e[x])
        for (int j=lnk[y];j;j=nxt[j])
            if (cmp(x,to[j])) ans+=cnt[to[j]],cnt[to[j]]++;
    for (auto y:e[x])
        for (int j=lnk[y];j;j=nxt[j])
            cnt[to[j]]=0;
}
```

9.10 一般图最小割

n 个点，m 条边的无向图，问最小的边权和使得删掉这些边之后把图分成不连通的两部分。
复杂度 $O(n^3+nm)$ ，如果把找最大值改成堆就可以做到 $O(n^2+nm\log_2 m)$ 。

```
int getfa(int x) {return x==fat[x]?x:fat[x]=getfa(fat[x]);}
int Mincut(int te,int &s,int &t){
    val[0]=-2e9;
    for (int i=1;i<=n;i++) val[i]=0,vis[i]=false;
    for (t=fat[1];te;te--){
        s=t;vis[s]=true;
        for (auto x:e[s]) if (!vis[fat[x.fr]]) val[fat[x.fr]]+=x.sc;
        t=0;
        for (int i=1;i<=n;i++) if (!vis[fat[i]] && val[fat[i]]>val[t]) t=fat[i];
        if (!t) return 0;
    }
    return val[t];
}
void Merge(int s,int t){
    fat[t]=s;for (int i=1;i<=n;i++) getfa(i);
    for (int i=1;i<=n;i++) val[i]=0;
    for (auto x:e[s]) val[fat[x.fr]]+=x.sc;
    for (auto x:e[t]) val[fat[x.fr]]+=x.sc;
    e[s].clear();
    for (int i=1;i<=n;i++) if (val[i]) e[s].push_back(mp(i,val[i]));
}
int SW(){
    int ans=2e9;
    for (int i=1;i<=n;i++) fat[i]=i;
    for (int i=n-1,s,t;i;i--){
        ans=min(ans,Mincut(i,s,t));
        if (!ans) break;
        Merge(s,t);
    }
    return ans;
}
```

10 网络流

10.1 最大流

```
bool BFS(int s,int t){
    int Head=0,Tail=0;
    dis[s]=0;cur[s]=lnk[s];que[++Tail]=s;vis[s]=++ti;
    while (Head!=Tail){
        int x=que[++Head];
        for (int j=lnk[x],u;j;j=nxt[j])
            if (e[j] && vis[u=to[j]]<ti)
                dis[u]=dis[x]+1,cur[u]=lnk[u],que[++Tail]=u,vis[u]=ti;
    }
    return vis[t]==ti;
}
LL DFS(int x,int t,int MIN=2147483647){
    if (x==t || !MIN) return MIN;
    LL f=0;int now;
    for (int &j=cur[x];j;j=nxt[j])
        if (dis[x]+1==dis[to[j]] && (now=DFS(to[j],t,min(MIN,e[j]))))
            {e[j]-=now;e[j^1]+=now;f+=now;if (!(MIN-=now)) break;}
    return f;
}
LL Dinic(int s,int t) {LL MF=0;while (BFS(s,t)) MF+=DFS(s,t);return MF;}
```

10.2 费用流

10.2.1 最小费用最大流

```
#define AM(x) ((x)+1<Maxn+5?(x)+1:0)
bool Spfa(int s,int t,LL &MF,LL &MC){
    for (int i=1;i<=Maxn;i++) dis[i]=1e18;MIN[t]=1e18;
    int Head=0,Tail=0;
    dis[s]=0;MIN[s]=1e18;vis[s]=true;que[++Tail]=s;
    while (Head!=Tail){
        int x=que[Head=AM(Head)];vis[x]=false;
        for (int j=lnk[x],u;j;j=nxt[j])
            if (e[j] && dis[x]+w[j]<dis[u=to[j]]){
                dis[u]=dis[x]+w[j];MIN[u]=min(MIN[x],(LL)e[j]);
                fa[u]=x;pre[u]=j;
                if (!vis[u]){
                    que[Tail=AM(Tail)]=u;vis[u]=true;
                    if (dis[u]<dis[AM(Head)]) swap(que[Tail],que[AM(Head)]);
                }
            }
    }
    if (MIN[t]==1e18) return false;
    MF+=MIN[t];MC+=dis[t]*MIN[t];
    for (int x=t,j=pre[x];x!=s;x=fa[x],j=pre[x]) e[j]-=MIN[t],e[j^1]+=MIN[t];
    return true;
}
void MCMF(int s,int t,LL &MF,LL &MC) {MF=MC=0;while (Spfa(s,t,MF,MC));}
```

10.2.2 最大费用可行流

刷费用流的过程中，如果 S 到 T 增广路的费用 ≤ 0 ，即可停止网络流过程，得到最大费用可行流。

10.2.3 负圈最小费用最大流

边权 $w \geq 0$ 时，建 $x \to y$ 边权为 w 的边。
边权 $w < 0$ 时，建 $x \to y$ 边权为 w ，上下界均为容量的边（实际表现为不用建），表示强制满流，并把费用计入答案。然后建 $y \to x$ 边权为 $-w$ 的边，用来退流。
然后刷 S 到 T 上下界最小费用最大流。
注意：这种费用流存在一种操作为把一个不经过 S 和 T 的环整体加流。

```
void Addedge(int x,int y,int z,int c){
    if (c>=0) Add(x,y,z,c); else {
        num[x]-=z;num[y]+=z;ansC+=z*c; // x->y full flow
        Add(y,x,z,-c);
    }
}

/* Build graph */

/* Legal flow */
for (int i=1;i<=n;i++)
    if (num[i]>0) Add(SS,i,num[i],0); else
    if (num[i]<0) Add(i,TT,-num[i],0);
Add(T,S,2e9,0);
int MF,MC;MCMF(SS,TT,MF,MC);
ansF+=e[E];ansC+=MC;

/* Max flow */
for (int j=lnk[SS];j;j=nxt[j]) e[j]=e[j^1]=0;
for (int j=lnk[TT];j;j=nxt[j]) e[j]=e[j^1]=0;
e[E]=e[E^1]=0;
MCMF(S,T,MF,MC);
ansF+=MF;ansC+=MC;
```

10.3 上下界网络流

10.3.1 无源汇上下界可行流

先让所有边的流量都等于下界，然后再加上一些附加流，使得所有边流量守恒。
假设现在所有边的流量已经等于下界了。我们对于一个点 i ，记录 $num[i] = i$ 点流入量 $- i$ 点流出量，那么如果：
1. $num[i] = 0$ ，说明 i 点是流量守恒的，不需要进行处理。
2. $num[i] > 0$ ，说明 i 点入 $>$ 出，连一条附加源 $SS \to i$ 的容量为 $num[i]$ 的边。
3. $num[i] < 0$ ，说明 i 点出 $>$ 入，连一条 $i \to$ 附加汇 TT 的容量为 $-num[i]$ 的边。
如果存在一种可行流使得 $SS \to TT$ 满流，那么就说明原网络存在可行流。
原网络中每条边的流量是下界 $+$ 流量（附加流）。

10.3.2 有源汇上下界可行流

连一条 $T \to S$ 的容量为 ∞ 的边，变回无源汇上下界可行流。
可行流的流量是 $T \to S$ 这条边的流量。

10.3.3 有源汇上下界最大流/最小流

先求出一个有源汇上下界可行流，然后刷 $S \to T$ 的最大流。
在刷最大流之前把多余的边也就是与 SS , TT 相连的边以及 $T \to S$ 的边“干掉”(程序实现时使其满载即可)。
最小流即刷 $T \to S$ 的最大流。
上下界费用流把 Dinic 换成 EK+Spfa 即可。

```
inline void Add(int x,int y,int L,int R){ // x->y limit [L,R]
    to[++E]=y;e[E]=R-L;nxt[E]=lnk[x];lnk[x]=E;
    to[++E]=x;e[E]=0;nxt[E]=lnk[y];lnk[y]=E;
    num[x]-=L;num[y]+=L;
}

S=0;T=maxn+1;SS=T+1;TT=SS+1;ful=0;
E=1;for (int i=S;i<=TT;i++) lnk[i]=num[i]=0;
/* Build graph */

/* Legal flow */
for (int i=S;i<=T;i++)
    if (num[i]>0) Add(SS,i,0,num[i]),ful+=num[i]; else
    if (num[i]<0) Add(i,TT,0,-num[i]);
Add(T,S,0,2e9);
int MF;
if (Dinic(SS,TT)==ful) MF=e[E]; // flow from S to T
else puts("illegal");

/* Max flow */
for (int j=lnk[SS];j;j=nxt[j]) e[j]=e[j^1]=0;
for (int j=lnk[TT];j;j=nxt[j]) e[j]=e[j^1]=0;
e[E]=e[E^1]=0;
MF+=Dinic(S,T);
```

10.4 最小割

10.4.1 切糕模型

有一块 $X \times Y \times Z$ 的切糕，每个点 (x, y, z) 都有不和谐值 $v(x, y, z)$ 。现在要切这块切糕，为每个直线 (x, y) 选出一个点 z ，一种切法的不和谐值为选出点的不和谐值之和，为了平整还需要满足两条相邻的直线选出的点距离不超过 D 。求最小不和谐值。
如果没有距离限制，我们可以把每个 (x, y) 都从 S 到 T 串起来，像这样：$S \to 1 \to 2 \to 3 \to ... \to T$ 将 $z-1 \to z$ 的容量设置为 $v(x, y, z)$ ，然后求最小割就是最优解。
但是有限制，我们可以用容量为 INF 的边实现“屏蔽”效果，对于网络中的点 (x, y, z) ，在 $(x, y, z-1) \to (x', y', z-1-D)$ 和 $(x', y', z+D) \to (x, y, z)$ 之间连容量为 INF 的边。
这样就会形成一个 $S \to (x, y, z-1) \to (x', y', z-1-D) \to (x', y', z+D) \to (x, y, z) \to T$ 的“通路”，从而阻止其他边被选。

10.4.2 占领模型

$n \times m$ 的网格，有两种方法占领一个格子：1. 花费 $c_{i,j}$ 。2. 该格子上下左右的格子已经被占领。
占领一个格子之后有 $b_{i,j}$ 的收益，求收益减去花费的最大值。
先假装所有点都占领了，明显不合法，所以对一个点要么买下来要么把周围的买下来，或抛弃收益。
建边：把 x 拆为 x_{in} 和 x_{out} ，连 $x_{in} \to x_{out} : b_x$ 。对于相邻的节点 y ：
$S \to x_{in} : c_x, x_{in} \to y_{in} : INF, x_{out} \to y_{out} : INF, y_{out} \to T : c_y$
为避免成环，将网格黑白染色。

10.4.3 最大权闭合图

给出一张带点权（有负权）的有向图，你可以选出一些点，如果这些点中任意一个点连接到的点均在选出的点中，则称这张图是闭合图，得到的价值是选出的点的点权之和，求最大权闭合图。
如果 x 的点权 $w_x > 0$ ，就建 (S, x, w_x) 的边，否则建 $(x, T, -w_x)$ 的边，然后这张图中的边 (u, v) 就建 $(u, v, +\infty)$ ，答案：

$$(\sum_{w_x>0} w_x) - MinCut(S, T)$$

经典模型：无向图，对于正权边 (x, y) ，如果 x 和 y（负权）选了，就可以获得这条边的贡献，求最大贡献。把边也当成点，选了 (x, y) 就必须选 x 和 y ，求最大权闭合图即可。

10.4.4 最大密度子图

无向图，点权 p_x ，边权 $w_e (w_e \geq 0)$ 。选出一些点 V' ，求最大密度子图（不允许不选）：

$$\frac{\sum_{x \in V'} p_x + \sum_{e \in E'} w_e}{|V'|}, E' = \{(u, v) | u \in V', v \in V', (u, v) \in E\}$$

01 分数规划，二分答案 mid ，得到：

$$\sum_{e \in E'} w_e - \sum_{x \in V'} (mid - p_x) \geq 0 \sum_{e \in E'} W_e - \sum_{x \in V'} P_x \geq 0$$

只要是形如 W_e, P_x 的式子都可以用以下方法处理：
建 $S \to i : U$ ， $x \to y : W_e$ ， $i \to T : U + 2P_i - d_i$ ，其中 d_i 为与 i 相连的所有边的 W_e 之和，U 是比所有 $d_i - 2P_i$ 都大的数。
求 $S \to T$ 的最小割，上述式子的等价条件为 $Un - MinCut(S, T) > 0$（不能 $= 0$ ，否则说明没有选点）。
如果强制选 i 点，只需要把 $S \to i$ 容量改成 INF 。
求方案：从 S 开始遍历非满载的边，能遍历到的点说明是被选的点。

```
inline int fcmp(DB a,DB b) {return fabs(a-b)<eps?0:(a<b?-1:1);}
bool check(DB mid){
    DB U=Max(d[i]-2*p[i]);
    /* Calculate w[x][y] and p[i] */
    E=1;for (int i=0;i<=Maxn;i++) lnk[i]=0,d[i]=0;
    for (int i=1;i<=m;i++)
        Add(x[i],y[i],w[i]),Add(y[i],x[i],w[i]),
        d[x[i]]+=w[i],d[y[i]]+=w[i];
    for (int i=1;i<=n;i++) Add(S,i,U),Add(i,T,U+2*p[i]-d[i]);
    return fcmp(U*n,Dinic(S,T))>0;
}
```

10.4.5 最小割选边

给出一张 n 个点 m 条有向边的图，现在要求 S,T 的最小割，问每一条边：1. 有没有可能出现在最小割中。2. 是否一定出现在最小割中。

先跑出随意一种最小割（最大流），然后在残量网络（没有满流的边）里缩点，结论：

1. 如果 $x \to y$ 没有满流，则一定不在最小割里。
2. $x \to y$ 可能出现：$SCC_x \neq SCC_y$ ；
3. $x \to y$ 一定出现：$SCC_S = SCC_x, SCC_y = SCC_T$ 。

10.5 神秘的超快网络流/费用流模版

```cpp
template <const int MAXV, class flowUnit, class costUnit, const costUnit COST_INF,
        const flowUnit FLOW_EPS = 0, const costUnit COST_EPS = 0, const int SCALE = 8>
struct FLOW {
    struct Edge {
        int to;
        flowUnit cap, resCap;
        costUnit cost;
        int rev;
        Edge(int to, flowUnit cap, costUnit cost, int rev) :
            to(to), cap(cap), resCap(cap), cost(cost), rev(rev) {}
    };
    int cnt[MAXV * 2], h[MAXV], stk[MAXV], top;
    flowUnit maxFlow, ex[MAXV];
    costUnit phi[MAXV], bnd, minCost, negCost;
    vector<int> hs[MAXV * 2];
    vector<Edge> adj[MAXV];
    typename vector<Edge>::iterator cur[MAXV];
    FLOW() {}
    void addEdge(int v, int w, flowUnit flow, costUnit cost) {
        if (v == w) {
            if (cost < 0) negCost += flow * cost;
            return;
        }
        adj[v].emplace_back(w, flow, cost, (long long)(adj[w].size()));
        adj[w].emplace_back(v, 0, -cost, (long long)(adj[v].size()) - 1);
    }
    void init(int V) {
        negCost = 0;
        for (int i = 0; i < V; i++) adj[i].clear();
    }
    flowUnit Flow(int V, int s, int t) {
        auto push = [&](int v, Edge &e, flowUnit df) {
            int w = e.to;
            if (abs(ex[w]) <= FLOW_EPS && df > FLOW_EPS) hs[h[w]].push_back(w);
            e.resCap -= df;
            adj[w][e.rev].resCap += df;
            ex[v] -= df; ex[w] += df;
        };
        if (s == t) return maxFlow = 0;
        fill(h, h + V, 0);    h[s] = V;
```

```
        fill(ex, ex + V, 0); ex[t] = 1;
        fill(cnt, cnt + V * 2, 0);
        cnt[0] = V - 1;
        for (int v = 0; v < V; v++) cur[v] = adj[v].begin();
        for (int i = 0; i < V * 2; i++) hs[i].clear();
        for (auto &&e : adj[s]) push(s, e, e.resCap);
        if (!hs[0].empty()) {
            for (int hi = 0; hi >= 0; ) {
                int v = hs[hi].back();
                hs[hi].pop_back();
                while (ex[v] > FLOW_EPS) {
                    if (cur[v] == adj[v].end()) {
                        h[v] = INT_MAX;
                        for (auto e = adj[v].begin(); e != adj[v].end(); e++)
                            if (e -> resCap > FLOW_EPS && h[v] > h[e -> to] + 1)
                                h[v] = h[e -> to] + 1; cur[v] = e;
                        cnt[h[v]]++;
                        if (--cnt[hi] == 0 && hi < V)
                            for (int i = 0; i < V; i++)
                                if (hi < h[i] && h[i] < V) {cnt[h[i]]--; h[i] = V + 1;}
                        hi = h[v];
                    } else if (cur[v]->resCap > FLOW_EPS && h[v] == h[cur[v]->to] + 1){
                        push(v, *cur[v], min(ex[v], cur[v] -> resCap));
                    } else cur[v]++;
                }
                while (hi >= 0 && hs[hi].empty()) hi--;
            }
        }
        return maxFlow = -ex[s];
    }
    pair<flowUnit, costUnit> Cost(int V, int s = -1, int t = -1) {
        auto costP = [&](int v, const Edge & e) {return e.cost + phi[v] - phi[e.to];};
        auto push = [&](int v, Edge & e, flowUnit df, bool pushToStack) {
            if (e.resCap < df) df = e.resCap;
            int w = e.to;
            e.resCap -= df;
            adj[w][e.rev].resCap += df;
            ex[v] -= df; ex[w] += df;
            if (pushToStack && FLOW_EPS < ex[e.to] && ex[e.to] <= df + FLOW_EPS)
                stk[top++] = e.to;
        };
        auto relabel = [&](int v, costUnit delta) {phi[v] -= delta + bnd;};
        auto lookAhead = [&](int v) {
            if (abs(ex[v]) > FLOW_EPS) return false;
            costUnit delta = COST_INF;
            for (auto &&e : adj[v]) {
                if (e.resCap <= FLOW_EPS) continue;
                costUnit c = costP(v, e);
                if (c < -COST_EPS) return false;
                else delta = min(delta, c);
            }
            relabel(v, delta);
```

```
            return true;
        };
        auto discharge = [&](int v) {
            costUnit delta = COST_INF;
            for (int i = 0; i < (int)(adj[v].size()); i++) {
                Edge &e = adj[v][i];
                if (e.resCap <= FLOW_EPS) continue;
                if (costP(v, e) < -COST_EPS) {
                    if (lookAhead(e.to)) {i--; continue;}
                    push(v, e, ex[v], true);
                    if (abs(ex[v]) <= FLOW_EPS) return;
                } else delta = min(delta, costP(v, e));
            }
            relabel(v, delta);
            stk[top++] = v;
        };
        minCost = 0;
        bnd = 0;
        costUnit mul = 2 << __lg(V);
        for (int v = 0; v < V; v++)
            for (auto &&e : adj[v]) {
                minCost += e.cost * e.resCap;
                e.cost *= mul;
                bnd = max(bnd, e.cost);
            }
        maxFlow = (s == -1 || t == -1) ? 0 : Flow(V, s, t);
        fill(phi, phi + V, 0);
        fill(ex, ex + V, 0);
        while (bnd > 1) {
            bnd = max(bnd / SCALE, costUnit(1));
            top = 0;
            for (int v = 0; v < V; v++)
                for (auto &&e : adj[v])
                    if (costP(v, e) < -COST_EPS && e.resCap > FLOW_EPS)
                        push(v, e, e.resCap, false);
            for (int v = 0; v < V; v++)
                if (ex[v] > FLOW_EPS) stk[top++] = v;
            while (top > 0) discharge(stk[--top]);
        }
        for (int v = 0; v < V; v++)
            for (auto &&e : adj[v]) {
                e.cost /= mul; minCost -= e.cost * e.resCap;
            }
        return pair(maxFlow, (minCost /= 2) += negCost);
    }
};

const int N = 250005;
const long long inf = 0X3F3F3F3F3F3F3F3F;

FLOW<N, int, long long, inf> flow;
```

10.6 Colin 的网络流/费用流模版

```cpp
inline bool getmin(int &a, int b) {return (a > b ? (a = b, true) : false);}
inline bool getmax(int &a, int b) {return (a < b ? (a = b, true) : false);}

// F is the type of flow
template<const int V, const int E, class F, const F flowInf>
struct MF {

    int tot = 1, S, T, hd[V], cur[V], dis[V];
    struct edge{int to, nxt; F cap;} e[E << 1];

    void clear() {tot = 1; memset(hd, 0, sizeof(hd));}
    void add(int u, int v, F w) {
        e[++tot].nxt = hd[u], hd[u] = tot, e[tot].to = v, e[tot].cap = w;
        e[++tot].nxt = hd[v], hd[v] = tot, e[tot].to = u, e[tot].cap = 0;
    }
    inline bool bfs() {
        static int q[V], qhd, qtl;
        memcpy(cur, hd, sizeof(hd));
        memset(dis, -1, sizeof(dis));
        q[qhd = qtl = 1] = S; dis[S] = 0;
        while (qhd <= qtl) {
            int u = q[qhd++];
            for (int i = hd[u], v; i; i = e[i].nxt)
                if (dis[v = e[i].to] == -1 && e[i].cap != 0) {
                    dis[v] = dis[u] + 1; q[++qtl] = v;
                }
        }
        return dis[T] != -1;
    }
    F dfs(int u, F rem) {
        if (u == T) return rem;
        F flow = 0;
        for (int i = cur[u], v; i && rem; i = e[i].nxt) {
            cur[u] = i; v = e[i].to;
            F nw = min(rem, e[i].cap);
            if (nw != 0 && dis[v] == dis[u] + 1) {
                int ret = dfs(v, nw);
                flow += ret; rem -= ret;
                e[i].cap -= ret; e[i ^ 1].cap += ret;
            }
        }
        if (flow == 0) dis[u] = -1;
        return flow;
    }
    F dinic(int source, int sink) {
        S = source; T = sink; F flow = 0;
        while (bfs()) flow += dfs(S, flowInf);
        return flow;
    }
};
```

```cpp

// F is the type of flow
// C is the type of cost
template<const int V, const int E, class F, class C, const F flowInf>
struct MCMF {

    int tot = 1, S, T, hd[V];
    struct edge{int nxt, to; F cap; C cst;} e[E << 1];

    void clear() {tot = 1; memset(hd, 0, sizeof(hd));}

    void add(int u, int v, F cap, C cst) {
        e[++tot] = {hd[u], v, cap, cst}; hd[u] = tot;
        e[++tot] = {hd[v], u, 0,  -cst}; hd[v] = tot;
    }

    int inid[V]; F fl[V]; C dis[V];

    inline bool spfa() {
        static queue<int> q;
        static bool inq[V];
        memset(inq, 0, sizeof(inq));
        memset(dis, 0x3f, sizeof(dis));
        C cstInf = dis[0];

        q.push(S); dis[S] = 0; fl[S] = flowInf;
        while (!q.empty()) {
            int u = q.front();
            q.pop(); inq[u] = false;
            for (int i = hd[u], v; i; i = e[i].nxt)
                if (e[i].cap != 0 && getmin(dis[v = e[i].to], dis[u] + e[i].cst)) {
                    inid[v] = i;
                    fl[v] = min(e[i].cap, fl[u]);
                    if (!inq[v]) {inq[v] = true; q.push(v);}
                }
        }
        return dis[T] < cstInf;
    }

    pair<F, C> EK(int source, int sink) {
        S = source; T = sink;
        F flow = 0; C cost = 0;
        while (spfa()) {
            flow += fl[T];
            cost += dis[T] * fl[T];
            for (int u = T; u != S; u = e[inid[u] ^ 1].to)
                e[inid[u]].cap -= fl[T], e[inid[u] ^ 1].cap += fl[T];
        }
        return make_pair(flow, cost);
    }
};
```

11 字符串

11.1 Trie

倍增技巧

如果从上往下一直跳某个字符，每个节点需要对于每个字符都开倍增数组，是不可接受的。如果我们记录 $who_{x,ch}$ 表示节点 x 往下一直跳 ch 字符最深能到哪里，就可以通过从下往上跳规避掉这个问题。

一种复杂度正确的暴力做法

Trie 可以对于每个节点记录 $O(size)$ 大小的节点信息。
每次插入一个数，只会让串长个节点的 si 增加，也就是说，元素个数总和为 $O(\sum len)$ 。

11.2 Manacher

```
void Manacher(char *a,int n){
    int pos=0,R=0;
    for (int i=1;i<=n;i++){
        if (i<R) p[i]=min(p[(pos<<1)-i],R-i); else p[i]=1;
        while (1<=i-p[i] && i+p[i]<=n && a[i-p[i]]==a[i+p[i]]) p[i]++;
        if (i+p[i]>R) pos=i,R=i+p[i];
    }
}

for (int i=1;i<=n;i++) a[(i<<1)-1]=s[i],a[i<<1]='%';
Manacher(a,(n<<1)-1);
```

11.3 AC 自动机

```
void getfai(){
    int Head=0,Tail=0;
    for (int i=0,u;i<26;i++)
        if (u=son[0][i]) fai[u]=0,que[++Tail]=u; else son[0][i]=0;
    while (Head!=Tail)
        for (int x=que[++Head],i=0,u;i<26;i++)
            if (u=son[x][i]) fai[u]=son[fai[x]][i],que[++Tail]=u;
            else son[x][i]=son[fai[x]][i];
}
```

11.4 Z 函数

z[i] 是从 i 开始的后缀与整个串最长公共前缀的长度。

```
int l = 0, r = 0;
for (int i = 2; i <= n; ++i) {
    if (i <= r) z[i] = min(z[i - l + 1], r - i + 1);
    while (i + z[i] <= n && t[z[i] + 1] == t[i + z[i]]) ++z[i];
    if (i + z[i] - 1 > r) {l = i; r = i + z[i] - 1;}
}
```

11.5 Lyndon 分解

S 是 Lyndon 串当且仅当 S 本身是所有后缀中的最小串。S 是 Lyndon 串可以推出 S 是所有循环表示中字典序最小的。各种性质与推论：

1. A, B 都是 Lyndon 串且 $A < B$ ，则 $A + B$ 也是 Lyndon 串，因为 $B > A + B$ 。
2. 如果 $S = A + B$ 是 Lyndon 串，则 $B > S > A$ 。
3. S 可以唯一划分成 $A_1 A_2 \cdots A_m$ ， A_i 均为 Lyndon 串，且 $A_i \geq A_{i+1}$ ，划分方法即 Lyndon 分解。
4. 若 Lyndon 串 $S = A + c + B$ ，则 $A + d$ 是 Lyndon 串（其中 c, d 为字符，且 $d > c$ ）。
5. Lyndon 串没有 Border，所以 Lyndon 串也没有周期。

Duval 算法

维护三个位置 i, j, k ， $S[1, i-1]$ 已经分解完毕。

现在 $S[i, k-1] = A^t + B$ ，其中 A 是 Lyndon 串，A^t 是多个 A 接一起。

B 是 A 的可空前缀，$j = k - |A|$ 表示 S_k 在 A 中的对应位置，接下来考虑加上 k 这个位置：

1. $S_k = S_j$ ，则 A 仍是周期，j, k 均往后移动。
2. $S_k > S_j$ ，由性质 3 $B + k$ 是 Lyndon 串且 $A < B + k$ ，前面全部合并，新的 A 更新为 $S[i, k]$ 。
3. $S_k < S_j$ ，则 $A > B + k$ ，所有的 A 都已经确认分解完毕，将 i 移动到 B 的开头。

```
for (int i=1;i<=n;){
    int j=i,k=j+1;
    for (;k<=n && s[k]>=s[j];k++) s[k]==s[j]?j++:j=i;
    for (;i<=j;i+=k-j) L[++m]=i,R[m]=i+(k-j)-1;
}
```

最小表示法

把串再接一遍，最后一个 $L_i \leq n$ 的 Lyndon 串就是最小的循环串。

这样求出的是最大位置，如果要最小位置，对于每次的循环串只取第一个 i 即可。

```
for (int i=1;i<=n;){
    int j=i,k=j+1;
    for (;k<=n && s[k]>=s[j];k++) s[k]==s[j]?j++:j=i;
    for (;i<=j;i+=k-j) if (i<=(n>>1)) pos=i;
}
```

前缀的最小后缀

对于一个前缀，Lyndon 分解之后，最小后缀就是最后一个 Lyndon 串。

```
for (int i=1;i<=n;){
    int j=i,k=j+1; ans[i]=i;
    for (;k<=n && s[k]>=s[j];k++)
        if (s[k]==s[j]) ans[k]=ans[j]+k-j,j++; else ans[k]=i,j=i;
    for (;i<=j;i+=k-j);
}
```

前缀的最大后缀

字符集翻转后对于前缀进行 Lyndon 分解，如果最大后缀不是最后一个 Lyndon 串，则一定是将这个 Lyndon 串作为前缀的串。

```
for (int i=1;i<=n;){
    int j=i,k=j+1; if (!ans[i]) ans[i]=i;
    for (;k<=n && s[k]<=s[j];k++){if (!ans[k]) ans[k]=i; s[k]==s[j]?j++:j=i;}
    for (;i<=j;i+=k-j);
}
```

11.6 后缀数组

11.6.1 倍增

```
void Sort(int n,int m){
    for (int i=0;i<=m;i++) ha[i]=0;for (int i=1;i<=n;i++) ha[rk[i]]++;
    for (int i=1;i<=m;i++) ha[i]+=ha[i-1];
    for (int i=n;i;i--) SA[ha[rk[sc[i]]]--]=sc[i];
}
void Make(char *s,int n,int m=255){ // notice s[i] can't be 0
    for (int i=1;i<=n;i++) rk[i]=s[i],sc[i]=i,sc[i+n]=0;Sort(n,m);
    for (int k=1,p=0;p<n;m=p,k<<=1){
        p=0;for (int i=n-k+1;i<=n;i++) sc[++p]=i;
        for (int i=1;i<=n;i++) if (SA[i]>k) sc[++p]=SA[i]-k;
        Sort(n,m);for (int i=1;i<=n;i++) sc[i]=rk[i];rk[SA[1]]=p=1;
        for (int i=2;i<=n;i++)
            rk[SA[i]]=(p+=sc[SA[i-1]]!=sc[SA[i]] || sc[SA[i-1]+k]!=sc[SA[i]+k]);
    }
    for (int i=1,k=0;i<=n;i++){
        if (rk[i]==1) continue;if (k) k--;
        while (s[i+k]==s[SA[rk[i]-1]+k]) k++;
        RMQ[0][rk[i]]=k;
    }
    for (int i=2;i<=n;i++) lg[i]=lg[i>>1]+1;
    for (int j=1;(1<<j)<n;j++)
        for (int i=2;i+(1<<j)-1<=n;i++)
            RMQ[j][i]=min(RMQ[j-1][i],RMQ[j-1][i+(1<<j-1)]);
}
int LCP(int x,int y){
    if (x>y) swap(x,y);x++;
    int k=lg[y-x+1];
    return min(RMQ[k][x],RMQ[k][y-(1<<k)+1]);
}
```

11.6.2 SA-IS

```
inline bool lmschar(int *tp,int x) {return x>0 && tp[x] && !tp[x-1];}
bool check(int *s,int *tp,int x,int y){
    if (s[x]!=s[y]) return false;
    for (x++,y++;!lmschar(tp,x) && !lmschar(tp,y);x++,y++)
        if (s[x]!=s[y]) return false;
    return s[x]==s[y];
}
void Sort(int *s,int *SA,int *tp,int *ha,int *lb,int *sb,int n,int m){
    for (int i=0;i<=n;i++)
        if (SA[i]>0 && !tp[SA[i]-1])
            SA[lb[s[SA[i]-1]]++]=SA[i]-1;
    for (int i=1;i<=m;i++) sb[i]=ha[i]-1;
    for (int i=n;i>=0;i--)
        if (SA[i]>0 && tp[SA[i]-1])
            SA[sb[s[SA[i]-1]]--]=SA[i]-1;
}
```

```cpp
int* SAIS(int *s,int n,int m){
    int *SA=new int[n+1],*pos=new int[n+1],*ID=new int[n+1],*tp=new int[n+1];
    int *ha=new int[m+1],*lb=new int[m+1],*sb=new int[m+1];
    for (int i=0;i<=m;i++) ha[i]=0;for (int i=0;i<=n;i++) ha[s[i]]++;
    lb[0]=sb[0]=0;
    for (int i=1;i<=m;i++) ha[i]+=ha[i-1],lb[i]=ha[i-1],sb[i]=ha[i]-1;
    tp[n]=1;for (int i=n-1;i>=0;i--) tp[i]=(s[i]==s[i+1]?tp[i+1]:s[i]<s[i+1]);
    int cnt=0;
    for (int i=1;i<=n;i++) if (tp[i] && !tp[i-1]) pos[cnt++]=i;
    for (int i=0;i<=n;i++) SA[i]=-1;
    for (int i=0;i<cnt;i++) SA[sb[s[pos[i]]]--]=pos[i];
    Sort(s,SA,tp,ha,lb,sb,n,m);
    for (int i=0;i<=n;i++) ID[i]=-1;
    int tot=1;bool fl=false;
    for (int i=1,lst=-1,x;i<=n;i++)
        if (lmschar(tp,x=SA[i])){
            if (lst>=0 && !check(s,tp,x,lst)) tot++;
            if (lst>=0 && ID[lst]==tot) fl=true;
            ID[x]=tot;lst=x;
        }
    ID[n]=0;
    int *t=new int[cnt];
    for (int i=0,K=0;i<=n;i++) if (ID[i]>=0) t[K++]=ID[i];
    int *ST;
    if (!fl){
        ST=new int[cnt+1];
        for (int i=0;i<cnt;i++) ST[t[i]]=i;
    } else ST=SAIS(t,cnt-1,tot);
    lb[0]=sb[0]=0;
    for (int i=1;i<=m;i++) lb[i]=ha[i-1],sb[i]=ha[i]-1;
    for (int i=0;i<=n;i++) SA[i]=-1;
    for (int i=cnt-1;i>=0;i--) SA[sb[s[pos[ST[i]]]]--]=pos[ST[i]];
    Sort(s,SA,tp,ha,lb,sb,n,m);
    /* Delete if necessary
    delete[] pos;delete[] ID;delete[] tp;
    delete[] ha;delete[] lb;delete[] sb;
    delete[] t;delete[] ST;
    */
    return SA;
}
void MakeSA(char *s,int n,int m=255){
    static int t[maxn+5];
    for (int i=1;i<=n;i++) t[i-1]=s[i];t[n]=0;
    SA=SAIS(t,n,m);
    for (int i=1;i<=n;i++) SA[i]++,rk[SA[i]]=i;
}
/* Delete if necessary
delete[] SA;
*/
```

11.7 （广义）后缀自动机

11.7.1 静态

```
const int maxi=size of character set;
inline int newnode() {pl++;for (int i=0;i<maxi;i++) son[pl][i]=0;return pl;}
// normal
int Extend(int p,int c,int id){
    int np=newnode();MAX[np]=MAX[p]+1;ID[id]=np;
    while (p && !son[p][c]) son[p][c]=np,p=fai[p];
    if (!p) {fai[np]=ro;return np;}
    int q=son[p][c];if (MAX[p]+1==MAX[q]) {fai[np]=q;return np;}
    int nq=newnode();MAX[nq]=MAX[p]+1;
    for (int i=0;i<maxi;i++) son[nq][i]=son[q][i];
    fai[nq]=fai[q];fai[q]=fai[np]=nq;
    while (p && son[p][c]==q) son[p][c]=nq,p=fai[p];
    return np;
}
// generalized
int Extend(int p,int c,int ID){
    if (son[p][c]){
        int q=son[p][c];if (MAX[p]+1==MAX[q]) return q;
        int nq=newnode();MAX[nq]=MAX[p]+1;
        for (int i=0;i<maxi;i++) son[nq][i]=son[q][i];
        fai[nq]=fai[q];fai[q]=nq;
        while (p && son[p][c]==q) son[p][c]=nq,p=fai[p];
        return nq;
    } else {
        int np=newnode();MAX[np]=MAX[p]+1;
        while (p && !son[p][c]) son[p][c]=np,p=fai[p];
        if (!p) {fai[np]=ro;return np;}
        int q=son[p][c];if (MAX[p]+1==MAX[q]) {fai[np]=q;return np;}
        int nq=newnode();MAX[nq]=MAX[p]+1;
        for (int i=0;i<maxi;i++) son[nq][i]=son[q][i];
        fai[nq]=fai[q];fai[q]=fai[np]=nq;
        while (p && son[p][c]==q) son[p][c]=nq,p=fai[p];
        return np;
    }
}
```

11.7.2 动态 (LCT 维护 parent 树)

```
struct SAM{
    int p,pl,ro,son[maxt+5][maxi],fai[maxt+5],MAX[maxt+5],vis[maxt+5];
    int to[maxt+5][2],fa[maxt+5],si[maxt+5][2];bool flip[maxt+5];
    #define is_ro(p) ((p)!=to[fa[p]][0] && (p)!=to[fa[p]][1])
    #define Son(p) ((p)==to[fa[p]][1])
    inline void Pushup(int p) {
        si[p][1]=si[to[p][0]][1]+si[to[p][1]][1]+si[p][0]+vis[p];
    }
    inline void Rotate(int t){
        int p=fa[t],d=Son(t);to[p][d]=to[t][d^1];to[t][d^1]=p;
```

```
        Pushup(p);Pushup(t);if (!is_ro(p)) to[fa[p]][Son(p)]=t;
        if (to[p][d]) fa[to[p][d]]=p;fa[t]=fa[p];fa[p]=t;
    }
    inline void Addflip(int p) {swap(to[p][0],to[p][1]);flip[p]^=1;}
    inline void Pushdown(int p) {
        if (flip[p]) flip[p]^=1,Addflip(to[p][0]),Addflip(to[p][1]);
    }
    inline void Splay(int p){
        static int top,stk[maxt+5];stk[top=1]=p;
        for (int i=p;!is_ro(i);i=fa[i]) stk[++top]=fa[i];
        while (top) Pushdown(stk[top--]);
        for (int pre=fa[p];!is_ro(p);Rotate(p),pre=fa[p])
            if (!is_ro(pre)) Rotate(Son(p)==Son(pre)?pre:p);
    }
    inline void Access(int p){
        for (int lst=0;p;Pushup(p),lst=p,p=fa[p])
            Splay(p),si[p][0]+=si[to[p][1]][1]-si[lst][1],to[p][1]=lst;
    }
    inline void Makero(int x) {Access(x);Splay(x);Addflip(x);}
    inline void Link(int x,int y) {
        Makero(x);Makero(y);fa[x]=y;si[y][0]+=si[x][1];Pushup(y);
    }
    inline void Cut(int x,int y) {
        Makero(x);Access(y);Splay(y);fa[x]=to[y][0]=0;Pushup(y);
    }
    int newnode(){
        pl++;for (int i=0;i<maxi;i++) son[pl][i]=0;vis[pl]=0;
        to[pl][0]=to[pl][1]=fa[pl]=si[pl][0]=si[pl][1]=flip[pl]=0;
        return pl;
    }
    void clear() {pl=0;ro=newnode();p=ro;}
    void Extend(int c){
        int np=newnode();MAX[np]=MAX[p]+1;vis[np]=1;Pushup(np);
        while (p && !son[p][c]) son[p][c]=np,p=fai[p];
        if (!p) {Link(np,ro);fai[np]=ro;p=np;return;}
        int q=son[p][c];if (MAX[p]+1==MAX[q]) {Link(np,q);fai[np]=q;p=np;return;}
        int nq=newnode();MAX[nq]=MAX[p]+1;
        for (int i=0;i<maxi;i++) son[nq][i]=son[q][i];
        Cut(q,fai[q]);Link(nq,fai[q]);Link(q,nq);Link(np,nq);
        fai[nq]=fai[q];fai[q]=fai[np]=nq;
        while (p && son[p][c]==q) son[p][c]=nq,p=fai[p];
        p=np;return;
    }
};
int Sum(SAM &tr,char *s){
    int p=tr.ro;for (int i=1;s[i];i++) p=tr.son[p][s[i]-'a'];
    if (!p) return 0;
    tr.Makero(tr.ro);tr.Access(p);tr.Splay(p);
    return tr.si[p][0]+tr.vis[p];
}
```

11.8 回文自动机

11.8.1 基础

```
const int maxi= /* size of character set */ ;
inline int newnode() {pl++;for (int i=0;i<maxi;i++) son[pl][i]=0;return pl;}
void Extend(char *s,int i){
    while (s[i]!=s[i-len[p]-1]) p=fai[p];
    if (!son[p][s[i]-'a']){
        int u=newnode(),k=fai[p];len[u]=len[p]+2;
        while (s[i]!=s[i-len[k]-1]) k=fai[k];
        fai[u]=son[k][s[i]-'a'];son[p][s[i]-'a']=u;
    }
    p=son[p][s[i]-'a'];num[p]++;
    ID[i]=p; // p is the node of position i
}

pl=-1;newnode();newnode(); // clear root
fai[0]=fai[1]=1;len[0]=0;len[1]=-1;
p=0;
// num[i] is the number of the palindrome of node i
for (int i=si;i>1;i--) num[fai[i]]+=num[i];
```

11.8.2 前端插入与记忆化

```
const int maxi= /* size of character set */ ;
int qui[maxn+5][maxi]; // quickly transform
inline int newnode() {pl++;for (int i=0;i<maxi;i++) son[pl][i]=qui[pl][i]=0;return pl;}
void Exback(int l,int i){ // left is l, and now insert a[i]
    int c=a[i];if (i-len[pb]-1<l || a[i-len[pb]-1]!=c) pb=qui[pb][c];
    if (!son[pb][c]){
        int u=newnode(),k=fai[pb];len[u]=len[pb]+2;
        if (a[i-len[k]-1]!=c) k=qui[k][c];k=son[k][c];
        for (int i=0;i<maxi;i++) qui[u][i]=qui[k][i];
        qui[u][a[i-len[k]]]=k;fai[u]=k;son[pb][c]=u;
    }
    pb=son[pb][c];if (len[pb]==i-l+1) pf=pb;
}
void Exfront(int i,int r){ // right is r, and now insert a[i]
    int c=a[i];if (i+len[pf]+1>r || a[i+len[pf]+1]!=c) pf=qui[pf][c];
    if (!son[pf][c]){
        int u=newnode(),k=fai[pf];len[u]=len[pf]+2;
        if (a[i+len[k]+1]!=c) k=qui[k][c];k=son[k][c];
        for (int i=0;i<maxi;i++) qui[u][i]=qui[k][i];
        qui[u][a[i+len[k]]]=k;fai[u]=k;son[pf][c]=u;
    }
    pf=son[pf][c];if (len[pf]==r-i+1) pb=pf;
}
pl=-1;newnode();newnode(); // clear root
fai[0]=fai[1]=1;len[0]=0;len[1]=-1;
for (int i=0;i<maxi;i++) qui[0][i]=1;
pb=pf=0;
```

11.8.3 Palindrome Series

在 PAM 上额外维护以下信息：

1. $dif_x = len_x - len_{fail_x}$ ，即 x 与 $fail_x$ 最长 Border 的差，也就是所在等差数列的公差。
2. anc_x 表示 x 往上遇到的第一个 $dif_p \neq dif_{fail_p}$ 的 p 的父亲，即所在等差数列的顶点的父亲。
3. sum_x 表示 $[x, anc_x)$ 路径上的当前信息（包含 x ，不包含 anc_x ）。

Palindrome Series 结论：

1. 一个串的长度在 $[2^k, 2^{k+1})$ 范围内的 Border 长度构成等差数列（Border Series）。
2. 字符串 T 是回文串 S 的一个回文后缀，当且仅当 T 是 S 的 Border。
3. 字符串 S 存在回文 Border T ，且 $|T| \geq \frac{|S|}{2}$ ，则 S 是回文串。
4. 如果 $dif_x > \frac{len_x}{2}$ ，则 $anc_x = fail_x$ 。
5. 如果 $dif_x = dif_{fail_x}$ ，则 $len_{fail_x} \geq \frac{len_x}{2}$ 。
6. 按照字符串前缀插入，则更新信息时，需要用到的 sum_{fail} 一定已经被更新过了。
7. $len_{anc_x} \leq \frac{len_x}{2}$ ， x 沿着 anc_x 只会跳 $\log_2 n$ 次。

Palindrome Series 储存信息：

$$
\begin{aligned}
f_i &= \sum \{f_j | S[j+1, i] \text{ is Palindrome}\} \\
sum_x &= \sum \{f_{i-len_x}, f_{i-len_{fail_x}}, \cdots, f_{i-len_t}\}, fail_t = anc_x \\
sum_{fail_x} &= \sum \{f_{j-len_{fail_x}}, \cdots, f_{j-len_t}\}, fail_t = anc_x \\
j &= i - dif_x, len_x = len_{fail_x} + dif_x \Rightarrow i - len_x = j - len_{fail_x} \\
sum_x &= sum_{fail_x} + f_{i-len_t}, fail_t = anc_x
\end{aligned}
$$

适用于枚举回文后缀统计信息的题。

```
int Extend(int p,char *s,int i){
    while (s[i]!=s[i-len[p]-1]) p=fai[p];
    if (!son[p][s[i]-'a']){
        int u=newnode(),k=fai[p];len[u]=len[p]+2;
        while (s[i]!=s[i-len[k]-1]) k=fai[k];
        fai[u]=son[k][s[i]-'a'];son[p][s[i]-'a']=u;
        dif[u]=len[u]-len[fai[u]];
        anc[u]=(dif[u]==dif[fai[u]]?anc[fai[u]]:fai[u]);
    }
    p=son[p][s[i]-'a'];
    return p;
}
p=Extend(p,s,i);
for (int x=p;x>1;x=anc[x]){
    sum[x]=f[i-(len[anc[x]]+dif[x])];
    if (dif[x]==dif[fai[x]]) sum[x]=ADD(sum[x],sum[fai[x]]);
    f[i]=ADD(f[i],sum[x]);
}
```

12 计算几何

对点 $P(x_0, y_0)$ 和直线 $Ax + By + C = 0$：

1. 点到直线的距离：$\dfrac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}$
2. 点关于直线的对称点：$\left(x_0 - 2A\dfrac{Ax_0 + By_0 + C}{A^2 + B^2}, y_0 - 2B\dfrac{Ax_0 + By_0 + C}{A^2 + B^2}\right)$
3. 点在直线上的投影点：$\left(x_0 - A\dfrac{Ax_0 + By_0 + C}{A^2 + B^2}, y_0 - B\dfrac{Ax_0 + By_0 + C}{A^2 + B^2}\right)$
4. 两直线夹角：$\theta = \arccos\dfrac{|A_1A_2 + B_1B_2|}{\sqrt{A_1^2 + B_1^2} \times \sqrt{A_2^2 + B_2^2}}$，$\theta \in \left[0 \sim \dfrac{\pi}{2}\right]$
5. 两平行直线的距离：$\dfrac{|C_1 - C_2|}{\sqrt{A^2 + B^2}}$

```
 typedef double T;
typedef long double ld;

#define let  const auto
#define lett const T
#define letp const P // P for Point
#define lets const S // S for Segment
#define letl const L // L for Line
#define letc const C // C for Convex

const T eps = 1e-8;
const double PI = 3.1415926535897932384;
#define z(x) (abs((x)) <= eps) // is zero

const ld a90 = atan(1) * 2;
const ld a180 = a90 * 2, a270 = a90 * 3, a360 = a90 * 4;
const ld dlt[2][2] = {0, a180, a360, a180};

struct P {
    T x, y;
    P (T x = 0, T y = 0) : x(x), y(y) {}
    P operator + (letp &p) const {return {x + p.x, y + p.y};}
    P operator - (letp &p) const {return {x - p.x, y - p.y};}
    P operator * (lett &d) const {return {x * d, y * d};}
    P operator / (lett &d) const {return {x / d, y / d};}
    P operator - () const {return {-x, -y};}

    T operator | (letp &p) const {return x * p.x + y * p.y;} // dot
    T operator ^ (letp &p) const {return x * p.y - y * p.x;} // cross

    P rot(double ang) const { // counterclockwise rotation (ang) angle
        double cosa = cos(ang), sina = sin(ang);
        return {x * cosa - y * sina, x * sina + y * cosa};
    }
    P rot(double cosa, double sina) const {
        return {x * cosa - y * sina, x * sina + y * cosa};
    }
```

```
    ld angle() const { // counterclockwise, start from (1, 0)
        if (fabs(x) < 1e-15) return a90 * (1 + 2 * (y < 0));
        else return atan(y / x) + dlt[y < 0][x < 0];
    }

    bool operator == (letp &p) const {return z(x - p.x) && z(y - p.y);}
    bool operator != (letp &p) const {return ! operator == (p);}
    bool operator < (letp &p) const {return z(x - p.x) ? y < p.y : x < p.x;}
    bool operator > (letp &p) const {return !(*this < p || *this == p);}

    // left(counterclockwise) = 1 | on = 0 | right(clockwise) = -1
    int ori(letp &p) const {T t = (*this) ^ p; return (t > eps) - (t < -eps);}

    T norm() const {return x * x + y * y;}
    T dis(letp &p) const {return sqrt(((*this) - p).norm());}
    P proj (letp &p) const {return (*this) * (((*this) | p) / norm());}
    P refl (letp &p) const {return proj(p) * 2 - p;}
} zero;

double abs(letp &p) {return sqrt(p.norm());}
P unit(letp &p) {return p / abs(p);}
P perp(letp &p) {return {-p.y, p.x};} // turn pi / 2 left(counterclockwise)
P perpr(letp &p) {return {p.y, -p.x};} // turn pi / 2 right(clockwise)

bool orth(letp &p, letp &q) {return z(p | q);} // orthogonal
bool para(letp &p, letp &q) {return z(p ^ q);} // parallel

struct argcmp { // compared by polar angle
    bool operator() (letp &a, letp &b) const {
        const auto quad = [](letp &a) {
            if (a.y < -eps) return 1; // halfplane with negative y
            if (a.y > eps) return 4;  // halfplane with positive y
            if (a.x < -eps) return 5; // negative x-axis
            if (a.x > eps) return 3;  // positive x-axis
            return 2; // origin
        };
        const int qa = quad(a), qb = quad(b);
        if (qa != qb) return qa < qb;
        const auto t = (a ^ b); //in the same quad
        /* sorted by length in increasing order when parallel
        if (z(t)) return norm(a) < norm(b) - eps; */
        return t > eps;
    }
};

struct L {
    P p, v;
    int ori (letp &a) const {return v.ori(a - p);}
    P inter(letl &l) const {return p + v * ((l.v ^ (p - l.p)) / (v ^ l.v));}
    L shift(letp &d) const {return {p + d, v};}
    L shiftl(double d) const {return {p + perp(v) * d / abs(v), v};}
    P proj(letp &a) const {return p + v.proj(a - p);}
```

```
    P refl(letp &a) const {return p + v.refl(a - p);}
    double dis(letp &a) const {return abs(v ^ (a - p)) / abs(v);}
};

bool orth(letl &p, letl &q) {return z(p.v | q.v);} // orthogonal
bool para(letl &p, letl &q) {return z(p.v ^ q.v);} // parallel


struct S {
    P a, b;

    // on endPs = -1 | out = 0 | in = 1
    int is_on(letp &p) const {
        if (p == a || p == b) return -1;
        return (p - a).ori(p - b) == 0 && ((p - a) | (p - b)) < -eps;
    }

    // inter on endPs = -1 | not inter = 0 | inter inside = 1
    int is_inter(letl &l) const {
        if (l.ori(a) == 0 || l.ori(b) == 0) return -1;
        return l.ori(a) != l.ori(b);
    }

    // inter on endPs = -1 | not inter = 0 | inter inside = 1
    int is_inter(lets &s) const {
        if (is_on(s.a) || is_on(s.b) || s.is_on(a) || s.is_on(b)) return -1;
        letl &l{a, b - a}, ls{s.a, s.b - s.a};
        return l.ori(s.a) * l.ori(s.b) == -1 && ls.ori(a) * ls.ori(b) == -1;
    }

    double dis(letp &p) const {
        if (((p - a) | (b - a)) < -eps || ((p - b) | (a - b)) < -eps)
            return min(abs(a - p), abs(b - p));
        return L{a, b - a}.dis(p);
    }

    double dis(lets &s) const {
        if (is_inter(s)) return 0.0;
        return min({dis(s.a), dis(s.b), s.dis(a), s.dis(b)});
    }
};

struct Polygon {
    vector<P> p; // counterclockwise
    Polygon(const vector<P> p = {}) : p(p) {}
    size_t nxt(const size_t i) const {return i == p.size() - 1 ? 0 : i + 1;}
    size_t pre(const size_t i) const {return i == 0 ? p.size() - 1 : i - 1;}
    T double_area() const {
        T sum = 0;
        for (size_t i = 0; i < p.size(); ++i) sum += (p[i] ^ p[nxt(i)]);
        return abs(sum);
    }
```

```
    ld area() const {return double_area() / 2.0;}
    ld circ() const {
        ld sum = 0;
        for (size_t i = 0; i < p.size(); ++i) sum += abs(p[i] - p[nxt(i)]);
        return sum;
    }

    // pair<is_on_edge, winding number> , out = [winding number = 0]
    pair<bool,int> winding(letp &a) const {
        int cnt = 0;
        for (size_t i = 0; i < p.size(); ++i) {
            letp u = p[i], v = p[nxt(i)];
            if (z((a - u) ^ (a - v)) && ((a - u) | (a - v)) <= eps) return {true, 0};
            if (z(u.y - v.y)) continue;
            letl uv{u, v - u};
            if (u.y < v.y - eps && uv.ori(a) <= 0) continue;
            if (u.y > v.y + eps && uv.ori(a) >= 0) continue;
            if (u.y < a.y - eps && v.y >= a.y - eps) cnt++;
            if (u.y >= a.y - eps && v.y < a.y - eps) cnt--;
        }
        return {false, cnt};
    }

    bool is_convex() const {
        bool pos = false, neg = false;
        for (size_t i = 0; i < p.size(); ++i) {
            int o = (p[i] - p[pre(i)]).ori(p[nxt(i)] - p[i]);
            if (o == 1) pos = true;
            if (o == -1) neg = true;
        }
        return !(pos && neg);
    }
};

struct C : Polygon {
    C (const vector<P> &p = {}) : Polygon(p) {}
    C operator + (letc &c) const { // Minkowski Sum
        const auto &p = this -> p;
        vector<S> e1(p.size()), e2(c.p.size());
        vector<S> edge(p.size() + c.p.size());
        vector<P> res; res.reserve(p.size() + c.p.size());

        const auto cmp = [](lets &u, lets &v) {
            return argcmp()(u.b - u.a, v.b - v.a);
        };

        for (size_t i = 0; i < p.size(); ++i) e1[i] = {p[i], p[this -> nxt(i)]};
        for (size_t i = 0; i < c.p.size(); ++i) e2[i] = {c.p[i], c.p[c.nxt(i)]};
        rotate(e1.begin(), min_element(e1.begin(), e1.end(), cmp), e1.end());
        rotate(e2.begin(), min_element(e2.begin(), e2.end(), cmp), e2.end());
        merge(e1.begin(), e1.end(), e2.begin(), e2.end(), edge.begin(), cmp);

```

```
        const auto check = [](const vector<P> &res, letp &u) {
            const auto b1 = res.back(), b2 = *prev(res.end(), 2);
            return (b1 - b2).ori(u - b1) == 0 && ((b1 - b2) | (u - b1)) >= -eps;
        };

        auto u = e1[0].a + e2[0].a;
        for (const auto &v : edge) {
            while (res.size() > 1 && check(res, u)) res.pop_back();
            res.push_back(u); u = u + v.b - v.a;
        }
        if (res.size() > 1 && check(res, res[0])) res.pop_back();
        return {res};
    }

    // O(log n) : on = -1 | out = 0 | in = 1
    int is_in(letp &a) const {
        const auto &p = this -> p;
        if (p.size() == 1) return a == p[0] ? -1 : 0;
        if (p.size() == 2) return S{p[0], p[1]}.is_on(a) ? -1 : 0;
        if (a == p[0]) return -1;
        if ((p[1] - p[0]).ori(a - p[0]) == -1) return 0;
        if ((p.back() - p[0]).ori(a - p[0]) == 1) return 0;
        let cmp = [&](letp &u, letp &v) {return (u - p[0]).ori(v - p[0]) == 1;};
        const size_t i = lower_bound(p.begin() + 1, p.end(), a, cmp) - p.begin();
        if (i == 1) return S{p[0], p[i]}.is_on(a) ? -1 : 0;
        if (i == p.size() - 1 && S{p[0], p[i]}.is_on(a)) return -1;
        if (S{p[i - 1], p[i]}.is_on(a)) return -1;
        return (p[i] - p[i - 1]).ori(a - p[i - 1]) > 0;
    }

    template<typename F> size_t extreme(const F &dir) const {
        let &p = this -> p;
        let check = [&](const size_t i) {return dir(p[i]).ori(p[nxt(i)] - p[i]) >= 0;};
        let dir0 = dir(p[0]);
        let check0 = check(0);
        if (!check0 && check(p.size() - 1)) return 0;
        const auto cmp = [&](letp &v) {
            const size_t vi = &v - p.data();
            if (vi == 0) return 1;
            let checkv = check(vi);
            let t = dir0.ori(v - p[0]);
            if (vi == 1 && checkv == check0 && dir0.ori(v - p[0]) == 0) return 1;
            return checkv ^ (checkv == check0 && t <= 0);
        };
        return partition_point(p.begin(), p.end(), cmp) - p.begin();
    }

    pair<size_t,size_t> tangent(letp &a) const {
        const size_t i = extreme([&](letp &u){return u - a;});
        const size_t j = extreme([&](letp &u){return a - u;});
        return {i, j};
    }
```

```
246
247     pair<size_t,size_t> tangent(letl &a) const {
248         const size_t i = extreme([&](...){return a.v;});
249         const size_t j = extreme([&](...){return -a.v;});
250         return {i, j};
251     }
252
253 };
254
255 C convexHull(vector<P> p) {
256     vector<P> st;
257     sort(p.begin(), p.end());
258     const auto check = [](const vector<P> &st, letp &u) {
259         const auto back1 = st.back(), back2 = *prev(st.end(), 2);
260         return (back1 - back2).ori(u - back2) <= 0;
261     };
262     for (letp &u : p) {
263         while (st.size() > 1 && check(st, u)) st.pop_back();
264         st.push_back(u);
265     }
266     size_t k=st.size();
267     p.pop_back(); reverse(p.begin(),p.end());
268     for (letp &u : p) {
269         while (st.size() > k && check(st, u)) st.pop_back();
270         st.push_back(u);
271     }
272     st.pop_back();
273     return {st};
274 }
275
276 vector<L> halfInter(vector<L> l, lett lim = 1e9) { // O(n log n)
277     const auto check = [](letl &a, letl &b, letl &c) { return a.ori(b.inter(c)) < 0;};
278     /* // no precision error, but the number will become x^3
279     const auto check = [](letl &a, letl &b, letl &c) {
280         letp t = (b.v ^ c.v);
281         letp p = (a.v | t), q = (b.p | t) + (b.v | (c.v ^ (b.p - c.p))) - (a.p | t);
282         return p.ori(q) < 0;
283     }; */
284     const auto cmp = [] (letl &a, letl &b) {
285         if (abs(a.v ^ b.v) <= eps && (a.v | b.v) >= -eps) return a.ori(b.p) == -1;
286         return argcmp()(a.v, b.v);
287     };
288     l.push_back({{-lim, 0}, {0, -1}}); l.push_back({{0, -lim}, {1, 0}});
289     l.push_back({{lim, 0}, {0, 1}});   l.push_back({{0, lim}, {-1, 0}});
290     sort(l.begin(), l.end(), cmp);
291     deque<L> q;
292     for (size_t i=0;i<l.size();i++) {
293         if (i && l[i - 1].v.ori(l[i].v) == 0 && (l[i - 1].v | l[i].v) > eps) continue;
294         while (q.size() > 1 && check(l[i], q.back(), q[q.size() - 2])) q.pop_back();
295         while (q.size() > 1 && check(l[i], q[0], q[1])) q.pop_front();
296         if (!q.empty() && q.back().v.ori(l[i].v) <= 0) return vector<L>();
297         q.push_back(l[i]);
```

```
    }
    while (q.size() > 1 && check(q[0], q.back(), q[q.size() - 2])) q.pop_back();
    while (q.size() > 1 && check(q.back(), q[0], q[1])) q.pop_front();
    return vector<L>(q.begin(), q.end());
}

C halfInterConvex(vector<L> l, lett lim = 1e9) {
    l = halfInter(l, lim);
    if (l.size() <= 1) return vector<P>();
    vector<P> res; res.resize(l.size());
    for (size_t i = 0; i < l.size(); ++i)
        res[i] = l[i].inter(l[i == l.size() - 1 ? 0 : i + 1]);
    return res;
}

struct Circle {
    P c; ld r;
    Circle(letp &c = zero, double r = 0.0) : c(c), r(r) {}
    Circle(letp &A = zero, letp &B = zero, letp &C = zero) {
        double x1 = A.x - B.x, y1 = A.y - B.y;
        double x2 = A.x - C.x, y2 = A.y - C.y;
        double e = ((A.x * A.x - B.x * B.x) - (B.y * B.y - A.y * A.y)) / 2.0;
        double f = ((A.x * A.x - C.x * C.x) - (C.y * C.y - A.y * A.y)) / 2.0;
        c.x = (e * y2 - y1 * f) / (x1 * y2 - y1 * x2);
        c.y = (x1 * f - e * x2) / (x1 * y2 - y1 * x2);
        r = A.dis(c);
    }
    bool operator == (const Circle &a) const {return c == a.c && z(r - a.r);}
    ld circ() const {return 2 * PI * r;}  // 周长
    ld area() const {return PI * r * r;}  // 面积
    // 点与圆的关系 : -1 圆上 | 0 圆外 | 1 圆内
    int is_in(letp &p) const {ld d = p.dis(c); return z(d-r) ? -1 : d < r - eps;}
    // 直线与圆关系 : 0 相离 | 1 相切 | 2 相交
    int relation(letl &l) const {
        ld d = l.dis(c);
        return (z(d - r) ? 1 : (d > r + eps ? 0 : 2));
    }
    // 圆与圆关系 : -1 相同 | 0 相离 | 1 外切 | 2 相交 | 3 内切 | 4 内含
    int relation(const Circle &a) const {
        if (*this == a) return -1;
        ld d = c.dis(a.c);
        if (d > r + a.r + eps) return 0;
        if (z(d - r - a.r)) return 1;
        if (z(d - abs(r - a.r))) return 3;
        if (d < abs(r - a.r) - eps) return 4;
        return 2;
    }
    // 直线与圆的交点
    vector<P> inter(letl &l) const {
        const ld d = l.dis(c);
        const P p = l.proj(c);
        const int t = relation(l);
```

```
        if (t == 0) return vector<P>();
        if (t == 1) return vector<P>{p};
        const ld k = sqrt(r * r - d * d);
        return vector<P>{p - unit(l.v) * k, p + unit(l.v) * k};
    }
    // 圆与圆交点
    vector<P> inter(const Circle &a) const {
        const ld d = c.dis(a.c);
        const int t = relation(a);
        if (t == -1 || t == 0 || t == 4) return vector<P>();
        P e = a.c - c; e = unit(e) * r;
        if (t == 1 || t == 3) {
            if (r * r + d * d - a.r * a.r >= -eps) return vector<P>{c + e};
            return vector<P>{c - e};
        }
        const ld costh = (r * r + d * d - a.r * a.r) / (2 * r * d);
        const ld sinth = sqrt(1 - costh * costh);
        return vector<P>{c + e.rot(costh, -sinth), c + e.rot(costh, sinth)};
    }
    // 圆与圆交面积
    ld inter_area(const Circle &a) const {
        const ld d = c.dis(a.c);
        const int t = relation(a);
        if (t == -1) return area();
        if (t < 2) return 0;
        if (t > 2) return min(area(), a.area());
        const ld costh1 = (r * r + d * d - a.r * a.r) / (2 * r * d);
        const ld costh2 = (a.r * a.r + d * d - r * r) / (2 * a.r * d);
        const ld sinth1 = sqrt(1 - costh1 * costh1);
        const ld sinth2 = sqrt(1 - costh2 * costh2);
        const ld th1 = acos(costh1), th2 = acos(costh2);
        return r * r * (th1 - costh1 * sinth1) + a.r * a.r * (th2 - costh2 * sinth2);
    }
    // 过圆外一点圆的切线
    vector<L> tangent(const P &a) const {
        const int t = is_in(a);
        if (t == 1) return vector<L>();
        if (t == -1) {
            const P v = {-(a - c).y, (a - c).x};
            return vector<L>{{a, v}};
        }
        P e = a - c; e = unit(e) * r;
        const ld costh = r / c.dis(a);
        const ld sinth = sqrt(1 - costh * costh);
        const P t1 = c + e.rot(costh, -sinth);
        const P t2 = c + e.rot(costh, sinth);
        return vector<L>{{a, t1 - a}, {a, t2 - a}};
    }
    // 两圆的公切线
    vector<L> tangent(const Circle &a) const {
        const int t = relation(a);
        vector<L> Ls;
```

```
        if (t == -1 || t == 4) return Ls;
        if (t == 1 || t == 3) {
            const P p = inter(a)[0], v = {-(a.c - c).y, (a.c - c).x};
            Ls.push_back({p, v});
        }
        const ld d = c.dis(a.c);
        const P e = unit(a.c - c);
        if (t <= 2) {
            const ld costh = (r - a.r) / d;
            const ld sinth = sqrt(1 - costh * costh);
            const P d1 = e.rot(costh, -sinth), d2 = e.rot(costh, sinth);
            const P u1 = c + d1 * r, u2 = c + d2 * r;
            const P v1 = a.c + d1 * a.r, v2 = a.c + d2 * a.r;
            Ls.push_back({u1, v1 - u1});
            Ls.push_back({u2, v2 - u2});
        }
        if (t == 0) {
            const ld costh = (r + a.r) / d;
            const ld sinth = sqrt(1 - costh * costh);
            const P d1 = e.rot(costh, -sinth), d2 = e.rot(costh, sinth);
            const P u1 = c + d1 * r, u2 = c + d2 * r;
            const P v1 = a.c - d1 * a.r, v2 = a.c - d2 * a.r;
            Ls.push_back({u1, v1 - u1});
            Ls.push_back({u2, v2 - u2});
        }
        return Ls;
    }
};
```