快读

```
inline bool read(int& a)
{
    int s = 0, w = 1;
    char ch = getchar();
    if(ch==EOF)
        return false;
    while (ch < '0' || ch>'9')
    {
        if (ch == '-')
            w = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9')
    {
        s = s * 10 + ch - '0';
        ch = getchar();
    }
    a = s * w;
    return true;
}
```

快输

```
void write(int x)
{
    if(x<0)
        putchar('-'),x=-x;
    if(x>9)
        write(x/10);
    putchar(x%10+'0');
    return;
}
```

随机数生成

```
mt19937 eng(time(0));
int randint(int a, int b)
{
    uniform_int_distribution<int> dis(a, b);
    return dis(eng);
}
```

快速幂

```
ll qpow(ll a,ll n,ll m)
{
    ll ans=1;
    while(n)
    {
        if(n&1)
            ans=(__int128_t)ans*a%m;
        a=(__int128_t)a*a%m;
        n>>=1;
    }
    return ans;
}
```

时间种子unordered_map

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x ^= x << 13;
        x ^= x >> 7;
        x ^= x << 17;
        return x;
    }
    size_t operator () (uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
unordered_map<uint64_t, int, custom_hash> safe_map;
```

数据结构

并查集

带权并查集

```
const int N=2e5+5;
int f[N],dis[N];
int find(int x)
{
    if(x!=f[x])
    {
        int t=f[x];
        f[x]=find(f[x]);
        dis[x]+=dis[t];
    }
    return f[x];
}
bool merge(int a,int b,int d)
{
    int ra=find(a),rb=find(b);
    d+=dis[a]-dis[b];
    if(ra==rb)
    {
        if(d!=0)
```

```
            return false;
        return true;
    }
    f[ra]=rb;
    dis[ra]-=d;
    return true;
}
```

树状数组

```
#define lowbit(x) ((x) & -(x))
int tree[N];
void update(int x,int d)
{
    while(x<=N)
    {
        tree[x]+=d;
        x+=lowbit(x);
    }
}
int sum(int x)
{
    int ans=0;
    while(x>0)
    {
        ans+=tree[x];
        x-=lowbit(x);
    }
    return ans;
}
```

二维树状数组

```
#define lowbit(x) ((x) & -(x))
void add(int x, int y, int d) {
    for (int i = x; i <= n; i += lowbit(i)) {
        for (int j = y; j <= m; j += lowbit(j)) {
            bit[i][j] += d;
        }
    }
}
int query(int x, int y) {
    int ret = 0;
    for (int i = x; i > 0; i -= lowbit(i)) {
        for (int j = y; j > 0; j -= lowbit(j)) {
            ret += bit[i][j];
        }
    }
    return ret;
}
```

线段树

区间修改查询区间和

```
struct SegmentTree{
    int a[N],tree[N<<2],tag[N<<2];
    int ls(int p){return p<<1;}
    int rs(int p){return p<<1|1;}
    void push_up(int p){
        tree[p]=tree[ls(p)]+tree[rs(p)];
    }
    void build(int p,int pl,int pr){
        tag[p]=0;
        if(pl==pr){
            tree[p]=a[pl];
            return;
        }
        int mid=(pl+pr)>>1;
        build(ls(p),pl,mid);
        build(rs(p),mid+1,pr);
        push_up(p);
    }
    void addtag(int p,int pl,int pr,int d){
        tag[p]+=d;
        tree[p]+=d*(pr-pl+1);
    }
    void push_down(int p,int pl,int pr){
        if(tag[p]){
            int mid=(pl+pr)>>1;
            addtag(ls(p),pl,mid,tag[p]);
            addtag(rs(p),mid+1,pr,tag[p]);
            tag[p]=0;
        }
    }
    void update(int L,int R,int p,int pl,int pr,int d){
        if(L<=pl&&pr<=R){
            addtag(p,pl,pr,d);
            return;
        }
        push_down(p,pl,pr);
        int mid=(pl+pr)>>1;
        if(L<=mid)
            update(L,R,ls(p),pl,mid,d);
        if(R>mid)
            update(L,R,rs(p),mid+1,pr,d);
        push_up(p);
    }
    int query(int L,int R,int p,int pl,int pr){
        if(L<=pl&&pr<=R)
            return tree[p];
        push_down(p,pl,pr);
        int mid=(pl+pr)>>1;
        int ans=0;
        if(L<=mid)
            ans+=query(L,R,ls(p),pl,mid);
        if(R>mid)
            ans+=query(L,R,rs(p),mid+1,pr);
```

```
        return ans;
    }
};
```

区间修改查询区间最值

```
#include <bits/stdc++.h>
using namespace std;
const int maxn=3e5+10;
const int inf =2e9;
struct Node{
    int l,r,res,tag;
};
struct SegmentTree{
    Node a[maxn*4];
    void tag_init(int i){
        a[i].tag=inf;
    }
    void tag_union(int fa,int i){
        if(a[fa].tag!=inf)a[i].tag=a[fa].tag;
    }
    void tag_cal(int i){
        if(a[i].tag!=inf)a[i].res=a[i].tag;
    }
    void pushdown(int i){
        tag_cal(i);
        if(a[i].l!=a[i].r){
            tag_union(i,i*2);
            tag_union(i,i*2+1);
        }
        tag_init(i);
    }
    void pushup(int i){
        if(a[i].l==a[i].r)return;
        pushdown(i*2);
        pushdown(i*2+1);
        a[i].res=min(a[i*2].res,a[i*2+1].res);
    }
    void build(int i,int l,int r){
        a[i].l=l,a[i].r=r;tag_init(i);
        if(l>=r)return;
        int mid=(l+r)/2;
        build(i*2,l,mid);
        build(i*2+1,mid+1,r);
    }
    void update(int i,int l,int r,int w){
        pushdown(i);
        if(a[i].r<l||a[i].l>r||l>r)return;
        if(a[i].l>=l&&a[i].r<=r){
            a[i].tag=w;
            return;
        }
        update(i*2,l,r,w);
        update(i*2+1,l,r,w);
        pushup(i);
    }
    int query(int i,int l,int r){
```

```
        pushdown(i);
        if(a[i].r<l||a[i].l>r||l>r)return inf;
        if(a[i].l>=l&&a[i].r<=r){
            return a[i].res;
        }
        return min(query(i*2,l,r),query(i*2+1,l,r));
    }
};
SegmentTree tri;
```

图论

树的直径

两次dfs

```
vector<pair<int,int>>arc[N];
void dfs(int x,int f,int d)
{
    dist[x]=d;
    fa[x]=fa;
    for(auto [it,i]:arc[x])
    {
        if(it==f)
            continue;
        dfs(it,x,d+i);
    }
}
```

树形dp

```
int dp[N],M=0;
bool vis[N];
void Dfs(int x,int f)
{
    vis[x]=true;
    for(auto [it,i]:arc[x])
    {
        if(vis[it]||it==f)
            continue;
        Dfs(it,x);
        M=max(M,dp[x]+dp[it]+i);
        dp[x]=max(dp[x],dp[it]+i);
    }
}
```

LCA

倍增

```
vector<int>arc[N];
int deep[N],fa[N][20];
void dfs(int x,int f)
{
    deep[x]=deep[f]+1;
    fa[x][0]=f;
    for(int i=1;i<=19;i++)
        fa[x][i]=fa[fa[x][i-1]][i-1];
    for(auto it:arc[x])
    {
        if(it==f)
            continue;
        dfs(it,x);
    }
}
int LCA(int x,int y)
{
    if(deep[x]<deep[y])
        swap(x,y);
    for(int i=19;i>=0;i--)
        if(deep[fa[x][i]]>=deep[y])
            x=fa[x][i];
    if(x==y)
        return x;
    for(int i=19;i>=0;i--)
        if(fa[x][i]!=fa[y][i])
            x=fa[x][i],y=fa[y][i];
    return fa[x][0];
}
```

tarjan

```
vector<pair<int,int>>arc[N];
int fa[N],ans[N];
bool vis[N];
int find(int x)
{
    return x==fa[x]?fa[x]:fa[x]=find(fa[x]);
}
void tarjan(int x,int f)
{
    vis[x]=true;
    for(auto [it,i]:arc[x])
    {
        if(it==f)
            continue;
        if(!vis[it])
        {
            tarjan(it,x);
            fa[it]=x;
        }
    }
    for(auto [it,i]:arc[x])
    {
        if(it==f)
```

```
            continue;
        if(vis[it])
            ans[i]=find(it);
    }
}
```

缩点

Tarjan缩点

```
const int N=1e4+5;
int a[N];//点权
vector<int>G[N];
int low[N],num[N],dfn,id[N];
int cnt,v[N];
stack<int>st;
void dfs(int x){
    low[x]=num[x]=++dfn;
    st.push(x);
    for(auto it:G[x]){
        if(!num[it]){
            dfs(it);
            low[x]=min(low[x],low[it]);
        }
        else if(!id[it])
            low[x]=min(low[x],num[it]);
    }
    if(low[x]==num[x]){
        cnt++;
        while(true){
            int tmp=st.top();
            st.pop();
            v[cnt]+=a[tmp];
            id[tmp]=cnt;
            if(x==tmp)
                break;
        }
    }
}
void Tarjan(int n){
    dfn=cnt=0;
    memset(low,0,sizeof(low));
    memset(num,0,sizeof(num));
    memset(id,0,sizeof(id));
    while(st.size())
        st.pop();
    for(int i=1;i<=n;i++)
        if(!num[i])
            dfs(i);
}
```

Korasaju缩点

```
const int N=1e4+5;
int a[N];
vector<int>G[N],rG[N];
vector<int>S;
bool vis[N];
int cnt,id[N];
void dfs1(int x){
    if(vis[x])
        return;
    vis[x]=true;
    for(auto it:G[x])
        dfs1(it);
    S.push_back(x);
}
int d[N],v[N];
void dfs2(int x){
    if(id[x])
        return;
    id[x]=cnt;
    v[cnt]+=a[x];
    for(auto it:rG[x])
        dfs2(it);
}
void Korasaju(int n){
    memset(vis,false,sizeof(vis));
    memset(id,0,sizeof(id));
    cnt=0;
    S.clear();
    for(int i=1;i<=n;i++)
        dfs1(i);
    reverse(S.begin(),S.end());
    for(auto it:S){
        if(!id[it]){
            cnt++;
            dfs2(it);
        }
    }
}
```

树链剖分

重链剖分

```
int sz[N],top[N],rk[N],id[N],son[N],fa[N],deep[N];
vector<int>G[N];
void dfs1(int x,int f){
    sz[x]=1;
    fa[x]=f;
    deep[x]=deep[f]+1;
    for(auto it:G[x]){
        if(it==f)
            continue;
        dfs1(it,x);
        sz[x]+=sz[it];
        if(!son[x]||sz[son[x]]<sz[it])
```

```
            son[x]=it;
    }
}
void dfs2(int x,int topx){
    top[x]=topx;
    id[x]=++num;
    rk[num]=x;
    if(!son[x])
        return;
    dfs2(son[x],topx);
    for(auto it:G[x]){
        if(it!=fa[x]&&it!=son[x])
            dfs2(it,it);
    }
}
```

```
dfs1(root,0);
dfs2(root,root);
```

树上区间修改/查询

```
struct SegmentTree{
    int a[N],tree[N<<2],tag[N<<2];
    int ls(int p){return p<<1;}
    int rs(int p){return p<<1|1;}
    void push_up(int p){
        tree[p]=tree[ls(p)]+tree[rs(p)];
        tree[p]%=mod;
    }
    void build(int p,int pl,int pr){
        tag[p]=0;
        if(pl==pr){
            tree[p]=a[rk[pl]];
            return;
        }
        int mid=(pl+pr)>>1;
        build(ls(p),pl,mid);
        build(rs(p),mid+1,pr);
        push_up(p);
    }
    void addtag(int p,int pl,int pr,int d){
        tag[p]+=d;
        tree[p]+=d*(pr-pl+1);
        tree[p]%=mod;
    }
    void push_down(int p,int pl,int pr){
        if(tag[p]){
            int mid=(pl+pr)>>1;
            addtag(ls(p),pl,mid,tag[p]);
            addtag(rs(p),mid+1,pr,tag[p]);
            tag[p]=0;
        }
    }
    void update(int L,int R,int p,int pl,int pr,int d){
        if(L<=pl&&pr<=R){
            addtag(p,pl,pr,d);
```

```
            return;
        }
        push_down(p,pl,pr);
        int mid=(pl+pr)>>1;
        if(L<=mid)
            update(L,R,ls(p),pl,mid,d);
        if(R>mid)
            update(L,R,rs(p),mid+1,pr,d);
        push_up(p);
    }
    int query(int L,int R,int p,int pl,int pr){
        if(L<=pl&&pr<=R)
            return tree[p];
        push_down(p,pl,pr);
        int mid=(pl+pr)>>1;
        int ans=0;
        if(L<=mid)
            ans+=query(L,R,ls(p),pl,mid);
        if(R>mid)
            ans+=query(L,R,rs(p),mid+1,pr);
        return ans;
    }
}Tr;
void add_range(int x,int y,int d){
    while(top[x]!=top[y]){
        if(deep[top[x]]<deep[top[y]])
            swap(x,y);
        Tr.update(id[top[x]],id[x],1,1,n,d);
        x=fa[top[x]];
    }
    if(deep[x]>deep[y])
        swap(x,y);
    Tr.update(id[x],id[y],1,1,n,d);
}
int query_range(int x,int y){
    int ans=0;
    while(top[x]!=top[y]){
        if(deep[top[x]]<deep[top[y]])
            swap(x,y);
        ans+=Tr.query(id[top[x]],id[x],1,1,n);
        ans%=mod;
        x=fa[top[x]];
    }
    if(deep[x]>deep[y])
        swap(x,y);
    ans+=Tr.query(id[x],id[y],1,1,n);
    return ans%mod;
}
void add_tree(int x,int d){
    Tr.update(id[x],id[x]+sz[x]-1,1,1,n,d);
}
int query_tree(int x){
    return Tr.query(id[x],id[x]+sz[x]-1,1,1,n)%mod;
}
```

网络流

二分图匹配

```
vector<int> G[N];
int Nx,Ny,k; //Nx,Ny是两个集合的大小；k是边数

int Mx[N],My[N];
int dx[N],dy[N];
int dis,u,v;
bool used[N];
bool searchP(){
    queue<int> Q;
    dis = INF;
    memset(dx,-1,sizeof(dx));
    memset(dy,-1,sizeof(dy));
    for(int i = 0;i < Nx;++i)
    if(Mx[i] == -1)    Q.push(i), dx[i] = 0;
    while(!Q.empty()){
        int u = Q.front();Q.pop();
        if(dx[u] > dis) break;
        int sz = G[u].size();
        for(int i = 0;i < sz;++i){
            int v = G[u][i];
            if(dy[v] == -1) {
                dy[v] = dx[u] + 1;
                if(My[v] == -1) dis = dy[v];
                else dx[My[v]] = dy[v] + 1, Q.push(My[v]);
            }
        }
    }
    return dis != INF;
}
bool DFS(int u){
    int sz = G[u].size();
    for(int i = 0;i < sz;++i){
        int v = G[u][i];
        if(!used[v] && dy[v] == dx[u] + 1){
            used[v] = true;
            if(My[v] != -1 && dy[v] == dis) continue;
            if(My[v] == -1 || DFS(My[v])){
                My[v] = u;
                Mx[u] = v;
                return true;
            }
        }
    }
    return false;
}
int MaxMatch(){
    int res = 0;
    memset(Mx,-1,sizeof(Mx));
    memset(My,-1,sizeof(My));
    while(searchP()){
        memset(used,false,sizeof(used));
        for(int i = 0;i < Nx;++i)
        if(Mx[i] == -1 && DFS(i)) ++res;
```

```
    }
    return res;
}
int main(){
    read(Nx);read(Ny);read(k);
    while(k--){read(u);read(v);if(v<=Ny) G[u-1].push_back(v-1);}
    printf("%d\n",MaxMatch());
}
```

数学

高斯消元

例题 洛谷 P3389

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=105;
double a[N][N];
double eps=1e-7;
int main() {
    ios::sync_with_stdio(false);cin.tie(nullptr);
    int n;
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n+1;j++)
            scanf("%lf",&a[i][j]);
    }
    for(int i=1;i<=n;i++)
    {
        int M=0,Mi=0;
        for(int j=i;j<=n;j++)
        {
            if(fabs(a[j][i])>M)
            {
                M=fabs(a[j][i]);
                Mi=j;
            }
        }
        for(int j=i;j<=n+1;j++)
            swap(a[Mi][j],a[i][j]);
        if(fabs(a[i][i])<eps)
        {
            printf("No Solution\n");
            return 0;
        }
        for(int j=n+1;j>=i;j--)
            a[i][j]/=a[i][i];
        for(int j=1;j<=n;j++)
        {
            if(j==i)
                continue;
            double temp=a[j][i]/a[i][i];
            for(int k=i;k<=n+1;k++)
```

```
            {
                a[j][k]-=temp*a[i][k];
            }
        }
    }
    for(int i=1;i<=n;i++)
        printf("%.2lf\n",a[i][n+1]);
    return 0;
}
```

GCD

欧几里得算法

```
int gcd(int a,int b)
{
    return b?gcd(b,a%b):a;
}
```

更相损减术

$$\gcd(a,b)=\gcd(b,a-b)=\gcd(a,a-b)$$

```
int gcd(int a,int b)
{
    while(a!=b)
    {
        if(a>b) a=a-b;
        else    b=b-a;
    }
    return a;
}
```

LCM

```
int lcm(int a,int b)
{
    return a/gcd(a,b)*b;
}
```

扩展欧几里得

返回d=gcd(a,b);以及ax+by=d的特解x,y

```
ll extend_gcd(ll a,ll b,ll &x,ll &y)
{
    if(b==0){x=1;y=0;return a;}
    ll d=extend_gcd(b,a%b,y,x);
    y-=a/b*x;
    return d;
}
```

逆元

扩展欧几里得

```
ll mod_inverse(ll a,ll m)
{
    ll x,y;
    extend_gcd(a,m,x,y);
    return (x%m+m)%m;
}
```

快速幂

```
ll mod_inverse(ll a,ll m)
{
    return qpow(a,m-2,m);
}
```

递推

```
void mod_inverse(ll n,ll p)
{
    inv[1]=1;
    for(int i=1;i<=n;i++)
        inv[i]=(ll)(p-p/i)*inv[p%i]%p;
}
```

欧拉筛

筛素数

```
bitset<N>vis;
void get_prime()
{
    for(int i=2;i<N;i++)
    {
        if(!vis[i])
        {
            vis[i]=true;
            p.push_back(i);
            for(int j=2;i*j<N;j++)
                vis[i*j]=true;
        }
    }
}
```

筛欧拉函数

```
bool vis[N];
int phi[N];
vector<int>p;
void get_phi()
{
    phi[1]=1;
    for(int i=2;i<N;i++)
```

```
    {
        if(!vis[i])
        {
            vis[i]=true;
            p.push_back(i);
            phi[i]=i-1;
        }
        for(auto p:p)
        {
            if(i*p>=N)
                break;
            vis[i*p]=true;
            if(i%p==0)
            {
                phi[i*p]=p*phi[i];
                break;
            }
            phi[i*p]=phi[i]*phi[p];
        }
    }
}
```

筛约数和

```
vector<int>p;
int phi[N],sig[N],num[N];
bool vis[N];
void init()
{
    phi[1]=sig[1]=1;
    for(int i=2;i<N;i++)
    {
        if(!vis[i])
        {
            vis[i]=true;
            p.push_back(i);
            phi[i]=i-1;
            sig[i]=num[i]=i+1;
        }
        for(auto j:p)
        {
            if(i*j>=N)
                break;
            vis[i*j]=true;
            if(i%j==0)
            {
                phi[i*j]=phi[i]*j;
                num[i*j]=num[i]*j+1;
                sig[i*j]=sig[i]/num[i]*num[i*j];
                break;
            }
            phi[i*j]=phi[i]*phi[j];
            num[i*j]=1+j;
            sig[i*j]=sig[i]*sig[j];
        }
    }
}
```

素数判定

Miller Rabin

```
bool is_prime(int x)
{
    if(x<3)
        return x==2;
    if(x%2==0)
        return false;
    int A[]={2,325,9375,28178,450775,9780504,1795265022},d=x-1,r=0;
    while(d%2==0)
        d>>=1,r++;
    for(auto a:A)
    {
        int v=qpow(a,d,x);
        if(v<=1||v==x-1)
            continue;
        for(int i=0;i<r;i++)
        {
            v=(__int128_t)v*v%x;
            if(v==x-1&&i!=r-1)
            {
                v=1;break;
            }
            if(v==1)
                return false;
        }
        if(v!=1)
            return false;
    }
    return true;
}
```

质因数分解

Pollard Rho

```
mt19937 eng(time(0));
int randint(int a, int b)
{
    uniform_int_distribution<int> dis(a, b);
    return dis(eng);
}
int Pollard_Rho(int n){
    if(n==4)    return 2;
    if(is_prime(n)) return n;
    while(true){
        int c=randint(1,n-1);
        auto f=[=](int x){return ((__int128_t)x*x+c)%n;};
        int t=f(0),r=f(f(0));
        while(t!=r){
            int d=__gcd(abs(t-r),n);
            if(d>1) return d;
            t=f(t),r=f(f(r));
        }
```

```
    }
}
```

离散对数

bsgs

```
ll BSGS(ll a, ll b, ll m)
{
    static unordered_map<ll, ll> hs;
    hs.clear();
    ll cur = 1, t = sqrt(m) + 1;
    for (int B = 1; B <= t; ++B)
    {
        (cur *= a) %= m;
        hs[b * cur % m] = B; // 哈希表中存B的值
    }
    ll now = cur; // 此时cur = a^t
    for (int A = 1; A <= t; ++A)
    {
        auto it = hs.find(now);
        if (it != hs.end())
            return A * t - it->second;
        (now *= cur) %= m;
    }
    return -1; // 没有找到，无解
}
```

扩展bsgs

```
// 修改版的BSGS，额外带一个系数
ll BSGS(ll a, ll b, ll m, ll k = 1)
{
    static unordered_map<ll, ll> hs;
    hs.clear();
    ll cur = 1, t = sqrt(m) + 1;
    for (int B = 1; B <= t; ++B)
    {
        (cur *= a) %= m;
        hs[b * cur % m] = B; // 哈希表中存B的值
    }
    ll now = cur * k % m;
    for (int A = 1; A <= t; ++A)
    {
        auto it = hs.find(now);
        if (it != hs.end()) return A * t - it->second;
        (now *= cur) %= m;
    }
    return -INF; // 这里因为要多次加1，要返回更小的负数
}
ll exBSGS(ll a, ll b, ll m, ll k = 1)
{
    ll A = a %= m, B = b %= m, M = m;
    if (b == 1) return 0;
    ll cur = 1 % m;
    for (int i = 0;; i++)
```

```
    {
        if (cur == B) return i;
        cur = cur * A % M;
        ll d = gcd(a, m);
        if (b % d) return -INF;
        if (d == 1) return BSGS(a, b, m, k * a % m) + i + 1;
        k = k * a / d % m, b /= d, m /= d; // 相当于在递归求解exBSGS(a, b / d, m / d, k * a / d % m)
    }
}
```

组合数

预处理阶乘

```
void init(int n)
{
    fac[0]=1;
    for(int i=1;i<=n;i++)
        fac[i]=fac[i-1]*i%mod;
    rev[n]=qpow(fac[n],mod-2,mod);//n must be less than mod
    for(int i=n;i>=1;i--)
        rev[i-1]=rev[i]*i%mod;
    assert(rev[0]==1);
}
```

Lucas

```
int C(int n,int m,int p)
{
    if(m>n)
        return 0;
    return fac[n]*rev[m]%p*rev[n-m]%p;
}
int Lucas(int n,int m,int p)
{
    if(m==0)
        return 1;
    return C(n%p,m%p,p)*Lucas(n/p,m/p,p)%p;
}
```

类欧几里得

$$f(x)=\frac{ax+b}{c},\text{求}x\in[0,n]\text{且}x\in Z\text{时},f(x)\text{下的整点个数之和}$$

$$\sum_{i=0}^{n}\lfloor\frac{ai+b}{c}\rfloor$$

```
ll f(ll a, ll b, ll c, ll n) {
    if (!a) return b / c * (n + 1);
    if (a >= c || b >= c)
        return f(a % c, b % c, c, n) + (a / c) * n * (n + 1) / 2 + (b / c) * (n
+ 1);
    ll m = (a * n + b) / c;
    return n * m - f(c, c - b - 1, a, m - 1);
}
```

BM线性递推

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define int long long
typedef unsigned long long ull;
#define dmp(x) cerr<<"DEBUG"<<__LINE__<<":"<<#x<<" "<<x<<endl
const int INF=0x3f3f3f3f;
typedef pair<int,int> pii;
const int mod=1e9+7;
int powmod(int a,int b){
    int res=1;a%=mod;
    assert(b>=0);
    while(b)
    {
        if(b&1) res=res*a%mod;
        a=a*a%mod;
    }
    return res;
}
int n;
namespace linear_seq{
    const int N=10010;
    int res[N],base[N],_c[N],_md[N];
    vector<int>Md;
    void mul(int *a,int *b,int k){
        for(int i=0;i<k+k;i++)  _c[i]=0;
        for(int i=0;i<k;i++)
            if(a[i])
                for(int j=0;j<k;j++)
                    _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
        for(int i=k+k-1;i>=k;i--)
            if(_c[i])
                for(int j=0;j<(int)Md.size();j++)
                    _c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])%mod;
        for(int i=0;i<k;i++)
            a[i]=_c[i];
    }
    int solve(int n,vector<int>a,vector<int>b){
        int ans=0,pnt=0;
        int k=(int)a.size();
        assert(a.size()==b.size());
        for(int i=0;i<k;i++)
            _md[k-1-i]=-a[i];
        _md[k]=1;
        Md.clear();
```

```
        for(int i=0;i<k;i++)
            if(_md[i]!=0)
                Md.push_back(i);
        for(int i=0;i<k;i++)
            res[i]=base[i]=0;
        res[0]=1;
        while((1ll<<pnt)<=n)
            pnt++;
        for(int p=pnt;p>=0;p--){
            mul(res,res,k);
            if((n>>p)&1){
                for(int i=k-1;i>=0;i--)
                    res[i+1]=res[i];
                res[0]=0;
                for(int j=0;j<(int)Md.size();j++)
                    res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
            }
        }
        for(int i=0;i<k;i++)
            ans=(ans+res[i]*b[i])%mod;
        if(ans<0)
            ans+=mod;
        return ans;
    }
    vector<int> BM(vector<int> s){
        vector<int> C(1,1),B(1,1);
        int L=0,m=1,b=1;
        for(int i=0;i<(int)s.size();i++){
            int d=0;
            for(int i=0;i<L+1;i++)
                d=(d+C[i]*s[n-i])%mod;
            if(d==0)
                ++m;
            else if(2*L<=n){
                vector<int> T=C;
                int c=mod-d*powmod(b,mod-2)%mod;
                while(C.size()<B.size()+m)
                    C.push_back(0);
                for(int i=0;i<B.size();i++)
                    C[i+m]=(C[i+m]+c*B[i])%mod;
                L=n+1-L;B=T;b=d;m=1;
            }
            else{
                int c=mod-d*powmod(b,mod-2)%mod;
                while(C.size()<B.size()+m)
                    C.push_back(0);
                for(int i=0;i<B.size();i++)
                    C[i+m]=(C[i+m]+c*B[i])%mod;
                ++m;
            }
        }
        return C;
    }
    int gao(vector<int>a,int n){
        vector<int> c=BM(a);
        c.erase(c.begin());
        for(int i=0;i<c.size();i++)
            c[i]=(mod-c[i])%mod;
```

```
        return solve(n,c,vector<int>(a.begin(),a.begin()+(int)c.size()));
    }
}
signed main() {
    ios::sync_with_stdio(false);cin.tie(0);
    vector<int>v;
    v.push_back(2);
    v.push_back(24);
    v.push_back(96);
    v.push_back(416);
    v.push_back(1536);
    v.push_back(5504);
    v.push_back(18944);
    v.push_back(64000);
    v.push_back(212992);
    v.push_back(702464);
    cin>>n;
    cout<<linear_seq::gao(v,n-1)<<"\n";
    return 0;
}
```

字符串

KMP

前缀数组

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j]) j = pi[j - 1];
        if (s[i] == s[j]) j++;
        pi[i] = j;
    }
    return pi;
}
```

模式匹配

```
vector<int> find_occurrences(string text, string pattern) {
    string cur = pattern + '#' + text;
    int sz1 = text.size(), sz2 = pattern.size();
    vector<int> v;
    vector<int> lps = prefix_function(cur);
    for (int i = sz2 + 1; i <= sz1 + sz2; i++) {
        if (lps[i] == sz2)
        v.push_back(i - 2 * sz2);
    }
    return v;
}
```

回文串

manacher

p[i]是以i为中心的最长回文串长度

```
int p[N<<1];
void change(string a)
{
    s+='$';s+='#';
    for(auto it:a)
    {
        s+=it;s+='#';
    }
    s+='&';
}
void manacher()
{
    int n=s.length();
    int R=0,C;
    for(int i=1;i<n;i++)
    {
        if(i<R)
            p[i]=min(p[C]+C-i,p[(C<<1)-i]);
        else
            p[i]=1;
        while(s[i+p[i]]==s[i-p[i]])
            p[i]++;
        if(p[i]+i>R)
        {
            R=p[i]+i;
            C=i;
        }
    }
}
```

字典树

```
struct Trie{
    int cnt=0,ch[N][26],sz[N];//sz[N]是以这个点结尾的字符串数量
    int newNode(){
        cnt++;
        sz[cnt]=0;
        memset(ch[cnt],0,sizeof(ch[cnt]));
        return cnt;
    }
    void add(string s){
        int now=0;
        for(auto it:s){
            int &c=ch[now][it-'a'];
            if(!c)
                c=newNode();
            now=c;
        }
        sz[now]++;
    }
```

```
    int find(string s){
        int now=0;
        for(auto it:s){
            now=ch[now][it-'a'];
            if(!now)
                return 0;
        }
        return sz[now];
    }
};
```

双哈

```
#define mp make_pair
#define fi first
#define se second
using ll = long long;
typedef pair<int, int> hashv;
const ll mod1 = 1e9 + 7;
const ll mod2 = 1e9 + 9;

hashv base = mp(13331, 2333);
hashv operator + (hashv a, hashv b) {
    int c1 = a.fi + b.fi, c2 = a.se + b.se;
    if(c1 >= mod1) c1 -= mod1;
    if(c2 >= mod2) c2 -= mod2;
    return mp(c1, c2);
}

hashv operator - (hashv a, hashv b) {
    int c1 = a.fi - b.fi, c2 = a.se - b.se;
    if(c1 < 0) c1 += mod1;
    if(c2 < 0) c2 += mod2;
    return mp(c1,c2);
}

hashv operator * (hashv a, hashv b) {
    return mp(1ll*a.fi*b.fi%mod1, 1ll*a.se*b.se%mod2);
}
```

计算几何

```
typedef pair<double,double>pll;
pll operator+(pll x,pll y){
    return {x.first+y.first,x.second+y.second};
}
pll operator-(pll x,pll y){
    return {x.first-y.first,x.second-y.second};
}
pll operator*(pll x,double k){
    return {x.first*k,x.second*k};
}
pll operator/(pll x,double k){
    return {x.first/k,x.second/k};
```

```
}
double len(pll x){
    return hypot(x.first,x.second);
}
double Dot(const pdd &a,const pdd &b){
    return a.first*b.first+a.second*b.second;
}
double Cross(const pdd &a,const pdd &b){
    return a.first*b.second-a.second*b.first;
}
```

实数精度

```
const double pi = acos(-1.0);           //圆周率，精确到15位小数：3.141592653589793
const double eps = 1e-8;                //偏差值，有时用1e-10，但是要注意精度
int sgn(double x){                      //判断x的大小
    if(fabs(x) < eps)  return 0;        //x==0，返回0
    else return x<0?-1:1;               //x<0返回-1，x>0返回1
}
int dcmp(double x, double y){           //比较两个浮点数
    if(fabs(x - y) < eps)   return 0;   //x==y，返回0
    else return x<y ?-1:1;              //x<y返回-1，x>y返回1
}
```

点

```
struct Point{
    double x,y;
    Point(){}
    Point(double x,double y):x(x),y(y){}
    Point operator + (Point B){return Point(x+B.x,y+B.y);}
    Point operator - (Point B){return Point(x-B.x,y-B.y);}
    Point operator * (double k){return Point(x*k,y*k);}
    Point operator / (double k){return Point(x/k,y/k);}
    bool operator == (Point B){return sgn(x-B.x)==0 && sgn(y-B.y)==0;}
};
double Distance(Point A, Point B){ return hypot(A.x-B.x,A.y-B.y); }
```

向量

```
typedef Point Vector;
double Dot(Vector A,Vector B){ return A.x*B.x + A.y*B.y; }
double Len(Vector A){return sqrt(Dot(A,A));}
double Len2(Vector A){return Dot(A,A);}
double Angle(Vector A,Vector B){return acos(Dot(A,B)/Len(A)/Len(B));}
double Cross(Vector A,Vector B){return A.x*B.y - A.y*B.x;}
double Area2(Point A,Point B,Point C){ return Cross(B-A, C-A);}
Vector Rotate(Vector A, double rad){  //逆时针旋转rad角度
    return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
}
Vector Normal(Vector A){return Vector(-A.y/Len(A), A.x/Len(A));} //求单位法向量
bool Parallel(Vector A, Vector B){return sgn(Cross(A,B)) == 0;}  //返回true表示平行或重合
```

线

```
struct Line{
    Point p1,p2;                    //（1）线上的两个点
    Line(){}
    Line(Point p1,Point p2):p1(p1),p2(p2){}
    Line(Point p,double angle){    //（4）根据一个点和倾斜角 angle 确定直
线,0<=angle<pi
        p1 = p;
        if(sgn(angle - pi/2) == 0){p2 = (p1 + Point(0,1));}
        else{p2 = (p1 + Point(1,tan(angle)));}
    }
    Line(double a,double b,double c){     //（2）ax+by+c=0
        if(sgn(a) == 0){
            p1 = Point(0,-c/b);
            p2 = Point(1,-c/b);
        }
        else if(sgn(b) == 0){
            p1 = Point(-c/a,0);
            p2 = Point(-c/a,1);
        }
        else{
            p1 = Point(0,-c/b);
            p2 = Point(1,(-c-a)/b);
        }
    }
};
typedef Line Segment;
int Point_line_relation(Point p, Line v){
    int c = sgn(Cross(p-v.p1,v.p2-v.p1));
    if(c < 0)return 1;              //1：p在v的左边
    if(c > 0)return 2;              //2：p在v的右边
    return 0;                       //0：p在v上
}
bool Point_on_seg(Point p, Line v){ //点和线段：0 点不在线段v上；1 点在线段v上
    return sgn(Cross(p-v.p1, v.p2-v.p1)) == 0 && sgn(Dot(p - v.p1,p - v.p2)) <=
0;
}
double Dis_point_line(Point p, Line v){
    return fabs(Cross(p-v.p1,v.p2-v.p1))/Distance(v.p1,v.p2);
}
Point Point_line_proj(Point p, Line v){
    double k = Dot(v.p2-v.p1,p-v.p1)/Len2(v.p2-v.p1);
    return v.p1+(v.p2-v.p1)*k;
}
Point Point_line_symmetry(Point p, Line v){
    Point q = Point_line_proj(p,v);
    return Point(2*q.x-p.x,2*q.y-p.y);
}
double Dis_point_seg(Point p, Segment v){
    if(sgn(Dot(p- v.p1,v.p2-v.p1))<0 || sgn(Dot(p- v.p2,v.p1-v.p2))<0)
        return min(Distance(p,v.p1),Distance(p,v.p2));
    return Dis_point_line(p,v);           //点的投影在线段上
}
int Line_relation(Line v1, Line v2){
    if(sgn(Cross(v1.p2-v1.p1,v2.p2-v2.p1)) == 0){
        if(Point_line_relation(v1.p1,v2)==0) return 1;  //1 重合
```

```
        else return 0;                                      //0 平行
    }
    return 2;                                               //2 相交
}
Point Cross_point(Point a,Point b,Point c,Point d){  //Line1:ab,  Line2:cd
    double s1 = Cross(b-a,c-a);
    double s2 = Cross(b-a,d-a);                         //叉积有正负
    return Point(c.x*s2-d.x*s1,c.y*s2-d.y*s1)/(s2-s1);
}
bool Cross_segment(Point a,Point b,Point c,Point d){    //Line1:ab,  Line2:cd
    double c1 = Cross(b-a,c-a),c2=Cross(b-a,d-a);
    double d1 = Cross(d-c,a-c),d2=Cross(d-c,b-c);
    return sgn(c1)*sgn(c2) < 0 && sgn(d1)*sgn(d2) < 0;  //1相交；0不相交
}
```

多边形

```
int Point_in_polygon(Point pt,Point *p,int n){  //点pt，多边形Point *p
    for(int i = 0;i < n;i++){                   //3：点在多边形的顶点上
        if(p[i] == pt)  return 3;
    }
    for(int i = 0;i < n;i++){                   //2：点在多边形的边上
        Line v=Line(p[i],p[(i+1)%n]);
        if(Point_on_seg(pt,v)) return 2;
    }
    int num = 0;
    for(int i = 0;i < n;i++){
        int j = (i+1)% n;
        int c = sgn(Cross(pt-p[j],p[i]-p[j]));
        int u = sgn(p[i].y - pt.y);
        int v = sgn(p[j].y - pt.y);
        if(c > 0 && u < 0 && v >=0) num++;
        if(c < 0 && u >=0 && v < 0) num--;
    }
    return num != 0;                            //1：点在内部；0：点在外部
}
double Polygon_area(Point *p, int n){    //Point *p表示多边形
    double area = 0;
    for(int i = 0;i < n;i++)
        area += Cross(p[i],p[(i+1)%n]);
    return area/2;                      //面积有正负，返回时不能简单地取绝对值
}
Point Polygon_center(Point *p, int n){        //求多边形重心
    Point ans(0,0);
    if(Polygon_area(p,n)==0) return ans;
    for(int i = 0;i < n;i++)
        ans = ans+(p[i]+p[(i+1)%n])*Cross(p[i],p[(i+1)%n]);
    return ans/Polygon_area(p,n)/6;
}
```

凸包

```
//Convex_hull()求凸包。凸包顶点放在ch中，返回值是凸包的顶点数
int Convex_hull(Point *p,int n,Point *ch){
    n = unique(p,p+n)-p;    //去除重复点
    sort(p,p+n);            //对点排序：按x从小到大排序，如果x相同，按y排序
    int v=0;
```

```
    //求下凸包。如果p[i]是右拐弯的，这个点不在凸包上，往回退
    for(int i=0;i<n;i++){
        while(v>1 && sgn(Cross(ch[v-1]-ch[v-2],p[i]-ch[v-1]))<=0) //把后面ch[v-1]
改成ch[v-2]也行
            v--;
        ch[v++]=p[i];
    }
    int j=v;
    //求上凸包
    for(int i=n-2;i>=0;i--){
        while(v>j && sgn(Cross(ch[v-1]-ch[v-2],p[i]-ch[v-1]))<=0) //把后面ch[v-1]
改成ch[v-2]也行
            v--;
        ch[v++]=p[i];
    }
    if(n>1) v--;
    return v;                       //返回值v是凸包的顶点数
}
Point p[N],ch[N];                   //输入点是p[]，计算得到的凸包顶点放在ch[]中
```

最近点对

使用前先sort(p,p+n,cmpxy)

```
bool cmpxy(Point A,Point B){return sgn(A.x-B.x)<0||(sgn(A.x-B.x)==0&&sgn(A.y-
B.y)<0);}
bool cmpy(Point A,Point B){return sgn(A.y-B.y)<0;}
double Distance(Point A, Point B){ return hypot(A.x-B.x,A.y-B.y); }
double Closest_Pair(int left,int right){
    double dis = INF;
    if(left == right) return dis;            //只剩1个点
    if(left + 1 == right) return Distance(p[left], p[right]);//只剩2个点
    int mid = (left+right)/2;                //分治
    double d1 = Closest_Pair(left,mid);      //求s1内的最近点对
    double d2 = Closest_Pair(mid+1,right);   //求s2内的最近点对
    dis = min(d1,d2);
    int k = 0;
    for(int i=left;i<=right;i++)             //在s1和s2中间附近找可能的最小点对
        if(fabs(p[mid].x - p[i].x) <= dis)   //按x坐标来找
            tmp_p[k++] = p[i];
    sort(tmp_p,tmp_p+k,cmpy);          //按y坐标排序，用于剪枝。这里不能按x坐标排序
    for(int i=0;i<k;i++)
        for(int j=i+1;j<k;j++){
            if(tmp_p[j].y - tmp_p[i].y >= dis)  break;    //剪枝
            dis = min(dis,Distance(tmp_p[i],tmp_p[j]));
        }
    return dis;  //返回最小距离
}
```

旋转卡壳

```
double Rotating_Calipers(Point *p,int n){
    double ans=0;
    int j=2;
    for(int i=0;i<n;i++){
        while(fabs(Cross(p[j]-p[i],p[j]-p[(i+1)%n])<fabs(Cross(p[(j+1)%n]-
p[i],p[(j+1)%n]-p[(i+1)%n])))
            j=(j+1)%n;
        ans=max(ans,max(Distance(p[i],p[j]),Distance(p[(i+1)%n],p[j])));
    }
    return ans;
}
```

半平面交

```
struct Line{    //半平面的表示
    Point p;      //直线上一个点
    Vector v;     //方向向量，它的左边是半平面
    double ang;   //极角，从x正半轴旋转到v的角度
    Line(){};
    Line(Point p, Vector v):p(p),v(v){ang = atan2(v.y, v.x);}
    bool operator < (Line &L){return ang < L.ang;}     //用于排序
};
//点p在线L左边，即点p在线L在外面：
bool OnLeft(Line L,Point p){return sgn(Cross(L.v,p-L.p))>0;}
Point Cross_point(Line a,Line b){    //两直线交点
    Vector u=a.p-b.p;
    double t=Cross(b.v,u)/Cross(a.v,b.v);
    return a.p+a.v*t;
}
vector<Point> HPI(vector<Line> L){     //求半平面交，返回凸多边形
    int n=L.size();
    sort(L.begin(),L.end());            //将所有半平面按照极角排序。
    int first,last;                     //指向双端队列的第一个和最后一个元素
    vector<Point> p(n);                 //两个相邻半平面的交点
    vector<Line> q(n);                  //双端队列
    vector<Point> ans;                  //半平面交形成的凸包
    q[first=last=0]=L[0];
    for(int i=1;i<n;i++){
        //情况1：删除尾部的半平面
        while(first<last && !OnLeft(L[i], p[last-1])) last--;
        //情况2：删除首部的半平面：
        while(first<last && !OnLeft(L[i], p[first]))  first++;
        q[++last]=L[i];     //将当前的半平面加入双端队列尾部
        //极角相同的两个半平面，保留左边：
        if(fabs(Cross(q[last].v,q[last-1].v)) < eps){
        last--;
            if(OnLeft(q[last],L[i].p)) q[last]=L[i];
        }
        //计算队列尾部半平面交点：
        if(first<last) p[last-1]=Cross_point(q[last-1],q[last]);
    }
    //情况3：删除队列尾部的无用半平面
    while(first<last && !OnLeft(q[first],p[last-1])) last--;
    if(last-first<=1) return ans;   //空集
    p[last]=Cross_point(q[last],q[first]);  //计算队列首尾部的交点。
    for(int i=first;i<=last;i++)  ans.push_back(p[i]);   //复制。
    return ans;                 //返回凸多边形
```

```
}
```

圆

```
struct Circle{ //圆
    Point c;      //圆心
    double r;     //半径
    Circle(){}
    Circle(Point c,double r):c(c),r(r){}
    Circle(double x,double y,double _r){c=Point(x,y);r = _r;}
};
int Point_circle_relation(Point p, Circle C){ //点和圆的关系
    double dst = Distance(p,C.c);
    if(sgn(dst - C.r) < 0) return 0;        //0 点在圆内
    if(sgn(dst - C.r) ==0) return 1;        //1 圆上
    return 2;                                //2 圆外
}
int Line_circle_relation(Line v,Circle C){ //直线和圆的位置关系
    double dst = Dis_point_line(C.c,v);
    if(sgn(dst-C.r) < 0) return 0;     //0 直线和圆相交
    if(sgn(dst-C.r) ==0) return 1;     //1 直线和圆相切
    return 2;                               //2 直线在圆外
}
int Seg_circle_relation(Segment v,Circle C){ //线段和圆的位置关系
    double dst = Dis_point_seg(C.c,v);
    if(sgn(dst-C.r) < 0) return 0;      //0线段在圆内
    if(sgn(dst-C.r) ==0) return 1;      //1线段和圆相切
    return 2;                           //2线段在圆外
}
//pa, pb是交点。返回值是交点个数
int Line_cross_circle(Line v,Circle C,Point &pa,Point &pb){ //直线和圆的交点
    if(Line_circle_relation(v, C)==2)  return 0;//无交点
    Point q = Point_line_proj(C.c,v);          //圆心在直线上的投影点
    double d = Dis_point_line(C.c,v);          //圆心到直线的距离
    double k = sqrt(C.r*C.r-d*d);
    if(sgn(k) == 0){                            //1个交点，直线和圆相切
        pa = q; pb = q; return 1;
    }
    Point n=(v.p2-v.p1)/ Len(v.p2-v.p1);       //单位向量
    pa = q + n*k;  pb = q - n*k;
    return 2;                                  //2个交点
}
```

最小圆覆盖

```
Point circle_center(const Point a, const Point b, const Point c){ //圆上三点定圆心
    Point center;
    double a1=b.x-a.x, b1=b.y-a.y, c1=(a1*a1+b1*b1)/2;
    double a2=c.x-a.x, b2=c.y-a.y, c2=(a2*a2+b2*b2)/2;
    double d =a1*b2-a2*b1;
    center.x =a.x+(c1*b2-c2*b1)/d;
    center.y =a.y+(a1*c2-a2*c1)/d;
    return center;
}
void min_cover_circle(Point *p, int n, Point &c, double &r){ //最小圆覆盖
    random_shuffle(p, p + n);              //随机函数，打乱所有点。这一步很重要
    c=p[0]; r=0;                           //从第1个点p0开始。圆心为p0，半径为0
```

```
    for(int i=1;i<n;i++)                    //扩展所有点
        if(sgn(Distance(p[i],c)-r)>0){      //点pi在圆外部
            c=p[i]; r=0;                    //重新设置圆心为pi，半径为0
            for(int j=0;j<i;j++)            //重新检查前面所有的点。
                if(sgn(Distance(p[j],c)-r)>0){    //两点定圆
                    c.x=(p[i].x + p[j].x)/2;
                    c.y=(p[i].y + p[j].y)/2;
                    r=Distance(p[j],c);
                    for(int k=0;k<j;k++)
                        if (sgn(Distance(p[k],c)-r)>0){    //两点不能定圆，就三点定圆
                            c=circle_center(p[i],p[j],p[k]);
                            r=Distance(p[i], c);
                        }
                }
        }
}
```

多项式

FFT

```
#include<bits/stdc++.h>
using namespace std;

const double PI = acos(-1.0);
const double eps=1e-4;

struct Complex
{
    double x, y;
    Complex(double x = 0.0, double y = 0.0):x(x),y(y){}
    Complex operator-(const Complex &b) const{return Complex(x - b.x, y - b.y);}
    Complex operator+(const Complex &b) const{return Complex(x + b.x, y + b.y);}
    Complex operator*(const Complex &b) const{return Complex(x * b.x - y * b.y,
x * b.y + y * b.x);}
};
/*
 * 进行 FFT 和 IFFT 前的反置变换
 * 位置 i 和 i 的二进制反转后的位置互换
 *len 必须为 2 的幂
 */
void change(Complex y[], int len)
{
    int i, j, k;
    for (int i = 1, j = len / 2; i < len - 1; i++)
    {
        if (i < j)
            swap(y[i], y[j]);
        // 交换互为小标反转的元素，i<j 保证交换一次
        // i 做正常的 + 1，j 做反转类型的 + 1，始终保持 i 和 j 是反转的
        k = len / 2;
        while (j >= k)
        {
            j = j - k;
            k = k / 2;
        }
```

```
        if (j < k)
            j += k;
    }
}

/*
 * 做 FFT
 *len 必须是 2^k 形式
 *on == 1 时是 DFT，on == -1 时是 IDFT
 */
void fft(Complex y[], int len, int on)
{
    change(y, len);
    for (int h = 2; h <= len; h <<= 1)
    {
        Complex wn(cos(2 * PI / h), sin(on * 2 * PI / h));
        for (int j = 0; j < len; j += h)
        {
            Complex w(1, 0);
            for (int k = j; k < j + h / 2; k++)
            {
                Complex u = y[k];
                Complex t = w * y[k + h / 2];
                y[k] = u + t;
                y[k + h / 2] = u - t;
                w = w * wn;
            }
        }
    }
    if (on == -1)
        for (int i = 0; i < len; i++)
            y[i].x /= len;
}
const int N = 1e7+5;
Complex a[N],b[N];
int res[N];
int main()
{
    ios::sync_with_stdio(false);cin.tie(0);
    int n,m;
    cin>>n>>m;
    int len=1;
    while(len<(n+1)*2||len<(m+1)*2)
        len<<=1;
    for(int i=0;i<=n;i++)
        cin>>a[i].x;
    for(int i=0;i<=m;i++)
        cin>>b[i].x;
    fft(a,len,1);
    fft(b,len,1);
    for(int i=0;i<len;i++)
        a[i]=a[i]*b[i];
    fft(a,len,-1);
    for(int i=0;i<len;i++)
        res[i]=(int)(a[i].x+0.5);
    for(int i=0;i<=n+m;i++)
        cout<<res[i]<<' ';
    cout<<'\n';
```

```
    return 0;
}
```

高精度

```
#ifndef __x86_64__
#error Only x86-64 targets are supported
#endif
#include<cstdint>
#include<vector>
#include<string>
#include<iosfwd>
#define __builtin_ia32_adc(x,y,flag) __asm__("addb   %3, %0\n\t" "adcq   %2,
%1\n\t" "setc   %0":"+r"(flag),"+r"(x):"r"(y),"i"(-1):"cc")

struct bigint{// made by dengyaotriangle!
    typedef unsigned long long u64;
    typedef unsigned __int128 u128;
    typedef std::size_t st;
    std::vector<u64> data;
    bigint(){}
    bigint(u64 x):data(x?std::vector<u64>{x}:std::vector<u64>{}){}
    bigint(const std::string &s){
        st pos=s.length();
        int cnt=0;
        u64 val=0;
        while(pos){
            pos--;
            if(cnt==64){
                data.push_back(val);
                val=0;cnt=0;
            }
            val|=(u64)(s[pos]=='1')<<cnt;
            ++cnt;
        }
        if(cnt&&val)data.push_back(val);
    }
    explicit operator std::string()const{
        if(data.empty())return "0";
        bool t=0;
        std::string ret;
        for(int i=63;i>=0;i--){
            t|=(data.back()>>i)&1;
            if(t)ret+='0'|((data.back()>>i)&1);
        }
        st i=data.size()-1;
        while(i){
            i--;
            for(int j=63;j>=0;j--)ret+='0'|((data[i]>>j)&1);
        }
        return ret;
    }
    explicit operator bool()const{return !data.empty();}
    explicit operator u64()const{return data.empty()?0:data[0];}
    st digit()const{
```

```
        if(data.empty())return 0;
        return (data.size()<<6)-__builtin_clzll(data.back());
    }
    bool operator==(const bigint &a)const{return a.data==data;}
    bool operator!=(const bigint &a)const{return a.data!=data;}
    bool operator<(const bigint &a)const{
        if(data.size()!=a.data.size())return data.size()<a.data.size();
        for(st i=data.size();i;){
            i--;
            if(data[i]!=a.data[i])return data[i]<a.data[i];
        }
        return 0;
    }
    bool operator>(const bigint &a)const{return a<(*this);}
    bool operator<=(const bigint &a)const{return !(*this>a);}
    bool operator>=(const bigint &a)const{return !(*this<a);}
    bigint &operator<<=(st n){
        if(data.empty())return *this;
        int w=n&63;st z=n>>6;
        st i=data.size();
        bool flg=0;
        if(w&&(data.back()>>(64-w)))data.push_back(0),flg=1;
        data.resize(data.size()+z);
        while(i){
            i--;
            if(flg)data[i+z+1]|=data[i]>>(64-w);
            data[i+z]=data[i]<<w;
            flg|=bool(w);
        }
        for(st i=0;i<z;i++)data[i]=0;
        return *this;
    }
    bigint &operator>>=(st n){
        int w=n&63;st z=n>>6,i=0;
        for(;i+z<data.size();i++){
            if(w&&i)data[i-1]|=data[i+z]<<(64-w);
            data[i]=data[i+z]>>w;
        }
        while(data.size()>i)data.pop_back();
        while(!data.empty()&&data.back()==0)data.pop_back();
        return *this;
    }
    bigint operator<<(st n)const{return bigint(*this)<<=n;}
    bigint operator>>(st n)const{return bigint(*this)>>=n;}
    bigint &operator+=(const bigint &a){
        data.resize(std::max(data.size(),a.data.size()));
        bool carry=0;
        for(st i=0;i<data.size();i++){
            u64 rg=0;
            if(i<a.data.size())rg=a.data[i];
            __builtin_ia32_adc(data[i],rg,carry);
        }
        if(carry)data.push_back(1);
        return *this;
    }
    bigint &operator-=(const bigint &a){
        bool carry=1;
        for(st i=0;i<data.size();i++){
```

```
            u64 rg=-1;
            if(i<a.data.size())rg=~a.data[i];
            __builtin_ia32_adc(data[i],rg,carry);
        }
        while(!data.empty()&&data.back()==0)data.pop_back();
        return *this;
    }
    bigint &operator++(){return *this+=bigint(1);}
    bigint &operator--(){return *this-=bigint(1);}
    bigint operator++(int){bigint tmp=*this;++*this;return tmp;}
    bigint operator--(int){bigint tmp=*this;--*this;return tmp;}
    bigint &operator*=(const bigint &a){
        std::vector<u64> ret(data.size()+a.data.size());
        for(st i=0;i<data.size();i++){
            u64 carry=0;bool wcarry=0;
            st k=i;
            for(st j=0;j<a.data.size();j++,k++){
                u128 r=data[i]*(u128)a.data[j]+carry;
                u64 cur=r;
                carry=r>>64;
                __builtin_ia32_adc(ret[k],cur,wcarry);
            }
            while(carry||wcarry){
                __builtin_ia32_adc(ret[k],carry,wcarry);
                carry=0;k++;
            }
        }
        while(!ret.empty()&&ret.back()==0)ret.pop_back();
        data=ret;
        return *this;
    }
    bigint &operator/=(const bigint &a){
        if(a.digit()>digit()){
            data.clear();
            return *this;
        }
        st z=digit()-a.digit();
        std::vector<u64> ret;
        while(1){
            bigint tmp=a<<z;
            if(tmp<=*this){
                *this-=tmp;
                st v1=z>>6;
                if(ret.size()<=v1)ret.resize(v1+1);
                ret[v1]|=(u64)(1)<<(z&63);
            }
            if(!z)break;
            z--;
        }
        data=ret;
        return *this;
    }
    bigint &operator%=(const bigint &a){
        if(a.digit()>digit())return *this;
        st z=digit()-a.digit();
        while(1){
            bigint tmp=a<<z;
            if(tmp<=*this)*this-=tmp;
```

```
            if(!z)break;
            z--;
        }
        return *this;
    }
    bigint operator+(const bigint &a)const{return bigint(*this)+=a;}
    bigint operator-(const bigint &a)const{return bigint(*this)-=a;}
    bigint operator*(const bigint &a)const{return bigint(*this)*=a;}
    bigint operator/(const bigint &a)const{return bigint(*this)/=a;}
    bigint operator%(const bigint &a)const{return bigint(*this)%=a;}
};
std::istream &operator>>(std::istream &st,bigint &a){
    std::string s;st>>s;a=bigint(s);return st;
}
std::ostream &operator<<(std::ostream &st,const bigint &a){
    return st<<(std::string)(a);
}
```