

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

[a, b]的随机数

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

xor shift

映射到 2^{64}

```

const ull mask = std::chrono::steady_clock::now().time_since_epoch().count();
ull shift(ull x){
    x^=x<<13;
    x^=x>>7;
    x^=x<<17;
    x^=mask;
    return x;
}

```

memset

int / long long

“较”的原则：加法不爆

- 极大值: 0x7f
- 较大值: 0x3f
- 较小值: 0xc0
- 极小值: 0x80

float

“较”的原则：保证一定精度

7f以上一直到be都是-0（很小的>-1.0的负数）

- 极大值: 0x7f
- 较大值: 0x4f
- 较小值: 0xce
- 极小值: 0xfe

double

“较”的原则：保证一定精度

- 极大值: 0x7f
- 较大值: 0x43
- 较小值: 0xc2
- 极小值: 0xfe

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;
}

```

pair哈希

```
struct pair_hash
{
    template <class T1, class T2>
    size_t operator () (pair<T1, T2> const &pair) const
    {
        size_t h1 = hash<T1>()(pair.first);
        size_t h2 = hash<T2>()(pair.second);
        return h1 ^ h2;
    }
};
unordered_set<pair<int, int>, pair_hash> st;
```

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<1||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

任意一点出发找到最远点 A ， A 一定在直径上，再从 A 出发找到最远点 B ， B 即为直径

```

vector<pair<int,int>>arc[N];
void dfs(int x,int f,int d)
{
    dist[x]=d;
    fa[x]=f;
    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

树上倍增

```
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

离线之后并查集找LCA

```
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);
}
}
}

```

最短路

Floyd

时间复杂度 $O(n^3)$, 空间复杂度 $O(n^2)$

```

for (k = 1; k <= n; k++)
    for (x = 1; x <= n; x++)
        for (y = 1; y <= n; y++)
            f[x][y] = min(f[x][y], f[x][k] + f[k][y]);

```

Bellman-ford

对于边 (u, v) , 松弛操作对应 $dis(v) = \min(dis(v), dis(u) + w(u, v))$

最短路存在的情况下, 最多经过 $n - 1$ 次松弛操作, 时间复杂度为 $O(nm)$

可以用于判图中是否有负环, 如果从 s 点没跑出负环, 只能说明从 s 点出发不能抵达负环, 并不能说明图中没有负环

可以建立一个超级源点, 向图上每一个节点连一个权值为0的边, 对超级源点执行 $Bellman - ford$

```

int dis[N];
bool bellmanford(int n, int s){//图的点数为n, 出发点为s
    memset(dis, 63, sizeof(dis));
    dis[s]=0;
    bool flag=false;
    for(int i=1; i<=n; i++){
        flag=false;
        for(int j=1; j<=n; j++){
            if(dis[j]==inf)
                continue;
            for(auto [it,w]:G[j]){
                if(dis[it]>dis[j]+w){
                    dis[it]=dis[j]+w;
                    flag=true;
                }
            }
        }
        if(!flag)
            break;
    }
    return flag;
}

```

SPFA

```
int dis[N],cnt[N];
bool vis[N];
queue<int>q;
bool spfa(int n,int s){
    memset(dis,63,sizeof(dis));
    dis[s]=0;vis[s]=true;
    q.push(s);
    while(q.size()){
        int tmp=q.front();
        q.pop();
        vis[tmp]=false;
        for(auto [it,w]:G[tmp]){
            if(dis[it]>dis[tmp]+w){
                dis[it]=dis[tmp]+w;
                cnt[it]=cnt[tmp]+1;
                if(cnt[it]>=n)
                    return false;
                if(!vis[it]){
                    q.push(it);
                    vis[it]=true;
                }
            }
        }
    }
    return true;
}
```

Dijkstra

优先队列实现

复杂度 $O(m \log m)$

```
int dis[N];
bool vis[N];
priority_queue<pii,vector<pii>,greater<pii>>pq; //{距离, 点}
void Dijkstra(int n,int s){
    memset(dis,63,sizeof(dis));
    dis[s]=0;
    pq.push({0,s});
    while(pq.size()){
        pii tmp=pq.top();
        pq.pop();
        if(vis[tmp.second])
            continue;
        vis[tmp.second]=true;
        for(auto [it,w]:G[tmp.second]){
            if(dis[it]>dis[tmp.second]+w){
                dis[it]=dis[tmp.second]+w;
                pq.push({dis[it],it});
            }
        }
    }
}
```

暴力实现

复杂度 $O(n^2)$

```
int dis[N];
bool vis[N];
void Dijkstra(int n,int s){
    memset(dis,0x3f,sizeof(dis));
    dis[s]=0;
    for(int i=1;i<=n;i++){
        int k=0,m=1e15;
        for(int j=1;j<=n;j++){
            if(!vis[j]&&dis[j]<m){
                k=j;m=dis[j];
            }
        }
        vis[k]=1;
        for(auto [it,w]:G[k])
            dis[it]=min(dis[it],dis[k]+w);
    }
}
```

差分约束

n 个变量 x_1, x_2, \dots, x_n 以及 m 个约束条件 $x_i - x_j \leq c_k$

约束是否有解，如果有解，给出一组解

$x_i - x_j \leq c_k \iff x_i \leq x_j + c_k$ ，类比单源最短路中的三角形不等式 $dist[y] \leq dist[x] + z$

将 j 向 i 连长度为 c_k 的有向边，设超级源点0，向每个点连一条权为0的有向边，跑spfa，若图中有负环，则无解；否则 $x_i = dist[i]$ 就是一组解

缩点

将强连通分量缩为一个点，原图变为DAG

Tarjan缩点

$num[N], low[N]$

- num 值：dfs时这个点的时间戳
- low 值：能返回的最远祖先的时间戳

相同 low 值的属于一个 SCC ，在dfs的同时把点按 SCC 分开

复杂度 $O(n + m)$

```
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);
}

```

Kosaraju缩点

1. 原图的反图（边的方向取反）的连通性不变
2. 按原图的dfs的逆序开始dfs反图，可以将强连通分量挖出来

复杂度 $O(n + m)$

```

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);
            }
        }
    }
}

```

2-SAT

n 个集合，每个集合两个元素，已知若干个 $\langle a, b \rangle$ ，表示 a 与 b 矛盾（ a, b 属于不同集合），从每个集合选一个元素，判断能否选 n 个两两不矛盾的元素

可以变为布尔方程，选 a 则必选 b ，则连 $a \rightarrow b$ 的有向边，在图上缩点之后判断是否有一个集合中的两个数在一个SCC里

树链剖分

重链剖分

- $id[x]$: x 点的dfs序
- $rk[x]$: dfs序为 x 的节点
- $top[x]$: x 所在重链的顶部节点

```

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());
}

```


树哈希

$$f(S) = (c + \sum_{x \in S} g(x)) \mod m$$

一般取 $c = 1$, g 为整数到整数的映射

```
const ull mask = std::chrono::steady_clock::now().time_since_epoch().count();
ull shift(int x){
    x^=x<<13;
    x^=x>>7;
    x^=x<<17;
    x^=mask;
    return x;
}
ull Hash[N];
set<ull>st;
void dfs(int x,int f){
    Hash[x]=1;
    for(auto it:G[x]){
        if(it==f)
            continue;
        dfs(it,x);
        Hash[x]+=shift(Hash[it]);
    }
    st.insert(Hash[x]);
}
```

数学

常见数论函数

欧拉函数

$$\varphi(x) = x \cdot \prod (1 - \frac{1}{p_i})$$

性质

$$\phi(x) = \sum_{d|x} \frac{\mu(d)}{d}$$

费马小定理

$$p \in Prim \Rightarrow a^{p-1} \equiv 1 \mod p$$

欧拉定理

$$(a, m) = 1 \Rightarrow a^{\phi(m)} \equiv 1 \mod m$$

扩展欧拉定理

$$a^b \equiv a^{b \bmod \phi(m) + \phi(m)} \bmod m \quad (b \geq \phi(m))$$

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_0, y_0

通解:

$$x = \frac{c}{d}x_0 + \frac{b}{d}t, y = \frac{c}{d}y_0 - \frac{a}{d}t$$

```
11 extend_gcd(11 a,11 b,11 &x,11 &y)
{
    if(b==0){x=1;y=0;return a;}
    11 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

$$a^{-1} \equiv a^{p-2} \pmod{p}$$

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

递推

求 $1 \sim n$ 的所有逆元

```
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

复杂度 $O(k \log n)$

```

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

找出一个约数的时间复杂度 $O(n^{\frac{1}{4}})$

```

mt19937_64 rnd(time(0));
namespace Pollard_Rho
{
    #define ldb long double

```

```

long long mul(long long x, long long y, long long mod)
{
    return ((x * y - (long long)((long double)x / mod * y) * mod) + mod) % mod;
}

long long gcd(long long a, long long b)
{
    return (b == 0 ? a : gcd(b, a % b));
}

long long ksm(long long a, long long b, long long mod)
{
    long long ans = 1; a %= mod;
    while (b) {if (b & 1)ans = mul(ans, a, mod); b >>= 1; a = mul(a, a,
mod);}
    return ans;
}

int pr[15] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
bool Miller_Rabin(long long n)
{
    if (n == 2 || n == 3)return 1;
    if (n % 2 == 0 || n == 1)return 0;
    long long d = n - 1;
    int s = 0;
    while (d % 2 == 0)s ++, d >>= 1;
    for (int i = 0; i <= 11; i ++)
    {
        if (pr[i] >= n)break;
        long long a = pr[i];
        long long x = ksm(a, d, n);
        long long y = 0;
        for (int j = 0; j <= s - 1; j ++)
        {
            y = mul(x, x, n);
            if (y == 1 && x != 1 && x != (n - 1))return 0;
            x = y;
        }
        if (y != 1)return 0;
    }
    return 1;
}

long long Pollard_Rho(long long n)
{
    long long now, pre, g;
    while (true)
    {
        now = pre = rnd() % (n - 1) + 1;
        g = 1;
        long long c = rnd() % (n - 1) + 1;
        for (int i = 1, fst = 1 ;; i ++)
        {
            now = (mul(now, now, n) + c) % n;
            g = mul(g, abs(now - pre), n);
            if (now == pre || !g)break;
            if (!(i & 127) || i == fst)
            {
                g = gcd(g, n);
                if (g > 1)return g;
                if (i == fst)pre = now, fst <<= 1;
            }
        }
    }
}

```

```

    }
}
}
void Find(long long n, map<long long, long long>& _P, int c = 1)
{
    if (n == 1) return ;
    if (Miller_Rabin(n))
    {
        _P[n] += c;
        return;
    }
    long long p = Pollard_Rho(n);
    int cnt = 0;
    while (!(n % p))
    {
        n /= p, cnt ++;
    }
    Find(p, _P, cnt * c);
    Find(n, _P, c);
}
}

```

离散对数

bsgs

对 $a, b \in \mathbb{Z}^+$, 可以以 $O(\sqrt{m})$ 的复杂度内求解

$$a^x \equiv b \pmod{m}$$

其中 $(a, m) = 1$, 解 $0 \leq x < m$, m 不一定是素数

取 $x = A\lceil\sqrt{m}\rceil - B$, 其中 $0 \leq A, B \leq \lceil\sqrt{m}\rceil$, 有 $a^{A\lceil\sqrt{m}\rceil - B} \equiv b \pmod{m}$

$$\iff a^{A\lceil\sqrt{m}\rceil} \equiv ba^B \pmod{m}$$

同时枚举左右两边, 用hashmap存, 可以 $O(\sqrt{m})$ 的复杂度内解决

```

11 BSGS(11 a, 11 b, 11 m)
{
    static unordered_map<11, 11> hs;
    hs.clear();
    11 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
    }
    11 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

a, m 不一定互质

```
// 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

$$\binom{n}{m} = C_n^m = \frac{P_n^m}{P_m} = \frac{n!}{m!(n-m)!}$$

组合恒等式

$$C_n^k = C_n^{m-k}$$

$$C_{n+1}^k = C_n^k + C_n^{k-1}$$

$$(C_n^0)^2 + (C_n^1)^2 + (C_n^2)^2 + \dots + (C_n^n)^2 = C_{2n}^n = \frac{(2n)!}{(n!)^2}$$

$$C_{-n}^k = \frac{(-n)(-n-1)(-n-2)\dots(-n-k+1)}{k!} = (-1)^k C_{n+k-1}^k$$

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

$$p \in Prim, C_n^m \equiv C_{n/p}^{m/p} \cdot C_{n \% p}^{m \% p} \pmod p$$

模数较小，但组合数很大

推论：

$$m, n \in \mathbb{Z}^+, p \in Prim, C_n^m \equiv \prod_{i=0}^k C_{n_i}^{m_i}$$

$$m = m_k p^k + \dots + m_1 p + m_0, n = n_k p^k + \dots + n_1 p + n_0$$

```
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;
}
```

Wilson定理

$$p \in Prim, (p-1)! \equiv -1 \pmod p$$

中国剩余定理

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_k \pmod{m_k} \end{aligned}$$

其中 m_i, m_j 两两互质

设：

$$M = \prod_{i=1}^k m_i, M_i = \frac{M}{m_i}, M_i^{-1} \cdot M_i \equiv 1 \pmod{m_i}$$

方程组在模 M 意义下有唯一解

$$x \equiv \sum_{i=1}^k a_i M_i M_i^{-1} \pmod{M}$$

升幂引理(LTE)

$v_p(n)$ 为 n 的标准分解中质因数 p 的幂次, 即 $v_p(n)$ 满足 $p^{v_p(n)} \mid n$ 且 $p^{v_p(n)+1} \nmid n$

以下设 $p \in Prim, x, y \in Z, p \nmid x, p \nmid y, n \in Z^+$

- 第一部分: $p \in Prim, (n, p) = 1$

1. 若 $p \mid (x - y)$, 则

$$v_p(x^n - y^n) = v_p(x - y)$$

- 2.

3. 若 $p \mid (x + y)$, n 奇, 则

$$v_p(x^n + y^n) = v_p(x + y)$$

- 第二部分: p 奇素数

1. 若 $p \mid (x - y)$, 则

$$v_p(x^n - y^n) = v_p(x - y) + v_p(n)$$

2. 若 $p \mid (x + y)$, 则对奇数 n 有

$$v_p(x^n + y^n) = v_p(x + y) + v_p(n)$$

- 第三部分: $p = 2$ 且 $p \mid (x - y)$

1. 对奇数 n 有

$$v_p(x^n - y^n) = v_p(x - y)$$

2. 对偶数 n 有

$$v_p(x^n - y^n) = v_p(x - y) + v_p(x + y) + v_p(n) - 1$$

类欧几里得

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

$$f(a, b, c, n) = \sum_{i=0}^n \left\lfloor \frac{ai + b}{c} \right\rfloor$$

时间复杂度 $O(\log n)$

```

11 f(11 a, 11 b, 11 c, 11 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);
    11 m = (a * n + b) / c;
    return n * m - f(c, c - b - 1, a, m - 1);
}

```

阶

定义

若满足 $a^n \equiv 1 \pmod m$ 的最小正整数 n 存在, 这个 n 称为 a 模 m 的阶, 记作 $n = \delta_m(a)$ 或 $\text{ord}_m(a)$

性质

- $a, a^2, \dots, a^{\delta_m(a)}$ 模 m 两两不同余
- 若 $a^n \equiv 1 \pmod m$, 则 $\delta_m(a) \mid n$
- $a^p \equiv a^q \Rightarrow p \equiv q \pmod{\delta_m(a)}$
- $m \in N^*, a, b \in Z, (a, m) = (b, m) = 1$, 则

$$\delta_m(ab) = \delta_m(a)\delta_m(b) \iff (\delta_m(a), \delta_m(b)) = 1$$

- $k \in N, m \in N^*, a \in Z, (a, m) = 1$, 则

$$\delta_m(a^k) = \frac{\delta_m(a)}{(\delta_m(a), k)}$$

原根

定义

若 $(g, m) = 1$ 且 $\delta_m(g) = \phi(m)$, 则称 g 为模 m 的原根

性质

若一个数 m 有原根, 则它原根的个数为 $\phi(\phi(m))$

原根存在定理

一个数 m 存在原根当且仅当 $m = 2, 4, p^\alpha, 2p^\alpha$, 其中 p 为奇素数, $\alpha \in N^*$

莫比乌斯反演

莫比乌斯函数

$$\mu(n) = \begin{cases} 1 & n = 1 \\ 0 & n \text{ 含有平方因子} \\ (-1)^k & k \text{ 为 } n \text{ 的本质不同质因子个数} \end{cases}$$

性质

- 积性函数

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & n = 1 \\ 0 & n \neq 1 \end{cases} \iff \sum_{d|n} \mu(d) = \varepsilon(n) = [n == 1], \mu * 1 = \varepsilon$$

$$[gcd(i, j) == 1] = \sum_{d|gcd(i, j)} \mu(d)$$

莫比乌斯变换

设 $f(n), g(n)$ 为数论函数

$$f(n) = \sum_{d|n} g(d) \Rightarrow g(n) = \sum_{d|n} \mu(d) f\left(\frac{n}{d}\right)$$

$f(n)$ 称为 $g(n)$ 的莫比乌斯变换, $g(n)$ 称为 $f(n)$ 的莫比乌斯逆变换 (反演)

$$f(n) = \sum_{n|d} g(d) \Rightarrow g(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right) f(d)$$

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;
}

```

斐波那契数

$$F_n = F_{n-1} + F_{n-2}, F_0 = 0, F_1 = 1$$

性质

- $F_{n-1}F_{n+1} - F_n^2 = (-1)^n$
- $F_{n+k} = F_kF_{n+1} + F_{k-1}F_n$
- $F_{2n} = F_n(F_{n+1} + F_{n-1})$
- $a \mid b \iff F_a \mid F_b$
- $\gcd(F_m, F_n) = F_{\gcd(m,n)}$

卡特兰(Catalan)数

$$H_n = \frac{C_{2n}^n}{n+1}, n \geq 2, n \in N^+, H_0 = H_1 = 1$$

$$H_n = \begin{cases} \sum_{i=1}^n H_{i-1}H_{n-i} & n \geq 2, n \in N^+ \\ 1 & n = 0, 1 \end{cases}$$

$$H_n = \frac{H_{n-1}(4n-2)}{n+1}$$

$$H_n = C_{2n}^n - C_{2n}^{n-1}$$

封闭形式

$$H(x) = \frac{1 - \sqrt{1-4x}}{2x} = \sum_{n \geq 0} C_{2n}^n \frac{1}{n+1} x^n$$

典型问题

- 有 $2n$ 个人排成一行进入剧场。入场费 5 元。其中只有 n 个人有一张 5 元钞票，另外 n 人只有 10 元钞票，剧院无其它钞票，问有多少种方法使得只要有 10 元的人买票，售票处就有 5 元的钞票找零？
- 一位大城市的律师在她住所以北 n 个街区和以东 n 个街区处工作。每天她走 $2n$ 个街区去上班。如果他从不穿越（但可以碰到）从家到办公室的对角线，那么有多少条可能的道路？
- 在圆上选择 $2n$ 个点，将这些点成对连接起来使得所得到的 n 条线段不相交的方法数？
- 对角线不相交的情况下，将一个凸多边形区域分成三角形区域的方法数？
- 一个栈（无穷大）的进栈序列为 $1, 2, \dots, n$ ，有多少个不同的出栈序列？
- n 个结点可构造多少个不同的二叉树？
- n 个 $+1$ 和 n 个 -1 构成 $2n$ 项 a_1, a_2, \dots, a_{2n} ，其部分和满足 $a_1 + a_2 + \dots + a_k \geq 0 (k = 1, 2, 3, \dots, 2n)$ ，序列个数为？

斯特林数

第二类斯特林数(斯特林子集数)

$S(n, k)$ 表示将 n 个两两不同的元素，划分为 k 个互不区分的非空子集的方案数

$$S(n, k) = S(n-1, k-1) + k \cdot S(n-1, k), S(n, 0) = [n == 0]$$

$$S(n, m) = \sum_{i=0}^m \frac{(-1)^{m-i} i^n}{i!(m-i)!}$$

第一类斯特林数(斯特林轮换数)

$s(n, k)$ 表示将 n 个两两不同的元素，划分为 k 个互不区分的非空轮换的方案数

$$s(n, k) = s(n-1, k-1) + (n-1) \cdot s(n-1, k), s(n, 0) = [n == 0]$$

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{//maxL是字符串总长
    int cnt=0,ch[maxL][26],sz[maxL],Cnt[maxL];//sz[maxL]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;
            Cnt[now]++;
        }
        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);
}

```

博弈论

Nim游戏

简介

n 堆物品，每堆有 a_i 个，两个玩家轮流取走任意一堆的任意个物品，但不能不取，取走最后一个物品的获胜

Nim和

定义Nim和= $a_1 \oplus a_2 \oplus \dots \oplus a_n$

当且仅当Nim和为0时，状态为必败状态，否则为必胜状态

SG函数

mex函数

值为不属于集合 S 中的最小非负整数

$$mex(S) = \min\{x\} (x \notin S, x \in N)$$

SG函数

设状态 x 的后继为 y_1, y_2, \dots, y_k ,

$$SG(x) = mex\{SG(y_1), SG(y_2), \dots, SG(y_k)\}$$

对于由 n 个有向图游戏组成的组合游戏，设起点分别为 s_1, s_2, \dots, s_n ，当且仅当 $SG(s_1) \oplus SG(s_2) \oplus \dots \oplus SG(s_n) \neq 0$ 时，这个游戏是先手必胜的，同时，这是一个组合游戏的游戏状态 x 的SG的

打表SG函数

记忆化搜索或者dp

多项式

常见的幂级数展开

$$e^x = 1 + x + \frac{1}{2!}x^2 + \cdots + \frac{1}{n!}x^n + \cdots$$

$$(1+x)^a = 1 + ax + \frac{a(a-1)}{2!}x^2 + \cdots + \frac{a(a-1)\cdots(a-n+1)}{n!}x^n + \cdots$$

$$\cos x = 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 + \cdots + \frac{(-1)^n}{(2n)!}x^{2n} + \cdots$$

$$\sin x = x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 + \cdots + \frac{(-1)^n}{(2n+1)!}x^{2n+1} + \cdots$$

$$\frac{1}{1+x} = 1 - x + x^2 + \cdots + (-1)^n x^n + \cdots$$

$$\ln(1+x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 + \cdots + \frac{(-1)^{n-1}}{n}x^n + \cdots$$

$$\frac{1}{1+x^2} = 1 - x^2 + x^4 + \cdots + (-1)^n x^{2n} + \cdots$$

生成函数

$$F(x) = \sum_n a_n k_n(x)$$

$k_n(x)$ 为核函数

- 普通生成函数: $k_n(x) = x^n$
- 指数生成函数: $k_n(x) = \frac{x^n}{n!}$
- 狄利克雷生成函数: $k_n(x) = \frac{1}{n^x}$

计算方式

通常在封闭形式和展开形式间转换进行计算

1. 对于任意多项式 $P(x), Q(x)$, 生成函数 $\frac{P(x)}{Q(x)}$ 的展开式都可以用待定系数法求出

当对分母进行因式分解但有重根时, 每有一个重根就要多一个分式

$$\text{如 } G(x) = \frac{1}{(1-x)(1-2x)^2} \Rightarrow G(x) = \frac{c_0}{1-x} + \frac{c_1}{1-2x} + \frac{c_2}{(1-2x)^2}$$

2. 牛顿二项式定理

$$C_r^k = \frac{r(r-1)(r-2)\cdots(r-k+1)}{k}, \quad k \in N, r \in R$$

$$(1+x)^\alpha = \sum_{n \geq 0} C_n^\alpha x^n$$

快速傅里叶变换(FFT)

以 $O(n \log n)$ 的速度计算两个 n 度多项式乘法

```
#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;
}

```

快速数论变换(NTT)

计算几何

```

typedef pair<double, double> p11;
p11 operator+(p11 x, p11 y){
    return {x.first+y.first, x.second+y.second};
}
p11 operator-(p11 x, p11 y){
    return {x.first-y.first, x.second-y.second};
}
p11 operator*(p11 x, double k){
    return {x.first*k, x.second*k};
}
p11 operator/(p11 x, double k){
    return {x.first/k, x.second/k};
}
double len(p11 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);
                    }
            }
    }
}
}

```

高精度

```

#ifndef __x86_64__
#error Only x86-64 targets are supported
#endif
#include<stdint>
#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);
}

```