ACM模板

1. 常用杂项

1.1. 读入输出优化

1.1.1. 解除流同步

```
ios::sync_with_stdio(false);
cin.tie(0);
cout.tie(0);
```

1.1.2. 快读

```
int read(){
    int x=0,f=1;
    char ch=getchar();
    while(!isdigit(ch)){
        if(ch=='-')
            f=-1;
        ch=getchar();
    }
    while(isdigit(ch)){
        x=x*10+ch-'0';
        ch=getchar();
    }
    return x*f;
}
```

1.1.3. 快写

```
void write(int x){
    static int sta[35];
    int top = 0;
    if(x<0){
```

```
        x=-x;
        putchar('-');
    }

    do{
        sta[top++]=x%10;
        x/=10;
    }while(x);
    while(top)
        putchar(sta[--top]+'0');
}
```

1.2. 文件操作

```
//重定向后printf/scanf/cin/cout等函数正常使用
freopen("data.in","r",stdin);
freopen("ans.out","w",stdout);

//关闭标准输入/输出流
fclose(stdin);
fclose(stdout);
```

1.3. 去重

```
vector<T>a;
sort(a.begin(),a.end());
a.erase(unique(a.begin(),a.end()),a.end());
```

1.4. 常用函数

1.4.1. 求第k小的数

```
//无返回值，只是将第k个放回正确位置
nth_element();
//vector使用参数（begin(),begin()+k,end())
//普通数组使用参数（数组名，数组名+k，数组名+n)
```

1.4.2. 全排列

```
//需提前排序
next_permutation();
//vector使用参数（begin(),end())
//普通数组使用参数（数组名,数组名+n)
```

1.5. 前缀和

1.5.1. 一维前缀和

```
const int MAX=100001;
int a[MAX];//存前缀和
//预处理前缀和
void init(int n){
    for(int i=1;i<=n;i++)
        a[i]+=a[i-1];
}
//查询区间和
int find(int l,int r){
    return a[r]-a[l-1];
}
```

1.5.2. 二位前缀和

```
const int MAX=1001;
int a[MAX][MAX];//存前缀和
//预处理前缀和，n行m列
void init(int n,int m){
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            a[i][j]+=a[i-1][j]+a[i][j-1]-a[i-1][j-1];
}
//查询区间和(x1,y1)->(x2,y2)子矩阵和
int find(int x1,int y1,int x2,int y2){
    return a[x2][y2]-a[x2][y1-1]-a[x1-1][y2]+a[x1-1][y1-1];
}
```

1.6. 差分

1.6.1. 一维差分

```
const int MAX=1000002;
int a[MAX];//存差分
//预处理差分
void init(int n){
    for(int i=n;i>=1;i--)
        a[i]-=a[i-1];
}
//区间修改
void modify(int l,int r,int k){
    a[l]+=k;
    a[r+1]-=k;
```

```
}
//单点查询
int find(int p){
    int ans=0;
    for(int i=1;i<=p;i++)
        ans+=a[i];
    return ans;
}
```

1.6.2. 二维差分

```
const int MAX=1002;
int a[MAX][MAX];//存差分
//预处理差分,n行m列
void init(int n,int m){
    for(int i=n;i>=1;i--)
        for(int j=m;j>=1;j--)
            a[i][j]=a[i][j]-a[i][j-1]-a[i-1][j]+a[i-1][j-1];
}
//区间修改(x1,y1)->(x2,y2)子矩阵+k
void modify(int x1,int y1,int x2,int y2,int k){
    a[x1][y1]+=k;
    a[x1][y2+1]-=k,a[x2+1][y1]-=k;
    a[x2+1][y2+1]+=k;
}
//单点查询
int find(int x,int y){
    int ans=0;
    for(int i=1;i<=x;i++)
        for(int j=1;j<=y;j++)
            ans+=a[i][j];
    return ans;
}
```

2. 排序

2.1. 快速排序（不稳定）

```
void qsort(int l,int r){
    if(r<=l)
        return;
    int flag=l;//哨兵节点
    int ll=l,rr=r;
    while(ll<rr){
        //两个循环顺序不能变，确保rr==ll时a[ll]<=a[flag]
        for(;rr>ll;rr--)
            if(a[rr]<a[flag]) break;
        for(;ll<rr;ll++)
```

```
            if(a[ll]>a[flag]) break;
        if(a[ll]>a[flag]&&a[rr]<a[flag])
            swap(a[ll],a[rr]);
    }
    swap(a[flag],a[ll]);
    qsort(l,ll-1);
    qsort(ll+1,r);
}
```

2.2. 归并排序（可求逆序对）

```
vector<int>a;
long long ans=0;//逆序对数
void merge_sort(int l,int r){
    if(r<=l)
        return;
    int mid=(l+r)>>1;
    merge_sort(l,mid);
    merge_sort(mid+1,r);
    vector<int>temp;
    int ll=l,rr=mid+1;
    while(!(ll==mid+1&&rr==r+1)){
        if(ll==mid+1||(rr<r+1&&a[rr]<a[ll])){
            temp.push_back(a[rr]);
            ans+=mid+1-ll;//求逆序对
            rr++;
        }else{
            temp.push_back(a[ll]);
            ll++;
        }
    }
    for(int i=0;i<temp.size();i++){
        a[l]=temp[i];
        l++;
    }
}
```

3. 数据结构

3.1. 向量

```
vector<T>a;
a.assign();//对向量中的元素赋值
a.front();//返回第一个元素
a.back();//返回最末一个元素
a.begin();//返回第一个元素的迭代器
a.end();//返回最末元素下一个位置的迭代器
a.clear();//清空所有元素
a.empty();//为空时返回真
```

```
a.erase();//删除指定元素
a.insert();//插入元素
a.pop_back();//移除最后一个元素
a.push_back();//在最后添加一个元素
a.rbegin();//返回尾部的逆迭代器
a.rend();//返回起始的逆迭代器
a.resize();//改变元素数量
a.size();//返回元素数量
a.swap();//交换两个向量
```

3.2. 栈

```
stack<T>a;
a.push();//入栈
a.pop();//出栈
a.top();//返回栈顶元素
a.empty();//栈为空返回真
a.size();//返回栈内元素数量
```

3.3. 队列

```
queue<T>q;
q.push();//入队
q.pop();//出队
q.front();//返回队首元素
q.back();//返回队尾元素
q.empty();//队为空返回真
q.size();//返回队内元素数量
```

3.4. 双端队列

```
deque<T>a;
a.assign();//赋值
a.at();//返回指定的元素
a.front();//返回首元素
a.back();//返回尾元素
a.begin();//返回指向第一个元素的迭代器
a.end();//返回指向尾部的迭代器
a.clear();//删除所有元素
a.empty();//为空返回真
a.erase();//删除一个元素
a.insert();//插入一个元素
a.pop_front();//删除头部的元素
a.pop_back();//删除尾部的元素
a.push_front();//在头部加入一个元素
a.push_back();//在尾部加入一个元素
```

```
a.rend();//返回指向头部的逆向迭代器
a.rbegin();//返回指向尾部的逆向迭代器
a.resize();//改变大小
a.size();//返回元素个数
a.swap();//和另一个双端队列交换元素
```

3.5. 优先队列

```
priority_queue<T>q;//降序,即top()为最大值
priority_queue<T,vector<T>,greater<T> >q;//升序,即top()为最小值
q.top();//返回优先级最高的元素
q.push();//加入一个元素
q.pop();//删除第一个元素
q.size();//返回元素个数
q.empty();//为空返回真
```

3.6. 并查集

```
const int MAX=100000;
int root[MAX];//存每个点的根节点
int tall[MAX];//存每个节点的高，优化合并
//初始化根节点为自己，高为0
void init(int n){
    for(int i=0;i<=n;i++)
        root[i]=i,tall[i]=0;
}
//递归查找根节点，并更新
int find(int n){
    if(n==root[n])
        return n;
    else
        return root[n]=find(root[n]);
}
//判断x,y是否在同一个集合
bool issame(int x,int y){
    return find(x)==find(y);
}
//合并x，y两个集合
void merge(int x,int y){
    x=find(x);
    y=find(y);
    if(x==y)
        return;
    else if(tall[x]<tall[y])
        root[y]=x;
    else{
        root[x]=y;
        if(tall[x]==tall[y])
            tall[x]++;
```

```
    }
}
```

3.7. ST表

```
const int MAX=100000;
const int p=20;//维护的最大区间，确保2*pow(2,p-1)>=数据规模
int a[MAX][p+1];//存ST表,从下标1开始存
int pre[p+1];
//预处理pre,并对读入的数据进行建表
void init(int n){
    pre[0]=1;
    for(int i=1;i<=p;i++)
        pre[i]=pre[i-1]*2;
    for(int i=1;i<=p;i++){
        for(int j=1;j+pre[i]-1<=n;j++)
            a[j][i]=max(a[j][i-1],a[j+pre[i-1]][i-1]);//维护从j开始长度为2^i区间的最
大值
    }
}
int find(int l,int r){
    int len=log2(r-l+1);
    return max(a[l][len],a[r-pre[len]+1][len]);
}
```

3.8. 线段树

```
//线段树维护区间最大最小值  first为最大值在a数组中下标  second为最小值在a数组中下标
modify直接修改a数组
const int MAX=300001;
int a[MAX];//原数组
P tree[4*MAX];//线段树
//建树
void build(int x,int l,int r){
    if(l==r){
        tree[x]={l,l};
        return;
    }
    int mid=(l+r)>>1;
    build(x*2,l,mid);
    build(x*2+1,mid+1,r);
    if(a[tree[x*2].first]<a[tree[x*2+1].first]){
        tree[x].first=tree[x*2+1].first;
    }else{
        tree[x].first=tree[x*2].first;
    }
    if(a[tree[x*2].second]<=a[tree[x*2+1].second]){
        tree[x].second=tree[x*2].second;
    }else{
```

```
            tree[x].second=tree[x*2+1].second;
        }
        return;
    }
    //单点修改，并更新区间最值
    void modify(int x,int l,int r,int p,int k){
        if(l==r){
            a[l]+=k;
            return;
        }
        int mid=(l+r)/2;
        if(p<=mid){
            modify(x*2,l,mid,p,k);
        }else{
            modify(x*2+1,mid+1,r,p,k);
        }
        if(a[tree[x*2].first]<a[tree[x*2+1].first]){
            tree[x].first=tree[x*2+1].first;
        }else{
            tree[x].first=tree[x*2].first;
        }
        if(a[tree[x*2].second]<=a[tree[x*2+1].second]){
            tree[x].second=tree[x*2].second;
        }else{
            tree[x].second=tree[x*2+1].second;
        }
        return;
    }
    //区间查询
    P get(int l,int r,int s,int t,int x){
        if(l==s&&r==t){
            return tree[x];
        }
        int mid=(s+t)/2;
        if(r<=mid){
            return get(l,r,s,mid,x*2);
        }else if(l>mid){
            return get(l,r,mid+1,t,x*2+1);
        }else{
            P temp1=get(l,mid,s,mid,x*2);
            P temp2=get(mid+1,r,mid+1,t,x*2+1);
            P ans;
            if(a[temp1.first]<a[temp2.first]){
                ans.first=temp2.first;
            }else{
                ans.first=temp1.first;
            }
            if(a[temp1.second]<=a[temp2.second]){
                ans.second=temp1.second;
            }else{
                ans.second=temp2.second;
            }
            return ans;
        }
```

```
}
```

4. 图论

4.1. 最短路

4.1.1. floyd（多源）--经典范围n=500

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
signed main(){
    int n,m;//n个点，m条边
    int inf=INT_MAX;//未开long long 使用INT_MAX 可能INT_MAX+INT_MAX溢出导致错误
    cin>>n>>m;
    int dp[n+1][n+1];
    fill(dp[0],dp[0]+(n+1)*(n+1),inf);//初始化为inf,即两个点之间不连接
    for(int i=1;i<=n;i++){
        dp[i][i]=0;//i->i距离为零
    }

    for(int i=0;i<m;i++){
        int u,v,w;
        cin>>u>>v>>w;//u->v 长为w
        dp[u][v]=min(dp[u][v],w);
        dp[v][u]=min(dp[v][u],w);//无向图
    }
    for(int k=1;k<=n;k++)
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                dp[i][j]=min(dp[i][j],dp[i][k]+dp[k][j]);//i->j,i->k->j
    return 0;
}
```

4.1.2. Dijkstra堆优化版（单源）

```
#include<bits/stdc++.h>
using namespace std;
#define P pair<int,int>
#define int long long
struct edge{
    int to,val; //to为这条边的终点，val为这条边的权
};
signed main(){
    int n,m,s;//n个点，m条边，s起始点
    int inf=INT_MAX;//未开long long 使用INT_MAX 可能INT_MAX+INT_MAX溢出导致错误
    cin>>n>>m>>s;
```

```
    vector<edge>a[n+1];//存图
    for(int i=0;i<m;i++){
        int u,v,w;//u->v 边权为w
        cin>>u>>v>>w;
        a[u].push_back({v,w});
        a[v].push_back({u,w});//无向图
    }
    int dp[n+1];//s至每个点的距离
    fill(dp,dp+n+1,inf);//初始s与所有点不连通
    dp[s]=0;//s至s距离为0
    priority_queue<P,vector<P>,greater<P> >ans;//P.first为s至这个点的距离,P.second
为从s到的这个点
    ans.push({0,s});
    while(!ans.empty()){
        P t=ans.top();//取最短距离
        ans.pop();//弹出最短距离
        if(dp[t.second]<t.first)
            continue;//如果弹出的距离不是最新的最短距离，则跳过
        else{
            //遍历这个点所连的所有边看是否可以更新到所连边的距离
            for(int i=0;i<a[t.second].size();i++){
                if(dp[a[t.second][i].to]>t.first+a[t.second][i].val){
                    dp[a[t.second][i].to]=t.first+a[t.second][i].val;
                    ans.push({dp[a[t.second][i].to],a[t.second][i].to});
                }
            }
        }
    }
    return 0;
}
```

4.1.3. SPFA（单源）--负权图

```
#include<bits/stdc++.h>
using namespace std;
#define P pair<int,int>
#define int long long
struct edge{
    int to,val; //to为这条边的终点，val为这条边的权
};
signed main(){
    int n,m,s;//n个点，m条边，s起始点
    int inf=INT_MAX;//未开long long 使用INT_MAX 可能INT_MAX+INT_MAX溢出导致错误
    cin>>n>>m>>s;
    vector<edge>a[n+1];//存图
    for(int i=0;i<m;i++){
        int u,v,w;//u->v 边权为w
        cin>>u>>v>>w;
        a[u].push_back({v,w});
        a[v].push_back({u,w});//无向图
    }
```

```
    int dp[n+1];//s至每个点的距离
    fill(dp,dp+n+1,inf);//初始s与所有点不连通
    dp[s]=0;//s至s距离为0
    int vist[n+1];//记录每个点的访问次数
    fill(vist,vist+n+1,0);//初始化为0
    int flag[n+1];//判断该点是否在队列里
    fill(flag,flag+n+1,0);//初始化为0
    queue<int>ans;
    ans.push(s);
    flag[s]=1;
    vist[s]++;
    bool is=0;//判断是否为负权图
    while(!ans.empty()){
        int t=ans.front();
        ans.pop();
        flag[t]=0;//点取出后不在队列里，标记清除
        for(int i=0;i<a[t].size();i++){
            if(dp[a[t][i].to]>dp[t]+a[t][i].val){
                dp[a[t][i].to]=dp[t]+a[t][i].val;
                if(flag[a[t][i].to]==0){
                    ans.push(a[t][i].to);
                    flag[a[t][i].to]=1;
                    vist[a[t][i].to]++;
                }
            }
            //如果一个点被访问的次数超过总点数，说明出现了负权边，应该退出
            if(vist[a[t][i].to]>n){
                is=1;
                break;
            }
        }
        if(is)
            break;
    }
    return 0;
}
```

4.2. 最小生成树

4.2.1. Kruskal

```
#include<bits/stdc++.h>
using namespace std;
struct edge{
    int v,u,val;//u,v为边的两个点，val为边权
};
const int MAX=200005;
int root[MAX];//存每个点的根节点
int tall[MAX];//存每个节点的高，优化合并
//初始化根节点为自己，高为0
void init(int n){
```

```
    for(int i=0;i<=n;i++)
        root[i]=i,tall[i]=0;
}
//递归查找根节点，并更新
int find(int n){
    if(n==root[n])
        return n;
    else
        return root[n]=find(root[n]);
}
//判断x,y是否在同一个集合
bool issame(int x,int y){
    return find(x)==find(y);
}
//合并x，y两个集合
void merge(int x,int y){
    x=find(x);
    y=find(y);
    if(x==y)
        return;
    else if(tall[x]<tall[y])
        root[y]=x;
    else{
        root[x]=y;
        if(tall[x]==tall[y])
            tall[x]++;
    }
}
bool cmp(edge a,edge b){
    return a.val<b.val;
}
signed main(){
    // n个点，m条边
    int n,m;
    cin>>n>>m;
    vector<edge>a(m);
    for(int i=0;i<m;i++){
        cin>>a[i].u>>a[i].v>>a[i].val;
    }
    sort(a.begin(),a.end(),cmp);
    // count统计已加入最小生成树的边的数量，ans统计此时最小生成树的总边权
    int count=0,ans=0;
    // 初始化每个点为未加入
    init(n);
    for(int i=0;i<m;i++){
        // 如果两个点不在一个集合说明至少有一个点未连通，加入最小生成树
        if(!issame(a[i].u,a[i].v)){
            ans+=a[i].val;
            merge(a[i].v,a[i].u);
            count++;
        }

    }
    // 加入的边数为节点数减一时，所有节点连通
```

```
    return 0;
}
```

5. 数论

5.1. 最大公约数

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

5.2. 最小公倍数

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

5.3. 快速幂

5.3.1. 第一种实现方法

```
//p为负,返回值为按|p|计算结果
int ksm(int n,int p){
    if(p==0)
        return 1;
    int he=ksm(n,p/2);
    if(p%2)
        return he*he*n;
    else
        return he*he;
}
```

5.3.2. 第二钟实现方法

```
//p不能为负
int ksm(int n,int p){
    int ans=1;
    int f=n;
    while(p){
        if(p&1)
            ans*=f;
        f*=f;
        p>>=1;
```

```
    }
    return ans;
}
```

5.4. 逆元

5.4.1. 费马小定理

$$n^{-1} = n^{p-2} \ mod \ p$$

```
const int mod=1e9+7;
int ksm(int n,int p){
    if(p==0)
        return 1;
    int he=ksm(n,p/2);
    if(p%2)
        return he*he%mod*n%mod;
    else
        return he*he%mod;
}
int inv(int n){
    return ksm(n,mod-2);
}
```

5.5. 组合数

$$C_n^m = \frac{n!}{m! \times (n-m)!}$$

```
#define int long long
const int mod=1e9+7;
int ksm(int n,int p){
    if(p==0)
        return 1;
    int he=ksm(n,p/2);
    if(p%2){
        return he*he%mod*n%mod;
    }else
        return he*he%mod;
}
int inv(int x){
    return ksm(x,mod-2);
}
//预处理阶乘
int pre[100001];
void init(){
    pre[0]=1;
    for(int i=1;i<=100000;i++){
        pre[i]=i*pre[i-1]%mod;
```

```
    }
}
//n为下标，m为上标
int C(int n,int m){
    return pre[n]*inv((pre[m]*pre[n-m])%mod)%mod;
}
```

5.6. 线性筛

```
const int MAX=100000002;
bool flag[MAX];//记录i是否为素数，值0为素数
vector<int>ans;//存储筛出的素数
void init(int n){
    flag[0]=1,flag[1]=1;
    for(int i=2;i<=n;i++){
        if(!flag[i]){
            ans.push_back(i);
        }
        for(int j=0;j<ans.size();j++){
            if(i*ans[j]>n)break;//可能会溢出，建议开long long
            flag[i*ans[j]]=1;
            if(i%ans[j]==0)break;//i%ans[j]==0说明在i之前已经被ans[j]筛过了
        }
    }
}
```

5.7. 分解质因数

```
vector<int> breakdown(int n){
    vector<int>ans;
    for(int i=2;i*i<=n;i++){
        if(n%i==0)//i为n的一个质因子
        {
            while(n%i==0)
                n/=i;//可以在此处统计该质因子的数量
            ans.push_back(i);
        }
    }
    if(n!=1)//最后一个质因子
    {
        ans.push_back(n);
    }
    return ans;
}
```

5.8. 高精度

```
struct lint{
    int gj[10010];
    lint(){
        memset(gj,0,sizeof(gj));
    }
    lint& operator * (const int& a){
        for(int i=0;i<10010;i++)
            gj[i]*=a;
        int j=0;
        for(int i=0;i<10010;i++){
            gj[i]+=j;
            j=(gj[i])/10;
            gj[i]%=10;
        }
        return *this;
    }
    lint operator / (const int& a){
        int j=0;
        lint tans;
        for(int i=10009;i>=0;i--){
            j=j*10+gj[i];
            tans.gj[i]=j/a;
            j=j%a;
        }
        return tans;
    }
    lint& operator + (const int& a){
        int j=0;
        int t=a;
        int i=0;
        while(t){
            gj[i]=gj[i]+j+t%10;
            j=gj[i]/10;
            gj[i]%=10;
            t/=10;
            i++;
        }
        return *this;
    }
    lint& operator = (const lint& a){
        for(int i=0;i<10010;i++){
            gj[i]=a.gj[i];
        }
        return *this;
    }
    bool operator == (const lint& a)const{
        for(int i=10009;i>=0;i--){
            if(gj[i]!=a.gj[i])
                return false;
        }
        return true;
    }
    bool operator < (const lint& a)const{
```

```
        for(int i=10009;i>=0;i--){
            if(gj[i]>a.gj[i])
                return false;
            if(gj[i]<a.gj[i])
                return true;
        }
        return true;
    }

};
```

6. 动态规划

6.1. 01背包

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int n,m;//n为物品数量，m为给定体积
    cin>>n>>m;
    int v[n],w[n];//v为物品体积，w为物品价值
    int dp[m+1];
    memset(dp,0,sizeof(dp));
    for(int i=0;i<n;i++)
        cin>>v[i]>>w[i];
    for(int i=0;i<n;i++)
        for(int j=m;j>=v[i];j--)
            dp[j]=max(dp[j],dp[j-v[i]]+w[i]);
    cout<<dp[m]<<endl;
}
```

6.2. 完全背包

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int n,m;//n为物品数量，m为给定体积
    cin>>n>>m;
    int v[n],w[n];//v为物品体积，w为物品价值
    int dp[m+1];
    memset(dp,0,sizeof(dp));
    for(int i=0;i<n;i++)
        cin>>v[i]>>w[i];
    for(int i=0;i<n;i++)
        for(int j=v[i];j<=m;j++){
            dp[j]=max(dp[j],dp[j-v[i]]+w[i]);
        }
    cout<<dp[m]<<endl;
}
```

7. 字符串

7.1. 字符串哈希

```
unsigned long long  Hash(string s){
    unsigned long long p=131,h=0;//p为进制，p一般取值为131或13331，h为hash值
    for(int i=0;i<s.length();i++)
        h=h*p+s[i];
    return h;//默认对2^64取模

}


//同一个字符串多次求字串
const int N=1000005;
const unsigned long long p=131;
unsigned long long h[N],pre[N];// h[i]存储字符串前i个字母的哈希值，pre[i]存储 P^i mod 2^64
//初始化
void Hash(string s){
    pre[0]=1;
    h[0]=0;
    for(int i=0;i<s.length();i++){
        h[i+1]=h[i]*p+s[i];
        pre[i+1]=pre[i]*p;
    }
}
//计算子串s[l->r]的哈希值,l、r为下标
unsigned long long get(int l,int r){
    return h[r+1]-h[l]*pre[r-l+1];
}
```

8. 计算几何

9. 其他

```
//矩阵乘法    案例    斐波那契数列
#include<bits/stdc++.h>
using namespace std;
const int mod = 1e9+7;
struct matrix{
    long long a[3][3];
    matrix(){memset(a,0,sizeof(a));}
    matrix operator*(const matrix &b)const {
        matrix res;
        for(int i=1;i<=2;i++)
            for(int j=1;j<=2;j++)
```

```
                for(int k=1;k<=2;k++)
                    res.a[i][j]=(res.a[i][j]+a[i][k]*b.a[k][j])%mod;
        return res;
    }
    void sete(int x){
        for(int i=1;i<=2;i++)
            a[i][i]=x;
    }

};
matrix qpow(matrix m,long long n)
{
    matrix ans;
    ans.sete(1);
    while(n>0)
    {
        if(n&1)
            ans=ans*m;
        m=m*m;
        n>>=1;
    }
    return ans;
}
void solve()
{
    matrix trans,f;
    trans.a[1][1]=trans.a[1][2]=trans.a[2][1]=1;
    f.a[1][1]=f.a[2][1]=1;
    long long n;
    scanf("%lld",&n);
    trans=qpow(trans,n-2);
    trans=trans*f;
    printf("%lld",trans.a[1][1]%mod);
}
int main()
{

solve();
    return 0;
}
```