

//匈牙利算法 解决二分图最大匹配

bool Hungry(int u) {

for(int i=0; i<Edge[u].size(); i++) {

int v=Edge[u][i];

if(!Mark[v]) {

Mark[v]=1;

if(Fa[v]==-1||Hungry(Fa[v])) {

Fa[v]=u;

return true;

}

}

}

return false;

}

int main() {

memset(Fa,-1,sizeof(Fa));

for(int i=1; i<=n; i++) {

memset(Mark,0,sizeof(Mark));

if(Hungry(i))Ans++;

}

//双向建边 最大匹配=Ans/2

//双向建边 如果染色 if(Col[i]&&Hungry(i)) 最大匹配=Ans

//最小点覆盖 = 最大匹配

//最小独立集 = n - 最大匹配

//最小不可相交路径覆盖 = n - 最大匹配

//最小可相交路径覆盖 =

// 用floyd求出原图的传递，若a到b有路径，那么a->b。然后就转化成最小不相交路径覆盖问题

}

//并查集

int Find(int x) {

if(Fa[x]==x)return Fa[x];

return Fa[x]=Find(Fa[x]);

}

//可删除并查集

void Solve() {

Num=n;

memset(Mark,0,sizeof(Mark));

for(int i=1; i<=n; i++)

Fa[i]=Id[i]=i;

if("加入") {

scanf("%d%d",&a,&b);

int x=Find(Id[a]),y=Find(Id[b]);

Fa[x]=y;

} else if("删除") {

scanf("%d",&a);

Fa[++Num]=Id[a]=Num;

}

}

//带权并查集

int Find(int x) {

if(Fa[x]==x)return Fa[x];

int Tmp=Fa[x];

Fa[x]=Find(Fa[x]);

Dep[x]=Dep[x]+Dep[Tmp];

return Fa[x];

}

int main() {

for(int i=0; i<=n; i++)

Fa[i]=i,Dep[i]=0;

for(int i=1,x,y,s; i<=m; i++) {

scanf("%d%d%d",&x,&y,&s);

int Fx=Find(x),Fy=Find(y);

if(Fx!=Fy) {

Dep[Fx]=Dep[y]+s-Dep[x];

Fa[Fx]=Fy;

}

}

}

//树状数组

int Sum[50005];

int Lowbit(int x) {

return x&-x;

}

int Query(int x) {//区间和

int res=0;

while(x>0) {

res+=Sum[x];

x-=Lowbit(x);

}

return res;

}

void Add(int x,int d) {

while(x<=n)

Sum[x]+=d,x+=Lowbit(x);

}

/\*

void Updata(int x){//最大值

while(x<=n){

Max[x]=A[x];

int lx=Lowbit(x);

for(int i=1; i<lx; i<<=1)

Max[x]=max(Max[x],Max[x-i]);

x+=Lowbit(x);

}

}

int Query(int L,int R){

int Ans = 0;

while(R>=L){

Ans=max(A[R],ans);

R--;

for(;R-lowbit(R)>=L;R-=lowbit(R))

Ans=max(Max[R],Ans);

}

return Ans;

}

\*/

//单调栈 解决i向左右查询第一个值

//单调队列 解决i从去心领域转移值 多用于动态规划

//点分治

int Mn=INF,Root,Smer;

void Get\_Root(int u,int Fa) {

Son[u]=1,Max\_Son[u]=0;

for(int i=Head[u]; i!=-1; i=Edge[i].Next) {

int v=Edge[i].To;

if(Vis[v]||v==Fa)continue;

Get\_Root(v,u);

Son[u]+=Son[v];

Max\_Son[u]=max(Max\_Son[u],Son[v]);

}

Max\_Son[u]=max(Max\_Son[u],Smer-Son[u]);

if(Max\_Son[u]<Mn) {

Mn=Max\_Son[u];

Root=u;

}

}

int Num;

void Get\_Dis(int u,int Fa,int dis) {//此处对应题型处理

Dis[++Num]=dis;

for(int i=Head[u]; i!=-1; i=Edge[i].Next) {

int v=Edge[i].To;

if(Vis[v]||v==Fa)continue;

Get\_Dis(v,u,dis+Edge[i].c);

}

}

int Calc(int u,int Len) {//单独计算每个子树的贡献

Num=0;

Get\_Dis(u,0,Len);

sort(Dis+1,Dis+Num+1);

int Cnt=0;

int L=1,R=Num,m;

while(L<=R) {

if(Dis[L]+Dis[R]>k)R--;//此处对应题型处理

else Cnt+=R-L,L++;

}

return Cnt;

}

void Solve(int u) {

Ans+=Calc(u,0);//容斥 有些题目不需要

Vis[u]=1;

for(int i=Head[u]; i!=-1; i=Edge[i].Next) {

int v=Edge[i].To;

if(Vis[v])continue;

Ans-=Calc(v,Edge[i].c);//容斥 有些题目不需要

Mn=INF,Root=0,Smer=Son[v];

Get\_Root(v,u);

Solve(Root);

}

}

//注意初始化 Smer=n Mn=INF

//St表 用于处理区间最大最小值 速度快于线段树

void St() {

for(int i=1; i<=n; i++)

dp[i][0]=A[i];//初始化

for(int j=1; (1<<j)<=n; j++)

for(int i=1; i+(1<<j)-1<=n; i++)

dp[i][j]=min(dp[i][j-1],dp[i+(1<<j-1)][j-1]);//min or max

}

int Query(int l,int r) {

int k=log2(r-l+1);

return min(dp[l][k],dp[r-(1<<k)+1][k]);

}

//线性基 解决求在一个序列中，取若干个数，使得它们的异或和最大

void Add(long long x) {

for(int i=60; i>=0; i--)

if(x&(1ll<<i)) {

if(D[i])x^=d[i];

else {

D[i]=x;

break;//插入成功就退出

}

}

}

long long ans() {//最大

ll anss=0;

for(int i=60; i>=0; i--) //记得从线性基的最高位开始

if((anss^d[i])>anss)anss^=d[i];

return anss;

}

/\*

如果是求整个序列能异或出的最小值

要另外看一看有没有元素不能插入线性基,如果有,那么最小值就是0

否则依然是最小的d[i]

如何判断一个数是否能被当前线性基中的元素异或得到

把它尝试插入进线性基里面去，假如可以插入，说明不能异或得到

\*/

//快速幂

long long Calc(long long a, long long p) {

long long Ans=1;

while(p) {

if(p&1)

Ans=(Ans\*a)%Mod;

a=(a\*a)%Mod;

p>>=1;

}

return Ans;

}

//最大公约数

long long Gcd(long long a,long b) {

if(b==0)return a;

return Gcd(b,a%b);

}

//费马逆元O1组合数

long long Fac[4100010],Inv[4100010];

void pre() {

Fac[0]=Fac[1]=Inv[0]=Inv[1]=1;

for(int i=2; i<=4100000; i++)

Fac[i]=Fac[i-1]\*i%Mod,Inv[i]=(Mod-Mod/i)\*Inv[Mod%i]%Mod;

for(int i=2; i<=4100000; i++)

Inv[i]=Inv[i-1]\*Inv[i]%Mod;

}

long long C(int m,int n) {

return Fac[m]\*Inv[m-n]%Mod\*Inv[n]%Mod;

}

//费马小定理求逆元

long long Pow(long long a,long long b) { //a的b次方求余p

long long ret=1;

while(b) {

if(b&1)ret=(ret\*a)%Mod;

a=(a\*a)%Mod;

b>>=1;

}

return ret;

}

long long Fermat(long long a) { //费马求a关于b的逆元

return Pow(a,p-2);

}

//欧拉筛

void Oula(int n) {

for(int i=2; i<=n; i++) {

if(!Vis[i])Prime[++Prime[0]]=i;

for(int j=1; j<=Prime[0]; j++) {

if(i\*Prime[j]>n) break;

Vis[i\*Prime[j]]=true;

if(i%Prime[j]==0)break;

}

}

}

//矩阵快速幂

void multi(long long A[][N],long long B[][N]) {

memset(Tmp,0,sizeof(Tmp));

for(int i=0; i<n; i++)

for(int j=0; j<n; j++)

for(int k=0; k<n; k++) {

Tmp[i][j]+=A[i][k]\*B[k][j]%m;

Tmp[i][j]%m;

}

for(int i=0; i<n; i++)

for(int j=0; j<n; j++)

A[i][j]=Tmp[i][j]%m;

}

void Pow(long long A[][N],long long x) {

memset(Ans,0,sizeof(Ans));

for(int i=0; i<n; i++)Ans[i][i]=1;

while(x) {

if(x&1)

multi(Ans,A);

multi(A,A);

x/=2;

}

for(int i=0; i<n; i++)

for(int j=0; j<n; j++)

A[i][j]=Ans[i][j]%m;

}

//FFT

const double Pi=acos(-1.0);

struct Complex {

double x,y;

Complex (double xx=0,double yy=0) {

x=xx,y=yy;

}

} A[10000005],B[10000005];

Complex operator+(Complex A,Complex B) {

return Complex(A.x+B.x,A.y+B.y);

}

Complex operator-(Complex A,Complex B) {

return Complex(A.x-B.x,A.y-B.y);

}

Complex operator\*(Complex A,Complex B) {

return Complex(A.x\*B.x-A.y\*B.y,A.x\*B.y+A.y\*B.x);

}

int n,m,lim;

int l,r[10000005];

void FFT(Complex \*A,int f) {

for(int i=0; i<lim; i++)

if(i<r[i])swap(A[i],A[r[i]]);

for(int mid=1; mid<lim; mid<<=1) {

Complex Tmp( cos(Pi/mid) , f\*sin(Pi/mid) );

for(int R=mid<<1,j=0; j<lim; j+=R) {

Complex w(1,0);

for(int k=0; k<mid; k++,w=w\*Tmp) {

Complex x=A[j+k],y=w\*A[j+mid+k];

A[j+k]=x+y;

A[j+mid+k]=x-y;

}

}

}

}

int main() {

scanf("%d%d",&n,&m);

//0-n 的多项式 卷积 0-m的多项式

for(int i=0; i<=n; i++)

scanf("%lf",&A[i].x);

for(int i=0; i<=m; i++)

scanf("%lf",&B[i].x);

lim=1,l=0;

while(lim<=n+m)

lim<<=1,l++;

for(int i=0; i<lim; i++)

r[i]=(r[i>>1]>>1)|((i&1)<<(l-1));

FFT(A,1),FFT(B,1);

for(int i=0; i<=lim; i++)

A[i]=A[i]\*B[i];

FFT(A,-1);

for(int i=0; i<=n+m; i++)

printf("%d ",(int)(A[i].x/lim+0.5));

}

//NTT

const int Mod=998244353,P1=3,P2=332748118;

int n,m,lim;

int l,r[10000005];

long long A[10000005],B[10000005];

long long Calc(long long a, long long p) {

long long Ans=1;

while(p) {

if(p&1)

Ans=(Ans\*a)%Mod;

a=(a\*a)%Mod;

p>>=1;

}

return Ans;

}

void NTT(long long \*Tmp,int tp) {

for(int i=0; i<lim; i++)if(i<r[i])swap(Tmp[i],Tmp[r[i]]);

for(int i=1; i<lim; i<<=1) {

long long W=Calc(tp?P1:P2,(Mod-1)/(i<<1));

for(int j=i<<1,k=0; k<lim; k+=j) {

long long w=1;

for(int l=0; l<i; l++,w=(w\*W)%Mod) {

int x=Tmp[k+l],y=w\*Tmp[k+i+l]%Mod;

Tmp[k+l]=(x+y)%Mod;

Tmp[k+i+l]=(x-y+Mod)%Mod;

}

}

}

}

int main() {

scanf("%d%d",&n,&m);

for(int i=0; i<=n; i++)

scanf("%ld",&A[i]);

for(int i=0; i<=m; i++)

scanf("%ld",&B[i]);

lim=1,l=0;

while(lim<=n+m)lim<<=1,l++;

for(int i=0; i<lim; i++)

r[i]=(r[i>>1]>>1)|((i&1)<<(l-1));

NTT(A,1),NTT(B,1);

for(int i=0; i<=lim; i++)

A[i]=(A[i]\*B[i])%Mod;

NTT(A,0);

long long inv=Calc(lim,Mod-2);

for(int i=0; i<=n+m; i++)

printf("%d ",(A[i]\*inv)%Mod);

}

//邻接表

struct node {

int To,c;

int Next;

} Edge[Maxn\*4];

int Head[Maxn\*4];

void Add(int x,int y,int c) {

Edge[++tot].To=y;

Edge[tot].c=c;

Edge[tot].Next=Head[x];

Head[x]=tot;

}

int main() {

for(int i=1; i<=2\*n; i++)

Edge[i].Next=Head[i]=-1;

for(int i=Head[x];i!=-1; i=Edge[i].Next) {

}

}

//Tarjan缩点后注意只有一个点的情况

//Trajan

void Tarjan(int u) {

S.push(u);

Mark[u]=1;

Dfn[u]=Low[u]=++Cnt1;

for(int i=0; i<Edge[u].size(); i++) {

int v=Edge[u][i];

if(!Dfn[v]) {

Tarjan(v);

Low[u]=min(Low[u],Low[v]);

} else if(Mark[v]) {

Low[u]=min(Low[u],Dfn[v]);

}

}

if(Dfn[u]==Low[u]) {

++Cnt2;

while(1) {

int x=S.top();

S.pop();

Mark[x]=0,Belong[x]=Cnt2;

if(u==x)break;

x=S.top();

}

}

}

//割点(删除该点出现不连通地情况) Tarjan

void Tarjan(int u,int root) {

int Child=0;

Dfn[u]=Low[u]=++Cnt;

for(int i=0; i<Edge[u].size(); i++) {

int v=Edge[u][i];

if(!Dfn[v]) {

Tarjan(v,root);

Low[u]=min(Low[u],Low[v]);

if(Low[v]>=Dfn[u]&&u!=root)//求割边此处为Low[u]>Dfn[v]

Mark[u]=1;

if(u==root)

Child++;

} else

Low[u]=min(Low[u],Dfn[v]);

}

if(u==root&&Child>=2)//特判根部

Mark[u]=1;

}

//拓扑排序 字典序最小 小顶堆优先队列

//拓扑排序 最小序列优先 大顶堆反向建边优先队列

//LCA 倍增

void Dfs(int u,int F) {

Mark[u]=1;

Fa[u][0]=F;

for(int i=1; i<=25; i++)

Fa[u][i]=Fa[Fa[u][i-1]][i-1];

for(int i=0; i<Edge[u].size(); i++) {

int v=Edge[u][i].To;

int c=Edge[u][i].Cost;

if(v==F)continue;

Dep[v]=Dep[u]+1;

Dfs(v,u);

}

}

void Up(int &x,int step) {

for(int i=0; i<=25; i++)

if(step&(1<<i))

x=Fa[x][i];

}

int Find(int x,int y) {

if(Dep[x]>Dep[y])swap(x,y);

Up(y,Dep[y]-Dep[x]);

if(x==y)return x;

for(int i=25; i>=0; i--)

if(Fa[x][i]!=Fa[y][i]) {

x=Fa[x][i];

y=Fa[y][i];

}

return Fa[x][0];

}

//DFS序

void Dfs(int u,int F) {

L[x]=++Cnt;

for(int i=0; i<Edge[x].size(); i++) {

int v=Edge[x][i];

if(v!=F)

dfs(v,u);

}

R[x]=Cnt;

}

//克鲁斯卡尔重构树 解决路径 最大|最小 边权|点权 问题

int main() {

for(int i=1,x,y; i<=m; i++) {

scanf("%d%d%d",&x,&y,&z);//连边 z=边权 边权按照顺序给出

int Fx=Find(x),Fy=Find(y);

if(Fx!=Fy) {

Son[++Cnt]=Sum[Fx]+Sum[Fy];//更新树上点权 根据题意变化

//构造克鲁斯卡尔树

Edge[Cnt].push\_back(node {Id[Fx],z});

Edge[Cnt].push\_back(node {Id[Fy],z});

Id[Fx]=Cnt;//Id 维护Fx指向的构造点

Fa[Fy]=Fx;

Sum[Fx]+=Sum[Fy];//维护树上权值的同时 维护并查集大小

Rt=Cnt;//根节点

}

//每个权值点作为叶节点

}//树是由 叶节点 和 构造出来的点组成根节点 组成的

}

//最短路 迪杰斯特拉算法

struct E {

int To,w;

};

vector<E>Edge[200005];

struct node {

int id,Dis;

bool operator <(const node&x)const {

return x.Dis<Dis;

}

};

priority\_queue<node>Q;

void Dij(int k) {

memset(Dis,-1,sizeof(Dis));

memset(Mark,0,sizeof(Mark));

Dis[k]=0;

Q.push(node {k,0});

while(!Q.empty()) {

int u=Q.top().id;

Q.pop();

if(Mark[u])continue;

Mark[u]=1;

for(int i=0; i<Edge[u].size(); i++) {

int v=Edge[u][i].To;

int c=Edge[u][i].w;

if(Dis[v]==-1||Dis[v]>Dis[u]+c) {

Dis[v]=Dis[u]+c;

Q.push(node {v,Dis[v]});

}

}

}

}

//如果边权只有0/1 求最短路 可用双向队列Bfs O(n)

//0的边放队列队首 1的边放队尾

//网络流

bool Bfs() {

for(int i=1; i<=n; i++)Dis[i]=-1;

Q.push(s);

Dis[s]=0;

while(!Q.empty()) {

int u=Q.front();

Q.pop();

for(int i=0,v; i<Edge[u].size(); i++) {

v=Edge[u][i];

c=A[u][v];

if(c&&Dis[v]==-1) {

Dis[v]=Dis[u]+1;

Q.push(v);

}

}

}

return Dis[t]!=-1;

}

int Dfs(int u,int Min) {

if(u==t)return Min;

long long Last=Min;

for(int i=0,v; i<Edge[u].size(); i++) {

v=Edge[u][i];

long long c=A[u][v];

if(c&&Dis[v]==Dis[u]+1) {

long long Tmp=Dfs(v,min(Last,c));

A[u][v]-=Tmp;

A[v][u]+=Tmp;

Last-=Tmp;

if(Last==0)break;

}

}

return Min-Last;

}

void Add(int u,int v,int c) {

if(A[u][v]==0) {//重边可能

Edge[u].push\_back(v);

Edge[v].push\_back(u);

}

A[u][v]+=c;//重边可能

}

int main() {

while(Bfs())Ans+=Dfs(s,Max);

}

int Next[200005];

void Get(string S) {

int n=S.size();

int j=-1;

Next[0]=j;

for(int i=1; i<n; i++) {

while(j>=0&S[i]!=S[j+1])

j=Next[j];

if(S[j+1]==S[i])j++;

Next[i]=j;

}

}

void Kmp(string T,string S) {

int n=T.size(),m=S.size();

int j=-1;

for(int i=0; i<n; i++) {

while(j>=0&&T[i]!=S[j+1])

j=Next[j];

if(T[i]==S[j+1])j++;

if(j==m-1) {

printf("%d\n",i-m+2);

}

}

}

#include<bitsdc++.h>  
#define ll long long  
using namespace std;  
const int M=1e6+3;  
struct node{  
    int x,opt,id;  
}a[M];  
long long ans[M];  
struct SAM {  
int maxlen[M\*2], trans[M\*2][26], link[M\*2], size, last;  
SAM() { size = last = 1; }  
inline void extend(int id) {  
int cur = (++ size), p;  
maxlen[cur] = maxlen[last] + 1;  
for (p = last; p && !trans[p][id]; p = link[p]) trans[p][id] = cur;  
if (!p) link[cur] = 1;  
else {  
int q = trans[p][id];  
if (maxlen[q] == maxlen[p] + 1) link[cur] = q;  
else {  
int clone = (++ size);  
maxlen[clone] = maxlen[p] + 1;  
for(int i=0;i<26;i++)trans[clone][i]=trans[q][i];  
link[clone] = link[q];  
for (; p && trans[p][id] == q; p = link[p]) trans[p][id] = clone;  
link[cur] = link[q] = clone;  
}  
}   
last = cur;  
}  
inline long long gao(){//本质不同字符字串个数   
        ll ans=0;  
         for(int i=2;i<=last;++i)ans+=maxlen[i]-maxlen[link[i]];  
        return ans;  
    }  
}sam;  
char s[M];  
int n,q;  
long long f[M];  
bool cmp(node a,node b){  
    return a.x<b.x;  
}  
int main(){  
    scanf("%d%d",&n,&q);  
scanf("%s",s+1);  
    // int tot=0;  
    // for (int i=1;i<=q;i++){  
    //     int l; int r;  
    //     scanf("%d%d",&l,&r);  
    //     tot++;  
    //     a[tot].x=l-1;  
    //     a[tot].opt=-1;  
    //     a[tot].id=i;  
    //     tot++;  
    //     a[tot].x=r;  
    //     a[tot].opt=1;  
    //     a[tot].id=i;  
    // }  
    // int k=1;  
    // sort(a+1,a+1+tot,cmp);  
   // for (int i=1;i<=tot;i++) cout<<a[i].x<<&apos; &apos;<<a[i].opt<<&apos; &apos;<<a[i].id<<endl;  
    // while (a[k].x==0) k++;   
    // long long last=0;  
for(int i=1;i<=n;i++){  
        sam.extend(s[i]-&apos;a&apos;);  
        f[i]=f[i-1]+sam.gao();  
        // b[i]=sam.gao()-last;  
        // last=sam.gao();  
        // cout<<sam.gao()<<&apos; &apos;;  
        // while (k<=tot&&a[k].x==i){  
        //     ans[a[k].id]+=sam.gao()\*a[k].opt;  
        //     k++;  
        // }  
    }  
    // cout<<endl;  
    // for (int i=1;i<=n;i++) cout<<b[i]<<&apos; &apos;;  
    // cout<<endl;  
    // for (int i=1;i<=q;i++) printf("%lld\n",ans[i]);  
    while (q--){  
        int l,r;  
        scanf("%d%d",&l,&r);  
        printf("%lld\n",f[r]-f[l-1]);  
    }  
    return 0;  
}

#include<bitsdc++.h>  
using namespace std;  
typedef long long ll;  
const int N=4e5+10;  
char s[N];  
int m;  
long long su[10][N];  
struct Palindromic\_Tree{  
int next[N][26];//next指针，next指针和字典树类似，指向的串为当前串两端加上同一个字符构成  
int fail[N];//fail指针，失配后跳转到fail指针指向的节点     最长回文后缀   
int len[N];//len[i]表示节点i表示的回文串的长度    
int S[N];//存放添加的字符  
ll cnt[N];//结点表示的本质不同的回文串的个数(调用count()后)  
int num[N];//结点表示的最长回文串的最右端点为回文串结尾的回文串个数  
int last;//指向上一个字符所在的节点，方便下一次add   
int n;//字符数组指针    
    int p;//节点指针  
       
int newnode(int x){//新建节点   
memset(next[p],0,sizeof(next[p]));  
cnt[p]=0;  
num[p]=0;  
len[p]=x;  
return p++;  
}  
void init(){  
p=0;  
newnode(0);  
newnode(-1);  
last=0;  
n=0;  
S[0]=-1;//开头放一个字符集中没有的字符，减少特判   
fail[0]=1;  
}  
int get\_fail(int x){//和KMP一样，失配后找一个尽量最长的    
while(S[n-len[x]-1]!=S[n]) x=fail[x];  
return x;  
}   
void add(int c,int k){  
c-=&apos;a&apos;;  
S[++n]=c;  
int cur=get\_fail(last);//通过上一个回文串找这个回文串的匹配位置  
if(!next[cur][c]){//如果这个回文串没有出现过，说明出现了一个新的本质不同的回文串  
int now=newnode(len[cur]+2);//新建节点  
fail[now]=next[get\_fail(fail[cur])][c];//和AC自动机一样建立fail指针，以便失配后跳转    
num[now]=num[fail[now]]+1;  
next[cur][c]=now;  
            su[k][now]=su[k-1][now]+1;  
}   
last=next[cur][c];  
cnt[last]++;  
        su[k][last]=su[k-1][last]+1;  
}  
void count(){  
for(int i=p-1;i>=0;i--) cnt[fail[i]]+=cnt[i];  
//父亲累加儿子的cnt，因为如果fail[v]=u，则u一定是v的子回文串！  
}  
}pam;  
int main(){  
    scanf("%d",&m);  
    pam.init();  
    for (int \_=1;\_<=m;\_++){  
        scanf("%s",s);  
     int len=strlen(s);   
     for(int i=0;i<len;i++){  
     pam.add(s[i],\_);  
     //printf("%d%c",pam.p-2," \n"[i==len-1]);  
}  
        if (\_!=m){  
            pam.add(&apos;\*&apos;,\_);  
            pam.add(&apos;#&apos;,\_);  
        }  
        // long long y=pam.solve(\_);  
    }  
    long long ans=0;  
    for (int i=2;i<=pam.p-1;i++)  
        if (su[m][i]>=m) ans++;  
    printf("%lld\n",ans);  
}