题解

20221114

T1 (Substr HNOI2006 最短母串问题)

考虑对给出的所有字符串建立AC自动机，AC自动机上的fail边指向最长的相同前后缀，所以可以直接沿树边BFS，第一个找到的结果即为答案。

用二进制数表示搜索时的状态，某一位为1则表示对应字符串已经被完全包含，为0则表示还未被包含。在插入每一个字符串时，在结束点打上标记，注意要用按位或而非直接赋值，因为有可能有重复的字符串。建立fail边时，需要将结束标记沿边传递，因为两个节点对应的字符串包含相同的前缀，且该前缀中包含有输入的字符串。搜索时记录一棵搜索树，标记搜索的路径，覆盖所有字符串后，沿搜索树回到树根即可求得最短的答案。时间复杂度$O(n \times 2^n)$

```
#include<bits/stdc++.h>
using namespace std;
template<typename T>
void read(T &res) {
    res=0;
    char ch=getchar();
    bool f=false;
    while(ch<'0'||ch>'9') {
        if(ch=='-') f=true;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9') {
        res=(res<<1)+(res<<3)+(ch^48);
        ch=getchar();
    }
    res=f?-res:res;
}
template<typename T,typename ...Args>
void read(T &res,Args &...args) {
    read(res);
    read(args...);
}
int t[1001][27],tot1,tot2,n,len,lans;
int fail[1001],fa[26000001],ans[26000001],ed[1001];
char str[51];
bool vis[1001][1<<14|1];
queue<int>q1;
void insert(int id,char *s) {
    int u=0;
    for(int i=1;s[i];i++) {
        if(!t[u][s[i]-'A']) {
            t[u][s[i]-'A']=++tot2;
        }
        u=t[u][s[i]-'A'];
    }
    ed[u]|=(1<<(id-1));
}
void build() {
```

```
    for(int i=0;i<='Z'-'A';i++) {
        if(t[0][i]) {
            q1.push(t[0][i]);
        }
    }
    while(!q1.empty()) {
        int u=q1.front();
        q1.pop();
        for(int i=0;i<='Z'-'A';i++) {
            if(t[u][i]) {
                fail[t[u][i]]=t[fail[u]][i];
                ed[t[u][i]]|=ed[t[fail[u]][i]];
                q1.push(t[u][i]);
            } else {
                t[u][i]=t[fail[u]][i];
            }
        }
    }
}
char tmp[1001];
queue<pair<int,int> >q;
void bfs() {
    int dcnt=0,tot=0;
    q.push(make_pair(0,0));
    vis[0][0]=true;
    while(!q.empty()) {
        int u=q.front().first;
        int s=q.front().second;
        q.pop();
        if(s==(1<<n)-1) {
            int lans=0;
            while(dcnt) {
                tmp[++lans]=ans[dcnt];
                dcnt=fa[dcnt];
            }
            for(int i=lans;i>=1;i--) {
                putchar(tmp[i]);
            }
            putchar('\n');
            break;
        }
        for(int i=0;i<='Z'-'A';i++) {
            if(!vis[t[u][i]][s|ed[t[u][i]]]) {
                vis[t[u][i]][s|ed[t[u][i]]]=true;
                q.push(make_pair(t[u][i],s|ed[t[u][i]]));
                fa[++tot]=dcnt;
                ans[tot]=i+'A';
            }
        }
        ++dcnt;
    }
}
int main() {
    freopen("substr.in","r",stdin);
    freopen("substr.out","w",stdout);
    read(n);
    tot1=1;
    for(int i=1;i<=n;i++) {
```

```
        memset(str,0,sizeof(str));
        scanf("%s",str+1);
        insert(i,str);
    }
    build();
    bfs();
    return 0;
}
```

T2 (flower)

分块，对于每个块，考虑预处理出对于1到10^5的所有k的答案。对每个块，建立一个大小为10^5的桶，再更新成前缀最大值。对于所有属于区间$[ak,a(k+1)-1]$的数，它们的最大值就是这些数中答案最大的那一个。枚举1到10^5，对于每个k，枚举所有可能的a，通过与处理的前缀最大值求出答案。设块长为b，则预处理每个块的时间复杂度为$\sum_{i=1}^{k} k/i$，即为$O(k\ln n)$，预处理总复杂度为$O(\frac{nk\ln n}{b})$。

查询即为常规的分块查询，故查询总复杂度为$O(mb+\frac{mn}{b})$。

总时间复杂度为$O(\frac{nk\ln k}{b}+mb+\frac{mn}{b})$，当$b=\sqrt{n+\frac{nk\ln k}{m}}$时取得最小值，为$O(m\sqrt{n+\frac{nk\ln k}{m}})$，其中，$nmk_{max}$均取得极限时，块长为1086最优，离线并根据$k_{max}$求块长会略快一些，时间限制建议扩大至2s。

```
#include<bits/stdc++.h>
using namespace std;
template<typename T>
void read(T &res) {
    res=0;
    char ch=getchar();
    bool f=false;
    while(ch<'0'||ch>'9') {
        if(ch=='-') f=true;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9') {
        res=(res<<1)+(res<<3)+(ch^48);
        ch=getchar();
    }
    res=f?-res:res;
}
template<typename T,typename ...Args>
void read(T &res,Args &...args) {
    read(res);
    read(args...);
}
template<typename T,typename ...Args>
void write(const T res) {
    if(res>9) {
        write(res/10);
    }
    putchar((res%10)^48);
}
int blsize,blcnt,a[100001],maxk,n,m;
class Block {
public:
    Block() {
        memset(ans,0,sizeof(ans));
```

```
        size=0;
    }
    int size,ans[100001];
}bl[1001];
int f[100001];
void init() {
    for(int i=1;i<=n;i++) {
        maxk=max(maxk,a[i]);
    }
    blsize=1086;
    for(int i=1;i<=n;i+=blsize) {
        memset(f,0,sizeof(f));
        ++blcnt;
        for(int j=i;j<=i+blsize-1;j++) {
            f[a[j]]=a[j];
        }
        for(int j=1;j<=100000;j++) {
            f[j]=max(f[j],f[j-1]);
        }
        for(int j=1;j<=100000;j++) {
            for(int k=0;k<=100000;k+=j) {
                bl[blcnt].ans[j]=max(bl[blcnt].ans[j],f[min(k+j-1,maxk)]%j);
            }
        }
    }
}
int query(int l,int r,int k) {
    int lb=(l-1)/blsize+1;
    int rb=(r-1)/blsize+1;
    int res=0;
    for(int i=l;i<=min(lb*blsize,r);i++) {
        res=max(res,a[i]%k);
    }
    for(int i=lb+1;i<=rb-1;i++) {
        res=max(res,bl[i].ans[k]);
    }
    for(int i=max((rb-1)*blsize+1,l);i<=r;i++) {
        res=max(res,a[i]%k);
    }
    return res;
}
int main() {
    freopen("flower.in","r",stdin);
    freopen("flower.out","w",stdout);
    read(n,m);
    for(int i=1;i<=n;i++) {
        read(a[i]);
    }
    init();
    for(int i=1;i<=m;i++) {
        int l=0,r=0,k=0;
        read(l,r,k);
        write(query(l,r,k));
        putchar('\n');
    }
    return 0;
}
```

T3 (refract)

考虑dp，$dp_{i,j}$表示按x排序，以第i个点为入射点的光线数，其中$j=0$表示左下方向右上方的折射，$j=1$表示右下方向左上方的折射。对于每组i，j，若$y_j > y_i$，则将$dp_{j,1}$加上$dp_{i,0}$；若$y_i < y_j$，则将$dp_{j,1}$加上$dp_{i,0}$，最终答案即为$\sum_{i=1}^{n} dp_{i,0}+dp_{i,1}-1$，时间复杂度$O(n^2)$

```
#include<bits/stdc++.h>
using namespace std;
template<typename T>
void read(T &res) {
    res=0;
    char ch=getchar();
    bool f=false;
    while(ch<'0'||ch>'9') {
        if(ch=='-') f=true;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9') {
        res=(res<<1)+(res<<3)+(ch^48);
        ch=getchar();
    }
    res=f?-res:res;
}
template<typename T,typename ...Args>
void read(T &res,Args &...args) {
    read(res);
    read(args...);
}
class Point {
public:
    Point() {
        x=y=0;
    }
    int x,y;
}p[6001];
bool operator<(const Point a,const Point b) {
    return a.x<b.x;
}
const long long mod=100000000711;
long long dp[6001][2],ans;
int n;
int main() {
    freopen("refract.in","r",stdin);
    freopen("refract.out","w",stdout);
    read(n);
    for(int i=1;i<=n;i++) {
        read(p[i].x,p[i].y);
    }
    sort(p+1,p+n+1);
    dp[1][0]=dp[1][1]=1;
    for(int i=2;i<=n;i++) {
        dp[i][0]=dp[i][1]=1;
        for(int j=i-1;j>=1;j--) {
            if(p[j].y>p[i].y) {
                dp[j][1]=(dp[j][1]+dp[i][0])%mod;
            } else {
                dp[i][0]=(dp[i][0]+dp[j][1])%mod;
```

```
            }
        }
    }
    for(int i=1;i<=n;i++) {
        ans+=dp[i][0];ans%=mod;
        ans+=dp[i][1]-1;ans%=mod;
    }
    printf("%lld\n",ans);
    return 0;
}
```

T4 (paint)

只有聪明人才看得到的std:

```
```

20221115

T1 (Reverse)

考虑BFS，每个的点可以一步到达的点可以在 $O(k)$ 的时间内求出，由于边权均为 1，每个点被第一次访问时的步数即为答案。时间复杂度 $O(nk)$。

考虑优化，算法主要问题在于每访问一个点都一定会扩展 $k/2$ 次，而这当中有很多点已经被访问过。可以发现，当k为奇数时，无论如何翻转，最终1所在位置的编号的奇偶都和s的奇偶相同；当k为偶数时，每次翻转后1所在位置编号的奇偶都会变化一次。但是无论如何，每次翻转可到达所有点的奇偶都相同。所以可以用两个set来维护当前还没有访问的点，对于每个起点，算出可到达的区间，用lower_bound从对应set中取出区间中的第一个点，向后枚举直到离开区间。这样避免了重复访问，时间复杂度$O(n\log n)$

```
#include<bits/stdc++.h>
using namespace std;
template<typename T>
void read(T &res) {
    res=0;
    char ch=getchar();
    bool f=false;
    while(ch<'0'||ch>'9') {
        if(ch=='-') f=true;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9') {
        res=(res<<1)+(res<<3)+(ch^48);
        ch=getchar();
    }
    res=f?-res:res;
}
template<typename T,typename ...Args>
void read(T &res,Args &...args) {
    read(res);
    read(args...);
}
template<typename T>
```

```cpp
void write(T res) {
    if(res<0) {
        putchar('-');
        res=-res;
    }
    if(res>9) {
        write(res/10);
    }
    putchar((res%10)^'0');
}
template<typename T,typename ...Args>
void write(T res,Args ...args) {
    write(res);
    write(args...);
}
int n,k,m,s,cnt;
bool vis[100001];
int ans[100001];
queue<pair<int,int> >q;
set<int>st1,st2;
int main() {
    freopen("reverse.in","r",stdin);
    freopen("reverse.out","w",stdout);
    read(n,k,m,s);
    memset(vis,0,sizeof(vis));
    memset(ans,-1,sizeof(ans));
    for(int i=1;i<=m;i++) {
        int tmp=0;
        read(tmp);
        vis[tmp]=true;
        ++cnt;
    }
    for(int i=1;i<=n;i++) {
        if(i==s) continue;
        if(vis[i]) continue;
        if(i&1) st1.insert(i);
        else st2.insert(i);
    }
    ans[s]=0;
    q.push(make_pair(s,0));
    while(!q.empty()) {
        int u=q.front().first;
        int st=q.front().second;
        q.pop();
        if(vis[u]) continue;
        vis[u]=true;
        ans[u]=st;
        int ll=max(1,u-k+1);
        int rr=min(n,u+k-1);
        int lr=ll+k-1;
        int rl=rr-k+1;
        int l=ll+lr-u;
        int r=rl+rr-u;
        if(((k&1)&&(u&1))||(!(k&1)&&!(u&1))) {
            set<int>::iterator it=st1.lower_bound(l);
            while(it!=st1.end()&&(*it)<=r) {
                q.push(make_pair(*it,st+1));
                set<int>::iterator tmp=it++;
```

```
                st1.erase(tmp);
            }
        } else {
            set<int>::iterator it=st2.lower_bound(l);
            while(it!=st2.end()&&(*it)<=r) {
                q.push(make_pair(*it,st+1));
                set<int>::iterator tmp=it++;
                st2.erase(tmp);
            }
        }
    }
    for(int i=1;i<=n;i++) {
        write(ans[i]);putchar(' ');
    }
    putchar('\n');
    return 0;
}
```

T2 (Silhouette)

转换原问题，即给出一个矩阵每行的最大值，每列的最大值，求可能的原矩阵有多少种。先将a、b从大到小排序，这样被改变的只是矩阵中每个点的位置，不会影响最终答案。这样矩阵中能取得的最大值相等的点就会构成一个矩形或L形（第一个为矩形，之后为L形）。枚举每个在a、b中出现过的数，找出对应的矩形或L形，分别计算答案。

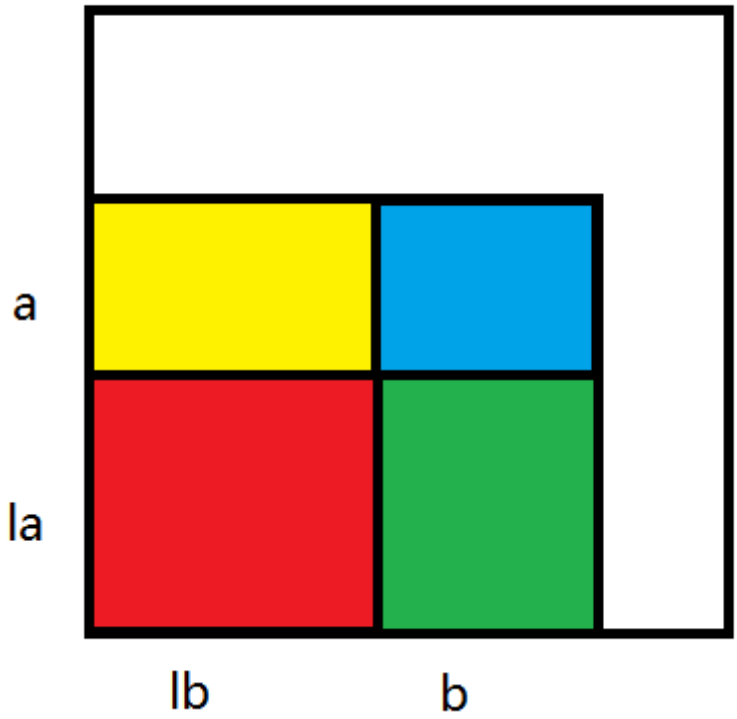
对于每个矩形或L形的答案，考虑容斥，分别计算出在每列都满足条件时，至少有i行不满足条件的情况数，令为$f(i)$，那么这个矩形或L形的答案即为 $\sum_{i=1}^{a}(-1)^i f(i)$

对于一个矩形，设其高、宽分别为a、b，其中能取得的最大值为w，则有

$$f(i)=C_a^i\times w^{ib}((w+1)^{a-i}-w^{a-i})^b$$

，其中，C_a^i表示从a行中选出i行的方案数，w^{ib}表示对于i个一定不能满足条件的行，一共有ib个点，其中的每个元素的取值有w中可能（0到$w-1$）。对于剩下的$a-i$行中的每一列，对于行，每个位置没有限制，可以选0到w共$w+1$个数，但是与此同时，还要保证这一列满足条件，就需要减去w^{a-i}，表示恰好一列中的每个位置都不满足条件的情况，括号外的b次幂表示一共有b个这样的列。

对于一个如图的L形



红色区域已经被处理过，故认为红色区域每一行、每一列都满足条件。那么绿色区域的每一行都满足条件，黄色区域的每一列都满足条件，而蓝色区域则没有限制条件。对于蓝色和绿色的区域，由于绿色区域中每行都满足条件，故至多有a行有可能不满足条件，从中选出i行，共C_a^i中情况。对于其中的b列，每一列有$a+la-i$个元素，且每一列都要满足条件，同上文矩形中的求法，共有$(w^i\times(w+1)^{a+la-i}-w^{a+la-i})^b$中情况。最后考虑黄色区域，其中有$i$行是不满足条件的，这$i$行中每个元素共有$w$种情况（0到$w-1$），另外$a-i$行中，每个元素共有$w+1$种情况，即黄色区域一共有$(w^i\times(w+1)^{a-i})^{lb}$种情况。将两者相乘即为

$$f(i)=C_a^i\times(w^i\times(w+1)^{a+la-i}-w^{a+la-i})^b\times(w^i\times(w+1)^{a-i})^{lb}$$

显然对于任何一个矩形，L形的递推式也成立。

枚举每个在a、b中出现过的数，分别按递推式计算即可，时间复杂度$O(n\log n)$。

```
#include<bits/stdc++.h>
using namespace std;
template<typename T>
void read(T &res) {
    res=0;
    char ch=getchar();
    bool f=false;
    while(ch<'0'||ch>'9') {
        if(ch=='-') f=true;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9') {
        res=(res<<1)+(res<<3)+(ch^48);
        ch=getchar();
    }
    res=f?-res:res;
}
template<typename T,typename ...Args>
void read(T &res,Args &...args) {
```

```
    read(res);
    read(args...);
}
const long long mod=1e9+7;
long long fpow(long long a,long long b) {
    long long ans=1;
    while(b) {
        if(b&1) {
            ans*=a;
            ans%=mod;
        }
        b>>=1;
        a*=a;
        a%=mod;
    }
    return ans;
}
long long exgcd(long long a,long long b,long long &x,long long &y) {
    if(b==0) {
        x=1,y=0;
        return a;
    }
    int g=exgcd(b,a%b,x,y);
    long long tmp=x;
    x=y;
    y=tmp-y*(a/b);
    return g;
}
long long calcInv(long long a,long long p) {
    long long x=0,y=0;
    exgcd(a,p,x,y);
    return ((x+p)%p+p)%p;
}
long long n,a[100001],b[100001],fac[100001],inv[100001],num[200001];
long long initFac(int n) {
    fac[0]=1;
    for(int i=1;i<=n;i++) {
        fac[i]=1ll*i*fac[i-1];
        fac[i]%=mod;
    }
    inv[n]=calcInv(fac[n],mod);
    for(int i=n-1;i>=0;i--) {
        inv[i]=1ll*(i+1)*inv[i+1];
        inv[i]%=mod;
    }
}
long long C(long long m,long long n) {
    return fac[m]*inv[m-n]%mod*inv[n]%mod;
}
long long calc(long long la,long long lb,long long ra,long long rb,long long w)
{
    long long a=ra-la,b=rb-lb,res=0;
    for(int i=0;i<=a;i++) {
        long long now=C(a,i);
            now*=fpow((fpow(w,i)*(fpow(w+1,a+la-i)-fpow(w,a+la-
i))%mod+mod)%mod,b);
            now%=mod;
            now*=fpow(fpow(w,i)*fpow(w+1,a-i)%mod,lb);
```

```
            now%=mod;
        if(i&1) res-=now;
        else res+=now;
        res=(res%mod+mod)%mod;
    }
    return res;
}
int main() {
    freopen("silhouette.in","r",stdin);
    freopen("silhouette.out","w",stdout);
    read(n);
    for(int i=1;i<=n;i++) {
        read(a[i]);
        num[i]=a[i];
    }
    for(int i=1;i<=n;i++) {
        read(b[i]);
        num[i+n]=b[i];
    }
    initFac(n);
    sort(num+1,num+n+n+1,[&](long long a,long long b) {
        return a>b;
    });
    sort(a+1,a+n+1,[&](long long a,long long b) {
        return a>b;
    });
    sort(b+1,b+n+1,[&](long long a,long long b) {
        return a>b;
    });
    int lst=0,la=0,lb=0,ra=0,rb=0;
    long long ans=1;
    for(int i=1;i<=n+n;i++) {
        while(num[i]==lst) ++i;
        if(i>n+n) break;
        lst=num[i];
        ra=la;rb=lb;
        while(a[ra+1]==num[i]) ++ra;
        while(b[rb+1]==num[i]) ++rb;
        ans*=calc(la,lb,ra,rb,num[i]);
        ans%=mod;
        la=ra;lb=rb;
    }
    printf("%lld\n",ans);
    return 0;
}
```

T3 (Seat)

std:

```
#include<iostream>
#define For(i,j,k) for (int i=j;i<=k;i++)
using namespace std;
const int N=1030;
int Mod,n,dp[N][N],f[N][N],g[N][N],inv[N],cnt[N],odd[N],pos[N];
bool vis[N];
```

```
int Pow(int x,int e) {
    int ret=1;
    while(e){
        if(e&1){
            ret=ret*x%Mod;
        }
        x=x*x%Mod;
        e>>=1;
    }
    return ret;
}

void add(int &x,int y) {
    x+=y;
    x=(x>=Mod?x-Mod:x);
}

int main() {
    freopen("seat.in","r",stdin);
    freopen("seat.out","w",stdout);
    scanf("%d%d",&n,&Mod);
    For(i,1,n){
        inv[i]=Pow(i,Mod-2);
    }
    vis[0]=vis[n+1]=true;
    For(i,1,n){
        int pl=0,pr=0;
        For(j,0,n){
            int r=j+1;
            while(!vis[r]){
                ++r;
            }
            if(r-j>pr-pl){
                pl=j,pr=r;
            }
            j=r-1;
        }
        int mx=(pr-pl)>>1;
        cnt[mx]++;
        pos[i]=pl+mx;
        vis[pl+mx]=true;
        if((pr-pl)&1){
            odd[mx]++;
        }
    }
    int sum=n;
    For(i,1,n){
        if(cnt[i]){
            int l=sum-cnt[i]+1,r=sum;
            if(i==1){
                For(j,l,r){
                    For(k,l,r){
                        dp[j][pos[k]]=inv[cnt[i]];
                    }
                }
            }
            else{
                For(j,0,cnt[i]){
```

```
                    For(k,0,odd[i]){
                        f[j][k] = 0;
                    }
                }
                f[0][odd[i]]=1;
                int p=l+odd[i]-1;
                For(j,1,cnt[i]){
                    int oddw=0,evenw=0;
                    For(k,0,odd[i]){
                        if(f[j-1][k]){
                            int frac=(cnt[i]-j+1)+k,w=0;
                            if(k){
                                w=f[j-1][k]*k*2%Mod*inv[frac]%Mod;
                                oddw=(oddw+w*inv[odd[i]*2])%Mod;
                                add(f[j][k-1],w);
                            }
                            if(cnt[i]-odd[i]){
                                w=f[j-1][k]*(frac-2*k)%Mod*inv[frac]%Mod;
                                evenw=(evenw+w*inv[cnt[i]-odd[i]])%Mod;
                                add(f[j][k],w);
                            }
                        }
                    }
                    For(u,l,p){
                        add(dp[l+j-1][pos[u]],oddw),add(dp[l+j-1]
[pos[u]+1],oddw);
                    }
                    For(u,p+1,r){
                        add(dp[l+j-1][pos[u]],evenw);
                    }
                }
                For(j,l,p){
                    int L=pos[j]-i+1,R=pos[j]+i;
                    For(v,L,R){
                        if(v!=pos[j]){
                            For(u,r+1,n){
                                int s=(v<pos[j]?v+i+1:v-i),w=dp[u]
[v]*inv[2]%Mod;
                                add(g[u][v],w),add(g[u][s],w);
                            }
                        }
                    }
                    For(v,L,R){
                        For(u,r+1,n){
                            dp[u][v]=g[u][v],g[u][v]=0;
                        }
                    }
                }
            }
            sum=l-1;
        }
    }
    For(i,1,n){
        For(j,1,n){
            printf("%d%c",dp[i][j],' ');
        }
        putchar('\n');
    }
```

```
    return 0;
}
```

T4 (Ancient SDOI2010 古代猪文)

题意：给出 n、g，求 $g^{\sum_{k|n} C_n^d} \bmod 999911659$

根据欧拉定理可得，$g^{\sum_{k|n} C_n^d} \bmod 999911659 = g^{\sum_{k|n} C_n^d \bmod 999911658}$

显然，999911658不是质数，不能使用Lucas定理。将它分解质因数，得到：$999911658 = 2 \times 3 \times 4679 \times 35617$，每个质因子的幂次恰好为1，则可以分别求出组合数对2、3、4679、35617取模的结果，记为 a_1、 a_2、 a_3、 a_4，令$x = \sum_{k|n} C_n^d \bmod 999911658$，则可以列出同余方程组：$x \equiv a_1 (\bmod 2)$，$x \equiv a_2 (\bmod 3)$，$x \equiv a_3 (\bmod 4679)$，$x \equiv a_4 (\bmod 35617)$，可以用中国剩余定理求得$x$的最小整数解。

Lucas定理：$C_m^n \equiv C_{m \bmod p}^{n \bmod p} \times C_{n/p}^{m/p} (\bmod p)$

可以发现，通过分别预处理出小于a_1、 a_2、 a_3、 a_4的所有阶乘模a_1、 a_2、 a_3、 a_4的值，并预处理出它们在模a_1、 a_2、 a_3、 a_4意义下的逆元，就可以在$O(1)$的时间内求出$C_{m \bmod p}^{n \bmod p}$，一次Lucas的复杂度降低到$O(\log_p n)$。

```
#include<bits/stdc++.h>
using namespace std;
template<typename T>
void read(T &res) {
    res=0;
    char ch=getchar();
    bool f=false;
    while(ch<'0'||ch>'9') {
        if(ch=='-') f=true;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9') {
        res=(res<<1)+(res<<3)+(ch^48);
        ch=getchar();
    }
    res=f?-res:res;
}
template<typename T,typename ...Args>
void read(T &res,Args &...args) {
    read(res);
    read(args...);
}
const long long mod=999911659ll;
long long n,g;
const long long md[4]={2,3,4679,35617};
long long fac[4][40001];
long long a[4];
inline long long fpow(long long a,long long b) {
    long long ans=1;
    while(b) {
        if(b&1) {
            ans*=a;
            ans%=mod;
        }
```

```
        b>>=1;
        a*=a;
        a%=mod;
    }
    return ans;
}
long long exgcd(long long a,long long b,long long &x,long long &y) {
    if(b==0) {
        x=1,y=0;
        return a;
    }
    int g=exgcd(b,a%b,x,y);
    long long tmp=x;
    x=y;
    y=tmp-y*(a/b);
    return g;
}
long long inv(long long a,long long p) {
    long long x=0,y=0;
    exgcd(a,p,x,y);
    return ((x+p)%p+p)%p;
}
long long C(long long n,long long m,int id) {
    if(m>n) return 0;
    long long a=1,b=1,p=md[id];
    a=fac[id][n]*inv(fac[id][n-m],p)%p;
    b=fac[id][m];
    return a*inv(b,p)%p;
}
long long Lucas(long long n,long long m,int id) {
    if(!m) return 1;
    long long p=md[id];
    return C(n%p,m%p,id)*Lucas(n/p,m/p,id)%p;
}
long long intChina() {
    long long ans=0;
    for(int i=0;i<4;i++) {
        long long x=0,y=0;
        exgcd((mod-1)/md[i],md[i],x,y);
        ans=(ans+x*((mod-1)/md[i])%(mod-1)*a[i]%(mod-1)+mod-1)%(mod-1);
    }
    return (ans+mod-1)%(mod-1);
}
int main() {
    freopen("ancient.in","r",stdin);
    freopen("ancient.out","w",stdout);
    read(n,g);
    long long ans=0;
    if(!g||g==mod) {
        puts("0");
        return 0;
    }
    for(int i=0;i<4;i++) {
        fac[i][0]=1;
        for(int j=1;j<=md[i];j++) {
            fac[i][j]=1ll*j*fac[i][j-1];
            fac[i][j]%=md[i];
        }
```

```
    }
    for(long long i=1;i*i<=n;i++) {
        if(n%i==0) {
            for(int j=0;j<4;j++) {
                a[j]+=Lucas(n,i,j);
                a[j]%=md[j];
            }
            if(n/i!=i) {
                for(int j=0;j<4;j++) {
                    a[j]+=Lucas(n,n/i,j);
                    a[j]%=md[j];
                }
            }
        }
    }
    printf("%lld\n",fpow(g,intChina()));
    return 0;
}
```