提高组day2试题 第四组题解

种田

题解

数据范围1:宇宙级难题

数据范围2:我们可以简单知道横纵分别最多种$a=\frac{N}{L}$,$b=\frac{M}{L}$颗槐树,称为解(a,b)。

当$a=b$的时候有所有$(k,k),k\le a$的解都可以满足要求，答案为a。

当$a\ne b$的时候，通过打表观察*，若有$(x-1,y-1)$是解，则$(kx-1,ky-1)$要么是解，要么根本无法放下;因此通过求解最大解(a,b)找到gcd(a+1,b+1)即是答案。

综合两种条件，我们可以在复杂度$O(\ln(\max\{N/L,M/L\}))$的复杂度内解决

数据范围3:如果我们已知a,b，我们可以在$O(1)$的复杂度内验证这是不是个合法的解，在此数据范围下有$max\{a,b\}<1000$那么我们可以分别枚举a，b并验证，这样复杂度就是$O(\frac{NM}{L^2})$的，可以通过

数据范围4:

通过数据范围2,我们只要求得a,b的最大值，我们就可以在O(1)的时间内找到答案

设雕像的间隔为x，那么有$N=aL+x(a+1),M=bL+x(b+1)$

既$a=\frac{N+L}{L+x}-1,b=\frac{M+L}{L+y}-1$

a,b均为整数则有$L+x$为$\gcd(N+L,M+L)$的分数约数

令$G=\gcd(N+L,M+L),k=\frac{G}{L+x}$,k为整数

则有$x=\frac{G}{k}-L>0$，可知此时$k=\lfloor\frac{G}{L}\rfloor$

可解得a,b相关的表达式，套用数据范围2中的算法即可得到答案

---

*:对于这里就是有$x=\frac{N-(a-1)L}{a}=\frac{M-(b-1)L}{b}$则有$x'=\frac{N-(ka-1)L}{ka}=\frac{M-(kb-1)L}{kb}$，这可以简单证明

标准代码

c++ 11

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

typedef long long s64;

void exgcd(s64 a,s64 &x,s64 b,s64 &y,s64 &d)
{
    if(b==0)
```

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

void work(s64 &l,s64 &r,s64 x,s64 d,s64 n)
{
    l=(1-x)/d;
    while(l*d<1-x)++l;
    while((l-1)*d>=1-x)--l;
    r=(n-x)/d;
    while(r*d>n-x)--r;
    while((r+1)*d<=n-x)++r;
}

int main()
{
    s64 n,m,l;
    cin>>n>>m>>l;
    s64 a=n+l,b=m+l,c=m-n;//ay-bx=c
    s64 y0,x0,d;
    exgcd(a,y0,b,x0,d);//ay0+bx0=d
    if(c%d){puts("0");exit(0);}
    s64 y1=y0*s64(c/d),x1=-x0*s64(c/d);//a(y1+k b/d)-b(x1+k a/d)=c
    s64 l1,r1,l2,r2;
    work(l1,r1,y1,b/d,m/l);
    work(l2,r2,x1,a/d,n/l);
    cout<<max(0LL,min(r1,r2)-max(l1,l2)+1);
}
```

野炊

题解

由于$S_{i,k},T_i<8$，我们可以把其二进制压缩为一个数S_i或T_i来表示，问题转化为给出N个数M个询问，每次找一个区间[L,R]，从里面选出K个数使其按位或的值为T。

数据范围1:

对于$R-L+1>K$的情况太难了不会。

对于$R-L+1=K$的情况只需要求出这个区间中所有数的按位或值，判断是否与T相同即可。

数据范围2:

对于每次询问，我们开一个K层循环枚举选取哪K个数就好，这个复杂度是$O(\Sigma C_{R-L+1}^{K})$的，小于数据范围里那个式子。

数据范围3:

我们可以知道对于S_i和T，他们都满足小于256这个性质，因此我们可以对这N个数开一个大小为$O(256N)$的数组做一个种类前缀和，这样对某个区间我们就可以通过$O(256)$的时间知道其中每种数总共出现了多少次。

对于K=2，就是选取两个数A,B满足$A|B=T$，这样我们只需要$O(256^2)$的时间枚举两个数的种类，并通过其分别个数简单计算出其贡献。总复杂度为$O(256^2M)$。

数据范围4:

对于某次询问T，我们需要考虑的数只有T的子集而已(认为A是B的子集当且仅当$A|B=B$),对于这些是T子集的数，我们任意选取K个数，所有方案分别的或和一定是K的子集，我们用这个值减去那些是真子集的方案，就是我们需要的答案。也就是$Ans_T=C_{cnt(T)}^K-\Sigma_{S_i\subseteq T}Ans_{S_i}$,其中$cnt(X)$表示为这个区间里有多少个数是S的真子集，可以通过类似种类前缀和的方式求得。

观察式子，我们转化为求取Ans_{S_i}，我们最多求取256个Ans,每次的复杂度为$O(256)$故单次查询的复杂度为$O(256^2)$

数据范围5:

与上方做法相同，但是通过归纳计算(或者简要观察)我们发现，若S_i与T有奇数偶位数相同，其最后的贡献的符号是+1,否则贡献符号是-1，这样式子变成了$Ans_T=\Sigma_{Si\subseteq T}sgn(bitcount(T-S_i))C_{cnt(S_i)}^K$，单次查询复杂度变为$O(256)$

需要注意的是，本题模数并不常见，常常有人把它和其它数搞混。

标准代码

c++

```
#include <stdio.h>
#include <vector>
typedef long long TYPE;
const TYPE mod=99824353;
const int MAXN=1E5+10;
const int MAXC=256;
const int INF=~0U>>1;
int input() {
    int x=0,c=0;
    do c=getchar(); while(c<'0'||'9'<c);
    do x=x*10+c-'0',c=getchar(); while('0'<=c&&c<='9');
//  printf(">>>%d\n",x);
    return x;
}
int sinput() {
    int n=input();
    int x=0;
    for(int i=0;i<n;++i) {
        x|=(1<<input());
    }//printf(">>%d\n",x);
    return x;
}
bool belong(int x,int y) {
    return (x|y)==x;
}
```

```
TYPE powi(TYPE a,TYPE b) {
    TYPE ans=1;
    while(b) {
        if(b&1)ans=ans*a%mod;
        a=a*a%mod;b>>=1;
    }return ans;
}
TYPE syb(int z) {
    //printf(">>%d\n",z);
    return z%2==0 ? 1LL:-1LL;
}
int bits[MAXC],cnts[MAXC];
std::vector<int>bl[MAXC];
int pre[MAXN][MAXC];
TYPE js[MAXN],ij[MAXN];
void setup(const int N=1E5) {
    bits[0]=0;
    for(int i=1;i<256;++i) {
        bits[i]=bits[i>>1]+i%2;
    }
    for(int i=0;i<256;++i) {
        for(int j=0;j<256;++j) {
            if(belong(i,j)) {
                bl[i].push_back(j);
                ++cnts[i];
            }
        }
    }
    js[0]=1;
    for(int i=1;i<=N;++i) {
        js[i]=js[i-1]*i%mod;
    }
    ij[N]=powi(js[N],mod-2);
    for(int i=N-1;i>=0;--i) {
        ij[i]=ij[i+1]*(i+1)%mod;
    }
}
TYPE C(TYPE N,TYPE M) {
    return N<M ? 0 :js[N]*ij[N-M]%mod*ij[M]%mod;
}
TYPE count(int l,int r,int k,int x) {
    TYPE ans=0;
    int *p=pre[r];
    int *q=pre[l-1];
    for(int i=0;i<cnts[x];++i) {
        int y=bl[x][i];
        ans+=syb(bits[x]-bits[y])*C(p[y]-q[y],k);
        ans+=mod;ans%=mod;
    //  printf(">>%d\n",ans);
    }return ans;
}
int main() {
    setup();
    /*
    for(int i=1;i<=10;++i) {
        for(int j=1;j<=i;++j) {
            printf("%lld ",C(i,j));
        }printf("\n");
```

```
    }*/
    int n=input();
    int m=input();
    for(int i=1;i<=n;++i) {
        int x=sinput();
        for(int k=0;k<256;++k) {
            pre[i][k]=pre[i-1][k];
            if(belong(k,x))pre[i][k]++;
        }
    //  printf("%d\n",pre[i][x]);
    }
    for(int i=1;i<=m;++i) {
        int l=input();
        int r=input();
        int k=input();
        int x=sinput();
        printf("%lld\n",count(l,r,k,x));
    }
    return 0;
}
```

算账

题解

通过一些简单的推断，我们可以知道Qsort的最大递归深度为N，达成要求的充分必要条件是每次randint选出来的数恰好是这个区间的最大或者最小值。以下用a代指A_List

---

数据范围1:可以求出所有1~N的全排列，然后向其中代入，检查递归深度是否达到了N。可以在检查中添加一些剪枝算法来加快检查。最终复杂度上限为$O(N!N^2)$

数据范围2:输入一共有192种情况，我们将这些情况在本地套用**数据范围2**的方法跑出，并将其编码在源程序里提交即可，编码空间复杂度为$O(N^2AB)$，检索时间复杂度为$O(1)$。

数据范围3:通过阅读程序我们可以知道，对于每次randint选中的数，Qsort会将小于它的值放在它的左边，大于它的值放在它的右边。这样，并且那些数的相对顺序会发生改变。若要深度最深，每次选中的数必定是当前区间最大或者最小值。

因此我们可以维护a的原始下标序列p，有$p[a[i]]=i$，按照递归顺序依次确定$a[i]$的每一位。当我们选出index时就是选出$p[x]=index$，此时我们可以令x为当前剩余区间的最大值或者最小值即可，然而题目要求求解字典序最大的，那么，我们可以知道x为当前区间的最大值的充分必要条件是对所有$p[a[i]]<p[x]$的$a[i]$其值都已经被确定(不然一定可以留下最大值填入当前区间中$p[a[i]]$最小的$a[i]$使所求序列变大)。

这样对区间$[L,R]$,需要$O(1)$的复杂度选择需要填入的值，$O(R-L+1)$的复杂度进行下标维护。每次$R-L+1$缩小1，最终复杂度为$O(N^2)$。

数据范围4:经过实验我们发现当选择的值是当前区间的最大或者最小值的时候，整个数组只有$a[L],a[R],a[index]$的值发生了变化，下标维护的复杂度变为$O(1)$总复杂度转化为$O(N)$

你应当注意到本题的模数与一些题目有些相似，请多加注意。

标准代码

c++

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

#define rep(i,l,r) for(int i=l;i<=r;++i)
#define per(i,r,l) for(int i=r;i>=l;--i)
const int N=1e5+5;
int n,d,__A__,__B__,a[N];

long long X=1;
int randint(int L,int R) {
    const long long A=__A__,B=__B__;
    X=(X*X+A*X+B)%99824353LL;
    return X%(R-L+1)+L;
}

void Qsort(int A[],int L,int R) {
    if(L>=R)return ;
    int l=L;
    int r=R;
    int index=randint(L,R);
    int key=A[index];
    std::swap(A[l],A[index]);
    while(l<r) {
        while(l<r&&A[r]>=key)--r;A[l]=A[r];
        while(l<r&&A[l]<=key)++l;A[r]=A[l];
    }A[l]=key;
    if(!(l==L||l==R))
        assert(0);
    Qsort(A,L,l-1);
    Qsort(A,l+1,R);
}

int main()
{
    cin>>n>>__A__>>__B__;
    for(d=1;d<n;d<<=1);d-=1;
    rep(i,1,n)a[i]=i;
    static int ans[N];
    int l=1,r=n,mn=1,ul=1,ur=n;
    rep(tmp,1,n)
    {
        int p=randint(l,r);
        if(a[p]==mn)
        {
            ans[a[p]]=ur--;
            swap(a[p],a[l]);
            swap(a[l],a[r]);
            --r;
        }
        else
        {
```

```
            ans[a[p]]=ul++;
            swap(a[p],a[l]);
            ++l;
        }
        while(ans[mn])++mn;
    }
    rep(i,1,n)printf("%d%c",ans[i]," \n"[i==n]);
}
```