2015年西安电子科技大学ACM校赛网络赛解题报告

xry111

May 10, 2015

1 A - IP查询

首先请注意，直接模拟肯定会MLE或者RE的，这也是这题AC率如此之低的原因。（我觉得好多人都被那句“su出的题目难度较低”骗了。）

1.1 二分查找

这个题显然可以用二分查找解决。把所有区间按起始点排个序，然后对于每次查询，找到最后一个起始点不在V之后的区间，判断这个区间是否覆盖了V，如果覆盖了则输出这个区间的ID，没覆盖就输出-1。

像我一样，写二分老是死循环或者RE的同学可以直接用STL的二分查找。参考资料：

http://www.cplusplus.com/reference/algorithm/lower_bound/

http://www.cplusplus.com/reference/algorithm/upper_bound/

http://www.cplusplus.com/reference/algorithm/binary_search/

时间复杂度：O((n+m)logn)

参考程序（by xry111, 446ms,1688KB）：

```cpp
#include<bits/stdc++.h>

using namespace std;

struct segment{
    int x,y,id;
    bool operator<(const segment& rhs) const {return x<rhs.x;}
};

segment orz[100000];

int main(){
    int T; scanf("%d",&T);
    while(T--){
        int n; scanf("%d",&n);
        for(int i = 0; i<n; i++)
            scanf("%d%d%d",&orz[i].x, &orz[i].y, &orz[i].id);
        sort(orz, orz+n);
        int m;scanf("%d",&m);
```

```
        while(m--){
            int v; scanf("%d",&v);
            if(n==0) {puts("-1"); continue;}
            segment *p = upper_bound(orz,orz+n,segment{v,0,0})-1;
            if(p->x<=v&&p->y>=v) printf("%d\n",p->id);
            else puts("-1");
        }
    }
    return 0;
}
```

1.2 线段树

这是一个区间信息维护题，自然想到用线段树解决。由于x、y的范围太大，不能用堆式存储，只能用数组模拟指针法存储。

时间复杂度：O((n+m)log(max{V}))

参考程序（by xry111, 723ms, 76688KB）：

```
#include<bits/stdc++.h>

using namespace std;

struct sgt_node{
    int lch,rch,v;
};

sgt_node sgt[100000*64];
int cnt;

int l,r,v;
void modify(int L, int R, int u){
    if(l<=L&&R<=r) sgt[u].v = v;
    else{
        int M = (L+R)>>1;
        if(!sgt[u].lch){
            sgt[cnt] = sgt_node{0,0,sgt[u].v};
            sgt[u].lch = cnt++;
        }
        if(!sgt[u].rch){
            sgt[cnt] = sgt_node{0,0,sgt[u].v};
            sgt[u].rch = cnt++;
        }
        if(l<=M) modify(L,M,sgt[u].lch);
        if(M<r) modify(M+1,R,sgt[u].rch);
    }
}

int x;
int query(int L, int R, int u){
```

```
    if(!sgt[u].lch&&!sgt[u].rch) return sgt[u].v;
    int M = (L+R)>>1;
    return x<=M?query(L,M,sgt[u].lch):query(M+1,R,sgt[u].rch);
}

const int LEN = 1e8+0.1;

int main(){
    int T; scanf("%d",&T);
    while(T--){
        int n; scanf("%d",&n);
        cnt = 1;
        sgt[0] = sgt_node{0,0,-1};
        //printf("n=%d\n",n);
        while(n--){
            scanf("%d%d%d",&l,&r,&v);
            modify(0,LEN,0);
        //  printf("cnt=%d\n",cnt);
        }
        int m; scanf("%d",&m);
        while(m--){
            scanf("%d",&x);
            printf("%d\n",query(0,LEN,0));
        }
    }
    return 0;
}
```

2 B - 简单逆序对

直接模拟的时间复杂度是$O(n^2)$，会超时。其实，对于每个数，我们把它之前所有大于它的数的个数求出来，再求和就行了。注意到所有数字都在0到9以内，所以我们可以从左往右扫描，维护0，1，...，9的个数，然后用暴力法就能算出每个数之前大于它的数的个数。最后记得取模。

时间复杂度：$O(10 \times n)$。

参考程序（by xry111, 393ms, 1688KB）：

```
#include<bits/stdc++.h>

using namespace std;

int cnt[10];
const int M = 1e9+7.5;
int main(){
    int T; scanf("%d",&T);
    while(T--){
        int n; scanf("%d",&n);
        memset(cnt,0,sizeof(cnt));
```

```
        long long ans = 0;
        for(int i = 0; i<n; i++){
            int t; scanf("%d",&t);
            for(int i = t+1; i<10; i++) ans += cnt[i];
            cnt[t]++;
        }
        printf("%lld\n",ans%M);
    }
    return 0;
}
```

3 C - 梦想庄园

因为出题人一般不来实验室，缺乏交流，这题的题目描述出了很多问题。由于数据没有环，显然满足无后效性，可以DP。用f[u]表示制作物品u的代价，则状态转移方程为

$$f[u] = min\{w + \sum_{v \in dep_u} f[v], c_u\}$$

dep_u表示u依赖的物品的集合。由于物品只有10000个，可以直接用DFS代替拓扑排序。最后的答案就是

$$\sum_{u \in task} f[u]$$

时间复杂度：$O(n + m + \sum_i C_i)$
参考程序：（by xry111, 41ms, 2140KB）

```
#include<bits/stdc++.h>

using namespace std;

vector<int> ch[10000];
int v[10000];
int w;

int dp[10000];
int dfs(int u){
    if(dp[u]!=-1) return dp[u];
    int ans = w;
    if(!ch[u].size()) return dp[u] = v[u];
    for(vector<int>::iterator it = ch[u].begin(); it!=ch[u].end(); it++){
        ans += dfs(*it);
    }
    return dp[u] = min(ans, v[u]);
}

void solve(){
    int n,m;scanf("%d%d%d",&n,&m,&w);
```

```
    memset(dp,-1,sizeof(dp));
    for(int i = 0; i<n; i++){
        ch[i].clear();
        int c; scanf("%d%d",v+i,&c);
        while(c--){
            int t; scanf("%d",&t);
            ch[i].push_back(t);
        }
    }
    int ans = 0;
    for(int i = 0; i<m; i++){
        int k; scanf("%d",&k);
        ans += dfs(k);
    }
    printf("%d\n",ans);
}

int main(){
    int t; scanf("%d",&t);
    while(t--) solve();
    return 0;
}
```

4 D - 修理OJ

这是临时换上来的一道送分题，只要暴力模拟，并且记着边算边取模就行了。为什么可以边算边取模呢？因为如果$a \equiv A \ (mod\ c)$且$b \equiv B \ (mod\ c)$，则

$$a \times b \equiv A \times B \ (mod\ c)$$

通过把a、b、A、B全写成nc+m（$0 \le m < c$）的形式，很容易证明这一点。
典型错误答案是pow(a,b)%c，因为pow是浮点函数，会丢失有效数字。
时间复杂度：$O(b)$
参考程序（by xry111，10ms，1688KB）：

```
#include<bits/stdc++.h>

int main(){
    int a,b,c;
    while(scanf("%d%d%d",&a,&b,&c)==3){
        int ans = 1;
        while(b--){
            ans = (ans*a)%c;
        }
        printf("%d\n",ans);
    }
    return 0;
}
```

5 E - Feibonaqi数列

5.1 矩阵快速幂

写出f满足的线性递推式的矩阵形式：

$$\begin{bmatrix} a_n \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ a_{n-2} \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

接下来的问题就是求矩阵的n-1次幂。这可以用快速幂实现。
时间复杂度：$O(logn)$
参考程序（by xry111, 71ms, 1084KB）：

```
#include<bits/stdc++.h>

using namespace std;

const int M = 1e9+7.1;

struct mat_t{
    int a[2][2];
    mat_t operator *(const mat_t& rhs) const{
        mat_t t;
        for(int i = 0; i<2; i++)
            for(int j = 0; j<2; j++){
                long long acc = 0LL;
                for(int k = 0; k<2; k++){
                    acc = acc + (long long)a[i][k]*rhs.a[k][j];
                    acc = acc % M;
                }
                t.a[i][j] = acc;
            }
        return t;
    }
};

mat_t mat[31];

int main(){
    mat[0].a[0][0] = 2;
    mat[0].a[0][1] = mat[0].a[1][0] = 1;
    mat[0].a[1][1] = 0;

    for(int i = 1; i<31; i++) mat[i] = mat[i-1] * mat[i-1];

    int n; while(scanf("%d",&n)==1){
        n-=1;
        mat_t A;
        A.a[0][0] = A.a[1][1] = 1;
        A.a[0][1] = A.a[1][0] = 0;
```

```
        for(int i = 0; i<31; i++){
            if(n&(1<<i)){
                A = A*mat[i];
            }
        }
        printf("%d\n", A.a[0][0]);
    }

    return 0;
}
```

5.2 通项公式

下面试着推导f的通项公式。首先写出数列f的生成函数

$$g(z) = \sum_{k=0}^{\infty} f(k)x^k$$

然后对生成函数应用通项公式:

$$\begin{aligned} g(z) &= z + \sum_{k=2}^{\infty} f(k)z^k \\ &= z + \sum_{k=2}^{\infty} 2f(k-1)z^k + \sum_{k=2}^{\infty} f(k-2)z^k \\ &= z + 2g(z)z + g(z)z^2 \end{aligned}$$

解这个关于g(z)的方程，得

$$g(z) = \frac{z}{1-2z-z^2}$$

为了简化这个式子，对分母分解因式。设分母等于$(1-ax)(1-bx) = 1-(a+b)x+abx^2$，比较系数得

$$\begin{cases} a+b=2 \\ ab=-1 \end{cases}$$

解这个方程，得到$a = 1+\sqrt{2}$，$b = 1-\sqrt{2}$。注意到分母的两个因式之间差模部分是$\sqrt{2}z$，可以把这个分式拆成两个分式之差

$$\begin{aligned} g(z) &= \frac{z}{[1-(1+\sqrt{2})z][1-(1-\sqrt{2})z]} \\ &= \frac{1}{2\sqrt{2}}\left[\frac{1}{1-(1+\sqrt{2})z} - \frac{1}{1-(1-\sqrt{2})z}\right] \end{aligned}$$

运用公式

$$\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k$$

得到

$$g(z) = \frac{1}{2\sqrt{2}} \sum_{k=0}^{\infty} [(1+\sqrt{2})^k - (1-\sqrt{2})^k] z^k$$

可见

$$f(n)=\frac{1}{2\sqrt{2}}[(1+\sqrt{2})^n-(1-\sqrt{2})^n]$$

（顺便多说一句，上面的过程在信号与系统中叫做z变换、部分分式展开和z逆变换，所以这个题非常适合信号90+的学霸去做。）
但是直接用这个公式做肯定是不行的，因为无理数$\sqrt{2}$在计算机中只能用浮点数表示，会损失精度。因此，我们需要求出模1000000007意义下的$\sqrt{2}$，即一个数x使得

$$x^2\equiv 2\ (mod\ 1000000007)$$

采用暴力枚举法就能求出x=59713600。然后为了除以2和$\sqrt{2}$，还要求2和59713600的逆元，因为1000000007是素数，可以用费马小定理

$$x^{p-1}\equiv 1\ (mod\ p)$$

显然可以看出

$$x\times x^{p-2}\equiv 1\ (mod\ p)$$

于是x^{p-2}就是x的逆元。这样，本题中所有的运算都转化成了乘法和指数运算，采用快速幂处理指数运算就行了。
时间复杂度：O(logn)
参考程序：（by xry111, 94ms, 1688KB）

```
#include<bits/stdc++.h>

using namespace std;

const int M = 1e9+7.1;
const int sqrt2 = 59713600;

int exp(int a, int e){
    if(e==0) return 1;
    long long t = exp(a,e>>1);
    t*=t; t%=M;
    if(e&1) t = (t*a)%M;
    return t;
}

int inv(int x){
    return exp(x,M-2);
}

int main(){
    int c = ((long long)inv(2)*inv(sqrt2))%M;
    int n; while(scanf("%d",&n)==1){
        int t1 = exp(1+sqrt2, n);
        int t2 = exp(1-sqrt2+M, n);
        long long t = (t1-t2)%M;
        if(t<0) t+=M;
        t*=c; t%=M;
```

```
        printf("%d\n",(int)t);
    }
    return 0;
}
```

6 F - 数字工程

6.1 普通方法

注意到每个数字都只能变得比原来小，发现这题具有无后效性，可以DP。用f[k]表示把正整数k变成1所需的最小能量，可以写出状态转移方程

$$f[k] = min\{f[k-1], min_{p|k}\{f[k/p]\}\} + 1$$

这里$a|b$表示a整除b。编程实现的时候不用真的枚举k，而是从前往后扫描，每次拿到一个数就把它当作k/p，用一个质数把它乘一下就得到k，更新f[k]即可。

时间复杂度：$O(\sum_{p \in prime} \frac{n}{p}) = O(nloglogn)$

参考程序：(by xry111, 70ms, 13408KB)

```
#include<bits/stdc++.h>

using namespace std;

int f[1000001];
int mark[1000001] = {0};
int prime[1000000], cnt_prime = 0;

int main(){
    for(int i = 2; i<=1000000; i++){
        if(!mark[i]){
            prime[cnt_prime++] = i;
            for(int j = i+i; j<=1000000; j+=i) mark[j] = 1;
        }
    }
    memset(f,0x3f,sizeof(f));
    f[1] = 0;
    for(int i = 1; i<1000000; i++){
        for(int j = 0; j<cnt_prime; j++){
            long long t = (long long)i*prime[j];
            if(t>1000000) break;
            f[t] = min(f[t], f[i]+1);
        }
        f[i+1] = min(f[i+1], f[i]+1);
    }
    int x; while(scanf("%d",&x)==1) printf("%d\n",f[x]);
    return 0;
}
```

6.2 文艺方法

很多人都百度出了一段很吊的代码，这代码我看不懂，也不会分析，直接Orz。
时间复杂度：？
参考代码：（by 某个上古大神, 74ms, 8776KB）

```
#include<stdio.h>
#include<string.h>
#define N      1000005
int a[N + 1];
int main()
{
    memset(a,0,sizeof(a));
    int p[N], pn = 0, an, m, i, j, t;

    for(i = 2; i <= N; a[i++] = 1);

    for(i = 2; i <= N; i++)
    if(a[i]) for(p[pn++] = i, j = i + i; j <= N; j += i) a[j] = 0;

    for(an = pn, i = 0; an < N - 1; i++)
    {
        if((t = p[i] + 1) <= N && !a[t]) a[p[an++] = t] = a[p[i]] + 1;

        for(j = 0, m = N / p[i]; j < pn && p[j] <= m; j++)
        if(!a[t = p[i] * p[j]]) a[p[an++] = t] = a[p[i]] + 1;
    }

    for(i = 3; i <= N; i++) a[i] += a[i - 1];

    while(scanf("%d", &i) != EOF){j=i;
     printf("%d\n", a[j] - a[i - 1]);}

    return 0;
}
```

6.3 二逼方法

由于xry111比较傻逼，他枚举质因数i和i的倍数j，试图用f[j]=min(f[j],f[j/i]+1)做状态转移。显然这是错的，因为这时f[j/i]未必已经算出来了。
于是，xry111就脑洞大开，把这一过程重复了10次，过掉了这题。为什么10次是足够的呢？因为1到10^6之间的数至多有10个不同质因数。
时间复杂度：$O(nlog(n)loglogn)$
参考代码：（by xry111, 455ms, 8896KB）

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;
```

```
int f[1000002];
int m[1000002] = {0};

int main(){
    memset(f, 0x3f, sizeof(f));
    f[1] = 0;
    m[1] = 1;
    for(int k=0; k<10; k++){
        for(int i = 1; i<=1000000; i++){
            if(!m[i]){
                f[i] = min(f[i], 1);
                for(int j = i+i; j<=1000000; j+=i){
                    m[j] = 1;
                    f[j] = min(f[j], f[j/i]+1);
                }
            }
            f[i+1] = min(f[i+1], f[i]+1);
        }
    }
    int n; while(scanf("%d",&n)==1) printf("%d\n",f[n]);
    return 0;
}
```

7 G - 数一的逆袭

最低位很好求（是D题的一个特殊情况，即a=2，b=n，oj_tot=10），但最高位不太容易求。考虑把2^n表示成十进制科学计数法

$$2^n = a \times 10^b \ (1 \le a < 10, b \in Z)$$

可以看出

$$a = 10^{nlg2 - \lfloor nlg2 \rfloor}$$

然后a的整数部分$\lfloor a \rfloor$就是答案。编程的时候要注意，由于使用了浮点函数pow，会引入误差（例如pow(2,3)=7.99999999999999911182），需要加一个eps再取整。
有些同学可能会怀疑浮点数的精度是否足够，但把10^6个答案全部输出后会发现并没有特别接近于某个整数的答案，因此不用担心。
时间复杂度：$O(max\{n\})$
参考程序（by lnever, 4ms, 4476KB）：

```
#include <iostream>
#include <stdio.h>
#include<string.h>
#include<cmath>
#include<algorithm>
#include<string>
#include<ctype.h>
using namespace std;
```

```
const double p=log10(2.0);
const double eps=1e-9;
int a[100010];
int main()
{
    a[0]=1;
    for(int i=1;i<=100000;i++)
    {
        a[i]=(a[i-1]*2)%10;
    }
    int t;
    cin>>t;
    while(t--)
    {
        int n;
        cin>>n;
        double m=0;
        m=p*n;
        m-=floor(m);
        printf("%.f %d\n",floor(pow(10.0,m)+eps),a[n]);
    }
    return 0;
}
```

8 H - 三数和

8.1 暴力+剪枝

这个题的数据都是随机的，所以用暴力法加上一些黑优化就能过。具体方法是，先对数组排序，然后DFS选出3个数。如果已经选出的数加上当前的数已经大于0，那么显然后面的都大于0，就不用继续找了。此外，找最后一个数的时候从后往前找，如果得到的结果小于0，则显然前面的都小于0，也不用找了。

时间复杂度：最坏$O(n^3)$，但测试数据里并没有数据使该算法运行得这么慢。

参考程序：（by sublimation, 555ms, 1696KB）

```
#include<bits/stdc++.h>
using namespace std;
int n,a[1001],ans[10],i,tt=0;
int dfs(int x,int y,int z){
    if (x>1 && ans[1]>=0) return 0;
    if (z+a[y]>0) return 0;
    //printf("%d %d %d \n",x,y,z);
    int i;
    if (x==3)
    {
        for (i=n;i>=y;i--)
        {
            if (z+a[i]<0) return 0;
```

```
            if (z+a[i]==0){
                z+=a[i];
                ans[3]=a[i];
                break;
            }
        }
        if (z==0)
        {
            printf("%d %d %d\n",ans[1],ans[2],ans[3]);
            tt=1;
        }
        return 0;
    }
    for (i=y;i<=n-3+x;i++)
    {
        ans[x]=a[i];
        dfs(x+1,i+1,z+a[i]);
    }
    return 0;
}
int main(){
    while (scanf("%d",&n)>0)
    {
        tt=0;
        memset(a,0,sizeof(a));
        for(i=1;i<=n;i++)
            scanf("%d",&a[i]);
        sort(a+1,a+n+1);
        dfs(1,1,0);
        if (tt==0) printf("Ren Chou Jiu Gai Duo Du Shu!\n");
    }
    return 0;
}
```

8.2 二分查找

首先把输入的数排个序，然后枚举前两个数a[i]，a[j]（$i < j$），那么第三个数必须是-a[i]-a[j]，才能使得三数和等于0。那么，只要在j以后的数组中二分查找-a[i]-a[j]就行了。

对自己二分查找没信心的同学请使用STL。参考资料:

http://www.cplusplus.com/reference/algorithm/lower_bound/

http://www.cplusplus.com/reference/algorithm/upper_bound/

http://www.cplusplus.com/reference/algorithm/binary_search/

时间复杂度：$O(n^2 logn)$

参考程序（by xry111, 381ms, 1700KB）：

```
#include<bits/stdc++.h>

using namespace std;
```

```

int a[1000];

const char *GG = "Ren Chou Jiu Gai Duo Du Shu!";

int main(){
    int n;while(scanf("%d",&n)==1){
        for(int i = 0; i<n; i++){
            scanf("%d",a+i);
        }
        sort(a, a+n);
        bool fl = 0;
        for(int i = 0; i<n-2; i++)
            for(int j = i+1; j<n-1; j++)
                if(binary_search(a+j+1,a+n,-a[i]-a[j])){
                    printf("%d %d %d\n",a[i],a[j],-a[i]-a[j]);
                    fl = 1;
                }
        if(!fl) puts(GG);
    }
}
```

8.3 哈希表

考虑进一步优化二分查找法的方法。注意到，只要能在$O(1)$时间内找到-a[i]-a[j]是否在数组内就能在$O(n^2)$时间内解决这题了。因为数组中所有的数绝对值不超过10^7，范围不大，只需要最原始的哈希表——一个数组。由于C/C++不支持负下标数组，你可以用Pascal写或者通过指针运算伪造一个负下标数组（见参考程序）。
时间复杂度：$O(n^2)$
参考程序（by xry111, 36ms, 79820KB）：

```
#include<bits/stdc++.h>

using namespace std;

const char *GG = "Ren Chou Jiu Gai Duo Du Shu!";
int _mark[(const int)(2e7+1.1)]={0};
int *mark = _mark+(const int)(1e7+0.1);
int a[1000];

int main(){
    int n; while(scanf("%d",&n)==1){
        for(int i = 0; i<n; i++){
            scanf("%d",a+i);
            mark[a[i]] = 1;
        }
        sort(a,a+n);
        int fl = 0;
        for(int i = 0; i<n; i++)
```

```
            for(int j = i+1; j<n; j++){
                int num3 = -a[i]-a[j];
                if(a[j]<num3)
                    if(abs(num3)<=(const int)(1e7+0.1)&&mark[num3]){
                        printf("%d %d %d\n",a[i],a[j],num3);
                        fl = 1;
                    }
            }
        if(!fl) puts(GG);
        /* Avoid memset(_mark, 0, sizeof(_mark))        */
        /* or time complexity will become O(n^2+10^7) */
        for(int i = 0; i<n; i++) mark[a[i]] = 0;
    }
    return 0;
}
```

8.4 扫描法

使用哈希表虽然能够在$O(n^2)$时间内解决这道题，但是内存占用太大了，让人十分不爽。实际上，将数组排序后，可以通过进一步运用单调性优化程序。具体方法是，首先暴力枚举第一个数x，然后设指针p指向第二个数，q指向第三个数。将p指向x之后的一个数，q指向最后一个数。由于数组是单调的，当p向后移动时，q只有向前移动才可能找到解。最后，当p、q重合时，第一个数是x的所有组合就都找出来了。

时间复杂度：$O(n^2)$

参考程序：（by lnever, 48ms, 1696KB）：

```
#include <iostream>
#include <stdio.h>
#include<string.h>
#include<algorithm>
#include<string>
#include<ctype.h>
using namespace std;
#define MAXN 10000
int a[1010];
int main()
{
    int n;
    while(scanf("%d",&n)!=EOF)
    {
        for(int i=0; i<n; i++)
        {
            scanf("%d",a+i);
        }
        sort(a,a+n);
        int ok=0;
        for(int i=0; i<n; i++)
        {
            int k=n-1;
```

```
            for(int j=i+1; j<n; j++)
            {
                while(a[i]+a[j]+a[k]>0)
                {
                    k--;
                }
                if(k<=j)
                    break;
                if(a[i]+a[j]+a[k]==0)
                {
                    ok=1;
                    printf("%d %d %d\n",a[i],a[j],a[k]);
                }
            }
        }
        if(ok==0)
        {
            puts("Ren Chou Jiu Gai Duo Du Shu!");
        }
    }
    return 0;
}
```

9 I - 找规律I

9.1 暴力

不难看出规律是

$$A_{nm} = \frac{n+m}{gcd(n,m)}$$

这里$gcd(x,y)$表示数x和y的最大公约数。因为n、m比较小，直接暴力求和就行了。
时间复杂度：每次询问$O(nm)$。
参考程序（by lnever, 112ms, 9656KB）：

```
#include <iostream>
#include <stdio.h>
#include<string.h>
#include<algorithm>
#include<string>
#include<ctype.h>
using namespace std;
#define MAXN 10000
int gcd(int a,int b)
{
    return b==0?a:gcd(b,a%b);
}
long long a[1010][1010];
void ini()
```

```
{
    for(int i=1;i<=1000;i++)
    {
        for(int j=1;j<=1000;j++)
        {
            a[i][j]=gcd(i,j);
        }
    }
}
int main()
{
    ini();
    int n,m;
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        long long ans=0;
        for(int i=1;i<=n;i++)
        {
            for(int j=1;j<=m;j++)
            {
                ans+=(i+j)/a[i][j];
            }
        }
        printf("%lld\n",ans);
    }
    return 0;
}
```

9.2 预处理

这个题没有说数组组数，所以最多是100组，暴力可以过。但是，如果数据组数更多，就需要预处理。设

$$B_{nm} = \sum_{i=1}^{n} \sum_{j=1}^{m} A_{ij}$$

可以用这个式子求出B

$$B_{nm} = B_{n-1,m} + B_{m-1,n} + B_{m-1,n-1} + A_{nm}$$

这样，对于每次查询，直接输出B_{nm}即可。
时间复杂度：预处理$O(nm)$，每次查询$O(1)$。
参考程序（by xry111, 85ms, 13432KB）：

```
#include<bits/stdc++.h>

using namespace std;

#define LLD "%lld"

int a[1001][1001] = {0};
```

```
long long sum[1001][1001] = {0};

int main(){
    for(int i = 1; i<=1000; i++){
        for(int j = 1; j<=1000; j++){
            a[i][j] = (i+j)/__gcd(i,j);
        }
    }

    for(int i = 1; i<=1000; i++){
        for(int j = 1; j<=1000; j++){
            sum[i][j] = sum[i-1][j]+sum[i][j-1]-sum[i-1][j-1]+a[i][j];
        }
    }

    int n,m; while(scanf("%d%d",&n,&m)==2){
        printf(LLD "\n",sum[n][m]);
    }

    return 0;
}
```

10 J - 找规律II

10.1 威尔逊定理

、

注意到10007是质数，令p=10007。容易看出，这题的规律是

$$A_{nm} = C_m^n = \frac{m!}{n!(m-n)!}$$

为了避免对0求逆元，首先特判掉结果为0的情况。设$x!$中质因数p的个数是y，则根据做1019（自然数的秘密）的经验知道

$$y = \sum_{k=1}^{\infty} \lfloor \frac{x}{p^k} \rfloor$$

这样就能求出分子中质因数p的个数a和分母中质因数的个数b。显然，$b \le a$。若$b < a$，则答案被p整除，模p后等于0。否则，分子、分母中的质因数p相互抵消，在计算各个阶乘时可以将其忽略。所以，**下面用x*代表除去x中所有质因子p后的结果。**
剩下的问题就是快速求$(m!)^* \ mod \ p$。这可以用威尔逊定理解决。
定理： $(n-1)! \equiv -1 \ (mod \ n)$当且仅当$n$是质数。
证明：
首先证明必要性。如果n不是质数，则它有一个质因数p。假设$(n-1)! \equiv -1 \ (mod \ n)$，则$(n-1)! \equiv -1 \ (mod \ p)$。但是，p小于n，所以p必然整除$(n-1)!$，矛盾。
然后证明充分性。设p是质数，则数1,2,...,p-1对p都有逆元。逆元为本身的数x满足

$$x^2 \equiv 1 \ (mod \ p)$$

该方程在1到p-1之内的解是1和p-1。除了这两个数以外，其他的数的逆元都不等于本身，且属于$[2, p-2]$。这样，2到p-2内所有数的乘积和1同余，因此

$$(p-1)! \equiv 1 \times (p-1) \equiv -1$$

证明完毕。
有了威尔逊定理，就能设计一种有效的算法求$(m!)^*\ mod\ p$。对于$m!$，设去掉所有p的倍数后剩下的数是y，并且用1到p-1之间对p同余的数代替所有数，就有

$$\begin{aligned} y &= 1 \times 2 \times ... \times (p-1) \times (p+1) \times (p+2) \times (2p-1) \times (2p+1) \times ... \times m \\ &\equiv (1 \times 2 \times ... \times p-1)^{\lfloor \frac{m}{p} \rfloor} \times 1 \times 2 \times ... \times (m\ mod\ p) \\ &\equiv (-1)^{\lfloor \frac{m}{p} \rfloor} (m\ mod\ p)! \end{aligned}$$

因为p只有10007，可以预处理出$1!, 2!, ..., 10006!$，求y的时间就是$O(1)$。下面考虑刚才去掉的p的倍数。对于其中的p，按照刚才的分析可以忽略。除以一个p之后，就是

$$\frac{1}{p}(p \times 2p \times 3p \times ... \times \lfloor \frac{m}{p} \rfloor p) = \lfloor \frac{m}{p} \rfloor !$$

这样就可以继续递归地求$(\lfloor \frac{m}{p} \rfloor !)^*$，然后乘上刚才得到的y就是$(m!)^*$。显然，每次问题的规模都缩小到原来的$1/p$，所以递归次数是$log_p m$，即求$m!$的时间复杂度是$O(logm)$。求出了$(m!)^*$、$(n!)^*$和$((m-n)!)^*$后，用乘法和逆元就能算出$C_m^n\ mod\ p$。
时间复杂度：预处理$O(p)$，每次查询$O(logm)$。
参考程序（by xry111, 0ms, 1732KB）：

```
#include<bits/stdc++.h>

using namespace std;

#define LLD "%lld"

int fact[10007];

typedef long long ll;
void exgcd(ll a, ll b, ll &g, ll &x, ll &y){
     if(!b) {x=1;y=0;g=a;}
     else{
          exgcd(b,a%b,g,y,x);
          y-=x*(a/b);
     }
}

ll inv(ll a, ll n){
     ll d,x,y;
     exgcd(a,n,d,x,y);
     return d==1?(x+n)%n:-1;
}

inline long long cnt_10007(long long x){
     static const long long
```

```
        A = 10007, B = 10007LL*10007, C = A*B, D = B*B;
    return x/A+x/B+x/C+x/D;
}

long long fact_expect_10007(long long x){
    if(!x) return 1LL;
    long long ret = fact[x%10007];
    ret = (ret * fact_expect_10007(x/10007))%10007;
    return ((x/10007)&1)?(ret*10006)%10007:ret;
}

int main(){
    fact[0] = 1;
    for(int i = 1; i<10007; i++) {
        fact[i] = (fact[i-1]*i)%10007;
    }
    long long n,m;
    while(scanf(LLD LLD,&n,&m)==2){
        if(cnt_10007(m)>cnt_10007(n)+cnt_10007(m-n))
            puts("0");
        else{
            long long ans = fact_expect_10007(m);
            ans = ans * inv(fact_expect_10007(n),10007);
            ans = ans % 10007;
            ans = ans * inv(fact_expect_10007(m-n),10007);
            ans = ans % 10007;
            printf(LLD "\n",ans);
        }
    }
    return 0;
}
```

10.2 Lucas定理

实际上，可以证明一个更强大的专用于组合数的定理——Lucas定理。
定理：若p是素数，a, b是正整数，且

$$a = \sum_{j=0}^{k} a_j p^j$$

$$b = \sum_{j=0}^{k} b_j p^j$$

这里$0 \le a_i, b_i \le p-1$是整数，则

$$C_a^b \equiv \prod_{j=0}^{k} C_{a_j}^{b_j}$$

证明：因为p是素数，对于$1 \le j < p$，有$C_p^j = \frac{p}{j}C_{p-1}^{j-1} \equiv 0\ (mod\ p)$。于是

$$\begin{aligned}(1+x)^p &= 1 + C_p^1 x + ... + C_p^{p-1}x^{p-1} + x^p \\ &\equiv 1 + x^p\ (mod\ p)\end{aligned}$$

所以

$$\begin{aligned}(1+x)^a &= (1+x)^{a_0} \times ((1+x)^p)^{a_1} \times ... \times ((1+x)^{p^k})^{a_k} \\ &\equiv (1+x)^{a_0} \times (1+x^p)^{a_1} \times ... \times (1+x^{p^k})^{a_k}\ (mod\ p)\end{aligned}$$

应用二项式定理，对比两边x^b的系数，有

$$C_a^b \equiv C_{a_0}^{b_0} \times C_{a_1}^{b_1} \times ... \times C_{a_k}^{b_k}\ (mod\ p)$$

证明完毕。
有了Lucas定理，就能设计一种有效算法求$C_a^b\ mod\ p$。将a, b都除以p，得到的余数就是a_0, b_0，商就是

$$a' = \sum_{j=0}^{k-1} a_{j+1}p^j$$

$$b' = \sum_{j=0}^{k-1} b_{j+1}p^j$$

运用Lucas定理，知道

$$\begin{aligned}C_a^b &\equiv \prod_{j=0}^{k} C_{a_j}^{b_j} \\ &= C_{a_0}^{b_0}\prod_{j=0}^{k-1} C_{a_{j+1}}^{b_{j+1}} \\ &\equiv C_{a_0}^{b_0} \times C_{a'}^{b'}\end{aligned}$$

这里a_0, b_0较小，$C_{a_0}^{b_0}$可以用预处理出来的阶乘及其逆元相乘得到，$C_{a'}^{b'}$则可以继续递归地求。显然，每次问题的规模都缩小到原来的$1/p$，所以求C_m^n的递归次数是$log_p m$，时间复杂度是$O(logm)$。
时间复杂度：预处理$O(p)$，每次询问$O(logm)$。
参考程序：（by fpcsong, 2ms, 2080KB）

```
#include<bits/stdc++.h>
using namespace std;
int fac[100010];
long long mod_pow(long long a,long long  n,long long  p)
{
    long long ret=1;
    long long A=a;
    while(n)
    {
        if (n & 1)
            ret=(ret*A)%p;
        A=(A*A)%p;
```

```
        n>>=1;
    }
    return ret;
}
void init(long long p)
{
    fac[0] = 1;
    for(int i = 1; i <= p; i++)
        fac[i] = fac[i-1]*i%p;
}
long long Lucas(long long a,long long k,long long p)
{
    long long re = 1;
    while(a && k)
    {
        long long aa = a%p;
        long long bb = k%p;
        if(aa < bb) return 0;
        re = re*fac[aa]*mod_pow(fac[bb]*fac[aa-bb]%p,p-2,p)%p;
        a /= p;
        k /= p;
    }
    return re;
}

int main()
{
    long long  n,m,p;
    init(10007);
    p=10007;
    while(cin>>n>>m)
    {
        cout << Lucas(m,n,p) << endl;
    }
    return 0;
}
```