数论

一、AcWing 1246. 等差数列

【题目描述】

数学老师给小明出了一道等差数列求和的题目。

但是粗心的小明忘记了一部分的数列，只记得其中N个整数。

现在给出这N个整数，小明想知道包含这N个整数的最短的等差数列有几项？

【输入格式】

输入的第一行包含一个整数N。

第二行包含N个整数$A_1, A_2, \ldots, A_N$（注意$A_1 \sim A_N$并不一定是按等差数列中的顺序给出）。

【输出格式】

输出一个整数表示答案。

【数据范围】

$2 \leq N \leq 100000$

$0 \leq A_i \leq 10^9$

【输入样例】

```
5
2 6 4 10 20
```

【输出样例】

```
10
```

【样例解释】

包含$2, 6, 4, 10, 20$的最短的等差数列是$2, 4, 6, 8, 10, 12, 14, 16, 18, 20$。

【分析】

假设一个等差数列$a_1 \sim a_n$为：$a, a+d, a+2d, \ldots, a+(n-1)d$，则每一项与第一项的差一定是公差$d$的倍数。由于等差数列的项数为$\frac{(a_n-a_1)}{d}+1$，要使项数最少，则$d$要最大，最大的$d$即为每一项与第一项的差的最大公约数。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 100010;
int a[N];
int n;

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

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> a[i];
    sort(a, a + n);
    int d = 0;
    for (int i = 1; i < n; i++) d = gcd(d, a[i] - a[0]);
    if (!d) cout << n << endl;
    else cout << (a[n - 1] - a[0]) / d + 1 << endl;
    return 0;
}
```

二、AcWing 1295. X的因子链

【题目描述】

输入正整数X，求X的大于1的因子组成的满足任意前一项都能整除后一项的严格递增序列的最大长度，以及满足最大长度的序列的个数。

【输入格式】

输入包含多组数据，每组数据占一行，包含一个正整数表示X。

【输出格式】

对于每组数据，输出序列的最大长度以及满足最大长度的序列的个数。

每个结果占一行。

【数据范围】

$1 \leq X \leq 2^{20}$

【输入样例】

```
2
3
4
10
100
```

【输出样例】

```
1 1
1 1
2 1
2 2
4 6
```

【分析】

（1）算术基本定理（唯一分解定理）：

对于任意正整数N，都可将其分解为若干个质数的乘积，即：$N = p_1^{a_1} * p_2^{a_2} * \cdots * p_k^{a_k}$。

我们要构成题中所说的最长序列，那么这个序列一定是由质因数组成的，因为如果其中出现了某个合数，那么一定可以将这个合数继续拆分成更多项。因此该序列可大概理解为：$p_1, p_1^2, p_1^2 * p_2, p_1^2 * p_2^2, p_1^2 * p_2^3, p_1^2 * p_2^3 * p_3, \ldots$。

那么序列的最大长度就为$a_1 + a_2 + \cdots + a_k$。

（2）多重集合组合数：

如何求这样序列的个数呢？先做一个映射，我们不存数的本身，我们存数的增量，原序列是$a_1, a_2, \ldots, a_t$，映射的序列是：$a_1, \frac{a_2}{a_1}, \frac{a_3}{a_2}, \ldots, \frac{a_t}{a_{t-1}}$，这两个序列是一一对应的，给我们第一个序列就可以求第二个序列，在第二个序列中每一个数都是X的质因子，因此序列个数就是X所有质因子的组合排列数。

多重集合组合数可以理解为是高中数学排列组合的一个应用，应用场合是在排列时去除重复相同项排列的情况。公式为：$\frac{(a_1+a_2+\cdots+a_k)!}{a_1!*a_2!*\cdots*a_k!}$。

（3）线性筛法：

线性筛法筛掉的一定是合数，且一定是用其最小质因子筛的，这样我们就可以把所有质数找出来，而且每个合数只会被最小质因子筛，因此我们可以使用$minp[i]$表示i的最小质因子。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

typedef long long LL;
const int N = (1 << 20) + 10;
int primes[N], minp[N];//minp[i]表示i的最小质因数
bool st[N];
int n, cnt;

void get_primes(int n)
{
    for (int i = 2; i <= n; i++)
    {
        if (!st[i]) primes[cnt++] = i, minp[i] = i;
        for (int j = 0; primes[j] <= n / i; j++)
        {
            st[i * primes[j]] = true;
            minp[i * primes[j]] = primes[j];
            if (i % primes[j] == 0) break;
        }
    }
}

int main()
{
    get_primes(1 << 20);
    while (cin >> n)
    {
        int fact[30], sum[30] = { 0 };//分别保存分解出的质因子及次数
```

```
        int k = 0, tot = 0;//k表示不同质因数的数量,tot表示所有质因数的数量
        while (n > 1)
        {
            fact[k] = minp[n];
            while (n % fact[k] == 0) n /= fact[k], sum[k]++, tot++;
            k++;
        }
        LL res = 1;
        for (int i = 2; i <= tot; i++) res *= i;//计算所有质因数数量的阶乘
        for (int i = 0; i < k; i++)
            for (int j = 2; j <= sum[i]; j++)
                res /= j;//除以各个质因数数量阶乘之积
        cout << tot << ' ' << res << endl;
    }
    return 0;
}
```

三、AcWing 1296. 聪明的燕姿

【题目描述】

城市中人们总是拿着号码牌，不停寻找，不断匹配，可是谁也不知道自己等的那个人是谁。

可是燕姿不一样，燕姿知道自己等的人是谁，因为燕姿数学学得好！

燕姿发现了一个神奇的算法：假设自己的号码牌上写着数字S，那么自己等的人手上的号码牌数字的所有正约数之和必定等于S。

所以燕姿总是拿着号码牌在地铁和人海找数字（喂！这样真的靠谱吗）。

可是她忙着唱《绿光》，想拜托你写一个程序能够快速地找到所有自己等的人。

【输入格式】

输入包含k组数据。

对于每组数据，输入包含一个号码牌S。

【输出格式】

对于每组数据，输出有两行。

第一行包含一个整数m，表示有m个等的人。

第二行包含相应的m个数，表示所有等的人的号码牌。

注意：你输出的号码牌必须按照升序排列。

【数据范围】

$1 \leq k \leq 100$

$1 \leq S \leq 2 \times 10^9$

【输入样例】

```
42
```

【输出样例】

```
3
20 26 41
```

【分析】

算术基本定理：$N = p_1^{a_1} * p_2^{a_2} * \cdots * p_k^{a_k}$

约数之和：$S = (1 + p_1 + p_1^2 + \cdots + p_1^{a_1}) * (1 + p_2 + \cdots + p_2^{a_2}) * \cdots * (1 + p_k + \cdots + p_k^{a_k})$

我们最暴力的想法一定就是枚举了，比如对一个数S，枚举$1 \sim S-1$的所有数的约数，再判断他们的约数和是否等于S，但是这么做显然会超时。

那么这个时候我们就可以用DFS进行搜索，看我们能不能得到符合条件的$(1 + p_k + p_k^2 + \cdots + p_k^{a_k})$能够整除$S$，即$S\%(1 + p_k + p_k^2 + \cdots + p_k^{a_k}) = 0$，那么我们就将$S/(1 + p_k + p_k^2 + \cdots + p_k^{a_k})$，再DFS到下一层。当$S = 1$时，说明已经凑出了约数和为$S$。

如果$a_i = 1$的话，$S = (1 + p_i)$，因为p_i为质数，那么$S-1$也一定为质数，那么这个时候只需要判断$S-1$是否为质数即可。我们枚举p_i时只要枚举到$\sqrt{S}$即可。

DFS的参数有$last, prod, S$，其中：$last$表示上一个枚举的质数是谁，我们这样枚举的目的就是先把前面符合条件的质数枚举完了再枚举后面的质数，这样不会带来重复，降低了时间复杂度。例如枚举2之后再DFS到下一层，那么这个时候就应该再从3开始进行枚举而不是再从头开始枚举。$prod$表示当前的所有最高次项$p_i^{a_i}$的乘积，当约数之和凑成S后，$prod$即为相对应的数，因为$N = p_1^{a_1} * p_2^{a_2} * \cdots * p_k^{a_k}$。$S$表示当前还需要凑的剩余的约数和。

每一层我们都应当判断$S-1$是否为质数，如果$S-1$是质数的话也需要记录，但是不需要返回上一层，继续往下搜即可。例如$24 = (1 + 23) = (1 + 2) * (1 + 7) = (1 + 3) * (1 + 5)$。假如第一层的时候判断$23$是质数，那么就记录$res[len++] = 1 * 23$，同时也需要继续往下做然后依次判断$2, 7$或者$3, 5$能不能满足条件，只要有$S-1$为质数，说明就有满足条件的数，这个时候就需要记录结果。

注意$S-1$一定要大于上一层的质数，因为我们要保证质数是从小到大枚举的，只有剩下的$S-1$是大于上一层的质数的时候，$(1+S)$才有可能成为最初的那个S的一个因子。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 50010;//质数筛到5e4时平方已经大于2e9了
int primes[N], res[N];
bool st[N];
int S, cnt, len;

void get_primes(int n)
{
    for (int i = 2; i <= n; i++)
    {
        if (!st[i]) primes[cnt++] = i;
        for (int j = 0; primes[j] <= n / i; j++)
        {
            st[i * primes[j]] = true;
            if (i % primes[j] == 0) break;
        }
    }
}

bool is_prime(int n)
{
    if (n <= 50000) return !st[n];
    for (int i = 0; primes[i] <= n / primes[i]; i++)
        if (n % primes[i] == 0) return false;
    return true;
}

//last表示上一个枚举的质数的下标,prod表示所有最高次项的乘积,S表示当前还剩下的需要凑的数
void dfs(int last, int prod, int S)
{
    if (S == 1) { res[len++] = prod; return; }

    if (S - 1 > (!~last ? 0 : primes[last]) && is_prime(S - 1))
```

```
        res[len++] = prod * (S - 1);//特判一下一次项(1+P)的情况
    for (int i = last + 1; primes[i] <= S / primes[i]; i++)
    {
        int p = primes[i];
        for (int j = 1 + p, t = p; j <= S; t *= p, j += t)
            if (S % j == 0) dfs(i, prod * t, S / j);
    }
}

int main()
{
    get_primes(50000);
    while (cin >> S)
    {
        len = 0;
        dfs(-1, 1, S);
        cout << len << endl;
        if (len)
        {
            sort(res, res + len);
            for (int i = 0; i < len; i++) cout << res[i] << ' ';
            cout << endl;
        }
    }
    return 0;
}
```

四、AcWing 1299. 五指山

【题目描述】

大圣在佛祖的手掌中。

我们假设佛祖的手掌是一个圆圈，圆圈的长为n，逆时针记为：$0,1,2,\ldots,n-1$，而大圣每次飞的距离为d。

现在大圣所在的位置记为x，而大圣想去的地方在y。

要你告诉大圣至少要飞多少次才能到达目的地。

注意：孙悟空的筋斗云只沿着逆时针方向翻。

【输入格式】

有多组测试数据。

第一行是一个正整数T，表示测试数据的组数；

每组测试数据包括一行，四个非负整数，分别为如来手掌圆圈的长度n，筋斗所能飞的距离d，大圣的初始位置x和大圣想去的地方y。

【输出格式】

对于每组测试数据，输出一行，给出大圣最少要翻多少个筋斗云才能到达目的地。

如果无论翻多少个筋斗云也不能到达，输出`Impossible`。

【数据范围】

$2 < n < 10^9$

$0 < d < n$

$0 \le x, y < n$

【输入样例】

```
2
3 2 0 2
3 2 0 1
```

【输出样例】

```
1
2
```

【分析】

扩展欧几里得：对于任意正整数对(a, b)，一定有非零整数x, y，使得$ax + by = gcd(a, b)$。

我们来分情况讨论一下扩展欧几里得定理（设$g = gcd(a, b)$）：

1. 当$b = 0$时，a和b的最大公约数为a，则$x = 1$；
2. 当$b \neq 0$时，$by + (a\%b)x = g \rightarrow by + (a - a/b * b)x = g \rightarrow ax + b(y - a/b * x) = g$。

即当我们用扩展欧几里得定理求x和y时，欧几里得定理每递归一次x不用变，y变为$y - a/b * x$即可。

如果我们求出一对(x, y)，记为(x_0, y_0)，那么其他的x和y可以通过x_0, y_0表示：令$a' = a/gcd(a, b), b' = b/gcd(a, b)$，则其他的$x$和$y$可以表示为：$x = x_0 + kb', y = y_0 - ka'$。

我们来证明一下：$ax_0+by_0=gcd(a,b),ax+by=gcd(a,b)$，如果$x_0$比$x$小，那么$y_0$就应该比$y$大，这样加起来的结果才可能相同。所以，$a(x-x_0)=b(y_0-y)$。两边同时除以一个$gcd(a,b)$，得到：$a'(x-x_0)=b'(y_0-y)$，那么$b'$就应该能整除$a'(x-x_0)$，又因为$a'$和$b'$互质，所以$b'$应该能整除$(x-x_0)$。所以$(x-x_0)=kb'$，则$x=x_0+kb',y=y_0-ka'$，这里的$k$可正可负。

回到本题，假设当前位置为s，目标位置为t，一共飞了y次，则$s+yd=t+xn$，即$-xn+yd=t-s$。式中除x,y以外其余的数均为常数，因此需要求这个方程所有解中的y的最小值。

扩展欧几里得能够求出$-xn+yd=gcd(n,d)$的一组解(x_0,y_0)，显然如果$t-s$不是$gcd(n,d)$的整数倍那么无解。如果有解，则我们将上述方程两边同时乘上$(t-s)/gcd(n,d)$，这样就可以得到原方程的一组解(x_0,y_0)。因为之前我们证过了获得一组解后其他解就可以表示出来了，我们只需要求一下所有解中的最小值就可以了（注意我们的y只能是正数，因为你总不能让大圣反着翻吧）。即求$y=y_0+k*(n/gcd(n,d))$的最小值即可，也就是$miny=y_0\%(n/gcd(n,d))$。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

typedef long long LL;

LL exgcd(LL a, LL b, LL& x, LL& y)
{
    if (!b)
    {
        x = 1, y = 0;
        return a;
    }
    LL res = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return res;
}

int main()
{
    int T;

    cin >> T;
```

```
    while (T--)
    {
        LL n, d, s, t, x, y;
        cin >> n >> d >> s >> t;
        LL g = exgcd(n, d, x, y);
        if ((t - s) % g) { puts("Impossible"); continue; }
        y *= (t - s) / g, n /= g;
        cout << (y % n + n) % n << endl;
    }
    return 0;
}
```

五、AcWing 1223. 最大比例

【题目描述】

X星球的某个大奖赛设了M级奖励。

每个级别的奖金是一个正整数。

并且，相邻的两个级别间的比例是个固定值。

也就是说：所有级别的奖金数构成了一个等比数列。

比如：$16, 24, 36, 54$，其等比值为：$3/2$。

现在，我们随机调查了一些获奖者的奖金数。

请你据此推算可能的最大的等比值。

【输入格式】

第一行为数字N，表示接下的一行包含N个正整数。

第二行N个正整数X_i，用空格分开，每个整数表示调查到的某人的奖金数额。

【输出格式】

一个形如A/B的分数，要求A, B互质，表示可能的最大比例系数。

【数据范围】

$0 < N < 100$

$0 < X_i < 10^{12}$

数据保证一定有解。

【输入样例1】

```
3
1250 200 32
```

【输出样例1】

```
25/4
```

【输入样例2】

```
4
3125 32 32 200
```

【输出样例2】

```
5/2
```

【输入样例3】

```
3
549755813888 524288 2
```

【输出样例3】

```
4/1
```

【分析】

假设原数列为一个公比为$\frac{p}{q}$的等比数列：$a, a(\frac{p}{q})^1, a(\frac{p}{q})^2, a(\frac{p}{q})^3, \ldots, a(\frac{p}{q})^n$。题中所给的数列为：$b_0, b_1, \ldots, b_m$。我们将$b_1$开始的每一项都除以$b_0$，则为$\frac{b_1}{b_0}, \frac{b_2}{b_0}, \ldots, \frac{b_m}{b_0} \rightarrow (\frac{p}{q})^{x_1}, (\frac{p}{q})^{x_2}, \ldots, (\frac{p}{q})^{x_m}$。

我们需要求的是最大的$(\frac{p}{q})^k$，由于$\frac{p}{q} > 1$，因此也就是求最大的k，即求$x_1, x_2, \ldots, x_m$的最大公约数。这里我们使用更相减损术，因为我们没有得到确切的$x_1 \sim x_m$是多少，要用更相减损术的是指数，所以假如要求$(\frac{p}{q})^{x_2 - x_1}$可以转换成$(\frac{p}{q})^{x_2} / (\frac{p}{q})^{x_1}$，不断两两相除，除到结果为$1$，即$x_1 = x_2$，此时幂次为$0$。注意此处求最大公约数时可以将分子和分母分开求解，因为分子和分母的幂次是相同的。

【代码】

```
#include <iostream>
#include <cstring>
```

```cpp
#include <algorithm>
using namespace std;

typedef long long LL;
const int N = 110;
LL x[N], a[N], b[N];
int n, cnt;

LL gcd(LL a, LL b)
{
    return b ? gcd(b, a % b) : a;
}

LL gcd_sub(LL a, LL b)
{
    if (a < b) swap(a, b);
    if (b == 1) return a;
    return gcd_sub(b, a / b);
}

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> x[i];
    sort(x, x + n);
    for (int i = 1; i < n; i++)
        if (x[i] != x[i - 1])//注意需要判重
        {
            LL g = gcd(x[i], x[0]);
            a[cnt] = x[i] / g;
            b[cnt++] = x[0] / g;
        }
    LL up = a[0], down = b[0];//最大比例系数的分子和分母
    for (int i = 1; i < cnt; i++)
        up = gcd_sub(up, a[i]), down = gcd_sub(down, b[i]);
    cout << up << '/' << down << endl;
    return 0;
}
```

六、AcWing 1301. C 循环

【题目描述】

对于C语言的循环语句，形如：

```
for (variable = A; variable != B; variable += C)
  statement;
```

请问在k位存储系统中循环几次才会结束。

若在有限次内结束，则输出循环次数。否则输出死循环。

【输入格式】

多组数据，每组数据一行四个整数A, B, C, k。

读入以`0 0 0 0`结束。

【输出格式】

若在有限次内结束，则输出循环次数。

否则输出`FOREVER`。

【数据范围】

$1 \leq k \leq 32$

$0 \leq A, B, C < 2^k$

【输入样例】

```
3 3 2 16
3 7 2 16
7 3 2 16
3 4 2 16
0 0 0 0
```

【输出样例】

```
0
2
32766
FOREVER
```

【分析】

---

k位存储系统就是数值只保留最后k位，也就是不会超过2^k。

那么本题就是求解方程：$(A+x*C)\%2^k=B$，即$A+x*C-y*2^k=B$，那么就和第四题一样了，我们只需要先求出一组解(x_0,y_0)，然后找出x最小的那组解即可。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

typedef long long LL;
LL A, B, C, k;

LL exgcd(LL a, LL b, LL& x, LL& y)
{
    if (!b)
    {
        x = 1, y = 0;
        return a;
    }
    LL res = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return res;
}

int main()
{
    while (cin >> A >> B >> C >> k, A || B || C || k)
    {
        k = 1ll << k;//特别要注意此处需要加ll,不然会爆int
        LL x, y;
        LL g = exgcd(C, k, x, y);
        if ((B - A) % g) { puts("FOREVER"); continue; }
        x *= (B - A) / g, k /= g;
        cout << (x % k + k) % k << endl;
    }
    return 0;
}
```

七、AcWing 1225. 正则问题

【题目描述】

考虑一种简单的正则表达式：

只由`x ( ) |`组成的正则表达式。

小明想求出这个正则表达式能接受的最长字符串的长度。

例如`((xx|xxx)x|(x|xx))xx`能接受的最长字符串是：`xxxxxx`，长度是**6**。

【输入格式】

一个由`x ( ) |`组成的正则表达式。

【输出格式】

输出所给正则表达式能接受的最长字符串的长度。

【数据范围】

输入长度不超过**100**，保证合法。

【输入样例】

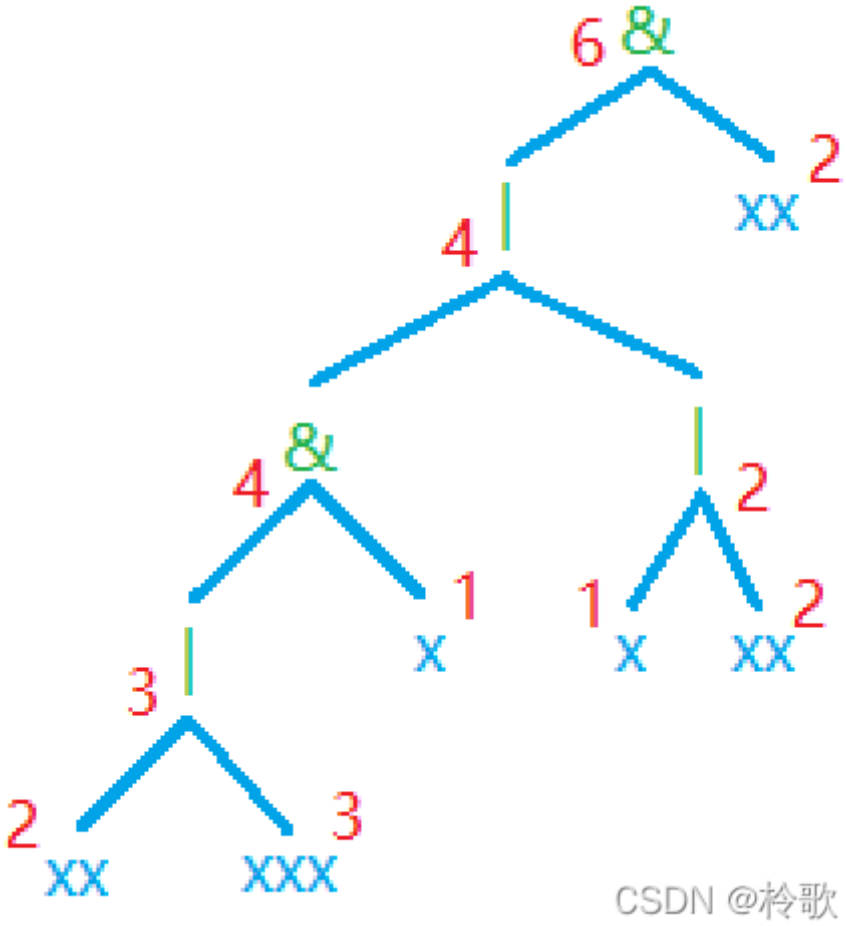
```
((xx|xxx)x|(x|xx))xx
```

【输出样例】

```
6
```

【分析】

先画出递归搜索树如下图所示：



如果遇到`|`则左右两边取最大值，否则直接拼接即可。注意搜索的时候有两种不同的递归，一种是遇到`(`递归，其结束递归的标志是相对应的`)`，还有一种是遇到`|`递归，该递归是求出两边的值然后取max，因此在`|`递归中遇到`)`不能回溯，`)`需要在`(`的递归层中回溯。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#include <string>
using namespace std;

string s;
int k;

int dfs()
{
    int res = 0;
    while (k < s.size())
    {
```

```
        if (s[k] == '(') { k++; res += dfs(); k++; }//k++分别跳过'('和')'
        else if (s[k] == ')') break;
        else if (s[k] == '|') { k++, res = max(res, dfs()); }//k++跳过'|'
        else { k++, res++; }//k++跳过'x'
    }
    return res;
}

int main()
{
    cin >> s;
    cout << dfs() << endl;
    return 0;
}
```

八、AcWing 1243. 糖果

【题目描述】

糖果店的老板一共有M种口味的糖果出售。

为了方便描述，我们将M种口味编号$1 \sim M$。

小明希望能品尝到所有口味的糖果。

遗憾的是老板并不单独出售糖果，而是K颗一包整包出售。

幸好糖果包装上注明了其中K颗糖果的口味，所以小明可以在买之前就知道每包内的糖果口味。

给定N包糖果，请你计算小明最少买几包，就可以品尝到所有口味的糖果。

【输入格式】

第一行包含三个整数N, M, K。

接下来N行每行K个整数$T_1, T_2, \ldots, T_K$，代表一包糖果的口味。

【输出格式】

一个整数表示答案。

如果小明无法品尝所有口味，输出-1。

【数据范围】

$1 \leq N \leq 100$

$1 \leq M, K \leq 20$

$1 \leq T_i \leq M$

【输入样例】

```
6 5 3
1 1 2
1 2 3
1 1 3
2 3 5
5 4 2
5 1 2
```

【输出样例】

```
2
```

【分析】

我们用二进制数$state$表示糖果口味的状态（将口味$1 \sim M$映射为$0 \sim M-1$），第i位为1表示有该种口味的糖果，则我们的目标就是选择最少的包数使得状态中的每一位均为1。

迭代加深：我们从小到大枚举答案$depth$，然后搜索购买$depth$包糖果是否能得到所有口味，如果$depth > M$时还无法得到所有口味，那么就无解，因为最极端的情况就是每一种口味只有唯一的一包糖与之对应，如果M包还得不到所有口味那么说明有口味没有在任何一包中出现过。

优化搜索顺序：优先搜索选择数量较少的口味，假如某种口味只在一包中出现，那么我们肯定先选上这一包。

可行性剪枝：我们可以使用估价函数$h(state)$表示至少还需要选几包糖才能将$state$全部变成1。我们在计算估价函数值的时候，如果需要选择某一种口味，那么就将这个口味的每一包糖都选上，然后记为选了一次，这样算出来的结果一定小于等于真实值。

注意本题可能有多包口味相同的糖，因此我们需要进行判重，将重复的删除，否则会超时。

【代码】

```
#include <iostream>
#include <cstring>

#include <algorithm>
```

```cpp
#include <vector>
#include <cmath>
using namespace std;

const int N = 30;
vector<int> col[N];//表示每列所有可选的行的状态
int n, m, k;

int lowbit(int x)
{
    return x & -x;
}

int h(int state)//返回至少还需要选几行能将state全变为1
{
    int res = 0;
    for (int i = (1 << m) - 1 - state; i; i -= lowbit(i))
    {
        int c = log2(lowbit(i));//最低位1的位置
        for (auto row : col[c]) i &= ~row;//将这一列能选的所有行都选上
        res++;
    }
    return res;
}

bool dfs(int depth, int state)//判断是否能够选择depth行使得state变为每一列全是1
{
    if (!depth || h(state) > depth) return state == (1 << m) - 1;
    int t = -1;//表示剩余的列中选择数量最少的列
    for (int i = (1 << m) - 1 - state; i; i -= lowbit(i))
    {
        int c = log2(lowbit(i));
        if (!~t || col[c].size() < col[t].size()) t = c;
    }
    for (auto row : col[t])
        if (dfs(depth - 1, state | row)) return true;
    return false;
}

int main()
{

    cin >> n >> m >> k;
```

```
    for (int i = 0; i < n; i++)
    {
        int state = 0;
        for (int j = 0; j < k; j++)
        {
            int c;
            cin >> c;
            state |= 1 << c - 1;//注意将1~M映射为0~M-1
        }
        for (int j = 0; j < m; j++)
            if (state >> j & 1) col[j].push_back(state);
    }
    //注意数据加强需要将重复的行去重
    for (int i = 0; i < m; i++)
    {
        sort(col[i].begin(), col[i].end());
        col[i].erase(unique(col[i].begin(), col[i].end()), col[i].end());
    }
    int depth = 0;//m列最多选择m行,即每列都只有唯一一行与之对应的情况,超出m行无解
    while (depth <= m && !dfs(depth, 0)) depth++;
    if (depth > m) cout << -1 << endl;
    else cout << depth << endl;
    return 0;
}
```