AcWing 879. 幸存者游戏

有nn个同学围成一圈，其idid依次为11~nn（nn号挨着11号）。

现在从11号开始报数，第一回合报到mm的人就出局，第二回合从出局的下一个人开始报数，报到m2m2的同学出局。

以此类推，直到最后一个回合报到mn−1mn−1的人出局，剩下最后一个同学。

输出这个同学的编号。

输入格式

共一行，包含两个整数n和m。

输出格式

输出最后剩下的同学的编号。

数据范围

n≤15,m≤5n≤15,m≤5

输入样例：

```
5 2
```

输出样例：

```
5
```

```
/*
模拟题
约瑟夫问题
1.DP方法
2.暴力 + 优化    k * m % r
bool数组表示每个数是不是被踢掉了
*/
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 16;

bool st[N];

int main()
{
    int n, m;
    cin >> n >> m;

    int p = 1;
    for (int i = 1, r = n; i <= n; i ++, r -- )//r表示圈里剩下的人数
```

```
    {
        int k = 1;//k表示m的k次方
        for (int j = 1; j <= i; j ++ ) k = k * m % r;
        if (k == 0) k = r;//余数是0，删掉第r个人；
        while (true)
        {
            if (!st[p])
            {
                k -- ;
                if (!k)
                {
                    st[p] = true;//删掉
                    break;
                }
            }
            p ++ ;
            if (p > n) p = 1;//一圈
        }
    }

    cout << p << endl;
    return 0;
}
```

AcWing 880. 数字对生成树

根据输入生成一棵树，并打印该树。

输入为N行数字对序列（例如：N = 2，数字对序列为(1, 2)，(2, 3)），其中数字代表该树的节点，左边数字代表的节点是右边数字代表的节点的父节点。

请根据输入的数字对序列生成一棵树，并根据广度优先的顺序打印该树。

注意，存在输入无法生成树的情况，例如(1, 2)，(1, 3)，(2, 4)，(3, 4)，根据该序列生成的图中，2节点和3节点同时为4节点的父节点，所以根据定义该图不是树。

如遇无法生成树的情况，请输出字符串”Not a tree”。

广度优先遍历树，并打印输出时，同一层级的节点根据输入时节点出现的顺序打印输出。

输入格式

第一行包含整数N。

接下来N行，每行包含两个不同的正整数，表示一个数字对，两数之间用英文逗号分隔，例如：1,2。

没有默认的输入顺序，第一行可能不是根节点，最后一行可能不是叶子节点。

可能有完全重复的行。

输出格式

输出共一行，如果可以生成树，请根据广度优先顺序，输出每个节点对应的数字，并用英文逗号隔开，同一层级的节点根据输入顺序输出。

如果无法生成树，请输出字符串”Not a tree”。

数据范围

1≤N≤100001≤N≤10000
整数对中的正整数均小于2的31次方。

输入样例1：

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

输出样例1：

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

输入样例2：

```
4
1,2
1,3
2,4
3,4
```

输出样例2：

```
Not a tree
```

输入样例3:

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

输出样例3：

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

```
/*
1.有一个根节点
2.可以从根节点遍历所有点
3.每个点只能被遍历一次

map可以很大
*/
#include <iostream>
#include <algorithm>
#include <map>
#include <vector>
#include <queue>
```

```
using namespace std;

map<int, vector<int>> h; // 存储整个图，例如h[5]存储 以5为起点的所有边
map<int, int> timestamp, dist;//timestamp表示什么时间插入的，dist表示离根节点的距离
map<int, bool> has_father;//是否有父节点
map<pair<int, int>, bool> edges;//是否有重边


vector<int> bfs(int root)
{
    queue<int> q;
    q.push(root);
    dist[root] = 0;
    vector<int> nodes;

    while (q.size())
    {
        auto t = q.front();
        q.pop();

        nodes.push_back(t);

        for (auto ver : h[t])//t的所有出边
        {
            if (dist.count(ver)) return vector<int>();
            dist[ver] = dist[t] + 1;
            q.push(ver);
        }
    }

    vector<vector<int>> ans;
    for (auto ver : nodes) ans.push_back({dist[ver], timestamp[ver], ver});//排序

    sort(ans.begin(), ans.end());//按字典序比较vector

    nodes.clear();
    for (auto t : ans) nodes.push_back(t[2]);

    return nodes;
}


int main()
{
    int n;
    cin >> n;

    int tm = 0;
    while (n -- )
    {
        int a, b;
        scanf("%d,%d", &a, &b);

        if (edges[{a, b}]) continue;
        edges[{a, b}] = true;

        if (timestamp.count(a) == 0) timestamp[a] = tm ++ ;
```

```
        if (timestamp.count(b) == 0) timestamp[b] = tm ++ ;

        has_father[b] = true;
        h[a].push_back(b);//加入这条边
    }

    int root = -1;
    for (auto x : timestamp)//是否有根节点
    {
        int p = x.first;
        if (!has_father[p])
        {
            root = p;
            break;
        }
    }

    if (!root) puts("Not a tree");
    else
    {
        auto res = bfs(root);//从根节点遍历

        if (res.size() < timestamp.size()) puts("Not a tree");//不能从根节点遍历到
其他点
        else
        {
            cout << res[0];
            for (int i = 1; i < res.size(); i ++ ) cout << ',' << res[i];
            cout << endl;
        }
    }

    return 0;
}
```

AcWing 881. 整理书架

图书管理员小P每天要整理书架，一个书架有N排，每一排书架上能摆放k本书，每本书上都有索引的数字编号，例如1,5,7等等。

小P喜欢从数字编号排列最整齐的书架开始整理，因为这样的话这排书架上的书就不用整理，按照整齐程度整理，最后整理最不整齐的那排书架。

那么能否请机智的你帮助小P找出整理书架的顺序呢?

整齐程度的定义：每排书架中书的编号存在的逆序对越少，这排书架就越整齐，一排书架中若书的编号完全升序即为最整洁。

逆序对的定义：在一个数组A中，在i < j的情况下，有A[i] > A[j]，则(i,j)就称 为数组A中的一个逆序对。

输入格式

第一行输入N，表示书架排数。

第二行输入k，表示每排书架上书的数量。

之后的N*k的数组表示每本书的数字编号。

输出格式

输出按照整齐程度，对各排书架重新排序后得到的新N*k的数组。

输出共一行，具体形式参考输出样例。

注意，逆序数相同则按书架原有顺序整理。

数据范围

1≤N,k≤2001≤N,k≤200,
0≤数字编号≤100000≤数字编号≤10000

输入样例：

```
4
8
0 1 2 3 4 5 6 7
11 6 5 7 3 2 2 0
2 3 6 1 9 3 5 4
0 2 4 5 3 10 6 7
```

输出样例：

```
[[0, 1, 2, 3, 4, 5, 6, 7], [0, 2, 4, 5, 3, 10, 6, 7], [2, 3, 6, 1, 9, 3, 5, 4],
[11, 6, 5, 7, 3, 2, 2, 0]]
```

```cpp
/*
矩阵，逆序对
每行排序，稳定排序stable_sort()
*/
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 210;

int n, m;
int a[N][N], cnt[N];
int index[N];//index[0]表示逆序对最少的那行的编号


bool cmp(int a, int b)//重载比较函数，比较每行的逆序对的大小
{
    return cnt[a] < cnt[b];
}


int main()
{
    cin >> n >> m;
    for (int i = 0; i < n; i ++ )
    {
        for (int j = 0; j < m; j ++ ) cin >> a[i][j];
        for (int j = 0; j < m; j ++ )//cnt表示逆序对的数量
            for (int k = j + 1; k < m; k ++ )
```

```
                if (a[i][j] > a[i][k])
                    cnt[i] ++ ;

        index[i] = i;
    }

    stable_sort(index, index + n, cmp);

    cout << '[';
    for (int i = 0; i < n; i ++ )
    {
        cout << '[';
        for (int j = 0; j < m; j ++ )
        {
            cout << a[index[i]][j];
            if (j != m - 1) cout << ", ";
        }
        cout << ']';
        if (i != n - 1) cout << ", ";
    }
    cout << ']';

    return 0;
}
```

AcWing 882. 飞机最低可俯冲高度

近日，埃航空难的新闻牵动了无数人的心。

据悉，空难很可能是由于波音737MAX飞机的失速保护系统错误触发所致。

在飞机进行高空飞行时，驾驶辅助系统如果检测到飞机失速，无法维持足够的飞行升力，会压低机头进行俯冲，以重新获得速度，进而获取足够的飞行升力，维持飞行高度。

但是在飞机进行低空飞行时，触发俯冲机制极有可能在飞机还未获得足够飞行速度并上升之前已经撞击地面。

鉴于半年内的两起事故，波音公司决定在低于一定高度时屏蔽自动俯冲机制，现提供K架飞机用于测试最低可俯冲高度，设定需要测试的海拔范围为1~H（单位米）（注意：测试高度只从整数中选取），请问最不理想情况下，至少需要多少次才能求出飞机的最低可俯冲高度？

输入格式

输入为整数K, H，用空格分隔。

K代表用于测试的飞机数量，H代表需要测试的高度范围为1~H米（包含H）。

输出格式

输出整数N，代表最坏情况下需要测试的次数。

数据范围

1≤K≤201≤K≤20
1≤H≤10001≤H≤1000

输入样例1：

```
1 1000
```

输出样例1：

```
1000
```

输入样例2:

```
15 1000
```

输出样例2：

```
10
```

样例解释

在样例#1中，只有一架飞机用来测试的情况下，从最高高度1000米，逐次减1m进行测试，直到飞机坠毁。

在样例#2中，飞机数量足够多，每次均使用二分法进行测试。

说明

1-H为低空飞行高度范围，所有大于H的高度都不属于低空飞行，不会在俯冲过程中撞击地面，不需要进行测试。

如果飞机俯冲测试过程中撞击地面坠毁，可以推断本次测试高度低于飞机实际最低可俯冲高度，可测试飞机数量减1。

如果飞机俯冲测试过程中撞击地面

```
/*
和扔鸡蛋一样
有单调性，DP

状态表示：f[i,j]表示有i层楼j个鸡蛋的情况下，最少需要测试多少次
=

第一个鸡蛋，从第k层楼上扔
    1.鸡蛋碎了：1 + f[k -1,j - 1]
    2.鸡蛋没碎：1 + f[i - k, j]
状态转移：f[i,j] = min(max(1 + f[k -1,j - 1],1 + f[i - k, j]))

*/
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 1010, M = 21;

int n, m;
int f[N][M];

int main()
```

```
{
    cin >> m >> n;

    for (int i = 1; i <= n; i ++ ) f[i][1] = i;
    for (int i = 1; i <= m; i ++ ) f[1][i] = 1;

    for (int i = 2; i <= n; i ++ )
        for (int j = 2; j <= m; j ++ )
        {
            f[i][j] = f[i][j - 1];
            for (int k = 1; k <= i; k ++ )
                f[i][j] = min(f[i][j], 1 + max(f[k - 1][j - 1], f[i - k][j]));
        }

    cout << f[n][m] << endl;

    return 0;
}
```