A – Maze

https://vjudge.net/contest/359331#status/MLongyu/A/0/

东东有一张地图，想通过地图找到妹纸。地图显示，0 表示可以走，1 表示不可以走，左上角是入口，右下角是妹纸，这两个位置保证为 0。既然已经知道了地图，那么东东找到妹纸就不难了，请你编一个程序，写出东东找到妹纸的最短路线。

Input

输入是一个 5 × 5 的二维数组，仅由 0、1 两数字组成，表示法阵地图。

Output

输出若干行，表示从左上角到右下角的最短路径依次经过的坐标，格式如样例所示。数据保证有唯一解。

Sample Input

0 1 0 0 0
0 1 0 1 0
0 1 0 1 0
0 0 0 1 0
0 1 0 1 0

Sample Output

(0, 0)
(1, 0)
(2, 0)
(3, 0)
(3, 1)
(3, 2)
(2, 2)
(1, 2)
(0, 2)
(0, 3)
(0, 4)
(1, 4)
(2, 4)
(3, 4)
(4, 4)

Hint

坐标(x, y)表示第 x 行第 y 列，行、列的编号从 0 开始，且以左上角为原点。

另外注意，输出中分隔坐标的逗号后面应当有一个空格。

解题思路：

迷宫问题可以理解成求最短路径问题，可以使用广度优先搜索方法（BFS）+队列来实现。题目中矩阵以 0/1 矩阵方式储存，可以设定四个方向的向量表示四个方向。

```
int dx[4] = {-1, 1, 0, 0};
int dy[4] = {0, 0, -1, 1};
```

每次移动进行判断以保证不会超出矩阵范围且不会走到已经走过的点，没确定一个点就把它

存入路径数组并且标记该点防止重复经过同一点。BFS 过程中如果发现到达了目标点就利用已经保存的路径进行回溯。经过点的存储结构为

```
struct point {
    int steps;//fangxiang
    int p[30];
    int x, y;
};
```

其中 steps 为到达当前点需要的步数，数组 p 为到达当前点需要经过哪些步骤：

```
if (u.x == endX && u.y == endY) {/*arrive at the destiny*/
    int xx = 0, yy = 0;
    cout << "(" << xx << ", " << yy << ")\n";
    for (int i = 1; i <= u.steps; i++) {
        xx = xx + dx[u.p[i]];
        yy = yy + dy[u.p[i]];
        cout << "(" << xx << ", " << yy << ")\n";
    }
    return;
}
```

完整代码：

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

int g[6][6];
bool visited[6][6];
struct point {
    int steps;//fangxiang
    int p[30];
    int x, y;
};
int startX = 0, startY = 0, endX = 4, endY = 4;
int dx[4] = {-1, 1, 0, 0};
int dy[4] = {0, 0, -1, 1};
point que[30], v, u, s;

void bfsPath() {
    s.x = startX;
    s.y = startY;
    s.steps = 0;
    s.p[s.steps] = 0;
    int f = 0, e = 0;
    que[e++] = s;
    visited[startX][startY] = true;
    while (f <= e) {
        u = que[f++];
        if (u.x == endX && u.y == endY) {/*arrive at the destiny*/
            int xx = 0, yy = 0;
```

```
            cout << "(" << xx << ", " << yy << ")\n";
            for (int i = 1; i <= u.steps; i++) {
                xx = xx + dx[u.p[i]];
                yy = yy + dy[u.p[i]];
                cout << "(" << xx << ", " << yy << ")\n";
            }
            return;
        }
        for (int i = 0; i < 4; ++i) {
            int nx = u.x + dx[i];
            int ny = u.y + dy[i];
            if (nx >= 0 && nx < 5 && ny >= 0 && ny < 5 && g[nx][ny] != 1 &&
                !visited[nx][ny]) {/*不出界且未走过*/
                v.x = nx;
                v.y = ny;
                v.steps = u.steps + 1;
                for (int j = 0; j < u.steps; ++j) {
                    v.p[j] = u.p[j];
                }
                v.p[v.steps] = i;
                que[e++] = v;
                visited[nx][ny] = true;
            }
        }
    }
}

int main() {
    int m = 5, n = 5;
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> g[i][j];
        }
    }
    memset(visited, false, sizeof(visited));
    bfsPath();
    //system("pause");
    return 0;
}
```

B - Pour Water

https://vjudge.net/contest/359331#problem/B

倒水问题 "fill A" 表示倒满 A 杯，"empty A"表示倒空 A 杯，"pour A B" 表示把 A 的水倒到 B 杯并且把 B 杯倒满或 A 倒空。

Input

输入包含多组数据。每组数据输入 A, B, C 数据范围 0 < A <= B 、C <= B <=1000 、A 和 B 互质。

Output

你的程序的输出将由一系列的指令组成。这些输出行将导致任何一个罐子正好包含 C 单位的水。每组数据的最后一行输出应该是“success”。输出行从第 1 列开始，不应该有空行或任何尾随空格。

Sample Input

2 7 5

2 7 4

Sample Output

fill B

pour B A

success

fill A

pour A B

fill A

pour A B

success

Notes

如果你的输出与 Sample Output 不同，那没关系。对于某个"A B C"本题的答案是多解的，不能通过标准的文本对比来判定你程序的正确与否。 所以本题由 SPJ（Special Judge）程序来判定你写的代码是否正确。

解题思路：

可以把整个倒水的过程形成得到的图当做一个六叉树（六种操作），其中部分分支经过判断无法生成，而我们要做的就是在这颗树中找到某一节点，此时两个水杯其中有一个水量为目标水量。和先前做过的迷宫思路一样，也是以最短路径找到目标点的过程，采用 BFS 广度优先搜索+队列的方法，从初始状态开始，逐一判断当前状态可进行的操作并生成相应分支直到其中一杯水满足要求。

在此过程中还要确保不会产生已产生的两个杯子的状态（由 AB[1001][1001]数组辅助实现），opNum[x][y]数组用于储存两个杯子水量分别为 x，y 状态的上一步操作的操作数

输出步骤时将每一步操作数存在一个数组然后利用它输出操作。

完整代码：

```
#include "iostream"
#include "queue"
#include "vector"
using namespace std;
struct status  //记录状态的结构体
{
```

```
    int A, B;  // A，B 杯子水含量
    status(int _A, int _B) {
        A = _A;
        B = _B;
    }
};
void Print(int& x, int& y, int& B);
int AB[1001][1001], opNum[1001][1001];
void bfs(int A, int B, int C)  //广度优先搜索
{
    queue<status> q;
    q.push(status(0, 0));
    for (int i = 0; i <= A; i++) {
        for (int j = 0; j <= B; j++) {
            AB[i][j] = -1;
        }
    }

    AB[0][0] = 0;
    while (!q.empty()) {
        status cur = q.front();
        q.pop();
        int a = cur.A, b = cur.B;
        //switch
        for (int i = 0; i < 6; i++) {
            int x = -1, y = -1;
            if (i == 0 && a < A && -1 == AB[A][b]) {
                x = A;
                y = b;
            }
            if (i == 1 && b < B && -1 == AB[a][B]) {
                x = a;
                y = B;
            }
            if (i == 2 && b < B && a > 0) {
                if (a > B - b && -1 == AB[a - B + b][B]) {
                    x = a - B + b;
                    y = B;
                }
                if (a < B - b && -1 == AB[0][b + a]) {
                    x = 0;
                    y = b + a;
                }
            }
```

```
            if (i == 3 && a < A && b > 0) {
                if (b > A - a && -1 == AB[A][b - A + a]) {
                    y = b - A + a;
                    x = A;
                }
                if (b < A - a && -1 == AB[b + a][0]) {
                    y = 0;
                    x = b + a;
                }
            }
            if (i == 4 && -1 == AB[0][b]) {
                x = 0;
                y = b;
            }
            if (i == 5 && -1 == AB[a][0]) {
                x = a;
                y = 0;
            }

            if (x != -1 && y != -1) {
                opNum[x][y] = i;
                AB[x][y] = a * B + a + b;
                if (x == C || y == C)  //罐子中包含 C 单位的水，结束搜索
                {
                    Print(x, y, B);
                    return;
                }
                q.push(status(x, y));
            }
        }
    }
}
void Print(int& x, int& y, int& B)  //输出
{
    int a = x, b = y;
    vector<int> ans;
    while (AB[a][b] != a * B + a + b) {
        int ta = AB[a][b] / (B + 1), tb = AB[a][b] % (B + 1);
        ans.push_back(opNum[a][b]);
        a = ta;
        b = tb;
    }
    for (int i = ans.size() - 1; i >= 0; i--) {
        switch (ans[i]) {
```

```
            case 0:
                cout << "fill A\n";
                break;
            case 1:
                cout << "fill B\n";
                break;
            case 2:
                cout << "pour A B\n";
                break;
            case 3:
                cout << "pour B A\n";
                break;
            case 4:
                cout << "empty A\n";
                break;
            default:
                cout << "empty B\n";
                break;
        }
    }
    cout << "success\n";
}
int main() {
    int A, B, C;
    while (cin >> A >> B >> C)
        if (0 == C)
            cout << "success\n";
        else
            bfs(A, B, C);
    // system("pause");/*吸取教训，这玩意不能一块带进 vj*/
    return 0;
}
```