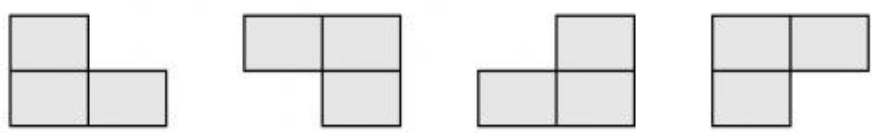
P1228 地毯填补问题

题目描述

相传在一个古老的阿拉伯国家里，有一座宫殿。宫殿里有个四四方方的格子迷宫，国王选择驸马的方法非常特殊，也非常简单：公主就站在其中一个方格子上，只要谁能用地毯将除公主站立的地方外的所有地方盖上，美丽漂亮聪慧的公主就是他的人了。公主这一个方格不能用地毯盖住，毯子的形状有所规定，只能有四种选择（如图）：



并且每一方格只能用一层地毯，迷宫的大小为 $2^k \times 2^k$的方形。当然，也不能让公主无限制的在那儿等，对吧？由于你使用的是计算机，所以实现时间为 1s。

输入格式

输入文件共 2 行。

第一行：k，即给定被填补迷宫的大小为$2^k \times 2^k$（$0<k\leqslant 10$）；

第二行：x,y，即给出公主所在方格的坐标（x为行坐标，y 为列坐标），x 和 y 之间有一个空格隔开。

输出格式

将迷宫填补完整的方案：每一补(行)为x y c（x,y为毯子拐角的行坐标和列坐标， c 为使用毯子的形状，具体见上面的图 1，毯子形状分别用 1,2,3,4 表示，x,y,c之间用一个空格隔开）。

输入样例

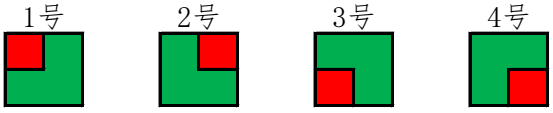
3
3 3

输出样例

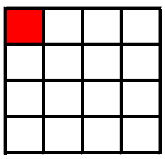
5 5 1
2 2 4
1 1 4
1 4 3
4 1 2
4 4 1
2 7 3
1 5 4
1 8 3
3 6 3
4 8 1
7 2 2
5 1 4
6 3 2
8 1 2
8 4 1
7 7 1
6 6 1
5 8 3
8 5 2
8 8 1

解析

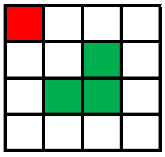
我们先从最简单的模型开始推理，假设当前的格子是2*2的大小。那么一共只有以下4种情况其中红色代表公主：



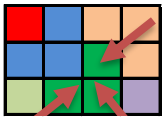
继续考虑更加复杂的情况，当迷宫为4*4大小时，该如何计算呢？例如下面这种情况：



这时，我们只要在中间的位置摆放一个朝向公主的地毯，就能把整个地图切割成4个2*2的小块，如下所示：



此时，这个地毯的三个角各占到了每个小正方形的一个角。对于三个小正方形来说，这个绿色的角就是他们的“公主”，只要他们使用面向公主的地毯就能够完美解决。如下所示：



通过分析，我们可以得出如下规律。当边长扩展到n时：

1、首先选择一个朝向公主的地毯，然后将迷宫平均切成4块。

2、然后分别对这四块小正方形继续进行递归切割操作，直到小正方形的边长为1。

编码

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

using namespace std;

//x1,y1是公主的位置
//x2,y2表示此正方形左上角的位置
//n为当前正方形的边长
void solve(int x1, int y1, int x2, int y2, int n) {
    //边长为1时，不需要继续切割
    if (n == 1) {
        return;
    }
    //在上半部
    if (x1 - x2 < (n >> 1)) {
        //目标在左上角
        if (y1 - y2 < (n >> 1)) {
            //使用1号地毯
            cout << (x2 + (n >> 1)) << ' ' << (y2 + (n >> 1)) <<
            ' ' << 1 << endl;
            //继续向下传递公主的位置和当前方块的左上角位置
            solve(x1, y1, x2, y2, (n >> 1));
            //分别将地毯的三个角作为公主位置进行传递，同时边长进行缩减
            solve(x2 + (n >> 1) - 1, y2 + (n >> 1),
                  x2, y2 + (n >> 1), (n >> 1));
            solve(x2 + (n >> 1), y2 + (n >> 1) - 1,
                  x2 + (n >> 1), y2, (n >> 1));
            solve(x2 + (n >> 1), y2 + (n >> 1),

                  x2 + (n >> 1), y2 + (n >> 1), (n >> 1));
        }
            //右上角
        else {
            cout << (x2 + (n >> 1)) << ' ' << (y2 + (n >> 1) - 1)
            ' ' << 2 << endl;
            solve(x2 + (n >> 1) - 1, y2 + (n >> 1) - 1,
                  x2, y2, (n >> 1));
            solve(x1, y1, x2, y2 + (n >> 1), (n >> 1));
            solve(x2 + (n >> 1), y2 + (n >> 1) - 1,
                  x2 + (n >> 1), y2, (n >> 1));
```

```
            solve(x2 + (n >> 1), y2 + (n >> 1),
                  x2 + (n >> 1), y2 + (n >> 1), (n >> 1));
        }
    }
        //在下半部
    else {
        //左下
        if (y1 - y2 < (n >> 1)) {
            cout << (x2 + (n >> 1) - 1) << ' ' << (y2 + (n >> 1))
            ' ' << 3 << endl;
            solve(x2 + (n >> 1) - 1, y2 + (n >> 1) - 1,
                  x2, y2, (n >> 1));
            solve(x2 + (n >> 1) - 1, y2 + (n >> 1),
                  x2, y2 + (n >> 1), (n >> 1));

            solve(x1, y1, x2 + (n >> 1), y2, (n >> 1));
            solve(x2 + (n >> 1), y2 + (n >> 1),
                  x2 + (n >> 1), y2 + (n >> 1), (n >> 1));
        }
            //右下
        else {
            cout << (x2 + (n >> 1) - 1) << ' ' << (y2 + (n >> 1) -
            ' ' << 4 << endl;
            solve(x2 + (n >> 1) - 1, y2 + (n >> 1) - 1,
                  x2, y2, (n >> 1));
            solve(x2 + (n >> 1) - 1, y2 + (n >> 1),
                  x2, y2 + (n >> 1), (n >> 1));
            solve(x2 + (n >> 1), y2 + (n >> 1) - 1,
                  x2 + (n >> 1), y2, (n >> 1));

            solve(x1, y1, x2 + (n >> 1), y2 + (n >> 1), (n >> 1));
        }
    }
}

int main() {
    int k, x, y;
    cin >> k >> x >> y;
    solve(x, y, 1, 1, 1 << k);
    return 0;
}
```


1) <<