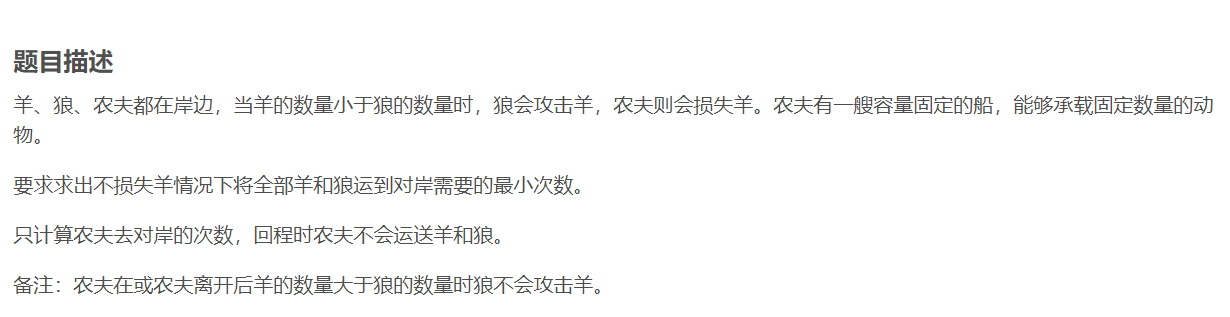
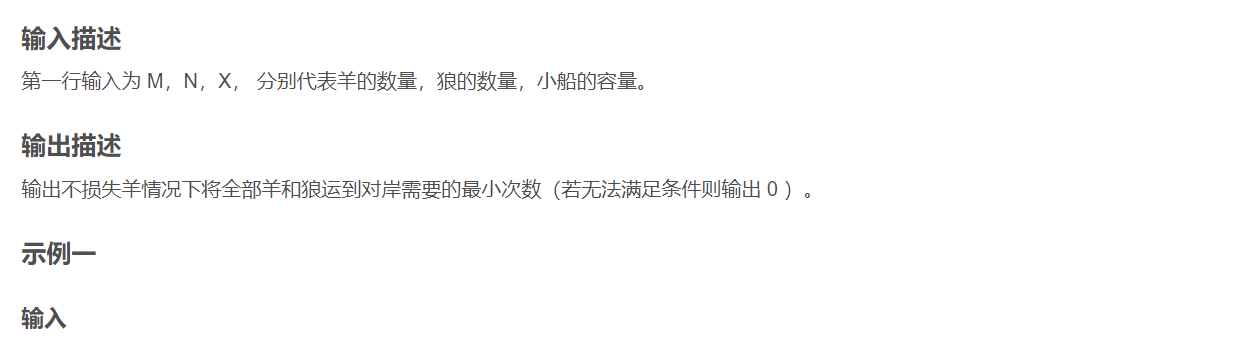
# **E卷-羊、狼、农夫过河[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**

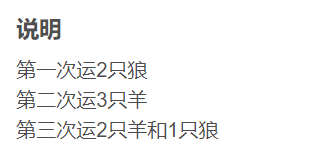




5 3 3

输出

3



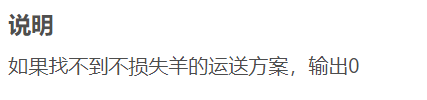
示例1

输入

541

输出

0





import java.util.Scanner;

public class Main {

// 表示极大值的常量，用于初始化最小操作次数

private static final long INF = (long) 1e18;

// 全局变量

private static int m, n, x;

private static int originM, originN;

private static long minn = INF; // 表示最小操作次数的变量，初始化为极大值

// 深度优先搜索函数，用来寻找最小操作次数

private static void dfs(int nowM, int nowN, int ci) {

// 如果当前羊和狼的数量之和小于等于 x，则更新最小操作次数并返回

if (nowM + nowN <= x) {

minn = Math.min(minn, ci + 1);

return;

}

// 遍历所有可能取出 i 只羊和 j 只狼的组合

for (int i = 0; i <= nowM; i++) {

for (int j = 0; j <= nowN; j++) {

// 跳过一些无效的组合（如同时取出 0 或者取出的总量大于 x）

if (i + j == 0 || i + j > x || (i != nowM && nowM - i <= nowN - j)) continue;

// 跳过一些无效的组合（如新组合和原始数量差值不符合约束条件）

if (originM != nowM - i && originM - nowM + i <= originN - nowN + j) continue;

// 递归调用 dfs 函数

dfs(nowM - i, nowN - j, ci + 1);

}

}

}

public static void main(String[] args) {

// 从标准输入读取 m, n, x 三个整数

Scanner scanner = new Scanner(System.in);

m = scanner.nextInt();

n = scanner.nextInt();

x = scanner.nextInt();

// 如果 m 和 n 的和小于等于 x，则立即打印结果 1 并退出

if (m + n <= x) {

System.out.println(1);

return;

}

// 保存初始的 m 和 n 值

originM = m;

originN = n;

// 初始调用 dfs 函数来开始深度优先搜索

dfs(m, n, 0);

// 输出结果，根据 minn 的值判断是否找到解

if (minn == INF) {

System.out.println(0);

} else {

System.out.println(minn);

}

}

}



# 从标准输入读取 m, n, x 三个整数

m, n, x = map(int, input().split())

# 如果 m 和 n 的和小于等于 x，则立即打印结果 1 并退出

if m + n <= x:

print(1)

exit(0)

# 保存初始的 m 和 n 值

originM = m

originN = n

minn = int(1e18) # 用一个极大值来表示最小操作次数的初始化值

# 定义深度优先搜索函数来寻找最小操作次数

def dfs(nowM, nowN, ci):

global minn

# 如果当前羊和狼的和小于等于 x，则更新最小操作次数并返回

if nowM + nowN <= x:

minn = min(minn, ci + 1)

return

# 遍历所有可能的取出 i只羊和 j 只狼的组合

for i in range(nowM + 1):

for j in range(nowN + 1):

# 跳过一些无效的组合

if i + j == 0 or i + j > x or (i != nowM and nowM - i <= nowN - j):

continue

# 跳过一些无效的组合

if originM != nowM - i and originM - nowM + i <= originN - nowN + j:

continue

# 递归调用 dfs 函数

dfs(nowM - i, nowN - j, ci + 1)

# 初始调用 dfs 函数来开始深度优先搜索

dfs(m, n, 0)

# 输出结果，根据 minn 的值判断是否找到解

if minn == int(1e18):

print(0)

else:

print(minn)



#include <iostream>

#include <algorithm>

using namespace std;

// 定义极大值常量，用来初始化最小操作次数

const long long INF = 1e18;

// 全局变量

int m, n, x;

int originM, originN;

long long minn =

INF; // 表示最小操作次数的变量，初始化为极大值

// 深度优先搜索函数，用来寻找最小操作次数

void dfs(int nowM, int nowN, int ci) {

// 如果当前羊和狼的数量之和小于等于 x，则更新最小操作次数并返回

if (nowM + nowN <= x) {

minn = min(minn, static\_cast<long long>(ci + 1));

return;

}

// 遍历所有可能取出 i 只羊和 j 只狼的组合

for (int i = 0; i <= nowM; ++i) {

for (int j = 0; j <= nowN; ++j) {

// 跳过一些无效的组合（如同时取出 0 或者取出的总量大于 x）

if (i + j == 0 || i + j > x || (i != nowM && nowM - i <= nowN - j)) continue;

// 跳过一些无效的组合（如新组合和原始数量差值不符合约束条件）

if (originM != nowM - i && originM - nowM + i <= originN - nowN + j) continue;

// 递归调用 dfs 函数

dfs(nowM - i, nowN - j, ci + 1);

}

}

}

int main() {

// 从标准输入读取 m, n, x 三个整数

cin >> m >> n >> x;

// 如果 m 和 n 的和小于等于 x，则立即打印结果 1 并退出

if (m + n <= x) {

cout << 1 << endl;

return 0;

}

// 保存初始的 m 和 n 值

originM = m;

originN = n;

// 初始调用 dfs 函数来开始深度优先搜索

dfs(m, n, 0);

// 输出结果，根据 minn 的值判断是否找到解

if (minn == INF) {

cout << 0 << endl;

} else {

cout << minn << endl;

}

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

// 全局变量声明

int m, n, x, originM, originN;

long long minn =

LLONG\_MAX; // 用极大值表示最小操作次数的初始化值

// 深度优先搜索函数，用于寻找最小操作次数

void dfs(int nowM, int nowN, int ci) {

// 如果当前羊和狼的和小于等于 x，则更新最小操作次数并返回

if (nowM + nowN <= x) {

if (ci + 1 < minn) {

minn = ci + 1;

}

return;

}

// 遍历所有可能的取出 i 只羊和 j 只狼的组合

for (int i = 0; i <= nowM; i++) {

for (int j = 0; j <= nowN; j++) {

// 跳过一些无效的组合

if (i + j == 0 || i + j > x || (i != nowM && nowM - i <= nowN - j)) {

continue;

}

// 跳过一些无效的组合

if (originM != nowM - i && originM - nowM + i <= originN - nowN + j) {

continue;

}

// 递归调用 dfs 函数

dfs(nowM - i, nowN - j, ci + 1);

}

}

}

int main() {

// 从标准输入读取 m, n, x 三个整数

scanf("%d %d %d", &m, &n, &x);

// 如果 m 和 n 的和小于等于 x，则立即打印结果 1 并退出

if (m + n <= x) {

printf("1\n");

return 0;

}

// 保存初始的 m 和 n 值

originM = m;

originN = n;

// 初始调用 dfs 函数来开始深度优先搜索

dfs(m, n, 0);

// 输出结果，根据 minn 的值判断是否找到解

if (minn == LLONG\_MAX) {

printf("0\n");

} else {

printf("%lld\n", minn);

}

return 0;

}



const readline = require('readline');

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

// 定义极大值常量，用于初始化最小操作次数

const INF = 1e18;

// 全局变量

let m, n, x;

let originM, originN;

let minn = INF; // 表示最小操作次数的变量，初始化为极大值

// 深度优先搜索函数，用来寻找最小操作次数

function dfs(nowM, nowN, ci) {

// 如果当前羊和狼的数量之和小于等于 x，则更新最小操作次数并返回

if (nowM + nowN <= x) {

minn = Math.min(minn, ci + 1);

return;

}

// 遍历所有可能取出 i 只羊和 j 只狼的组合

for (let i = 0; i <= nowM; i++) {

for (let j = 0; j <= nowN; j++) {

// 跳过一些无效的组合（如同时取出 0 或者取出的总量大于 x）

if (i + j === 0 || i + j > x || (i !== nowM && nowM - i <= nowN - j)) continue;

// 跳过一些无效的组合（如新组合和原始数量差值不符合约束条件）

if (originM !== nowM - i && originM - nowM + i <= originN - nowN + j) continue;

// 递归调用 dfs 函数

dfs(nowM - i, nowN - j, ci + 1);

}

}

}

// 从标准输入读取 m, n, x 三个整数

rl.on('line', (input) => {

// 将输入按空格分割并转换为整数

[m, n, x] = input.split(' ').map(Number);

// 如果 m 和 n 的和小于等于 x，则立即打印结果 1 并退出

if (m + n <= x) {

console.log(1);

rl.close();

return;

}

// 保存初始的 m 和 n 值

originM = m;

originN = n;

// 初始调用 dfs 函数来开始深度优先搜索

dfs(m, n, 0);

// 输出结果，根据 minn 的值判断是否找到解

if (minn === INF) {

console.log(0);

} else {

console.log(minn);

}

// 关闭输入接口

rl.close();

});



package main

import (

"bufio"

"fmt"

"math"

"os"

"strconv"

"strings"

)

var (

m, n, x int

originM, originN int

minn int64 = math.MaxInt64 // 用一个极大值来表示最小操作次数的初始化值

)

// 深度优先搜索函数来寻找最小操作次数

func dfs(nowM, nowN, ci int) {

// 如果当前羊和狼的和小于等于 x，则更新最小操作次数并返回

if nowM+nowN <= x {

if int64(ci+1) < minn {

minn = int64(ci + 1)

}

return

}

// 遍历所有可能的取出 i 只羊和 j 只狼的组合

for i := 0; i <= nowM; i++ {

for j := 0; j <= nowN; j++ {

// 跳过一些无效的组合

if i+j == 0 || i+j > x || (i != nowM && nowM-i <= nowN-j) {

continue

}

// 跳过一些无效的组合

if originM != nowM-i && originM-nowM+i <= originN-nowN+j {

continue

}

// 递归调用 dfs 函数

dfs(nowM-i, nowN-j, ci+1)

}

}

}

func main() {

// 从标准输入读取 m, n, x 三个整数

reader := bufio.NewReader(os.Stdin)

input, \_ := reader.ReadString('\n')

input = strings.TrimSpace(input)

values := strings.Split(input, " ")

m, \_ = strconv.Atoi(values[0])

n, \_ = strconv.Atoi(values[1])

x, \_ = strconv.Atoi(values[2])

// 如果 m 和 n 的和小于等于 x，则立即打印结果 1 并退出

if m+n <= x {

fmt.Println(1)

return

}

// 保存初始的 m 和 n 值

originM = m

originN = n

// 初始调用 dfs 函数来开始深度优先搜索

dfs(m, n, 0)

// 输出结果，根据 minn 的值判断是否找到解

if minn == math.MaxInt64 {

fmt.Println(0)

} else {

fmt.Println(minn)

}

}