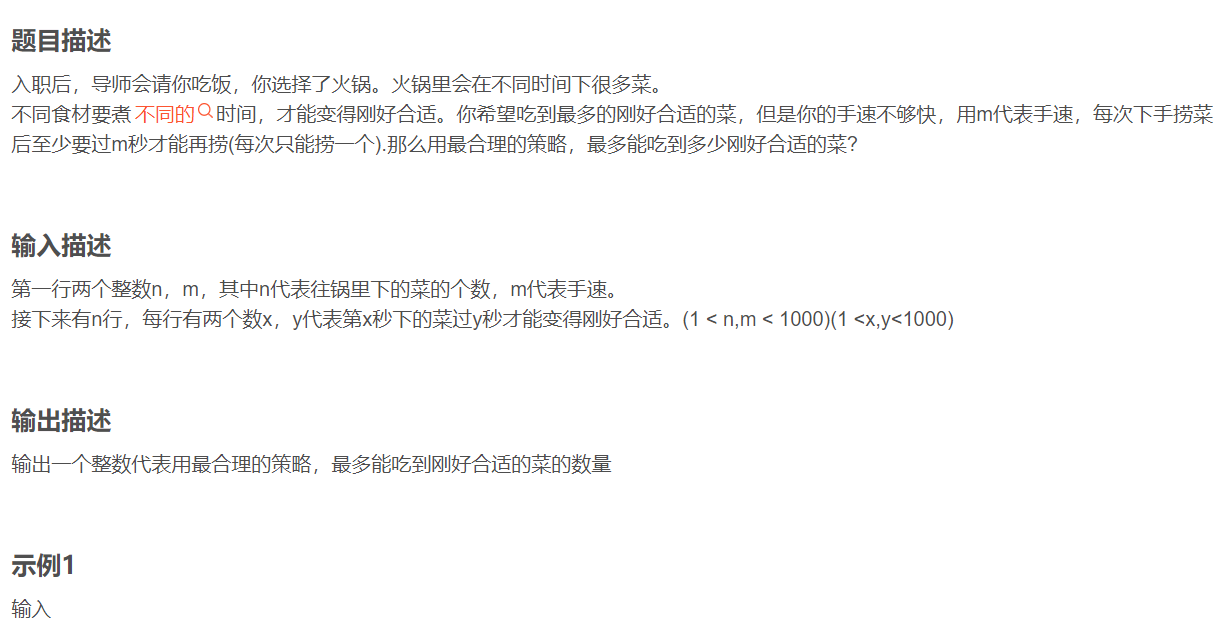
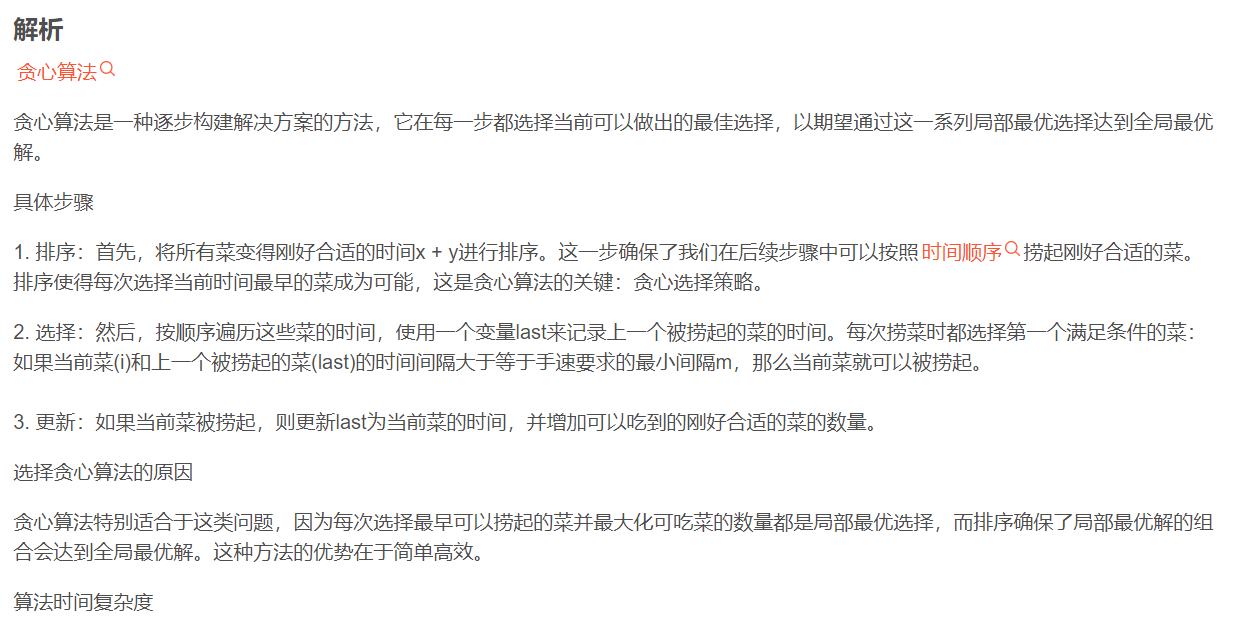
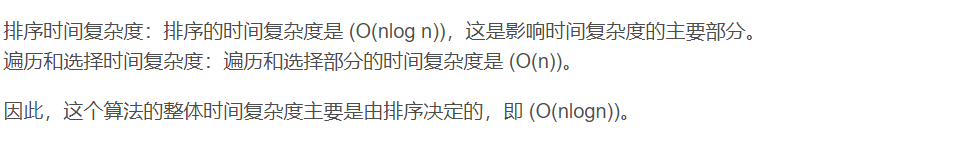
# **E卷-导师请吃火锅[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go ）**











import java.util.Scanner;

import java.util.Arrays;

public class Main {

public static void main(String[] args) {

// 创建用于输入的扫描器对象

Scanner sc = new Scanner(System.in);

// 读取输入的两个整数n和m

int n = sc.nextInt();

int m = sc.nextInt();

// 创建一个数组来存储每个菜刚好合适的时间

int[] a = new int[n];

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

int x = sc.nextInt(); // 读取菜的下锅时间

int y = sc.nextInt(); // 读取菜的变得刚好合适的时间

a[i] = x + y; // 计算菜刚好合适的时间

}

// 将所有菜刚好合适的时间排序

Arrays.sort(a);

// 初始化可以吃到刚好合适的菜的数量

int res = 1;

// 初始化上一次捞菜的时间为第一个菜刚好合适的时间

int last = a[0];

// 遍历剩余的菜

for (int i = 1; i < n; i++) {

// 检查当前菜的时间与上次捞菜之间的间隔是否大于等于手速要求的最小间隔

if (a[i] - last >= m) {

res++; // 可以再捞一个菜，计数增加

last = a[i]; // 更新上次捞菜的时间

}

}

// 输出最多能吃到刚好合适的菜的数量

System.out.println(res);

}

}



# 读取输入的两个整数n和m

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

# 创建一个列表来存储每个菜刚好合适的时间

a = []

for \_ in range(n):

x, y = map(int, input().split()) # 读取每个菜的下锅时间x和适合时间y

a.append(x + y) # 计算每个菜变得刚好合适的时间，并加入列表

# 将所有菜刚好合适的时间排序

a.sort()

# 初始化可以吃到刚好合适的菜的数量为1（至少可以吃到第一个菜）

res = 1

# 初始化上一个被捞起来的菜的时间为第一个菜刚好合适的时间

last = a[0]

# 遍历剩余的菜

for i in a[1:]:

if i - last >= m: # 如果当前菜刚好合适的时间与上次捞菜之间的间隔大于等于手速要求的最小间隔

res += 1 # 可以再捞一个菜

last = i # 更新上次捞菜的时间为当前菜刚好合适的时间

# 输出最多能吃到刚好合适的菜的数量

print(res)



#include <iostream>

#include <algorithm> // 用于排序

using namespace std;

int main() {

int n, m;

// 读取输入的两个整数n和m

cin >> n >> m;

// 创建一个数组来存储每个菜刚好合适的时间

int a[n];

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

int x, y;

// 读取菜的下锅时间和变得刚好合适的时间

cin >> x >> y;

// 计算菜刚好合适的时间并存储在数组中

a[i] = x + y;

}

// 将所有菜刚好合适的时间排序

sort(a, a + n);

// 初始化可以吃到刚好合适的菜的数量

int res = 1;

// 初始化上一次捞菜的时间为第一个菜刚好合适的时间

int last = a[0];

// 遍历剩余的菜

for (int i = 1; i < n; i++) {

// 检查当前菜的时间与上次捞菜之间的间隔是否大于等于手速要求的最小间隔

if (a[i] - last >= m) {

res++; // 可以再捞一个菜，计数增加

last = a[i]; // 更新上次捞菜的时间

}

}

// 输出最多能吃到刚好合适的菜的数量

cout << res << endl;

return 0;

}



#include <stdio.h>

#include <stdlib.h>

int compare(const void\* a, const void\* b) {

return (\*(int\*)a - \* (int\*)b); // 比较函数，用于qsort排序

}

int main() {

int n, m;

// 读取输入的两个整数n和m

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

// 创建一个数组来存储每个菜刚好合适的时间

int a[n];

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

int x, y;

// 读取菜的下锅时间和变得刚好合适的时间

scanf("%d %d", &x, &y);

// 计算菜刚好合适的时间并存储在数组中

a[i] = x + y;

}

// 将所有菜刚好合适的时间排序

qsort(a, n, sizeof(int), compare);

// 初始化可以吃到刚好合适的菜的数量

int res = 1;

// 初始化上一次捞菜的时间为第一个菜刚好合适的时间

int last = a[0];

// 遍历剩余的菜

for (int i = 1; i < n; i++) {

// 检查当前菜的时间与上次捞菜之间的间隔是否大于等于手速要求的最小间隔

if (a[i] - last >= m) {

res++; // 可以再捞一个菜，计数增加

last = a[i]; // 更新上次捞菜的时间

}

}

// 输出最多能吃到刚好合适的菜的数量

printf("%d\n", res);

return 0;

}



const readline = require("readline");

// 创建接口以便读取输入

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

let input = [];

// 读取输入的行

rl.on("line", function (line) {

input.push(line.trim());

});

rl.on("close", function () {

// 读取输入的两个整数n和m

let [n, m] = input[0].split(" ").map(Number);

// 创建一个数组来存储每个菜刚好合适的时间

let a = [];

for (let i = 1; i <= n; i++) {

let [x, y] = input[i].split(" ").map(Number); // 读取菜的下锅时间和变得刚好合适的时间

a.push(x + y); // 计算菜刚好合适的时间并加入数组

}

// 将所有菜刚好合适的时间排序

a.sort((a, b) => a - b);

// 初始化可以吃到刚好合适的菜的数量

let res = 1;

// 初始化上一次捞菜的时间为第一个菜刚好合适的时间

let last = a[0];

// 遍历剩余的菜

for (let i = 1; i < n; i++) {

// 检查当前菜的时间与上次捞菜之间的间隔是否大于等于手速要求的最小间隔

if (a[i] - last >= m) {

res++; // 可以再捞一个菜，计数增加

last = a[i]; // 更新上次捞菜的时间

}

}

// 输出最多能吃到刚好合适的菜的数量

console.log(res);

});



package main

import (

"bufio"

"fmt"

"os"

"sort"

"strconv"

"strings"

)

func main() {

// 创建一个读取器，从标准输入读取数据

reader := bufio.NewReader(os.Stdin)

// 读取第一行输入

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

line = strings.TrimSpace(line)

parts := strings.Split(line, " ")

// 读取n和m

n, \_ := strconv.Atoi(parts[0])

m, \_ := strconv.Atoi(parts[1])

// 创建一个切片来存储每个菜刚好合适的时间

a := make([]int, n)

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

// 读取每个菜的下锅时间和变得刚好合适的时间

line, \_ = reader.ReadString('\n')

line = strings.TrimSpace(line)

parts = strings.Split(line, " ")

x, \_ := strconv.Atoi(parts[0])

y, \_ := strconv.Atoi(parts[1])

// 计算菜刚好合适的时间并存储在切片中

a[i] = x + y

}

// 将所有菜刚好合适的时间排序

sort.Ints(a)

// 初始化可以吃到刚好合适的菜的数量

res := 1

// 初始化上一次捞菜的时间为第一个菜刚好合适的时间

last := a[0]

// 遍历剩余的菜

for i := 1; i < n; i++ {

// 检查当前菜的时间与上次捞菜之间的间隔是否大于等于手速要求的最小间隔

if a[i]-last >= m {

res++ // 可以再捞一个菜，计数增加

last = a[i] // 更新上次捞菜的时间

}

}

// 输出最多能吃到刚好合适的菜的数量

fmt.Println(res)

}