// 排队想到队列，这里用数组模拟队列，维护所有信息

// 假设银行有K个窗口提供服务，窗口前设一条黄线，所有顾客按到达时间在黄线后排成一条长龙。

// 当有窗口空闲时，下一位顾客即去该窗口处理事务。当有多个窗口可选择时，假设顾客总是选择编号最小的窗口。

// 。当有多个窗口可选择时，假设顾客总是选择编号最小的窗口。

// // 状态2

// 本题要求输出前来等待服务的N位顾客的平均等待时间、最长等待时间、最后完成时间，并且统计每个窗口服务了多少名顾客。

// // 求平均等待，最长等待，最后完成的时间，每个窗口服务的顾客数量

// // 输入第1行给出正整数N（≤1000）

// // 暴力n方

// // 输入第1行给出正整数N（≤1000），为顾客总人数；随后N行，每行给出一位顾客的到达时间T和事务处理时间P

// // 这里假设每位顾客事务被处理的最长时间为60分钟。

// // 被服务到时，当前时间减来到的时间就是等待时间

// // 要储存窗口状态,服务的顾客,数量

// // 要枚举时间的话不太现实,应该枚举订单,已经过去的时间,

// // 遍历窗口状态,已过时间是当前枚举的顾客来到的时间,此时遍历窗口,有空则进,保存状态,没则等待时间加最小值,比较最长等待,加入总等待

// // 窗口保存当前时间加当前顾客服务时间

// 卡壳,不知道忽略了哪里,忽略了关键点，被题目混淆了。 pta题真毒

**// 这里假设每位顾客事务被处理的最长时间为60分钟。又是一道歧义描述题**

// 有可能是超过60要重新排队，有可能是超过60看作是60，有可能是数据小于60（搁这考我语文是吧）

// 题目数据有61，说明这肯定不是指t <= 60,

// 剩下的就凭感觉了，题目如果有说明重新排队的流程的话，就是重新排队，没有就是大于60看作6

**有歧义的题，把可能的情况列出来，由简单到困难一个个试**

#include <iostream>

#include <cstring>

#include <cstdio>

#include <algorithm>

using namespace std;

const int N = 1010;

int window[N],come[N],t[N];

int cnt[N];

int nowt;

int main(){

int n;

cin >> n;

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

cin >> come[i] >> t[i];

if(t[i] > 60) t[i] = 60;

}

int k;

cin >>k;

int maxwait = 0;

int totalwait = 0;

int finaltime = 0;

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

nowt = come[i];

int mint = 0x3f3f3f3f;

int minj;

bool flag = false;

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

if(nowt >= window[j] && !flag){

window[j] = nowt + t[i];

cnt[j]++;

flag = true;

}

}

if(!flag){

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

if(mint > window[j]){

minj = j;

mint = window[j];

}

}

int waittime = window[minj] - nowt;

totalwait += waittime;

maxwait = max(maxwait,waittime);

window[minj] += t[i];

cnt[minj]++;

}

}

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

finaltime = max(finaltime,window[i]);

}

printf("%.1lf %d %d\n",totalwait\*1.0/n\*1.0,maxwait,finaltime);

cout << cnt[0];

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

cout<<" "<< cnt[i];

}

return 0;

}

0

二刷

// 假设银行有K个窗口提供服务，

// 当有多个窗口可选择时，假设顾客总是选择编号最小的窗口

// N位顾客的平均等待时间、最长等待时间、最后完成时间，并且统计每个窗口服务了多少名顾客。

// ，每行给出一位顾客的到达时间T和事务处理时间P，并且假设输入数据已经按到达时间先后排好了顺序

// 这里假设每位顾客事务被处理的最长时间为60分钟

// 又忘记先看数据范围。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

// 范围1000，10，可以放开手脚写

// 求题目的时间，可以枚举时间，可以枚举顾客二元组，枚举时间的话，每过一秒都需要遍历一遍顾客，而且最大时间是不知道的，还要遍历一遍顾客来找

// 所以选择遍历顾客二元组、

// 再来看题目要求的东西怎么求，平均等待时间是要先求出总等待，

// 等待时间需要知道窗口状态，用一个数组记录窗口空闲的时间，一个数组记录窗口服务人数，还要一个变量保存最大等待时间

// 遍历窗口如果顾客来到的时间小于任何一个窗口空闲时刻的话，就找出里面最小的时刻，

// 用这个时刻减去顾客来到的时间就是等待时间了，这个时刻对应的窗口人数还要加一每次算出后要加入总等待，

// 还要和最长等待作比较。

// 如果有窗口的空闲时刻小于顾客到来时刻，顾客就去第一个遍历到的该窗口，

// 窗口服务人数加1，空闲时刻改为顾客到来时刻加服务时间

// 所有顾客遍历完之后，遍历窗口，找出里面的最大值就是最后完成时间

// 注意点这里假设每位顾客事务被处理的最长时间为60分钟。用例中有61，所以大于60变成60，或者记录时间重新排队

#include <iostream>

#include <cstring>

#include <algorithm>

#include <cstdio>

using namespace std;

const int N = 1e3 + 10;

int window\_free\_time[N],window\_conut[N];

int customer\_come[N],customer\_service[N];

int main(){

int customer\_num,window\_num;

int max\_wait = 0,total\_wait = 0,final\_time = 0;

cin >> customer\_num;

for(int i = 0;i < customer\_num;i++){

cin >> customer\_come[i] >> customer\_service[i];

if(customer\_service[i] > 60) customer\_service[i] = 60;

}

cin >> window\_num;

for(int i = 0;i < customer\_num;i++){

bool flag = false;

for(int j = 0;j < window\_num;j++){

if(customer\_come[i] >= window\_free\_time[j]){

flag = true;

window\_free\_time[j] = customer\_come[i] + customer\_service[i];

window\_conut[j]++;

// 卡壳，看到窗口服务的总人数不对劲才碰巧找到了这里的bug，找到了可以服务的窗口就应该立刻break，

// 否则会把能服务的窗口全都找个遍，代码细节把窝不到位，该break的地方就该break

break;

}

}

if(!flag){

int minn = 0x3f3f3f3f;

int minj = 0;

for(int j = 0;j < window\_num;j++){

if(minn > window\_free\_time[j]){

minn = window\_free\_time[j];

minj = j;

}

}

int wait\_time = window\_free\_time[minj] - customer\_come[i];

total\_wait += wait\_time;

max\_wait = max(max\_wait,wait\_time);

window\_free\_time[minj] += customer\_service[i];

window\_conut[minj]++;

}

}

for(int i = 0;i < window\_num;i++){

final\_time = max(final\_time,window\_free\_time[i]);

}

printf("%.1lf %d %d\n",total\_wait\*1.0/customer\_num\*1.0,max\_wait,final\_time);

cout << window\_conut[0];

for(int i = 1;i < window\_num;i++){

cout <<" " << window\_conut[i];

}

return 0;

}