1 Introduction

Before we may proceed to the solution of the problem we shall make some observations about the properties of the problem. Firstly, any conclusion we can make on the number of pilgrims in certain moment must be based on the fact that Jack never had to deal with fractional numbers of euros. This lead us to a claim:

Claim 1

Suppose that part of our input looks like that: **IN**/**OUT** a_1 **PAY** k_1 **PAY** k_2 ... **PAY** k_n **IN**/**OUT** a_2 and after a_1 pilgrims joined (or left) the trip there could have been exactly x people in the group. Then we have that x is a divisor of $k_1 + k_2 + \ldots + k_n$. Moreover, any x of such property can be number of people in the group after line **PAY** k_n.

Indeed, if it was not so when a_2 people join or leave the group Jack would have to deal with fractional numbers of euros, which is guaranteed to have never occured.

Observation 2

Information given in **COLLECT** is irrelevant. Thus any input have the same solutions as the same input with all occurences of **COLLECT** k lines removed.

Indeed, since we are given only one page of booklet, Jack could have collected enough money just before the beginning of page.

Observation 3

No information can be gained from the initial sequence of **PAY** lines.

Assume input is a sequence $i_1, i_2, ..., i_n$ such that $i_1, ..., i_k$ are **PAY** lines, i_{k+1} is **IN** or **OUT** line and i_j is not **COLLECT** line for any $1 \leq j \leq n$. Furthermore, assume x is the solution for input $i_{k+1}, i_{k+2}, \ldots, i_n$. Then x is also a solution for input $i_1, i_2, ..., i_n$ for the reason being that the previous page of booklet could have ended with a sequence $j_1, j_2, \ldots, j_m$ of **PAY** lines such that x is a divisor of the sum of money that Jack possessed.

Observation 4

If input ends with a sequence of **PAY** lines we may ignore them.

Indeed, whatever amount of money was paid at the end Jack did not have to deal with fractional number of euros since no one came in nor came out after.

Observation 5

If input consists of no **PAY** lines (thus only **IN** and **OUT** lines) and $\Delta p_1, \Delta p_2, \ldots, \Delta p_n$ is a sequence of changes of number of people in the group (Δp_i corresponds to i-th line: **IN** Δp_i or **OUT** $-\Delta p_i$) then any number greater or equal to $min_k - (\Delta p_1 + \Delta p_2 + \ldots + \Delta p_k)$ can be the number of pilgrims at the beginning of the page.

Having considered all the observations and the claim we may propose a scheme of the problem solution:

1. Remove all **COLLECT** lines.
2. Remove the initial and ending sequence of **PAY** lines (if they occur).
3. If (after 2 preprocessing) there are no **PAY** lines compute the minimal number of pilgrims basing on observation 5.

4. In the other case we have input consisting of interlacing sequences of **IN/OUT** and **PAY** lines.

5. Take the first sequence of **PAY** lines, compute its sum S of values paid.

6. For every divisor of S check whether it can be a number of people in the remaining input sequence.

We need to remember to compute all the divisors of S in no longer than $O(\sqrt{S})$ time.

The last dragon left to slay is checking whether given number m can be number of people in the remaining input sequence. This can be done by single loop over the sequence. We will remember the number of people if at the beginning there were m pilgrims. This value will need to be updated after each **IN/OUT** line. For each block of **PAY** lines we shall compute its sum and if it is divisible by the number of people m is a solution and it is not in the other case.

2 Model solution

Model solution (file *pil.{cpp/java}*)is an implementation of the above algorithm. Firstly input is filtered respectively to points $1)$ and $2)$. Later on, the call of *no_pays()* fuction handles the case when input consists no **PAY** lines. Then the main program fuction is called. It sums the values of the first remaining block of **PAY** lines (call this value sum), and for each pair of its divisors j and sum/j it adds them to result if they are possible numbers of people in the group. This is done by calling the function *possible()*, having ensured that the considered value is greater or equal to $1+$ adjustment respective to number of people joined/left in preceeding **IN/OUT** lines. The function *possible()* implements the idea described at the end of the previous section.

Let N be the size of input. Time complexity of the model solution is $O(N\sqrt{2000N})$ and memory complexity is $O(N)$.

3 Slow solution 1

Slow solution 1 (*pils1.{cpp/java}*) is very similar to the model solution. The only difference is that it computes all the divisors of S by simple looping over all numbers lesser than S, which takes significantly longer time. Let N be the size of input. Time complexity of the model solution is $O(2000N^2)$ and memory complexity is $O(N)$

4 Tests

Test cases consists of files:

- *pil0.in* — example test,
- *pil1.in* — simple correctness tests,

- *pil2.in* — more sophisticated correctness tests.
- *pil3.in* — effectiveness tests consisting of short (1-3 element) **IN/OUT** initial and ending blocks and a large (40-48 element) block of **PAY**s with respectively large amounts of money (1850-2000). Thus sum of money is sufficiently large to make slow solution run significantly longer.
- *pil4.in* — random effectiveness tests. A test case consists of interlacing blocks of **IN/OUT** (that are usually short) lines and **PAY** lines. First significant block of **PAY**s is quite long so that the number whose all divisors need to be computed grows on average to $50,000$-$70,000$.

The first two files were written manually, others are created automatically by program *pilingen.cpp* for the reason being that effectiveness tests should consist of around $10,000$ cases, wich makes s total of $400,000$-$500,000$ lines. Since **COLLECT** lines are irrelevant for the problem solution only a few of them are put randomly in last two test files.