Yandex.Algorithm 2011
Round 1

A. Domino

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

We all know the problem about the number of ways one can tile a $2 \times n$ field by 1×2 dominoes. You probably remember that it goes down to Fibonacci numbers. We will talk about some other problem below, there you also are going to deal with tiling a rectangular field with dominoes.

You are given a $4 \times n$ rectangular field, that is the field that contains four lines and n columns. You have to find for it any tiling by 1×2 dominoes such that each of the n - 1 potential vertical cuts along the grid lines intersects at least one domino, splitting it in two. No two dominoes in the sought tiling should overlap, each square of the field should be covered by exactly one domino. It is allowed to rotate the dominoes, that is, you can use 2×1 as well as 1×2 dominoes.

Write a program that finds an arbitrary sought tiling.

Input

The input contains one positive integer n ($1 \le n \le 100$) — the number of the field's columns.

Output

If there's no solution, print "-1" (without the quotes). Otherwise, print four lines containing n characters each — that's the description of tiling, where each vertical cut intersects at least one domino. You should print the tiling, having painted the field in no more than 26 colors. Each domino should be painted a color. Different dominoes can be painted the same color, but dominoes of the same color should not be side-neighbouring. To indicate colors you should use lowercase Latin letters. Print any of the acceptable ways of tiling.

Examples

input
4

output
yyzz bccd bxxd yyaa

B. Embassy Queue

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

In an embassy of a well-known kingdom an electronic queue is organised. Every person who comes to the embassy, needs to make the following three actions: show the ID, pay money to the cashier and be fingerprinted. **Besides, the actions should be performed in the given order.**

For each action several separate windows are singled out: k_1 separate windows for the first action (the first type windows), k_2 windows for the second one (the second type windows), and k_3 for the third one (the third type windows). The service time for one person in any of the first type window equals to t_1. Similarly, it takes t_2 time to serve a person in any of the second type windows. And it takes t_3 to serve one person in any of the third type windows. Thus, the service time depends only on the window type and is independent from the person who is applying for visa.

At some moment n people come to the embassy, the i-th person comes at the moment of time c_i. The person is registered under some number. After that he sits in the hall and waits for his number to be shown on a special board. Besides the person's number the board shows the number of the window where one should go and the person goes there immediately. Let's consider that the time needed to approach the window is negligible. The table can show information for no more than one person at a time. The electronic queue works so as to immediately start working with the person who has approached the window, as there are no other people in front of the window.

The Client Service Quality inspectors noticed that several people spend too much time in the embassy (this is particularly tiresome as the embassy has no mobile phone reception and 3G). It was decided to organise the system so that the largest time a person spends in the embassy were minimum. Help the inspectors organise the queue. Consider that all actions except for being served in at the window, happen instantly.

Input

The first line contains three space-separated integers k_1, k_2, k_3 ($1 \le k_i \le 10^9$), they are the number of windows of the first, second and third type correspondingly.

The second line contains three space-separated integers t_1, t_2, t_3 ($1 \le t_i \le 10^5$), they are the periods of time needed to serve one person in the window of the first, second and third type correspondingly.

The third line contains an integer n ($1 \le n \le 10^5$), it is the number of people.

The fourth line contains n space-separated integers c_i ($1 \le c_i \le 10^9$) in the non-decreasing order; c_i is the time when the person number i comes to the embassy.

Output

Print the single number, the maximum time a person will spend in the embassy if the queue is organized optimally.

Please, do not use the `%lld` specificator to read or write 64-bit integers in C++. It is preferred to use the `cin`, `cout` streams (also you may use the `%I64d` specificator).

Examples

input
1 1 1 1 1 1 5 1 1 1 1 1
output
7

input
2 1 1 5 1 1 5 1 2 3 3 5
output
13

Note

In the first test 5 people come simultaneously at the moment of time equal to 1. There is one window of every type, it takes 1 unit of time to be served at each window. That's why the maximal time a person spends in the embassy is the time needed to be served at the windows (3 units of time) plus the time the last person who comes to the first window waits (4 units of time).

Windows in the second test work like this:

The first window of the first type: $[1, 6)$ — the first person, $[6, 11)$ — third person, $[11, 16)$ — fifth person

The second window of the first type: $[2, 7)$ — the second person, **$[7, 12)$** — the fourth person

The only second type window: $[6, 7)$ — first, $[7, 8)$ — second, $[11, 12)$ — third, $[12, 13)$ — fourth, $[16, 17)$ — fifth

The only third type window: $[7, 8)$ — first, $[8, 9)$ — second, $[12, 13)$ — third, $[13, 14)$ — fourth, $[17, 18)$ — fifth

We can see that it takes most time to serve the fifth person.

C. Petya and Tree

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

One night, having had a hard day at work, Petya saw a nightmare. There was a binary search tree in the dream. But it was not the actual tree that scared Petya. The horrifying thing was that Petya couldn't search for elements in this tree. Petya tried many times to choose key and look for it in the tree, and each time he arrived at a wrong place. Petya has been racking his brains for long, choosing keys many times, but the result was no better. But the moment before Petya would start to despair, he had an epiphany: every time he was looking for keys, the tree didn't have the key, and occured exactly one mistake. "That's not a problem!", thought Petya. "Why not count the expectation value of an element, which is found when I search for the key". The moment he was about to do just that, however, Petya suddenly woke up.

Thus, you are given a *binary search tree*, that is a tree containing some number written in the node. This number is called the *node key*. The number of children of every node of the tree is equal either to 0 or to 2. The nodes that have 0 children are called *leaves* and the nodes that have 2 children, are called *inner*. An inner node has the *left child*, that is the child whose key is less than the current node's key, and the *right child*, whose key is more than the current node's key. Also, a key of any node is strictly larger than all the keys of the left subtree of the node and strictly smaller than all the keys of the right subtree of the node.

Also you are given a set of *search keys*, all of which are distinct and differ from the node keys contained in the tree. For each key from the set its search in the tree is realised. The search is arranged like this: initially we are located in the tree root, if the key of the current node is larger that our search key, then we move to the left child of the node, otherwise we go to the right child of the node and the process is repeated. As it is guaranteed that the search key is not contained in the tree, the search will always finish in some leaf. The key lying in the leaf is declared the *search result*.

It is known for sure that during the search we make a mistake in comparing exactly once, that is we go the wrong way, but we won't make any mistakes later. All possible mistakes are equiprobable, that is we should consider all such searches where exactly one mistake occurs. Your task is to find the expectation (the average value) of the search result for every search key, considering that exactly one mistake occurs in the search. That is, for a set of paths containing exactly one mistake in the given key search, you should count the average value of keys containing in the leaves of those paths.

Input

The first line contains an odd integer n $(3 \le n < 10^5)$, which represents the number of tree nodes. Next n lines contain node descriptions. The $(i + 1)$-th line contains two space-separated integers. The first number is the number of parent of the i-st node and the second number is the key lying in the i-th node. The next line contains an integer k $(1 \le k \le 10^5)$, which represents the number of keys for which you should count the average value of search results containing one mistake. Next k lines contain the actual keys, one key per line.

All node keys and all search keys are positive integers, not exceeding 10^9. All $n + k$ keys are distinct.

All nodes are numbered from 1 to n. For the tree root "-1" (without the quote) will be given instead of the parent's node number. It is guaranteed that the correct binary search tree is given. For each node except for the root, it could be determined according to its key whether it is the left child or the right one.

Output

Print k real numbers which are the expectations of answers for the keys specified in the input. The answer should differ from the correct one with the measure of absolute or relative error not exceeding 10^{-9}.

Examples

input
7 -1 8 1 4 1 12 2 2 2 6 3 10 3 14 1 1

output
8.0000000000

input
3 -1 5 1 3 1 7 6 1 2 4 6 8 9

output
7.0000000000

```
7.0000000000
7.0000000000
3.0000000000
3.0000000000
3.0000000000
```

Note

In the first sample the search of key 1 with one error results in two paths in the trees: (1, 2, 5) and (1, 3, 6), in parentheses are listed numbers of nodes from the root to a leaf. The keys in the leaves of those paths are equal to 6 and 10 correspondingly, that's why the answer is equal to 8.

D. Sum of Medians

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

In one well-known algorithm of finding the k-th order statistics we should divide all elements into groups of five consecutive elements and find the median of each five. A median is called the middle element of a sorted array (it's the third largest element for a group of five). To increase the algorithm's performance speed on a modern video card, you should be able to find a sum of medians in each five of the array.

A *sum of medians* of a sorted k-element set $S = \{a_1, a_2, ..., a_k\}$, where $a_1 < a_2 < a_3 < ... < a_k$, will be understood by as

$$\sum_{i \bmod 5=3}^{i \le k} a_i.$$

The $\bmod$ operator stands for taking the remainder, that is $x \bmod y$ stands for the remainder of dividing x by y.

To organize exercise testing quickly calculating *the sum of medians* for a changing set was needed.

Input

The first line contains number n ($1 \le n \le 10^5$), the number of operations performed.

Then each of n lines contains the description of one of the three operations:

- `add` x — add the element x to the set;
- `del` x — delete the element x from the set;
- `sum` — find the *sum of medians* of the set.

For any `add` x operation it is true that the element x is not included in the set directly before the operation.

For any `del` x operation it is true that the element x is included in the set directly before the operation.

All the numbers in the input are positive integers, not exceeding 10^9.

Output

For each operation `sum` print on the single line *the sum of medians* of the current set. If the set is empty, print 0.

Please, do not use the `%lld` specificator to read or write 64-bit integers in C++. It is preferred to use the `cin`, `cout` streams (also you may use the `%I64d` specificator).

Examples

input
6 add 4 add 5 add 1 add 2 add 3 sum
output
3

input
14 add 1 add 7 add 2 add 5 sum add 6 add 8 add 9 add 3 add 4 add 10 sum del 1 sum
output
5 11 13

E. Guard Towers

time limit per test: 1.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

In a far away kingdom lives a very greedy king. To defend his land, he built n guard towers. Apart from the towers the kingdom has two armies, each headed by a tyrannical and narcissistic general. The generals can't stand each other, specifically, they will never let soldiers of two armies be present in one tower.

During defence operations to manage a guard tower a general has to send part of his army to that tower. Each general asks some fee from the king for managing towers. As they live in a really far away kingdom, each general evaluates his fee in the following weird manner: he finds two remotest (the most distant) towers, where the soldiers of his army are situated and asks for the fee equal to the distance. Each tower is represented by a point on the plane with coordinates (x, y), and the distance between two points with coordinates (x_1, y_1) and (x_2, y_2) is determined in this kingdom as $|x_1 - x_2| + |y_1 - y_2|$.

The greedy king was not exactly satisfied with such a requirement from the generals, that's why he only agreed to pay one fee for two generals, equal to the maximum of two demanded fees. However, the king is still green with greed, and among all the ways to arrange towers between armies, he wants to find the cheapest one. Each tower should be occupied by soldiers of exactly one army.

He hired you for that. You should find the minimum amount of money that will be enough to pay the fees. And as the king is also very scrupulous, you should also count the number of arrangements that will cost the same amount of money. As their number can be quite large, it is enough for the king to know it as a remainder from dividing by $10^9 + 7$.

Two arrangements are distinct if the sets of towers occupied by soldiers of the first general are distinct.

Input

The first line contains an integer n $(2 \le n \le 5000)$, n is the number of guard towers. Then follow n lines, each of which contains two integers x, y — the coordinates of the i-th tower $(0 \le x, y \le 5000)$. No two towers are present at one point.

Pretest 6 is one of the maximal tests for this problem.

Output

Print on the first line the smallest possible amount of money that will be enough to pay fees to the generals.

Print on the second line the number of arrangements that can be carried out using the smallest possible fee. This number should be calculated modulo 1000000007 $(10^9 + 7)$.

Examples

input
2 0 0 1 1
output
0 2

input
4 0 0 0 1 1 0 1 1
output
1 4

input
3 0 0 1000 1000 5000 5000
output
2000 2

Note

In the first example there are only two towers, the distance between which is equal to 2. If we give both towers to one general, then we well have to pay 2 units of money. If each general receives a tower to manage, to fee will be equal to 0. That is the smallest possible fee. As you can easily see, we can obtain it in two ways.