Central Europe Regional Contest

University of Zagreb

Faculty of Electrical Engineering and Computing

November 17–19, 2017

Hosted by:

Sponsored by:

icpc global
programming
tools sponsor

Creating Efficiencies.
Stimulating Growth.

Organized by:

CROATIAN COMPUTER
SCIENCE ASSOCIATION

Problem A: Assignment Algorithm

Time limit: 1 s
Memory limit: 512 MiB

A low-budget airline is designing a sophisticated algorithm that will assign more desirable seats to passengers who buy tickets earlier. Their airplane has r rows of seats, where r is an even integer. There are also 3 *exit rows* in the airplane; those rows do not contain any seats but only provide access to the emergency exits. One exit row is in the very front of the airplane (before the first row of seats), one in the very back (behind the last row of seats) and one right in the middle. The rows are numbered with integers 1 through $r + 3$ with row numbers increasing from the front to the back of the airplane. Rows numbered 1, $r/2 + 2$ and $r + 3$ are exit rows while all the other rows are seat rows.

The seating configuration is "3–3–3" — each seat row contains three groups of three seats with the passenger aisles between the groups. Seats in the same row are denoted with consecutive letters left to right corresponding to the pattern "`ABC.DEF.GHI`".

When a passenger purchases a ticket, she is assigned a seat according to the following rules:

1. If there is an empty seat in a row directly after an exit row, all other rows are ignored in the following step (but they are not ignored when balancing the airplane in the last step).
2. First, we select a seat row with the largest number of empty seats. If there are multiple such rows, we select the one closest to an exit row (distance between rows a and b is simply $|a - b|$). If there are still multiple such rows, we select the one with the lowest number.
3. Now, we consider empty seats in the selected row and select one with the highest *priority*. Seat priorities, from highest to lowest are as follows:
   (a) Aisle seats in the middle group (`D` or `F`).
   (b) Aisle seats in the first and third group (`C` or `G`).
   (c) Window seats (`A` or `I`).
   (d) Middle seat in the middle group (`E`).
   (e) Other middle seats (`B` or `H`).

   If there are two empty seats with the same highest priority, we consider the balance of the entire airplane. The airplane's *left side* contains all seats with letters `A`, `B`, `C` or `D`, while the *right side* contains all seats with letters `F`, `G`, `H` or `I`. We select an empty seat in the side with more empty seats. If both sides have the same number of empty seats, we select the seat in the left side of the airplane.

Some of the airplane's seats are already reserved (possibly using a completely different procedure from the one described above). Determine the seats assigned to the next n passengers purchasing a ticket.

Input

The first line contains two integers r and n ($2 \leq r \leq 50$, $1 \leq n \leq 26$) — the number of seat rows in the airplane (always an even integer) and the number of new passengers purchasing tickets. The following $r + 3$ lines contain the current layout of the airplane. The j-th line contains exactly 11 characters — the layout of the row j of the airplane. Exit rows and aisles are denoted with the "`.`" characters. The "`#`" character denotes a reserved seat, while the "`-`" character denotes a seat that is currently empty. You may assume there will be at least n empty seats in the airplane.

Output

Output $r + 3$ lines containing the final layout of the airplane. The layout should be exactly the same as in input with the following exception: the seat assigned to the j-th passenger purchasing a ticket

should be denoted with the j-th lowercase letter of the English alphabet.

Example

input

```
2 17
...........
---.#--.---
...........
---.---.---
...........
```

output

```
...........
hnd.#lb.fpj
...........
kqg.cma.eoi
...........
```

input

```
6 26
...........
---.---.###
#-#.---.---
---.###.---
...........
---.###.---
#--.#-#.--#
#--.--#.#-#
...........
```

output

```
...........
gke.aic.###
#-#.mzo.r-v
x-p.###.n-t
...........
fjb.###.dlh
#-s.#-#.w-#
#-u.qy#.#-#
...........
```

Problem B: Buffalo Barricades

Time limit: 5 s
Memory limit: 512 MiB

A pasture in the wild west can be represented as a rectangular grid embedded in the upper-right quadrant of a standard coordinate system. A herd of n buffalos is scattered throughout the grid, each occupying a unit square. Buffalos are numbered 1 through n; buffalo j is located in the unit square whose upper-right corner is the point with integer coordinates (x_j, y_j). The coordinate axes represent two rivers meeting at the origin, restricting buffalo movement downwards and leftwards.

A total of m settlers arrive, one by one, and each claims a piece of land using the following procedure:

1. The settler picks a point with integer coordinates and installs a single fence post at that point. The point he picks is guaranteed to be free of any previously installed fence posts or fences. Moreover, no two fence posts will have the same x-coordinate and no two fence posts will have the same y-coordinates.
2. Starting from the fence post, the settler builds horizontal and vertical fence segments leftwards and downwards, respectively. Each segment is built to be as long as possible — i. e. until it reaches the river or another fence.
3. The settler claims all the land in the connected area bounded with fences and rivers whose upper-right corner is his fence post. Of course, he claims all the buffalos inside as well. Note that settlers arriving later may claim pieces of land already claimed by earlier settlers.

For each settler, find the total number of buffalos he claimed when he arrived.

Input

The first line contains an integer n ($1 \leq n \leq 300\,000$) — the number of buffalos. The j-th of the following n lines contains two integers x_j and y_j ($1 \leq x_j, y_j \leq 10^9$) — the location of the j-th buffalo. No two buffalos will share the same location.

The following line contains an integer m ($1 \leq m \leq 300\,000$) — the number of settlers. The j-th of the following m lines contains two integers x'_j and y'_j ($1 \leq x'_j, y'_j \leq 10^9$) — the coordinates of the fence post installed by the j-th settler. All x'_j are different and all y'_j are different.

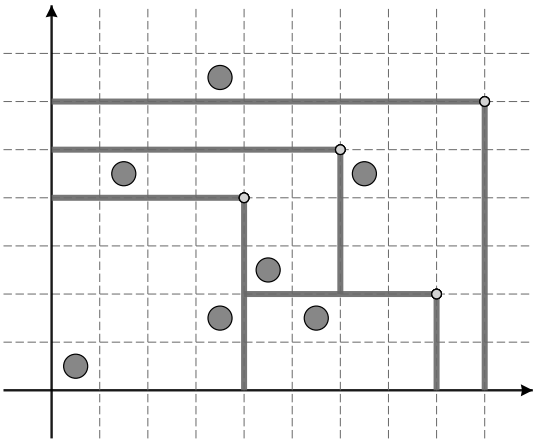
Output

Output m lines. The j-th line should contain the number of buffalos claimed by the j-th settler upon arrival.

Example

input

```
7
1 1
4 2
6 2
5 3
2 5
4 7
7 5
4
4 4
8 2
9 6
6 5
```



output

```
2
1
3
2
```

Problem C: Cumulative Code

Time limit: 7 s
Memory limit: 512 MiB

As you probably know, a *tree* is a graph consisting of n nodes and $n-1$ undirected edges in which any two nodes are connected by exactly one path. In a *labeled tree* each node is labeled with a different integer between 1 and n.

The *Prüfer code* of a labeled tree is a unique sequence associated with the tree, generated by repeatedly removing nodes from the tree until only two nodes remain. More precisely, in each step we remove the *leaf* with the smallest label and append the label of its *neighbour* to the end of the code. Recall, a leaf is a node with exactly one neighbour. Therefore, the Prüfer code of a labeled tree is an integer sequence of length $n-2$. It can be shown that the original tree can be easily reconstructed from its Prüfer code.

The *complete binary tree of depth k*, denoted with C_k, is a labeled tree with 2^k-1 nodes where node j is connected to nodes $2j$ and $2j+1$ for all $j < 2^{k-1}$. Denote the Prüfer code of C_k with $p_1, p_2, \ldots, p_{2^k-3}$.

Since the Prüfer code of C_k can be quite long, you do not have to print it out. Instead, you need to answer n questions about the sums of certain elements on the code. Each question consists of three integers: a, d and m. The answer is the sum of the of the C_k's Prüfer code elements p_a, p_{a+d}, p_{a+2d}, $\ldots$, $p_{a+(m-1)d}$.

Input

The first line contains two integers k and q ($2 \le k \le 30$, $1 \le q \le 300$) — the depth of the complete binary tree and the number of questions. The j-th of the following q lines contains the j-th question: three positive integers a_j, d_j and m_j such that a_j, d_j and $a_j + (m_j - 1)d_j$ are all at most $2^k - 3$.

Output

Output 1 lines. The j-th line should contain a single integer — the answer to the j-th question.

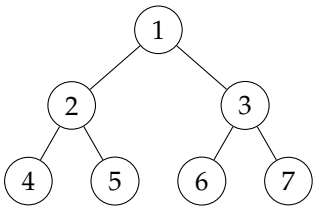
Example

input

```
3 5
1 1 1
2 1 1
3 1 1
4 1 1
5 1 1
```

output

```
2
2
1
3
3
```



input

```
4 4
2 1 5
4 4 3
4 8 1
10 3 2
```

output

```
18
15
5
13
```

input

```
7 1
1 1 125
```

output

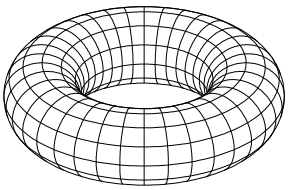
```
4031
```

In the first example above, when constructing the Prüfer code for C_3 the nodes are removed in the following order: 4, 5, 2, 1, 6. Therefore, the Prüfer code of C_3 is 2, 2, 1, 3, 3.

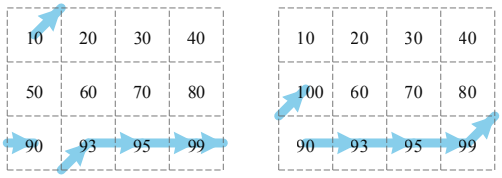
Problem D: Donut Drone

Time limit: 8
Memory limit: 512 MiB

You are building a simulation in which a drone explores a volatile torus-shaped planet. Technically, the drone is moving across a *toroidal grid* — a rectangular grid that wraps around circularly in both dimensions. The grid consists of cells organized into r rows numbered 1 through r top to bottom and c columns numbered 1 through c left to right. Each grid cell has a certain *elevation* — a positive integer.



A toroidal grid.



The paths in two `move` events in the second example input.

The drone is initially located in the cell in the first row and first column. In each *step* the drone considers three cells: the cell directly to the right, the cell diagonally right-down and the cell diagonally right-up (wrapping around if necessary). The drone flies to the cell with the largest elevation of the three.

Two types of events may happen during the simulation:

- "`move` k" — The drone makes k steps.
- "`change` a b e" — The elevation of the cell in row a column b changes to e.

Find the drone's position immediately after each `move` event. You may assume that at each point in time, no sequence of three circularly consecutive cells in the same column will have the same elevation. Hence, each drone step is well defined.

Input

The first line contains two integers r and c ($3 \leq r, c \leq 2\,000$) — the number of rows and the number of columns of the toroidal grid. The i-th of the following r lines contains a sequence of c integers $e_{i,1}, e_{i,2}, \ldots, e_{i,c}$ ($1 \leq e_{i,j} \leq 10^9$) — the initial elevations of cells in row i.

The following line contains an integer m ($1 \leq m \leq 5\,000$) — the number of events. The j-th of the following m lines contains the j-th event and is either of the form "`move` k" where k is an integer such that $1 \leq k \leq 10^9$ or "`change` a b e" where a, b and e are integers such that $1 \leq a \leq r$, $1 \leq b \leq c$ and $1 \leq e \leq 10^9$.

Output

Output w lines where w is the number of `move` events in the input — the j-th line should contain the drone's position (row and column numbers) after the j-th `move` event in the input.

Example

input

```
4 4
1 2 9 3
3 5 4 8
4 3 2 7
5 8 1 6
4
move 1
move 1
change 1 4 100
move 1
```

output

```
4 2
1 3
1 4
```

input

```
3 4
10 20 30 40
50 60 70 80
90 93 95 99
3
move 4
change 2 1 100
move 4
```

output

```
3 1
2 1
```

Problem E: Embedding Enumeration

Time limit: 4 s
Memory limit: 512 MiB

As you probably know, a *tree* is a graph consisting of n nodes and $n-1$ undirected edges in which any two nodes are connected by exactly one path. In a *labeled tree* each node is labeled with a different integer between 1 and n. In general, it may be hard to visualize trees nicely, but some trees can be neatly embedded in rectangular grids.

Given a labeled tree G with n nodes, a *2 by n embedding* of G is a mapping of nodes of G to the cells of a rectangular grid consisting of 2 rows and n columns such that:

- Node 1 is mapped to the cell in the upper-left corner.
- Nodes connected with an edge are mapped to neighboring grid cells (up, down, left or right).
- No two nodes are mapped to the same cell.

Find the number of 2 by n embeddings of a given tree, modulo $10^9 + 7$.

Input

The first line contains an integer n ($1 \leq n \leq 300\,000$) — the number of nodes in G. The j-th of the following $n-1$ lines contains two different integers a_j and b_j ($1 \leq a_j, b_j \leq n$) — the endpoints of the j-th edge.

Output

Output the number of 2 by n embeddings of the given tree, modulo $10^9 + 7$.

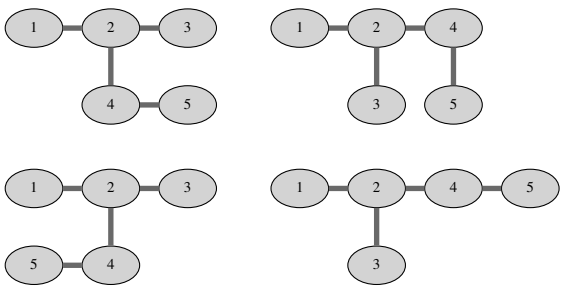
Example

input

```
5
1 2
2 3
2 4
4 5
```

output

```
4
```



All 4 embeddings of the tree in the example input are given in the figure above.

Problem F: Faulty Factorial

Time limit: 3 s
Memory limit: 512 MiB

The *factorial* of a natural number is the product of all positive integers less than or equal to it. For example, the factorial of 4 is $1 \cdot 2 \cdot 3 \cdot 4 = 24$. A *faulty factorial* of length n is similar to the factorial of n, but it contains a fault: one of the integers is *strictly smaller* than what it should be (but still at least 1). For example, $1 \cdot 2 \cdot 2 \cdot 4 = 16$ is a faulty factorial of length 4.

Given the length n, a *prime* modulus p and a target remainder r, find some faulty factorial of length n that gives the remainder r when divided by p.

Input

The first line contains three integers n, p and r ($2 \leq n \leq 10^{18}$, $2 \leq p < 10^7$, $0 \leq r < p$) — the length of the faulty factorial, the prime modulus and the target remainder as described in the problem statement.

Output

If there is no faulty factorial satisfying the requirements output "-1 -1". Otherwise, output two integers — the index k of the fault ($2 \leq k \leq n$) and the value v at that index ($1 \leq v < k$). If there are multiple solutions, output any of them.

Examples

input	input
4 5 1	4 127 24
output	**output**
3 2	-1 -1

In the first example, the output describes the faulty factorial $1 \cdot 2 \cdot 2 \cdot 4 = 2^4 \equiv 1 (\mathrm{mod}\ 5)$.

Problem G: Gambling Guide

Time limit: 3 s
Memory limit: 512 MiB

A railroad network in a nearby country consists of n cities numbered 1 through n, and m two-way railroad tracks each connecting two different cities. Tickets can only be purchased at automated machines installed at every city. Unfortunately, hackers have tampered with the ticket machines and now they all work as follows: when a single coin is inserted in the machine installed at city a, the machine dispenses a single one-way ticket from a to a *random* neighboring city. More precisely, the destination city is chosen uniformly at random among all cities directly connected to a with a railroad track. Destinations on different tickets originating in the same city are independent.

A computer science student needs to travel from city 1 (where she lives) to city n (where a regional programming contest has already started). She knows how the machines work (but of course cannot predict the random choices) and has a map of the railway network. In each city, when she purchases a ticket, she can either immediately use it and travel to the destination city on the ticket, or discard the ticket and purchase a new one. She can keep purchasing tickets indefinitely. The trip is finished as soon as she reaches city n.

After doing some calculations, she has devised a traveling strategy with the following properties:

- The probability that the trip will eventually finish is 1.
- The expected number of coins spent on the trip is the smallest possible.

Find the expected number of coins she will spend on the trip.

Input

The first line contains two integers n and m ($1 \leq n, m \leq 300\,000$) — the number of cities and the number of railroad tracks. Each of the following m lines contains two different integers a and b ($1 \leq a, b \leq n$) which describe a railroad track connecting cities a and b. There will be at most one railroad track between each pair of cities. It will be possible to reach city n starting from city 1.

Output

Output a single number — the expected number of coins spent. The solution will be accepted if the absolute or the relative difference from the judges solution is less than 10^{-6}.

Example

input

```
4 4
1 2
1 3
2 4
3 4
```

output

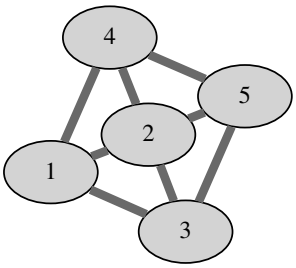
```
3.0000000000
```

input

```
5 8
1 2
1 3
1 4
2 3
2 4
3 5
5 4
2 5
```

output

```
4.1111111111
```



Problem H: Hidden Hierarchy

Time limit: 1 s
Memory limit: 512 MiB

You are working on the user interface for a simple text-based file explorer. One of your tasks is to build a navigation pane displaying the *directory hierarchy*. As usual, the filesystem consists of directories which may contain files and other directories, which may, in turn, again contain files and other directories etc. Hence, the directories form a hierarchical tree structure. The top-most directory in the hierarchy is called the *root* directory. If directory d directly contains directory e we will say that d is the *parent directory* of e while e is a *subdirectory* od d. Each file has a *size* expressed in bytes. The directory size is simply the total size of all files directly or indirectly contained inside that directory.

All files and all directories except the root directory have a *name* — a string that always starts with a letter and consists of only lowercase letters and "." (dot) characters. All items (files and directories) directly inside the same parent directory must have unique names. Each item (file and directory) can be uniquely described by its *path* — a string built according to the following rules:

- Path of the root directory is simply "/" (forward slash).
- For a directory d, its path is obtained by concatenating the directory names top to bottom along the hierarchy from the root directory to d, preceding each name with the "/" character and placing another "/" character at the end of the path.
- For a file f, its path is the concatenation of the parent directory path and the name of file f.

We display the directory hierarchy by *printing* the root directory. We print a directory d by outputting a line of the form "$m_d \sqcup p_d \sqcup s_d$" where p_d and s_d are the path and size of directory d respectively, while m_d is its *expansion marker* explained shortly. If d contains other directories we must choose either to *collapse* it or to *expand* it. If we choose to expand d we print (using the same rules) all of its subdirectories in lexicographical order by name. If we choose to collapse directory d, we simply ignore its contents.

The expansion marker m_d is a single blank character when d does not have any subdirectories, "+" (plus) character when we choose to collapse d or a "-" (minus) character when we choose expand d.

Given a list of files in the filesystem and a threshold integer t, display the directory hierarchy ensuring that each directory of size at least t is printed. Additionally, the total number of directories printed should be minimal. Assume there are no empty directories in the filesystem — the entire hierarchy can be deduced from the provided file paths. Note that the root directory has to be printed regardless of its size. Also note that a directory of size at least t only has to be *printed*, but not necessarily *expanded*.

Input

The first line contains an integer n ($1 \leq n \leq 1\,000$) — the number of files. Each of the following n lines contains a string f and an integer s ($1 \leq s \leq 10^6$) — the path and the size of a single file. Each path is at most 100 characters long and is a valid file path according to the rules above. All paths will be different.

The following line contains an integer t ($1 \leq t \leq 10^9$) — the threshold directory size.

Output

Output the minimal display of the filesystem hierarchy for the given threshold as described above.

Example

input

```
9
/sys/kernel/notes 100
/cerc/problems/a/testdata/in 1000000
/cerc/problems/a/testdata/out 8
/cerc/problems/a/luka.cc 500
/cerc/problems/a/zuza.cc 5000
/cerc/problems/b/testdata/in 15
/cerc/problems/b/testdata/out 4
/cerc/problems/b/kale.cc 100
/cerc/documents/rules.pdf 4000
10000
```

output

```
- / 1009727
- /cerc/ 1009627
  /cerc/documents/ 4000
- /cerc/problems/ 1005627
- /cerc/problems/a/ 1005508
  /cerc/problems/a/testdata/ 1000008
+ /cerc/problems/b/ 119
+ /sys/ 100
```

input

```
8
/b/test/in.a 100
/b/test/in.b 1
/c/test/in.a 100
/c/test/in.b 1
/c/test/pic/in.a.svg 10
/c/test/pic/in.b.svg 10
/a/test/in.a 99
/a/test/in.b 1
101
```

output

```
- / 322
+ /a/ 100
- /b/ 101
  /b/test/ 101
- /c/ 121
+ /c/test/ 121
```

input

```
2
/a/a/a 100
/b.txt 99
200
```

output

```
+ / 199
```

Problem I: Intrinsic Interval

Time limit: 3 s
Memory limit: 512 MiB

Given a permutation π of integers 1 through n, an *interval* in π is a consecutive subsequence consisting of consecutive numbers. More precisely, for indices a and b where $1 \leq a \leq b \leq n$, the subsequence $\pi_a^b = (\pi_a, \pi_{a+1}, \ldots, \pi_b)$ is an interval if sorting it would yield a sequence of consecutive integers. For example, in permutation $\pi = (3, 1, 7, 5, 6, 4, 2)$, the subsequence π_3^6 is an interval (it contains the numbers 4 through 7) while π_1^3 is not.

For a subsequence π_x^y its *intrinsic interval* is any interval π_a^b that contains the given subsequence ($a \leq x \leq y \leq b$) and that is, additionally, as short as possible. Of course, the *length* of an interval is defined as the number of elements it contains.

Given a permutation π and m of its subsequences, find some intrinsic interval for each subsequence.

Input

The first line contains an integer n ($1 \leq n \leq 100\,000$) — the size of the permutation π. The following line contains n different integers $\pi_1, \pi_2, \ldots, \pi_n$ ($1 \leq \pi_j \leq n$) — the permutation itself.

The following line contains an integer m ($1 \leq m \leq 100\,000$) — the number of subsequences. The j-th of the following m lines contains integers x_j and y_j ($1 \leq x_j \leq y_j \leq n$) — the endpoints of the j-th subsequence.

Output

Output m lines. The j-th line should contain two integers a_j and b_j where $1 \leq a_j \leq b_j \leq n$ — the endpoints of some intrinsic interval of the j-th subsequence $\pi_{x_j}^{y_j}$.

Example

input

```
7
3 1 7 5 6 4 2
3
3 6
7 7
1 3
```

output

```
3 6
7 7
1 7
```

input

```
10
2 1 4 3 5 6 7 10 8 9
5
2 3
3 7
4 7
4 8
7 8
```

output

```
1 4
3 7
3 7
3 10
7 10
```

Problem J: Justified Jungle

Time limit: 6 s
Memory limit: 512 MiB

As you probably know, a *tree* is a graph consisting of n nodes and $n-1$ undirected edges in which any two nodes are connected by exactly one path. A *forest* is a graph consisting of one or more trees. In other words, a graph is a forest if every connected component is a tree. A forest is *justified* if all connected components have the same number of nodes.

Given a tree G consisting of n nodes, find all positive integers k such that a justified forest can be obtained by erasing exactly k edges from G. Note that erasing an edge never erases any nodes. In particular when we erase all $n-1$ edges from G, we obtain a justified forest consisting of n one-node components.

Input

The first line contains an integer n ($2 \leq n \leq 1\,000\,000$) — the number of nodes in G. The k-th of the following $n-1$ lines contains two different integers a_k and b_k ($1 \leq a_k, b_k \leq n$) — the endpoints of the k-th edge.

Output

The first line should contain all wanted integers k, in increasing order.

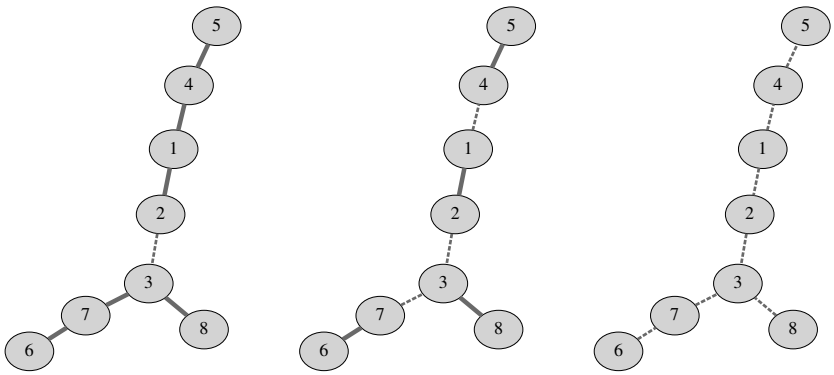
Example

input

```
8
1 2
2 3
1 4
4 5
6 7
8 3
7 3
```

output

```
1 3 7
```

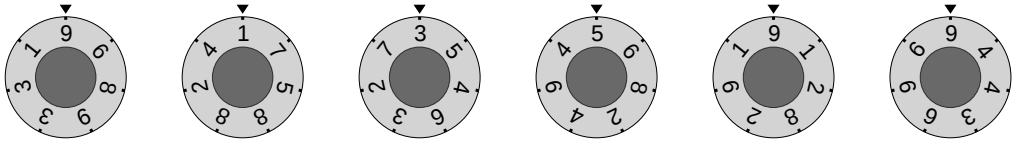


Figures depict justified forests obtained by erasing 1, 3 and 7 edges from the tree in the example input.

Problem K: Kitchen Knobs

Time limit: 3 s
Memory limit: 512 MiB

You are cooking on a gigantic stove at a large fast-food restaurant. The stove contains n heating elements arranged in a line and numbered with integers 1 through n left to right. Each element is operated by its *control knob*. The knobs are a bit unusual: each knob is marked with seven non-zero digits evenly distributed around a circle. The *power* of the heating element is equal to the positive integer obtained by reading the digits on its control knob clockwise starting from the top of the knob.



Initial positions of the control knobs in the first example input below.

In a single step, you can rotate one or more *consecutive* knobs by any number of positions in any direction. However, all knobs rotated in one step need to be rotated by the same number of positions in the same direction.

Find the smallest number of steps needed to set all the heating elements to maximal possible power.

Input

The first line contains an integer n ($1 \leq n \leq 501$) — the number of heating elements. The j-th of the following n lines contains an integer x_j — the initial power of the j-th heating element. Each x_j consists of exactly seven non-zero digits.

Output

Output a single integer — the minimal number of steps needed.

Example

input

```
6
9689331
1758824
3546327
5682494
9128291
9443696
```

output

```
3
```

input

```
7
5941186
3871463
8156346
9925977
8836125
9999999
5987743
```

output

```
2
```

In the first example, one of the ways to achieve maximal possible power is: rotate knobs 2 through 3 by 3 positions in the counterclockwise direction, rotate knob 3 by 3 positions in the counterclockwise direction, and rotate knobs 4 through 6 by 2 positions in the clockwise direction.

Problem L: Lunar Landscape

Time limit: 2 s
Memory limit: 512 MiB

A satellite is surveying a possible rover landing area on the moon. The landing area is modeled as a square grid embedded in the standard coordinate system.

The satellite has taken n photos, each capturing a square area of the surface. Careful camera calibration has ensured that all photos are aligned with the grid — all four vertices have integer coordinates. Due to the satellite's changing orbit there are two types of photos:

- Photos of type `A` have sides that are parallel to coordinate axes. Such a photo is specified by giving the integer coordinates (x, y) of the square's middle point and the length of its side a — always an even integer.
- Photos of type `B` have sides at a 45° angle to the coordinate axes. Such a photo is specified by giving the integer coordinates (x, y) of the square's middle point and the length of its diagonal d — always an even integer.

Find the total surface area captured in the satellite photos.

Input

The first line contains an integer n ($1 \leq n \leq 200\,000$) — the number of photos. The j-th of the following n lines is either of the form "`A` x_j y_j a_j" or "`B` x_j y_j d_j" representing a photo of type `A` or `B`, respectively. The x_j and y_j are the integer coordinates of the middle point of the photo ($-1\,000 \leq x_j, y_j \leq 1\,000$). The a_j and d_j are even integers ($2 \leq a_j, d_j \leq 1\,000$) — the side length and the diagonal length, respectively.

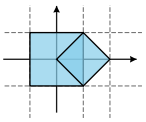
Output

Output a number with exactly two digits after the decimal point — the total area of the surface. The answer has to exactly correspond to the judge's solution (no rounding errors are tolerated).

Example

input

```
2
A 0 0 2
B 1 0 2
```



output

```
5.00
```

input

```
8
A -7 10 4
B 3 10 8
A -6 6 6
A -2 5 8
B 3 -1 8
B -7 -4 8
A 3 9 2
B 8 6 6
```

output

```
205.50
```