# Problem A
## Amalgamated Artichokes
### Time limit: 5 seconds

Fatima Cynara is an analyst at Amalgamated Artichokes (AA). As with any company, AA has had some very good times as well as some bad ones. Fatima does trending analysis of the stock prices for AA, and she wants to determine the largest decline in stock prices over various time spans. For example, if over a span of time the stock prices were 19, 12, 13, 11, 20 and 14, then the largest decline would be 8 between the first and fourth price. If the last price had been 10 instead of 14, then the largest decline would have been 10 between the last two prices.

Picture by Hans Hillewaert via Wikimedia Commons

Fatima has done some previous analyses and has found that the stock price over any period of time can be modelled reasonably accurately with the following equation:

$$\text{price}(k) = p \cdot (\sin(a \cdot k + b) + \cos(c \cdot k + d) + 2)$$

where $p$, $a$, $b$, $c$ and $d$ are constants. Fatima would like you to write a program to determine the largest price decline over a given sequence of prices. Figure A.1 illustrates the price function for Sample Input 1. You have to consider the prices only for integer values of $k$.

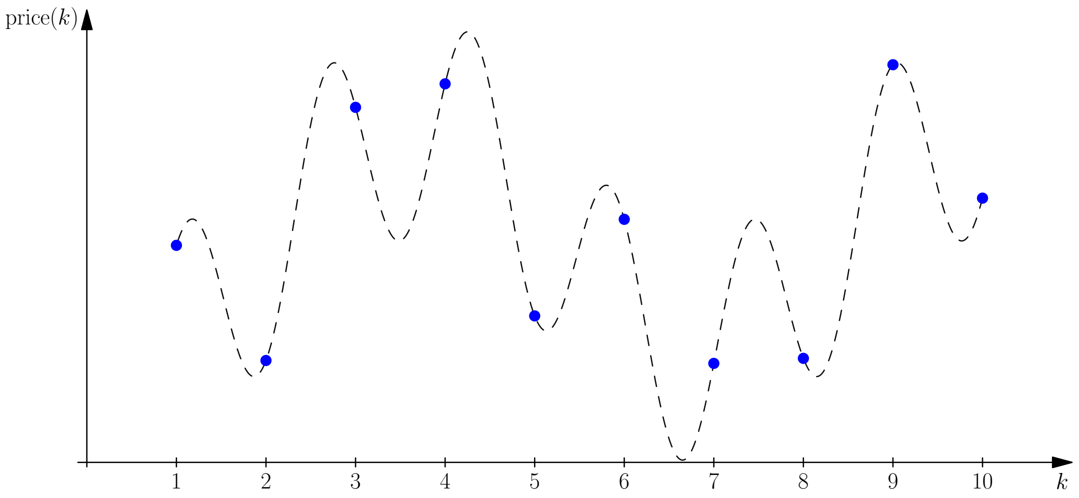

Figure A.1: Sample Input 1. The largest decline occurs from the fourth to the seventh price.

## Input

The input consists of a single line containing 6 integers $p$ ($1 \le p \le 1\,000$), $a$, $b$, $c$, $d$ ($0 \le a, b, c, d \le 1\,000$) and $n$ ($1 \le n \le 10^6$). The first 5 integers are described above. The sequence of stock prices to consider is $\text{price}(1), \text{price}(2), \ldots, \text{price}(n)$.

## Output

Display the maximum decline in the stock prices. If there is no decline, display the number $0$. Your output should have an absolute or relative error of at most $10^{-6}$.

| Sample Input 1 | Sample Output 1 |
|---|---|
| `42 1 23 4 8 10` | `104.855110477` |

| Sample Input 2 | Sample Output 2 |
|---|---|
| `100 7 615 998 801 3` | `0.00` |

| Sample Input 3 | Sample Output 3 |
|---|---|
| `100 432 406 867 60 1000` | `399.303813` |

# Problem B
## Asteroids
### Time limit: 2 seconds

The year is 2115. The asteroid communication relay system was set up a decade ago by the Asteroid Communication Ministry. It is running fine except for one small problem – there are too many asteroids! The smaller ones not only keep interfering with the signals from the relay stations but they are also a danger to all the maintenance aircrafts that fly between the stations. These small asteroids must be destroyed! The Interplanetary Coalition to Prevent Catastrophes (ICPC) has been charged with removing these dangerous asteroids and has hired an elite team of hot-shot pilots for the job. Han Duo is the captain of this team of asteroid destroyers. Armed with his missiles, Han flies through the asteroid belt blowing up any asteroid that the ICPC deems a nuisance.

The ICPC is having some unfortunate budgetary problems. One result of this is that Han and his team do not have as many missiles as they would like, so they cannot blow up all the troublesome asteroids. But the asteroids are small and the missiles are powerful. So if two asteroids are near each other and line up properly, it is possible to take out both with a single missile.

Han's screen displays asteroids as non-rotating two-dimensional simple convex polygons, each of which moves at a fixed velocity. He has decided that the best time to hit two asteroids is when the overlap of the two polygons is at a maximum. For example, Figure B.1, which illustrates Sample Input 1, shows two asteroids and snapshots of their subsequent positions at 1-second intervals. The two asteroids start touching after 3 seconds and the maximum overlap area occurs between 4 and 5 seconds.

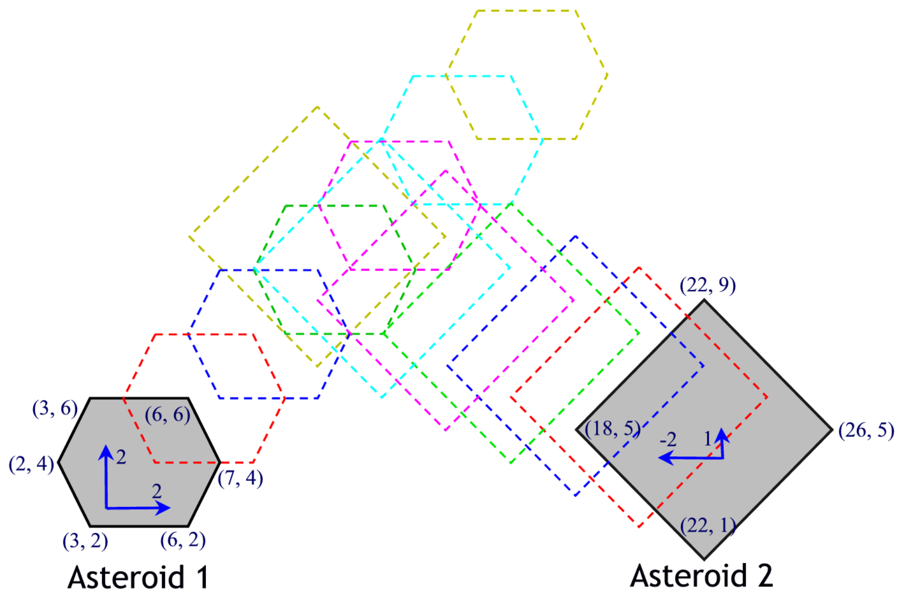

Figure B.1: Sample Input 1. Two asteroids with crossing paths.

Calculating when the maximum overlap occurs for two asteroids requires a bit of programming, but unfortunately Han slept through most of his coding classes at the flight academy. This is where you come in.

## Input

The input consists of two asteroid specifications. Each has the form $n\ x_1\ y_1\ x_2\ y_2\ \ldots\ x_n\ y_n\ v_x\ v_y$ where $n$ ($3 \leq n \leq 10$) is the number of vertices, each $x_i, y_i$ ($-10\,000 \leq x_i, y_i \leq 10\,000$) are the coordinates of a vertex of the asteroid on Han's screen given in clockwise order, and $v_x, v_y$ ($-100 \leq v_x, v_y \leq 100$) are the $x$ and $y$ velocities (in units/second) of the asteroid. The $x_i$, $y_i$ values specify the location of each asteroid at time $t = 0$, and the polygons do not intersect or touch at this time. The maximum length of any side of an asteroid is $500$. All numbers in the input are integers.

## Output

Display the time in seconds when the two polygons have maximum intersection, using the earliest such time if there is more than one. If the two polygons never overlap but touch each other, treat it as an intersection where the common area is zero and display the earliest such time. If the polygons never overlap or touch, display `never` instead. You should consider positive times only. Your output should have an absolute or relative error of at most $10^{-3}$.

| Sample Input 1 | Sample Output 1 |
|---|---|
| `6 3 2 2 4 3 6 6 6 7 4 6 2 2 2`<br>`4 18 5 22 9 26 5 22 1 -2 1` | `4.193518` |

| Sample Input 2 | Sample Output 2 |
|---|---|
| `4 0 0 0 2 2 2 2 0 -1 1`<br>`4 10 0 10 2 12 2 12 0 1 1` | `never` |

# Problem C
## Catering
Time limit: 4 seconds

Picture from Wikimedia Commons

Paul owns a catering company and business is booming. The company has $k$ catering teams, each in charge of one set of catering equipment. Every week, the company accepts $n$ catering requests for various events. For every request, they send a catering team with their equipment to the event location. The team delivers the food, sets up the equipment, and instructs the host on how to use the equipment and serve the food. After the event, the host is responsible for returning the equipment back to Paul's company.

Unfortunately, in some weeks the number of catering teams is less than the number of requests, so some teams may have to be used for more than one event. In these cases, the company cannot wait for the host to return the equipment and must keep the team on-site to move the equipment to another location. The company has an accurate estimate of the cost to move a set of equipment from any location to any other location. Given these costs, Paul wants to prepare an Advance Catering Map to service the requests while minimizing the total moving cost of equipment (including the cost of the first move), even if that means not using all the available teams. Paul needs your help to write a program to accomplish this task. The requests are sorted in ascending order of their event times and they are chosen in such a way that for any $i < j$, there is enough time to transport the equipment used in the $i^{th}$ request to the location of the $j^{th}$ request.

## Input

The first line of input contains two integers $n$ ($1 \leq n \leq 100$) and $k$ ($1 \leq k \leq 100$) which are the number of requests and the number of catering teams, respectively. Following that are $n$ lines, where the $i^{th}$ line contains $n - i + 1$ integers between $0$ and $1\,000\,000$ inclusive. The $j^{th}$ number in the $i^{th}$ line is the cost of moving a set of equipment from location $i$ to location $i + j$. The company is at location $1$ and the $n$ requests are at locations $2$ to $n + 1$.

## Output

Display the minimum moving cost to service all requests. (This amount does not include the cost of moving the equipment back to the catering company.)

| Sample Input 1 | Sample Output 1 |
|---|---|
| 3 2<br>40 30 40<br>50 10<br>50 | 80 |

**Sample Input 2**

```
3 2
10 10 10
20 21
21
```

**Sample Output 2**

```
40
```

# Problem D
## Cutting Cheese
Time limit: 3 seconds

Of course you have all heard of the International Cheese Processing Company. Their machine for cutting a piece of cheese into slices of exactly the same thickness is a classic. Recently they produced a machine able to cut a spherical cheese (such as Edam) into slices – no, not all of the same thickness, but all of the same weight! But new challenges lie ahead: cutting Swiss cheese.

Picture by Jon Sullivan via Wikimedia Commons

Swiss cheese such as Emmentaler has holes in it, and the holes may have different sizes. A slice with holes contains less cheese and has a lower weight than a slice without holes. So here is the challenge: cut a cheese with holes in it into slices of equal weight.

By smart sonar techniques (the same techniques used to scan unborn babies and oil fields), it is possible to locate the holes in the cheese up to micrometer precision. For the present problem you may assume that the holes are perfect spheres.

Each uncut block has size $100 \times 100 \times 100$ where each dimension is measured in millimeters. Your task is to cut it into $s$ slices of equal weight. The slices will be 100 mm wide and 100 mm high, and your job is to determine the thickness of each slice.

## Input

The first line of the input contains two integers $n$ and $s$, where $0 \leq n \leq 10\,000$ is the number of holes in the cheese, and $1 \leq s \leq 100$ is the number of slices to cut. The next $n$ lines each contain four positive integers $r$, $x$, $y$, and $z$ that describe a hole, where $r$ is the radius and $x$, $y$, and $z$ are the coordinates of the center, all in micrometers.

The cheese block occupies the points $(x, y, z)$ where $0 \leq x, y, z \leq 100\,000$, except for the points that are part of some hole. The cuts are made perpendicular to the $z$ axis.

You may assume that holes do not overlap but may touch, and that the holes are fully contained in the cheese but may touch its boundary.

## Output

Display the $s$ slice thicknesses in millimeters, starting from the end of the cheese with $z = 0$. Your output should have an absolute or relative error of at most $10^{-6}$.

**Sample Input 1**

```
0 4
```

**Sample Output 1**

```
25.000000000
25.000000000
25.000000000
25.000000000
```

**Sample Input 2**

```
2 5
10000 10000 20000 20000
40000 40000 50000 60000
```

**Sample Output 2**

```
14.611103142
16.269801734
24.092457788
27.002992272
18.023645064
```

# Problem E
## Evolution in Parallel
Time limit: 2 seconds

It is 2178, and alien life has been discovered on a distant planet. There seems to be only one species on the planet and they do not reproduce as animals on Earth do. Even more amazing, the genetic makeup of every single organism is identical!

The genetic makeup of each organism is a single sequence of nucleotides. The nucleotides come in three types, denoted by 'A' (Adenine), 'C' (Cytosine), and 'M' (Muamine). According to one hypothesis, evolution on this planet occurs when a new nucleotide is inserted somewhere into the genetic sequence of an existing organism. If this change is evolutionarily advantageous, then organisms with the new sequence quickly replace ones with the old sequence.

It was originally thought that the current species evolved this way from a single, very simple organism with a single-nucleotide genetic sequence, by way of mutations as described above. However, fossil evidence suggests that this might not have been the case. Right now, the research team you are working with is trying to validate the concept of "parallel evolution" – that there might actually have been two evolutionary paths evolving in the fashion described above, and eventually both paths evolved to the single species present on the planet today. Your task is to verify whether the parallel evolution hypothesis is consistent with the genetic material found in the fossil samples gathered by your team.

## Input

The input begins with a number $n$ ($1 \leq n \leq 4\,000$) denoting the number of nucleotide sequences found in the fossils. The second line describes the nucleotide sequence of the species currently living on the planet. Each of the next $n$ lines describes one nucleotide sequence found in the fossils.

Each nucleotide sequence consists of a string of at least one but no more than $4\,000$ letters. The strings contain only upper-case letters `A`, `C`, and `M`. All the nucleotide sequences, including that of the currently live species, are distinct.

## Output

Display an example of how the nucleotide sequences in the fossil record participate in two evolutionary paths. The example should begin with one line containing two integers $s_1$ and $s_2$, the number of nucleotide sequences in the fossil record that participate in the first path and second path, respectively. This should be followed by $s_1$ lines containing the sequences attributed to the first path, in chronological order (from the earliest), and then $s_2$ lines containing the sequences attributed to the second path, also in chronological order. If there are multiple examples, display any one of them. If it is possible that a sequence could appear in the genetic history of both species, your example should assign it to exactly one of the evolutionary paths.

If it is impossible for all the fossil material to come from two evolutionary paths, display the word `impossible`.

**Sample Input 1**

```
5
AACCMMAA
ACA
MM
ACMAA
AA
A
```

**Sample Output 1**

```
1 4
MM
A
AA
ACA
ACMAA
```

**Sample Input 2**

```
3
ACMA
ACM
ACA
AMA
```

**Sample Output 2**

```
impossible
```

**Sample Input 3**

```
1
AM
MA
```

**Sample Output 3**

```
impossible
```

**Sample Input 4**

```
4
AAAAAA
AA
AAA
A
AAAAA
```

**Sample Output 4**

```
0 4
A
AA
AAA
AAAAA
```

# Problem F
## Keyboarding
### Time limit: 4 seconds

How many keystrokes are necessary to type a text message? You may think that it is equal to the number of characters in the text, but this is correct only if one keystroke generates one character. With pocket-size devices, the possibilities for typing text are often limited. Some devices provide only a few buttons, significantly fewer than the number of letters in the alphabet. For such devices, several strokes may be needed to type a single character. One mechanism to deal with these limitations is a virtual keyboard displayed on a screen, with a cursor that can be moved from key to key to select characters. Four arrow buttons control the movement of the cursor, and when the cursor is positioned over an appropriate key, pressing the fifth button selects the corresponding character and appends it to the end of the text. To terminate the text, the user must navigate to and select the Enter key. This provides users with an arbitrary set of characters and enables them to type text of any length with only five hardware buttons.

In this problem, you are given a virtual keyboard layout and your task is to determine the minimal number of strokes needed to type a given text, where pressing any of the five hardware buttons constitutes a stroke. The keys are arranged in a rectangular grid, such that each virtual key occupies one or more connected unit squares of the grid. The cursor starts in the upper left corner of the keyboard and moves in the four cardinal directions, in such a way that it always skips to the next unit square in that direction that belongs to a different key. If there is no such unit square, the cursor does not move.

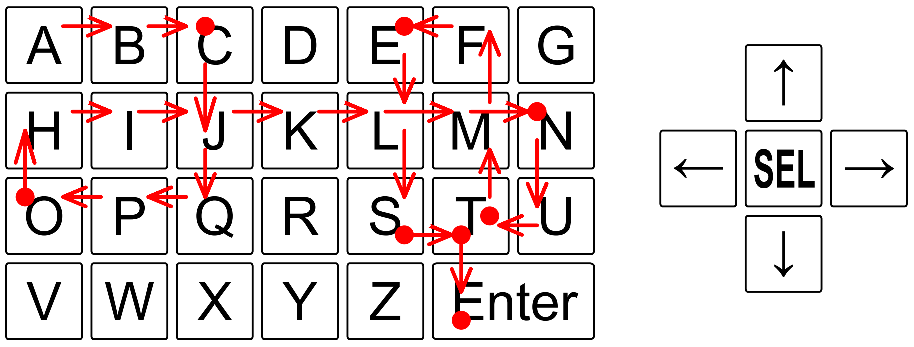

Figure F.1: Sample Input 1. An example virtual keyboard and hardware buttons.

Figure F.1, illustrating Sample Input 1, shows a possible way to type CONTEST using 30 strokes on an example virtual keyboard. The red dots represent the virtual keys where the select button was pressed.

## Input

The first line of the input contains two integers $r$ and $c$ ($1 \leq r, c \leq 50$), giving the number of rows and columns of the virtual keyboard grid. The virtual keyboard is specified in the next $r$ lines, each of which contains $c$ characters. The possible values of these characters are uppercase letters, digits, a dash, and an asterisk (representing Enter). There is only one key corresponding to any given character. Each key is made up of one or more grid squares, which will always form a connected region. The last line of the input contains the text to be typed. This text is a non-empty string of at most 10 000 of the available characters other than the asterisk.

## Output

Display the minimal number of strokes necessary to type the whole text, including the Enter key at the end. It is guaranteed that the text can be typed.

**Sample Input 1**

```
4 7
ABCDEFG
HIJKLMN
OPQRSTU
VWXYZ**
CONTEST
```

**Sample Output 1**

```
30
```

**Sample Input 2**

```
5 20
12233445566778899000
QQWWEERRTTYYUUIIOOPP
-AASSDDFFGGHHJJKKLL*
--ZZXXCCVVBBNNMM--**
--------------------
ACM-ICPC-WORLD-FINALS-2015
```

**Sample Output 2**

```
160
```

**Sample Input 3**

```
2 19
ABCDEFGHIJKLMNOPQZY
X*****************Y
AZAZ
```

**Sample Output 3**

```
19
```

**Sample Input 4**

```
6 4
AXYB
BBBB
KLMB
OPQB
DEFB
GHI*
AB
```

**Sample Output 4**

```
7
```

# Problem G
## Pipe Stream
Time limit: 2 seconds

Your hometown has hired some contractors – including you! – to manage its municipal pipe network. They built the network, at great expense, to supply Flubber to every home in town. Unfortunately, nobody has found a use for Flubber yet, but never mind. It was a Flubber network or a fire department, and honestly, houses burn down so rarely, a fire department hardly seems necessary.

Picture by Nevit via Wikimedia Commons

In the possible event that somebody somewhere decides they want some Flubber, they would like to know how quickly it will flow through the pipes. Measuring its rate of flow is your job.

You have access to one of the pipes connected to the network. The pipe is $l$ meters long, and you can start the flow of Flubber through this pipe at a time of your choosing. You know that it flows with a constant real-valued speed, which is at least $v_1$ meters/second and at most $v_2$ meters/second. You want to estimate this speed with an absolute error of at most $\frac{t}{2}$ meters/second.

Unfortunately, the pipe is opaque, so the only thing you can do is to knock on the pipe at any point along its length, that is, in the closed real-valued range $[0, l]$. Listening to the sound of the knock will tell you whether or not the Flubber has reached that point. You are not infinitely fast. Your first knock must be at least $s$ seconds after starting the flow, and there must be at least $s$ seconds between knocks.

Determine a strategy that will require the fewest knocks, in the worst case, to estimate how fast the Flubber is flowing. Note that in some cases the desired estimation might be impossible (for example, if the Flubber reaches the end of the pipe too quickly).

## Input

The input consists of multiple test cases. The first line of input contains an integer $c$ ($1 \le c \le 100$), the number of test cases. Each of the next $c$ lines describes one test case. Each test case contains the five integers $l$, $v_1$, $v_2$, $t$ and $s$ ($1 \le l, v_1, v_2, t, s \le 10^9$ and $v_1 < v_2$), which are described above.

## Output

For each test case, display the minimal number of knocks required to estimate the flow speed in the worst case. If it might be impossible to measure the flow speed accurately enough, display `impossible` instead.

**Sample Input 1**

```
3
1000 1 30 1 1
60 2 10 2 5
59 2 10 2 5
```

**Sample Output 1**

```
5
3
impossible
```

# Problem H
## Qanat
Time limit: 2 seconds

A *qanat* is an irrigation system widely used to deliver water in hot, arid climates. The technology was originally developed by Persians over 2000 years ago. In Morocco, qanats are known as khettara and are still used today in the southern part of the country.

The basic feature of a qanat is an essentially horizontal channel that brings water from an underground water source to an outlet near a civilization. There is also a shaft known as a *mother well* that rises vertically from the underground water source to the surface of a mountain or hill. Creating such a system is extremely expensive, and was especially so in ancient times, since all of the materials excavated from the channel and mother well must be carried above ground, either through the channel outlet or the top of the mother well. To aid in the construction, there are often one or more additional vertical shafts placed at strategic locations above the underground channel. Although these shafts must also be excavated, they provide a means for lifting additional dirt from the horizontal channel as illustrated in Figure H.1.

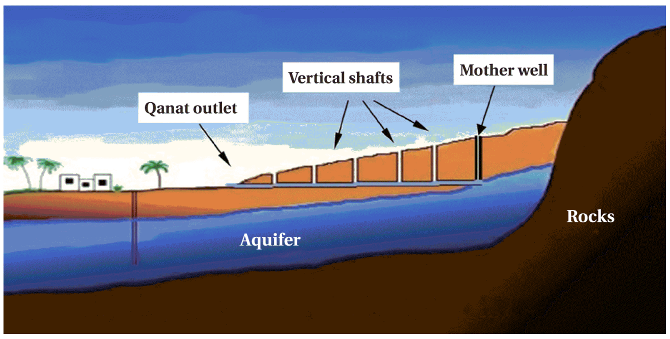

Figure H.1: An illustration of a qanat.

For this problem, model the cross-section of a qanat as shown in Figure H.2, with the channel outlet at $(0, 0)$, the water source at $(w, 0)$, and the top of the mother well at $(w, h)$ with $w > h$. The surface of the mountain extends along a straight line from $(w, h)$ to $(0, 0)$.

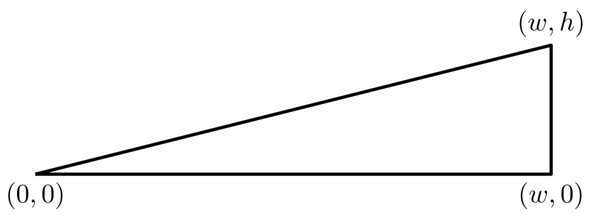

Figure H.2: A simplified model of a qanat cross-section.

Every qanat must have a vertical mother well from the water source to the mountain surface above, along with $n$ additional vertical shafts. The channel and all shafts are modeled as line segments. Your goal is to determine the placement for those additional shafts so as to minimize the overall excavation cost. This cost is equal to the sum of the distances that each piece of excavated dirt must be transported to reach the surface (using any combination of horizontal and vertical movement). For example, the cost of excavating a continuous section of dirt starting from the surface and going along a path of length $\ell$ (possibly including turns) is $\int_0^\ell x\,dx = \frac{1}{2}\ell^2$.

## Input

The input consists of a single line containing three integers $w$ ($1 \leq w \leq 10\,000$), $h$ ($1 \leq h < w$), and $n$ ($1 \leq n \leq 1\,000$). The value $w$ is the horizontal distance from the water source to the qanat outlet. The value $h$ is the vertical distance from the water source to the mountain surface. The value $n$ is the number of vertical shafts that must be used in addition to the mother well.

## Output

First, display the minimum overall excavation cost. Next, display the $x$-coordinates, in increasing order, for $n$ optimally placed vertical shafts. If $n > 10$, display only the first 10 $x$-coordinates. Answers within an absolute or relative error of $10^{-4}$ will be accepted. You may assume that there is a unique solution. No test case will result in a shaft within 0.001 units from the outlet of the qanat channel or from another shaft.

**Sample Input 1**

```
8 4 1
```

**Sample Output 1**

```
31.500000
3.000000
```

**Sample Input 2**

```
195 65 2
```

**Sample Output 2**

```
12220.000000
48.000000
108.000000
```

**Sample Input 3**

```
10000 1 1000
```

**Sample Output 3**

```
30141.885677
9.956721
19.913443
29.870164
39.826887
49.783610
59.740334
69.697060
79.653786
89.610515
99.567245
```

# Problem I
## Ship Traffic
Time limit: 3 seconds

Ferries crossing the Strait of Gibraltar from Morocco to Spain must carefully navigate to avoid the heavy ship traffic along the strait. Write a program to help ferry captains find the largest gaps in strait traffic for a safe crossing.

Your program will use a simple model as follows. The strait has several parallel shipping lanes in east-west direction. Ships run with the same constant speed either eastbound or westbound. All ships in the same lane run in the same direction. Satellite data provides the positions of the ships in each lane. The ships may have different lengths. Ships do not change lanes and do not change speed for the crossing ferry.

The ferry waits for an appropriate time when there is an adequate gap in the ship traffic. It then crosses the strait heading northbound along a north-south line at a constant speed. From the moment a ferry enters a lane until the moment it leaves the lane, no ship in that lane may touch the crossing line. Ferries are so small you can neglect their size. Figure I.1 illustrates the lanes and ships for Sample Input 1. Your task is to find the largest time interval within which the ferry can safely cross the strait.

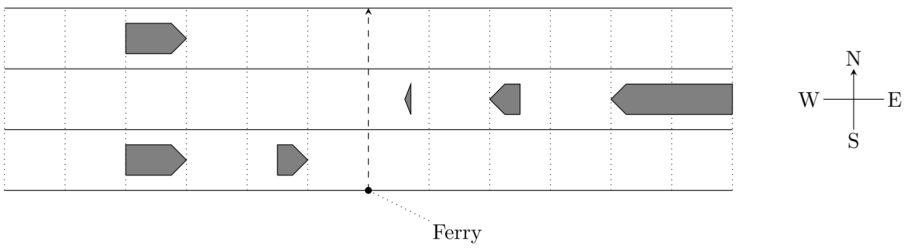

Figure I.1: Sample Input 1.

## Input

The first line of input contains six integers: the number of lanes $n$ ($1 \le n \le 10^5$), the width $w$ of each lane ($1 \le w \le 1\,000$), the speed $u$ of ships and the speed $v$ of the ferry ($1 \le u, v \le 100$), the ferry's earliest start time $t_1$ and the ferry's latest start time $t_2$ ($0 \le t_1 < t_2 \le 10^6$). All lengths are given in meters, all speeds are given in meters/second, and all times are given in seconds.

Each of the next $n$ lines contains the data for one lane. Each line starts with either `E` or `W`, where `E` indicates that ships in this lane are eastbound and `W` indicates that ships in this lane are westbound. Next in the line is an integer $m_i$, the number of ships in this lane ($0 \le m_i \le 10^5$ for each $1 \le i \le n$). It is followed by $m_i$ pairs of integers $l_{ij}$ and $p_{ij}$ ($1 \le l_{ij} \le 1\,000$ and $-10^6 \le p_{ij} \le 10^6$). The length of ship $j$ in lane $i$ is $l_{ij}$, and $p_{ij}$ is the position at time 0 of its forward end, that is, its front in the direction it moves.

Ship positions within each lane are relative to the ferry's crossing line. Negative positions are west of the crossing line and positive positions are east of it. Ships do not overlap or touch, and are sorted in increasing order of their positions. Lanes are ordered by increasing distance from the ferry's starting point, which is just south of the first lane. There is no space between lanes. The total number of ships is at least 1 and at most $10^5$.

## Output

Display the maximal value $d$ for which there is a time $s$ such that the ferry can start a crossing at any time $t$ with $s \leq t \leq s + d$. Additionally the crossing must not start before time $t_1$ and must start no later than time $t_2$. The output must have an absolute or relative error of at most $10^{-3}$. You may assume that there is a time interval with $d > 0.1$ seconds for the ferry to cross.

**Sample Input 1**

```
3 100 5 10 0 100
E 2 100 -300 50 -100
W 3 10 60 50 200 200 400
E 1 100 -300
```

**Sample Output 1**

```
6.00000000
```

**Sample Input 2**

```
1 100 5 10 0 200
W 4 100 100 100 300 100 700 100 900
```

**Sample Output 2**

```
50.00000000
```

# Problem J
## Tile Cutting
Time limit: 15 seconds

Youssef is a Moroccan tile installer who specializes in mosaics like the one shown on the right. He has rectangular tiles of many dimensions at his disposal, and the dimensions of all his tiles are integer numbers of centimeters. When Youssef needs parallelogram-shaped tiles, he cuts them from his supply on hand. To make this work easier, he invented a tile cutting machine that superimposes a centimeter grid on the cutting surface to guide the cuts on the tiles. Due to machine limitations, aesthetic sensibilities, and Youssef's dislike of wasted tiles, the following rules determine the possible cuts.

1. The rectangular tile to be cut must be positioned in the bottom left corner of the cutting surface and the edges must be aligned with the grid lines.
2. The cutting blade can cut along any line connecting two different grid points on the tile boundary as long as the points are on adjacent boundary edges.
3. The four corners of the resulting parallelogram tile must lie on the four sides of the original rectangular tile.
4. No edge of the parallelogram tile can lie along an edge of the rectangular tile.

Figure J.1 shows the eight different ways in which a parallelogram tile of area 4 square centimeters can be cut out of a rectangular tile, subject to these restrictions.

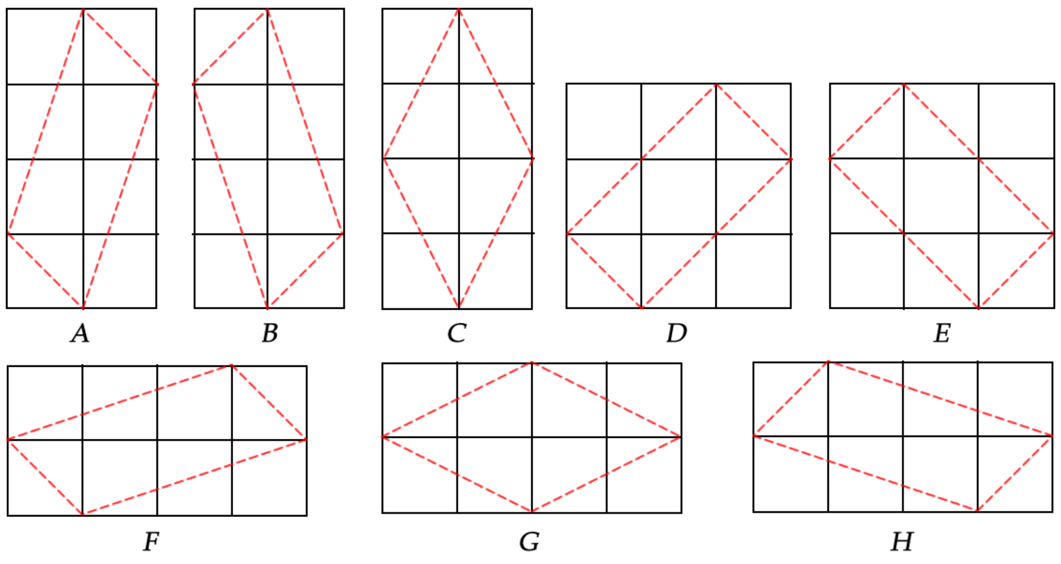

Figure J.1: The eight different ways for cutting a parallelogram of area 4.

Youssef needs to cut tiles of every area between $a_{\text{lo}}$ and $a_{\text{hi}}$. Now he wonders, for which area $a$ in this range can he cut the maximum number of different tiles?

## Input

The input consists of multiple test cases. The first line of input contains an integer $n$ ($1 \leq n \leq 500$), the number of test cases. The next $n$ lines each contain two integers $a_{\text{lo}}, a_{\text{hi}}$ ($1 \leq a_{\text{lo}} \leq a_{\text{hi}} \leq 500\,000$), the range of areas of the tiles.

## Output

For each test case $a_{\text{lo}}$, $a_{\text{hi}}$, display the value $a$ between $a_{\text{lo}}$ and $a_{\text{hi}}$ such that the number of possible ways to cut a parallelogram of area $a$ is maximized as well as the number of different ways $w$ in which such a parallelogram can be cut. If there are multiple possible values of $a$ display the smallest one.

| Sample Input 1 | Sample Output 1 |
|---|---|
| 2<br>4 4<br>2 6 | 4 8<br>6 20 |

# Problem K
## Tours
Time limit: 3 seconds

The Arca Carania Mountain national park is opening up for tourist traffic. The national park has a number of sites worth seeing and roads that connect pairs of sites. The park commissioners have put together a set of *round tours* in the park in which visitors can ride buses to view various sites. Each round tour starts at some site (potentially different sites for different tours), visits a number of other sites without repeating any, and then returns to where it started. At least 3 different sites are visited in each round tour. At least one round tour is possible in the national park.

The park commissioners have decided that, for any given road, all buses will be operated by a single company. The commissioners do not want to be accused of favoritism, so they want to be sure that each possible round tour in the park has exactly the same number of roads assigned to each bus company. They realize this may be difficult to achieve. Thus, they want to learn what numbers of bus companies allow for a valid assignment of companies to roads.

Consider Sample Input 1, which is illustrated in Figure K.1. There are a total of three round tours for these sites. Some company is assigned road 1-3. It must also be assigned some road on the round tour 1-2-3-4-1, say 2-3. But then it is assigned to two of the three roads on the round tour 1-2-3-1, and no other company can match this – so there can be no other companies. In Sample Input 2 there is only one round tour, so it is enough to assign the roads of this tour equally between companies.

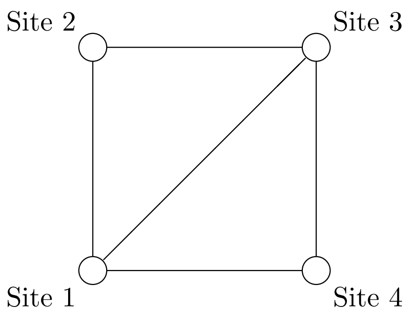

Figure K.1: Sample Input 1.

### Input

The first line of input contains two integers $n$ ($1 \leq n \leq 2\,000$), which is the number of sites in the park, and $m$ ($1 \leq m \leq 2\,000$), which is the number of roads between the sites. Following that are $m$ lines, each containing two integers $a_i$ and $b_i$ ($1 \leq a_i < b_i \leq n$), meaning the sites $a_i$ and $b_i$ are connected by a bidirectional road. No pair of sites is listed twice.

### Output

Display all integers $k$ such that it is possible to assign the roads to $k$ companies in the desired way. These integers should be in ascending order.

**Sample Input 1**

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

**Sample Output 1**

```
1
```

**Sample Input 2**

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

**Sample Output 2**

```
1 3
```

# Problem L
## Weather Report
Time limit: 2 seconds

You have been hired by the Association for Climatological Measurement, a scientific organization interested in tracking global weather trends over a long period of time. Of course, this is no easy task. They have deployed many small devices around the world, designed to take periodic measurements of the local weather conditions. These are cheap devices with somewhat restricted capabilities. Every day they observe which of the four standard kinds of weather occurred: *Sunny*, *Cloudy*, *Rainy*, or *Frogs*. After every $n$ of these observations have been made, the results are reported to the main server for analysis. However, the massive number of devices has caused the available communication bandwidth to be overloaded. The Association needs your help to come up with a method of compressing these reports into fewer bits.

For a particular device's location, you may assume that the weather each day is an independent random event, and you are given the predicted probabilities of the four possible weather types. Each of the $4^n$ possible weather reports for a device must be encoded as a unique sequence of bits, such that no sequence is a prefix of any other sequence (an important property, or else the server would not know when each sequence ends). The goal is to use an encoding that minimizes the expected number of transmitted bits.

## Input

The first line of input contains an integer $1 \leq n \leq 20$, the number of observations that go into each report. The second line contains four positive floating-point numbers, $p_{\text{sunny}}$, $p_{\text{cloudy}}$, $p_{\text{rainy}}$, and $p_{\text{frogs}}$, representing the respective weather probabilities. These probabilities have at most 6 digits after the decimal point and sum to 1.

## Output

Display the minimum expected number of bits in the encoding of a report, with an absolute or relative error of at most $10^{-4}$.

**Sample Input 1**

```
2
0.9 0.049999 0.05 0.000001
```

**Sample Output 1**

```
1.457510
```

**Sample Input 2**

```
20
0.25 0.25 0.25 0.25
```

**Sample Output 2**

```
40.000000
```

This page is intentionally left blank.

# Problem M
## Window Manager
Time limit: 2 seconds

The past few years have seen a revolution in user interface technology. For many years, keyboards and mice were *the* tools used to interact with computers. But with the introduction of smart phones and tablets, people are increasingly using their computers by tapping and moving their fingers on the screen. Naturally this has led to new paradigms in user interface design. One important principle is that objects on the display obey "physical" laws. In this problem, you will see an example of this.

You have been hired to build a simulator for the window manager to be used in the next generation of smart phones from Advanced Cellular Manufacturers (ACM). Each phone they produce will have a rectangular screen that fully displays zero or more rectangular windows. That is, no window exceeds the boundaries of the screen or overlaps any other window. The simulator must support the following commands.

- `OPEN` $x$ $y$ $w$ $h$ — open a new window with top-left corner coordinates $(x, y)$, width $w$ pixels and height $h$ pixels.
- `CLOSE` $x$ $y$ — close an open window that includes the pixel at $(x, y)$. This allows a user to tap anywhere on a window to close it.
- `RESIZE` $x$ $y$ $w$ $h$ — set the dimensions of the window that includes the pixel at $(x, y)$ to width $w$ and height $h$. The top-left corner of the window does not move.
- `MOVE` $x$ $y$ $d_x$ $d_y$ — move the window that includes the pixel at $(x, y)$. The movement is either $d_x$ pixels in the horizontal direction or $d_y$ pixels in the vertical direction. At most one of $d_x$ and $d_y$ will be non-zero.

The `OPEN` and `RESIZE` commands succeed only if the resulting window does not overlap any other windows and does not extend beyond the screen boundaries. The `MOVE` command will move the window by as many of the requested pixels as possible. For example, if $d_x$ is 30 but the window can move only 15 pixels to the right, then it will move 15 pixels.

ACM is particularly proud of the `MOVE` command. A window being moved might "bump into" another window. In this case, the first window will push the second window in the same direction as far as appropriate, exactly as if the windows were physical objects. This behavior can cascade – a moving window might encounter additional windows which are also pushed along as necessary. Figure M.1 shows an example with three windows, where window A is moved to the right, pushing the other two along.

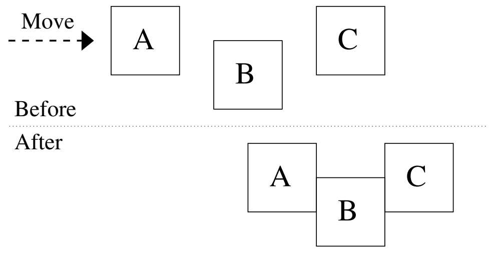

Figure M.1: `MOVE` example

## Input

The first line of input contains two positive integers $x_{\max}$ and $y_{\max}$, the horizontal and vertical dimensions of the screen, measured in pixels. Each is at most $10^9$ (ACM is planning on building displays with very high resolution). The top-left pixel of the screen has coordinates $(0, 0)$. Each of the following lines contains a command as described above. One or more spaces separate the command name and the parameters from each other. The command parameters are integers that satisfy these conditions: $0 \leq x < x_{\max}$, $0 \leq y < y_{\max}$, $1 \leq w, h \leq 10^9$, and $|d_x|, |d_y| \leq 10^9$. There will be at most 256 commands.

## Output

The output must follow the format illustrated in the sample output below.

Simulate the commands in the order they appear in the input. If any errors are detected during a command's simulation, display the command number, command name, and the first appropriate message from the following list, and ignore the results of simulating that command (except as noted).

- `no window at given position` — for the `CLOSE`, `RESIZE`, and `MOVE` commands — if there is no window that includes the pixel at the specified position.
- `window does not fit` — for the `OPEN` and `RESIZE` commands — if the resulting window would overlap another window or extend beyond the screen boundaries.
- `moved` $d'$ `instead of` $d$ — for the `MOVE` command — if the command asked to move a window $d$ pixels, but it could only move $d'$ pixels before requiring a window to move beyond the screen boundaries. The values $d$ and $d'$ are the absolute number of pixels requested and moved, respectively. The window is still moved in this case, but only for the smaller distance.

After all commands have been simulated and any error messages have been displayed, indicate the number of windows that are still open. Then for each open window, in the same order that they were opened, display the coordinates of the top-left corner $(x, y)$, the width, and the height.

**Sample Input 1**

```
320 200
OPEN 50 50 10 10
OPEN 70 55 10 10
OPEN 90 50 10 10
RESIZE 55 55 40 40
RESIZE 55 55 15 15
MOVE 55 55 40 0
CLOSE 55 55
CLOSE 110 60
MOVE 95 55 0 -100
```

**Sample Output 1**

```
Command 4: RESIZE - window does not fit
Command 7: CLOSE - no window at given position
Command 9: MOVE - moved 50 instead of 100
2 window(s):
90 0 15 15
115 50 10 10
```