A. Brookebond s’en va en guerre…

Famous military leader marshal Brookebond who has never lost a battle (truth to be told, he has never won one either) is sincerely convinced that all military operations can be planned on the globe. Let’s not reveal the depth of his misconceptions to the poor marshal. Instead, let’s help him by writing a program to compute the distance between two points on the surface of the Earth given in the geographic coordinates. An order from the marshal Brookebond declares the Earth to be a perfect sphere having radius of 6370 kilometers. And the orders, as we all know, are not to be discussed…

Geographic latitude and longitude are measured in degrees and minutes with the accuracy to one minute (one degree containing 60 minutes, of course). The latitude is measured from 90 degrees of northern latitude (**N**) for the North Pole to 90 degrees of southern latitude (**S**) for the South Pole; the latitude of any point on the equator is 0 degrees (**N** or **S**, does not matter). The longitude of is measured from 180 degrees of eastern longitude (**E**) to 180 degrees of western longitude (**W**); for point on meridians with longitude 180 or 0, E and W are equivalent.

Input data is in the text file GLOBE.IN. The first and second line of the file contain the coordinates (latitude and longitude) of one point each. The designation of a latitude contains two integers (degrees and minutes) and a letter **N** or **S**. Similarly, the designation of a longitude contains two integers (degrees and minutes) and a letter **E** or **W**. Adjacent values on a line are separated by one or more spaces.

Output data must be written to the text file GLOBE.OUT. The first and only line of the file must contain the distance (in kilometers) between the points from the input file with the precision of 1 meter.

Example input
```
55 0 N 40 0 E
59 0 N 49 30 E
```

Example output
```
725.979
```

B. The Domino Principle

There is a configuration of **N** domino stones that is to be beautifully destroyed. For each stone **i** in the given configuration, we know the set $\mathbf{D_i}$ of the stones that will fall at time **t**+1 (if they have not fallen before) if the stone **i** falls at time **t**.

Write a program to determine which stone should be pushed to fall at the time 0 to achieve complete destruction of the configuration that takes the longest possible time.

Input data is in the text file DOMINO.IN. The first line of the file contains the number of stones **N** in the configuration ($1 \le \mathbf{N} \le 1000$). Next **N** lines describe sets $\mathbf{D_i}$ ($1 \le \mathbf{i} \le \mathbf{N}$). Each of these lines contains first the potency of the set $\mathbf{D_i}$ and then the numbers of stones in the set separated by one or more spaces.

Output data should be written to text file DOMINO.OUT. The first line of the file must contain the time of fall of the last stone in the configuration, and the second line - the number of the stone that should be pushed first. If the maximal time can be achieved starting from several initial stones, output the maximal possible number. If the configuration can’t be destroyed completely, the first and only line of the file must contain the word ‘**impossible**’.

Example input
```
3
2 2 3
1 3
0
```

Example output
```
1
1
```

C. Broken line

There are **K** line segments on a plane. These are all segments of a broken line, in arbitrary order. Write a program to check whether the broken line is closed.

Input data is in the text file LINE.IN with (4***K**+1) lines, each of them consists of no more then 250 characters. The first line of the file contains the number of segments **K** (1≤**K**≤3000). Each segment is described by the coordinates **X'**, **Y'**, **X''**, and **Y''** of its end-points. All coordinates are non-negatives integers and written in separate lines (so, the coordinates of one segment are in four sequential lines).

Output data must be written to the text file LINE.OUT. The first and only line of the file must contain the number 1 if the broken line is closed, and 0 otherwise.

Example input
```
3
0
0
0
1
1
0
0
0
1
0
0
1
```

Example output
```
1
```

D. Split convex polygon

Two polygons on a plane are described by the coordinates of their vertices in anticlockwise order. The polygons are otherwise arbitrary but it is known that no three sequential vertices lie on the same line. It is suspected that the two polygons are parts of one bigger **convex** polygon that was split in two.

Write a program to check the guess on the condition that it is allowed to move and rotate, but not to invert the parts to match them together. All computations should be made with the precision of 0.001.

Input data is in the text file POLY.IN. The first line contains an integer $\mathbf{N_1}$ ($3 \leq \mathbf{N_1} \leq 100$) – the number of vertices of the first polygon. Each of the following $\mathbf{N_1}$ lines contains two real numbers **X**, **Y** separated by one space – the coordinates of one vertex. The rest of the file describes the second polygon in similar manner.

Output data must be written to the text file POLY.OUT. The first and only line of the file must contain the number 1 if the guess is correct, and 0 otherwise.

Example input
```
7
0 0
4 0
4 1
3 3.5
4 4
4 5
0 5
7
6 0
11 0
11 4
10 4
9.5 5
7 4
6 4
```
Example output
```
1
```

E. Paid Roads

A network of **m** roads connects **N** cities (numbered from 1 to **N**). There may be more than one road connecting one city with another. Some of the roads are paid. There are two ways to pay for travel on a paid road **i** from city $\mathbf{a_i}$ to city $\mathbf{b_i}$:

- in advance, in a city $\mathbf{c_i}$ (which may or may not be the same as $\mathbf{a_i}$);
- after the travel, in the city $\mathbf{b_i}$.

The payment is $\mathbf{P_i}$ in the first case and $\mathbf{R_i}$ in the second case.

Write a program to find a minimal-cost route from the city 1 to the city **N**.

Input data is in the text file TRIP.IN. The first line of the file contains the values of **N** and **m**. Each of the following **m** lines describes one road by specifying the values of $\mathbf{a_i}$, $\mathbf{b_i}$, $\mathbf{c_i}$, $\mathbf{P_i}$, $\mathbf{R_i}$ ($1 \le \mathbf{i} \le \mathbf{m}$). Adjacent values on the same line are separated by one or more spaces. All values are integers, $1 \le \mathbf{m}, \mathbf{N} \le 10$, $0 \le \mathbf{P_i}, \mathbf{R_i} \le 100$, $\mathbf{P}_i \le \mathbf{R_i}$ ($1 \le \mathbf{i} \le \mathbf{m}$).

Output data must be written to the text file TRIP.OUT. The first and only line of the file must contain the minimal possible cost of a trip from the city 1 to the city **N**. If the trip is not possible for any reason, the line must contain the word '**impossible**'.

Example input
```
4 5
1 2 1 10 10
2 3 1 30 50
3 4 3 80 80
2 1 2 10 10
1 3 2 10 50
```

Example output
```
110
```

F. Ray

There is a structure consisting of **K** homogenous slices of equal thickness. The refraction index of the **i**-th slice is $\mathbf{N_i}$. A ray of light travels in the first slice at the angle α to the normal. Write a program to compute the angle that the ray will have to the normal in the last slice.

It is known that the formula $N_1\sin\alpha=N_2\sin\beta$ holds in a two-sliced structure with the refraction indices of the slices equal to N_1 and N_2, respectively, when a ray of light passes from the first slice to the second, traveling at the angle α to the normal in the first slice, and at the angle β to the normal in the second slice.

Input data is in the text file RAY.IN. On the first line of the file is the number of slices **K** ($1\le\mathbf{K}\le100$). Each of the following **K** lines contains the refraction index $\mathbf{N_i}$ of the corresponding slice. All the refraction indices are real numbers in the range from 0.000001 to 2000. The last line of the file contains the angle α between the ray and the normal in the first slice. The angle is a real number in the range from 0 to π /2 and is given in radians.

Output data must be written to the text file RAY.OUT. The first and only line of the file must contain the angle β between the ray and the normal in the last slice. The answer must be given with the precision of 0.001. If the ray does not reach the last slice, the output must be the word '**NO**'.

An example of the input data
```
2
1
2
0.02
```

An example of the output data
```
0.01
```

G. RPG

To complete a role-playing game (RPG), you must send your character through **N** quests. The quests may be passed in any order, but after starting a quest you can't drop it and switch to another. If your character fails even one quest you fail the whole game. The probability of succeeding in a quest depends on the experience points (**XP**) of your character at the beginning of the quest, and is determined by the formula

$$\begin{cases} 0, & \text{if } XP < a_i \\ \dfrac{XP - a_i}{b_i - a_i}, & \text{if } a_i \le XP < b_i \\ 1, & \text{if } XP \ge b_i \end{cases}$$ where $\mathbf{a_i}$ and $\mathbf{b_i}$ are parameters of the **i**-th quest.

After successful completion of the **i**-th quest, your character obtains $\mathbf{S_i}$ experience points. Initially, it/he/she has **D** points.

Write a program to determine such an order of the quests that the probability of completing the whole game successfully is the highest possible.

Input data is in the text file RPG.IN. The first line of the file contains the values **N** ($1 \le \mathbf{N} \le 10$) and **D**. Each of the following **N** lines describes one quest, containing the values $\mathbf{a_i}$, $\mathbf{b_i}$, $\mathbf{S_i}$. The values **D**, $\mathbf{a_i}$, $\mathbf{b_i}$, $\mathbf{S_i}$ ($1 \le \mathbf{i} \le \mathbf{N}$) are all integers in the range from 0 to 1000 and $\mathbf{a_i} < \mathbf{b_i}$ ($1 \le \mathbf{i} \le \mathbf{N}$). Adjacent values on the same line are separated by one or more spaces.

Output data must be written to the text file RPG.OUT. The first line of the file must contain the highest possible probability with the precision of 0.001. The second line must contain the order of quests that ensures this probability. If there are several orders with the same probability, output any one of them.

Example input
```
3 300
350 380 100
100 200 100
440 450 100
```

Example output
```
1.000
2 1 3
```

H. Pots

You are given two pots, having the volume of **A** and **B** liters respectively. The following operations can be performed:

1) FILL(i) fill the pot **i** ($1 \le \mathbf{i} \le 2$) from the tap;
2) DROP(i) empty the pot **i** to the drain;
3) POUR(i,j) pour from pot **i** to pot **j**; after this operation either the pot **j** is full (and there may be some water left in the pot **i**), or the pot **i** is empty (and all its contents have been moved to the pot **j**).

Write a program to find the shortest possible sequence of these operations that will yield exactly **C** liters of water in one of the pots.

Input data is in the text file POTS.IN. On the first and only line are the numbers **A**, **B**, and **C**. These are all integers in the range from 1 to 100 and $\mathbf{C} \le \max(\mathbf{A},\mathbf{B})$.

Output data should be written to the text file POTS.OUT. The first line of the file must contain the length of the sequence of operations **K**. The following **K** lines must each describe one operation. If there are several sequences of minimal length, output any one of them. If the desired result can't be achieved, the first and only line of the file must contain the word '**impossible**'.

Example input
```
3 5 4
```

Example output
```
6
FILL(2)
POUR(2,1)
DROP(1)
POUR(2,1)
FILL(2)
POUR(2,1)
```