German Collegiate Programming Contest 2019

July 6th

Problems

This page is intentionally left (almost) blank.

Problem A: Assessing Genomes

The world is at the brink of extinction. A mutated virus threatens to destroy all living organisms. As a last hope, a team of super-smart scientists, including – of course – you, is currently working on an antivirus. Unfortunately, your team is unable to analyse the DNA in time. They sequenced n parts of the virus' DNA and need to match them with n available strands for antiviruses. As the algorithms expert, you need to implement a specialised procedure to solve this problem. Your approach needs to be fast – there is not much time left!

You first need to determine the *repetition score* of each DNA sequence. The repetition score of a sequence s is equal to the length of the shortest sequence u such that s is equal to the k-fold repetition of u, for some positive integer k. For instance, `ATGATG` has a repetition score of 3, since it can be produced by repeating `ATG` two times. On the other hand, `ATATA` has a repetition score of 5, as it cannot be produced from any proper substring.

Once you obtained the scores of all sequences, you need to match the n antivirus sequences with the n virus sequences in a way that minimises the damage caused by the virus. When two sequences are matched, the damage caused by the virus is equal to the squared difference between the two repetition scores. For instance, matching the antivirus sequence `ATGATG` with the virus sequence `ATATA` causes $(3-5)^2 = 4$ units of damage.

If you match the DNA sequences optimally, what is the minimal total damage caused by the virus, taken as a sum over all matched pairs?

Input

The input consists of:

- A line with an integer n ($1 \leq n \leq 50$), the number of DNA sequences of the virus and antivirus each.
- n lines, each with a virus DNA sequence.
- n lines, each with an antivirus DNA sequence.

Each DNA sequence is a non-empty string with a length of at most 250 and consists of lowercase letters `a-z` and uppercase letters `A-Z`.

Output

Output one integer, the minimal total damage.

Sample Input 1

```
2
TTTTTT
TATG
TATATA
AAAGAAAG
```

Sample Output 1

```
1
```

Sample Input 2

```
3
abcdef
aaa
bab
AbAb
xyzxyz
X
```

Sample Output 2

```
10
```

This page is intentionally left (almost) blank.

Problem B: Bouldering

After a few particularly long afternoons of procrastinating in his box, playing video games all night long, Carl decided it was finally time to start his New Year's Resolution – going to the bouldering gym.

There, he took to one of the easier walls and tried to make his way up. Unfortunately, he could never quite reach the top, as he would always run out of stamina and fall down.

While climbing, he noticed the holds all have different shapes with some of them being much harder to hold than others, so gripping them uses up different amounts of stamina. Frustrated, he asks one of the regulars at the bouldering hall how to scale the wall – you. Show him the shortest way up that he can take without running out of stamina.

The bouldering wall is a rectangular grid of cells of size 1×1 where holds can be installed. For this problem we do not consider the varying sizes of the holds, so you can assume them to be the shape of a singular point exactly in the middle of the cell. Carl can only move from one hold to another if their distance (the Euclidean distance between the centres of the cells) does not exceed his arms' reach.

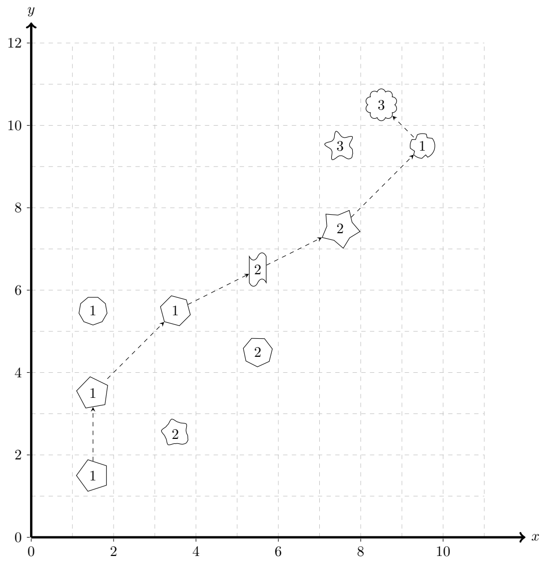

Figure B.1: Sample test case 1

Input

The input consists of:

- A line with four integers h, w, r and s ($2 \leq h \leq 25, 1 \leq w \leq 25, 1 \leq r \leq 3, 1 \leq s \leq 10^9$) where h and w are the height and width of the bouldering wall, r is the reach of Carl's arms and s is a numerical representation of Carl's stamina.
- h lines, each with w characters, describing the holds on the bouldering wall. Each character is either a digit c ($1 \leq c \leq 9$), which means that a hold with difficulty c is installed at this position, or "`.`", which means there is no hold installed.

The first line corresponds to the top of the bouldering wall and the last line to the bottom.

A sequence of holds is a valid route for Carl if the following conditions are satisfied:

- The route starts at the bottommost hold and ends at the topmost hold. There will always be a unique bottommost and a unique topmost hold, and these are guaranteed to be distinct.
- The sum of difficulty levels of the used holds is at most s.
- The Euclidean distance between any two consecutive holds along the route is at most r.

Output

Output the total length of a shortest route Carl can climb to reach the topmost hold without running out of stamina. Your answer should have an absolute or relative error of at most 10^{-6}. If it is not possible for Carl to reach the top, output `impossible`.

Sample Input 1

```
12 11 3 11
...........
........3..
.......3.1.
...........
.......2...
.....2.....
.1.1.......
.....2.....
.1.........
...2.......
.1.........
...........
```

Sample Output 1

```
13.543203766865055
```

Sample Input 2

```
8 16 3 15
......1.........
....1..1.1......
..2........1....
...2......1.....
.....4.1..2..1..
................
.......1........
................
```

Sample Output 2

```
6.414213562373095
```

Sample Input 3

```
10 10 2 10
...2......
..........
...5.2....
..........
.....3....
....5.....
..2....2..
..1.......
....2.....
..1.......
```

Sample Output 3

```
impossible
```

Problem C: Colourful Chameleons

When Christian is not doing sophisticated research in software validation, he is breeding chameleons. His chameleons are of a special species and only occur in one of n distinct colours. Unlike their counterparts in nature, Christian's chameleons change their colour only under very specific circumstances. It took Christian quite a long time to develop his so-called *Crazy Chameleon Colouring Concept* (C^4) which works as follows. If he puts $n-1$ chameleons with distinct colours in a special breeding terrarium, gives them strawberry cheese and liver sausage to eat, and lets them stay in the dark over night, then there will be y chameleons in the terrarium the next morning. The crazy thing is that none of the y chameleons has any of the $n-1$ original colours. Instead, all of them take on the colour that was not present in the breeding terrarium initially.

Last weekend Christian let a friend borrow his chameleons for the Christopher Street Day. Of course his friend wanted the chameleons to look as colourful as possible. That's why Christian applied the C^4 method in a way such that there were at least $n-1$ chameleons available of every single colour. Therefore, Christian currently has $x_i \geq n-1$ chameleons of the ith colour. However, the next holiday coming up is St. Patrick's Day and Christian thinks that it would be super cool if all his chameleons took on the same colour for this special event. He is wondering whether such a state can be reached by exclusively applying the C^4 method. Because he is running low on strawberry cheese, he would like to know the minimal number of C^4 applications needed for all his chameleons to take on the same colour - provided that this is even possible.

Input

The input consists of:

- A line with three integers n, c, and y:
  - n ($2 \leq n \leq 10^5$), the number different colours that the Chameleons can take on.
  - c ($1 \leq c \leq n$), the colour that all chameleons should have in the end.
  - y ($n-1 \leq y \leq 10^9$), the number of chameleons in the breeding terrarium after applying the C^4 method.
- A line with n integers $x_1, \ldots, x_n$ ($n-1 \leq x_i \leq 10^9$ for all i), where x_i is the number of chameleons of the ith colour that Christian possesses initially.

Output

If it is impossible for Christian's chameleons to all take on colour c by exclusively applying the C^4 method, output `impossible`. Otherwise output two integer numbers a and b, where a is the minimal number of C^4 applications necessary for all chameleons to have colour c, and b is the total number of chameleons that Christian possesses in the end.

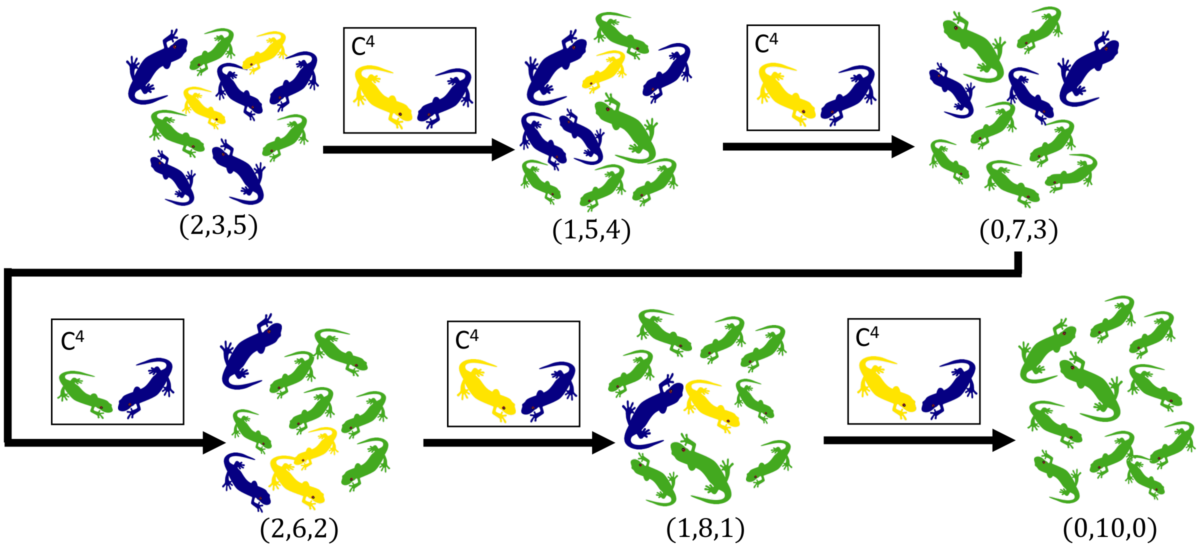

Figure C.1: Illustration of the first sample case. Initially, Christian owns 2 yellow, 3 green, and 5 blue chameleons. After 5 applications of the C^4 method he has 10 green chameleons. The vectors (y, g, b) denote the number of yellow, green, and blue chameleons available in the respective breeding step.

Sample Input 1

```
3 2 2
2 3 5
```

Sample Output 1

```
5 10
```

Sample Input 2

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

Sample Output 2

```
impossible
```

Problem D: Dungeon Crawler

In the famous video game Wizards & Wyvern, the protagonist finds himself in some kind of maze structure and has to find a way out. Each level is constructed with several caves, platforms, rooms or other kind of interesting spots that are connected with paths of different types. In each spot, there is a single distinguishable and unique artefact that can be activated by the hero. To escape from one level, the player has to activate the correct artefacts in the correct order. Riddles hidden inside the maze help him to solve the level.

You hate riddles and therefore obtained a map of a single level and its solution from the internet. You only need to find out whether this map represents the current level you are in.

Overview of the first sample case.

While wandering around from spot to spot, the player can see the artefact in his current spot as well as all paths leading away from it. The game implements 26 different types of paths such as paved roads (`P`), bridges (`B`) or dirt roads (`D`). In each spot all paths leading away are of different types.

This is an interactive problem. You do not know your initial position but you can always decide which path to follow. For each spot you visit, you are provided with the name of the artefact there and the types of paths leading away. Your task is to find out whether this level matches the one on your map.

Interaction Protocol

Your submission will be interacting with a special program called the *grader*. This means that the output of your submission is sent to the grader and the output of the grader is sent to the standard input of your submission. This interaction must follow a specific protocol:

The grader starts with the description of your map:

- One integer n ($2 \leq n \leq 1\,000$), the number of spots on the map.
- n lines, the ith of which contains an integer k followed by k pairs $t_1\ m_1, \ldots, t_k\ m_k$, describing the paths leading away from spot i. Each path is given by its type t_j (t_j is an uppercase letter) and destination m_j ($1 \leq m_j \leq n, m_j \neq i$).

All paths are bidirectional and appear twice in the input, once for each side. You can safely assume that all spots are reachable from any position in the maze.

Then, the game proceeds in turns. In each turn, the grader sends you the description of your current spot, in the following format:

- One line with a lowercase string a ($1 \leq |a| \leq 20$) and an uppercase string s ($1 \leq |s| \leq 26$), the name of the artefact you see at this spot and the description of the paths leading away. No two spots will have the same artefact. All letters in s are distinct and uppercase with no particular order. The order can vary when entering the same spot again.

Your submission must respond with one of the following:

- `W` c – you want to walk along the path of type c, where c is a valid uppercase character.
- `R` i – you are sure the level corresponds to the map. The integer i ($1 \leq i \leq n$) should be the number of the spot on the initial map that you believe you are currently in.
- `R ambiguous` – the map matches the level, but the matching is not unique.
- `R no` – the level does not correspond to your map.

Continue on next page ...

After every valid response starting with `W`, the grader gives you the description of the new spot you are currently in. You are not allowed to do more than 250 000 walks. After a response starting with `R`, the game ends. When you send an invalid response, the game ends and your submission will be judged as “Wrong Answer”. After each command you send, you should *flush* the standard output to ensure that it is sent to the game. For example, you can use `fflush(stdout)` in C++, `System.out.flush()` in Java, and `sys.stdout.flush()` in Python.

Sample Input 1

```
3
2 D 2 B 3
2 P 3 D 1
2 P 2 B 1
fountain DB

obelisk PD

crystals PB

fountain DB
```

Sample Output 1

```





W D

W P

W B

R 1
```

Sample Input 2

```
3
2 D 2 B 3
2 P 3 D 1
2 P 2 B 1
obelisk PD

fountain LD
```

Sample Output 2

```





W D

R no
```

Sample Input 3

```
2
2 P 2 D 2
2 D 1 P 1
fountain PD

obelisk PD

fountain PD
```

Sample Output 3

```




W P

W D

R ambiguous
```

In the example interactions above, the output of the grader is on the left and one possible correct output of a submission is on the right.
The empty lines on both sides only serve to emphasise the chronological order of requests and answers. The grader will never output any empty lines.

Problem E: Election Meddling

Your university definitely needs a new building solely dedicated to computer science. The department has the money for it, but the city council does not want to approve the building permit for the new building. That's unfortunate!

Luckily, council elections are right around the corner. You have decided to participate in that election with the Interest Council Party for Computer science (ICPC). Once you have a majority of the votes in the council, you and your party colleagues can approve the new building.

The council elections are held in a number of districts to which the voters are assigned. Each district elects one council member. Each party fields one candidate per district and every voter has only one vote. The candidate with the most votes in each district will be elected as a council member.

Since the new building is very important to you and your party colleagues, you don't want to take any chances. In case of a tie in the council or in an election in a district no one knows what happens. So your objective is to win an outright majority in the council – i.e. strictly more than half of the members. Further, in each district you consider won, the ICPC candidate must receive strictly more votes than any other candidate.

You and your colleagues of the ICPC have created an incredibly sophisticated simulation of the election. Thus, you know exactly how many voters will vote for each candidate in every district. Sadly, your model does not predict a victory for the ICPC in the election. Here comes the tricky question: How many voters do you have to bribe at least in order to win the election? If you bribe a voter, he will change his previously preferred party (which you know due to your meticulous modelling) to the ICPC. People who didn't want to vote cannot be bribed.

Input

The input consists of:

- One line with two integers w and p ($2 \leq w, p \leq 1\,000$), the number of districts and the number of parties running in the election. The parties are numbered 1 to p and the ICPC is party 1.
- w lines, each with p integers $v_1, \ldots, v_p$ ($0 \leq v_i \leq 1\,000$ for each i) giving the projected results for a district. v_i denotes the number of votes that will be cast for party i.

It is guaranteed that there is at least one voter in each district, i.e. the sum of all v_i per district will always be at least one.

Output

Output the minimum number of voters that have to be bribed in order for the ICPC to win a majority of the seats in the council.

Sample Input 1

```
3 3
0 2 2
1 2 3
0 2 3
```

Sample Output 1

```
4
```

Sample Input 2

```
2 4
0 1 2 9
1 0 0 0
```

Sample Output 2

```
5
```

Sample Input 3

```
3 3
1 0 0
0 1000 0
0 9 5
```

Sample Output 3

```
6
```

Problem F: Final Standings

The GCPC 2019 is finally over. You have worked for five hours and have solved as many problems as possible. However, you still do not know which place you have got, since the scoreboard is still frozen. Of course you think that you and your team have won GCPC 2019.

As there is always a lengthy pause between the end of the contest and unfreezing the scoreboards (someone has to print the certificates and someone has to re-compile the solution slides), you want to use your time to determine how probable it is that your team has won GCPC 2019.

You know the final scoreboard, that is, for all teams which problems they have solved prior to the freeze and which problems they have attempted during the freeze. Also you know from last year how good each team is and you trust your own estimates of each problem's difficulty. To be precise, if a team with strength $s \in [0, 1]$ has attempted to solve a problem with difficulty $d \in [0, 1]$, you assume that the probability of solving the problem is $s \cdot d$.

As your team was very quick in solving the simple problems this year, you assume that a team can only have a higher place than you, if they have solved more problems (you assume that you will always win the tie-break).

Input

- The first line of input contains two integers t and p ($1 \le t, p \le 100$), the number of teams and the number of problems in the contest. The teams are numbered 1 to t and the problems are numbered 1 to p. Your team is team t.
- The second line contains $t - 1$ real numbers $s_1, \ldots, s_{t-1}$ ($0 \le s_i \le 1$ for each i), where s_i is the strength of team i.
- The third line contains p real numbers $d_1, \ldots, d_p$ ($0 \le d_j \le 1$ for each j), where d_j is the difficulty of problem j.
- Then follow $t - 1$ lines describing the state of the problems for the other teams. Every such line contains p characters $c_1, \ldots, c_p$, where c_j in the ith line describes the state of problem j for team i and is `X` if the team has solved the problem before the freeze, `?` if the team submitted a program for this problem after the freeze, and `-` otherwise.
- The last line of input describes the problems your team solved with p characters, each being either `X` or `-`.

All real numbers in the input have at most six decimals.

Output

Output a single real number, the probability that your team won GCPC 2019. Your answer should have an absolute or relative error of at most 10^{-6}.

Sample Input 1

```
3 3
0.95 0.95
0.95 0.95 0.95
? ? ?
X X -
X X -
```

Sample Output 1

```
0.264908
```

Sample Input 2

```
2 5
0.5
0.1 0.2 0.3 0.4 0.5
? ? ? ? ?
X - - - -
```

Sample Output 2

```
0.8387625
```

Problem G: Game of Falling Blocks

In this problem you have to program a simple AI for the game *Tetris*. The objective is simple: complete at least one row. More specifically, for the purposes of this problem we play Tetris according to the following modified set of rules:

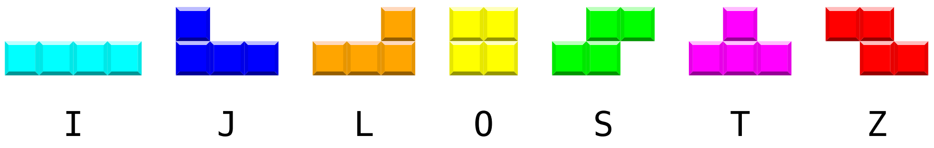

Figure G.1: The seven tetrominoes in their initial orientations.

- The playing area is a grid of height 20 and width 10, initially empty.
- One by one, the game presents the player with a random sequence of tetromino shaped pieces, as seen in Figure G.1. The game uses a bag randomiser for this, that is, the first seven pieces are all distinct, then the next seven pieces are distinct, and so on.
- The player may shift each piece sideways and rotate it by multiples of 90 degrees. These adjustments take place *above* the grid.
- The piece then drops down into the playing area and locks into place as soon as some part of it cannot drop any further because part of some previous piece is in the way. It is *not* possible to shift or rotate the piece during the drop.
- The game ends as soon as one of the following happens:
  - A horizontal row of the grid is fully occupied by tetromino tiles, the player wins.
  - Some piece does not fully drop into the grid, the player loses.

Interaction Protocol

Your submission will be interacting with a special program called the *grader*. This means that the output of your submission is sent to the grader and the output of the grader is sent to the standard input of your submission. This interaction must follow a specific protocol:

The grader repeatedly sends random pieces (see the note below), denoted by one of the uppercase letters $\{$`I`, `J`, `L`, `O`, `S`, `T`, `Z`$\}$ as in Figure G.1 above. In reply to each piece, your submission must answer with its desired placement, in the following format:

- Two integers r and x ($0 \leq r \leq 3, 1 \leq x \leq 10$), where r indicates how many times the piece should be rotated in clockwise direction, and x indicates the leftmost column occupied by the piece (columns are indexed from left to right, starting at 1).

The grader will then rotate and position the piece accordingly and drop it in the desired location.

After sending each placement, you should *flush* the standard output to ensure that it is sent to the grader. For example, you can use `fflush(stdout)` in C++, `System.out.flush()` in Java, and `sys.stdout.flush()` in Python.

The game ends as soon as your submission successfully completes a row or sends an invalid placement, whichever happens first (if both happen at the same time, the invalid placement takes precedence). A placement is invalid if some part of the piece would have to be placed outside of the playing area, either to the top or to the side.

If your submission completes a row, the grader will send a single character `W`. Upon receiving this, your submission must terminate with exit code `0` as usual.

Your submission will be accepted if it follows the protocol above and completes a row. If it makes an invalid placement, it will be judged as "Wrong Answer".

As described above, the sequence of pieces is determined by a bag randomiser. The seed values for this randomiser have been fixed in advance, such that every submission is run on the same piece sequences.

Sample Input 1	**Sample Output 1**
S	
	1 1
O	
	2 4
T	
	2 2
J	
	3 9
I	
	0 6
W	

In the example interaction above, one possible output of the grader is on the left and one possible correct output of a submission is on the right.
The empty lines on both sides only serve to emphasise the chronological order. The grader will never output any empty lines.

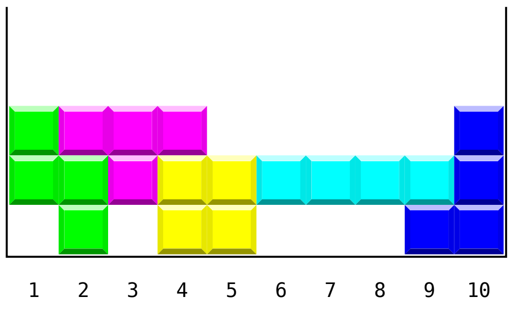

Figure G.2: Illustration of the example interaction.

Problem H: Historical Maths

The history of Numeristan is quite remarkable. As we all know the maharajahs of Numeristan were a superstitious bunch. Every year they consulted their chief magicians about their lucky number and all calculations during this year had to be done in a positional numeral system[1] with this lucky number as the base. Naturally this lead to a lot of confusion throughout the years. On the flip side it makes determining the year of ancient documents really easy for historians.

Recently an old manuscript was discovered which only features a simple multiplication of two numbers. As there are no other clues about the year of origin, some historians have asked you for help to determine the base of the system the calculation was performed in.

Input

The input consists of three lines, each describing a positive integer with no leading zeroes. Each line consists of:

- One integer n ($1 \leq n \leq 1\,000$), the number of digits of the currently described integer.
- n integers $d_{n-1}, \ldots, d_0$ ($0 \leq d_i \leq 2^{30}$ for each i, $d_{n-1} \neq 0$), the digits of the integer. The most significant digit is d_{n-1}, the least significant digit is d_0.

The first two lines correspond to the factors, the third line to the product of the multiplication.

Output

Output a possible base the multiplication was performed in. If there are multiple possible bases, you may output any of them. If no possible base exists, output `impossible`.

Sample Input 1

```
2 2 0
1 2
3 1 0 0
```

Sample Output 1

```
4
```

Sample Input 2

```
3 5 1 2
2 11 3
5 4 5 1 12 6
```

Sample Output 2

```
13
```

Sample Input 3

```
2 3 2
2 3 2
3 10 12 4
```

Sample Output 3

```
impossible
```

[1]A positional numeral system of base b only contains the digits $0, 1, \ldots, b-1$. An integer n has one or more digits d_i. Each digit contributes to the total value of n with its position i and its own value, according to the formula $n = \sum_{i=0}^{\infty} d_i \cdot b^i$. Leading digits of value 0 are usually omitted. For instance, the integer 512 in base 13 has the total value of $5 \cdot 13^2 + 1 \cdot 13^1 + 2 \cdot 13^0$. Well known and used positional numeral systems are the binary system (which is also the positional numeral system with the smallest valid base), the decimal system as well as the hexadecimal system.

This page is intentionally left (almost) blank.

Problem I: Insertion Order

A friend of yours is currently taking a class on algorithms and data structures. Just last week he learned about binary search trees and the importance of using self-balancing trees in order to keep the tree height low and guarantee fast access to every node.

Recall that a binary search tree is a binary tree with each node storing a key, and the property that the key of each node is greater than all keys in the left subtree of that node and less than all keys in the right subtree. A new key is inserted into the tree by adding a new leaf node with that key in the only position such that the property is maintained, as seen in the figure below.

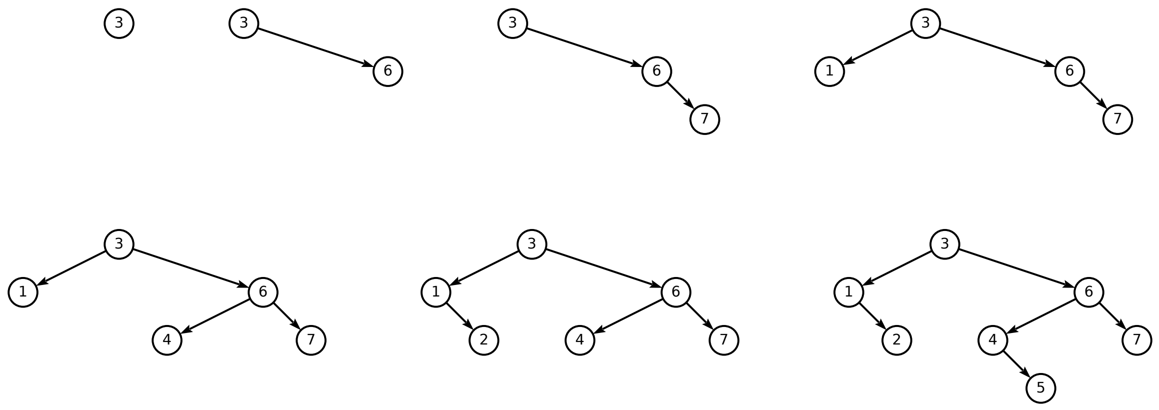

Figure I.1: Illustration of the first sample case.

To illustrate to him just how bad things can get without self-balancing, you want to show him that it is possible to build trees of nearly any height by carefully choosing an insertion order.

You are given two integers n and k and want to construct a binary search tree with n nodes of height k (the height of a tree is the maximal number of nodes on a path from the root to a leaf). To do so, you need to find a permutation of the integers from 1 to n such that, when they are inserted into an empty binary search tree in that order (without self-balancing), the resulting tree has height k.

Input

The input consists of two integers n and k ($1 \leq k \leq n \leq 2 \cdot 10^5$), where n is the number of nodes in the tree and k is the exact height the tree should have.

Output

If there is no solution, output `impossible`. Otherwise, output one line with n integers, the requested permutation. If there is more than one solution, any one of them will be accepted.

Sample Input 1

```
7 4
```

Sample Output 1

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

Sample Input 2

```
8 3
```

Sample Output 2

```
impossible
```

This page is intentionally left (almost) blank.

Problem J: Jazz Enthusiast

Kai is listening to his favourite jazz playlist. He likes to turn on crossfading between songs, so during the last seconds of a song, it will slowly fade out while the next one fades in. This happens between any two consecutive songs, but the beginning of the first song and the end of the last song will be played normally.

Determine the total amount of time it takes Kai to listen to the whole playlist.

Input

The input consists of:

- Two integers n and c ($1 \leq n \leq 100$, $1 \leq c \leq 10$), giving the number of songs and the crossfade time in seconds.
- n lines of the form `m:ss` (`0:30` $\leq$ `m:ss` $\leq$ `9:59`), giving the song lengths (with one digit for the number of whole minutes and two digits for the remaining seconds).

Output

Output a string of the form `hh:mm:ss`, giving the total time it takes to listen through the whole playlist (with two digits for the number of whole hours, two digits for the number of remaining whole minutes, and two digits for the remaining seconds).

Sample Input 1

```
3 5
3:59
0:52
9:40
```

Sample Output 1

```
00:14:21
```

Sample Input 2

```
1 10
6:00
```

Sample Output 2

```
00:06:00
```

This page is intentionally left (almost) blank.

Problem K: Keeping the Dogs Out

Your best friend keeps telling you about her neighbour's annoying dogs – apparently, they constantly lay waste to her garden and leave smelly "gifts". Today, she has decided to start working on a solution: building a wall to keep them out.

Of course the wall must have the same height in all places (i.e. be rectangular), and it cannot have any holes that would let the dogs through. Your friend does not really care about the wall's exact dimensions, though.

She has already bought the stones she plans to use (and she wants to use all of them!) and sorted them by size. All of them have the same width, but the length and height may vary. For each stone however, its length is equal to its height and both are a power of two. The width of the wall must be equal to the width of the stones, i.e. you may not rotate the stones and you may not put two stones behind each other.

Given the side lengths and quantities of the stones, determine if it is possible to build a wall using all available stones, and if so, what length and height such a wall might have.

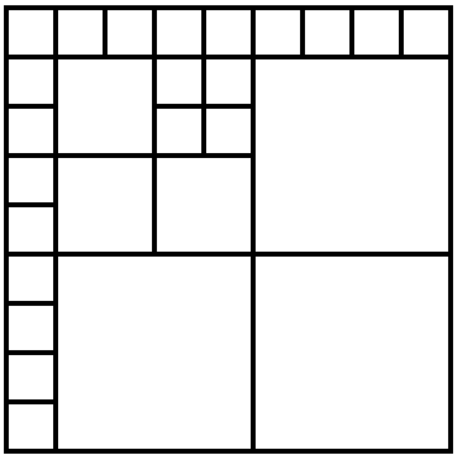

Figure K.1: Illustration of the first sample.

Input

The input consists of:

- One line with an integer n ($0 \leq n \leq 25$), indicating that the largest stone has a side length of 2^n.
- One line with $n + 1$ integers $m_0, \ldots, m_n$ ($0 \leq m_i \leq 10^{15}$ for each i, $m_n \geq 1$), where m_i describes the number of stones of side length 2^i.

It is guaranteed that the combined area of all stones is at most 10^{15}.

Output

If it is possible to build a rectangular wall without any holes, output one line with two integers, the length and height of a possible wall that can be built. Otherwise, output `impossible`. If multiple solutions exist, output any of them.

Sample Input 1

```
2
21 3 3
```

Sample Output 1

```
9 9
```

Sample Input 2

```
2
0 3 2
```

Sample Output 2

```
impossible
```

Problem L: Long-Exposure Photography

Gwen has painted some rectangles with completely black surfaces on a large sheet of graph paper. To ensure that everything looks neat and tidy, all of her rectangles' edges lie on the lines of the paper. Now she intends to take a photo of the paper while spinning it very fast. The exposure time of the camera is long enough to cover an entire rotation of the plane. In the resulting photo, areas covered by rectangles throughout the entire rotation will appear black, whereas areas never covered by a rectangle will appear white. The remaining "blurry" areas may vary in brightness, but Gwen simply considers all of those to be grey.

Now, before taking the photo, Gwen tells you the coordinates of the rectangles she has painted and asks you to determine how much of the photo's surface area will be black and how much of it will be grey.

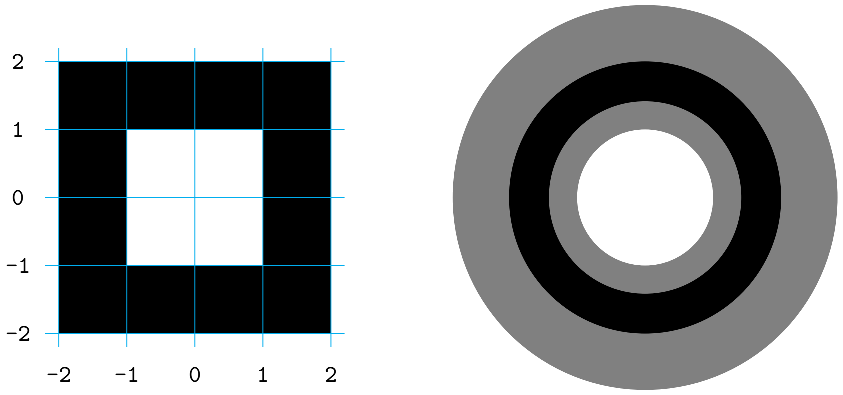

Figure L.1: The first sample input and the resulting photo.

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 1\,000$), giving the number of rectangles.
- n lines, each with four numbers: two integers x_{min}, y_{min} ($-10^9 \leq x_{min}, y_{min} \leq 10^9$) giving the coordinates of the rectangle's bottom left corner (that is, the corner with minimum x and y values) and two integers w, h ($1 \leq w, h \leq 200$) giving the rectangle's width and height.

The plane is rotated around the origin $(0, 0)$, and you may safely assume that the entire sheet of paper is fully captured by the camera throughout the entire rotation.

Output

Output two real numbers, the first being the area of the black and the second being the area of the grey portion of the photo's surface. A number is considered correct if its absolute or relative error does not exceed 10^{-6}.

Sample Input 1

```
4
-2 1 4 1
-2 -2 1 4
-2 -2 4 1
1 -2 1 4
```

Sample Output 1

```
6.2831853
15.7079633
```

Sample Input 2

```
1
0 0 2 2
```

Sample Output 2

```
0.0
25.1327412
```

Problem M: Move & Meet

Ernesto and Penelope are playing a board game on an infinite grid. Instead of rolling dice, both of them have generated a random number, and now they have to move their pieces that number of times. A single move consists of placing the piece in an adjacent cell; moving diagonally or waiting in place are not legal moves. However, it *is* permitted to move the piece in the direction it just came from in the previous move.

Ernesto and Penelope are trying to move in such a way that their pieces will end up in the same cell. Is there a cell for which this is possible?

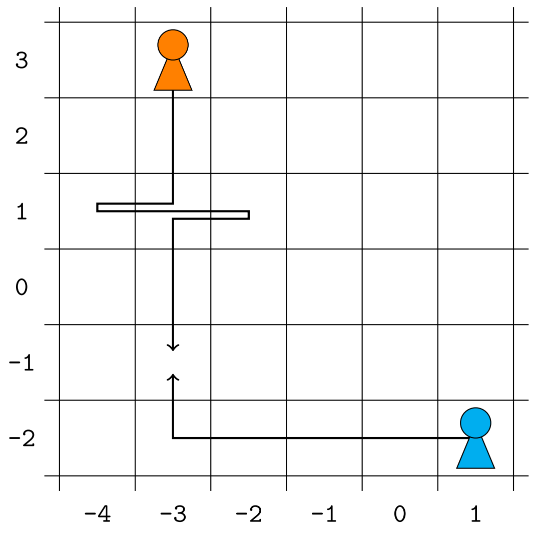

Figure M.1: Visualisation of the first sample, including possible paths for the given output.

Input

The input consists of two lines, both containing three integers x, y ($-10^{12} \leq x, y \leq 10^{12}$) and d ($0 \leq d \leq 10^{12}$), giving for either player the pieces' initial coordinates and the randomly generated number.

Output

If there is a cell that both players can end up on, output its coordinates. If there are multiple valid solutions, any will be accepted. If there is no valid cell, output `impossible`.

Sample Input 1

```
1 -2 5
-3 3 8
```

Sample Output 1

```
-3 -1
```

Sample Input 2

```
0 -1000000000000 0
0 -1000000000000 0
```

Sample Output 2

```
0 -1000000000000
```

Sample Input 3

```
-5 -426 932111
83 -870 478692
```

Sample Output 3

```
impossible
```

This page is intentionally left (almost) blank.