Northwestern European Regional Contest 2015

NWERC 2015

Linköping, November 29

Problems

A Assigning Workstations
B Better Productivity
C Cleaning Pipes
D Debugging
E Elementary Math
F Flight Plan Evaluation
G Guessing Camels
H Hole in One
I Identifying Map Tiles
J Jumbled Communication
K Kitchen Combinatorics

Do not open before the contest has started.

This page is intentionally left (almost) blank.

Problem A
Assigning Workstations
Time limit: 10 seconds

Picture by NASA via WikiMedia Commons

Penelope is part of the admin team of the newly built supercomputer. Her job is to assign workstations to the researchers who come here to run their computations at the supercomputer.

Penelope is very lazy and hates unlocking machines for the arriving researchers. She can unlock the machines remotely from her desk, but does not feel that this menial task matches her qualifications. Should she decide to ignore the security guidelines she could simply ask the researchers not to lock their workstations when they leave, and then assign new researchers to workstations that are not used any more but that are still unlocked. That way, she only needs to unlock each workstation for the first researcher using it, which would be a huge improvement for Penelope.

Unfortunately, unused workstations lock themselves automatically if they are unused for more than m minutes. After a workstation has locked itself, Penelope has to unlock it again for the next researcher using it. Given the exact schedule of arriving and leaving researchers, can you tell Penelope how many unlockings she may save by asking the researchers not to lock their workstations when they leave and assigning arriving researchers to workstations in an optimal way? You may assume that there are always enough workstations available.

Input

The input consists of:

- one line with two integers n ($1 \leq n \leq 300\,000$), the number of researchers, and m ($1 \leq m \leq 10^8$), the number of minutes of inactivity after which a workstation locks itself;
- n lines each with two integers a and s ($1 \leq a, s \leq 10^8$), representing a researcher that arrives after a minutes and stays for exactly s minutes.

Output

Output the maximum number of unlockings Penelope may save herself.

Sample Input 1	Sample Output 1
3 5 1 5 6 3 14 6	2

Sample Input 2	Sample Output 2
5 10 2 6 1 2 17 7 3 9 15 6	3

Problem B
Better Productivity
Time limit: 2 seconds

ACME Inc. is reorganizing their factory, in order to maximize their productivity of useless trinkets. The new factory design consists of p independent and identical production lines. Each production line can be assigned some number of workers.

The actual work is of course all done by machines, so the production rate of a production line is independent of the actual number of workers assigned to it. However, the workers work in shifts, and due to local safety regulations, a production line can only be active while all of its workers are present (any worker who arrives before the last person to arrive, or leaves after the first person leaves, will be spending the inactive time having a coffee break). In other words, the productivity of a production line equals the length of the timespan during which all of the workers assigned to this production line are present. Crucially, the productivity of each line *must* be positive (i.e., the last worker to arrive for a line must arrive strictly before the first worker for that line leaves), since otherwise the workers feel that their jobs are meaningless.

Unfortunately, due to some pesky labor laws, ACME are not allowed to fire any of their workers, which means that each of their n workers must be assigned to some production line, even though this may actually decrease the overall productivity of the factory.

All these rules and regulations are making a headache for ACME management. Can you help them figure out the maximum possible total productivity (sum of productivities of the p production lines) of their new factory?

Input

The input consists of:

- one line containing two integers n and p ($1 \leq p \leq n \leq 200$), the number of employees and the number of production lines;
- n lines each containing two integers a and b ($0 \leq a < b \leq 100\,000$), representing a worker that arrives at time a and leaves at time b.

You may assume that there exists at least one valid assignment of workers to production lines.

Output

Output the maximum productivity level possible for the factory.

Sample Input 1	Sample Output 1
4 2 1 3 1 5 4 6 2 7	4

This page is intentionally left (almost) blank.

Problem C
Cleaning Pipes
Time limit: 7 seconds

Picture of sink hole by Ogar93 via Wikimedia Commons

Linköping has a quite complex water transport system. Around Linköping there are several wells from which water is drawn. The water is then transported to other locations using pipes. Each pipe is a straight canal from one of the wells to some location in the city.

All the pipes are at the same depth under ground. Therefore, whenever two pipes cross, they form an intersection. Luckily the pipe system was constructed in such a way that exactly two pipes meet at each such intersection. The wells do not count as intersections. Any number of pipes (including zero or more than two) may originate at each well.

The intersections pose a problem, since dirt (a mixture of lime and other "remains") tends to get stuck there. This dirt causes the pipes to degrade and collapse, leading to the formation of large sink holes. Such sink holes have a mesmerising effect on the students in Linköping, causing them to neglect their studies and remain uneducated, which in the long run will lead to a collapse of not just the pipe system but of the very fabric of society. Therefore it is imperative that the pipes are regularly cleaned. The Nordic Water Extraction and Redistribution Company (NWERC) – which is in charge of Linköping's waterpipes – has an ample fleet of robots to perform this task. A robot can be inserted into a pipe at the well where the pipe begins. The robot then goes through the pipe all the way to its end and cleans all intersections along the way. After reaching the end, the robot turns around and returns back to the well where it started. In order to prevent robot collisions, government regulations stipulate that whenever two pipes intersect, at most one of them may contain a robot.

Since the whole water system has to be shut down while it is being cleaned (another government regulation), the NWERC would like to do the cleaning quickly, by using a single batch of cleaning robots, all started at the same time.

Your task is to verify whether this can be done – i.e., whether we can simultaneously insert robots into a subset of the pipes in such a way that the robots will clean all intersections and there will be no risk of two robots colliding.

Input

The input consists of:

- one line with two integers w ($1 \leq w \leq 1\,000$), the number of wells, and p ($1 \leq p \leq 1\,000$), the number of pipes;
- w lines, the ith of which contains two integers x_i and y_i ($-10\,000 \leq x, y \leq 10\,000$), the position of well number i (the wells are numbered from 1 to w);
- p lines each with three integers s ($1 \leq s \leq w$), the well at which the pipe starts, and x and y ($-10\,000 \leq x, y \leq 10\,000$), the position at which the pipe ends.

Each pipe will contain exactly one well, the one at which it starts. Any point shared by more than two pipes will be a well. Any two pipes share at most one common point. The common point of two pipes may be the endpoint of one or both of them. All pipes have positive length.

Output

If it is possible to clean all intersections as described above, output "`possible`". Otherwise, output "`impossible`".

Sample Input 1

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

Sample Output 1

```
impossible
```

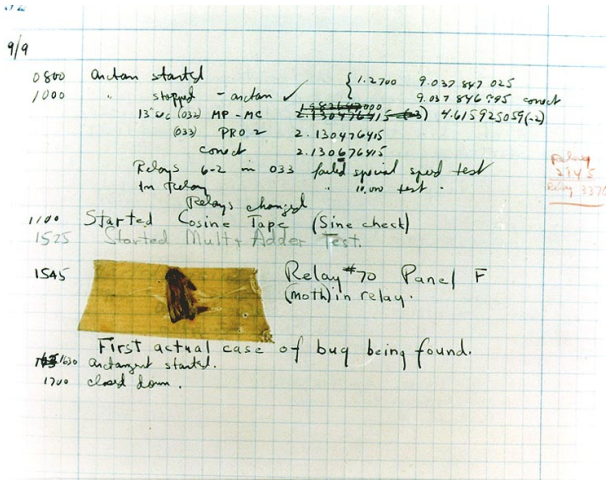
Sample Input 2

```
2 3
0 0
0 10
1 5 15
1 2 15
2 10 10
```

Sample Output 2

```
possible
```

Problem D
Debugging
Time limit: 3 seconds



Picture in public domain via Wikimedia Commons

Your fancy debugger will not help you in this matter. There are many ways in which code can produce different behavior between debug and release builds, and when this happens, one may have to resort to more primitive forms of debugging.

So you and your printf are now on your own in the search for a line of code that causes the release build to crash. Still you are lucky: adding printf statements to this program affects neither the bug (it still crashes at the same original code line) nor the execution time (at least not notably). So even the naive approach of putting a printf statement before each line, running the program until it crashes, and checking the last printed line, would work.

However, it takes some time to add each printf statement to the code, and the program may have a lot of lines. So perhaps a better plan would involve putting a printf statement in the middle of the program, letting it run, seeing whether it crashes before the added line, and then continuing the search in either the first or second half of the code.

But then again, running the program may take a lot of time, so the most time-efficient strategy might be something in between. Write a program that computes the minimum worst-case time to find the crashing line (no matter where it is), assuming you choose an optimal strategy for placing your printf statements.

We're releasing the new version in five hours, so this issue is escalated and needs to be fixed ASAP.

Input

The input consists of one line with three integers:

- n ($1 \leq n \leq 10^6$), the number of code lines;
- r ($1 \leq r \leq 10^9$), the amount of time it takes to compile and run the program until it crashes;
- p ($1 \leq p \leq 10^9$), the time it takes to add a single printf line.

You have already run the program once and therefore already know that it does crash somewhere.

Output

Output the worst-case time to find the crashing line when using an optimal strategy.

Sample Input 1	Sample Output 1
1 100 20	0

Sample Input 2	Sample Output 2
10 10 1	19

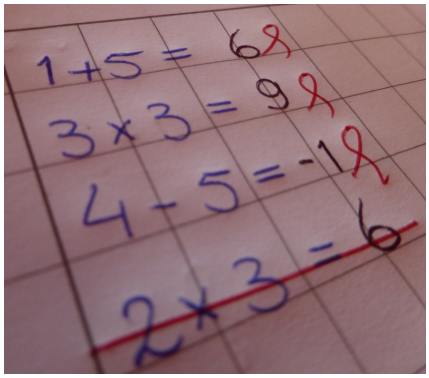
Sample Input 3	Sample Output 3
16 1 10	44

Problem E
Elementary Math
Time limit: 10 seconds

Ellen is teaching elementary math to her students and the time for the final exam has come. The exam consists of n questions. In each question the students have to add ($+$), subtract ($-$) or multiply ($*$) a pair of numbers.

Ellen has already chosen the n pairs of numbers. All that remains is to decide for each pair which of the three possible operations the students should perform. To avoid students getting bored, Ellen wants to make sure that the n correct answers to her exam are all different.

Please help Ellen finish constructing the exam by automating this task.



Example exam by Ellen

Input

The input consists of:

- one line with one integer n ($1 \leq n \leq 2\,500$), the number of pairs of numbers;
- n lines each with two integers a and b ($-10^6 \leq a, b \leq 10^6$), a pair of numbers used.

Output

For each pair of numbers (a, b) in the same order as in the input, output a line containing a valid equation. Each equation should consist of five parts: a, one of the three operators, b, an equals sign ($=$), and the result of the expression. All the n expression results must be different.

If there are multiple valid answers you may output any of them. If there is no valid answer, output a single line with the string "`impossible`" instead.

Sample Input 1

```
4
1 5
3 3
4 5
-1 -6
```

Sample Output 1

```
1 + 5 = 6
3 * 3 = 9
4 - 5 = -1
-1 - -6 = 5
```

Sample Input 2

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

Sample Output 2

```
impossible
```

This page is intentionally left (almost) blank.

Problem F
Flight Plan Evaluation
Time limit: 6 seconds

When flying between two places, constructing a good flight plan is important. In general there is a wide range of different factors to consider, the most important being fuel consumption and weather forecasts (especially winds). In this problem, we will evaluate flight plans with respect to a third statistic, namely how much of the flight is over water, and how much is over ground. This statistic is not relevant per se, yet many passengers seem to prefer flying over land – either because they are afraid of flying over water, or simply because the view tends to be slightly more interesting when flying over land.

For this problem, we assume that the earth is a perfect sphere with radius 6370 km. We model each continent of the earth as a polygon on this sphere – a closed sequence of line segments, where a line segment between two points consists of the shortest spherical arc between these two points. The two end-points of a line segment can not be the same point, or antipodal (diametrically opposite) points. Similarly a flight route is modeled as a sequence of waypoints connected by line segments, but unlike the line segments of a polygon these line segments may cross themselves and will not necessarily end up where they started.

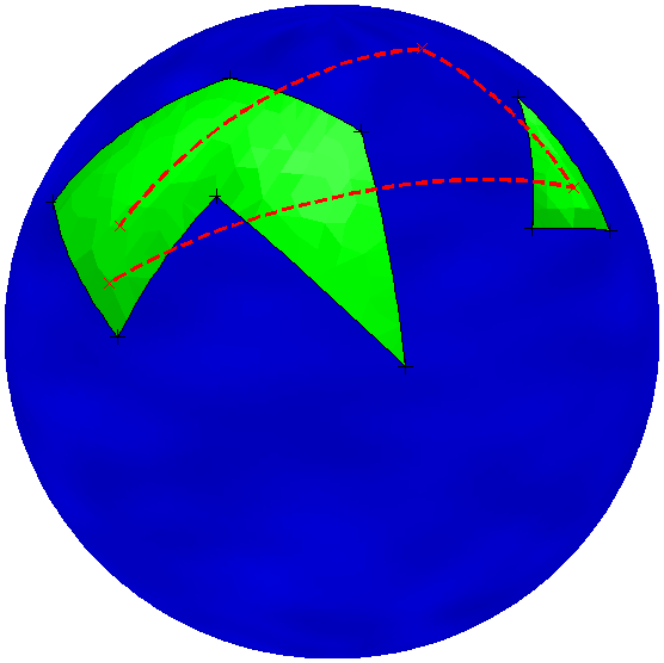


Figure F.1: The second sample input

In order to simplify the problem, we additionally make the following two assumptions:

- No waypoint of a flight route lies within 0.1 km of any shoreline (a line segment that is part of a polygon).
- No vertex of any continent polygon lies within 0.1 km of the flight route.

All coordinates on the sphere are represented as a pair of latitude and longitude (both in degrees). A point with latitude ± 90 is the north/south pole, and points with latitude 0 are the points on the equator.

Input

The input consists of:

- one line with an integer $1 \leq c \leq 30$, the number of continents;
- c lines, each describing a continent. Each such line starts with an integer $3 \leq n \leq 30$, the number of vertices in the polygon describing the continent. This is followed by n pairs of integers $\phi_1, \lambda_1, \ldots, \phi_n, \lambda_n$, where $-90 \leq \phi_i \leq 90$ and $0 \leq \lambda_i \leq 359$ are the latitude and longitude of the ith vertex of the continent;
- one line describing the flight plan. The line starts with an integer $2 \leq m \leq 30$, the number of waypoints. This is followed by m pairs of integers $\phi_1, \lambda_1, \ldots, \phi_m, \lambda_m$, where $-90 \leq \phi_i \leq 90$ and $0 \leq \lambda_i \leq 359$ are the latitude and longitude of the ith waypoint of the route.

A continent cannot cross itself. No continent will touch or contain any other continent. Continents are given in counterclockwise order, in the sense that if you go from the first vertex of the polygon to the second one, the interior of the continent is on your left hand side.

The first and last waypoints of the route will always be inside a continent (but not necessarily the same continent).

Output

Output two real numbers l and w, where l is the total length of the flight (in km), and w is the percentage of the flight that is over water. The numbers should be accurate to an absolute or relative error of at most 10^{-6}.

Sample Input 1

```
1
4 -45 0 45 0 45 90 -45 90
5 0 180 0 359 0 160 0 170 0 180
```

Sample Output 1

```
40023.890406734 25.0000000000
```

Sample Input 2

```
2
6 62 80 28 49 10 80 37 95 8 134 51 129
3 52 188 29 165 24 188
4 19 77 33 180 69 169 29 75
```

Sample Output 2

```
21243.902224493 52.066390024
```

Problem G
Guessing Camels
Time limit: 10 seconds

Jaap, Jan, and Thijs are on a trip to the desert after having attended the ACM ICPC World Finals 2015 in Morocco. The trip included a camel ride, and after returning from the ride, their guide invited them to a big camel race in the evening. The camels they rode will also participate and it is customary to bet on the results of the race.

One of the most interesting bets involves guessing the complete order in which the camels will finish the race. This bet offers the biggest return on your money, since it is also the one that is the hardest to get right.

Jaap, Jan, and Thijs have already placed their bets, but the race will not start until an hour from now, so they are getting bored. They started wondering how many pairs of camels they have put in the same order. If camel c is before camel d on Jaap's, Jan's and Thijs' bet, it means that all three of them put c and d in the same order. Can you help them to calculate the number of pairs of camels for which this happened?

Jaap, Jan and Thijs on camels (in some order). Photo by Tobias Werth, cc by-sa.

Input

The input consists of:

- one line with an integer n ($2 \leq n \leq 200\,000$), the number of camels;
- one line with n integers $a_1, \ldots, a_n$ ($1 \leq a_i \leq n$ for all i), Jaap's bet. Here a_1 is the camel in the first position of Jaap's bet, a_2 is the camel in the second position, and so on;
- one line with Jan's bet, in the same format as Jaap's bet;
- one line with Thijs' bet, in the same format as Jaap's bet.

The camels are numbered $1, \ldots, n$. Each camel appears exactly once in each bet.

Output

Output the number of pairs of camels that appear in the same order in all 3 bets.

Sample Input 1	Sample Output 1
3 3 2 1 1 2 3 1 2 3	0

Sample Input 2	Sample Output 2
4 2 3 1 4 2 1 4 3 2 4 3 1	3

Problem H
Hole in One
Time limit: 5 seconds

Janine recently went to her local game store and bought "Hole in One", a new mini-golf game for her computer. As indicated by the name, the objective of the game is to shoot a ball into a hole using just one shot. The game also borrows elements from brick breaker style games: in the playing field, several walls are placed that will be destroyed upon being hit by the ball. The score of a successful shot depends on the number of destroyed walls, so Janine wonders: what is the maximum number of walls that can be hit while performing a "Hole in One"?

For the purposes of this problem you can think of the playing field as a cartesian plane with the initial position of the ball at the origin. The walls are non-intersecting axis-parallel line segments in this plane (i.e., parallel to either the x axis or the y axis). The diameter of the ball is negligible so it is represented as a single point.

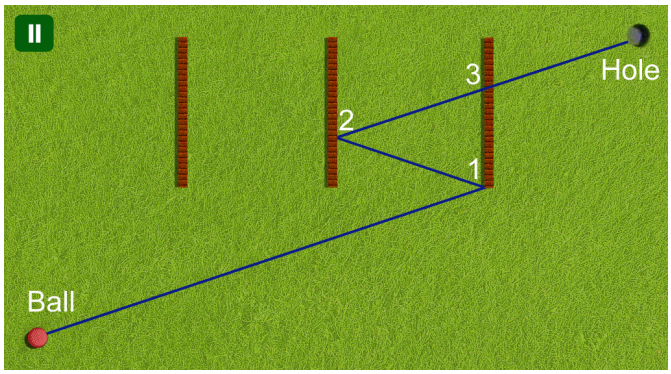


Figure H.1: Illustration of the first sample input: The ball first bounces off two walls at points 1 and 2. When it passes point 3 the wall has already vanished.

Whenever the ball hits a wall, two things happen:

- The direction of the ball changes in the usual way: the angle of incidence equals the angle of reflection.
- The wall that the ball touched is destroyed. Following common video game logic, no rubble of the wall remains; it will be as though it vanished.

The behaviour of the ball is also affected by the power of the shot. In particular, an optimal shot may need to first roll over the hole, then hit some more walls, and only later drop into the hole.

Input

The input consists of:

- one line with one integer n ($0 \leq n \leq 8$), the number of walls;
- one line with two integers x and y, the coordinates of the hole;
- n lines each with four integers x_1, y_1, x_2, and y_2 (either $x_1 = x_2$, or $y_1 = y_2$, but not both), representing a wall with end points (x_1, y_1) and (x_2, y_2).

The hole is not at the origin and not on a wall. The walls do not touch or intersect each other. No wall lies completely on the x axis or the y axis. All coordinates in the input are integers with absolute value at most $1\,000$.

Output

If there is no way to shoot the ball such that it reaches the hole, print "`impossible`". Otherwise, print the maximum number of walls that can be destroyed in a single "Hole in One" shot.

Sample Input 1

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

Sample Output 1

```
2
```

Sample Input 2

```
1
2 0
1 -1 1 1
```

Sample Output 2

```
impossible
```

Sample Input 3

```
2
-2 4
2 4 2 2
0 6 -2 6
```

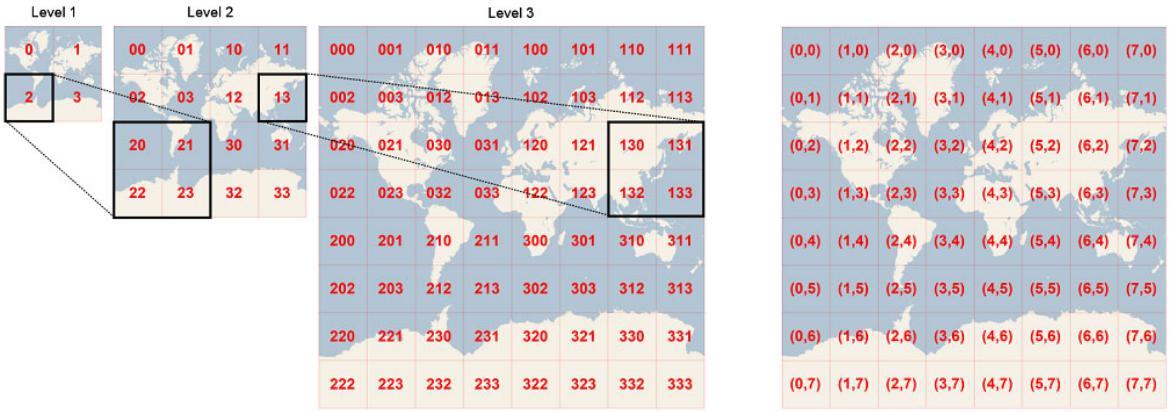
Sample Output 3

```
2
```

Problem I
Identifying Map Tiles
Time limit: 1 second

Map websites such as Bing Maps and Google Maps often store their maps as many different image files, called tiles. The lowest zoom level (level 0) consists of a single tile with a low-detail image of the whole map, zoom level 1 consists of four tiles each containing a slightly more detailed version of a quarter of the map, and in general zoom level n contains 4^n different tiles that each contain a part of the map.

One way of identifying a tile is by means of a *quadkey*. A quadkey is a string of digits uniquely identifying a tile at a certain zoom level. The first digit specifies in which of the four quadrants of the whole map the tile lies: `0` for the top-left quadrant, `1` for the top-right quadrant, `2` for the bottom-left quadrant and `3` for the bottom-right quadrant. The subsequent digits specify in which sub quadrant of the current quadrant the tile is. The quadkeys for zoom levels 1 to 3 are shown in Figure I.1(a).



(a) Quadkeys for zoom levels 1 to 3

(b) Coordinates for zoom level 3

Figure I.1: Visualisation of the two representations. The images are taken from the MSDN.

Another way of identifying a tile is to give the zoom level and x and y coordinates, where $(0, 0)$ is the left-top corner. The coordinates for the tiles of zoom level 3 are shown in Figure I.1(b). Given the quadkey of a tile, output the zoom level and x and y coordinates of that tile.

Input

The input consists of:

- one line with a string s ($1 \leq \text{length}(s) \leq 30$), the quadkey of the map tile.

The string s consists of only the digits '`0`', '`1`', '`2`' and '`3`'.

Output

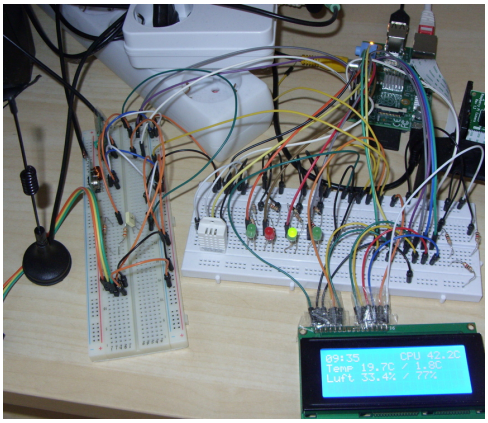
Output three integers, the zoom level and the x and y coordinates of the tile.

Sample Input 1	Sample Output 1
3	1 1 1

Sample Input 2	Sample Output 2
130	3 6 2

Problem J
Jumbled Communication
Time limit: 5 seconds

Your best friend Adam has recently bought a Raspberry Pi and some equipment, including a wireless temperature sensor and a 433MHz receiver to receive the signals the sensors sends. Adam plans to use the Raspberry Pi as an in-door display for his weather sensor. As he is very good with electronics, he quickly managed to get the receiver to receive the signals of the sensor. However, when he looked at the bytes sent by the sensor he could not make heads or tails of them. After some hours looking through a lot of websites, he found a document explaining that his weather sensor scrambles the data it sends, to prevent it from being used together with products from other manufacturers.



Luckily, the document also describes how the sensor scrambles its communication. The document states that the sensor applies the expression `x ^ (x << 1)` to every byte sent. The `^` operator is bit-wise XOR[1], e.g., `10110000 ^ 01100100` = `11010100`. The `<<` operator is a (non-circular) left shift of a byte value[2], e.g., `10111001 << 1` = `01110010`.

In order for Adam's Raspberry Pi to correctly interpret the bytes sent by the weather sensor, the transmission needs to be unscrambled. However, Adam is not good at programming (actually he is a pretty bad programmer). So he asked you to help him and as a good friend, you are always happy to oblige. Can you help Adam by implementing the unscrambling algorithm?

Input

The input consists of:

- one line with an integer n ($1 \leq n \leq 10^5$), the number of bytes in the message sent by the weather sensor;
- one line with n integers $b_1, \ldots, b_n$ ($0 \leq b_i \leq 255$ for all i), the byte values of the message.

Output

Output n byte values (in decimal encoding), the unscrambled message.

Sample Input 1	Sample Output 1
5 58 89 205 20 198	22 55 187 12 66

[1] In bit-wise XOR, the ith bit of the result is 1 if and only if exactly one of the two arguments has the ith bit set.

[2] In `x << j`, the bits of `x` are moved j steps to the left. The j most significant bits of x are discarded, and j zeroes are added as the least significant bits of the result.

This page is intentionally left (almost) blank.

Problem K
Kitchen Combinatorics
Time limit: 4 seconds

Picture by HarshLight via Flickr

The world-renowned Swedish Chef is planning a gourmet three-course dinner for some muppets: a starter course, a main course, and a dessert. His famous Swedish cook-book offers a wide variety of choices for each of these three courses, though some of them do not go well together (for instance, you of course cannot serve chocolate moose and sooted shreemp at the same dinner).

Each potential dish has a list of ingredients. Each ingredient is in turn available from a few different brands. Each brand is of course unique in its own special way, so using a particular brand of an ingredient will always result in a completely different dinner experience than using another brand of the same ingredient.

Some common ingredients such as pølårber may appear in two of the three chosen dishes, or in all three of them. When an ingredient is used in more than one of the three selected dishes, Swedish Chef will use the same brand of the ingredient in all of them.

While waiting for the meecaroo, Swedish Chef starts wondering: how many different dinner experiences are there that he could make, by different choices of dishes and brands for the ingredients?

Input

The input consists of:

- one line containing five integers r, s, m, d, n, where $1 \leq r \leq 1\,000$ is the number of different ingredients that exist, $1 \leq s, m, d \leq 25$ are the number of available starter dishes, main dishes, and desserts, respectively, and $0 \leq n \leq 2\,000$ is the number of pairs of dishes that do not go well together.
- one line containing r integers $b_1, \ldots, b_r$, where $1 \leq b_i \leq 100$ is the number of different brands of ingredient i.
- $s + m + d$ lines describing the s starter dishes, then the m main dishes, then the d desserts. Each such line starts with an integer $1 \leq k \leq 20$ denoting the number of ingredients of the dish, and is followed by k distinct integers $i_1, \ldots, i_k$, where for each $1 \leq j \leq k$, $1 \leq i_j \leq r$ is an ingredient.
- n lines each containing two incompatible dishes. Each dish is identified by an integer $1 \leq j \leq s + m + d$, referring to the j'th dish given in the input (so $1 \leq j \leq s$ refers to the starter dishes, $s < j \leq s + m$ refers to the main dishes, and $s + m < j \leq s + m + d$ refers to the desserts).

Each pair of incompatible dishes in the input consists of two dishes of different types, and any one pair of dishes is listed at most once.

Output

If the number of different dinner experiences Swedish Chef can make is at most 10^{18}, then output that number. Otherwise, output "`too many`".

Sample Input 1

```
6 1 1 1 0
2 3 1 5 3 2
2 1 2
3 3 4 5
1 6
```

Sample Output 1

```
180
```

Sample Input 2

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

Sample Output 2

```
22
```

Sample Input 3

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

Sample Output 3

```
0
```

Sample Input 4

```
10 1 1 1 0
100 100 100 100 100 100 100 100 100 100
4 1 2 3 4
3 5 6 7
3 8 9 10
```

Sample Output 4

```
too many
```