NORTHWESTERN EUROPEAN REGIONAL CHAMPIONSHIP 2012

CONTEST PROBLEM SET

NOVEMBER 25, 2012

Regionals 2012

acm International Collegiate Programming Contest

IBM event sponsor

This page was intentionally left almost blank

A - Admiral

Michiel Adriaenszoon de Ruyter is the most famous admiral in Dutch history and is well known for his role in the Anglo-Dutch Wars of the 17th century. De Ruyter personally commanded a flagship and issued commands to allied warships during naval battles.

In De Ruyter's time, graph theory had just been invented and the admiral used it to his great advantage in planning his naval battles. Waypoints at sea are represented by vertices, and possible passages from one waypoint to another are represented as directed edges. Given any two waypoints W_1 and W_2, there is at most one passage $W_1 \rightarrow W_2$. Each directed edge is marked with the number of cannonballs that need to be fired in order to safely move a ship along that edge, sinking the enemy ships encountered along the way.

One of De Ruyter's most successful tactics was the *De Ruyter Manoeuvre*. Here, two warships start at the same waypoint, and split up and fight their way through the enemy fleet, joining up again at a destination waypoint. The manoeuvre prescribes that the two warships take disjunct routes, meaning that they must not visit the same waypoint (other than the start and end-points), or use the same passage during the battle.

Being Dutch, Admiral De Ruyter did not like to waste money; in 17th century naval warfare, this meant firing as few expensive cannonballs as possible.

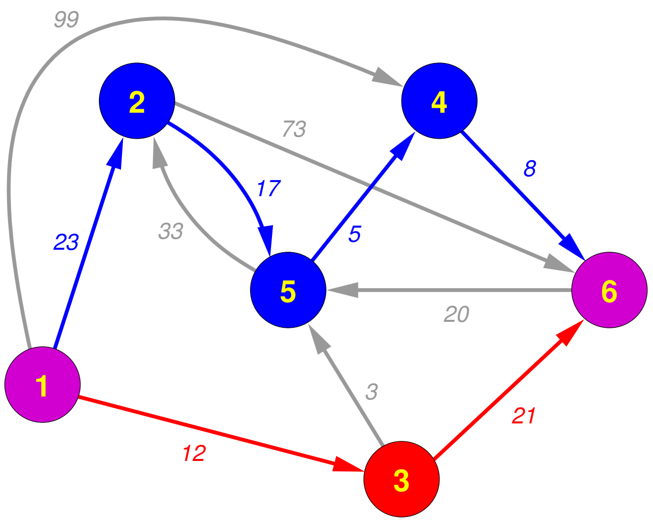


Figure 1: A particular instance of De Ruyter's tactic, visualised as a graph. Two ships ('red' and 'blue') move from a shared starting point (1) to a shared endpoint (6). The red ship's route is $1 \rightarrow 3 \rightarrow 6$ (firing 33 canonballs along the way); the blue ship's route is $1 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 6$ (firing 53 canonballs along the way). In total, 86 canonballs are fired during the manoeuvre. Except for the start- and end-point, no vertices or edges are visited by both ships.

Input

For each test case, the input consists of:

- A line containing two integers v $(3 \leq v \leq 1000)$ and e $(3 \leq e \leq 10000)$, the number of waypoints and passages, respectively.
- Then, e lines follow: for each passage, a line containing three integers:
  1. a_i $(1 \leq a_i \leq v)$, the starting-point of a passage, which is represented by a waypoint;
  2. b_i $(1 \leq b_i \leq v)$ and $(a_i \neq b_i)$, the end-point of a passage, which is represented by a waypoint. All passages are directed passages;
  3. c_i $(1 \leq c_i \leq 100)$, the number of cannonballs that are fired when travelling along this passage.

The starting waypoint is 1 and the destination waypoint is v. There are always at least two disjunct routes from waypoint 1 to waypoint v.

Output

For each test case, the output consists of a single positive integer: the smallest possible sum of cannonballs fired by both ships when reaching the destination waypoint.

Example

input	output

6 11
1 2 23
1 3 12
1 4 99
2 5 17
2 6 73
3 5 3
3 6 21
4 6 8
5 2 33
5 4 5
6 5 20
	86

B - Beer Pressure

Ever since the middle of the 20th century, sociologists have been studying the fascinating topic of beer-related behaviour among the student population in Delft. One of the great successes has been the development of a highly accurate model that describes the intricate ritual of *pub selection*, that can be observed in the afternoon after the courses for the day are done.

Students select a pub by casting votes. What makes it interesting from a sociological perspective is the mix of dominant behaviour by some students, and behaviour driven by peer pressure combined with randomness by other students.

Specifically, the model used to describe the pub selection ritual is this:

- The dominant students, who have a strong preference for a particular pub, will (loudly) announce their vote. Several dominant students may vote for the same pub.
- Next, the remaining students vote, one by one. These non-dominant students vote probabilistically and are subject to peer pressure; the probability that they will vote for any specific pub is equal to the number of votes cast for that specific pub so far, divided by the total number of votes cast so far.
- Finally, when all votes are in, the pub with the highest number of votes is selected. If several pubs received the same highest number of votes, one of those is selected at random, with equal probability.

For example, in one particular instance with seven students, the five dominant students start by declaring their votes for three different pubs. Three dominant students proclaim a preference for the first pub; the fourth dominant student announces her preference for a second pub, and the last dominant student votes for the third pub. This leaves the initial vote-count at $(3, 1, 1)$.

After that, the remaining two non-dominant students start to vote, one after the other. The first of them will pick either pub number one with probability $\frac{3}{5}$, pub number two with probability $\frac{1}{5}$, or pub number three with probability $\frac{1}{5}$. He happens to pick pub number three, and the vote-count becomes $(3, 1, 2)$.

Finally, the last student will pick either pub number one with probability $\frac{1}{2}$, pub number two with probability $\frac{1}{6}$, or pub number three with probability $\frac{1}{3}$. She, too, picks pub number three.

This leaves the final vote-count at $(3, 1, 3)$: a tie between pub one and pub three. This tie is broken by a coin-toss; with probability $\frac{1}{2}$, pub one is selected for that evening's drinking.

Problem

You are a sociologist observing the ritual described above. After the dominant students are done declaring their votes, you want to know what the probabilities are for each pub that they will be serving drinks to the students that night.

Input

For each test case, the input consists of two lines:

- a line containing two positive integers: n, denoting the number of pubs in this trial ($n \leq 5$); and k, the total number of students, both dominant and non-dominant ($k \leq 50$).
- a line containing n positive integers $(\alpha_1, \alpha_2, \ldots, \alpha_n)$, denoting the vote-count right after all dominant students have cast their vote.

Furthermore, it is guaranteed that $k \geq \sum_{i=1}^{n} \alpha_i$.

Output

For each test case, write n lines of output containing the probability that the i'th pub will be selected ($1 \leq i \leq n$), ordered by ascending i.

Each line shall be of the form 'pub i: *percentage* %', where *percentage* is a floating point number rounded to 2 digits after the decimal point.

A space should separate the percentage number and the subsequent percent sign.

Example

input	output

`3 7`
`3 1 1`
	`pub 1: 93.33 %`
	`pub 2: 3.33 %`
	`pub 3: 3.33 %`

C - Cycling

On a bicycle trip through the city, a significant amount of time is spent waiting for traffic lights. If only you could reduce this lost time, maybe you would finally manage to get to class in time for the first lecture.

Note that the amount of time lost on a red traffic light is more than just the time spent standing still at the light. After the light turns green, additional time is lost while the bicycle accelerates.

In this problem, we assume a theoretical model of a bicycle trip, based on the following rules:

- The bicycle moves forward or stands still, but it never moves backwards. The bicycle does not have a maximum speed, but you may rest assured that relativistic effects will not be involved in this problem.
- The bicycle can increase speed at a maximum acceleration of 0.5 meters per second per second.
- The bicycle can instantaneously reduce its speed to any value between zero and the current speed.
- The bicycle cannot go through a red light.
- Each traffic light turns red and green according to a fixed, continuously repeating rhythm. (These traffic lights don't turn yellow.)

It should be obvious that the theoretical model deviates from reality in several ways. For example, Dutch cyclists hardly ever stop for red lights. Also, the modelled bicycle can decelerate at an infinite rate, while most student's bikes do not have any braking capability to speak of. We ignore these differences for now and focus on the theory.

Problem

You are standing with your bike at point $X = 0$ at time $T = 0$ with zero speed.
You are in an enormous hurry and would like to arrive at point $X = X_{\text{dest}}$ as soon as possible.

Your task is to find a pattern of accelerating and braking such that you safely pass all traffic lights and arrive in X_{dest} at the earliest possible time.

It is allowed to brake and/or stop at any point during the trip, including (of course) for red traffic lights. However, it may be more efficient to figure out some way in which you can cycle past the traffic lights while they are green.

Input

The input for each test case consists of the following items:

- A line containing a floating point number X_{dest} and an integer L.
  X_{dest} is the total distance to travel in meters ($1 \leq X_{\text{dest}} \leq 10000$);
  L is the number of traffic lights to pass ($0 \leq L \leq 10$).
- L lines describing the traffic lights, listed in order of increasing X position.
  Each of these lines contains 3 floating point numbers:
  - X_i, the position of the traffic light in meters from the start ($0 < X_i < X_{dest}$);
  - R_i, the duration of each red-light period of this traffic light in seconds ($10 \leq R_i \leq 500$);
  - G_i, the duration of each green-light period of this traffic light in seconds ($10 \leq G_i \leq 500$).

All traffic lights turn red at time $T = 0$.
Traffic light i turns green for the first time at $T = R_i$.

There is never more than one traffic light in the same position.

Output

For each test case, write one line of output containing a floating point number: the earliest time at which the cyclist can reach the destination.

The answer should be rounded to 3 digits after the decimal point.

The test cases will be such that very small inaccuracies will not cause errors in the final answer after rounding.

Example

input	output

```
410.0 2
200.0 15.0 15.0
225.0 31.0 10.0
410.0 2
200.0 15.0 15.0
225.0 35.1 15.0
410.0 2
200.0 15.0 15.0
225.0 45.0 10.0
```
	```
	41.497
	52.623
	57.213
	```

D - Digital Clock

Electronic equipment commonly uses 7-segment elements to show numbers. A 7-segment element gets its name because it uses 7 line segments to show a shape: 3 horizontal segments and 4 vertical segments. Each segment can be independently switched on and off, making it possible to show any digit from 0 to 9.

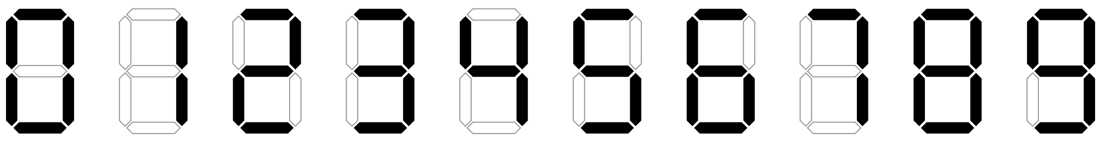


Figure 1: The 7-segment patterns of the decimal digits.

To show more than 1 digit at the same time, a larger display is constructed by putting multiple 7-segment elements next to each other. In this case, each segment in each digit can be independently switched on and off. For example, electronic alarm clocks typically use four 7-segment elements to show the time of day: two digits for the hour (00 to 23) and two digits for the minutes (00 to 59).

You managed to find an old alarm clock in a dusty corner of the basement. Unfortunately, it does not seem to work very well: it does not always show the correct time. You suspect that this is due to faulty wires in the 7-segment display.

As a result of the faults, some of the segments may not be working at all; they never light up, no matter which digits the clock tries to put on its display. On the other hand, the segments that do sometimes light up should all be working correctly. You thus assume that each of the 28 segments on the clock is either completely broken (never lights up) or is working perfectly fine (lights up precisely when it should).

Problem

You want to know what time it is, of course, but the silly clock is not making it very easy. You have been watching the clock for some time, writing down the pattern of digits on its display every minute. (Note that it is possible that the display stays the same for a few minutes. When that happens, you just write down the same pattern several times.)

Your task is to find out what time the clock was really trying to show when you wrote down the first pattern of the sequence. The answer must be consistent with all observations of the clock display, for the first minute as well as for the following minutes, under the assumption that each display segment is either completely broken (never lights up) or is working perfectly fine.

There may be multiple possible answers. In that case, you should make a list of all possible answers, in order of increasing time of day.

It is still possible that the clock is really broken in some other way than just faulty segments. In that case, it may happen that there is no possible answer which is consistent with all observations.

Input

For each test case, there is one line of input containing the following items:

- A positive integer N ($1 \leq N \leq 50$), the number of minutes you have been watching the clock.
- N items, each representing a pattern of digits that was observed on the clock.
  Each pattern is formatted as two decimal digits, followed by a '`:`' character, followed by two more decimal digits.

  The patterns are listed in the order in which they were seen on the clock.

It is theoretically possible for a faulty 7-segment display to show a shape which does not correspond to any of the digits from 0 to 9. However, for some mysterious reason, this never happened during the time you were watching the clock.

Output

For each test case, give one line of output.

If there is at least one possible answer, print a list of all possible answers separated by spaces. Each possible answer must be a valid 24-hour clock time, formatted as two digits (00 to 23), followed by a '`:`' character, followed by two digits (00 to 59).
The list of possible answers must be printed in order of increasing time of day.

If there is no possible answer, print the word '`none`'.

Example

input	output

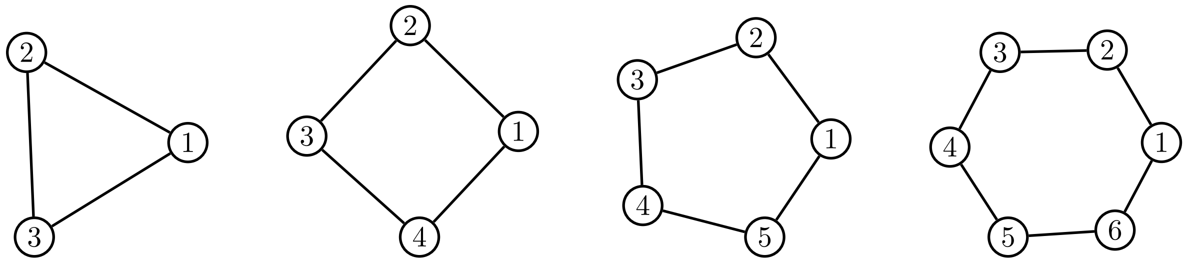
`1 88:88`
`2 23:25 23:26`
`3 71:57 71:57 71:07`
	`none`
	`23:25`
	`00:58 03:58 07:58 08:58`

E - Edge Case

In graph theory, a *matching* or *independent edge set* in a graph $G = (V, E)$ is a set of edges $M \subseteq E$ such that no two edges in the matching M share a common vertex.

Recently you saw in the news that "The Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel" (informally, the Nobel Prize in Economics) for 2012 was awarded to Alvin E. Roth and Lloyd S. Shapley for, amongst other things, their algorithm for finding a matching satisfying certain criteria in a bipartite graph. Since you have also heard that matchings in *cycle graphs* have applications in chemistry your thoughts centre around a plan for a beautiful future where your Christmas shopping is more luxurious than ever!

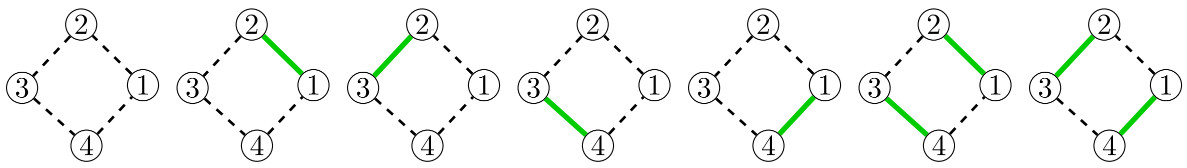
The cycle graph, C_n, $n \geq 3$, is a simple undirected graph, on vertex set $\{1, \ldots, n\}$, with edge set $E(C_n) = \{\{a, b\} \mid |a - b| \equiv 1 \bmod n\}$. It is 2-regular, and contains n edges. The graphs C_3, C_4, C_5, and C_6 are depicted in Figure 1.



(a) C_3 (b) C_4 (c) C_5 (d) C_6

Figure 1: The graphs C_3, C_4, C_5, and C_6.

Your first step towards Nobel Prize fame is to be able to compute the number of matchings in the cycle graph C_n. In Figure 2 the seven matchings of the graph C_4 are depicted.



(a) M_1 (b) M_2 (c) M_3 (d) M_4 (e) M_5 (f) M_6 (g) M_7

Figure 2: The matchings of C_4. The edges that are part of the respective matching are coloured green, while the edges left out of the matching are dashed. $M_1 = \emptyset$, $M_2 = \{\{2, 1\}\}$, $M_3 = \{\{3, 2\}\}$, $M_4 = \{\{4, 3\}\}$, $M_5 = \{\{1, 4\}\}$, $M_6 = \{\{2, 1\}, \{4, 3\}\}$, and $M_7 = \{\{3, 2\}, \{1, 4\}\}$.

Input

For each test case, you get a single line containing one positive integer: n, with $3 \leq n \leq 10000$.

Output

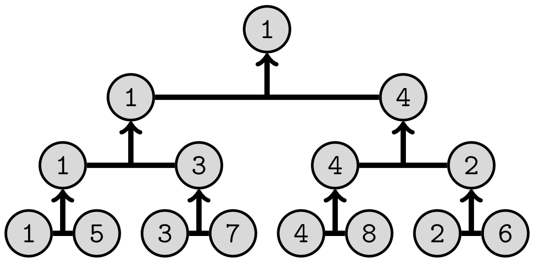
For each test case, a row containing the number of matchings in C_n.

Example

input	output
3	4
4	7
100	792070839848372253127

F - Foul Play

A European soccer tournament has n participating teams. In the first round, $n/2$ matches are played such that each team plays against one other team. After every round, only the winning teams advance to the next round. In the second round, $n/4$ matches are played such that each first-round winner plays against one other first-round winner. Eventually, only two teams are left to play a final match. The winner of the final match is the champion of the tournament.



The Dutch soccer team is probably not the best team in the world. However, they are still a pretty good team. They can easily win from at least half of the other teams. Moreover, for every team t that the Dutch cannot beat in a direct confrontation, there is another team t' that beats t, but is beaten by the Dutch team.

The Dutch coach wants to manipulate the tournament such that the Dutch team will become champion. He happens to know, for each pair of teams, which team would certainly win if a match was played between them.

Problem

For each two teams, you know beforehand which one would win if they played against each other. (Since this is a knock-out tournament, no ties will occur.) Furthermore, you know for sure that your favourite team can beat at least half of the other teams, and for every team t that your favourite team cannot beat, there is a team t' that beats t but is itself beaten by your favourite team.

Determine a tournament schedule such that your favourite team wins the tournament.

Input

For each test case, the input is as follows:

- One line containing the number of teams n, where n is a power of two and $2 \leq n \leq 1024$. Teams are numbered from 1 to n, where team 1 is your favourite team.
- n lines, each containing a string of n binary digits.
  The k-th digit on the j-th line is '1' if team j would certainly win from team k, otherwise it is '0'.

  A team cannot play against itself, therefore the j-th digit on the j-th line is '0'.

  If $j \neq k$, the k-th digit on the j-th line is different from the j-th digit on the k-th line.

Output

For each test case, print $n - 1$ lines of output, specifying a tournament schedule that ensures victory for team 1.

The first $n/2$ lines describe the first round of the tournament. The next $n/4$ lines describe the second round, if any, etc. The last line describes the final match.

Each line contains two integers x and y, indicating that team x plays a match against team y.

If there are multiple tournament schedules where team 1 wins, any one of those tournament schedules will be accepted as a correct answer.

Example

input	**output** (other answers may also be correct)

`4`
`0110`
`0011`
`0000`
`1010`
`8`
`00111010`
`10101111`
`00010010`
`01000101`
`00110010`
`10101011`
`00010000`
`10101010`
	`1 3`
	`2 4`
	`1 2`
	`1 5`
	`3 7`
	`4 8`
	`2 6`
	`1 3`
	`4 2`
	`1 4`

G - Guards

The royal castles in Molvania follow the design of king Sane, first of his dynasty. He ruled by divide and conquer. Therefore, all castles are built according to a hierarchical pattern based on interconnected buildings. A building consists of *halls* and *corridors* that connect halls.

Initially, a castle consists of only one building (the *main building*). When its population grows, the castle is extended as follows: A new peripheral building is constructed, attached to one of the existing buildings. Like any other building, the new building also consists of halls and corridors. An additional corridor is created to connect a hall in the existing building to a hall in the new building. That corridor is the only way to access the new building.

The number of halls in a building is at most 10.

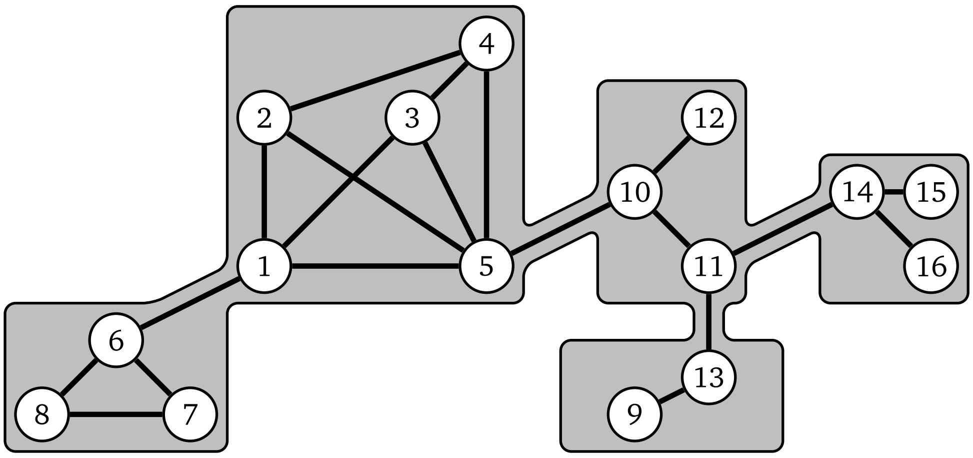


Figure 1: The castle layout of the example provided below.

In times of turmoil, the king monitors all corridors by strategically placing guards in halls. He asks you to determine the least number of guards required to monitor all corridors in the castle (as he wants to keep his personal guard as large as possible). Note that since the last fire, there are no doors in the castle, so we can safely assume that a guard placed in a hall can monitor all connecting corridors.

Input

The input contains a number of castle descriptions. Within a castle, each hall is identified by a unique number between 1 and 10000. Each castle is recursively defined, starting with a description of the main building:

1. A line containing three integers, representing the number of halls in this building ($2 \leq n \leq 10$), the number of corridors in this building ($1 \leq m \leq 45$), and the number of peripheral buildings that were later attached to this building ($0 \leq w \leq 10$).
2. For each of the m corridors:
   - A line containing two integers (each $\leq$ 10000), representing the two halls connected by this corridor. Both halls are located inside the current building.

3. For each of the w peripheral buildings:
   - A line containing two integers, describing the corridor that leads to this peripheral building. The first integer represents a hall in the current building, while the second integer represents a hall in the peripheral building.
   - The structure of the peripheral building and any newer buildings that were later attached to it, described by repeating rules 1 to 3.

The castle is fully connected: any hall is directly or indirectly reachable from any other hall. Corridors with the same start and end hall do not exist, and for every two halls there is at most one corridor between them.

Output

For each castle, print a single line containing a positive integer: the minimum number of guards to place in halls such that all corridors in the castle are monitored.

Example

input	output

```
5 8 2
1 2
2 4
3 4
1 3
1 5
2 5
3 5
4 5
1 6
3 3 0
6 7
7 8
8 6
5 10
3 2 2
10 11
10 12
11 13
2 1 0
13 9
11 14
3 2 0
14 15
14 16
```
	`8`

H - Hip To Be Square

None of the numbers 6, 10, 15 is a square, but their product, the number 900, is a square. We are interested in sets of positive integers, the product of which is a square. We call such a set HIP (this stands for Has Interesting Product). Evidently {6, 10, 15} is HIP, and so is {25}.

More generally, given a set of positive integers, does it have a non-empty subset which is HIP, and if so, for which of the HIP subsets will the product be minimal?

To make things slightly easier for you, we restrict our attention to intervals.

Input

Each test case consists of two integers a and b on a single line ($1 < a < b \leq 4900$). These integers describe the interval $A = \{\, x \in \mathbb{N} \,|\, a \leq x \leq b \,\}$.

Output

For each test case, print the least number k such that the product of the elements of some non-empty subset $X \subseteq A$ equals k^2. If no such number exists, print '`none`'. The number k will be less than 2^{63}.

Example

input	output

`20 30`
`101 110`
`2337 2392`
	`5`
	`none`
	`3580746020392020480`

This page was intentionally left almost blank

I - Idol

Karl is competing in the preliminary round of a talent show called North-Western European Idol (NWEI), and wants to advance to the next round: World Idol. In the talent show, each contestant gets 10 minutes to impress the judges. After all the contestants have performed, each of the judges will cast two distinct votes. A vote can be either in favour of a contestant (meaning this contestant should advance) or against a contestant (meaning this contestant should not advance). The number of contestants that advance to the next round is not known in advance; if there are only very bad contestants, then it is possible that nobody will advance, or if everybody is amazing, then everybody may advance.

Karl is afraid that the judges will not appreciate his programming talents, and hence wants to use his other talent to advance to the next round: hacking. Having gained access to the jury system, Karl is capable of overriding the regular process of counting votes, and instead selecting exactly which contestants advance to the next round. The only problem is, he has to be careful not to arouse suspicion.

Each judge expects that at least one of his (or her) own two votes corresponds to the outcome of the contest. If the outcome contradicts both votes, the judge becomes alarmed. As an example, assume judge Harry casts a vote in favour of Pete and a vote against Sally. If Sally advances and Pete does not, judge Harry will be alarmed and may discover Karl's tampering with the system.

Since Karl's programming talents are limited (otherwise he would not have needed his hacking talents), he needs you to make a program that finds out if there is a set of contestants, which includes himself, that he can select to advance to the next round by hacking the jury system, such that it does not alarm any of the judges.

Input

For each test case, the input is as follows:

- One line containing two positive integers: the number of contestants n ($2 \leq n < 1000$) and the number of judges m ($1 \leq m < 2000$).
- m lines containing the votes of each judge.
  Each of these line contains two integers: the numbers a ($1 \leq |a| \leq n$), and b ($1 \leq |b| \leq n$), the two votes of this judge ($|a| \neq |b|$).

  A vote $x < 0$ means that the vote is against advancement of contestant $|x|$.
  A vote $x > 0$ means that the vote is in favour of contestant $|x|$.

Contestants are numbered $1 \ldots n$.
Karl is contestant 1.

Output

For each test case, print one line of output containing the word '`yes`' if there is a set of contestants that advances to the next round that includes Karl, and does not alarm any of the judges. If there is no such set of contestants, the line should contain '`no`'.

Example

input	output

`4 3`
`1 2`
`-2 -3`
`2 4`
`2 4`
`1 2`
`1 -2`
`-1 2`
`-1 -2`
	`yes`
	`no`

J - Joint Venture

Liesbeth and Jan are building a robot for a course project and have discovered that they need to fit two pieces of Lego into an opening.

The opening is x centimetres wide and the sum of the lengths of the two pieces has to be *precisely* equal to the width of the opening, or else the robot will break during the project demonstration, with catastrophic consequences for the grades of the two students.

Photo by Alan Chia

Luckily, Liesbeth and Jan were able to sneak into the physics laboratory late one night to measure the lengths of their remaining Lego pieces very accurately. Now they just need to select two pieces that will fit the opening perfectly.

Input

For each test case, you get:

- a line containing one positive integer: x, denoting the width of the opening in centimetres, with $1 \leq x \leq 20$.
- a line containing one non-negative integer: n, denoting the remaining number of Lego pieces Liesbeth and Jan have access to, with $0 \leq n \leq 1000000$.
- n lines containing positive integers ℓ, denoting lengths of Lego pieces in nanometres. Liesbeth and Jan have told you that no piece of Lego is longer than 10 centimetres, or 100000000 nanometres.

Output

For each test case, a row containing the word '`danger`' if no two pieces of Lego exist that precisely fit into the opening, or '`yes` ℓ_1 ℓ_2', with $\ell_1 \leq \ell_2$, should two such pieces of lengths ℓ_1 and ℓ_2 exist.

In case multiple solutions exist, a solution maximising the size difference $|\ell_1 - \ell_2|$ must be printed.

Example

input	output
1 4 9999998 1 2 9999999	yes 1 9999999

K - Key Insight

Alice and Bob love to send each other messages, but they don't like it when other people read their messages. Your friend Charles is very interested in what Alice and Bob send to each other, but since Alice and Bob are encrypting their messages he is unable to read them, even though he is able to intercept the encrypted messages (called "ciphertext").

Recently, Charles has not only intercepted an encrypted message, he also knows the original content of this message (which is called "plaintext"). To help him decrypt future messages between Alice and Bob, you are asked to write a program that will help him look for the encryption key.

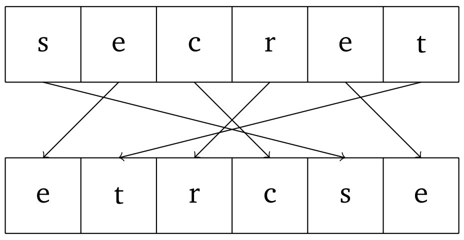


Figure 1: Example of transposition.

Charles has informed you that he knows that Alice and Bob are using a transposition block cipher. This means that for each block of k characters in the message, the characters within the block are re-ordered into one of $k!$ possible permutations during encryption. Each permutation is determined by its unique corresponding encryption key. The key corresponding to the permutation shown in Figure 1 would be some representation of $(123456) \to (514362)$. Since your only task is counting (possible) keys, the actual representation is not relevant.

Fortunately, Charles does know the block size k, and he knows that the plaintext and ciphertext that he intercepted consist of one or more full blocks of length k (i.e., no incomplete blocks) that have each been encrypted with the same key.

Given the plaintext M and ciphertext C that Charles has intercepted, your program will compute the number of possible encryption keys.

Input

For each test case, the input contains three lines:

- One line containing a positive integer k, the block size ($k \geq 1$).
- One line containing M, the plaintext ($1 \leq |M| \leq 100,\ |M|$ is a multiple of k).
- One line containing C, the ciphertext ($|C| = |M|$).

Both plaintext and ciphertext consist only of lower-case letters.

Output

For each test case, print one line containing the number of possible encryption keys of size k. This number will not exceed $2^{63} - 1$. If it is impossible to obtain M from C by a transposition cipher of block size k, print '0' (the number zero).

Example

input	output
4 treewood ertedowo 1 nwerc ncrew 6 secret etrcse 1 impossibru youdontsay	1 0 2 0