# Problem A. Buffered Buffet

| | |
|---|---|
| Source file name: | Buffered.c, Buffered.cpp, Buffered.java, Buffered.py |
| Input: | `Standard` |
| Output: | `Standard` |

Finally, you are allowed again to invite your friends over for a delicious buffet! However, you still need to take social distancing measures into account. To complicate matters further, your friends are coming from various different countries. Each of these countries has slightly different social distancing rules. For example, your friends from the US are required to sit at least 2 meters from another person, while your friends from France only need to sit 1 meter apart from others.

CC-BY 2.0 By Mike Fleming on https://flic.kr/p/5NYZPq

You would like for all your friends to sit around one large circular table. You want to position your friends at this table in such a way that everyone complies with the social distancing requirements of their respective country. To make matters a bit easier, the distance between two people sitting at the table is calculated along the circumference. What is the minimum circumference of the table such that everyone can comply with their social distancing requirement?

## Input

The input consists of:

- One line containing an integer $n$ ($2 \leq n \leq 10^5$), the number of people sitting at the table (yourself included).
- One line containing $n$ integers $d_1, \ldots, d_n$ ($1 \leq d_i \leq 10^9$), how many meters each person needs to sit apart from others (measured along the circumference of the table).

## Output

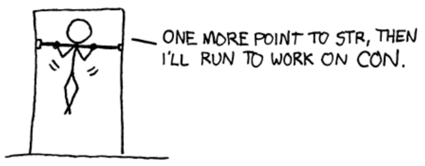
Output the minimum circumference of the table in meters such that everyone can sit at the table while respecting their social distancing measures.

## Example

| Input | Output |
|---|---|
| 2<br>2 3 | 6 |
| 3<br>2 2 3 | 8 |

# Problem B. BnPC

| | |
|---|---|
| Source file name: | Bnpc.c, Bnpc.cpp, Bnpc.java, Bnpc.py |
| Input: | `Standard` |
| Output: | `Standard` |



CC BY-NC 2.5 by xkcd.com

You are playing your favorite game, Basements and Pigeonlike Creatures, for the umpteenth time. You know the game pretty well, but you have never spent enough time on it to figure out the best strategy. That is, until now. The game consists of a certain sequence of events, such as battling a monster or saving a cat from a tree, and you need to complete all events to win. Attached to each event is an attribute, such as strength, and a threshold, some positive integer. If your attribute score matches or exceeds the threshold, you successfully complete the event! If not, it is unfortunately game over and your total score will be zero.

If you complete all the events successfully, your score depends on how well you did during these events. If your attribute score matches the threshold of an event exactly, you get 0 points, barely scraping by that event. If you exceed the threshold, you get points equal to your attribute score that was used for that event.

You are now at the final part of the game, but first you have some attribute points to spend to increase your attribute scores. You know what events will happen during the final part of the game, so all that is left is to figure out what attributes to increase.

## Input

The input consists of:

- One line containing an integer $n$ ($1 \leq n \leq 10^5$) and an integer $k$ ($1 \leq k \leq 10^9$), the number of attributes and the number of attribute points you can still spend.
- $n$ lines, each containing a distinct attribute name, and an integer $s$ ($0 \leq s \leq 10^9$), the current score you have in that attribute.
- One line containing an integer $l$ ($1 \leq l \leq 10^5$), the number of events.
- $l$ lines each describing one event, containing the name of the attribute that is used, and an integer $t$ ($0 \leq t \leq 10^9$), the threshold for this event.

Attribute names consist of upper case English letters (`A-Z`), and have a length between 1 and 20 characters inclusive.

## Output

Output the maximum score you can get from the events.

## Example

| Input | Output |
|---|---|
| 2 3<br>STR 15<br>CON 12<br>2<br>STR 17<br>CON 14 | 0 |
| 3 7<br>JUMP 5<br>RUN 7<br>FLY 0<br>4<br>FLY 0<br>JUMP 6<br>RUN 10<br>RUN 8 | 31 |

# Problem C. Cangaroo

| | |
|---|---|
| Source file name: | Cangaroo.c, Cangaroo.cpp, Cangaroo.java, Cangaroo.py |
| Input: | `Standard` |
| Output: | `Standard` |

Pixabay License by pen_ash on Pixabay

Let us talk about the big elephant in the room[1]: you've had a kangaroo in your room for a while now and you need to hide it without raising suspicion, since you want to keep the animal. Hiding an animal of this size is difficult: if you use a lot of space, it is obvious that you are hiding something from your friends. Hence, you want to use as little space as possible to hide the kangaroo.

When the kangaroo was placed against the wall, you took a black and white picture of the animal. Looking around in the house, the only tools you found to hide the kangaroo with were empty tin cans. The dimensions of the tin cans correspond with $2 \times 2$ pixels in the picture and these cans cannot overlap. So, you can make a *cangaroo* and if someone asks why you have cans in the shape of a kangaroo, you simply say it is a bad joke of yours.

The position of each can has to exactly correspond to a block of $2 \times 2$ pixels in the picture, and they cannot be shifted or rotated to only partially cover some pixels. Furthermore, cans cannot float in the air, so every can has to be supported either by the floor, which is just below the bottom row of the picture, or by another can, for which at least one of the left and right half must directly rest on another can. The structure does not otherwise need to be balanced.

What is the minimum number of cans needed to hide the kangaroo?

## Input

The input consists of:

- One line containing two integers $n$ $(2 \le n \le 100)$ and $m$ $(2 \le m \le 10)$, the height and width of your room. Both $n$ and $m$ are even.
- $n$ lines, each containing $m$ characters that are either '`.`' or '`#`', where '`#`' marks a position that needs to be hidden by a can.

## Output

Output the minimal number of $2 \times 2$ cans that is required to hide the kangaroo in the room.

[1]You also wanted an elephant but this did not fit with the kangaroo in your room, sadly.

## Example

| Input | Output |
|---|---|
| `4 4` | `3` |
| `....` | |
| `..#.` | |
| `.##.` | |
| `##..` | |
| `4 4` | `4` |
| `#.#.` | |
| `....` | |
| `#...` | |
| `....` | |
| `14 8` | `15` |
| `........` | |
| `....##..` | |
| `...###..` | |
| `....##..` | |
| `.....#..` | |
| `...####.` | |
| `..#####.` | |
| `.######.` | |
| `.#####..` | |
| `.###....` | |
| `.##.....` | |
| `..##....` | |
| `...#....` | |
| `.###....` | |

## Problem D. Decelerating Jump

| | |
|---|---|
| Source file name: | Decelerating.c, Decelerating.cpp, Decelerating.java, Decelerating.py |
| Input: | `Standard` |
| Output: | `Standard` |

An athlete is participating in a new sport that is the perfect mix of hopscotch and triple jumping. For this jury sport, $n$ squares are laid out on the ground in a line, with equal distances between them. The first phase is the approach, where an athlete sprints towards the first square, in which they start their first jump. Then, they may jump on any number of other squares, and must finally land in the last square.

A typical hopscotch court, the inspiration for this new sport.

The jury has given a predetermined number of points to jumping in each square, and the score of the athlete will be the sum of the scores of all the squares they jump in, including the very first and last squares.

Due to the nature of this sport, once the athlete starts jumping, they can not accelerate anymore, and the length of consecutive jumps can never increase. Naturally, it is also impossible to reverse direction.

Given the points the jury has allocated to each square, find the maximal possible score an athlete can get on this event.

### Input

The input consists of:

- One line with an integer $n$ ($2 \leq n \leq 3000$), the number of squares.
- One line with $n$ integers $p_1, p_2, \ldots, p_n$ ($-10^9 \leq p_i \leq 10^9$), the number of points the jury awards for jumping on each of the squares.

### Output

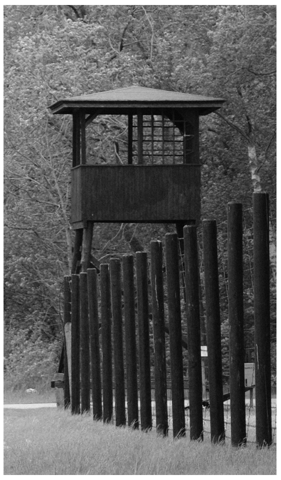
Output the maximum score that an athlete can get.

### Example

| Input | Output |
|---|---|
| 4<br>1 -1 1 1 | 3 |
| 4<br>1 1 -1 1 | 2 |
| 3<br>-1 1 -1 | -1 |

## Problem E. Entering Enemy Encampment

| | |
|---|---|
| Source file name: | Encampment.c, Encampment.cpp, Encampment.java, Encampment.py |
| Input: | `Standard` |
| Output: | `Standard` |


Watchtower by Ragnar Groot Koerkamp

A new two-player game simulates two countries fighting over a territory. This territory contains a number of strategic positions which are suitable for setting up war camps. The players take turns setting up a camp at one of these spots.

In this territory, there are also a number of trails that connect two positions each. Whenever a player sets up a new camp, this player will send a small group of soldiers over every adjacent trail that leads to an enemy encampment. Each of these groups of soldiers will raid the enemy camp. In other words, for every trail, the second player to capture one of the incident positions can launch at most one raid over this trail. When every strategic position has been claimed, the game ends. The player that performed the most raids wins the game.

Assuming both players play optimally, who wins?

### Input

The input consists of:

- One line containing two integers $n$ ($1 \le n \le 20$), the number of strategic positions, and $m$ ($0 \le m \le \frac{n(n-1)}{2}$), the number of trails.
- $m$ lines, each containing two integers $a$ and $b$ ($1 \le a, b \le n$, $a \ne b$), indicating that there is a trail between position $a$ and position $b$.

There is at most one trail between every pair of strategic positions.

### Output

If the player who goes first wins, output "`player 1`". If the player who goes second wins, output "`player 2`". If the players tie, output "`tie`".

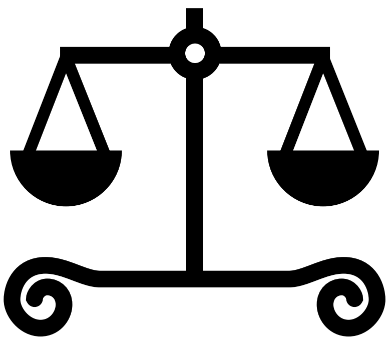
### Example

| Input | Output |
|---|---|
| 3 3<br>1 2<br>2 3<br>1 3 | tie |
| 2 1<br>1 2 | player 2 |
| 5 7<br>3 4<br>2 1<br>4 5<br>1 4<br>1 3<br>2 3<br>2 4 | player 1 |

# Problem F. Fair Play

| | |
|---|---|
| Source file name: | Fairplay.c, Fairplay.cpp, Fairplay.java, Fairplay.py |
| Input: | `Standard` |
| Output: | `Standard` |

Together with your coworker, Larry, you are organizing the exciting Billiards and Pool Competition for your coworkers in your small company. You and Larry are usually on the same page, and surely he will approve of your latest idea. You even bought a nice prize for your coworkers to win and you hope that they are as excited as you are. You want to maximise fun.


Balanced scales
(from WikiMedia Commons)

It would thus be nice to try to avoid complete walkovers: that is no fun for either player. After some thought, you think it is good to suggest to Larry to divide the players into groups of two. That way, you can compensate for a player's strength by pairing them with a weaker player. In fact, it would be perfect if every team had the exact same strength! Before you tell Larry your plans, you decide to first figure out whether this is possible.

According to your model, synergy plays a negligible role in determining team strength, and the strength of a team is simply determined by the strength of its individual members. Every coworker has a certain skill level in both billiards and pool, as indicated by two integers. When two coworkers are teamed up, their total skill is the sum of their individual skills. Can you divide everyone into teams of two such that every team has the exact same skill in both pool and billiards?

## Input

The input consists of:

- A line with an integer $n$ ($2 \leq n \leq 10^5$), the number of coworkers you have.
- Then follow $n$ lines containing two integers $b$ and $p$ ($-10^6 \leq b, p \leq 10^6$), the skill in billiards and pool, respectively, of each coworker.

## Output

Output "`possible`" if it is possible to divide all coworkers into teams of two with equal skill. Output "`impossible`" otherwise.

## Example

| Input | Output |
|---|---|
| 6<br>2 1<br>3 0<br>3 0<br>4 2<br>4 2<br>5 1 | possible |
| 4<br>1 0<br>0 1<br>-2 0<br>0 -2 | impossible |

## Problem G. Group Project

| | |
|---|---|
| Source file name: | Group.c, Group.cpp, Group.java, Group.py |
| Input: | `Standard` |
| Output: | `Standard` |

The big day has finally arrived: today you are going to form groups of two in which you will do the end-of-the-year project. When you arrive at school, you learn that the teacher of the other class is sick, and that your teacher, Mr. B.A.P. Cee, will also have to make groups for the other class. Mr. B.A.P. Cee is a smart guy and realizes that he can use these unfortunate circumstances to his advantage.

CC-BY-NC 2.0 By Mark Knobil, knobil on Flickr

Ending up with groups of one should be avoided at all cost, so mixing the students of the two classes may avoid this situation. However, while it is easy to pair up two students from the same class, it is more difficult to match up students from different classes. Throughout the years there has been a lot of rivalry between the two groups, and many students dislike students in the other class. Mr. B.A.P. Cee knows which pairs of students will result in a fight and a failed project.

You are given a list of pairs of students who cannot work together. How many disjoint groups of two can Mr. B.A.P. Cee make that will not result in a failed project?

### Input

The input consists of:

- A line with two integers $n$ ($1 \leq n \leq 10^5$), the number of students, and $m$ ($0 \leq m \leq 2 \cdot 10^5$), the number of pairs of students who cannot work together.
- $m$ lines, each with two distinct integers $i$ and $j$ ($1 \leq i, j \leq n$, $i \neq j$), giving a pair of students who cannot work together.

Students are identified by the numbers 1 through $n$. It is guaranteed that it is possible to split the students into two classes in such a way that all students from the same class get along.

### Output

Output the number of pairs of students Mr. B.A.P. Cee can make without making any pair of students who cannot work together.

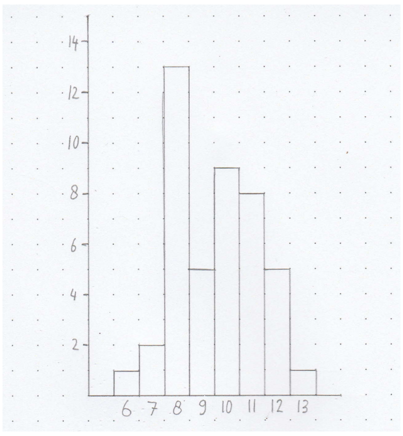
## Example

| Input | Output |
|---|---|
| 3 2 | 1 |
| 1 2 | |
| 3 1 | |
| 5 6 | 2 |
| 1 4 | |
| 2 4 | |
| 3 4 | |
| 1 5 | |
| 2 5 | |
| 3 5 | |
| 6 6 | 3 |
| 1 4 | |
| 2 5 | |
| 3 6 | |
| 1 5 | |
| 3 5 | |
| 2 6 | |

## Problem H. Histogram

| | |
|---|---|
| Source file name: | Histogram.c, Histogram.cpp, Histogram.java, Histogram.py |
| Input: | `Standard` |
| Output: | `Standard` |

The term “histogram” was first introduced in 1895 by Karl Pearson. This is also the year that the portable electric drill was invented. It may not be a coincidence that we speak about “data drilling” today. Most things back then were not electric however, and histograms were still hand crafted. Nowadays, we like things electric and automated. The Bureaucratic Affiliation for Printing Charts needs to create a lot of histograms for a lot of data points, so they come to you for help to automate the printing of histograms.



Manually drawn histogram for the number of problems in the BAPC and its preliminaries, 1997–2020. *If only we could automate this...*

Given a list of data points, print a histogram.

### Input

The input consists of:

- One line containing three integers $n$, $s$, and $k$ ($1 \le n, s, k \le 1000$), the number of bins, the size per bin, and the number of data points, respectively.
- One line containing $k$ integer data points $x$ ($1 \le x \le n \cdot s$).

### Output

Output the histogram. That is, write a rectangle of characters. The rectangle should be as wide as the number of bins. Each column of the rectangle should have

- a “`-`” at the bottom;
- on top of that, as many “`#`” as there are items in the bin;
- on top of that, as many “`.`” as are needed to fill out the rectangle.

The rectangle should be just high enough to display a “`#`” for each element in every bin, but no higher.

### Example

| Input | Output |
|---|---|
| `3 1 6`<br>`1 2 3 2 1 2` | `.#.`<br>`##.`<br>`###`<br>`---` |
| `3 10 5`<br>`1 9 11 29 24` | `#.#`<br>`###`<br>`---` |

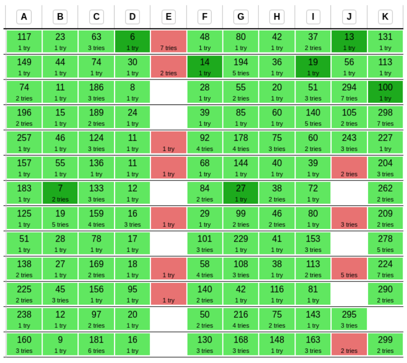
## Problem I. Implementation Irregularities

| | |
|---|---|
| Source file name: | Irregularities.c, Irregularities.cpp, Irregularities.java, Irregularities.py |
| Input: | `Standard` |
| Output: | `Standard` |

Impressed by the performance of the top teams at the recent BAPC preliminaries, you started to wonder whether teams were allowed to use one or multiple computers to implement their solutions.

Instead of unnecessarily bothering the organization with more questions, you will figure this out by yourself. Being a jury member, you already have estimates for the computer time required to solve each problem.

Using this information, and the time in the contest at which the top team solved each of their solved problems, compute the minimal number of computers used by the team.



| A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|
| 117 1 try | 23 1 try | 63 3 tries | 6 1 try | 7 tries | 48 1 try | 80 1 try | 42 1 try | 37 2 tries | 13 1 try | 131 1 try |
| 149 1 try | 44 1 try | 74 1 try | 30 1 try | 2 tries | 14 1 try | 194 5 tries | 36 1 try | 19 1 try | 56 1 try | 113 1 try |
| 74 2 tries | 11 1 try | 186 3 tries | 8 1 try | | 28 1 try | 55 2 tries | 20 1 try | 51 3 tries | 294 7 tries | 100 1 try |
| 196 2 tries | 15 1 try | 189 2 tries | 24 1 try | | 39 1 try | 85 1 try | 60 1 try | 140 5 tries | 105 2 tries | 298 7 tries |
| 257 1 try | 46 1 try | 124 3 tries | 11 1 try | 1 try | 92 4 tries | 178 4 tries | 75 3 tries | 60 2 tries | 243 3 tries | 227 1 try |
| 157 1 try | 55 1 try | 136 1 try | 11 1 try | 1 try | 68 1 try | 144 1 try | 40 1 try | 39 1 try | 2 tries | 204 3 tries |
| 183 1 try | 7 2 tries | 133 3 tries | 12 1 try | | 84 2 tries | 27 1 try | 38 2 tries | 72 1 try | | 262 2 tries |
| 125 1 try | 19 5 tries | 159 4 tries | 16 3 tries | 1 try | 29 1 try | 99 2 tries | 46 2 tries | 80 1 try | 3 tries | 209 2 tries |
| 51 1 try | 28 1 try | 78 1 try | 17 1 try | | 101 3 tries | 229 1 try | 41 1 try | 153 3 tries | | 278 5 tries |
| 138 2 tries | 27 1 try | 169 2 tries | 18 1 try | 1 try | 58 4 tries | 108 3 tries | 38 1 try | 113 2 tries | 5 tries | 224 7 tries |
| 225 2 tries | 45 3 tries | 156 1 try | 95 1 try | 1 try | 140 2 tries | 42 1 try | 116 1 try | 81 1 try | | 290 2 tries |
| 238 1 try | 12 1 try | 97 2 tries | 20 1 try | | 50 2 tries | 216 4 tries | 75 2 tries | 143 1 try | 295 3 tries | |
| 160 3 tries | 9 1 try | 181 6 tries | 16 1 try | | 130 3 tries | 168 3 tries | 148 1 try | 163 3 tries | 2 tries | 299 2 tries |

Scoreboard of the BAPC 2021 preliminaries

The team may work on multiple problems before getting any one of them accepted. Furthermore, the contestants are great multitaskers and can work on a single problem using multiple computers at the same time, but each computer can only be used for one problem at a time.

## Input

The input consists of:

- One line containing an integer $n$ ($1 \leq n \leq 10^5$), the number of problems in the contest.
- One line containing $n$ integers $t_1, t_2, \ldots, t_n$ ($1 \leq t_i \leq 10^4$), the computer time required to solve problem $i$.
- One line containing $n$ integers $s_1, s_2, \ldots, s_n$ ($1 \leq s_i \leq 10^9$ or $s_i = -1$), the time at which problem $i$ was solved, or $-1$ if it was not solved.

It is guaranteed that the team solved at least one problem.

## Output

Output the minimum number of computers used by the team.

## Example

| Input | Output |
|---|---|
| `11`<br>`50 8 10 6 300 5 6 3 18 5 12`<br>`117 23 63 6 -1 48 80 42 37 13 131` | `1` |
| `1`<br>`10`<br>`3` | `4` |
| `2`<br>`2 4`<br>`3 3` | `2` |
| `2`<br>`4 6`<br>`10 10` | `1` |

# Problem J. Jail or Joyride

| | |
|---|---|
| Source file name: | Joyride.c, Joyride.cpp, Joyride.java, Joyride.py |
| Input: | `Standard` |
| Output: | `Standard` |

imgflip.com/memegenerator
/Left-Exit-12-Off-Ramp

A group of teenagers has stolen a fast sports car for a Saturday night joyride. The local police department has only one car available to catch the teenagers red handed and put them in a youth detention center.

The city consists of a set of junctions and bidirectional roads, each of a certain length. The teenagers stay at a certain junction until just before the police car arrives at this junction. At that moment, the teenagers want to get to a junction as far as possible from their current location, without using the road the police car is on. They quickly look at a map to determine all junctions within the city which are reachable without using the road with the police car. Then the teenagers determine the distance to each of these junctions using their satnav system and randomly pick one of the furthest located junctions. Note that the satnav system does not know about the location of the police car, and will not take it into account when computing the distance. The sports car then drives instantly to that junction using any route which does not pass by the police car, while the police is left behind dumbfounded. The youngsters will wait there until the police car makes a new approach. The only way for the police to catch the teenagers is by approaching them while they are in a dead end (a junction with only one incoming road). Figure 1 shows how the police can capture the teenagers in the first sample case.

Since time is precious for the police, they need you to find out if it is possible to catch the joyriders with absolute certainty. And if so, what is the minimal distance they need to drive to be guaranteed to catch the youngsters, assuming the police uses an optimal strategy?

## Input

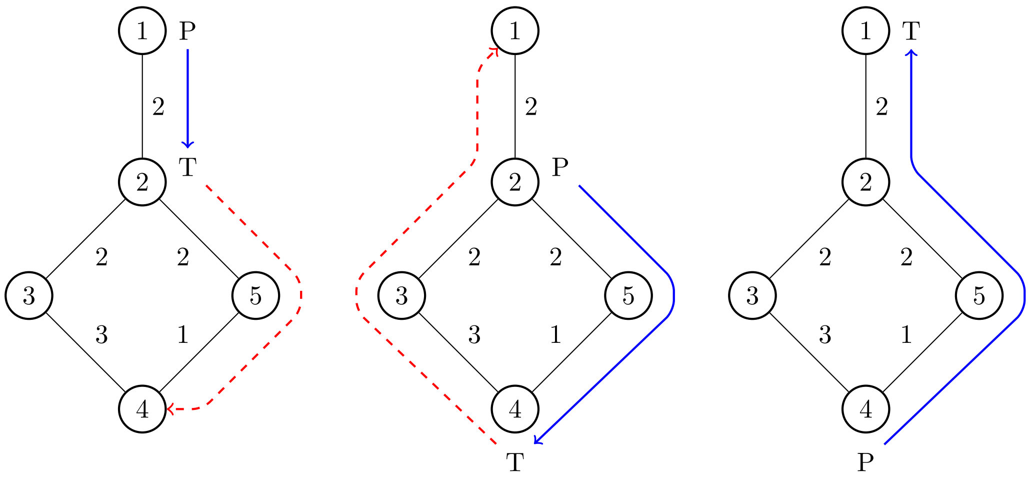
The input consists of:

- One line containing four integers: $n$ ($2 \leq n \leq 300$), the number of junctions, $m$ ($1 \leq m \leq \frac{n(n-1)}{2}$), the number of roads, $p$ ($1 \leq p \leq n$) the initial position of the police car, and $t$ ($1 \leq t \leq n$, $t \neq p$) the initial position of the group of teenagers.
- Then follow $m$ lines, each containing three integers $a$, $b$ and $\ell$ ($1 \leq a, b \leq n$, $a \neq b$, and $1 \leq \ell \leq 10^9$), indicating a road between junctions $a$ and $b$ with a length of $\ell$.

There is at most one road between every pair of junctions and you can reach any junction from any other junction by making use of the roads.

## Output

If it is possible to catch the teenagers with absolute certainty, output the minimal distance that the police car needs to cover to achieve this. Otherwise, output "`impossible`".



Possible movements of the police (P) and teenagers (T) in the first sample case. The movement of the police (solid blue arrows) takes time according to the length of the edges, while the movement of the teenagers (dashed red arrows) is instant.

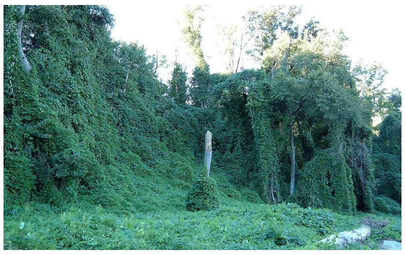
## Example

| Input | Output |
|---|---|
| 5 5 1 2<br>1 2 2<br>2 3 2<br>3 4 3<br>4 5 1<br>2 5 2 | 10 |
| 5 5 1 3<br>1 2 2<br>2 3 2<br>3 4 3<br>4 5 1<br>2 5 2 | impossible |

## Problem K. Kudzu Kniving

| | |
|---|---|
| Source file name: | Kudzu.c, Kudzu.cpp, Kudzu.java, Kudzu.py |
| Input: | `Standard` |
| Output: | `Standard` |

You can't deny it anymore: the kudzu vines in your garden have grown out of control. Some years ago, you planted a single seedling which you received as a gift from your Mathematics teacher. You vaguely remember her explaining how it grows:


Kudzu overgrowing trees
By Scott Ehardt on Wikipedia

- It started as a single root vertex, labelled 0.
- Every year, it grows some new vertices and edges: if at the start of the year it has $n$ vertices, then during this year, it will grow a new edge and vertex from each of the $n$ vertices. If an old vertex had index $v$, then the new vertex which grows from it will have index $v + n$.
- It can be shown that after $i$ years, your kudzu plant has exactly $2^i$ vertices, numbered 0 to $2^i - 1$.

Today, it is time to get out your machete knife and remove a number of branches (subtrees) from the kudzu, one by one. You plan on pruning the tree in a fancy shape, which will probably not stay intact for a long time given how fast your kudzu is growing, but at least you promise yourself to keep pruning every year. After deciding which branches you want to cut off today, you call up the Branching And Pruning Company to ask if they can dispose of the plant waste. They want to know exactly how much they need to clean up. You feel like you should be able to compute that, but how?

When a subtree rooted at some vertex $v$ is removed, this means that vertex $v$ will be removed, together with all vertices which have grown from it (and the vertices which have grown from those, and so on). Figure 2 shows this process for the second sample case.

Given the indices of the roots of the subtrees which you will remove, compute the number of vertices which will be removed for each of these removed subtrees. Since these numbers may be large, you should find them modulo $10^9 + 7$.

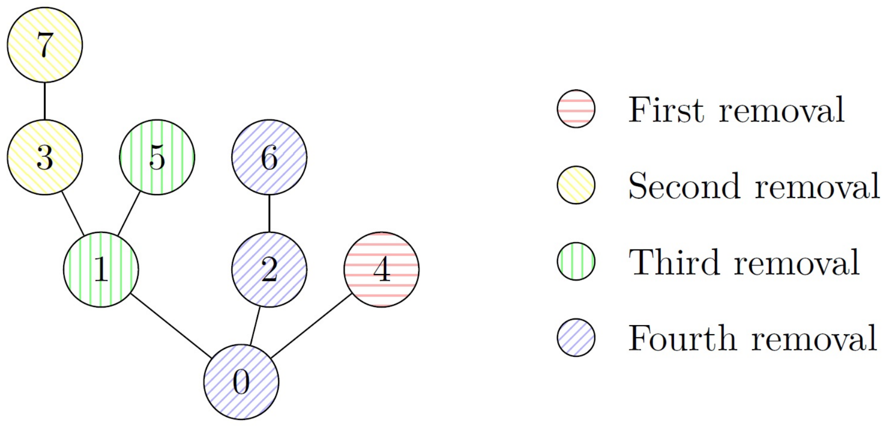
## Input

The input consists of:

- One line containing two integers $a$ ($0 \leq a \leq 10^6$), the age of the tree in years, and $m$ ($1 \leq m \leq 10^5$), the number of subtrees which will be removed.
- $m$ lines, each with an integer $v$ ($0 \leq v \leq 10^9$), the index of a vertex to be removed from the tree. It is guaranteed that $v$ will not yet have been removed.

## Output

Output $m$ lines. The $i$th line should contain the number of vertices removed in the $i$th removal, modulo $10^9 + 7$.



The tree of the second example case. The different colours indicate in which removal the vertices are removed.

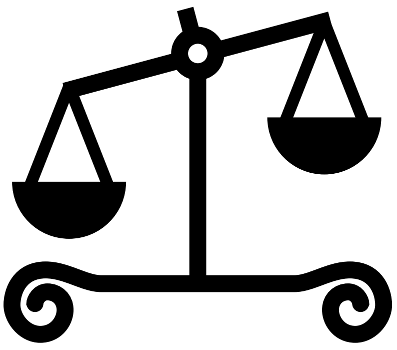
## Example

| Input | Output |
|---|---|
| 4 1<br>0 | 16 |
| 3 4<br>4<br>3<br>1<br>0 | 1<br>2<br>2<br>3 |
| 5 5<br>6<br>3<br>1<br>18<br>2 | 4<br>8<br>8<br>1<br>3 |
| 42 1<br>0 | 46480318 |

## Problem L. Lopsided Lineup

| | |
|---|---|
| Source file name: | Lopsided.c, Lopsided.cpp, Lopsided.java, Lopsided.py |
| Input: | `Standard` |
| Output: | `Standard` |

Together with your coworker, Sergey, you are organizing the exciting Billiards and Pool Competition for your coworkers in your small company. However, communication has not been great between you two. You are not sure you and Sergey think alike, but as far as you are concerned, this would be a great opportunity to do some team building. The actual prizes are meaningless, but there is possibly a lot to be gained from this in team bonding. You want to maximise result.


Unbalanced scales
(from WikiMedia Commons)

You start reading some pseudo-scientific books on team management, and after some research, you conclude that there are two good ways of team bonding: people feel more connected after either a triumphant victory or a crushing defeat. This gives you a great idea: if you divide your coworkers into two groups that are as far apart in skill level as possible, both teams will experience improved bonding! You therefore think it is optimal to try to make the teams as unbalanced as possible. Make sure, however, that the teams are of equal size.

With a bit of work you come up with a nice model for the strength of a team. You think team strength is mainly determined by how well two players play together, whether they encourage one another and complement each other's weaknesses. Whenever two players $i$ and $j$ are in the same team, they increase the team score by an integer $c_{i,j}$. The total score of a team is thus equal to the sum of $c_{i,j}$, over all unordered pairs of players $i$ and $j$ in the team.

### Input

The input consists of:

- One line with an even integer $n$ ($2 \leq n \leq 1000$), the total number of players.
- $n$ lines, the $i$th of which contains $n$ integers $c_{i,1}, c_{i,2}, \ldots, c_{i,n}$ ($-10^6 \leq c_{i,j} \leq 10^6$).
  For any $i$ and $j$, it is guaranteed that $c_{i,i} = 0$ and $c_{i,j} = c_{j,i}$.

### Output

Output the maximum possible difference in strength between two teams of equal size.

## Example

| Input | Output |
|---|---|
| 6 | 0 |
| 0 4 -6 2 3 -3 | |
| 4 0 2 -6 0 0 | |
| -6 2 0 0 2 2 | |
| 2 -6 0 0 -1 5 | |
| 3 0 2 -1 0 -4 | |
| -3 0 2 5 -4 0 | |
| 4 | 6 |
| 0 1 2 2 | |
| 1 0 8 -3 | |
| 2 8 0 5 | |
| 2 -3 5 0 | |