Nordic Collegiate Programming Contest

NCPC 2019

October 5, 2019

Problems

Do not open before the contest has started.

Advice, hints, and general information

- The problems are **not** sorted by difficulty.
- Your solution programs must read input from *standard input* (e.g. `System.in` in Java or `cin` in C++) and write output to *standard output* (e.g. `System.out` in Java or `cout` in C++). For further details and examples, please refer to the documentation in the help pages for your favorite language on Kattis.
- For information about which compiler flags and versions are used, please refer to the documentation in the help pages for your favorite language on Kattis.
- Your submissions will be run multiple times, on several different inputs. If your submission is incorrect, the error message you get will be the error exhibited on the first input on which you failed. E.g., if your instance is prone to crash but also incorrect, your submission may be judged as either "Wrong Answer" or "Run Time Error", depending on which is discovered first. The inputs for a problem will always be tested in the same order.
- If you think some problem is ambiguous or underspecified, you may ask the judges for a clarification request through the Kattis system. The most likely response is "No comment, read problem statement", indicating that the answer can be deduced by carefully reading the problem statement or by checking the sample test cases given in the problem, or that the answer to the question is simply irrelevant to solving the problem.
- In general we are lenient with small formatting errors in the output, in particular whitespace errors within reason, and upper/lower case errors are often (but not always) ignored. But not printing any spaces at all (e.g. missing the space in the string "`1 2`" so that it becomes "`12`") is typically not accepted. The safest way to get accepted is to follow the output format exactly.
- For problems with floating point output, we only require that your output is correct up to some error tolerance. For example, if the problem requires the output to be within either absolute or relative error of 10^{-4}, this means that

  - If the correct answer is 0.05, any answer between 0.0499 and $.0501$ will be accepted.
  - If the correct answer is 500, any answer between 499.95 and 500.05 will be accepted.

  Any reasonable format for floating point numbers is acceptable. For instance, "17.000000", "0.17e2", and "17" are all acceptable ways of formatting the number 17. For the definition of reasonable, please use your common sense.
- An *interactive* problem is a problem where your program will be communicating back and forth with a judge program, e.g. playing a game against an opponent.

  In these problems, you still read from standard input and write to standard output just as in other problems, but for these problems you need to also make sure to *flush* your output after sending a message to the judge program (otherwise it may just be placed in an internal buffer in your program and not sent). If you are using output streams in Java or C++ this can be done by calling the `.flush()` method of your output stream.

  For interactive problems we sometimes also provide a simple testing tool program that you can use to test-run your solution. When available you will find this tool under "Download" to the right of the problem statement on the page for the problem in Kattis. Note that this tool is generally only meant to facilitate basic testing of your program, and that your solution might not necessarily be tested in the same way when submitted.

Problem A
Alphabet Animals
Time limit: 2 seconds

You are playing a game in which a group of players take turns saying animal names. The animal name you say when it is your turn must start with the same letter as the previously said animal ends with and it must not have been said previously in this round of the game. If there is no valid name or you cannot come up with one you are eliminated.

Given the last animal name said before your turn and a list of all names not yet used, can you make it through this turn? If so, can you make sure to eliminate the next player?

Solution to Sample Input 1 by Kuebi via Wikimedia Commons, cc by-s

Input

The first line of input contains a single word, the animal that the previous player just said. The next line contains a single integer n ($0 \leq n \leq 10^5$), the number of valid unused animal names. Each of the following n lines contains one valid unused animal name.

All animal names (including the one the previous player said) are unique and consist of at least 1 and at most 20 lower case letters '`a`'-'`z`'.

Output

If there is any animal name you can play that eliminates the next player, output the first such name from the input list, followed by an exclamation mark. Otherwise, if there is any animal name that you can play, output the first such name. Otherwise, output a question mark (in this case you will just have to make up a fake name in the hope that the others will trust you that this is a real animal).

Sample Input 1
```
pig
2
goat
toad
```

Sample Output 1
```
goat
```

Sample Input 2
```
dog
2
snake
emu
```

Sample Output 2
```
?
```

Sample Input 3
```
hare
3
bee
cat
eagle
```

Sample Output 3
```
eagle!
```

This page is intentionally left blank.

Problem B
Building Boundaries
Time limit: 1 second

Maarja wants to buy a rectangular piece of land and then construct three buildings on that land.

The boundaries of the buildings on the ground must have rectangular sizes $a_1 \times b_1$, $a_2 \times b_2$, and $a_3 \times b_3$. They can touch each other but they may not overlap. They can also be rotated as long as their sides are horizontal and vertical.

What is the minimum area of land Maarja has to buy?

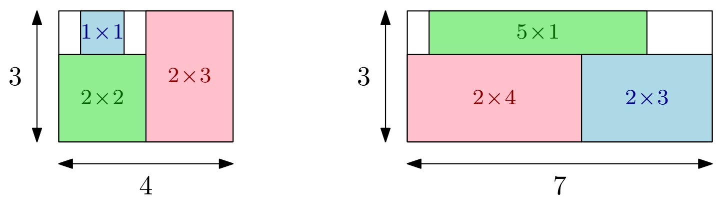


Figure B.1: Illustration of the two test scenarios in Sample Input 1 and their solutions. In the second scenario the 5×1 building has been rotated by 90 degrees.

Input

The input consists of multiple test scenarios. The first line of input contains a single integer t ($1 \leq t \leq 1000$), the number of scenarios. Then follow the t scenarios. Each scenario consists of a single line, containing six integers a_1, b_1, a_2, b_2, a_3 and b_3 ($1 \leq a_1, b_1, a_2, b_2, a_3, b_3 \leq 10^9$).

Output

For each test scenario, output the minimum area of land such that Maarja can construct the three buildings.

Sample Input 1	Sample Output 1

2
2 3 2 2 1 1
2 4 5 1 2 3
	12
	21

This page is intentionally left blank.

Problem C
Cocoa Coalition
Time limit: 1 second

Alice and Bob decide to share a chocolate bar, which is an n by m rectangular grid of chocolate cells. They decide that Alice should get $a < n \cdot m$ pieces and that Bob should get $b = n \cdot m - a$ pieces. To split the chocolate bar, they repeatedly take a single piece of chocolate and break it either horizontally or vertically, creating two smaller pieces of chocolate. See Figure C.1 for an example.

What is the minimum number of splits that Alice and Bob need to perform in order to split the n-by-m chocolate bar into two piles consisting of a and b chocolate cells?

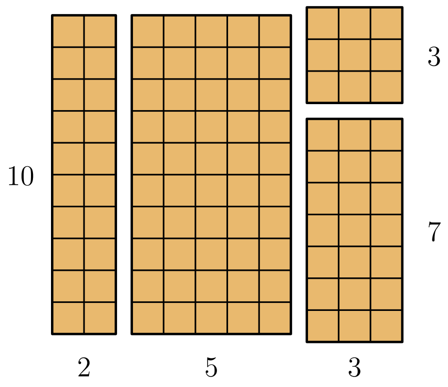


Figure C.1: Illustration of a solution to Sample Input 2, showing the original 10-by-10 chocolate bar split three times into pieces of size 10-by-2, 10-by-5, 3-by-3 and 7-by-3. Giving Alice the 10-by-5 and 7-by-3 pieces, she gets a total of $50 + 21 = 71$ chocolate cells.

Input

The input consists of a single line, containing the three integers n, m and a ($1 \leq n, m \leq 10^6$, $1 \leq a < n \cdot m$).

Output

Output the minimum number of splits needed to achieve the desired division of chocolate.

Sample Input 1	Sample Output 1
`3 10 9`	`1`

Sample Input 2	Sample Output 2
`10 10 71`	`3`

This page is intentionally left blank.

Problem D
Dungeon Dawdler
Time limit: 3 seconds

Oh no! The wicked wizard Nocoproco has captured you and locked you into his dungeon, from which there is no escape. Looks like you will be here for a while, so you might as well make good use of the time and scout out the place. Fortunately, there are no monsters lurking about in the dungeon, so exploring should be relatively safe. But there are still some complications.

There may be up to two hidden trapdoors in the dungeon. Walking into a trapdoor causes you to fall through it and end up in a different part of the dungeon. To make matters worse, there are few distinguishing features in the dungeon so you have no immediate way of recognizing previously visited locations, and it is easy to get disoriented. However, you do have a keen sense of direction and can always tell which way is north.

The dungeon is a rectangular grid where each cell is either a wall, a plain open space, or a trapdoor. From an open space, the only thing you can discern in the dim light is which of the four adjacent cells (in the directions north, east, south, and west) are walls, and you can then walk to any adjacent cell which is not a wall. If you walk into a trapdoor, you fall into it before you can react, but you have time to observe the four cells adjacent to the trapdoor before falling.

It is possible to go from every location in the dungeon to any other location (but it may require using a trapdoor, as in Sample Interaction 1 below). The dungeon also consists of a single area – if the trapdoors were changed into plain open spaces it would still be possible to go from every location to any other location. The endpoints of the trapdoors may be at any open space in the dungeon, but not on another trapdoor, and the endpoints of the two trapdoors are not at the same position. Your starting point is neither a trapdoor nor an endpoint of a trapdoor.

Write a program to explore the dungeon and create a complete map of it.

Interaction

This is an interactive problem, proceeding in rounds. In each round of interaction, your program first reads information about your current surroundings, and then responds by taking an action.

The information about your current surroundings is given on a single line. The line starts with four characters c_N, c_E, c_S and c_W describing the surroundings of the cell you just walked to (or for the first round, the starting cell), in the directions north, east, south and west respectively. Each character is either '`#`', indicating that the cell in that direction is a wall, or '`.`', indicating that you can walk there. Then on the same line follows either the word "`trap`" if the cell you walked to is a trapdoor, or "`ok`" if it is not. If the cell is a trapdoor, the line contains a third string, describing the surroundings of the endpoint of the trapdoor in the same format as above.

There are two types of actions: moving to an adjacent cell, and reporting that you are done. To move to an adjacent cell, output a single line containing one of '`N`', '`E`', '`S`', and '`W`', corresponding to moving north, east, south, and west respectively. To report that you are done, output a line containing "`done`", followed by a complete map of the dungeon. Output the map as a rectangular grid of minimum size, including all walls seen. First output a line with two integers h and w indicating the height and width of the grid. Then output h lines (from north to south) each containing exactly w characters (from west to east), using the following symbols:

1. '`#`' for walls, and for other unreachable cells that are outside the dungeon.
2. '`S`' for your starting position.
3. '`A`' and '`B`' for the locations of the two trapdoors.
4. '`a`' and '`b`' for the corresponding locations of the endpoints of the two trapdoors.
5. '`.`' for the remaining walkable cells.

It does not matter which of the trapdoors you label 'A' and which of them you label 'B', as long as 'a' is the endpoint of the trapdoor labelled 'A', and similarly for 'b'. If there is only one trapdoor, you may label it either 'A' or 'B'. After outputting the map the interaction ends and your program should exit.

There are at most 500 reachable positions in the dungeon. Your solution may take at most 10^5 steps. If you attempt to walk into a wall, your program will be judged as Wrong Answer. *To facilitate testing of your solutions, we provide a simple tool that you can download from the Kattis page for this problem.* Remember to flush your standard output buffer after each action!

Read **Sample Interaction 1** **Write**

```
#.#. ok
```

```
E
```

```
#.#. trap .###
```

```
N
```

```
##.. trap #.##
```

```
E
```

```
#.#. ok
```

```
E
```

```
#.#. ok
```

```
E
```

```
#.#. trap .###
```

```
done
4 7
#######
#b.SAB#
#####a#
#######
```

Read **Sample Interaction 2** **Write**

```
.##. ok
```

```
W
```

```
..## ok
```

```
N
```

```
#..# ok
```

```
E
```

```
##.. ok
```

```
done
4 4
####
#..#
#.S#
####
```

Problem E
Eeny Meeny
Time limit: 2 seconds

Picture by Gina Harbach Glenn Green on Flickr, cc by-nd

"Eeny meeny miny moe" is a well-known nursery rhyme in English, used (among other things) by kids to "randomly" select members of a team. It exists in many variations, one of which goes like this:

> Eeny, meeny, miny, moe,
> Catch a tiger by the toe.
> If he hollers, let him go,
> Eeny, meeny, miny, moe.

Similar verses exist in most languages, such as "Ulle dulle dof" in Finnish, "Akka bakka bonka rakka" in Norwegian, and "Ole dole doff" in Swedish.

Two teams are to be selected for a game and the rhyme is used to select one kid for a team at a time, alternating between the two teams, until all kids have been selected. The kids are standing in a circle. In each selection round we start counting the kids in clockwise order around the circle, skipping one kid for every word in the rhyme, until the last word. The kid matching the last word is chosen for the current team and then the next round starts. In all rounds but the first, the counting starts at the next remaining kid (in clockwise order) after the one that was selected in the previous round. See Figure E.1 for an example.

Given such a rhyme, and a group of kids, can you tell which kids will be in which team?

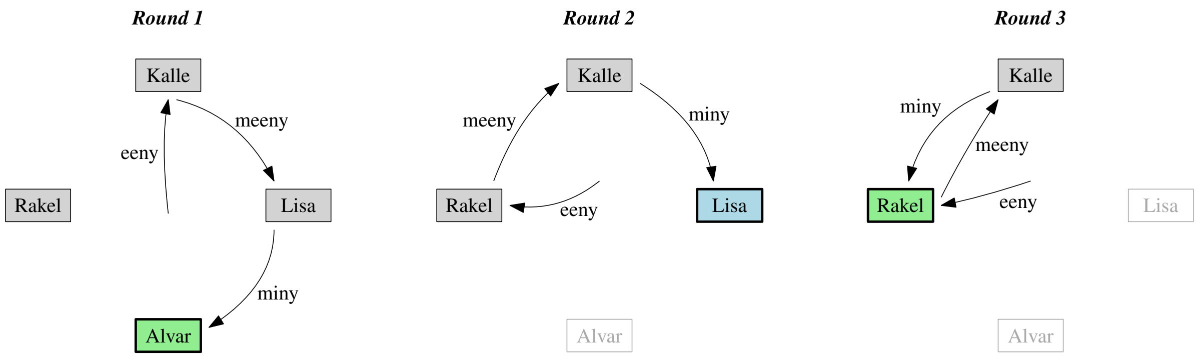


Figure E.1: Illustration of the first three rounds of Sample Input 1. In rounds 1 and 3, Alvar and Rakel get selected for the first team, and in round 2, Lisa is selected for the second team. In round 4 (not shown), only Kalle remains and is selected for the second team.

Input

The first line of input contains the rhyme, consisting of a list of words separated by spaces. The second line of input contains an integer n ($1 \leq n \leq 100$), the number of kids. Then follow the names of the kids, one per line. The kids are given in clockwise order and the first kid listed is the one at which counting starts in the first round.

All words and names consist only of upper and lower case letters '`A`'-'`Z`' and '`a`'-'`z`'. No input line is empty or longer than 100 characters (excluding the newline character at the end of the line).

Output

Output the two teams, starting with the one whose first member is chosen first. For each team, output the number of kids in the team, followed by the names of the kids in the team, in the same order as they were chosen for the team.

Sample Input 1

```
eeny meeny miny
4
Kalle
Lisa
Alvar
Rakel
```

Sample Output 1

```
2
Alvar
Rakel
2
Lisa
Kalle
```

Sample Input 2

```
Every Other
3
a
b
c
```

Sample Output 2

```
2
b
c
1
a
```

Problem F
Flow Finder
Time limit: 4 seconds

Last summer, Carla the Cartographer went on an expedition on behalf of the National Center for Positioning and Charting (the NCPC). The goal was to measure water flows of a river system far away in the north. However, the area is quite remote and Carla is not the adventurous type, so she was only able to measure the water flows in some of the locations. Carla is now worried that the NCPC will send her back to this wilderness next summer, so she consulted some algorithm experts (you) to see if it is possible to reconstruct the missing data.

The river system is represented by a rooted tree with n vertices numbered from 1 to n. The leaves of this tree are the sources, and the other vertices correspond to confluences (places where multiple rivers join together). Water flows from higher-numbered vertices to lower-numbered vertices. Vertex 1, the root of the tree, is the mouth of the river system, where it flows into the ocean. The water flow of a source can be any positive integer, while the water flow of a confluence is the sum of the water flows of its children. You will be given this tree along with the water flows at some of its vertices, and your task is to find the water flows at all the vertices or determine that this is impossible.

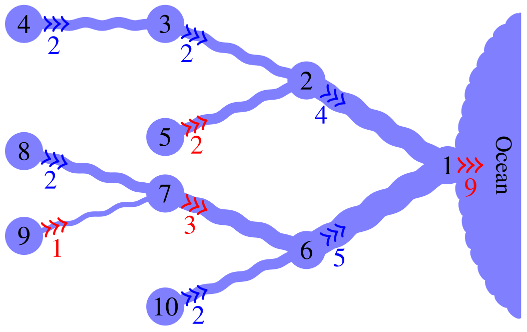


Figure F.1: Illustration of Sample Input 1 and its solution. Water flows from left to right. The flows shown in red, from vertices 1, 5, 7 and 9, are not given but can be deduced to be 9, 2, 3 and 1 respectively.

Input

The first line of input contains an integer n ($2 \leq n \leq 3 \cdot 10^5$), the number of vertices in the tree. Then follows a line containing $n - 1$ integers $p_2, \ldots, p_n$ ($1 \leq p_i < i$), where p_i is the number of the parent of vertex i.

Finally there is a line containing n integers $a_1, \ldots, a_n$ ($0 \leq a_i \leq 10^9$), where a_i represents the water flow at vertex i. If a_i is equal to 0, then the water flow at that vertex is unknown. Otherwise, a_i is equal to the water flow at vertex i. Note that the upper bound 10^9 on a_i does not apply to the unknown values, they can be any positive integer.

Output

If all the n water flows can be reconstructed uniquely, then output them in increasing order of vertex number. Otherwise, output "`impossible`". Note that it is possible that there is no way of reconstructing the water flows (if the data provided by Carla is inconsistent somehow). In that case you should also output "`impossible`".

Sample Input 1

```
10
1 2 3 2 1 6 7 7 6
0 4 2 2 0 5 0 2 0 2
```

Sample Output 1

```
9
4
2
2
2
5
3
2
1
2
```

Sample Input 2

```
5
1 2 2 1
4 0 0 0 1
```

Sample Output 2

```
impossible
```

Sample Input 3

```
4
1 1 1
3 2 1 0
```

Sample Output 3

```
impossible
```

Problem G
Game of Gnomes
Time limit: 1 second

Picture by Danielle Walquist Lynch via flickr, cc by

The enemy and their massive army is approaching your fortress, and all you have to defend it is a legion of guardian gnomes. There is no hope of winning the battle, so your focus will instead be on causing as much damage as possible to the enemy.

You have n gnomes at your disposal. Before the battle, they must be divided into at most m non-empty groups. The battle will then proceed in turns. Each turn, your gnomes will attack the enemy, causing one unit of damage for each living gnome. Then the enemy will attack by throwing a lightning bolt at one of the m groups. The lightning bolt kills k of the gnomes in that group, or all of them if the number of living gnomes in the group is less than k. The battle ends when all gnomes are dead. The enemy will always throw the lightning bolts in an optimal way such that the total damage caused by the gnomes is minimized.

Now you wonder, what is the maximum amount of damage you can cause to the enemy if you divide the gnomes into groups in an optimal way?

For example, if as in Sample Input 1 you have $n = 10$ gnomes that are to be divided into $m = 4$ groups, and the lightning bolt does at most $k = 3$ damage, then an optimal solution would be to create one large group of size 7 and three small groups of size 1. In the first round, you cause 10 damage and the lightning bolt reduces the large group by 3. In the next round, you cause 7 damage and the large group is reduced down to size 1. In the remaining four rounds you do 4, 3, 2, and 1 damage respectively and the lightning bolt removes one group each round. In total you do $10 + 7 + 4 + 3 + 2 + 1 = 27$ damage.

Input

The input consists of a single line containing the three integers n, m, and k ($1 \leq n \leq 10^9$, $1 \leq m, k \leq 10^7$), with meanings as described above.

Output

Output the maximum amount of damage you can cause to the enemy.

Sample Input 1	Sample Output 1
`10 4 3`	`27`

Sample Input 2	Sample Output 2
`5 10 100`	`15`

Problem H
Hot Hike
Time limit: 2 seconds

Picture by dan lundmark on flickr, cc by

In order to pass time during your vacation, you decided to go on a hike to visit a scenic lake up in the mountains. Hiking to the lake will take you a full day, then you will stay there for a day to rest and enjoy the scenery, and then spend another day hiking home, for a total of three days. However, the accursed weather this summer is ridiculously warm and sunny, and since severe dehydration is not at the top of your priority list you want to schedule the three-day trip during some days where the two hiking days are the least warm. In particular you want to minimize the maximum temperature during the two hiking days.

Given the current forecast of daily maximum temperatures during your vacation, what are the best days for your trip?

Input

The first line of input contains an integer n ($3 \leq n \leq 50$), the length of your vacation in days. Then follows a line containing n integers $t_1, t_2, \ldots, t_n$ ($-20 \leq t_i \leq 40$), where t_i is the temperature forecast for the i'th day of your vacation.

Output

Output two integers d and t, where d is the best day to start your trip, and t is the resulting maximum temperature during the two hiking days. If there are many choices of d that minimize the value of t, then output the smallest such d.

Sample Input 1

```
5
23 27 31 28 30
```

Sample Output 1

```
2 28
```

Sample Input 2

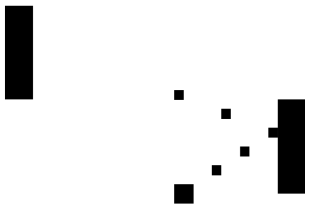
```
4
30 20 20 30
```

Sample Output 2

```
1 30
```

This page is intentionally left blank.

Problem I
Incremental Induction
Time limit: 2 seconds


Pong via Wikimedia Commons, public domain

The Nordic Collegiate Pong Championship (NCPC) is an insanely competive tournament where every contestant plays exactly one game of Pong against every other contestant. The last game of the tournament just finished, so only one item now remains on the programme: the traditional diploma ceremony, where all this year's participants get inducted into the NCPC Hall of Fame.

According to the ancient customs, contestants who have not been inducted into the Hall of Fame yet (the pathetic nobodies) must stay on the left side of the stage, whereas contestants who have been inducted (the awesome legends) must be on the right side of the stage. Then, when a contestant is receiving their diploma, they will symbolically walk from the left to the right side of the stage and thus become an awesome legend. Only one contestant is inducted into the Hall of Fame at a time, and every contestant starts on the left side initially.

The NCPC Head of Jury believes it reflects badly on her if too many of the awesome legends on the right have lost matches against pathetic nobodies on the left, but she quickly realizes that it might be impossible to avoid this at every point in time during the diploma ceremony. However, she certainly wants to keep such atrocities at a minimum. Specifically, she wants to find the smallest number k for which there exists an order of handing out diplomas to the contestants, such that at no point there were more than k games played where an awesome legend lost against a pathetic nobody.

Input

The first line of input contains a single integer n ($1 \leq n \leq 5\,000$), the number of contestants. Then follows $n-1$ lines, the i^{th} of which contains a binary string of length i. The j^{th} character on the i^{th} line is 1 if contestant $i+1$ defeated contestant j, and 0 if contestant j defeated contestant $i+1$.

Output

Output a single integer k, the smallest number according to the requirements above.

Sample Input 1

```
4
1
01
100
```

Sample Output 1

```
1
```

Sample Input 2

```
5
0
00
100
1100
```

Sample Output 2

```
3
```

Problem J
Jealous Youngsters
Time limit: 3 seconds

Picture by Bryon Lippincott on Flickr, cc by

It is ~~chaos~~ playtime at kindergarten again, and teacher Tom is having some difficulties. You see, kids often have disagreements about who should play with which toy. Whenever a kid perceives any form of injustice in the allocation of toys to kids, they start crying uncontrollably. The kids are clearly poor decision makers, so today Tom has a new plan: rather than letting the kids choose toys by themselves, he will assign toys to them in a way that prevents any crying.

Tom has been studying the behavior of the kids and now understands exactly under what circumstances they start crying. First, a child will cry if they have no toy to play with. Second, even having a toy, a child A will cry if there is some other child B playing with some toy T such that A envies B for playing with T, and A would rather play with T than with A's current toy. Additionally, Tom has observed the following four behavioural properties of children:

(a) **Envious:** children envy those who have played more than them with any given toy. If child A played strictly more with toy T yesterday than child B did, then B will be envious of A for playing with T today.

(b) **Inflexible:** a child would rather play with some toy it played with yesterday, and the earlier they first played with it yesterday, the more they want to play with it today. All toys that a child did not play with at all yesterday are equally (un)desirable to the child.

(c) **Cannot multitask:** a child never plays with more than one toy at a time.

(d) **Uncooperative:** children are bad at sharing, and two children never play with the same toy at the same time.

Tom has recorded which toys were played with by which kid yesterday, taking note of when each kid started playing with a toy. Using this information, Tom aims to make one fixed assignment of toys for all of today that, if possible, prevents any crying.

Input

The first line of input contains two integers n and m ($1 \leq n, m \leq 1\,000$), the number of kids and the number of toys. The kids are numbered from 1 to n and the toys from 1 to m. Then follows a line containing two integers d and e ($1 \leq d \leq 10^9$ and $0 \leq e \leq 10^6$), the total duration of yesterday's playtime in microseconds and the number of events Tom recorded yesterday.

Then follow e lines giving the events Tom recorded yesterday. Each event is described by a line containing three integers s, k and t ($0 \leq s < d$, $1 \leq k \leq n$, and $0 \leq t \leq m$), indicating that kid k started playing with toy t at time s (in microseconds since the start of yesterday's playtime). If $t = 0$ then kid k stopped playing with any toy at time s.

The events are given in non-decreasing order of time (i.e., the values of s are non-decreasing). All toy changes for all kids prior to playtime ending have been recorded by Tom (in particular if one kid k_1 steals a toy from some other kid k_2, there is also an event indicating k_2's change of toy at the same microsecond, even if k_2 stops playing with any toy). At the start of the playtime, no kid is playing with any toy (but they can start playing with a toy at time 0). At time d all kids still playing with toys stopped playing with them but Tom did not record these events. Kids do not switch toys more than once per microsecond.

Output

If there is an assignment of toys to kids such that no kid will start crying today, output n distinct integers, the i^{th} of which is the toy that kid i should play with. If there are multiple possible solutions, output any one of them. Otherwise, if there is no such assignment, output "`impossible`".

Sample Input 1

```
2 3
6 7
0 1 1
0 2 2
1 1 3
2 1 2
2 2 1
3 2 3
4 2 1
```

Sample Output 1

```
1 2
```

Sample Input 2

```
2 1
20 3
0 1 1
10 1 0
10 2 1
```

Sample Output 2

```
impossible
```

Sample Input 3

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

Sample Output 3

```
2
```

Problem K
Keep it Cool
Time limit: 2 seconds

Picture by Fitness First via Vimeo, cc by

As the workload of the semester is ramping up you get the task of refilling the fridge in the lab with soda. The fridge has s slots, each with a capacity for d bottles of soda, and you have n new soda bottles to add to the fridge. The sodas currently in the fridge are all nice and cold, but the new ones are not and need to be cooled in the fridge for a while until they are ready to drink.

You can only refill the fridge from the front, so in an ideal world, you would first take out all the sodas currently in the fridge, then put in the n new ones, and then put the old and cold sodas in front of the new ones. But in an ideal world you would also not have two exams and a homework deadline coming. You are simply way too busy to do all this work.

Instead, you are going to just put the new bottles in the front of the fridge and hope for the best. However, you can still to be clever about which slots to put the new sodas in. Each time a student comes for a soda, they will take one from the front of a uniformly random non-empty slot in the fridge. You decide to add the new bottles to the fridge so as to maximize the probability that all the next m students getting a soda from the fridge will get a cold one.

Input

The first line of input contains four integers n, m, s and d ($1 \leq n, m, s, d \leq 100$), the number of new soda bottles, number of students to optimize for, number of slots in the fridge, and capacity of each slot, respectively. Then follows a line containing s integers $c_1, \ldots, c_s$ ($0 \leq c_i \leq d$ for each i), where c_i is the number of soda bottles currently in slot i of the fridge.

You may assume that there is free space for all the n new bottles in the fridge.

Output

If there is a chance that all the next m students will get a cold bottle, then output s integers describing a refill scheme for the n soda bottles that maximizes the probability of this happening. The i^{th} of these s integers indicates how many of the new bottles are placed in the front of slot i in the fridge. If there are multiple optimal refill schemes, output any one of them. Otherwise, if it is impossible for all the next m students to get a cold soda, output "`impossible`" instead.

Sample Input 1	Sample Output 1
5 3 3 4 0 1 4	2 3 0

Sample Input 2	Sample Output 2
2 7 6 4 0 1 2 2 0 1	impossible