FAU Winter Contest 2014

February 01, 2014

The Problem Set

No	Title
A	Balloons very easy
B	Cycling
C	Diff
D	Dungeon
E	Full House Of Cards easy
F	Pizza Delivery easy
G	Pizza Roll
H	Rigidity
I	Scrobble
J	Security
K	Touch Game

Good luck and have fun!

hosted by

sponsored by

Problem A
Balloons

As you know, floating balloons play a crucial role in all types of contests since the ancient Greek Olympic games. The best athletes were prized with the laurel wreathes and the appropriate amount of colorful balloons – the symbol of strength, freedom, and wisdom.
In our fascinating modern age, the programming challenges have become the most spectacular sport events ever. Following in the old tradition, the participants of the programming challenges are prized with a balloon for each solved problem.
Given the distribution of balloons over the teams, can you compute the best team as well as the easiest problem?

Input

The first line contains the integers M and N ($0 < M \leq 16; 0 < N \leq 1024$) where M is the number of problems and N the number of teams in the contest. Then M lines follow each describing a problem giving its name and color, which is encoded by a single unique capital letter (e.g., `B` for *Blue*).
Each of the following N lines gives the name and a sequence of balloon colors for one team. As this is no freshmen contest, you may safely assume that each team has solved at least one problem. Both a problem and a team name consists of up to 64 characters (upper/lower case letters, numbers and underline).

Output

Print both the name of the team with the most solved problems and the name of the problem solved by the most teams, each on a single line. If there are multiple teams or problems satisfying this condition, any of them will do.

Sample Input I

```
3 2
The_Ballons R
The_Card_Trick G
The_Pizza B
some_very_cool_name B
even_cooler_name RGB
```

Sample Output I

```
even_cooler_name
The_Pizza
```

Sample Input II

```
3 2
The_Ballons R
The_Card_Trick G
The_Pizza B
some_very_cool_name BG
even_cooler_name GB
```

Sample Output II

```
even_cooler_name
The_Card_Trick
```

This page is intentionally left (almost) blank.

Problem B
Cycling

Andreas and Michael are very ambitious amateur cyclists. They keep telling me that in a group of cyclists, alternating who is leading may save energy or even let the group cycle faster. Out of curiousity I looked up the formula behind that effect.
The speed V_{group} of a group of X persons is determined as follows: $V_{group} = W^{\sqrt{X}} \cdot V_{slowest}$, where $V_{slowest}$ is the speed of the slowest cyclist in the group and W is a given wind factor.
Given the speed of each of N cyclists as well as the wind factor W, can you calculate for me the speed of the fastest possible subgroup that can be formed?

Input

The input starts with one line containing N and W ($1 < N < 100\,000$; $1.0 \leq W \leq 1.2$) where N is the total number of cyclists and W the wind factor as described above. The second line contains N integers specifying the speeds of every single cyclist each between 1 and 10 000, inclusive.

Output

Print the speed of the fastest possible group. Your output must have an absolute or relative error of up to 10^{-5}.

Sample Input I

```
5 1
7 32 7 14 1
```

Sample Output I

```
32
```

Sample Input II

```
5 1.2
8888 10000 8334 8334 123
```

Sample Output II

```
12001
```

Sample Input III

```
3 1.023
42 42 42
```

Sample Output III

```
43.6872
```

This page is intentionally left (almost) blank.

Problem C
Diff

The lecture on compiler construction has attracted many students, and now their compiler implementations have to be graded. These implementations are based on a code skeleton, so that the students have less boring busywork to do and can focus on the interesting parts of a compiler.
As Jakob is too lazy to read through hundreds of lines of Java code, he is looking for a way to only examine modified parts of the source – i.e. he is looking for the *diff*[1] between the students' versions and the skeleton. Jakob remembers that in the Wikipedia article about *diff* there was a comment that the longest common subsequence could be used to construct the *diff* between two files. Two semesters ago he posed an exercise in the course on algorithms and data structures, where students had to determine the longest common subsequence of two strings, so he already has (several) implementations to determine the longest common subsequence. All he needs now is a way to compute the *diff*, given the old and new versions of the file and their longest common subsequence.
Your task is to compute exactly this (linewise) *diff*, given two versions of a file and a longest common subsequence of their lines.[2].

Input

Input starts with three integers o, n, and c on one line ($0 \leq c \leq o, n \leq 2\,048$) giving the number of lines for the **o**ld, **n**ew and longest **c**ommon subsequence files.
Then several lines follow giving the old version (o lines), the new version (n lines), and the longest common subsequence (c lines). The two versions of the file and the longest common subsequence are separated each by one blank line.
Every line of the old version, the new version and the longest common subsequence will consist of at least one (and up to 2048) non-blank characters (followed by a newline character), and will contain neither a space (" "), nor plus sign ("+") nor a minus sign ("-").

Output

Print the *diff* between the old and the version of the file in the following format:

- Every line of the output contains one line of the old or the new version of the file, prefixed by either a space (" "), a plus ("+") or a minus sign ("-").
- If you take the output, remove lines marked with a plus sign and remove the prefix characters (space or minus sign) from the remaining lines, the result shall be equal to the *old* version.
- If you take the output, remove lines marked with a minus sign and remove the prefix characters (space or plus sign) from the remaining lines, the result shall be equal to the *new* version.
- If you take the output, remove lines marked with a plus *or* a minus sign and remove the prefix characters (space character) from the remaining lines, the result shall be equal to a longest common subsequence of the old and the new version.
- The solution is not necessarily unique. If several valid versions of the diff exist, any of them will be accepted.

[1] A *diff* is a short comparison between two versions of a file, highlighting the common parts, and those parts unique to either version.
[2] Remember: A *subsequence* of lines of a file is a sequence of lines that you get if you delete zero or more lines from the file (*without* reordering the remaining lines). A *common* subsequence of the lines of two files is a sequence of lines that is a subsequence of both files at the same time. A *longest* common subsequence is a common subsequence that is as long as possible, i.e. every sequence of lines that is longer than that longest common subsequence can not be a common subsequence.

Sample Input

```
7 8 5
this_is_a_test
@__
this_line_gets_deleted
this_line_stays_the_same
this_line_gets_changed
@_
end_of_test

this_is_a_test
@__
this_line_stays_the_same
this_line_gets_changed,_like_so
@_
these_two_lines_are_added
@_
end_of_test

this_is_a_test
@__
this_line_stays_the_same
@_
end_of_test
```

Sample Output

```
 this_is_a_test
 @__
-this_line_gets_deleted
 this_line_stays_the_same
-this_line_gets_changed
+this_line_gets_changed,_like_so
 @_
+these_two_lines_are_added
+@_
 end_of_test
```

Problem D
Dungeon

You want to finish the design of a map for a computer game. Both the dungeon with rooms and doors and the enemies are designed. The *start* and *target room* are determined. Your job is now to place the two big bosses in separate *boss rooms*.
The constraints for such a boss room are:

- The player must be able to reach the boss room from the start room
- The player must be able to reach the target room from the boss room
- The player must be able to decide which of the bosses he fights first and be able to fight the other boss afterwards.

The player might sneak around one boss, when he is in the room of the boss and visit this room later to fight him.
It is already determined how awesome each of the bosses would be in one of the rooms. You have to pick the boss rooms in such a way, that the sum of awesomeness is maximized in order to make your game most awesome.
Some doors can only be passed in one direction and some can be passed in both. Two rooms are connected by at most one door. The layout of the level may be impossible to draw (at least in two dimensions), because rooms may overlap. That is no problem in the computer game, because at most two rooms will be shown at once (when the door between these rooms is open).

Input

The input starts with the number of test cases on the first line (at most 10). On the first line of each test case the number R of rooms and the number D of doors is given ($2 \leq R \leq 50\,000$; $0 \leq D \leq 200\,000$). The second and third line contain R integers each. The ith number on the second line is the awesomeness gained, if you select the ith room as boss room for the first boss. The ith number on the third line gives the awesomeness for the second boss, if placed in the ith room, accordingly. The awesomeness is an integer between 0 and 9 000. D lines follows, each describing a door by two integers separated by the door type. The integers give the (zero based) indices of the two connected rooms. The door type is '`-`' for one-way and '`=`' for two-way doors. One way doors allow access from the first to the second room. The room with id 0 is the *start room* and the room with id 1 is the *target room*.

Output

Print one line of output, containing the maximal sum of awesomeness that can be reached for the given dungeon. If a assignment according to the above rules is not possible output "`Impossible`".

Sample Input

```
2
2 1
20 10
30 25
0 = 1
4 3
1 2 3 4
1 2 3 4
0 = 2
1 - 0
1 = 3
```

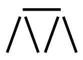
Sample Output

```
45
Impossible
```

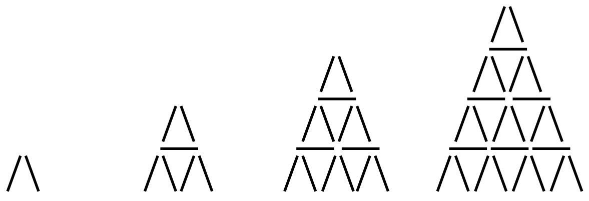
This page is intentionally left (almost) blank.

Problem E
Full House Of Cards

Like each time after the ICPC, you and all the other contestants meet at a bar drinking beer and sometimes playing cards. During a long tournament, they try to find the *Awful Card Master* (ACM). The only thing that changes each time is the card game to be played (e.g. Black Jack or Poker). But no matter which game is picked, the result always depends on randomness. Everyone in the bar agrees that an important title like the *Awful Card Master* should not depend on luck! So they determine another tactic: the target is to build a full house of cards.

1. You can lean two cards against each other in some up-side-down V shape:
2. You can connect two of those V shapes with a card, laid horizontally on top: 
3. Each horizontal card from rule 2 must be covered with another V shape from rule 1.

The contestant with the tallest house wins. For example, Alice builds a house of height 1 (a), Bob of height 2 (b) and so on (c) and so on (d).


(a) 1 floor (b) 2 floors (c) 3 floors (d) 4 floors

To win the contest, you want to build the tallest house. Given h, how many cards do you need to build a house with h floors?

Input

The only integer on the first line denotes the number of test cases $1 \leq T \leq 1\,000$. Each of the following T lines describes one test case. The input per test case is a single integer $1 \leq h \leq 10^9$, describing the height the house should have.

Output

Print one line per test case, containing the number of cards that are needed to build a house of the specified height. You may assume the result fits in the range of a 64 bit signed integer.

Sample Input

```
10
1
2
3
4
10
423
900000000
800000000
700000000
987654321
```

Sample Output

```
2
7
15
26
155
268605
1215000000450000000
960000000400000000
735000000350000000
1463191587178783722
```

This page is intentionally left (almost) blank.

Problem F
Pizza Delivery

It is a very old tradition to serve pizza buns at the local programming contests at the FAU. It is not only important for the survival of the contestants, but also a good way to attract freshman students.

A common recipe[3] (for three persons) is:

Ingredients

amount		item
metric	imperial	
200 g	1/2 lb	diced ham
200 g	1/2 lb	diced salami
200 g	1/2 lb	sliced mushrooms
200 g	1/2 lb	chopped sweet pepper
200 g	1/2 lb	grated mozzarella cheese
200 ml	1 cup	cream
		oregano
		garlic powder
9		buns

Figure 1 – Why not have some tasty pizza buns now?

Directions

1. Mix all ingredients together and let it steep for about half an hour. Meanwhile slice the buns in half.
2. Apply the mixture onto the halfs and place the buns on a cookie sheet. Heat the oven to 350 degrees F (175° C). Bake the buns for about 15 minutes.
3. Enjoy your meal!

But this challenge is not about making pizza buns, but rather delivering them to your team.
At the moment, you have a bit of spare time (otherwise you would not have read the recipe) and you decide to fetch some pizza buns for your team.
After arriving at the buffet, you see several plates with different diameters and you want to know how many of the buns will fit onto each of them – without overlapping themselves. For beauty reasons you want every bun to be in contact with the rim of the plate (but without overhanging). And, of course, you do not want to stack them, otherwise the sticky toppings will glue them together.
As the the pizza bun bakery will change in future contests and, hence, the new buns may have a different dimension, you naturally want to find a generic algorithm to solve the problem at hand.
Don't forget: neither you nor one of your teammates will eat more than three pizza buns at once (and you do not want to let them cool down), so you will never take more than nine buns.

You can assume that both the plates and the buns have a perfect round shape.

Input

The input starts with the number of test cases t ($t < 100$). Each of them consists of two integers b and p (with $0 < b, p \leq 500$), the diameter in millimeters of bun and plate.

Output

For each test case print one line containing the maximum number of buns you can put onto the plate. Remember: you will never take more than nine buns.

Sample Input

```
2
80 180
70 270
```

Sample Output

```
3
8
```

[3]This is **not** the top secret *Chair of Computer Science 2*-recipe – it (and its image) is taken from `http://www.chefkoch.de/rezepte/432351134416557/Superschnelle-Pizzabroetchen.html`

This page is intentionally left (almost) blank.

Problem G
Pizza Roll

The pizza roll industry is booming and your ICPC-hosts want to earn their fair share of the world market by creating a delicious side business. The pizza rolls have three main ingredients: dough (D), salami (S), and a secret mighty topping (MT). For starters, the founders selected six different types of pizza rolls with the following required amounts of ingredients:

1. Vegetarian: 100 D, 0 S, 150 MT
2. Meaty: 150 D, 100 S, 50 MT
3. Super meaty: 100 D, 200 S, 50 MT
4. Fake light: 50 D, 100 S, 150 MT
5. Mighty: 50 D, 0 S, 200 MT
6. Heart-attack-inducing-fat: 200 D, 200 S, 200 MT

Due to the experiences of numerous programming contests (aka market research), the market price for the different types are well-known and the demand is considered to be unbounded. Hence, they simply have to come up with a plan to turn their available resources into as much profit as possible. As they can not leave their computer-scientist-selves behind, they hired you to find a proper algorithm that takes the available ingredients and the achievable prices for each type as input and computes the maximum achievable total income under the given constraints.

Input

There is only one test case given in 2 lines. The first line of a test contains three numbers representing the total available resources in the following order: dough, salami, and mighty topping (each value $\leq 30\,000$). The second line contains six numbers specifying the price for each pizza roll type in the order introduced above: vegetarian, meaty, super meaty, fake light, mighty, heart-attack-inducing-fat (each price $\leq 10\,000$). All values, resources and prices, are integer.

Output

One line per test case containing the maximum achievable total income based on the available ingredients and the given prices in the test case.

Sample Input I

```
150 100 300
200 50 100 300 1 1000
```

Sample Output I

```
500
```

Sample Input II

```
550 500 300
1 10 100 1000 5000 10000
```

Sample Output II

```
10110
```

Sample Input III

```
30000 30000 30000
1 1 1 1 1 1
```

Sample Output III

```
300
```

This page is intentionally left (almost) blank.

Problem H
Rigidity

The department of Computer Science has been growing for years and years in Erlangen. To handle the masses of students and employees, the university decides to build a new tower. But they need your help to construct a robust building.
The skeleton will be built of balks with a hole at each end. Two or more balks may be connected by an axis, which fixes the relative positions of the respective end. The balks still can rotate around that axis and move freely, basically.

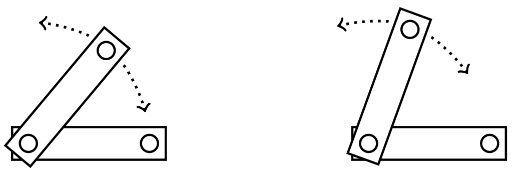


Figure 2 – The balks can rotate freely around their connecting axis.

The architect of the university used this pattern to design a whole tower as a graph. His plan only consists of the connection axis and which axes are connected by a balk. So he keeps the balk lengths as a secret. The architect wants you to determine, if a given graph of balks defines a rigid building in principle (i.e. if one can not rotate or move any balks or axes relative to the others). He further restricts the definition of rigidity to graphs that are constructed as follows. The following graphs are considered rigid:

1. A single balk.
2. A rigid graph to which a new axis is added, together with two new balks that connect the new axis with two previously existing axis.
3. A rigid graph of which one balk is subdivided by a new axis at the point of division, and a new balk added, connecting the new axis to a previously exisiting axis.

To make the building even more robust, the architect is allowed to add an arbitrary number of additional balks (but no axis) to the graph afterwards.

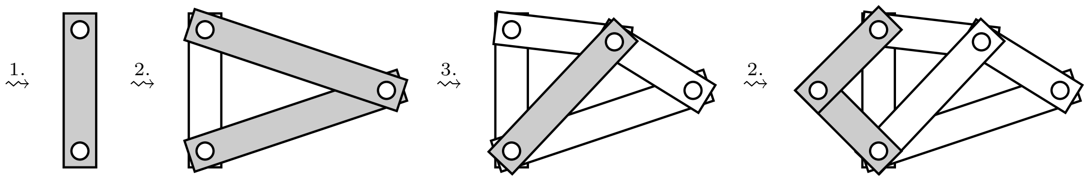


Figure 3 – Example construction of a rigid graph. Note that the architect may add additonal balks afterwards.

There might be many ways to draw the given graph, but the architect does not care about that. You may assume that, for ridigity concerns, it does not matter how the tower actually looks or how long the balks are. Assume that the balks are of appropriate length such that it will wiggle if the plan allows it in principle.

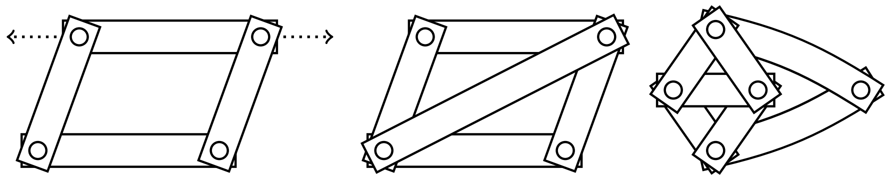


Figure 4 – Three example graphs, the left one is movable, while the others are rigid.

Input

The first line of the input contains the number of test cases on a line (< 10). Each test case starts with a line containing two integers, the number of axes $1 \leq A \leq 50$ and the number of balks $1 \leq B \leq 100$. Then B lines follow, each containing two integers c d ($0 \leq c, d < A$, $c \neq d$). It describes that there is a balk spanning from the cth to the dth axis. You may assume that each axis is used by at least one balk, the graph is connected and planar.

Output

For each test case, print a line with either `Movable` or `Rigid` answering the problem described above.

Sample Input I

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

Sample Output I

```
Movable
Rigid
```

Sample Input II

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

Sample Output II

```
Rigid
```

Problem I
Scrobble

You are playing a word game called *Scrobble* with your friends. The game consists of a number of 6-sided dice with one letter on each side. The letters on the dice do not have to be distinct, so there can be a die with 6 `a`'s.
You can arrange dice next to each other on the table and then a word is formed by letters on the top side of the dice from left to right. Note that rotating `'p'` upside down would read like `'d'`, but this is not allowed since all letters on the dice have clearly marked orientation and cannot be rotated.
Each player is given N dice and tries to build the longest English word that can be found in the huge Oxford English dictionary your friends have. You thought of an English word w but you are not sure if this word can be built using the N dice you were assigned, so you decided to write a program that will help you with the game.

Input

The first line contains a word w of length at least 1 and at most 200. The next line contains an integer $N, 1 \leq N \leq 200$. The next N lines each contain a six-letter word describing the letters on one of your dice.
All letters in the input will be lower-case and from `'a'` to `'z'`.

Output

If it is possible to build word w from the N dice you were given, print `Possible`. Otherwise print `Impossible`.

Sample Input I

```
dog
4
abcdef
qwerty
hjklop
ground
```

Sample Output I

```
Possible
```

Sample Input II

```
photographer
12
abcdef
qwerty
hjklop
ground
rrrrrr
oooooo
aaaaaa
pppppp
hhhhhh
oooooo
tttttt
eeeeee
```

Sample Output II

```
Impossible
```

This page is intentionally left (almost) blank.

Problem J
Security

The Brazilian police has decided to take new security measures in order to be prepared for the FIFA World Cup scheduled for June, this year. The police has a map of Rio de Janeiro, depicted as a graph with N crossings and M roads. Each road has unit length and connects two different crossings.
K of these crossings host many illegal activities throughout the year and are thus considered being of high risk. The police department wants to place officers in every high risk crossing, to be able to deal with unexpected events. Unfortunately, due to lack of funds, there are only $K - 1$ officers available for the job.
In order to still cover every high risk crossing, one single officer has the responsibility of watching over two interest points. For a higher degree of safety and in order not to overwhelm that one officer, the police chief wants the two high risk crossings to be as close as possible to each other. However, finding the minimal distance between any two high risk crossings is a hard problem for the police staff, since it requires knowledge about graph theory and mathematics. They need your help!

Input

The first line of input contains three integers: N, M and K, where N represents the number of crossings, M the number of roads and K the crossings of high risk ($1 < K \leq N \leq 1\,000\,000$; $0 < M \leq 1\,000\,000$). The second line contains K different numbers, the 0-based indices of the high risk crossings. Then M lines follow, each holding two integers a_i and b_i, which define a road by its vertex indices ($0 \leq a_i, b_i < N, a_i \neq b_i$).

Output

The output contains one integer, representing the minimal distance between any two high risk crossings. If there is no connecting path between any two crossings, then output “`Impossible`”.

Sample Input I

```
8 7 3
0 4 7
0 1
1 2
0 2
2 3
3 4
2 5
5 7
```

Sample Output I

```
3
```

Sample Input II

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

Sample Output II

```
Impossible
```

This page is intentionally left (almost) blank.

Problem K
Touch Game

Recently, I've developed a simple single-player touch screen game: the player gets two grids of equal size, where some grid cells contain a ball. The goal of the game is to make the two grids identical by modifying the grid on the left side. By clicking on cell, you *invert* a cell – if the cell is empty, a ball will be added, removed otherwise. Not only the touched cell is inverted but also cells that lie on the same diagonals and have distance less or equal 2 to the touched cell.

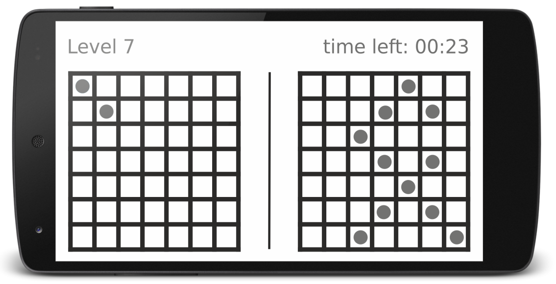


Figure 5 – This is my current level 7 (see sample II). You can finish the level in three steps: by clicking on coordinates $(0,0)$ (upper left corner), on $(0,5)$, and on $(5,5)$.

I did create many levels, but I am a bit unsure about their difficulty. I could ask many people to try those levels and rate their difficulties but I decided to go for a different approach: given the original and the target grid the difficulty of a level is given by the minimal number of touches.

Input

The input consists of exactly one level, starting with two integers h and w on one line, where h specifies the height and w the width of the grids ($1 \leq h \leq 20; 1 \leq w \leq 7$). Then follow h lines specifying the two grids (original on the left, target on the right). '`-`' marks an empty cell, while '`*`' is a cell with a ball.

Output

If the level is solvable, print the minimal number of touches, otherwise '`Impossible`'.

Sample Input I

```
5 7
*---*-- -------
-*-*--- -------
--*---- -------
-*-*--- -------
*---*-- -------
```

Sample Output I

```
1
```

Sample Input II

```
7 7
*------ ----*--
-*----- ---*-*-
------- --*----
------- ---*-*-
------- ----*--
------- ---*-*-
------- --*---*
```

Sample Output II

```
3
```

Sample Input III

```
3 3
*-- *--
--- ---
--* ---
```

Sample Output III

```
Impossible
```

This page is intentionally left (almost) blank.