GCPC 2012

30.06.2012

acm International Collegiate Programming Contest

The Problem Set

Good luck and have fun!

hosted by

FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

sponsored by

accenture High performance. Delivered.

Problem A
Battleship

When playing battleship, players take turns trying to sink the other players navy. Each player may take shots at one coordinate at once. If he hits one of his enemy's ships and the enemy has any other ships left, he may continue. Else, the other player may take shots. After hitting a ship at one coordinate, shooting at that coordinate again counts as a miss.
The game is finished when every part of every ship of a navy of one player has been hit. The first player starts, and every player gets the same number of turns. That means, that the second player might get another turn even if all his ships have been sunk.
The game ends with a draw if both navies are completely sunk, or if there are still ships left after all shots have been fired.

Tom the little spy watches a game of battleships between two fleet admirals. As he has successfully tapped the communication wires, he can intercept the shot orders. However, he was unable to determine which admiral ordered what shot to be fired.
After the game, he successfully breaks into the super secret game management facility, and manages to secure the fleet deployment maps. As he wants to determine which fleet admiral is more dangerous, he wants to know which admiral won.
He transmits the deployment maps and the shot orders, and wants you to determine which admiral won.

Input

Input starts with one line, containing the number of test cases t ($0 < t \leq 20$).
Every test case starts with a line, containing three integers w, h and n ($1 \leq w, h \leq 30; 1 \leq n \leq 2000$), describing the width and height of the fleet deployment maps, and the number of shots.
The next h lines contain the deployment map for player one. Each line contains w field descriptions, where '_' means "water" and '`#`' means "ship". Then follow h lines containing the deployment map for player two.
The following n lines contain the shot orders; each order consists of two integers, the x and y coordinate of the shot. The x coordinate indicates the column of the shot, running from 0 to $w-1$, with 0 meaning the leftmost column. The y coordinate indicates the line of the shot, running from 0 to $h-1$, with 0 meaning the last line, and $h-1$ meaning the first line of the respective map.
Please note that there may be more shot orders than needed to end the game.

Output

For every test case, print one of "`player one wins`", "`player two wins`" or "`draw`", on a line.

Sample Input

```
2
4 4 5
----
----
#___
----
#___
___#
----
----
0 0
1 1
0 3
0 2
0 1
2 2 5
_#
#_
#_
#_
1 0
0 0
0 0
1 1
0 1
```

Sample Output

```
player two wins
draw
```

Problem B
BrainFuckVM

Given a brainfuck program, determine whether it terminates or enters an endless loop.

A brainfuck interpreter has a data array (consisting of unsigned 8-bit integers) with an index, the so called "*data index*"; the entry of the array pointed to by the data index is the so called "*current entry*". A brainfuck program consists of a sequence of the following eight instructions:

`-`	decrease *current entry* by 1 (modulo 2^8)
`+`	increase *current entry* by 1 (modulo 2^8)
`<`	decrease *data index* by 1
`>`	increase *data index* by 1
`[`	jump behind the matching `]`, if the *current entry* is equal to 0
`]`	jump behind the matching `[` if the *current entry* is *not* equal to 0
`.`	print the *current entry* as character
`,`	read a character and save it into the *current entry.* On end of input save 255.

Interpretation of a brainfuck program starts at the first instruction, and terminates if the instruction pointer leaves the end of the program. After an instruction is executed, the instruction pointer advances to the instruction to the right (except, of course, if the loop instructions `[` or `]` cause a jump).

In addition to the program, you will be given the size of the data array. The entries of the data array shall be unsigned 8-bit integers, with usual over- or underflow behaviour. At the start of the program, the data array entries and the data index shall be initialized to zero.

Incrementing or decrementing the data index beyond the boundaries of the data array shall make it re-enter the data array at the other boundary; e.g. decrementing the data index when it is zero shall set it to the size of the data array minus one.

The `[` and `]` instructions define loops and are allowed to nest. Every given program will be well-formed, i.e. when traversing the program from left to right, the number of `[` instructions minus the number of `]` instructions will always be greater or equal zero, and at the end of the program it will be equal to zero.

For the purposes of the problem, discard the output of the interpreted program. [1]

Input

The input starts with a line containing the number of test cases t $(0 < t \leq 20)$. Each test case consists of three lines. The first line contains three integers s_m, s_c, s_i, describing the size of the memory, the size of the program code and the size of the input $(0 < s_m \leq 100\,000; 0 < s_c, s_i < 4\,096)$.
The second line contains the brainfuck program code to be executed; it consists of s_c characters.
The third line contains the input of the program, as text (only printable, non-whitespace characters). Once the program has consumed all available input, the input instruction shall set the current cell to 255.

Output

For each test case, print one line, containing either "`Terminates`" or "`Loops`", depending on whether the program either terminates after a finite number of steps, or enters an endless loop. If the program loops, also print the indices (0-based) of the two `[` and the `]` symbols in the code array that correspond to the endless loop. You may safely assume that after 50 000 000 instructions, a program either terminated or hangs in an endless loop, which then was executed at least once. During each iteration of the endless loop at most 50 000 000 instructions are executed.

[1] For purposes of debugging, you may examine it. The third program from the sample input would print "ICPC"; the fourth program, when given an arbitrary string, will print a brainfuck program, that in turn will print said arbitrary string when executed.

Sample Input

```
4
10 4 3
+-.,
qwe
1000 5 1
+[+-]
a
100 74 4
+++++[->++<]>[-<++++++++>]<[->+>+>+>+<<<<]>+++>--->++++++++++>---<<<.>.>.>.
xxyz
9999 52 14
+++++[>+++++++++<-],+[-[>--.++>+<<-]>+.->[<.>-]<<,+]
this_is_a_test
```

Sample Output

```
Terminates
Loops 1 4
Terminates
Terminates
```

Problem C
Candy Distribution

Kids like candies, so much that they start beating each other if the candies are not fairly distributed. So on your next party, you better start thinking before you buy the candies.
If there are K kids, we of course need $K \cdot X$ candies for a fair distribution, where X is a positive natural number. But we learned that always at least one kid looses one candy, so better be prepared with **exactly** one spare candy, resulting in $(K \cdot X) + 1$ candies.
Usually, the candies are packed into bags with a fixed number of candies C. We will buy some of these bags so that the above constraints are fulfilled.

Input

The first line gives the number of test cases t $(0 < t < 100)$. Each test case is specified by two integers K and C on a single line, where K is the number of kids and C the number of candies in one bag $(1 \leq K, C \leq 10^9)$. As you money is limited, you will never buy more than 10^9 candy bags.

Output

For each test case, print one line. If there is no such number of candy bugs to fulfill the above constraints, print "`IMPOSSIBLE`" instead. Otherwise print the number of candy bags, you want to buy. If there is more than one solution, any will do.

Sample Input

```
5
10 5
10 7
1337 23
123454321 42
999999937 142857133
```

Sample Output

```
IMPOSSIBLE
3
872
14696943
166666655
```

This page is intentionally left (almost) blank.

Problem D
Outsourcing

Mr. Cooper is a manufacturer of science fiction action figures and he thinks that his local factory causes too many costs. He once heard of certain foreign countries where workers are much less expensive and also more dedicated. So he decided to look for an available action figure factory in some low-wage country to follow the current trend of Outsourcing.

It is of course indispensable for him to know if the new factory is capable of producing exactly the same sorts of action figures as the current one. The manufacturing process is organised in terms of assembly stations and transfer stations. An assembly station receives parts from a transfer station, performs a specific operation on the parts and delivers it to a transfer station. The factories have one starting transfer station delivering raw parts and one final transfer station receiving the completed action figures.

One sort of action figures needs a precise sequence $s_1, s_2, s_3, \ldots, s_\ell$ of assembly operations. A factory can produce these action figures if there are transfer stations $t_0, t_1, \cdots, t_\ell$ such that t_0 is the starting station, t_ℓ is the final station, and for all $1 \leq i \leq \ell$ there is an assembly station that receives from t_{i-1}, delivers to t_i, and performs operation s_i.

Hence, Mr. Cooper wants to know if the local factory and the foreign factory can produce exactly the same sorts of action figures. He recognizes that answering this question may be an involved challenge. So, he decides to spend an amount of his yet to save money on you, to make you develop a computer solution for his problem.

Input

Input starts with a line containing one integer t, the number of test cases ($0 < t \leq 100$). Each test case starts with a line of six integers M_1, N_1, K_1, M_2, N_2, and K_2, where the local factory has M_1 assembly stations, N_1 transfer stations and K_1 different assembly operations ($1 \leq M_1 \leq 10^5; 1 \leq N_1 \leq 250; 1 \leq K_1 \leq 250$). The transfer stations are numbered from 0 to $N_1 - 1$ and 0 denotes the starting station and $N_1 - 1$ the final station. Then input is followed by M_1 lines specifying the assembly stations of the local factory. Every line contains three integers, T_{in}, T_{out}, S, where T_{in} is the transfer station delivering to the assembly station, T_{out} is the transfer station receiving the assembly results and S gives the performed operation by a number between 0 and $K_1 - 1$. For every transfer station it is guaranteed that there are not two receiving assembly stations performing the same operation. The foreign factory is described analogously by M_2, N_2 and K_2 and consequently the next M_2 lines of input describe the assembly stations of the foreign factory.

Output

For every test case print a line containing "`eligible`" if the local and the foreign factory are capable of manufacturing exactly the same action figures and otherwise print "`not eligible`".

Sample Input

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

Sample Output

```
eligible
not eligible
```

This page is intentionally left (almost) blank.

Problem E
Pizza Hawaii

You are travelling in a foreign country. Although you are also open to eat some regional food, you just cannot resist after you have found an Italian restaurant which offers pizza. Unfortunately the menu is written in the foreign language, so the list of ingredients of the pizzas are incomprehensible to you. What will you do?
One thing that you notice is that each pizza has an Italian name, which sounds quite familiar to you. You even remember for each named pizza what the ingredients of this pizza usually are. You want to use that information to figure out what the possible meaning of each word on the list of ingredients is.

Input

The first line of the input gives the number of test cases t ($0 < t \leq 20$). The first line of each input gives the number n of pizzas on the menu ($1 \leq n \leq 60$). The following $3 \cdot n$ lines describe the pizzas on the menu. Each pizza description starts with one line containing the name of the pizza. The name of the pizza consists of between 3 and 20 uppercase and lowercase letters. The next line starts with an integer m_i, giving the number of ingredients of the pizza on the menu ($1 \leq m_i \leq 20$). The rest of the line contains the m_i ingredients separated by spaces. Each ingredient is a word consisting of between 2 and 20 lowercase letters. The third line of each pizza description gives the ingredients in your native language in the same format. Note that the number of ingredients may differ, because each restaurant may use slightly different ingredients for pizzas with the same name, so the ingredients you remember for a pizza with that name may not match the actual ingredients.

Output

For each test case print all pairs of words (w_1, w_2) where w_1 is an ingredient in the foreign language that could be the same ingredient as w_2 because w_1 and w_2 appear on the same set of pizzas. Sort the pairs in increasing lexicographical order by w_1, and in case of a tie in increasing lexicographical order by w_2. Print a blank line between different test cases.

Sample Input

```
2
3
Hawaii
4 tomaten schinken ananas kaese
4 pineapple tomatoes ham cheese
QuattroStagioni
6 tomaten kaese salami thunfisch spinat champignons
6 mushrooms tomatoes cheese peppers ham salami
Capricciosa
6 champignons kaese tomaten artischocken oliven schinken
5 cheese tomatoes mushrooms ham artichoke
1
Funghi
3 tomaten kaese champignons
3 cheese tomatoes mushrooms
```

Sample Output

```
(ananas, pineapple)
(artischocken, artichoke)
(champignons, mushrooms)
(kaese, cheese)
(kaese, ham)
(kaese, tomatoes)
(oliven, artichoke)
(salami, peppers)
(salami, salami)
(spinat, peppers)
(spinat, salami)
(thunfisch, peppers)
(thunfisch, salami)
(tomaten, cheese)
(tomaten, ham)
(tomaten, tomatoes)

(champignons, cheese)
(champignons, mushrooms)
(champignons, tomatoes)
(kaese, cheese)
(kaese, mushrooms)
(kaese, tomatoes)
(tomaten, cheese)
(tomaten, mushrooms)
(tomaten, tomatoes)
```

This page is intentionally left (almost) blank.

Problem F
Roller coaster fun

Jimmy and his friends like to visit large theme parks. In the current theme park there are many roller coasters which then are categorized by Jimmy. He assigns a fun value to each coaster; however, the fun decreases with each run.

More formal: for a specific roller coaster i, Jimmy assigns two fun coefficients a_i and b_i. While riding this roller coaster for the k-th time, Jimmy gains a fun value of $f(i,k) = a_i - (k-1)^2 \cdot b_i$. If $f(i,k)$ is non-positive, riding the roller coaster is no longer funny.

Jimmy tries to maximize the total fun until he leaves the park. Can you tell Jimmy how much fun he can gain for a given time?

Input

The input consists of a single test case.
The first line contains the integer N, where N is the amount of different roller coasters in the theme park ($0 < N \leq 100$).
The following N lines contain the integers a_i, b_i and t_i where a_i and b_i are the fun coefficients as specified above and t_i is the time for a single ride with the i-th roller coaster ($0 \leq a_i \leq 1\,000$; $0 \leq b_i \leq 1\,000$; $0 < t_i \leq 25\,000$).
The next line contains a positive integer Q denoting the number of times that Jimmy is visiting the park ($0 \leq Q \leq 1\,000$). Each of the following Q lines contains an integral time T_i that Jimmy spends during his i-th visit ($0 \leq T_i \leq 25\,000$).

Output

For each of the Q possible times, print one line containing the maximal total fun value if Jimmy spends T_i minutes in the theme park.

Sample Input I

```
2
5 0 5
7 0 7
4
88
5
6
7
```

Sample Output I

```
88
5
5
7
```

Sample Input II

```
1
100 3 2
5
2
3
4
5
100
```

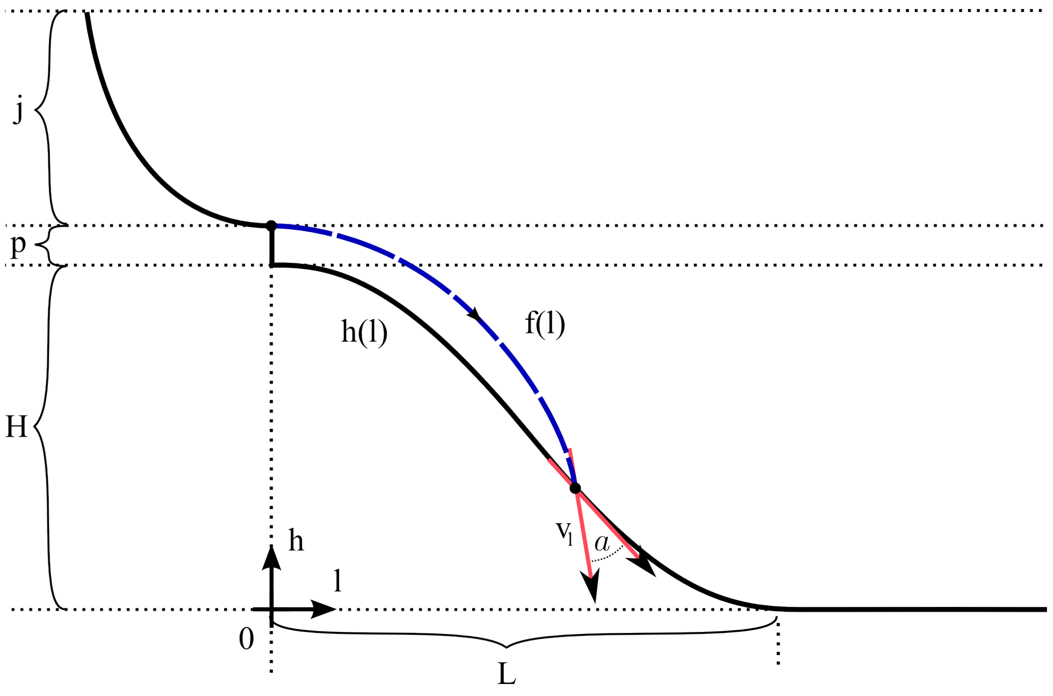
Sample Output II

```
100
100
197
197
435
```

This page is intentionally left (almost) blank.

Problem G
Ski Jumping

Ski jumping is one of the most popular winter sport competitions. In the chase of records, ski jumping hills become larger and larger. To ensure the safety of the competitors, landing speed and angle must not exceed critical margins defined by the FIS. Today, it's your task to assess these values for a newly constructed ski jumping arena shown in the figure.



Instead of doing measurements in the field, you can use a little math to solve your problem, since the hill has the following shape:

$$h(l) = \begin{cases} H & l < 0 \\ H \cdot \left(1 - 2 \cdot \left(\frac{l}{L}\right)^2\right) & 0 \le l < \frac{L}{2} \\ 2H \cdot \left(\frac{l}{L} - 1\right)^2 & \frac{L}{2} \le l < L \\ 0 & L \le l \end{cases} \tag{1}$$

where l is the position on the x-axis with its origin in the beginning of the hill. H is the height and L is the width of the hill; j is the maximum starting height of the ski-jump and p is the height difference between the end of the (ski-jump) approach and the top of the hill. Assuming that friction plays no important role and since the critical margins are defined for a flight without any influence of wind, you may utilize the following flight curve:

$$f(l) = -\frac{g}{2} \cdot \left(\frac{l}{v_0}\right)^2 + H + p \qquad (\, 0 \le l \;\wedge\; f(l) \ge h(l)\,) \tag{2}$$

where v_0 is the speed gained in the approach. You can obtain this value from the law of energy conservation. Potential and kinetic energy are defined as follows:

$$E_{\text{kin}} = \frac{1}{2} \times \text{mass} \times \text{speed}^2\,, \qquad E_{\text{pot}} = \text{mass} \times g \times \text{height}\,. \tag{3}$$

In all equations, g is the gravitational constant ($g \approx 9.81\,\mathrm{m\,s^{-2}}$).

Hints:
The inner product of two vectors $\vec{a}$ and $\vec{b}$ is defined as:

$$\vec{a} \cdot \vec{b} = |\vec{a}| \cdot |\vec{b}| \cdot \cos \measuredangle(\vec{a}, \vec{b}) \tag{4}$$

Input

Input starts with the number of test cases t on a single line ($0 < t < 160\,000$).
Every test case consists of a single line containing four positive integers j, p, H, and L as defined in the problem statement ($0 < j, p, H, L \leq 500$). The unit of all values is meter.

Output

For every test case, print one line containing

- the landing position l on the x-axis,
- the landing speed $|v_l|$ of the jumper (in meters per second), and
- the speed-angle α (in degree) with respect to the hill (see the figure).

The values must be separated by a single blank. An absolute or relative error of 10^{-4} is tolerated.

Sample Input

```
3
50 5 10 100
50 5 30 100
50 5 50 100
```

Sample Output

```
40.82482905 33.83045965 12.93315449
81.04978134 40.31656580 26.21334827
104.8808848 45.38832449 46.36470132
```

Problem H
Suffix Array Re-construction

It has been a long day at your new job. You have spent all day optimizing the most important Suffix-Array data structures your new employer, the GCPC ([G]lobal Suffix [C]ollecting and [P]rocessing [C]ollective),works with. The moment you were just about to shut down your workstation you stop and stare at the monitor. Your test run just has revealed that large portions of the important data must be corrupted. Sadly, the Company's backup servers just crashed yesterday, and now you may have destroyed the valuable Suffix-Arrays.

On inspecting the data, you find that it could hardly be worse. A lot of the suffixes are missing and even the ones remaining might be broken. You have found examples wherein parts of the letters have been replaced by random letters, and in some parts you find a single '`*`', your placeholder character you used in the software. This placeholder has replaced an arbitrarily large substring. Furthermore, you found some inconsistent strings, for which it is not clear which version of the character is the right one. Your only chance now is to hope and pray that a recovery is possible.

The data is given as a list of suffixes, together with the start-position of the suffix. You also still have a list of the length of all the data-sets the company has in stock. Can you possibly reconstruct at least the base strings? If so, one could run one of those fancy new Suffix-Array algorithms to reconstruct the full data-set again.

Input

Each test set consists of multiple test cases t ($0 < t \leq 100$). The number of test cases is given on a single line at the beginning of the input. Every test case is composed as follows. First, on a single line, the length l of the desired output string is given, together with the number of (partially broken) suffixes s ($1 \leq l \leq 10\,000$; $1 \leq s \leq 10\,000$). Then s lines follow, giving the position p of the suffix in the string and the suffix ($1 \leq p \leq l$). The suffix will contain only characters from the set of $\{a, \ldots, z, A, \ldots, Z, ., *\}$ (the '.' has no special meaning). The total sum of characters given for the suffixes will not exceed 250 000.

Output

Whenever it is possible to reconstruct the lost input data print the reconstructed sentence, else print "`IMPOSSIBLE`" on a single line. For our case, the recovery is only possible if the set of possible characters for a position in the string contains exactly one character.

Sample Input

```
2
6 6
6 a
5 aa
4 a*a
3 aaaa
2 aaaaa
1 aaaaaa
6 6
6 b
5 aa
4 a*a
3 aaaa
2 aaaaa
1 aaaaaa
```

Sample Output

```
aaaaaa
IMPOSSIBLE
```

This page is intentionally left (almost) blank.

Problem I
Touchscreen Keyboard

Nowadays, people do not use hardware keyboards but touchscreens. Usually, they touch on the wrong letters with their chunky fingers, because screen space is precious and the letters therefore too small.
Usually, a spell checker runs after typing a word and suggests other words to select the correct spelling from. Your job is to order that list so that more likely words are on top.
The typical touchscreen keyboard looks like this:

```
qwertyuiop
asdfghjkl
zxcvbnm
```

You should use the distance between the letters to type a word: the distance is the sum of the horizontal and vertical distance between the typed and proposed letter. Assume you typed a `w`, the distance to `e` is 1, while the distance to `z` is 3.
The typed word and the list of words from the spell checker all have the same length. The distance between two words is the sum of the letter distances. So the distance between `ifpv` and `icpc` is 3.

Input

The first line of the input specifies the number of test cases t ($0 < t < 20$). Each test case starts with a string and an integer l on one line. The string gives the word that was typed using the touchscreen keyboard, while l specifies the number of entries in the spell checker list ($0 < l \leq 10$). Then follow l lines, each with one word of the spell checker list. You may safely assume that all words of one test case have the same length and no word is longer than 10 000 characters (only lowercase 'a' - 'z'). Furthermore, each word appears exactly once in the spell checker list on one test case.

Output

For each test case, print the list of words sorted by their distance ascending. If two words have the same distance, sort them alphabetically. Print the distance of each word in the same line.

Sample Input

```
2
ifpv 3
iopc
icpc
gcpc
edc 5
wsx
edc
rfv
plm
qed
```

Sample Output

```
icpc 3
gcpc 7
iopc 7
edc 0
rfv 3
wsx 3
qed 4
plm 17
```

This page is intentionally left (almost) blank.

Problem J
Track Smoothing

Bob has the task to plan a racing track of a specific length. He thought it is a great idea to just form a convex polygon that has exactly the required length. Then he was told that racing cars cannot drive such sharp corners like in his plan.
He has to ensure some minimal radius for all curves in his track. Bobs loves the shape of his track, so he don't want to change it too much. His plan is to scale the track down around the balance point of the polygon that specifies his track. Then, he wants to draw the new track with a line that has a constant distance to the scaled track. The chosen distance should be minimal distance that fulfils the given minimum radius constraint. He asks you to write a program to calculate the scale factor for a given track and minimum radius so that the resulting track has the same length as the one of his original plan.
Bob made some drawings of the first test case:

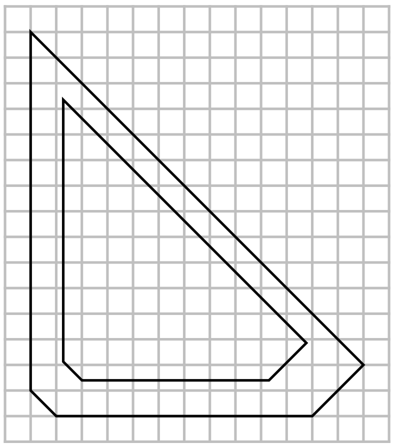


Figure 1 – Original track and scaled down track

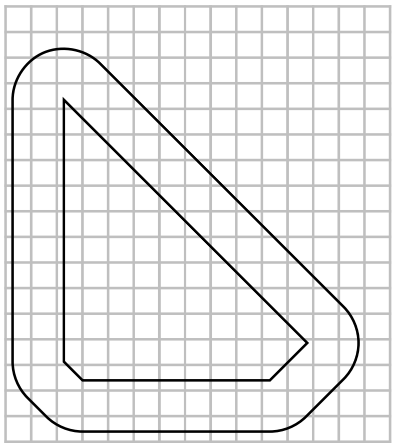


Figure 2 – Scaled down track and resulting track

Input

The input starts with the number of test cases t ($0 < t \leq 100$). Every test case starts with a line consisting of two integers: the minimal required radius r and the number of points n of the original track polygon ($0 \leq r \leq 1\,000$; $3 \leq n \leq 10\,000$). Then n lines follow, where each line specifies 2D-coordinates as two integers x_i and y_i ($-10\,000 \leq x_i, y_i \leq 10\,000$). $(0, 0)$ is the lower left corner. These are the coordinates of the original track in counterclockwise direction. You may safely assume that the area of the given polygon is non-empty.

Output

For each test case print out one line. If it is possible to construct a course according to the above description, output the scaling factor, "`Not possible`" otherwise. The factor must have a relative or absolute error smaller than 10^{-5}.

Sample Input

```
2
20 5
10 0
110 0
130 20
0 150
0 10
1 5
0 0
1 0
2 0
2 1
0 1
```

Sample Output

```
0.730494
Not possible
```

This page is intentionally left (almost) blank.

Problem K
Treasure Diving

Legends often tell of great treasures. But you rarely get the chance to actually stumble upon those treasures. Most of them are lost in the sea, or hidden below mountains. But as you learned from one of your biggest idols, treasures do belong into a museum. And now you have the chance to make that happen.

On an expedition you found a large cave network. A native shaman has spoken about incredible values his ancestors have hidden in the caves. He even gave you an ancient map, depicting the cave network and the location of the treasures within. Sadly, the cave network is completely flooded. Since the trip out here takes forever, you decided to do a short dive and scout out the cave network. But on your arrival back at the entrance to the cave network you get the news… a volcano just erupted nearby. It is next to guaranteed that the lava will cover the entries to the cave network and the treasures will be lost forever.

That puts you on the spot. You only have a short time left, and only one lousy tank of air. So it is all on you. You only have time for a single dive. But how could you possible decide which route to take? The cave network is huge, and you should definitely try and rescue as much of the treasures as possible. You think back to your times as a computer scientist at the university… And then it hits you. You still have your laptop with you. You could write a program to help you figure out the best you can do in rescuing the treasures.
You may assume that neither locating or picking up a treasure within a cave nor the traversal of a cave consumes any air.

Input

Each test set consists of multiple test cases. The file starts with a single number t ($0 < t \leq 2000$) on a single line, denoting the number of testcases in the file. Each test case starts with two integers n and m on a single line, where n the number of caves and m the number of the connecting tunnels in the network ($1 \leq n \leq 10\,000$; $0 \leq m \leq 50\,000$). This line is followed by m lines, giving a description of the tunnels of the cave as three integers a b and l with a, b denoting caves and l giving the amount of air necessary for diving through the tunnel ($0 \leq a, b < n; 0 \leq l \leq 500$). After the tunnels follows an integer i on a single line, giving the number of idols in the cave system ($0 \leq i \leq 8$). This line is followed by a single line containing i integers $p_1, \ldots, p_i$ giving the caves withing the network conaining an idol ($0 \leq p_1, \ldots, p_i < n$). The input is concluded by a single number, giving the liters of air a you have available ($0 \leq a \leq 1\,000\,000$). You will always start (and end) at the node with the label 0.

Output

For each test case print a number X on a single line, where X is replaced by the maximal number of idols the diver can recover.

Sample Input

```
3
5 3
0 1 10
0 2 20
0 3 30
4
1 2 3 4
30
5 3
0 1 10
0 2 20
0 3 30
4
1 2 3 4
60
5 3
0 1 10
0 2 20
0 3 30
4
1 2 3 4
10000
```

Sample Output

```
1
2
3
```