Rujia Liu's Present 6
Happy 30th Birthday to Myself

My 28th Birthday

2nd December, 2012
UVa Online Judge

Problems

A Special "Happy Birthday" Song!!!
Baby Me
"Center" of perimeter midpoints
Detecting Code Snippets
Egyptian Fractions (HARD version)
Finding Black Circles
Good Friends
Hadamard Gate
In an effort to Change History
Jin Ge Jin Qu hao
King of Fighters explained
Lovely Magical Curves
Melody "Creation"
Never 7, Ever 17 and Water
Optimizing Key Signature
Planning mobile robot on Tree (EASY Version)
Qualle? Quale?

(Do you know why some of the letters are red?)

As usual, there is a gift package on the contest website that contains some additional I/O data, special judge or data visualizer. Please make best use of it :)

Thanks Md. Mahbubul Hasan for problem A, B, C, H, J, K, N, Yubin Wang for problem D, E, F, G, I, L, N, P, Q, Yao Li for problem F, G, M, N, O, P, Feng Chen for problem A, B, C, H, J, K, M, O, and Yi Yang for problem M and O.

I hope you enjoy this contest, and my birthday :)

Hello, everyone! My name is Rujia Liu. I used to do a lot of problem solving and problemsetting, but after graduated from Tsinghua University, I'm spending more and more time on my company **L**

(You may realized that the paragraph above is copied from the texts of my 3rd , 4th and 5th contest, but that's me, lazy me.)

This time, my contest is all about myself. Every problem has a short story that is related to me. If you're interested in more details, you can ask me (rujia.liu@gmail.com) :)

me in a singing competition

This is a very important birthday for me, so I really want you enjoy this contest and know more about me. Please feel free to email me if you're interested in some non-algorithmic aspects of this contest (e.g. where to find the games I mentioned in the problems).

Ah, forgot to mention, problem A, B, C and K are good starts :)

Best regards,
Rujia Liu

A Special "Happy Birthday" Song!!!

There are *n* people (excluding myself) in my 30th birthday party. They sing the traditional "happy birthday" song:

Happy birthday to you! Happy birthday to you! Happy birthday to Rujia! Happy birthday to you!!!

Since I love music, I want to hear something more interesting, not that everyone sings together. Ah yes, I want one person to sing one word!

For example, there are three people: Mom, Dad, Girlfriend, I'd like them to sing like this:

```
Mom: Happy
Dad: birthday
Girlfriend: to
Mom: you
Dad: Happy
Girlfriend: birthday
Mom: to
Dad: you
Girlfriend: Happy
Mom: birthday
Dad: to
Girlfriend: Rujia
Mom: Happy
Dad: birthday
Girlfriend: to
Mom: you
```

Very nice, right? What if there are more than 16 people? That's easy: repeat the song until everyone has sung at least once :)

Please, don't stop in the middle of the song.

Input

There is only one test case. The first line contains a single integer *n* (1<=*n*<=100). Then each of the next *n* lines contains a capitalized name (i.e. one upper-case letter followed by zero or more lower-case letters). Each name contains at most 100 characters and do not have whitespace characters inside.

Output

Output the song, formatted as above.

Sample Input	Output for Sample Input
3 Mom Dad Girlfriend	Mom: Happy Dad: birthday Girlfriend: to Mom: you Dad: Happy Girlfriend: birthday Mom: to Dad: you Girlfriend: Happy Mom: birthday Dad: to Girlfriend: Rujia Mom: Happy Dad: birthday Girlfriend: to Mom: you

---

Problemsetter: Rujia Liu
Special Thanks: All of you, for participating in this contest; Mom and Dad, for giving birth to me :)

Baby Me

When I was born, I was 5 斤 2 两。Sorry for non-Chinese people. Here's what it means:

1 斤=0.5kg
1 两=0.05kg

So 5 斤 2 两 means 0.5*5+0.05*2=2.6kg.

Given similar information for other babies, your task is to find out their weights in kg.

Input

The first line contains the number of test cases *T*(*T*<=100). Each test case contains a string in format "*a* 斤 *b* 两"(1<=*a*<=10, 1<=*b*<=9) or "*a* 斤"(1<=*a*<=10).

The input file will be encoded with UTF-8 without BOM (if you don't know what it is, you can safely ignore it).

Output

For each test case, print the ACCURATE weight in kg (without trailing zeros).

Sample Input	Output for Sample Input
3 5 斤 2 两 7 斤 3 两 6 斤	Case 1: 2.6 Case 2: 3.65 Case 3: 3

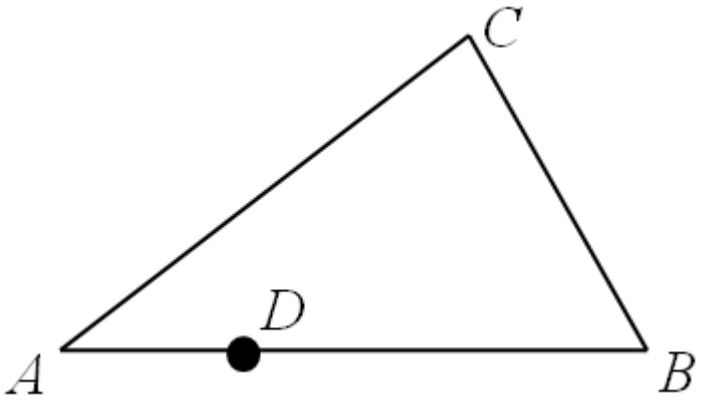
Problemsetter: Rujia Liu
Special thanks: Md. Mahbubul Hasan, Feng Chen

"Center" of perimeter midpoints

When I was a high school student, I learned that given a triangle ABC, denote D, E, F as the midpoints of AB, BC and CA, then three segments CD, AE, BF intersects at one point: the centroid.

Then I thought about the following question: if we change "midpoint" by "perimeter midpoint", can CD, AE, BF still intersect at one point?

To be precise, if CA+AD = DB+BC, we say D is the "perimeter midpoint" on AB.



It's not difficult to see that there is exactly one such point lying strictly inside the segment AB. Point E and F are defined similarly and also have unique positions.

Help (the younger) me to find out the answer!

Input

The first line contains the number of test cases T(T<=100). Each test case contains 6 integers x_1, y_1, x_2, y_2, x_3, y_3, whose absolute values do not exceed 100. These integers represent three non-collinear points A(x_1, y_1), B(x_2, y_2), C(x_3, y_3).

Output

For each test case, if CD, AE, BF intersect at one point, print the position of the intersection to 6 decimal places. Otherwise print "ERROR" (without quotes).

Sample Input	Output for Sample Input
2 -1 0 1 0 0 1 0 0 5 0 3 3	Case 1: 0.000000 0.171573 Case 2: 2.362911 0.665041

Problemsetter: Rujia Liu
Special thanks: Md. Mahbubul Hasan, Feng Chen

Detecting Code Snippets

Have you tried to modify code by changing some names of the identifiers? For example, the following codes can be changed to each other.

`int i, j;`
`i = 3;`
`j = i + 1;`
	`int a, i;`
	`a = 3;`
	`i = a + 1;`

However, "int" cannot be changed, because it's a keyword, not an identifier. Similarly, operators like "=" or "+" can't be changed, either.

To simplify the I/O, we use one upper-case letter to denote one kind of token that cannot be changed, and use one lower-case letter to denote an identifier whose name can be changed. However, two different lower-case letters cannot be changed to the same letter.

For example, if we use the following table:

Token	`int`	`,`	`;`	`=`	`3`	`+`	`1`
letter	A	B	C	D	E	F	G

Then the first program can be written as `AiBjCiDECjDiFGC`, the second one is `AaBiCaDECiDaFGC`.

Given a snippet (a small piece of code), can you find all its occurrences (possibly overlapping) in a large program?

Input

The first line contains the number of test cases T(T<=100). Each test case contains two lines, the first line is the program to be searched in, and the second line is the snippet. Both lines will contain letters only. There will be at most 10^6 characters in either string. The total input size will be less than 10M bytes.

Output

For each test case, print the number of occurrences of the snippet in the program.

Sample Input	Output for Sample Input

`2`
`ccddef`
`aab`
`ABdefDEabcABcaa`
`ABabc`
	`Case 1: 2`
	`Case 2: 1`

Bonus

Be sure to test your program with the data provided in our gift package.

Explanation

In the first sample, "ccd" and "dde" are both "changed" version of "aab". "def" is not counted because a cannot be changed into both d and e. In the second sample, "DEabc" is not counted because "AB" cannot be changed into "DE". "ABcaa" is not counted because b and c cannot be both changed to a.

Problemsetter: Rujia Liu
Special thanks: Gelin Zhou, Yubin Wang

Egyptian Fractions (HARD version)

Given a fraction *a*/*b*, write it as a sum of different Egyptian fraction. For example, 2/3=1/2+1/6.

There is one restriction though: there are *k* restricted integers that should not be used as a denominator. For example, if we can't use 2~6, the best solution is:

2/3=1/7+1/8+1/9+1/12+1/14+1/18+1/24+1/28

The number of terms should be minimized, and then the large denominator should be minimized. If there are several solutions, the second largest denominator should be minimized etc.

Input

The first line contains the number of test cases *T*(*T*<=100). Each test case begins with three integers *a*, *b*, *k*(2<=*a*<*b*<=876, 0<=*k*<=5, gcd(*a*,*b*)=1). The next line contains *k* different positive integers not greater than 1000.

Output

For each test case, print the optimal solution, formatted as below.

Sample Input	Output for Sample Input
5 2 3 0 19 45 0 2 3 1 2 5 121 0 5 121 1 33	Case 1: 2/3=1/2+1/6 Case 2: 19/45=1/5+1/6+1/18 Case 3: 2/3=1/3+1/4+1/12 Case 4: 5/121=1/33+1/121+1/363 Case 5: 5/121=1/45+1/55+1/1089

Extremely Important Notes

It's not difficult to see some inputs are harder than others. For example, these inputs are very hard input for every program I have:

596/829=1/2+1/5+1/54+1/4145+1/7461+1/22383
265/743=1/3+1/44+1/2972+1/4458+1/24519
181/797=1/7+1/12+1/2391+1/3188+1/5579
616/863=1/2+1/5+1/80+1/863+1/13808+1/17260
22/811=1/60+1/100+1/2433+1/20275
732/733=1/2+1/3+1/7+1/45+1/7330+1/20524+1/26388

However, I don't want to give up this problem due to those hard inputs, so I'd like to restrict the input to "easier" inputs only. I know that it's not a perfect problem, but it's true that you can still have fun and learn something, isn't it?

Some tips:

- Watch out for floating-point errors if you use double to store intermediate result. We didn't use double.
- Watch out for arithmetic overflows if you use integers to store intermediate result. We carefully checked our programs for that.

Problemsetter: Rujia Liu
Special thanks: Yubin Wang

Finding Black Circles

There are some black circles completely drawn on a white paper. Given the digital image of the paper, could you find the circles?

The width and height of the digital image are *w* and *h* pixels. Each pixel is a 1x1 square. The center of the top-left pixel is (0,0) and the center of the bottom-right pixel is (*w*-1,*h*-1). For each circle, the center coordinates and the radius are all integers. If a circle passes through a pixel (merely touching its border is not considered passing), the pixel is rendered black (1), otherwise it is white (0). Due to noises, at most 2% black pixels might become white. No white pixels will become black.

Input

The first line contains the number of test cases *T*(*T*<=20). Each test case begins with two integers *w* and *h* (30<=*w*,*h*<=100). The following *h* lines contain the digital image. There will be at least one and at most five circles. The radius of each circle will be at least 5. The judge input will be carefully chosen to avoid ambiguities and confusions.

Output

For each test case, print the number of circles *k*, and *k* tuples (*r*,*x*,*y*), each describing a circle centered at (*x*,*y*) with radius *r*. Tuples should be sorted lexicographically (first *r*, then *x*, and then *y*).

Sample Input

```
1
30 30
000000000000000000000000000000
000000000000011111110000000000
000000000000110000011000000000
000000000001100000001100000000
000000000011000000000110000000
000000001100000000000011000000
000000111111110000000001000000
000001110100011100000001000000
000011000100000110000000000000
000110000100000011000001000000
001100000100000001100001000000
001000000110000000100011000000
011000000011000000110110000000
010000000001100000011100000000
010000000000110000011000000000
010000000000011111110000000000
010000000000000000010000000000
010000000000000000010000000000
011000000000000000110000000000
001000000000000000100000000000
001000000000000001100000000000
000110000000000011000000000000
000011000000000110000000000000
000001110000011100000000000000
000000111111110000000000000000
000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
```

Output for Sample Input

```
Case 1: 2 (7,16,8) (9,10,15)
```

Bonus

Be sure to test your program with the data provided in our gift package.

Problemsetter: Rujia Liu
Special thanks: Yubin Wang, Yao Li

Good Friends

There are *n* people in a class. Some of them are good friends. They go out frequently. Given *m* activities, find out which people are good friends.

In this problem, if a set of at least two people attended in at least 20% activities together (possibly with some other people), they're regarded as good friends.

Note that "good friends" should be maximal. That means, if you add another person to the set, they will not be "good friends" anymore.

Input

The first line contains the number of test cases *T*(*T*<=5). Each test case begins with two integers *n*, *m* (2<=*n*<=30, 5<=*m*<=10,000), the number of people in the class, and the number of activities. Each of the *m* lines begins with an integer *k* (2<=*k*<=*n*), the number of people attending the activity, then *k* different integers (1~*n*) followed. People are numbered 1 to *n*.

Output

For each test case, print the number of "good friends" sets in the first line, and then print one line for each set. The numbers in each set should be sorted in increasing order. Sets should be sorted in lexicographical order. *Print a blank line after each test case.*

Sample Input	Output for Sample Input
1 10 10 7 1 2 3 4 5 6 7 7 2 4 5 6 7 9 10 8 1 2 4 5 6 7 9 10 6 2 5 6 7 8 10 6 1 3 4 6 9 10 8 1 2 4 5 6 7 9 10 6 1 2 3 5 6 7 6 2 5 6 7 8 10 7 2 5 6 7 8 9 10 8 2 3 4 5 6 7 9 10	Case 1: 6 1 2 3 5 6 7 1 2 4 5 6 7 9 10 1 3 4 6 2 3 4 5 6 7 2 5 6 7 8 10 3 4 6 9 10

Data Generation

Judge inputs are generated randomly this way:

- *n* = 30, *m* = 10000
- Generate four reference sets of people, each having 8~12 elements
- For each activity, randomly import one reference set, then add a random number of random people

Bonus

Be sure to test your program with the data provided in our gift package.

Problemsetter: Rujia Liu
Special thanks: Yubin Wang, Yao Li

Hadamard Gate

If you know a little bit of quantum computers, this problem is:

Given n Hadamard Gates in series and an input qubit, predict the measurement of the output.

If you don't know about quantum computers, keep on reading.

In quantum physics, superposition principle states that if a quantum system (e.g. an electron) can be in one of two states (denoted by |0> and |1>), it can also be in any linear superposition of those two states $a|0> + b|1>$, where a and b are two *complex numbers*, normalized so that $|a|^2 + |b|^2 = 1$. Such a superposition, $a|0> + b|1>$, is the basic unit of encoded information in quantum computers, called *qubit* (pronounced "cubit").

An elementary quantum operation is analogous to an elementary gate like the AND or NOT gate in classical circuit. One of the most important examples is the *Hadamard gate*, denoted by H, which operates on a single qubit. On input |1> or |0>, it outputs:

$$H(|1\rangle) = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$
$$H(|0\rangle) = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

Due to linearity of quantum physics, the output for an arbitrary superposition $a|0>+b|1>$ is aH(|0>)+bH(|1>).

However, the linear superposition is the private world of the quantum system. For us to get a glimpse of its state, we must make a measurement, and when we do so, we get a single bit of information - 0 or 1. If the state is $a|0>+b|1>$, then the outcome of the measurement is 0 with probability $|a|^2$ and 1 with probability $|b|^2$ (luckily we normalized so $|a|^2+|b|^2=1$).

Input

The first line contains the number of test cases T(T<=100). Each test case contains a single line of format "a_0 a_1 b_0 b_1 n", where a_0, a_1, b_0, b_1 are real numbers with at most 4 decimal places, denoting a qubit $(a_0+a_1i)|0> + (b_0+b_1i)|1>$, and n ($1<=n<=10^6$) is the number of Hadamard Gates.

Output

For each test case, print the probability that the measurement is 0, to 6 decimal places.

Sample Input	Output for Sample Input
2 1.0 0.0 0.0 0.0 1 0.017133 0.704420 0.410273 0.578943 1	Case 1: 0.500000 Case 2: 0.914848

Notes

(The following information is interesting, but will NOT help you solve this problem)

The act of measurement causes the system to change it state.

Take an electron as an example, if the outcome of the measurement is 0, then the new state of the system is |0> (the ground state), and if the outcome is 1, the new state is |1> (the excited state). This feature of quantum physics, that a measurement disturbs the system and forces it to choose (in this case ground or excited state), is another strange phenomenon with no classical analog.

You may conclude that quantum physics is completely different from, and unrelated to classical physics, but things are not that simple. Search the web for "Schrödinger's cat". You'll get amazed.

Furthermore, take a look at this year (2012)'s Nobel Prize in Physics[1], which was awarded "for ground-breaking experimental methods that enable measuring and manipulation of individual quantum systems".

References

[1] "The Nobel Prize in Physics 2012". Nobelprize.org. 12 Nov 2012 http://www.nobelprize.org/nobel_prizes/physics/laureates/2012/

Problemsetter: Rujia Liu
Special thanks: Md. Mahbubul Hasan, Feng Chen

In an effort to Change History

Have you ever hoped to change history, like me? I guess so.

Many people think it's logically impossible to change history, because the new history would evolve into a new present (here "present" means "now", not "gift"), leading to some contradicting facts.

While I haven't find a way to do time-travel and change history (if you know the way, please tell me!!!), at least I don't think it's logically impossible, because we can never sense the whole world (or worlds, if you believe in parallel universe/multiverse theory[1]). If we're clever enough, we can make a very small change to the history, which would evolve into a different, but consistent present (that means we can't sense the difference), which would in turn evolve into a better future!

Let's make a thought experiment [2]:

- There are x possible events in the past, labeled a_1~a_x.
- There are y possible events now (but some of them may be unable to sense), labeled b_1~b_y.
- There are z possible events that may happen in near future, labeled c_1~c_z.
- Present events only depend on past events, with known deterministic rules.
- Future events only depend on present events, with known deterministic rules.
- Make some changes to the past events (happened->not happened, and vice versa) so that the *sensed* present events remain the same, then some of the future events will change.

All these future events are good, so I want to maximize the number of these future events that will actually happen, by changing some of the past events. If there are more than one way to change history, make the smallest change (i.e. change the minimal number of past events).

To simplify the problem, each of the rules mentioned above is described by "event=formula", where "formula" is a string representation of a boolean formula which satisfies:

- Only three operators: AND (&&), OR (||), NOT (!) are supported.
- NOT has the highest priority and will not be repeated. i.e. !!x is invalid (but !(!x) is valid).
- AND and OR has the medium and lowest priority. The associativity of both AND and OR is left-to-right. i.e. x&&y&&z is actually (x&&y)&&z.
- Parentheses have usual meanings.
- There will be no whitespace characters within the formula.

Input

The first line contains a single integer T(T<=1000), the number of test cases. Each test case begins with three positive integers x, y, z (1<=x,y,z<=15). The second line contains x 0-1 integers, describing the past (before we change it). The i-th integer is 1 if and only if event a_i happened in the past. Each of the following y lines contains the formula of b_1~b_y (in this order). If the formula is preceded by an asterisk ('*'), that means we can sense whether that event is happening now (i.e. that boolean variable should not be changed). Otherwise that event can't be sensed. The following z lines contain the formulae of c_1~c_z (in this order), in the same format, except that there will be no asterisks. The lines containing rules will not have any whitespace characters inside.

Output

For each test case, print `"Increased from a to b"`. If we're unable to get more good future events, print `"Unable to improve future."`. If there is any solution, print the list of changed past event in the second line. If there is more than one solution, print the lexicographically smallest (when doing comparison, regard the solution as a list of integers). *Print a blank line after each test case.*

Sample Input	Output for Sample Input
3 2 1 2 0 1 b1=a1&&a2 c1=b1 c2=b1 2 1 2 0 1 *b1=a1&&a2 c1=b1 c2=b1 3 4 5 0 0 0 *b1=a1&&(a3||a2) b2=!a2 b3=a2&&a3 b4=a1 c1=!b2 c2=b3 c3=b4 c4=b2||!b4 c5=!b2||!b4	Case 1: Increased from 0 to 2. a1 Case 2: Unable to improve future. Case 3: Increased from 2 to 4. a2 a3

Bonus

Be sure to test your program with the data provided in our gift package.

Notes

[1] See: http://en.wikipedia.org/wiki/Multiverse
[2] See: http://en.wikipedia.org/wiki/Thought_experiment

Problemsetter: Rujia Liu
Special thanks: Yubin Wang

Jin Ge Jin Qu hao

(If you smiled when you see the title, this problem is for you ^_^)

For those who don't know KTV, see: http://en.wikipedia.org/wiki/Karaoke_box

There is one very popular song called Jin Ge Jin Qu(劲歌金曲). It is a mix of 37 songs, and is extremely long (11 minutes and 18 seconds)[1].

Why is it popular? Suppose you have only 15 seconds left (until your time is up), then you should select another song as soon as possible, because the KTV will not crudely stop a song before it ends (people will get frustrated if it does so!). If you select a 2-minute song, you actually get 105 extra seconds! ....and if you select Jin Ge Jin Qu, you'll get 663 extra seconds!!!

Now that you still have some time, but you'd like to make a plan now. You should stick to the following rules:

- Don't sing a song more than once (including Jin Ge Jin Qu).
- For each song of length *t*, either sing it for exactly *t* seconds, or don't sing it at all.
- When a song is finished, always immediately start a new song.

Your goal is simple: sing as many songs as possible, and leave KTV as late as possible (since we have rule 3, this also maximizes the total lengths of all songs we sing) when there are ties.

Input

The first line contains the number of test cases T(T<=100). Each test case begins with two positive integers n, t(1<=n<=50, 1<=t<=10^9), the number of candidate songs (BESIDES Jin Ge Jin Qu) and the time left (in seconds). The next line contains n positive integers, the lengths of each song, in seconds. Each length will be less than 3 minutes[2]. It is guaranteed that the sum of lengths of all songs (including Jin Ge Jin Qu) will be strictly larger than t.

Output

For each test case, print the maximum number of songs (including Jin Ge Jin Qu), and the total lengths of songs that you'll sing.

Sample Input	Output for Sample Input
2 3 100 60 70 80 3 100 30 69 70	Case 1: 2 758 Case 2: 3 777

Explanation

In the first example, the best we can do is to sing the third song (80 seconds), then Jin Ge Jin Qu for another 678 seconds.

In the second example, we sing the first two (30+69=99 seconds). Then we still have one second left, so we can sing Jin Ge Jin Qu for extra 678 seconds. However, if we sing the first and third song instead (30+70=100 seconds), the time is already up (since we only have 100 seconds in total), so we can't sing Jin Ge Jin Qu anymore!

Bonus

Be sure to test your program with the data provided in our gift package.

Notes

[1] I know that there are Jin Ge Jin Qu II and III, and some other unofficial versions. But in this problem please forget about them.

[2] I know that most songs are longer than 3 minutes. But don't forget that we could manually "cut" the song after we feel satisfied, before the song ends. So here "length" actually means "length of the part that we want to sing".

Problemsetter: Rujia Liu
Special thanks: Md. Mahbubul Hasan, Feng Chen

King of Fighters explained

King of Fighters (KOF) is one of my favorite fighting games. So when I tried to make my own fighting games (though I never actually started…), I took sometime investigating how the KOF system works.

After sometime, I came up with the following simple model:

- At any time, each person can be in one of its designed states (e.g. standing, running, jumping, punching etc).
- Each state has a set of frames.
- Each frame is an image (frame of animation) plus two areas: attacking area and weak area. Note that both areas can be composed of several disjoint regions.
- Taking into account both characters' positions, if one character's attacking area overlaps (with non-zero intersection area) with his opponent's weak area, the opponent gets hit.
- It's possible that both characters get hit at the same time.

(Athena in KOF97. In HTML version of this problem, it's a GIF animation)

For simplicity, both attacking areas and weak areas are approximated by union of rectangles. The rectangles might be intersecting. Only their union represents the attacking areas and weak areas.

Your task is to decide, given the positions of these areas, who is getting hit.

Input

The first line contains the number of test cases T(T<=100). Each test case contains two parts in the same format, describing the first character, then the second one. The first line of each part contains two integers a and w (0<=a,w<=5), the number of rectangles in the attacking area and weak area, respectively. Each of the a lines contains four non-negative integers x_1, y_1, x_2, y_2 (0<=x_1<x_2<=100, 0<=y_1<y_2<=100), that means the set of points (x,y) satisfying x_1<=x<=x_2, y_1<=y<=y_2 is in the attacking area. The next w lines describe the weak area in the same format.

Output

For each test case, print "`First`" if only the first character is hit, "`Second`" if only the second character is hit, "`Both`" if both are hit, "`Neither`" if neither is hit.

Sample Input

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

Output for Sample Input

```
Case 1: Second
Case 2: Both
Case 3: Neither
```

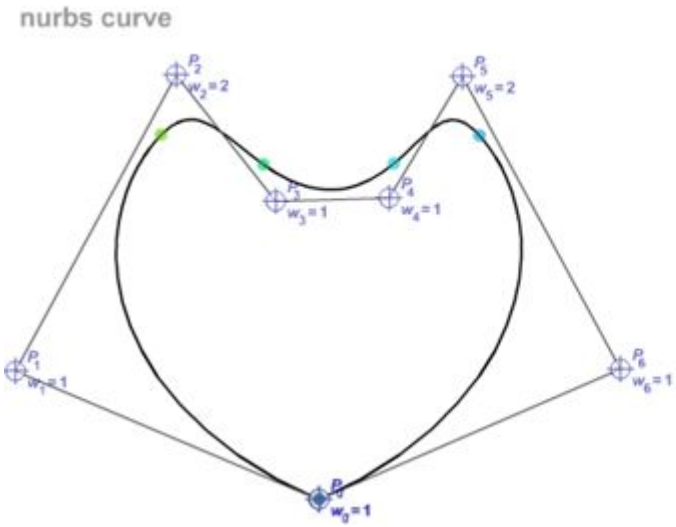
Bonus

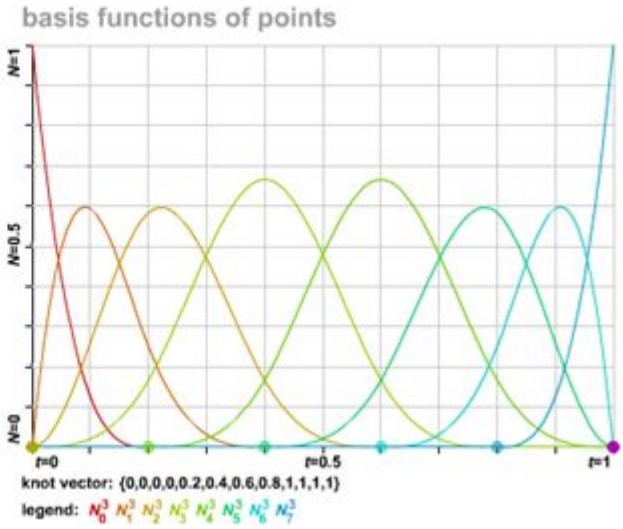
Be sure to test your program with the data provided in our gift package.

Problemsetter: Rujia Liu
Special thanks: Md. Mahbubul Hasan, Feng Chen, SNK (for creating KOF series)

Lovely Magical Curves

NURBS Curves are lovely and magical, because you can make a lot of interesting shapes from it:





Given two NURBS curves, your task is the find all their intersection points.

If you're not familiar with NURBS curves, here we go:

NURBS is a parametric curve which takes the following form:

$$C(u) = \frac{\sum_{i=1}^{n} w_i N_{i,k}(u) P_i}{\sum_{i=1}^{n} w_i N_{i,k}(u)}$$

Where u is the parameter, n is the number of control points, k is the degree of the curve, P_i and w_i are the location and weight of the i-th control point.

The basis function $N_{i,k}$ is defined recursively below:

$$N_{i,k}(u) = \frac{u - t_i}{t_{i+k} - t_i} N_{i,k-1}(u) + \frac{t_{i+k+1} - u}{t_{i+k+1} - t_{i+1}} N_{i+1,k-1}(u)$$

$$N_{i,0}(u) = \begin{cases} 1, & if\ t_i \le u < t_{i+1} \\ 0, & else \end{cases}$$

Where t_i is the i-th knot value. **In the formula above, 0/0 is deemed to zero**.

To understand the formulae above, here are some brief explanations of the parameters:

Degree. The *degree* is a positive integer. NURBS lines and polylines are usually degree 1 (linear curve), NURBS circles are degree 2 (quadratic curve), and most free-form curves are degree 3 or 5.

Control Points. The control points are a list of at least degree+1 points. One of easiest ways to change the shape of a NURBS curve is to move its control points[1]. Each control point has an associated number called weight. In this problem, weights are positive numbers. If you increase the weight of a control point, the curve is pulled toward that control point and away from other control points.

Knots. The knot vector is defined as U = [t_1, t_2, ... , t_m]. The relation between the number of knots m, the degree k, and the number of control points n is given by $m = n + k + 1$[2]. The sequence of knots in the knot vector U is assumed to be non-decreasing, i.e. $t_i <= t_{i+1}$. Each successive pair of knots represents an interval [t_i, t_{i+1}) for the parameter values to calculate a segment of a shape. **Thus, the whole NURBS curve is defined within [t_1, t_m)**. The number of times a knot value is duplicated is called the knot's *multiplicity*, which should be no more than the *degree*. Duplicate knot values in the middle of the knot list make a NURBS curve less smooth.

If you're still puzzled after reading all the information above, suppose we're moving u from t_1 towards t_m (but never reach t_m), then the point $C(u)$ will move long the NURBS curve we define.

Input

The first line contains the number of test cases T(T<=25). Each test case contains two parts, one for each NURBS curve. Each curve begins with two integers n and m (2<=n<=20), the number of control points and the number of knots. Each of the next n lines contains three real numbers x, y, w (0<=x,y<=10, 0<w<=10), describing a control point (x, y) with weight w. The next line contains m real numbers, describing the knot vector. The first knot value is always 0 and the last one is always 1. The degree of both NURBS curves will be 1, 2, 3 or 5.

Output

For each test case, print the number of intersection points in the first line, then each point is printed in a following line. The coordinates should be rounded to three decimal places, and points should be sorted lexicographically (i.e. points with smaller x-coordinate comes earlier). Inputs are carefully designed so that the minimal difference of x-coordinate between any two intersection points will be at least 0.005 (otherwise the sorting result might be affected by numerical stability). *Print a blank line after each test case.*

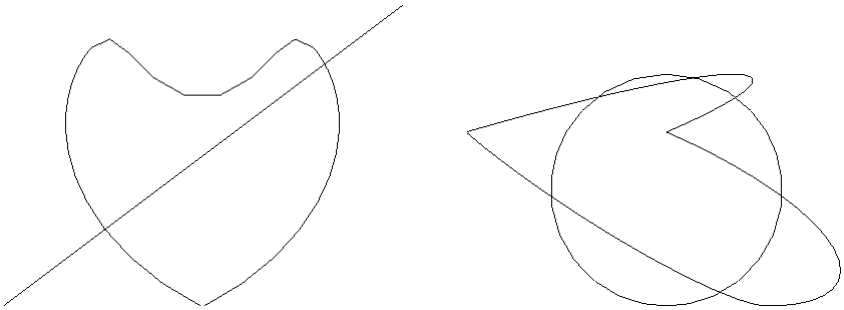
Sample Input	Output for Sample Input
2 8 12 2 0 1 0 1 1 1 3 2 1.5 2 1 2.5 2 1 3 3 2 4 1 1 2 0 1 0 0 0 0 0.2 0.4 0.6 0.8 1 1 1 1 2 4 0 0 1 4 3 1 0 0 1 1 7 10 1 1.732 1 0 0 0.5 2 0 1 4 0 0.5 3 1.732 1 2 3.464 0.5 1 1.732 1 0 0 0 0.333 0.333 0.667 0.667 1 1 1 7 10 0 1.732 1 2 0 0.5 3 0 1 6 0 0.5 2 1.732 1 6 3.464 0.5 0 1.732 1 0 0 0 0.333 0.333 0.667 0.667 1 1 1	Case 1: 2 (1.029, 0.772) (3.221, 2.416) Case 2: 6 (0.847, 1.092) (1.307, 2.078) (2.283, 2.274) (2.538, 0.133) (2.693, 2.078) (3.153, 1.092)

Bonus

You may find the data visualizer and additional testdata in the gift package useful. It requires Python and PyOpenGL.

Notes

[1] You can try it out: http://geometrie.foretnik.net/files/NURBS-en.swf
[2] In OpenNURBS/Rhinoceros website, $m = n + k - 1$. The algorithm presented here is referred as "some older algorithms". When solving this problem, please stick to this problem description.
[3] The pictures of the samples are shown below:



Problemsetter: Rujia Liu
Special thanks: Yubin Wang

Melody "Creation"

I have written two songs, one is called "烟雨"[1] and the other one "沧浪"[2].

Writing songs requires a lot of work, especially when you need to give a performance on a stage, in a serious competition. You need to polish your lyrics, melody, orchestration[3] again and again, and you need a lot of rehearsals with your friends (who play the piano/flute etc, or sing harmony).

Here is the photo of "my team" before the performance of "沧浪".

You may wonder how I wrote songs. The short answer is: I didn't "write" songs, I just "select" from music fragments that automatically came into my mind. So it is essential to note down the melody quickly when it suddenly appears.

Unfortunately, I don't have absolute pitch or well-developed long-term relative pitch, so I have to stick to the movable do system[4]. I use 1~7 to represent do, re, mi, fa, sol, la to si, and #1 to represent #Do etc. When I'm changing the key, I'll write syllables in both keys (formatted as "s1=s2" that means "this note is syllable s1 in the old key, which is also syllable s2 in the new key"), like this:

5 5 6 5 1=4 3 | 1 1 2 1 5 4 | 1=5 5 5 3 1 7=3 2 | b7 b7 6 4 5=2 1 ||

Well, I admit this is a weird way to transcript the birthday song, but ... You know what I mean, right?

However, after I wrote down the whole thing, I often find a lot of ridiculous modulations (i.e. changing key) -- I just couldn't understand why I changed the key. Luckily, I don't need to know why. All I need is a small tool that rewrites the (possibly weird) melody in a "reasonable" way. By "reasonable", I mean a good balance between the number of modulations and the number of accidentals (i.e. sharp or flats).

Given the maximum number of accidentals, find a transcript with minimal modulations.

Notes:

- Don't worry about the missing octaves information (e.g. the "5 5 5 3" part actually contains three "so"s in two different octaves) and rhythm information. I can always remember them.
- I always explicitly written down every accidental (both before and after rewriting). For example, if I write "b7 7", the second note is a normal "si". Also, I never write things like "b4" or "#7" (both before and after conversion), I'll write "3" and "1" instead.
- When counting accidentals, "#5=b3" contains two accidentals, though it's only one note.
- If there are multiple solutions, print the lexicographically smallest one (the melody should be regarded as a sequence of strings). For example, "1 2 b3 ||" can be rewritten to "1=2 3 4 ||" or "1 2=3 4". The second one is better. However, "6 7 1" is better than both, because there is no modulation at all!
- You may change the first note. For example, the optimal way to express both "b7 b7 b7 b7 ||" and "5 5 5 5 ||" is "1 1 1 1 ||".

Input

The first line contains the number of test cases T(T<=100). Each test case contains two lines. The first line contains the maximal number of accidentals, and the second line contains the initial transcript, ending with a double barline (||). Adjacent symbols are separated by a single space. There will be no more than 100 notes/barlines in each transcript.

Output

For each test case, print the transcript with minimal modulations. If there are multiple solutions, print the lexicographically smallest one (don't forget the transcript is regarded as a sequence of strings, not a big string). Barlines ("|" and "||") should be output as-is.

Sample Input	Output for Sample Input
5 0 5 5 6 5 1=4 3 | 1 1 2 1 5 4 | 1=5 5 5 3 1 7=3 2 | b7 b7 6 4 5=2 1 || 1 b7 b7 b7 b7 7 || 0 b7 b7 b7 b7 7 || 5 6 7 6 7 7 7 || 1 1 #1 2 #2 3 #4 ||	Case 1: 5 5 6 5 1 7 | 5 5 6 5 2 1 | 5 5 5 3 1 7 6 | 4 4 3 1 2 1 || Case 2: 1 1 1 1 #1 || Case 3: 3 3 3 3 4 || Case 4: #2 4 #2 4 4 4 || Case 5: 2 #2 3 4=3 4 5 ||

Background

For those who are not familiar with music, here is some information:

- There are 12 different syllables in the movable do system: 1, #1, 2, #2, 3, 4, #4, 5, #5, 6, #6, 7. The pitch interval of adjacent syllables is one semitone. You can label them 0~11, and the "pitch interval" calculation is done mod 12. For example, the pitch interval of #6 and 2 is 2-10=4 (mod 12), you can also get this by counting: #6 ->7 -> 1 -> #1 -> 2. Four semitones.
- When we hear "1 2 3", we may also consider it "4 5 6", because the sequence of "adjacent pitch interval" of both melody is (2, 2). Similarly, "2 3 4 5" and "6 7 1 2" (the last "1 2" is in a higher octave) are similar, because their sequence of "adjacent pitch interval" are both (2, 1, 2). Here "similar" means "we can rewrite either melody to the other".
- Now consider the last example, "1 #1 2 #2 3 #4", the pitch interval sequence is (1, 1, 1, 1, 2). The rewritten transcript has one modulation, so it can be divided into two parts: "2 #2 3 4=3" and "4=3 4 5", the pitch interval sequence of the first part is (1, 1, 1), and the second sequence is (1, 2). Note that in the first part "4=3" uses its old syllable "4", and in the second part, "4=3" uses its new syllable, "3". Another solution with minimal number of modulation but lexicographically larger, is "3 4 #4 5=3 4 5 ||"

Notes

[1] This song was written in 2005, but did not get any serious treatment until in 2007, when I participated in Tsinghua University's Campus singers' competition.
[2] This song was written especially for the final round of Tsinghua University's Campus singers' competition, April 2008.
[3] The lyrics of both songs were written by Bing Song, harmony primarily by Kai Chung Tam and Zhen Shang, orchestration was done by Jun Huang, and the initial piano score was written by Qindi Li. I thank them from the bottom of my heart.
[4] It's called "首调唱法" in Chinese. See http://en.wikipedia.org/wiki/Solf%C3%A8ge

---

Problemsetter: Rujia Liu
Special thanks: Yubin Wang, Yi Yang, Yao Li, Feng Chen

Never7, Ever17 and Water

Infinity is one of my favorite game series. If you have ever played Never7, you'll know that Haruka, Kurumi and Izumi all like water.



If you have ever played Ever17, you'll know that the entire story of Ever17 happened under the sea.

So "water" is very important in these two games. That's why I made this problem, in which you need to design an amazing water system.

It is a circulation system (i.e. no "source" or "sink" of water) consisting of m pipes connecting n junctions. No water is created or destroyed during circulation, so for each junction, the total amount of water coming into it should be the same of the amount going out of it, for each unit time.

Pipes are deformable, so we can easily control the water speed through them. However, each pipe has a pair of lower-bound and upper-bound, and the actual water speed through it must lie within the bounds.

Pipes are transparent, so you can actually see how fast the water is going through each pipe. To make it look beautiful, the water speed should be as balanced as possible. I.e. the difference between the maximal water speed and the minimal water speed should be minimized.

Could you find the optimal design?

Input

The first line contains the number of test cases T(T<=100). Each test case begins with two integers n and m (2<=n<=50, 1<=m<=200), the number of junctions and the number of pipes. Each of the following m lines contains four integers u, v, b, c (1<=u,v<=n, u!=v, 0<=b<=c<=100), that means there is a pipe connecting junction u and v, and the speed of water flowing from u to v, denoted by f, should satisfy b<=f<=c. Junctions are numbered 1 to n, and pipes are numbered 1 to m (in the same order that they appear in the input). Note that the pipes need not be straight, so two junctions can be connected by several pipes.

Output

For each test case, print the minimal difference between the maximum speed and the minimum speed to five decimal places. If there is no solution, print -1.

Sample Input	Output for Sample Input
3 4 4 1 2 1 4 2 3 2 5 3 4 3 6 4 1 4 7 3 3 1 2 1 2 2 3 2 3 3 1 3 4 2 3 1 2 3 3 2 1 0 10 2 1 0 10	Case 1: 0.00000 Case 2: -1 Case 3: 1.50000

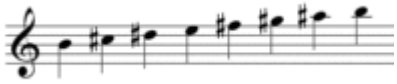
Problemsetter: Rujia Liu
Special thanks: Md. Mahbubul Hasan, Yubin Wang, Yao Li, KID (for creating infinity series)

Optimizing Key Signature

"A key signature is not the same as a key; key signatures are merely notational devices. They are convenient principally for diatonic or tonal music. Some pieces that change key (modulate) insert a new key signature on the staff partway, while others use accidentals: natural signs to neutralize the key signature and other sharps or flats for the new key."

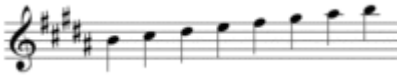
-- Wikipedia

You see, key signatures can be really confusing at times. For example, consider the follow score, whose key signature is "0 sharp/flat" (which usually indicates C major/A minor):



(In ASCII format: B #C #D E #F #G #A B)

It's more reasonable to rewrite the score in the key signature "5 sharp" (which usually indicates B major/g# minor):



(In ASCII format: B C D E F G A B)

This is more "natural" because no accidentals (i.e. sharp/flat/natural sign) exist.

Given a music piece with a key signature, your task is the find the best key signature that minimizes the number of accidentals. Under that condition, the number of sharps/flats in the key signature should be minimized. Note that you CANNOT change the staff position of any note. For example, you can't change #G to bA even if this can save accidentals.

To simplify this problem, we only consider sharp, flat and natural accidentals (no double accidentals or half-sharps etc), and all the notes are in the same octave. No notes will be tied to the same note across the barline.

In your optimized score, no courtesy or cautionary accidental should be placed by a note whose pitch is, strictly speaking, already given by the key signature. However, in the original piece, such accidentals might exist to make people's life easier, because even musicians may sometimes get confused when reading the score at speed.

Important notes:

- Don't forget the effect of accidental last until the end of the measure. For example, the measure bA A A A contains only one accidental, but the last three notes are also affected, so all four notes are of the same pitch.
- The effect of the accidental has to be understood in relation to the "natural" meaning of the note's staff position, so in the key signature with 3 sharps, a bG note is actually TWO semitones lower than G (which is actually #G, thanks to the key signature).

Input

The first line contains the number of test cases *T*(*T*<=100). Each test case begins with the initial key signature formatted as "m#/b", that means the key signature has *m* sharps/flats (0<=*m*<=7). If *m*=0, then the #/b part is omitted. The next line contains the notes, separated by barlines '|' and ending with a double barline "||". Consecutive notes and barlines are separated by a single space. Each note is formatted as two characters, where the first is either empty or one of #(sharp), b(flat) or n(natural), the second character is one of C,D,E,F,G,A,B (upper-case). There will be at most 10 measures and 100 notes in each score.

Output

For each test case, print the minimum number of accidentals in the first line, and then the best key signatures having the least number of sharps/flats, in the same format as the input. Key signatures should be sorted lexicographically. *Print a blank line after each test case.*

Sample Input	Output for Sample Input
4 0 C D E #F G | A B C || 1b F F bB B | B B #F F bD || 7# C D bE E F bG | G A bB || 0 #F #C bB bE ||	Case 1: 0 1# Case 2: 1 4b Case 3: 3 5# Case 4: 2 2# 2b

Appendix

Staff. In standard Western musical notation, the staff, or stave, is a set of five horizontal lines and four spaces that each represent a different musical pitch.

See: http://en.wikipedia.org/wiki/Staff_%28music%29

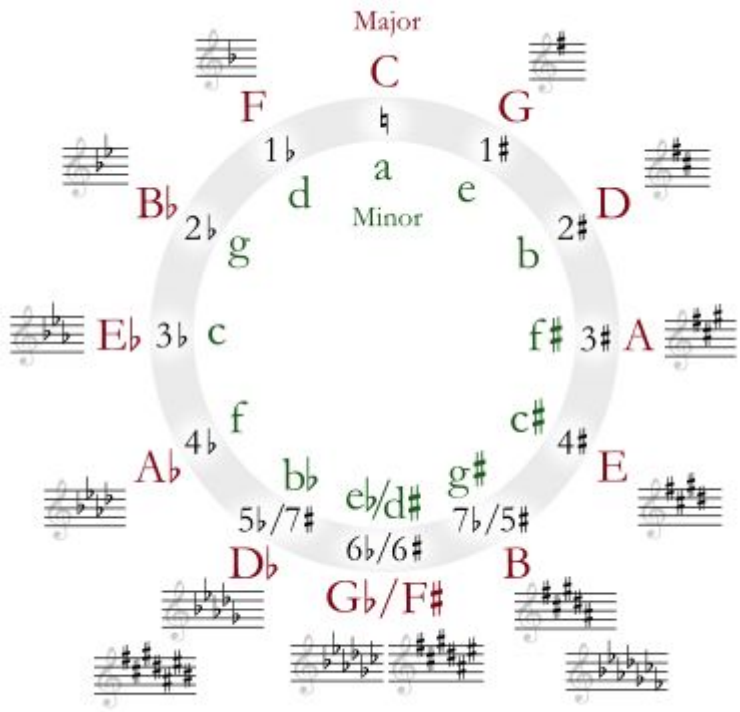
Key signature. A key signature is a series of sharp or flat symbols placed on the staff, designating notes that are to be consistently played one semitone higher or lower than the equivalent natural notes unless otherwise altered with an accidental...

See: http://en.wikipedia.org/wiki/Key_signature

Accidentals. An accidental is a note whose pitch (or pitch class) is not a member of a scale or mode indicated by the most recently applied key signature ...In most cases, a sharp raises the pitch of a note one semitone while a flat lowers it a semitone. A natural is used to cancel the effect of a flat or sharp. This system of accidentals operates in conjunction with the key signature, whose effect continues throughout an entire piece, unless canceled by another key signature. An accidental can also be used to cancel or reinstate the flats or sharps of the key signature...accidentals have been understood to continue for the remainder of the measure in which they occur, so that a subsequent note on the same staff position is still affected by that accidental, unless marked as an accidental on its own...

See: http://en.wikipedia.org/wiki/Accidental_%28music%29

Finally, looking that Circle of fifth if you're not sure about some key signatures:



Problemsetter: Rujia Liu
Special thanks: Yi Yang, Yao Li, Feng Chen

Planning mobile robot on Tree (EASY Version)

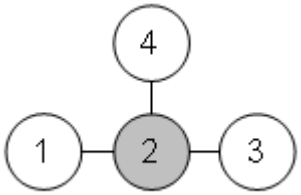
We are given a connected, undirected graph G on *n* vertices. There is a mobile robot on one of the vertices; this vertex is labeled *s*. Each of several other vertices contains a single movable obstacle. The robot and the obstacles may only reside at vertices, although they may be moved across edges. No vertex may ever contain more than one movable entity (robot or obstacles).

In one step, we may move either the robot or one of the obstacles from its current position *v* to a vacant vertex adjacent to *v*. Our goal is to move the robot to a designated vertex *t* using the smallest number of steps possible.

Let us call this graph motion planning with one robot, or GMP1R for short. In this problem, we restrict the graph G to be a tree, namely TMP1R.

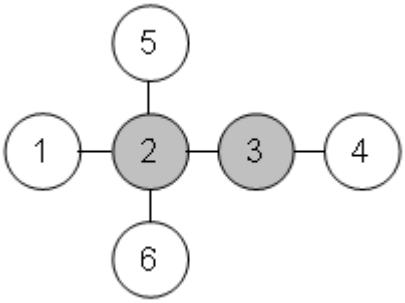
Here are some examples (gray circles represent obstacles).

Example 1 (s=1, t=3):



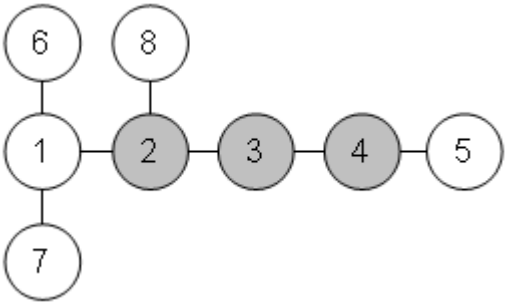
Move the obstacle 2-4, and then move the robot 1-2-3. Total: 3 moves.

Example 2 (s=1, t=4):



Move obstacle 2-5, then 3-2-6, and then move the robot 1-2-3-4. Total: 6 moves.

Example 3 (s=1, t=5):



Move the robot 1-6, then obstacle 2-1-7, then robot 6-1-2-8, then obstacle 3-2-1-6, then 4-3-2-1, and finally robot 8-2-3-4-5. Total: 16 moves.

Input

The first line contains the number of test cases T(T<=340). Each test case begins with four integers n, m, s, t(4<=n<=15, 0<=m<=n-2, 1<=s,t<=n, s!=t), the number of vertices, the number of obstacles and the label of the source and target. Vertices are numbered 1 to n. The next line contains m different integers not equal to s, the vertices containing obstacles. Each of the next n-1 lines contains two integers u and v (1<=u<v<=n), that means there is an edge u-v in the tree.

Output

For each test case, print the minimum number of moves k in the first line. Each of the next k lines contains two integers a and b, that means to move the robot/obstacle from a to b. If there is no solution, print `-1`. If there are multiple solutions, any will do. *Print a blank line after each test case.*

Sample Input

```
3
4 1 1 3
2
1 2
2 3
2 4
6 2 1 4
2 3
1 2
2 3
3 4
2 5
2 6
8 3 1 5
2 3 4
1 2
2 3
3 4
4 5
1 6
1 7
2 8
```

Output for Sample Input

```
Case 1: 3
2 4
1 2
2 3

Case 2: 6
2 5
3 2
2 6
1 2
2 3
3 4

Case 3: 16
1 6
2 1
1 7
6 1
1 2
2 8
3 2
2 1
1 6
4 3
3 2
2 1
8 2
2 3
3 4
4 5
```

Bonus

You may find the sample data and special judge program in the gift package useful.

Notes

TMP1R can be solved in $O(n^4)$ time. That's overkill for this problem, but if you're interested, please take a look at this:

Vincenzo Auletta , Domenico Parente , Pino Persiano, *A New Approach to Optimal Planning of Robot Motion on a Tree with Obstacles,* in Proc. of the 4-th European Symposium on Algorithms, 1996.

Problemsetter: Rujia Liu
Special thanks: Yubin Wang, Yao Li

Qualle? Quale?

If you speak German, you should know the first word in the title. If you speak Italian, you should know the second one :)

Why I use these two words? Because I don't have better choices :(I want my problems' titles to start with A, B, C, etc. This is problem Q so it has to begin with the letter Q.

Sometimes it's difficult. Each problem title has to tell people something about the problem itself, so it can't be arbitrary. If I can't find a suitable title in English, I have to try other languages like Chinese (for example, see the last three problems in Rujia Liu's Present 5 ^_^).

Here is an example.

No	English	French	Chinese
1	A	B	C
2	D	-	B
3	C	B	-
4	E	-	E
5	C	A	-

The title of problem 1 in English starts with A, and the French version starts with B. The Chinese title of problem 1 starts with C. A hyphen means "N/A", so problem 2 doesn't have a French version.

One possible combination is (note that each problem should be used exactly once):

Problem A	Problem 1 in English
Problem B	Problem 3 in French
Problem C	Problem 5 in English
Problem D	Problem 2 in English
Problem E	Problem 4 in Chinese

Could you tell me all the possible language combinations? For each combination, all the languages in it must be used (i.e. You can't say the combination is {English,Chinese} if none of the problems actually used the Chinese version).

Input

The first line contains the number of test cases T(T<=500). Each test case begins with two integers n, m (3<=n<=26, 1<=m<=5), the number of problems and the number of languages. The next n lines contains the table containing the first m upper-case letters or '-'. The j-th column in the i-th row in the table is the first letter of the title in the j-th language. A special character '-' means that version does not exist.

Output

For each test case, print all possible combinations in a single line. Each combination is a string of languages used (languages are labeled 1 to m) in increasing order. Shorter combinations should come first. Combinations of the same length should be sorted in increasing order. If the problemset cannot be made, print -1.

Sample Input

```
4
5 3
ABC
D-B
CB-
E-E
CA-
3 2
AB
C-
-C
4 4
-A--
BB--
--CC
--D-
3 4
AAAA
BBBB
CCCC
```

Output for Sample Input

```
Case 1: 12 123
Case 2: -1
Case 3: 23 123 234 1234
Case 4: 1 2 3 4 12 13 14 23 24 34 123 124 134 234
```

Notes

The judge input for this problem is randomly generated, so don't worry if your algorithm might fail on some deliberately hand-crafted test cases.

Problemsetter: Rujia Liu
Special thanks: Yubin Wang