– Beehives –

Bees are one of the most industrious insects. Since they collect nectar and pollen from flowers, they have to rely on the trees in the forest. For simplicity they numbered the n trees from 0 to $n-1$. Instead of roaming around all over the forest, they use a particular list of paths. A path is based on two trees, and they can move either way i.e. from one tree to another in straight line. They don't use paths that are not in their list.

As technology has been improved a lot, they also changed their working strategy. Instead of hovering over all the trees in the forest, they are targeting particular trees, mainly trees with lots of flowers. So, they planned that they will build some new hives in some targeted trees. After that they will only collect their foods from these trees. They will also remove some paths from their list so that they don't have to go to a tree with no hive in it.

Now, they want to build the hives such that if one of the paths in their new list go down (some birds or animals disturbs them in that path) it's still possible to go from any hive to another using the existing paths.

They don't want to choose less than two trees and as hive-building requires a lot of work, they need to keep the number of hives as low as possible. Now you are given the trees with the paths they use, your task is to propose a new bee hive colony for them.

INPUT

Input starts with an integer T ($T \leq 50$), denoting the number of test cases.

Each case starts with a blank line. Next line contains two integers n ($2 \leq n \leq 500$) and m ($0 \leq m \leq 20000$), where n denotes the number of trees and m denotes the number of paths. Each of the next m lines contains two integers u v ($0 \leq u, v < n, u \neq v$) meaning that there is a path between tree u and v. Assume that there can be at most one path between tree u to v, and needless to say that a path will not be given more than once in the input.

OUTPUT

For each case, print the case number and the number of beehives in the proposed colony or '**impossible**' if its impossible to find such a colony.

NOTE

Dataset is huge. Use faster I/O methods.

INPUT EXAMPLE

```
3

3 3
0 1
1 2
2 0

2 1
0 1

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

OUTPUT EXAMPLE

```
Case 1: 3
Case 2: impossible
Case 3: 3
```

– Bits Equalizer –

You are given two non-empty strings S and T of equal lengths. S contains the characters '0', '1' and '?', whereas T contains '0' and '1' only. Your task is to convert S into T in minimum number of moves. In each move, you can

1. change a '0' in S to '1'
2. change a '?' in S to '0' or '1'
3. swap any two characters in S

As an example, suppose S = "01??00" and T = "001010". We can transform S into T in 3 moves:

- Initially S = "01??00"
- Move 1 – change $S[2]$ to '1'. S becomes "011?00"
- Move 2 – change $S[3]$ to '0'. S becomes "011000"
- Move 3 – swap $S[1]$ with $S[4]$. S becomes "001010"
- S is now equal to T

INPUT

The first line of input is an integer C ($C \leq 200$) that indicates the number of test cases. Each case consists of two lines. The first line is the string S consisting of '0', '1' and '?'. The second line is the string T consisting of '0' and '1'. The lengths of the strings won't be larger than 100.

OUTPUT

For each case, output the case number first followed by the minimum number of moves required to convert S into T. If the transition is impossible, output -1 instead.

INPUT EXAMPLE

```
3
01??00
001010
01
10
110001
000000
```

OUTPUT EXAMPLE

```
Case 1: 3
Case 2: 1
Case 3: -1
```

– LCM Pair Sum –

One of your friends desperately needs your help. He is working with a secret agency and doing some encoding stuffs. As the mission is confidential he does not tell you much about that, he just want you to help him with a special property of a number. This property can be expressed as a function $f(n)$ for a positive integer n. It is defined as:

$$f(n) = \sum_{\substack{1 \le p \le q \le n \\ lcm(p,q)=n}} (p+q)$$

In other words, he needs the sum of all possible pairs whose least common multiple is n. (The least common multiple (LCM) of two numbers p and q is the lowest positive integer which can be perfectly divided by both p and q). For example, there are 5 different pairs having their LCM equal to 6 as (1, 6), (2, 6), (2, 3), (3, 6), (6, 6). So $f(6)$ is calculated as $f(6) = (1+6)+(2+6)+(2+3)+(3+6)+(6+6) = 7+8+5+9+12 = 41$.

Your friend knows you are good at solving this kind of problems, so he asked you to lend a hand. He also does not want to disturb you much, so to assist you he has factorized the number. He thinks it may help you.

INPUT

The first line of input will contain the number of test cases T ($T \le 500$). After that there will be T test cases. Each of the test cases will start with a positive number C ($C \le 15$) denoting the number of prime factors of n. Then there will be C lines each containing two numbers P_i and a_i denoting the prime factor and its power (P_i is a prime between 2 and 1000) and ($1 \le a_i \le 50$). All the primes for an input case will be distinct.

OUTPUT

For each of the test cases produce one line of output denoting the case number and $f(n)$ modulo 1000000007. See the output for sample input for exact formatting.

INPUT EXAMPLE

```
3
2
2 1
3 1
2
2 2
3 1
1
5 1
```

OUTPUT EXAMPLE

```
Case 1: 41
Case 2: 117
Case 3: 16
```

– RNA Secondary Structure –

RNA, which stands for Ribonucleic Acid, is one of the major macromolecules that are essential for all known forms of life. It is made up of a long chain of components called nucleotides. Each component is made up of one of 4 bases and are represented using A, C, G or U. The primary structure of RNA is a sequence of these characters. The secondary structure of RNA refers to the base pairing interactions between different components. More specifically, base A can pair up with base U and base C can pair up with base G. The stability of the RNA secondary structure depends on the total number of base pairs that can be formed. The final structure is the one that contains the maximum number of base pairs.

Let's represent the primary structure as a string consisting of characters from the set (ACGU). The rules of secondary structure formation are as follows:

1. Any base A can form a pair with any base U
2. Any base C can form a pair with any base G
3. Each base can be part of at most one pair.
4. Let's assume $w < x$, $y < z$ and $w < y$. If base at index w forms a pair with base at index x and base at index y forms a pair with base at index z, then one of the following two conditions must be true:

$$\text{y} > \text{x}$$
$$\text{z} < \text{x}$$

5. There can be at most K pairs between C and G.

You will be given the primary structure of the RNA of a certain species and your job is to figure out the total number of base pairings in the final secondary structure based on the constraints mentioned above.

You will be given the primary structure in a compressed format that uses Run-length encoding. In this type of data compression, consecutive characters having the same value is replaced with a single character followed by its frequency. For example, "AAAACCGAAUUG" will be represented using "A4C2G1A2U2G1". That means the primary structure will be given in the format $< c_1 f_1 c_2 f_2 c_3 f_3 \ldots c_n f_n >$, where c_i is from the set (ACGU) and f_i is a positive integer.

The species that we are dealing with have the following properties:

1. $f1 + f2 + f3 + ...fn \leq 10050$
2. $f1 \leq 5000$

3. $fn \leq 5000$
4. $f2 + f3 + f4 + \ldots + fn - 1 \leq 50$

INPUT

The first line of input is an integer T ($T \leq 200$) that indicates the number of test cases. Each case contains two lines. The first line is the primary structure given in run-length encoded format. The second line gives you the value of K ($0 \leq K \leq 20$), that gives an upper limit on the number of $C - G$ base pairs that can be in the final secondary structure.

OUTPUT

For each case, output the case number followed by the maximum number of base pairs that can be formed. Look at the samples for exact format.

INPUT EXAMPLE

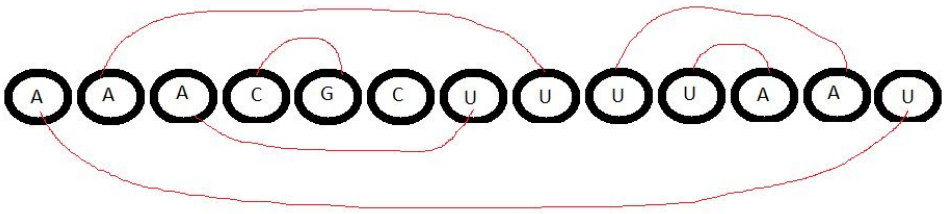
```
3
A3C1G1C1U4A2U1
1
A3C1G1C1U4A2U1
0
A100U200
2
```

OUTPUT EXAMPLE

```
Case 1: 6
Case 2: 5
Case 3: 100
```

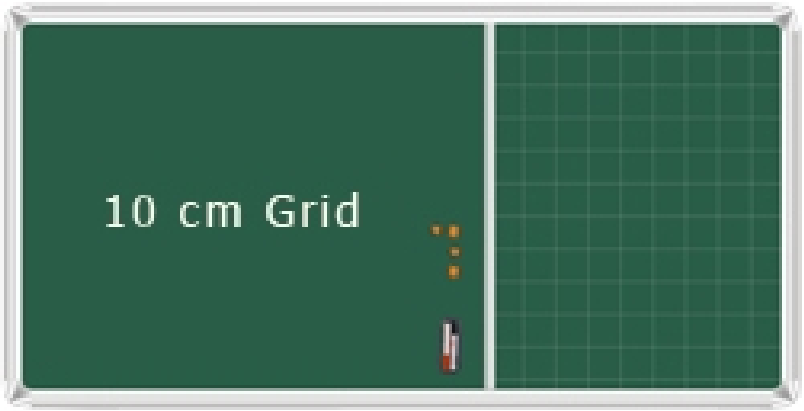
ILLUSTRATION

One possible final secondary structure for case 1 is depicted below that shows the 6 base pairings.



– Old School Days –

I don't know how you feel about your school time, but I become maudlin when I remember those days. One of our teachers told us at the final year that among all the educational institutions in life one misses his/her school days the most. And yes, I miss those days a lot.



Let me tell you that we had a grid board in our school and it was not as banal as it looks in the picture. The board was colorful and we also had different color chalks to use on it. Can you imagine how exciting it was for me when I first saw this board? In class break we used to draw on this board and play different games, few of them I can recall.

One of them was like this– firstly two players will mark some grid points. Then they will toss deciding who plays first. At each turn a player marks four points as A, B, C and D. Then join (A, B), (B, C), (C, D) and (D, A) to form a quadrilateral. Twice of the area of that quadrilateral is added to his score and the turn changes. (In case, if you are wondering why twice– it is just to ensure that the score is always integer) A player can not draw a quadrilateral if it was drawn before. However, you can use previously used points. For example, suppose there are 5 points on the grid, P, Q, R, S and T. First player can choose, $(A, B, C, D) = (P, Q, R, S)$, but then the second player can not choose $(A, B, C, D) = (R, S, P, Q)$ because both of them depicts the same quadrilateral. If both of the players play optimally to maximize their own score I wonder what could be the sum of their scores.

So your task is to construe this game. You are given co-ordinates of N distinct points, if two players play the above mentioned game optimally then what is the sum of their scores?

INPUT

First line of the input data contains a positive integer T denoting the number of test cases ($T <= 50$).

For each case, first line contains a positive integer N ($N <= 700$). Hence follows N co-ordinates of the points (x, y). In case you don't know, I should say - I am not from your time, I was brought here by a few scientists from future. And in my time we use huge boards so the absolute value of the co-ordinates can be as large as 10^6. Just to ensure that no one can draw a degenerate quadrilateral, no three points will be collinear.

OUTPUT

For each test case output the case number followed by sum of the scores. Since this score can be huge just print the answer mod 1000003.

INPUT EXAMPLE

```
2

4
2 0
0 2
-2 0
0 -2

5
2 0
0 2
-2 0
0 -2
2 2
```

OUTPUT EXAMPLE

```
Case 1: 16
Case 2: 72
```

– Sentry Robots –

We need to guard a set of points of interest using sentry robots that can not move or turn. We can position a sentry at any position facing either north, south, east or west. Once a sentry is settled, it guards the points of interest that are infront of it. If two or more points are in the same row or column a single robot can guard them all. Unfortunately, there are also some obstacles that the robot cannot see through.

From a set of points of interest and obstacles lying on a grid, calculate the minimum number of robots needed to guard all the points. In order to guard a point of interest, a robot must be facing the direction of this point and must not be any obstacles in between.

Given the following grid, where # represents an obstacle and * a point of interest, the minimum number of robots needed is 2 (a possible position and orientation is shown using arrows for each robot). Note that this is not the actual input or output, just a figure.

```
Grid                Solution
. . . . . .         . . . . . .
. * # * . .         . * # * . .
. . # . . .         . . # . . .
. * # * . .         . ↑ # ↑ . .
```

For the following grid we need 4 robots because of the obstacles.

```
Grid                Solution
. * * . .           . → * . .
. * # * .           . ↑ # ↑ .
. # * . .           . # ↓ . .
. . * . .           . . * . .
```

INPUT

The first line of the input has an integer C representing the number of test cases that follow. Before each test case there is an empty line.

For each case, the first line has 2 integers, Y and X, representing the height and width of the grid. The next line has an integer that indicates the number of points of interest P. The following P lines will have the positions py and px of the points of interest, one point per line. The next line has an integer that indicates the number of obstacles W. The following W lines will have the positions wy and wx of an obstacle, one per line.

OUTPUT

For each test case print the minimum number of robots needed to guard all the points of interest, one per line.

CONSTRAINTS

$1 \leq C \leq 50$
$1 \leq Y, X \leq 100$
$0 \leq P \leq Y \times X$
$0 \leq W \leq Y \times X$
$0 \leq P + W \leq Y \times X$
$1 \leq px, wx \leq X$
$1 \leq py, wy \leq Y$

INPUT EXAMPLE

```
2

4 6
4
2 2
2 4
4 2
4 4
3
2 3
3 3
4 3

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

OUTPUT EXAMPLE

```
2
4
```

– How do spiders walk on water? –

Some types of spiders are able to walk on water. But spiders can have problems when walking near a waterfall.

Water in ponds or calm parts of rivers are suitable for spiders walking on water. When spiders realize water moves they jump in the opposite direction. However, the danger for spiders resides in the increasing speed of water as they move away from the area of calm waters. A spider is powerful enough to walk against the current if the water speeds is less than or equal to its jumping power P.

Waterfalls do a sharp increase in water speed, hence, whenever water just before the edge of a waterfall moves slower than the jumping power of a spider, the spider can't detect the waterfall and may fall, because he thinks that can easily return to the zone of calm waters. If a spider is walking in a zone where the speed of water is higher than its jumping power, then the spider will fall.

Usually, the speed of water as it approaches to waterfalls doesn't follow any pattern, but we have discovered that in some cases the water speed at distance m from calm waters depends linearly on the speed at $m-1$ and $m-2$. In such cases, we have measured the speed of the water at the first few meters, so the water speed at each point m up to the waterfall can be calculated.

We want to know the minimum distance to the waterfall that the spider can go and safely return.

INPUT

The input consists of several cases, one per line. Each case is defined by several integer numbers: D, P, S_0, S_1, S_2, ...

D is the distance in meters from calm waters to the waterfall ($2 < D \leq 10000$). P is the jumping power of the spider ($1 < P \leq 1000$). Remaining numbers in the line represent the speed of water as it approaches to the waterfall. S_0 is the speed water moves in the area of calm waters, S_1 is the speed at a distance of one meter from the area of calm waters, S_2 is the speed at a distance of two meters from the area of calm waters, and so on. Depending on the case this sequence can be longer, maximum is $D+1$ values. When the sequence of water speeds doesn't follow a known pattern $D+1$ values will be provided, otherwise a minimum of four numbers. All the sequences are nondecreasing sequences.

OUTPUT

The output will be an integer in a different line for each case indicating the minimum distance to the waterfall that the spider can be on water. If the jumping power of the spider is greater than the water speed just before the waterfall then the output must be: `The spider may fall!`, because the spider cannot detect the waterfall. If the jumping power of the spider is lower than the water speed at any area of the pond or river the output must be: `The spider is going to fall!`.

INPUT EXAMPLE

```
 3 3 0 1 1 1
10 2 0 1 1 2
10 3 0 1 1 2
10 4 0 1 1 2
10 5 0 1 1 2
10 6 0 1 1 2
10 7 0 1 1 2
10 8 0 1 1 2
10 9 0 1 1 2
10 3 1 1 2 2 2 3 3 3 3 4 4
10 5 1 1 2 2 2 3 3 3 3 4 4
10 5 6 6 6 6 8 8 8 11 30 41 42
10 2 0 1 1 1 1 1 1 1 1 1 1
50 200 0 3 6 15 36
```

OUTPUT EXAMPLE

```
The spider may fall!
7
6
6
5
5
5
4
4
2
The spider may fall!
The spider is going to fall!
The spider may fall!
45
```

– Shares –

You are a successful business man who uses to invest some money in the shares market. As a successful man you manage a network of well prepared ~~spies~~ assistants that can assure you the values of the shares for the next day. Each day you have a capital that you can spend in the market according to your assistants suggestions. In addition, you can only buy packs of shares from several salesmen.

Your goal is to select which packs should be bought in order to maximize the profits without exceeding the amount of capital you have.

INPUT

The first line contains the maximum capital C that you can invest ($0 < C \leq 2^{30}$). The next line has two integers, the number of total shares N ($0 < N \leq 500$) and the number of packs P ($0 < P \leq 50000$). Each one of the following N lines describe the N shares. Each line contains two integers a_i and t_i representing the current price and the expected price for the next day of the ith share ($1 \leq i \leq N$), respectively. Finally, the following P lines contain the information of the packs, one per line. For each line, the first integer R represents the number of different shares that contains this pack. Then for each share type you have two integers s_j and q_j ($1 \leq j \leq R$), where s_j is the id of the jth share and q_j is the quantity of the jth share in this pack.

OUTPUT

An integer that indicates the maximum expected profit for the next day.

INPUT EXAMPLE

```
500
4 6
10 15
8 6
20 15
12 12
3 1 6 2 7 3 8
3 3 8 1 10 2 4
3 4 10 2 5 1 10
2 1 4 2 4
1 3 2
2 4 3 2 1
```

OUTPUT EXAMPLE

```
52
```

INPUT EXAMPLE

```
200000000
5 30
2800 3500
1400 4800
2900 2800
500 3800
3300 4700
2 2 13 4 15
4 4 1 1 22 3 17 5 22
1 3 2
1 3 6
4 1 11 2 5 3 7 5 15
1 5 1
4 2 26 1 21 3 8 5 26
2 3 5 2 26
4 2 30 4 12 3 7 5 14
3 3 8 2 20 5 3
1 5 30
2 1 29 3 3
5 3 3 1 20 5 26 4 9 2 25
3 1 2 2 16 3 5
2 5 5 4 26
5 2 18 5 10 4 18 1 12 3 30
3 2 5 3 27 5 4
4 3 2 4 8 1 20 2 6
3 2 14 1 1 4 22
5 2 23 3 26 1 27 5 3 4 6
1 2 16
4 1 13 4 10 2 23 5 2
1 1 14
1 2 20
1 3 14
2 3 21 1 22
1 2 27
3 5 24 1 26 3 13
5 4 15 3 3 2 21 1 5 5 16
4 2 22 5 1 4 10 1 30
```

OUTPUT EXAMPLE

```
2168800
```

– The Moon of Valencia –

It is well known that the Moon of Valencia is magical. Everyone talks about a mystery that happens at night. People remember what time they entered the first bar, what time arrived to the hotel and how happy they arrived, but nobody remembers the bars and pubs they visited.

The Valencia hotels have hired you for developing an application that helps customers to remember. The application will inform customers with one of all the possible sequences of bars and pubs. Customers have to provide the following information: departure time and place, arrival time and place, and degree of satisfaction on arrival.

The application uses a map with the location of each bar or pub. Each bar or pub produces a different degree of satisfaction when visited. But people gets angry when walks from one place to another, that's the reason why walking between different places reduces the degree of satisfaction. The reduction considered depends on the amount of minutes people needs to get one place from another one. If the amount of minutes is not an integer the remaining seconds should be considered a portion of a minute, i.e., 30 seconds imply 0.5 minutes. People walk at a speed of 4 km/h, can stay in a bar or pub the time they like, but for getting the satisfaction must remain at least 15 minutes. People can decide not to visit a bar or pub, i.e., they can use a path from the origin to the target and to enter in a subset of all the bars or pubs reachable with the path. Entering to the departure place is optional, like entering others places. The goal grade of satisfaction is computed up to the door of the target place, without entering it. So the grade of satisfaction of the target place (bar, pub or hotel) should not be computed.

INPUT

Input consists of several test cases. Each case begins with the map description, which is followed by the list of arrivals your application have to check. The description of a map begins with the word `MAP` in capital letters followed by two integer numbers, P and M, where P is the number of places and M is the number of paths connecting two places. ($1 \leq P \leq 64$). Places and paths are described one per line. Each place is described with two coordinates (real numbers represent kilometers), its grade of satisfaction (a real number), the ID and the name. The paths connecting two places are identified by the their identifiers. Each pair of places can only be connected by one path.

Figure 1 shows the map corresponding to the first map in the example input case. It is guaranteed that no crossing paths exist.

The list of arrivals to be processed begins after a line with the word `ARRIVALS` in uppercase. Each arrival is described in a single line, including departure time, departure place, arrival time, arrival place and the grade of satisfaction on arrival, a real number.

OUTPUT

The output for each case must begin with the word `MAP` in capital letters followed by a number indicating the number of test case. The first test case is `MAP 1`, second is `MAP 2`, and so on. For each arrival proposed in the input it must appear a line in the output specifying a valid path found or the string `Impossible!` indicating that it is not possible to find a path from origin to target with the required grade of satisfaction. A path found will be valid if the absolute difference between the required grade of satisfaction and the obtained one is less than 0.1, and contains no loops, i.e. a place can't appear in the path found more than once.

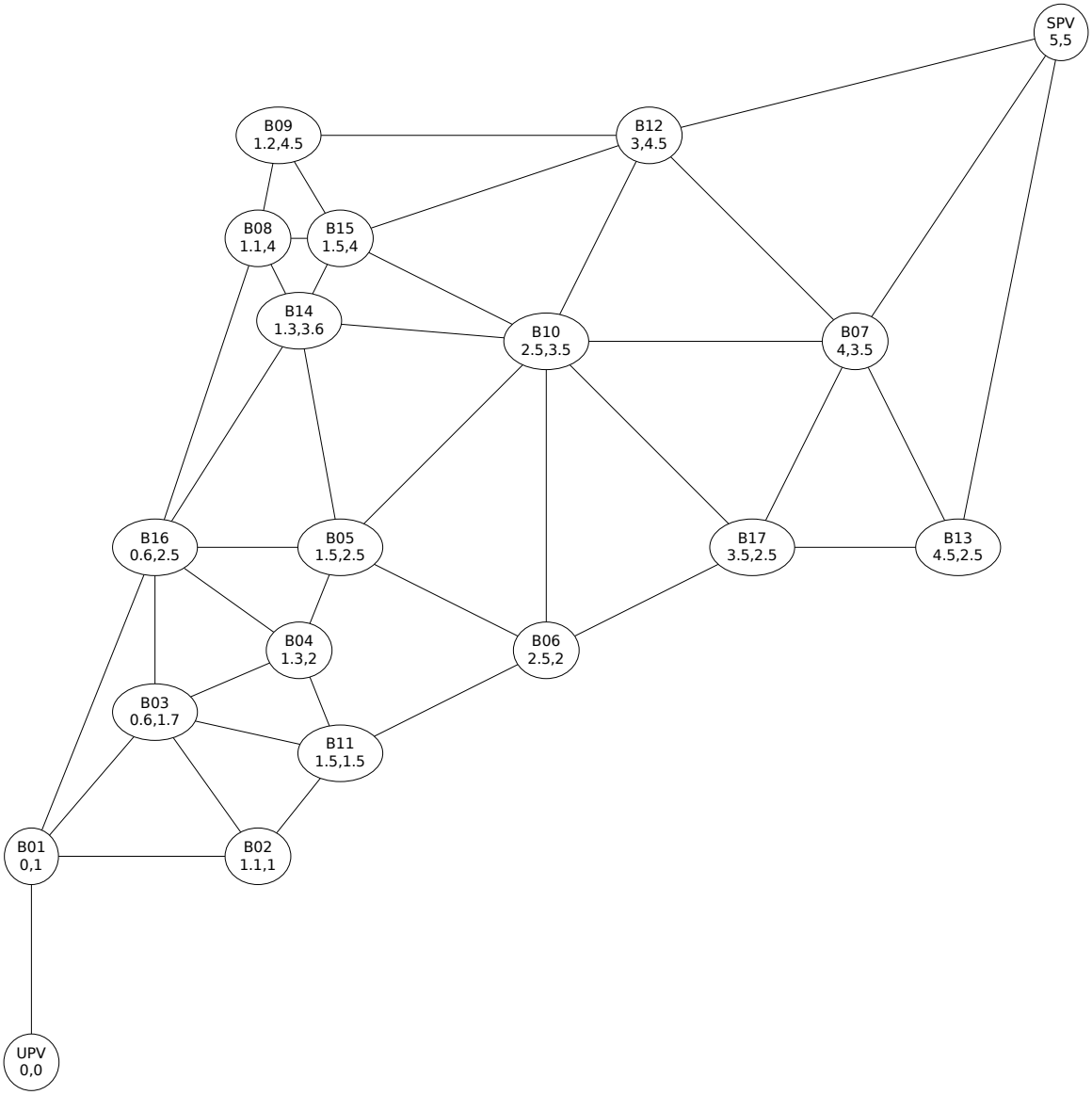


Figure 1: Representation of the first map in the input example.

Each path found must begin with the string `PATH FOUND:` in capital letters, followed by the obtained grade of satisfaction with three decimal digits and then the sequence of places from origin up to target. The ID of the unvisited places must appear preceded by the `!` sign. Notice as the ID of the target place never is preceded by this sign.

Hint: set up your solution for running as fast as possible by using this example, it should be enough for all test cases.

INPUT EXAMPLE

```
MAP 19 40
  0     0     0  UPV  Universitat Politecnica de Valencia
  5     5     0  SPV  Contest hotel
  0     1    35  B01  The Object
  1.1   1    42  B02  Opera
  0.6   1.7  33  B03  New York
  1.3   2    55  B04  Blue Note
  1.5   2.5  23  B05  The Popes
  2.5   2    13  B06  Petrol
  4     3.5  12  B07  King of Kings
  1.1   4    14  B08  O Salati
  1.2   4.5  13  B09  The Snails
  2.5   3.5  34  B10  The Earth
  1.5   1.5  55  B11  Cafe Coffee
  3     4.5  31  B12  Vermouth house
  4.5   2.5  45  B13  Jamon Session
  1.3   3.6  24  B14  Let's go to eat
  1.5   4    34  B15  I'm hungry
  0.6   2.5  53  B16  The Gecko
  3.5   2.5  43  B17  The Black Sheep
UPV B01
B01 B02
B01 B03
B01 B16
B02 B03
B02 B11
B16 B08
B16 B14
B16 B03
B03 B04
B03 B11
B04 B11
B04 B16
B04 B05
B05 B14
B08 B09
B08 B15
B08 B14
B11 B06
B14 B15
B05 B06
B05 B16
B05 B10
B15 B09
B15 B10
B09 B12
B06 B10
B06 B17
B10 B07
B10 B17
B10 B12
B10 B14
B12 B15
B12 B07
B12 SPV
B17 B07
B17 B13
B07 B13
B07 SPV
```

```
B13 SPV
ARRIVALS
23:00  UPV 03:00  SPV   9.0
23:00  UPV 03:00  SPV   8.0
23:00  UPV 03:00  SPV   7.0
23:00  UPV 03:00  SPV   6.0
23:00  UPV 03:00  SPV   5.0
23:00  UPV 03:00  SPV   4.0
23:00  UPV 03:00  SPV   3.0
23:00  UPV 03:00  SPV   2.0
23:00  UPV 03:00  SPV   1.0
23:00  UPV 03:00  SPV   0.0
23:00  UPV 03:00  SPV  -1.0
23:00  UPV 03:00  SPV  -2.0
23:00  UPV 03:00  SPV  -30.0
23:00  UPV 03:00  SPV  -40.0
23:00  B05 03:00  B10   40.0
23:00  B05 03:00  B10   30.0
23:00  B05 03:00  B10   20.0
23:00  B05 03:00  B10   10.0
23:00  B05 03:00  B10    0.0
23:00  B05 03:00  B10  -10.0
23:00  B05 03:00  B10  -20.0
23:00  B05 03:00  B10  -30.0
23:00  B05 03:00  B10  -40.0
MAP 2 1
 0  0 0 UPV Universitat Politecnica de Valencia
10 10 0 SPV Hotel Silken Puerta de Valencia
UPV SPV
ARRIVALS
23:00  UPV  1:00  SPV   9.0
23:00  UPV  1:00  SPV   8.0
```

OUTPUT EXAMPLE

```
MAP 1
PATH FOUND:    9.002   UPV  B01  B16  B04 !B11  B06 !B17 !B13 SPV
PATH FOUND:    7.929   UPV  B01  B16  B14  B08  B09 !B12 SPV
PATH FOUND:    6.973   UPV  B01  B16  B04 !B11  B06 !B10 !B07 SPV
PATH FOUND:    6.028   UPV  B01  B16  B05  B10 !B17 !B07 SPV
PATH FOUND:    4.995   UPV  B01  B16  B04 !B05 !B14 !B15 !B10  B07 SPV
PATH FOUND:    4.078   UPV  B01  B16  B08 !B15  B12  B07 SPV
PATH FOUND:    3.028   UPV  B01  B16  B05 !B10  B12 !B07 SPV
PATH FOUND:    1.929   UPV  B01  B16  B14 !B15  B08  B09 !B12 SPV
PATH FOUND:    0.912   UPV  B01  B16  B03 !B04 !B05 !B06  B17 !B13 !B07 SPV
PATH FOUND:    0.028   UPV  B01  B16  B05 !B10  B17 !B13 !B07 SPV
PATH FOUND:   -0.986   UPV  B01  B16  B08 !B09 !B15  B12 SPV
PATH FOUND:   -1.973   UPV  B01  B16  B03 !B04 !B05  B10 !B17 !B07 SPV
PATH FOUND:  -29.953   UPV  B01  B03 !B16  B05 !B10 !B14  B15 !B12 SPV
PATH FOUND:  -39.913   UPV  B01  B03 !B16  B08 !B09 !B15  B12 !B07 SPV
PATH FOUND:   40.069   B05  B16  B14 !B08  B09 !B15 B10
PATH FOUND:   30.012   B05  B16  B08 !B15 B10
PATH FOUND:   19.979  !B05  B04 !B03  B02 !B01  B16 !B08 !B09 !B15 B10
PATH FOUND:   10.004   B05  B14  B08 !B15 !B09  B12 B10
PATH FOUND:    0.004  !B05  B14  B08 !B15  B09  B12 B10
PATH FOUND:   -9.966   B05 !B14 !B15  B12 B10
PATH FOUND:  -20.012   B05 !B06 !B17  B07  B12  B15 !B14 B10
PATH FOUND:  -30.012  !B05  B06 !B17  B07  B12  B15 !B14 B10
PATH FOUND:  -40.018  !B05 !B04 !B16  B14 !B15 B10
MAP 2
Impossible!
Impossible!
```

– Countdown –

The "Countdown" TV show has a part that consists of obtaining a number by combining six different numbers using the basic mathematical operations: addition, subtraction, product and division. The basic rules for the game are:

- The contestant selects six of twenty-four shuffled tiles. The tiles are arranged into two groups: four "large numbers" (25, 50, 75 and 100) and the remainder "small numbers", which comprise two each of the numbers 1 to 10. Hence the tiles have the values {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100}.
- The contestant chooses how many large numbers are in the selection; anywhere from none.
- The contestants then have thirty seconds to get a number as close to the target as possible by combining the six selected numbers using addition, subtraction, multiplication and division.
- Not all numbers need to be used.
- A number can be used as many times as it appears.
- Fractions are not allowed, only positive integers may be used at any stage of the calculation.

Example:

- Contestant requests two large numbers and four small numbers.
- Selection is: 75 50 2 3 8 7
- Randomly generated target is: 812
- Contestant declares result: 813
- Contestant gives details: 75 + 50 = 125; 125 - 8 = 117; 117 × 7 = 819; 3 × 2 = 6; 819 - 6 = 813
- Expert notes: 50 + 8 = 58; 7 × 2 = 14; 14 × 58 = 812

Your task is to write a program that calculates the best sequence of operations that lead to the target number T. If there is no way to get T, give the closest solution.

INPUT

The input consists of several cases, one per line. The first line indicates the number of cases C to process ($1 \leq C \leq 50$). Each of the following C lines contains six natural numbers from the set {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100} and another natural number T ($1 \leq T \leq 999$) that indicates the target.

OUTPUT

The output for each case will be a set of lines with the following format:

- First line: `Target:` number T
- n lines: sequence of operations, the format is $operand_2$ *operator* $operand_2$ = *result*
- Last line: `Best approx:` number obtained
- Blank line

See example for a better understanding. If there is more than one best approximation, all of them will be considered valid. The sequence of operations should be valid, you should never use a value before you obtain it. It is OK to print more operations than needed as long as they are valid. Note that all the numbers and operators must be separated by at least one space.

INPUT EXAMPLE

```
3
1 75 100 5 3 25 25
100 100 100 100 100 75 345
1 3 1 10 100 75 345
```

OUTPUT EXAMPLE

```
Target: 25
Best approx: 25

Target: 345
100 + 100 = 200
75 + 100 = 175
200 * 175 = 35000
35000 - 100 = 34900
34900 / 100 = 349
Best approx: 349

Target: 345
100 - 10 = 90
3 * 90 = 270
270 + 75 = 345
Best approx: 345
```