Submissions

Alien Numbers

40pt	Not attempted **320/432 users** correct (74%)
80pt	Not attempted **271/338 users** correct (80%)

Always Turn Left

40pt	Not attempted **108/135 users** correct (80%)
80pt	Not attempted **96/114 users** correct (84%)

Egg Drop

40pt	Not attempted **56/82 users** correct (68%)
80pt	Not attempted **26/53 users** correct (49%)

Shopping Plan

40pt	Not attempted **43/67 users** correct (64%)
80pt	Not attempted **16/52 users** correct (31%)

Top Scores

sclo	480
jdmetz	480
lordmonsoon	480
ardiankp	480
krijgertje	480
ilyakor	400
Edu	400
Jonick	400
zibada	400
gpascale	400

Problem A. Alien Numbers

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 40 points	Solve A-small
Large input 80 points	Solve A-large

Problem

The decimal numeral system is composed of ten digits, which we represent as "0123456789" (the digits in a system are written from lowest to highest). Imagine you have discovered an alien numeral system composed of some number of digits, which may or may not be the same as those used in decimal. For example, if the alien numeral system were represented as "oF8", then the numbers one through ten would be (F, 8, Fo, FF, F8, 8o, 8F, 88, Foo, FoF). We would like to be able to work with numbers in arbitrary alien systems. More generally, we want to be able to convert an arbitrary number that's written in one alien system into a second alien system.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. Each case is a line formatted as

```
alien_number source_language target_language
```

Each language will be represented by a list of its digits, ordered from lowest to highest value. No digit will be repeated in any representation, all digits in the alien number will be present in the source language, and the first digit of the alien number will not be the lowest valued digit of the source language (in other words, the alien numbers have no leading zeroes). Each digit will either be a number 0-9, an uppercase or lowercase letter, or one of the following symbols `!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~`

Output

For each test case, output one line containing "Case #**x**: " followed by the alien number translated from the source language to the target language.

Limits

1 ≤ **N** ≤ 100.

Small dataset

1 ≤ num digits in **alien_number** ≤ 4,
2 ≤ num digits in **source_language** ≤ 16,
2 ≤ num digits in **target_language** ≤ 16.

Large dataset

1 ≤ **alien_number** (in decimal) ≤ 1000000000,
2 ≤ num digits in **source_language** ≤ 94,
2 ≤ num digits in **target_language** ≤ 94.

Sample

```
Input                      Output
4                          Case #1: Foo
9 0123456789 oF8           Case #2: 9
Foo oF8 0123456789         Case #3: 10011
13 0123456789abcdef 01     Case #4: JAM!
CODE O!CDE? A?JM!.
```

Problem B. **Always Turn Left**

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 40 points — Solve B-small

Large input 80 points — Solve B-large

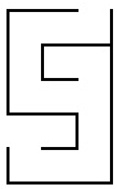
Problem

You find yourself standing outside of a perfect maze. A maze is defined as "perfect" if it meets the following conditions:

1. It is a rectangular grid of rooms, **R** rows by **C** columns.
2. There are exactly two openings on the outside of the maze: the entrance and the exit. The entrance is always on the north wall, while the exit could be on any wall.
3. There is exactly one path between any two rooms in the maze (that is, exactly one path that does not involve backtracking).

You decide to solve the perfect maze using the "always turn left" algorithm, which states that you take the leftmost fork at every opportunity. If you hit a dead end, you turn right twice (180 degrees clockwise) and continue. (If you were to stick out your left arm and touch the wall while following this algorithm, you'd solve the maze without ever breaking contact with the wall.) Once you finish the maze, you decide to go the extra step and solve it again (still always turning left), but starting at the exit and finishing at the entrance.

The path you take through the maze can be described with three characters: 'W' means to walk forward into the next room, 'L' means to turn left (or counterclockwise) 90 degrees, and 'R' means to turn right (or clockwise) 90 degrees. You begin outside the maze, immediately adjacent to the entrance, facing the maze. You finish when you have stepped outside the maze through the exit. For example, if the entrance is on the north and the exit is on the west, your path through the following maze would be `WRWWLWWLWWLWLWRRWRWWWRWWRWLW`:



If the entrance and exit were reversed such that you began outside the west wall and finished out the north wall, your path would be `WWRRWLWLWWLWWLWWRWWRWWLW`. Given your two paths through the maze (entrance to exit and exit to entrance), your code should return a description of the maze.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. Each case is a line formatted as

```
entrance_to_exit exit_to_entrance
```

All paths will be at least two characters long, consist only of the characters 'W', 'L', and 'R', and begin and end with 'W'.

Output

For each test case, output one line containing "Case #**x**:" by itself. The next **R** lines give a description of the **R** by **C** maze. There should be **C** characters in each line, representing which directions it is possible to walk from that room. Refer to the following legend:

Character	Can walk north?	Can walk south?	Can walk west?	Can walk east?
1	Yes	No	No	No
2	No	Yes	No	No
3	Yes	Yes	No	No
4	No	No	Yes	No
5	Yes	No	Yes	No
6	No	Yes	Yes	No
7	Yes	Yes	Yes	No

8	No	No	No	Yes
9	Yes	No	No	Yes
a	No	Yes	No	Yes
b	Yes	Yes	No	Yes
c	No	No	Yes	Yes
d	Yes	No	Yes	Yes
e	No	Yes	Yes	Yes
f	Yes	Yes	Yes	Yes

Limits

1 ≤ **N** ≤ 100.

Small dataset

2 ≤ len(entrance_to_exit) ≤ 100,
2 ≤ len(exit_to_entrance) ≤ 100.

Large dataset

2 ≤ len(entrance_to_exit) ≤ 10000,
2 ≤ len(exit_to_entrance) ≤ 10000.

Sample

Input
```
2
WRWWLWWLWWLWLWRRWRWWWRWWRWLW WWRRWLWLWWLWWLWWRWWRWWLW
WW WW
```

Output
```
Case #1:
ac5
386
9c7
e43
9c5
Case #2:
3
```

Problem C. Egg Drop

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 40 points — Solve C-small

Large input 80 points — Solve C-large

Problem

Imagine that you are in a building with **F** floors (starting at floor 1, the lowest floor), and you have a large number of identical eggs, each in its own identical protective container. For each floor in the building, you want to know whether or not an egg dropped from that floor will break. If an egg breaks when dropped from floor *i*, then all eggs are guaranteed to break when dropped from any floor *j* ≥ *i*. Likewise, if an egg doesn't break when dropped from floor *i*, then all eggs are guaranteed to never break when dropped from any floor *j* ≤ *i*.

We can define *Solvable(F, D, B)* to be true if and only if there exists an algorithm to determine whether or not an egg will break when dropped from any floor of a building with **F** floors, with the following restrictions: you may drop a maximum of **D** eggs (one at a time, from any floors of your choosing), and you may break a maximum of **B** eggs. You can assume you have at least **D** eggs in your possession.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. Each case is a line formatted as:

```
F D B
```

Solvable(F, D, B) is guaranteed to be true for all input cases.

Output

For each test case, output one line containing "Case #**x**: " followed by three space-separated integers: F_{max}, D_{min}, and B_{min}. The definitions are as follows:

- F_{max} is defined as the largest value of **F'** such that *Solvable(**F'**, D, B)* is true, or -1 if this value would be greater than or equal to 2^{32} (4294967296).
  (In other words, F_{max} = -1 if and only if *Solvable(2^{32}, D, B)* is true.)
- D_{min} is defined as the smallest value of **D'** such that *Solvable(F, **D'**, B)* is true.
- B_{min} is defined as the smallest value of **B'** such that *Solvable(F, D, **B'**)* is true.

Limits

1 ≤ **N** ≤ 100.

Small dataset

1 ≤ **F** ≤ 100,
1 ≤ **D** ≤ 100,
1 ≤ **B** ≤ 100.

Large dataset

1 ≤ **F** ≤ 2000000000,
1 ≤ **D** ≤ 2000000000,
1 ≤ **B** ≤ 2000000000.

Sample

Input	Output
2 3 3 3 7 5 3	Case #1: 7 2 1 Case #2: 25 3 2

Problem D. Shopping Plan

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 40 points — Solve D-small

Large input 80 points — Solve D-large

Problem

You have a list of items you need to buy today, and you know the locations (represented as points on a cartesian grid) of a few stores in the area. You also know which of these stores are selling each item on your list, and at what price each store sells it. Given the price of gas, what is the minimum amount you need to spend in order to buy all the items on your shopping list and then drive back home? You start and end the journey at your house, which is located at (0,0).

To make matters interesting, some of the items on your list may be perishable. Whenever you make a purchase that includes one or more perishable items, you cannot drive to another store without first stopping back at your house. Every item on your shopping list is guaranteed to be sold by at least one store, so the trip will always be possible.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. Each case starts with a line formatted as

```
num_items num_stores price_of_gas
```

The next line contains the **num_items** items on your shopping list. The items will be space separated, and each item will consist of only lowercase letters. If an item is perishable, its name will be followed by a single exclamation point. There will be no duplicate items on your list. The next **num_stores** lines will each be formatted as

```
x_pos y_pos item1:price1 item2:price2 ...
```

Each of these lines gives the location of one store, along with the items available at that store and their corresponding prices. Only items which are on your shopping list will appear in these lists. Perishable items will not end with exclamation points on these lists. No item will be repeated in a store's list. Each store will offer at least one item for sale. No two stores will be at the same location, and no store will be located at (0,0).

Output

For each test case, output one line containing "Case #**x**: " followed by the minimum possible cost of the trip, rounded to seven decimal places. Don't forget about **price_of_gas**, which is the amount of money you must spend per unit distance that you drive.

Limits

1 ≤ **N** ≤ 100,
0 ≤ **price_of_gas** ≤ 1000,
-1000 ≤ **x_pos** ≤ 1000,
-1000 ≤ **y_pos** ≤ 1000,
1 ≤ price of each item ≤ 1000.

Small dataset

1 ≤ **num_items** ≤ 5,
1 ≤ **num_stores** ≤ 10.

Large dataset

1 ≤ **num_items** ≤ 15,
1 ≤ **num_stores** ≤ 50.

Sample

Input	Output
2 1 2 10 cookies	Case #1: 400.0000000 Case #2: 519.2920690

```
0 2 cookies:400
4 0 cookies:320
3 3 5
cookies milk! cereal
0 2 cookies:360 cereal:110
4 0 cereal:90 milk:150
-3 -3 milk:200 cookies:200
```

Code Jam Beta 2008

A. Triangle Trilemma
B. The Price is Wrong
C. Random Route
D. Hexagon Game

Questions asked

Submissions

Triangle Trilemma
10pt Not attempted **244/318 users** correct (77%)
10pt Not attempted **200/260 users** correct (77%)

The Price is Wrong
15pt Not attempted **110/175 users** correct (63%)
25pt Not attempted **67/96 users** correct (70%)

Random Route
30pt Not attempted **42/76 users** correct (55%)
30pt Not attempted **38/51 users** correct (75%)

Hexagon Game
25pt Not attempted **8/29 users** correct (28%)
45pt Not attempted **6/15 users** correct (40%)

Top Scores

malcin	190
marek.cygan	190
SnapDragon	165
ardiankp	145
Astein	130
rem	130
RAVEman	120
yuhch123	120
Lovro	120
lukasP	120

Problem A. Triangle Trilemma

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 10 points — Solve A-small

Large input 10 points — Solve A-large

Problem

You're interested in writing a program to classify triangles. Triangles can be classified according to their internal angles. If one of the internal angles is exactly 90 degrees, then that triangle is known as a "right" triangle. If one of the internal angles is greater than 90 degrees, that triangle is known as an "obtuse" triangle. Otherwise, all the internal angles are less than 90 degrees and the triangle is known as an "acute" triangle.

Triangles can also be classified according to the relative lengths of their sides. In a "scalene" triangle, all three sides have different lengths. In an "isosceles" triangle, two of the sides are of equal length. (If all three sides have the same length, the triangle is known as an "equilateral" triangle, but you can ignore this case since there will be no equilateral triangles in the input data.)

Your program must determine, for each set of three points, whether or not those points form a triangle. If the three points are not distinct, or the three points are collinear, then those points do not form a valid triangle. (Another way is to calculate the area of the triangle; valid triangles must have non-zero area.) Otherwise, your program will classify the triangle as one of "acute", "obtuse", or "right", and one of "isosceles" or "scalene".

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. Each case is a line formatted as

```
x1 y1 x2 y2 x3 y3
```

Output

For each test case, output one line containing "Case #**x**: " followed by one of these strings:

- `isosceles acute triangle`
- `isosceles obtuse triangle`
- `isosceles right triangle`
- `scalene acute triangle`
- `scalene obtuse triangle`
- `scalene right triangle`
- `not a triangle`

Limits

1 ≤ **N** ≤ 100,
x1, **y1**, **x2**, **y2**, **x3**, **y3** will be integers.

Small dataset

0 ≤ **x1**, **y1**, **x2**, **y2**, **x3**, **y3** ≤ 9

Large dataset

-1000 ≤ **x1**, **y1**, **x2**, **y2**, **x3**, **y3** ≤ 1000

Sample

```
Input           Output
8               Case #1: isosceles obtuse triangle
0 0 0 4 1 2     Case #2: scalene acute triangle
1 1 1 4 3 2     Case #3: isosceles acute triangle
2 2 2 4 4 3     Case #4: scalene right triangle
3 3 3 4 5 3     Case #5: scalene obtuse triangle
4 4 4 5 5 6     Case #6: isosceles right triangle
5 5 5 6 6 5     Case #7: not a triangle
6 6 6 7 6 8     Case #8: not a triangle
7 7 7 7 7 7
```

Problem B. The Price is Wrong

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 15 points	Solve B-small
Large input 25 points	Solve B-large

Problem

You're playing a game in which you try to guess the correct retail price of various products for sale. After guessing the price of each product in a list, you are shown the same list of products sorted by their actual prices, from least to most expensive. (No two products cost the same amount.) Based on this ordering, you are given a single chance to change one or more of your guesses.

Your program should output the smallest set of products such that, if you change your prices for those products, the ordering of your guesses will be consistent with the correct ordering of the product list. The products in the returned set should be listed in alphabetical order. If there are multiple smallest sets, output the set which occurs first lexicographically.

For example, assume these are your initial guesses:

- `code = $20`
- `jam = $15`
- `foo = $40`
- `bar = $30`
- `google = $60`

If the correct ordering is `code jam foo bar google`, then you would need to change two of your prices in order to match the correct ordering. You might change one guess to read `jam = $30` and another guess to read `bar = $50`, which would match the correct ordering and produce the output set `bar jam`. However, the output set `bar code` comes before `bar jam` lexicographically, and you can match the correct ordering by changing your guesses for these items as well.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. Each case consists of two lines. The first line contains the list of products, space separated, sorted from least to most expensive. Each product's name will consist of only lowercase letters and underscores. There will be no duplicate products in the list. The second line contains your initial guesses for each product in the list, respectively. All guesses will be integers between 1 and 100, inclusive. The number of guesses will be equal to the number of products, and no two guesses will be the same. The guesses will be space separated. (Note that although the initial guesses all happen to be integers, you are allowed to change your guesses to any amounts, not just integers.)

Output

For each test case, output one line containing "Case #**x**: " followed by the set of products for which you must change your prices. The products should be in alphabetical order and space separated. The constraints guarantee that you will need to change the price of at least one product for each test case.

Limits

$1 \le \mathbf{N} \le 100$.

Small dataset

$2 \le$ number of products in list ≤ 8

Large dataset

$2 \le$ number of products in list ≤ 64

Sample

Input	Output
1 code jam foo bar google 20 15 40 30 60	Case #1: bar code

Code Jam Beta 2008

A. Triangle Trilemma

B. The Price is Wrong

C. Random Route

D. Hexagon Game

Questions asked

Submissions

Triangle Trilemma

10pt	Not attempted **244/318 users** correct (77%)
10pt	Not attempted **200/260 users** correct (77%)

The Price is Wrong

15pt	Not attempted **110/175 users** correct (63%)
25pt	Not attempted **67/96 users** correct (70%)

Random Route

30pt	Not attempted **42/76 users** correct (55%)
30pt	Not attempted **38/51 users** correct (75%)

Hexagon Game

25pt	Not attempted **8/29 users** correct (28%)
45pt	Not attempted **6/15 users** correct (40%)

Top Scores

malcin	190
marek.cygan	190
SnapDragon	165
ardiankp	145
Astein	130
rem	130
RAVEman	120
yuhch123	120
Lovro	120
lukasP	120

Problem C. Random Route

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 30 points	Solve C-small
Large input 30 points	Solve C-large

Problem

Where do you want to go today, and how do you want to get there? You decide to choose the answer to both questions at random.

You will be given a list of roads. Each road connects one city to another city (all roads are one-way), and each road takes a certain amount of time to drive. You will also be given a starting city. Consider all cities that you're able to drive to, not including your starting city, and choose one of them at random with uniform probability to be your destination. Now consider every fastest route from your starting city to your destination city, and choose one of these routes at random with uniform probability. This will be the route on which you end up driving.

For each road in the input, your program must output the probability that you will end up driving on that road, given the behavior outlined above.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. Each case begins with a line formatted as

```
num_roads starting_city
```

This will be followed by **num_roads** lines, each formatted as

```
city1 city2 time
```

Each line represents a one-way road that starts at `city1` and ends at `city2`, and takes `time` hours to drive. All cities will be formatted as strings consisting of only lowercase letters and underscores. For each road, `city1` will not be equal to `city2`, and `time` will be an integer between 1 and 100000, inclusive. The starting city is guaranteed to appear as `city1` on at least one road; therefore, there will always be at least one possible destination (and at least one shortest route to that destination).

Output

For each test case, output one line containing "Case #**x**: " followed by the probability that you will drive on each road, in the same order that the roads were listed in the input. Probabilities should be space separated and formatted so there are exactly seven digits after the decimal point. Each probability must be within a distance of 1e-6 from the correct answer to be judged as correct.

Limits

1 ≤ **N** ≤ 100.

Small dataset

2 ≤ **num_roads** ≤ 25.

Large dataset

2 ≤ **num_roads** ≤ 50.

Sample

Input	Output
1 5 san_francisco san_francisco los_angeles 6 los_angeles san_diego 2 san_francisco san_diego 8 los_angeles san_diego 2 san_francisco los_angeles 6	Case #1: 0.4500000 0.2000000

Submissions

Triangle Trilemma

10pt	Not attempted **244/318 users** correct (77%)
10pt	Not attempted **200/260 users** correct (77%)

The Price is Wrong

15pt	Not attempted **110/175 users** correct (63%)
25pt	Not attempted **67/96 users** correct (70%)

Random Route

30pt	Not attempted **42/76 users** correct (55%)
30pt	Not attempted **38/51 users** correct (75%)

Hexagon Game

25pt	Not attempted **8/29 users** correct (28%)
45pt	Not attempted **6/15 users** correct (40%)

Top Scores

malcin	190
marek.cygan	190
SnapDragon	165
ardiankp	145
Astein	130
rem	130
RAVEman	120
yuhch123	120
Lovro	120
lukasP	120

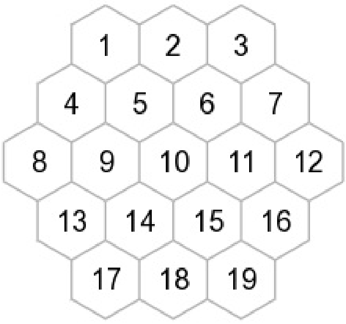
Problem D. Hexagon Game

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 25 points	Solve D-small
Large input 45 points	Solve D-large

Problem

You are playing a game on a hexagonal board of size **S**. The middle row is composed of **S** hexagons, and the top and bottom rows each have (**S** + 1) / 2 hexagons. (**S** will be odd.) The hexagons are numbered starting with 1 in the upper left, and increasing left-to-right and top-to-bottom. Here is a hexagonal board of size **S**=5:



The game starts with **S** checkers on the board. Multiple checkers might start in the same position. Each checker also has an associated integer value between 0 and 50, inclusive. A turn consists of choosing a checker and moving it to an adjacent position, which increments your score by the value of that checker. Checkers cannot move off the board. Each position can contain any number of checkers at the same time.

The game ends when all the checkers are lined up in a straight row, with exactly one checker per hexagon. There are three possible ending configurations on any board. For **S**=5, the game will end when checkers are in positions (1, 5, 10, 15, 19), or in positions (3, 6, 10, 14, 17), or in positions (8, 9, 10, 11, 12). Your program must output the smallest possible score of a finished game.

For example, assume the checkers start in positions (1, 2, 5, 15, 19). The checker in position 1 has a value of 1, the checkers in positions 2 and 5 have values of 3, and the checkers in positions 15 and 19 have values of 0. You could move the checker from position 1 into position 5 and then position 10. Both of these moves add one point to your score. Then you could move the checker from position 2 into position 1, adding three points to your score. This game would end with a score of 5, which is the lowest possible score for this starting configuration.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. Each case consists of two lines. The first line contains the starting positions of the checkers, space separated. The second line contains the values of each checker, respectively, space separated.

Output

For each test case, output one line containing "Case #**x**: " followed by the minimum possible score of a finished game.

Limits

1 ≤ **N** ≤ 100.

Small dataset

3 ≤ **S** ≤ 15; **S** is odd.

Large dataset

3 ≤ **S** ≤ 75; **S** is odd.

Sample

Input	Output

```
1              Case #1: 5
1 2 5 15 19
1 3 3 0 0
```

Problem A. Old Magician

Practice Contest

A. Old Magician

B. Square Fields

C. Cycles

Questions asked 4

Submissions

Old Magician

5pt Not attempted **203/214 users** correct (95%)

10pt Not attempted **193/198 users** correct (97%)

Square Fields

10pt Not attempted **146/157 users** correct (93%)

25pt Not attempted **107/128 users** correct (84%)

Cycles

15pt Not attempted **126/146 users** correct (86%)

35pt Not attempted **20/41 users** correct (49%)

Top Scores

gawry	100
bmerry	100
Olexiy	100
ACRush	100
ardiankp	100
gepa	100
natalia	100
Alexus	100
almelv	100
OpenGL	100

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 5 points	Solve A-small
Large input 10 points	Solve A-large

Problem

A magician does the following magic trick. He puts W white balls and B black balls in his hat and asks someone from the audience, say Bob, to remove pairs of balls in whatever order Bob would desire. After removing a pair of balls, Bob is asked to place a white ball back into the hat if they are the same color. Otherwise he is asked to place a black ball into the hat.

When Bob is left with only one ball in the hat, he asks the magician what color the last ball is. Needless to say, the magician can't see the order by which Bob does the replacements.

The problem is that the magician, like most magicians, is old and sometimes forgets how to do the trick. Being the kind person you are, you are going to help the magician.

For each pair of numbers (W,B) you are asked to output one of the following:

- "WHITE" - if the last ball in the hat will be white for sure.
- "BLACK" - if the last ball in the hat will be black for sure.
- "UNKNOWN" - if you can't be sure of the last ball's color.

Input

The first line of the input file contains the number of cases, **N**. **N** test cases follow.

Each case contains **W** and **B** on a line separated by a space.

Output

For each input case, you should output:

```
Case #X: Y
```

where **X** is the number of the test case and **Y** is either "WHITE", "BLACK" or "UNKNOWN" as explained above. (quotes for clarity)

Limits

$0 < \mathbf{N} \leq 1000$
$\mathbf{W} + \mathbf{B} > 0$

Small dataset

$0 \leq \mathbf{W} \leq 1000$
$0 \leq \mathbf{B} \leq 1000$

Large dataset

$0 \leq \mathbf{W} \leq 10^9$
$0 \leq \mathbf{B} \leq 10^9$

Sample

```
Input     Output
2         Case #1: BLACK
3 1       Case #2: WHITE
3 6
```

Top Scores

gawry	100
bmerry	100
Olexiy	100
ACRush	100
ardiankp	100
gepa	100
natalia	100
Alexus	100
almelv	100
OpenGL	100

Problem B. Square Fields

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 10 points	Solve B-small
Large input 25 points	Solve B-large

Problem

You are given **n** points in the plane. You are asked to cover these points with **k** squares.

The squares must all be the same size, and their edges must all be parallel to the coordinate axes.

A point is covered by a square if it lies inside the square, or on an edge of the square.

Squares can overlap.

Find the minimum length for the squares' edges such that you can cover the **n** points with **k** squares.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. The first line of each test contains two positive integers **n** and **k**. Each of the next **n** lines contains a point as two integers separated by exactly one space. No point will occur more than once within a test case.

Output

For each test case, you should output one line containing "Case #**X**: **Y**" (quotes for clarity), where **X** is the number of the test case, starting from 1, and **Y** is the minimum length for the squares' edges for that test case.

Limits

The points' coordinates are non-negative integers smaller than 64000.

$1 \le \mathbf{N} \le 10$

Small dataset

$1 \le \mathbf{k} < \mathbf{n} \le 7$

Large dataset

$1 \le \mathbf{k} < \mathbf{n} \le 15$

Sample

Input	Output

`2`
`5 2`
`1 1`
`2 2`
`3 3`
`6 6`
`7 8`
`3 2`
`3 3`
`3 6`
`6 9`
	`Case #1: 2`
	`Case #2: 3`

Submissions

Old Magician
5pt Not attempted **203/214 users** correct (95%)
10pt Not attempted **193/198 users** correct (97%)

Square Fields
10pt Not attempted **146/157 users** correct (93%)
25pt Not attempted **107/128 users** correct (84%)

Cycles
15pt Not attempted **126/146 users** correct (86%)
35pt Not attempted **20/41 users** correct (49%)

Top Scores

gawry	100
bmerry	100
Olexiy	100
ACRush	100
ardiankp	100
gepa	100
natalia	100
Alexus	100
almelv	100
OpenGL	100

Problem C. **Cycles**

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 15 points	Solve C-small
Large input 35 points	Solve C-large

Problem

You are given a complete undirected graph with **n** nodes numbered from 1 to **n**. You are also given **k** *forbidden* edges in this graph.

You are asked to find the number of Hamiltonian cycles in this graph that don't use any of the given **k** edges. A Hamiltonian cycle is a cycle that visits each vertex exactly once. A cycle that contains the same *edges* is only counted once. For example, cycles 1 2 3 4 1 and 1 4 3 2 1 and 2 3 4 1 2 are all the same, but 1 3 2 4 1 is different.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. The first line of each test case contains two integers, **n** and **k**. The next **k** lines contain two integers each, representing the vertices of a forbidden edge. There will be no self-edges and no repeated edges.

Output

For each test case, output one line containing "Case #**X**: **Y**", where **X** is the case number (starting from 1) and **Y** is the number of Hamiltonian cycles that do not include any of those **k** edges. Print your answer modulo 9901.

Limits

1 ≤ **N** ≤ 10,
0 ≤ **k** ≤ 15.

Small dataset

3 ≤ **n** ≤ 10

Large dataset

3 ≤ **n** ≤ 300

Sample

Input	Output
2 4 1 1 2 8 4 1 2 2 3 4 5 5 6	Case #1: 1 Case #2: 660

In the first sample input, there is only one cycle: 1 3 2 4 1.