Problem A: $\textbf{Con} + \cfrac{\textbf{tin}}{\textbf{ued} + \cfrac{\textbf{Frac}}{\textbf{tions}}}$

The (simple) continued fraction representation of a real number r is an expression obtained by an iterative process of representing r as a sum of its integer part and the reciprocal of another number, then writing this other number as the sum of its integer part and another reciprocal, and so on. In other words, a continued fraction representation of r is of the form

$$r = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}}$$

where $a_0, a_1, a_2, \ldots$ are integers and $a_1, a_2, \ldots > 0$. We call the a_i-values *partial quotients*. For example, in the continued fraction representation of 5.4 the partial quotients are $a_0 = 5, a_1 = 2$, and $a_2 = 2$. This representation of a real number has several applications in theory and practice.

While irrational numbers like $\sqrt{2}$ require an infinite set of partial quotients, any rational number can be written as a continued fraction with a unique, finite set of partial quotients (where the last partial quotient is never 1 in order to preserve uniqueness). Given two rational numbers in continued fraction representation, your task is to perform the four elementary arithmetic operations on these numbers and display the result in continued fraction representation.

Input

Time Limit: 3 secs, No. of Test Cases: 500, Input File Size 13.1K

Each test case consists of three lines. The first line contains two integers n_1 and n_2, $1 \leq n_i \leq 9$ specifying the number of partial quotients of two rational numbers r_1 and r_2. The second line contains the partial quotients of r_1 and the third line contains the partial quotients of r_2. The partial quotients satisfy $|a_0| \leq 10$ and $0 < a_i \leq 10$, the last partial quotient will never be 1, and r_2 is non-zero. A line containing two 0's will terminate input.

Output

For each test case, display the case number followed by the continued fraction representation of $r_1 + r_2$, $r_1 - r_2$, $r_1 \times r_2$, and r_1 / r_2 in order, each on a separate line. Use 64-bit integers for all of your calculations (`long long` in C++ and `long` in Java).

Sample Input

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

Sample Output

```
Case 1:
11
0 5
30 4 6
1 27
```

Problem B: A Cure for the Common Code

You've been tasked with relaying coded messages to your fellow resistance fighters. Each coded message is a sequence of lower-case letters that you furtively scrawl on monuments in the dead of night.

Since you're writing these messages by hand, the longer the message, the greater the likelihood of being caught by the evil empire while writing. Because of this you decide it would be worthwhile to come up with a simple encoding that might allow for shorter messages. After thinking about it for a while, you decide to use integers and parentheses to indicate repetition of substrings when doing so shortens the number of characters you need to write. For example, the 10 character string

abcbcbcbca

could be more briefly written as the 7 character string

a4(bc)a

If a single letter is being repeated, parentheses are not needed. Also, repetitions may themselves be repeated, so you can write the 20 character string

abbbcdcdcdabbbcdcdcd

as the 11 character string

2(a3b3(cd))

and so forth.

Input Time Limit: 5 secs, No. of Test Cases: 39, Input File Size 2.95K

Each test case consists of a single line containing a string of lower-case letters of length ≤ 500. A line containing a single 0 will terminate the input.

Output

For each test case, output the number of characters needed for a minimal encoding of the string.

Sample Input

```
abcbcbcbca
abbbcdcdcdabbbcdcdcd
0
```

Sample Output

```
Case 1: 7
Case 2: 11
```

Problem C: Domiyahtzee!

Yahtzee is a well known dice game in which players get points for various combinations of five dice rolls. Some of the combinations and their payoffs are shown Table 1 (note: the payoffs for the first two are slightly different than in the official rules of Yahtzee):

Combination	Description	Points
3-of-a-Kind	3 dice showing same face	Sum of the three dice
4-of-a-Kind	4 dice showing same face	Sum of the four dice
Full House	3-of-a-Kind and a pair	25
Small Straight	4 consecutive values on any 4 dice	30
Large Straight	5 consecutive values	40
Yahtzee	5 dice showing same face	first Yahtzee - 50 subsequent Yahtzees - 100 each

Table 1

Domiyahtzee! is a well known version of Yahtzee which we just invented. It uses a standard set of 21 dominoes containing all possible combinations of the numbers 1 through 6 – (1,1), (1,2), ..., (1,6), (2,2), (2,3), ..., (6,6). The game is played as follows: you are given a 5×5 grid which is filled with 12 of the 21 dominoes along with a value between 1 and 6 placed in a random square. An example grid is shown below – the "4" in the fourth row and column is the lone "singleton" value:

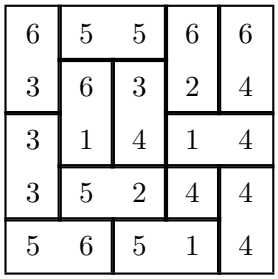


Figure 1

You score points in Domiyahtzee! for each combination in Table 1 found in any row, column or long diagonal. The grid above would score for the Full House in row 1, the Small Straight in row 4, the 3-of-a-Kind in column 1, the Small Straight in column 3, the 4-of-a-Kind in column 5 and the Full House in the first long diagonal for a total score of $25 + 30 + 9 + 30 + 16 + 25 = 135$. However, you can attempt to improve your score by replacing any one domino on the grid with any of the remaining 9 unused dominoes. For example, if you were to replace the (5,5) in the first row with the unused (6,6) domino, the grid would now score 50 (for the Yahtzee in row 1) + 30 + 9 + 18 (for the new 3-of-a-Kind in column 2) + 40 (for the new Large Straight in column 3) + 16 + 25 = 188. The object of the game, of course, is to find the replacement which maximizes your score. In the above example, replacing the (5,6) in row 5 with a (3,2) leads to the highest scoring grid.

Input Time Limit: 3 secs, No. of Test Cases: 51, Input File Size 3.88K

The input file starts with an integer n indicating the number of test cases in the file. Each test case consists of 13 domino specifications of the form `H` n_1 n_2, `V` n_1 n_2, or `S` n_1, indicating either a horizontal or vertical domino, or a singleton value (there is exactly 1 singleton in each test case). These specifications may be over multiple lines. As you read in each domino specification you place it in the first available location going row-wise left-to-right, top-to-bottom.

Output

For each test case, output the maximum obtainable score for the given grid of dominoes involving at most one domino replacement.

Sample Input

```
1
V 6 3 H 5 5 V 6 2 V 6 4 V 6 1
V 3 4 V 3 3 H 1 4 H 5 2 S 4 V 4 4
H 5 6 H 5 1
```

Sample Output

```
Case 1: 218
```

Freetime Challenge!

What's the highest scoring Domiyahtzee grid that you can make? Our highest scoring grid is worth 498 points.

Problem D: Generalized Roman Numerals

The ancient Romans developed a terrible numbering system in which I, V, X, L and C stood for 1, 5, 10, 50 and 100, respectively. So XXXVII represents 37 (10+10+10+5+1+1). They typically wrote the numerals in non-increasing order. However, when a single Roman numeral is written before one that is larger, we subtract the smaller from the larger. So we can write IV and IX to represent 4 and 9 (subtracting 1), or XL and XC to represent 40 and 90 (subtracting 10). To represent 94, we would write XCIV.

VIC is generally not considered a traditional Roman numeral, but we can interpret this as another representation of 94: VI is 6, so VIC is 100-6. In general, if we have two expressions a and b representing values $v(a)$ and $v(b)$, then we say that $v(ab)$ is $v(a) + v(b)$ if $v(a) \geq v(b)$, and $v(b) - v(a)$ otherwise.

Unfortunately, this generalization introduces some ambiguity, since different orders of evaluation may result in different values. For example, consider IVX: IV is 4 and X is 10, so by that reasoning IVX is 6. However, I is 1 and VX is 5, so this suggests that IVX is actually 4. To remedy this ambiguity, we allow the addition of parentheses. The question arises: for a given string of Roman numeral characters, how many different values can be obtained using different placements of parentheses?

Input Time Limit: 3 secs, No. of Test Cases: 33, Input File Size 0.678K

Each test case consists of a single string containing only the characters I, V, X, L and C. The length of this string will be ≤ 50. A line containing of a single 0 will terminate input.

Output

For each test case, output all possible distinct values that can be represented by the string via the addition of parentheses. Display these values in increasing order.

Sample Input

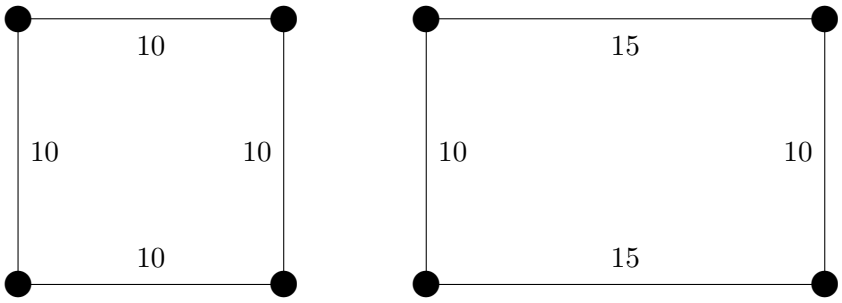
```
IVX
XIXIX
0
```

Sample Output

```
Case 1: 4 6
Case 2: 8 10 28 30 32
```

Problem E: Inspectors

Yuri is a rich business owner. His company has factories in many cities across Russia, all of which need regular, monthly inspections. Now Yuri not only has a large portfolio, but a large family as well and he would like to see some of his children start to learn how the business is run. He figures the best way to do this is to have them perform the factory inspections, but he has two concerns. First, he wants each child to be assigned at least two different factories – that way they will be able to compare different management styles at the factories and get a better idea of what works and what doesn't work. This of course requires them to travel between the factories, which brings up Yuri's second problem: while he won't be paying his children anything (their weekly allowance is more than adequate), he's worried about the total of all the travel costs, and would like it to be as small as possible. Each child is expected to inspect each of their assigned factories once a month, always ending back at the factory they started at, so to minimize cost he figures he need only assign the factories to minimize the total distance of these factory tours. The more he thinks about this, the more important saving money becomes. He's willing to use as few as only one of his children if that's the cheapest way to inspect all the factories. For example, if Yuri had four factories placed as in the figure on the left below, he could employ just one child, who could visit each factory for a monthly distance of 40 (assume here that the unlisted distances between factories are all > 10). However, if his factories were located as shown on the right, he would need to employ two children to obtain the monthly minimal distance of 40 (note: he could use two children to achieve the minimal distance in the first figure as well).



Since the number of factories changes over time, Yuri would like a program to determine the minimum distance he can expect for any layout of factories.

Input Time Limit: 3 secs, No. of Test Cases: 101, Input File Size 716K

The input file begins with a line containing a single integer m indicating the number of test cases. Each test case starts with an integer n indicating the number of factories, where $2 \leq n \leq 100$. The next $n-1$ lines contain the positive integer distances between the factories: line 1 contains $n-1$ values specifying the distances between factory 1 and factories $2, 3, 4, \ldots, n$; line 2 contains $n-2$ values specifying the distances between factory 2 and factories $3, 4, \ldots, n$; and so on. The maximum distance between any two factories is 1000. All distances between any three cities will satisfy the triangle inequality.

Output

For each test case display the minimum factory tour distance. You should always assume that Yuri has at least $n/2$ kids for each set of n factories.

Sample Input

```
2
4
10 20 10
10 20
10
4
15 20 10
10 20
15
```

Sample Output

```
Case 1: 40
Case 2: 40
```

Problem F: Path of Least Persistence

Bob Roberts owns a company that makes novelty games for parties. One of his favorites is a puzzle labyrinth, where players must navigate a path through a rectangular grid based on puzzles they must solve at each grid location. The solution to each puzzle indicates which grid square to go to next, and this continues until the player reaches the destination square. Bob has software to lay out the intended path, but there was a glitch in the algorithm and the resulting grids often have either very long paths or paths that don't lead to the destination. For example, consider the 3x3 grid shown on the left in Figure 1.

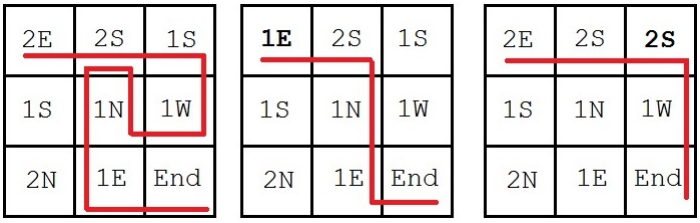


Figure 1

The solution to each puzzle is shown in each grid square, which gives the number of squares to move and the direction to move (either North, South, East or West). In this grid the path from the start square (always the upper left) to the destination square (always the lower right) has six segments, which is very unsatisfying to many customers (who aren't persistent enough to solve so many puzzles). Bob noticed this when setting up the grid, and because he didn't have much time, he looked to see if there was one grid square solution he could change to make the path shorter. After a quick look, he changed the puzzle in the upper left square so that its solution changed from 2E to 1E (the middle figure) which led to a path of length 3 – not ideal, but better than the original path. After the party was over, Bob realized that there was an even better solution, shown in the right figure, that led to a path of length 2 (note that by path length we mean the number of puzzles solved while on the path)

Changing a solution to a puzzle is somewhat complicated and time consuming, so Bob only wants to change at most one puzzle when setting up a grid. He has come to you to help solve the following problem: given a grid layout, which is the one square direction to change to minimize the path from the start to the destination. Bob realizes that sometimes there is no answer (since it might take two or more changes to actually get a path) and sometimes there is no need to change any square, since any change would result in a path no shorter than the one that already exists. He is confident that you can handle all these cases.

Input Time Limit: 3 secs, No. of Test Cases: 76, Input File Size 1157K

The first line of each case contains two positive integers n m indicating the number of rows and columns in the grid, where $n, m \leq 100$. Following that are $nm - 1$ direction specifications for the grid, given row-wise left to right starting at the topmost row. These are in the form r`D`, where r is the number of grid squares to move and `D` is the direction to move, either `'N'` (up), `'S'` (south), `'E'` (right), or `'W'` (you figure it out). Each grid will contain at least 2 grid squares. A line containing two 0's will terminate input.

Output

For each test case, output the case number followed by one of three possible answers. If there is no way to change one square to get a path to the destination, output `impossible`. If the existing path in the grid cannot be made shorter, output `none` l, where l = length of the existing path. Otherwise, output the row and column of the square to change (where the top row is row 0 and the leftmost column is column 0) followed by the new direction for that square, followed by the length of the new, shortest path. If there are two or more changes that could be used, display the one that has the lowest row number. If there are two or more with the lowest row number, display the one with the lowest column number. Finally, if there is still a tie use the change that comes first in the following ordering: `1E` < `1N` < `1S` < `1W` < `2E` < ... etc.

Sample Input

```
3 3
2E 2S 1S 1S 1N 1W 2N 1E
2 4
1E 1W 1W 1W 1W 1W 1W
1 4
3E 1E 1E
0 0
```

Sample Output

```
Case 1: 0 2 2S 2
Case 2: impossible
Case 3: none 1
```

Problem G: Speed Skills

Anna has just gotten a new car and is dying to take it out on the highway. To minimize her chance of getting a speeding ticket she wants to make sure she is not going that much faster than anyone else on the highway. However, she doesn't want to go too slow, regardless of the speed of the other drivers. She has hit on the following strategy. Every time she passes a car, she will reduce her speed to the average of her old speed and the other car's speed; when she is passed by someone, she increases her speed likewise. To carry out this strategy, she has built a little gizmo which checks for passing or passed cars 4 times each second. It then adjusts her car's speed to the average of her current speed and those of all the cars either being passed or passing her, rounding the average down to the nearest integer. For example, if at a particular quarter second Anna is going 60, is being passed by a car going 65 and is also passing another car going 59, her car changes its speed to $\lfloor (60 + 65 + 59)/3 \rfloor = 61$. For the purposes of this problem, Anna's car is considered passing or being passed if the front of the other car is anywhere between the back and front of Anna's car, inclusive. All cars will be considered to have the same length and all cars (except Anna's) travel at a constant speed. Also, assume that Anna's car changes speed instantaneously.

Input Time Limit: 3 secs, No. of Test Cases: 50, Input File Size 32.6K

The first line of each test case contains four non-negative integers l s d n, where l is Anna's current location on the highway (in car lengths), s is her current speed (in car lengths per second), d is the location of a destination point on the highway (where $l < d \leq 100000000$), and n is the number of other vehicles on the road (all going in Anna's direction). The maximum value of n is 100. The next n lines contains n pairs of integers l_i $s_i, 1 \leq i \leq n$, indicating the location and speed of each of the other n vehicles. All locations refer to the front of the car in question. All input refers to location and times at time $t = 0$, which is also the time Anna's gizmo takes its first reading. A line containing four 0's will terminate input.

Output

For each test case output a single line containing the time Anna reaches the destination point (in seconds) and her current speed at that time (in car lengths per second), using the format shown below. The destination point is considered reached the moment the front of Anna's car reaches it. If her destination arrival time is on a quarter second and she is being passsed or is passing someone at that same instant, output her new averaged speed. Round all output to the nearest ten-thousandth.

Sample Input

```
300 20 1000 2
200 30
2 40
300 20 1000 2
200 30
100 40
0 0 0 0
```

Sample Output

```
Case 1: Anna reaches her destination at time 27.0469 at a speed of 32
Case 2: Anna reaches her destination at time 26.6667 at a speed of 30
```

Problem H: Time Warp

Tim Ang is a bit of a nerd. Check that – he's a HUGE nerd. When you ask him the time, he might say something like "20 after 8", which seems normal, but other times he'll say things like "90 after 8" or "126 til 4", which gives you pause. When you ask him about this, Tim say that "20 after 8" means the first time after 8 that the hour and minute hands of the clock make an angle of 20 degrees; "126 til 4" means the closest time before 4 that the hands make an angle of 126 degrees. As Tim walks away snickering, you resolve that you will write a program that will automatically convert Tim's times to our more normal, non-nerdy times. That'll show the little geek!

Input Time Limit: 3 secs, No. of Test Cases: 8640, Input File Size 94.6K

The input file starts with an integer n indicating the number of test cases. Each test case consists of a single line of the form a `after` b or a `til` b, where a and b are integers satisfying $0 \leq a < 360$, and $1 \leq b \leq 12$. All angles are measured starting at the hour hand and moving clockwise until reaching the minute hand (so, for example, at 9 o'clock the hands make an angle of 90 degrees and at 3 o'clock they make an angle of 270).

Output

For each test case, output the time in the format `h:m:s`, where `h, m` and `s` are the hour, minutes and seconds closest to the given angle and hour and $1 \leq$ `h` ≤ 12. Always use two digits to represent the number of minutes and seconds.

Sample Input

```
4
20 after 8
126 til 4
180 til 1
0 after 12
```

Sample Output

```
Case 1: 8:47:16
Case 2: 3:39:16
Case 3: 12:32:44
Case 4: 1:05:27
```

Problem I: Watch, Man!

Lou Va is the curator of a world-renowned art museum. While he has hundreds of pieces of artwork on display, there are certain ones that are extremely valuable and need extra levels of security. Lou has set up a collection of security locations throughout the museum and plans to place security guards there to keep an eye on the valuable art.

After careful analysis of the art pieces and the security guards' resumes, Lou has assigned different *levels* to each art piece and each guard. A level n art piece is one which must be in view of at least n security guards. A level m security guard is one who has enough experience to keep an eye on up to m pieces of art. Lou wants to place the guards so that each piece of art can be watched by the appropriate number of guards.

One more complication: this is a modern-art musuem, and its layout is non-conventional. Walls of all the rooms are either straight line segments or arcs of circles. The first two layouts in Figure 1 show one such room, with different placements of guards and art pieces (with their respective level values shown). In addition, there may be one or more interior sets of walls that can block the guards' views, as shown in the third layout below. Any set of interior walls forms a simple closed loop, and if there is more than one such set of interior walls, none will intersect or nest within each other.

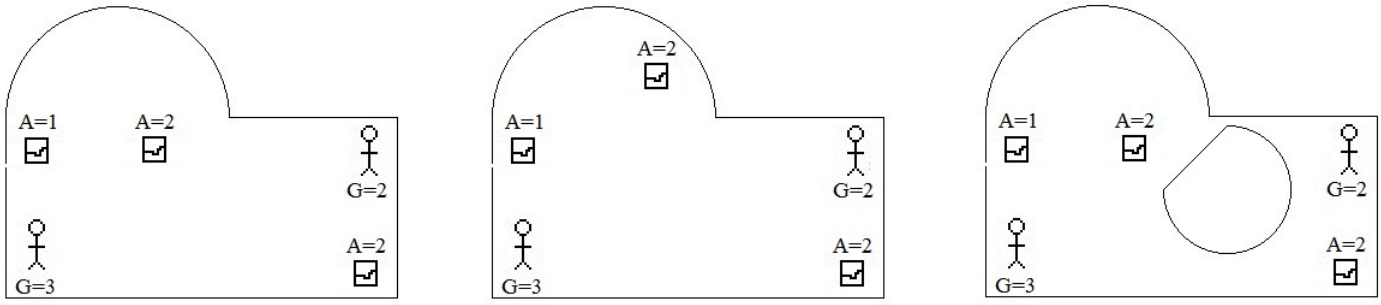


Figure 1

As the second and third examples show, sometimes the placements of the guards is not sufficient to watch all of the art pieces. Lou needs to know this, as he will then change either the placements of the guards or the art pieces. He has come to you for some help.

Input Time Limit: 3 secs, No. of Test Cases: 48, Input File Size 23.7K

The first line of each test case contains three integers n a g indicating the number of wall sets, art pieces and guards, respectively, where $1 \leq n$ and $0 \leq a, g \leq 100$. Following this are descriptions of each of the n wall sets, where the first set is the set of outer walls. Each description starts with an integer m indicating the number of walls in the wall set. This is followed by m sets of integer coordinates x_i y_i each followed by either an '`s`' or a '`c`'. An '`s`' indicates that point (x_i, y_i) should be connected by a straight-line wall to the next point, and a '`c`' indicates it should be connected with a circular arc. Following a '`c`' are two integers giving the dx and dy values of the tangent line of the circle at the point (x_i, y_i). Two examples of this are shown in Figure 2.

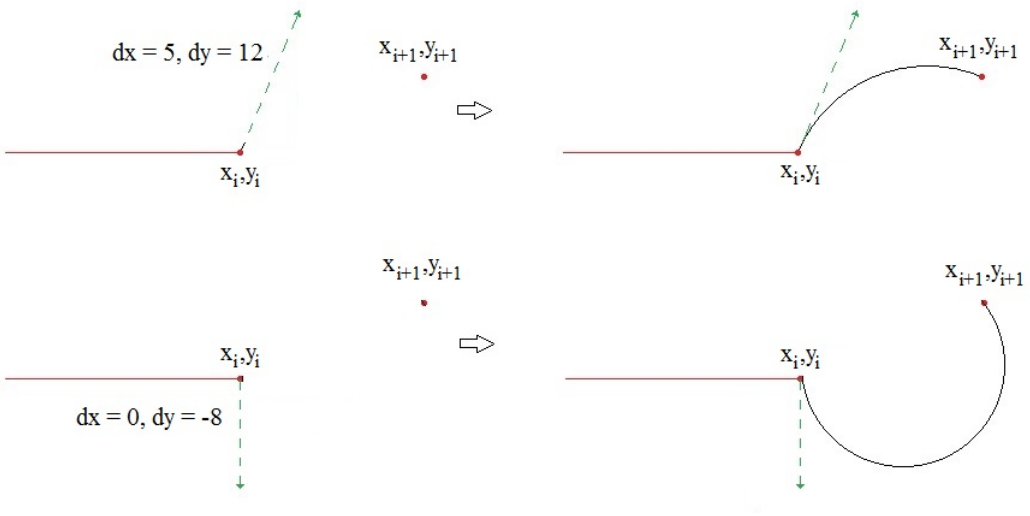


Figure 2

The total number of walls for all the rooms in any test case will be ≤ 125. Following the wall specifications are a integer coordinates giving the location of the art pieces, each followed by a positive level number. Next are g integer coordinates giving the location of the guards, each followed by a positive level number. No wall corner will lie on a line segment connecting an art piece and a guard, and no line between an art piece and a guard will ever be tangent to a curved wall. All coordinates are in the range -150000 to 150000. A line containing three 0's will terminate input.

Output

For each test case, output the case number and either `Yes` or `No` indicating whether or not the guards in their current locations can watch the art pieces at the appropriate levels.

Sample Input (shown across multiple columns)

```
1 3 2
5
0 0 s
0 20 c 0 1
20 20 s
40 20 s
40 0 s
2 18 1
15 18 2
38 2 2
2 2 3
38 18 2
```

```
1 3 2
5
0 0 s
0 20 c 0 1
20 20 s
40 20 s
40 0 s
2 18 1
18 24 2
38 2 2
2 2 3
38 18 2
```

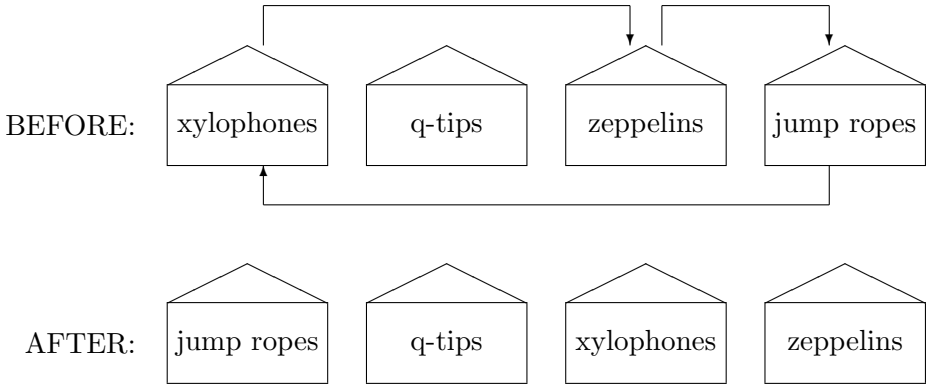
```
2 3 2
5
0 0 s
0 20 c 0 1
20 20 s
40 20 s
40 0 s
2
23 19 c 1 0
15 11 s
2 18 1
15 18 2
38 2 2
2 2 3
38 18 2
0 0 0
```

Sample Output

```
Case 1: Yes
Case 2: No
Case 3: No
```

Problem J: The Cost of Moving

You've been put in charge of reorganizing the inventory at Amalgamated, Inc. AI has factories at various sites around the country, each site manufacturing different sets of products. Currently, each site has all of their products stored in a row of warehouses, with each warehouse storing one type of product. Your idea is to move these products around so that the warehouses store the items in alphabetical order. For example, one site which manufactures xylophones, q-tips, zeppelins and jump ropes stores them in four warehouses as shown at the top of the following figure:



The arrows show how far each product would have to be moved in order for the warehouse to store them in alphabetical order, which is shown at the bottom of the figure.

Management likes your idea but is concerned about the cost of moving all of the products. The farther a product has to be moved, the more it costs, so before committing to any re-ordering they would like to know the total length that all products have to be moved at any given site. In the site above, xylophones would have to be moved a total length of 2 (measured in warehouses), zeppelins would have to move 1 and jump ropes would have to move 3, for a total cost of 6. What you need is a program that can determine this movement cost automatically.

Input **Time Limit: 3 secs, No. of Test Cases: 37, Input File Size 52.0K**

There will be multiple test cases. Each test case will start with a line containing a positive integer `n` indicating the number of products at the site. Following that will be one or more lines containing the `n` names of the products in their current order. Each name will be a single string, and a single space will separate each consecutive pair of names on any line. The maximum value of `n` is 1000, and no two product names will be the same. A single zero will terminate input.

Output

For each test case, output the site number followed by the total length of movements needed to re-organize the products. Follow the format shown in the Sample Output.

Sample Input

```
4
xylophones q-tips zeppelins jumpropes
12
partridges turtledoves frenchhens callingbirds goldenrings
geese swans milkers dancers leapers pipers drummers
0
```

Sample Output

```
Site 1: 6
Site 2: 48
```

Problem K: A Jaw-dropping Problem to Vex the Quizzically Freakish

You all know what a pangram is, so I don't have to tell you that it's a phrase or sentence that uses every letter of the alphabet at least once. You also know what the most famous pangram is, so I won't waste your time reminding you that it's:

The quick brown fox jumps over a lazy dog

BUT, do you know what a double pangram is? Yes, yes, you're right, it's a phrase or sentence that uses every letter at least twice. And a triple pangram is one that uses every letter at least three times. And a ... well, we could go on, but the practice contest is only one and a half hours so we'll stop here.

Input Time Limit: 3 secs, No. of Test Cases: 113, Input File Size 13.0K

There will be multiple test cases. The input file starts with an integer `n` indicating the number of cases. Each test case will be a single line containing upper and lower case letters, as well as other non-alphabetic characters such as digits, punctuation and spaces.

Output

For each test case, output the case number followed by one of the following phrases:

```
Not a pangram
Pangram!
Double pangram!!
Triple pangram!!!
```

Select the phrase that best describes the test case. For example, even though a triple pangram is also a double (and single) pangram, the phrase `Triple pangram!!!` best describes it. There will be no test cases which use every letter of the alphabet more than three times.

Sample Input

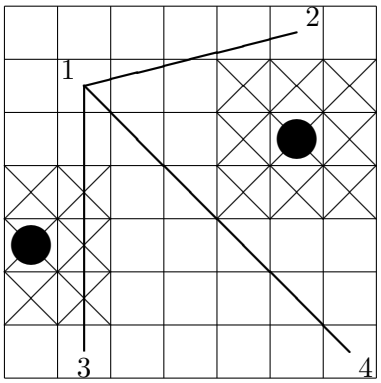
```
3
The quick brown fox jumps over a lazy dog.
The quick brown fox jumps over a laconic dog.
abcdefghijklmNOPQRSTUVWXYZ-zyxwvutsrqpon   2013/2014      MLKJIHGFEDCBA
```

Sample Output

```
Case 1: Pangram!
Case 2: Not a pangram
Case 3: Double pangram!!
```

Problem L: This Too Shall Pass

Tessa is a high school senior who plays on the soccer team. Her coaches are trying to put an emphasis on passing, and they want to help players learn how to recognize when a teammate is open for a pass. They've decided to model the field as a square grid like the one shown below. Each player (for both the offense and the defense) takes up one grid square - the offensive players are indicated with numbers and the defensive players with black circles. Each defensive player also has the ability to move to any neighboring square in order to intercept a pass; the squares that each defender can guard are shown with an X in them. Assume player 1 has the ball. Another player is considered open if the line drawn from the center of that person's square to the center of player 1's square does not touch any of the squares that can be reached by a defender. If the line touches a defenders area even at a single point, then the pass could be intercepted. Offensive players never block a pass to other offensive players.



In the above example, Player 1 can pass the ball to Player 2, but not to Players 3 and 4.

Having set up this great model, the coaches suddenly realize that they don't have any ability to write code to determine who is open and who is isn't. One of the players has given them your name as a computer whiz, so it's time to get your game face on, put it all on the line, give 110% and code one for the Gipper.

Input Time Limit: 3 secs, No. of Test Cases: 36, Input File Size 63.5K

The input file will consist of multiple test cases. Each case starts with four positive integers `r c o d` indicating the number of rows (`r`) and columns (`c`) in the grid and the number of offensive and defensive players (`o` and `d`, respectively). Both `r` and `c` will be ≤ 50. Following this will be `o` lines containing two integers giving the row and column location of an offensive player (row and column numbering start at 0). Following this will be `d` analogous lines for the defenders. The offensive players are numbered $1, 2, 3, \ldots$ in the order that they appear in the input, and offensive player 1 is the one with the ball. No two players will ever be in the same grid square. A line with four zeros will terminate input.

Output

For each test case, output the case number followed by a list of all the players that Player 1 can pass the ball to. Output the numbers in increasing order, separated by a single space. Label each test case as shown below.

Constraints

$2 \le M \le N \le 1\,000$ and $1 \le P \le 10\,000$
$1 \le X, Y, a_i, b_i \le N$ and $X \ne Y$ and $a_i \ne b_i$
$1 \le c_i \le 1\,000\,000\,000$

Output

The output should be a single integer greater than zero: the largest fee so that SWERC can provide cheapest way to transfer money between X and Y. However, if there is no value such that this happens, output `Impossible` instead. If the fee on each transfer can be in nitely large, output `Infinity` .

Sample Input 1

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

Sample Output 1

```
3
```

Sample Input 2

```
3 4 1 2
1 2 6
1 3 2
1 2 7
2 3 3
2
1 2
```

Sample Output 2

```
Infinity
```

Sample Input 3

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

Sample Output 3

```
Impossible
```

Sample Output 1 Explanation

If the extra fee is 4 or more, then SWERC can not provide the cheapest transaction fee. Example: if the fee is 4, SWERC provides a cost of 20, using banks 1, 3, 4, 5 and 6, in this order. However, using bank 2 as an intermediary, we can pay only 19.