acm International Collegiate Programming Contest

IBM event sponsor

2008

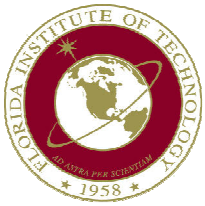
2008 ACM ICPC
Southeast USA Regional
Programming Contest

25 October, 2008

PROBLEMS

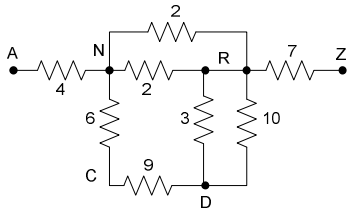
Hosted by:

Florida Institute of Technology
Armstrong Atlantic State University
University of South Alabama



A: Series / Parallel Resistor Circuits

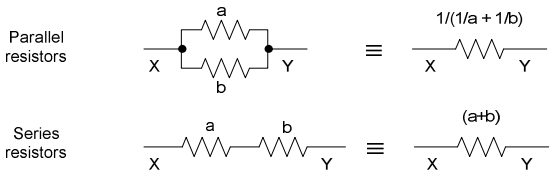
A series / parallel resistor circuit is shown below.



The resistance value is given next to each resistor. Connection points (wires connecting two or more resistors together, are denoted by an uppercase letter. A and Z are reserved for the names of the connection points which are the endpoints of the circuit. Our goal is to calculate the equivalent resistance of the circuit (i.e., the equivalent resistance between A and Z).

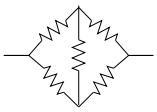
Within the circuit, a resistor can be specified by a triple consisting of the connection points at either endpoint, and the resistance. Resistor 9 could be specified as either (C, D, 9) or (D, C, 9). A circuit specification is the set of all resistor specifications.

A pair of resistors is in series if one of either of their endpoints have a common connection point that is not use by any other resistor (e.g., resistor 6 and 9, are both connected to C, which is not connected to anything else). Two series resistors can be replaced by an equivalent single resistor whose resistance is the sum of the replaced resistors (15, in the previous example). A pair of resistors is in parallel if both their endpoints have common connection points (e.g., resistors 3 and 10 above, both are connected to R and D). Two parallel resistors can be replaced by an equivalent single resistor whose resistance is the inverse of the sum of the inverses of the two resistors ($(1/3 + 1/10)^{-1} = 2.307692$, in the previous example).



The equivalent resistance of a well-formed series-parallel resistor [1]circuit can be determined by successively replacing a series or parallel resistor pair by the single equivalent resistor, until only one is left. If this technique fails, the circuit is not well-formed.

[1] A Wheatstone Bridge circuit, shown on the right, is not a well-formed series-parallel circuit.



Input

There will be multiple circuit specifications. The first input line for each circuit specification is an integer ***N*** (***N*** **<= 1000**), being the number of resistors in the circuit. This is followed by ***N*** lines, each being a resistor specification in the form: ***X Y r***, where ***X*** and ***Y*** are uppercase characters, and ***r*** is a positive integer resistance (***r*** **< 100**). The equivalent resistance is guaranteed never to be greater than 100. A line with a single **0** terminates the input.

Output

For each circuit, if the circuit is well-formed and reduces to a single equivalent resistance between A and Z, print the equivalent resistance of the circuit from A to Z, rounded to (and displayed to) 3 decimal places. If the circuit is not well formed, or if there is no equivalent resistance between A and Z, simply print the number **-1.000**. There should be no blank lines between outputs.

Sample Input

```
8
N R 2
D R 3
R N 2
R D 10
Z R 7
C D 9
N C 6
A N 4
2
A Z 3
Z A 10
2
P A 6
P Z 9
5
A B 1
B Z 4
A C 8
C Z 19
B C 12
0
```

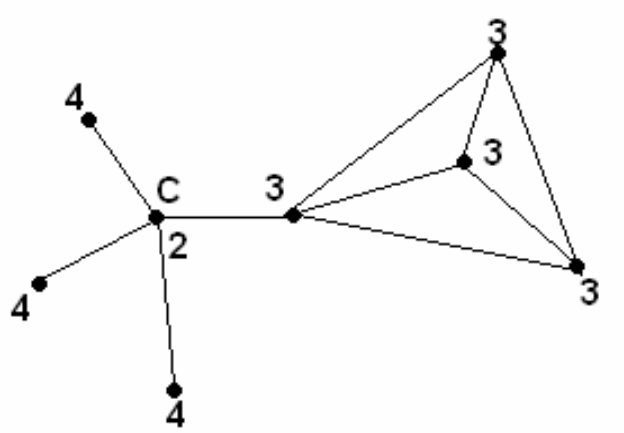
Sample Output

```
11.945
2.308
15.000
-1.000
```

B: The Heart of the Country

The nation of Graphia is at war. The neighboring nations have for long watched in jealousy as Graphia erected prosperous cities and connected them with a network of highways. Now they want a piece of the pie.

Graphia consists of several cities, connected by highways. Graphian terrain is rough, so the only way to move between the cities is along the highways. Each city has a certain number of troops quartered there. Graphia's military command knows that it will require a certain number of troops, **K**, to defend any city. They can defend a city with the troops stationed there, supported by the troops in any other city which is directly connected with a highway, with no cities in between. Any troops further away than that simply cannot get there in time. They also know that their enemies will onlyattack one city at a time – so the troops in a city can be used to defend that city, as well as any of its neighbors. However, if a city can't be defended, then the military command must assume that the troops quartered in that city will be captured, and cannot aid in the defense of Graphia. In the case below, suppose ***K*=10**. City C might seem well defended, but it will eventually fall.



Graphia's leadership wants to identify the Heart of their country – the largest possible group of cities that can mutually defend each other, even if all of the other cities fall.

More formally, a city is defensible if it can draw a total of at least **K** troops from itself, and from cities *directly adjacent* to it. A set of cities is defensible if every city in it is defensible, using *only* troops from itself and adjacent cities *in that set*. The Heart of the country is the largest possible defensible set of cities - that is, no other defensible set of cities has more cities in it.

Input

There will be several data sets. Each set begins with two integers, ***N*** and ***K***, where ***N*** is the number of cities (**3 <= *N* <= 1000**), and ***K*** is the number of troops required to defend a city. The cities are numbered **0** through ***N*-1**.

On the next ***N*** lines are descriptions of the cities, starting with city **0**. Each of the city description lines begins with an integer ***T***, indicating the number of troops quartered in that city (**0 <= *T* <= 10000**). This is followed by an integer ***M***, indicating the number of highways going out of that city, and then ***M*** integers, indicating the cities those highways go to. No two highways will go from and to the same cities, so every city in each list will be unique. No highway will loop from a city back to the same city. The highways go both ways, so that if city ***I*** is in city ***J***'s list, then it's guaranteed that city ***J*** will be in city ***I***'s list in the input. The input will end with a line with two space-separated **0**'s.

Output

For each data set, print two integers on a single line: The number of cities in the heart of the country, and the number of troops in the heart of the country. Print a space between the integers. There should be no blank lines between outputs.

Sample Input

```
4 900
100 2 1 2
200 2 0 3
500 2 0 3
1000 2 1 2
4 900
100 3 1 2 3
200 3 0 3 2
500 3 1 3 0
1000 3 2 1 0
0 0
```

Sample Output

```
3 1700
4 1800
```

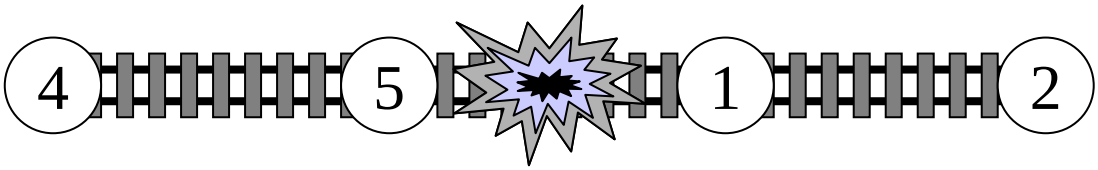
C: Lawrence of Arabia

T. E. Lawrence was a controversial figure during World War I. He was a British officer, who served in the Arabian theater. He led a group of Arab nationals in guerilla strikes against the Ottoman Empire. His primary targets were the railroads. A highly fictionalized version of his exploits was presented in the blockbuster movie, “Lawrence of Arabia”.

You are to write a program to help Lawrence figure out how to best use his limited resources. You have some information from British Intelligence. First, the rail line is completely linear – there are no branches, no spurs. Next, British Intelligence has assigned a Strategic Value to each depot – an integer from 1 to 5. But, a depot is of no use on its own, it only has value if it is connected to other depots. The Strategic Value of the entire railroad is calculated by adding up the products of the Strategic Values for every pair of depots that are connected, directly or indirectly, by the rail line. Consider this railroad:

Its Strategic Value is **`4*5 + 4*1 + 4*2 + 5*1 + 5*2 + 1*2 = 49`**.

Now, suppose that Lawrence only has enough resources for one attack. He cannot attack the depots themselves – they’re too well defended. He must attack the rail line between depots, in the middle of the desert. Consider what would happen if Lawrence attacked this rail line right in the middle:



The Strategic Value of the remaining railroad is **`4*5 + 1*2 = 22`**. But, suppose Lawrence attacks between the 4 and 5 depots:

The Strategic Value of the remaining railroad is **`5*1 + 5*2 + 1*2 = 17`**. This is Lawrence’s best option.

Given a description of a railroad and the number of attacks that Lawrence can perform, figure out the smallest Strategic Value that he can achieve for that railroad.

The Input

There will be several data sets. Each data set will begin with a line with two integers, ***N*** and ***M***. ***N*** is the number of depots on the railroad (**1 <= *N* <= 500**), and ***M*** is the number of attacks Lawrence has resources for (**0 <= *M* < *N***). On the next line will be ***N*** integers, each from **1** to **5**, indicating the Strategic Value of each depot in order. End of input will be marked by a line with two space-separated **0**'s.

The Output

For each data set, print a single integer, indicating the smallest Strategic Value for the railroad that Lawrence can achieve with his attacks. Print each integer on its own line. There should be no blank lines between outputs.

Sample Input

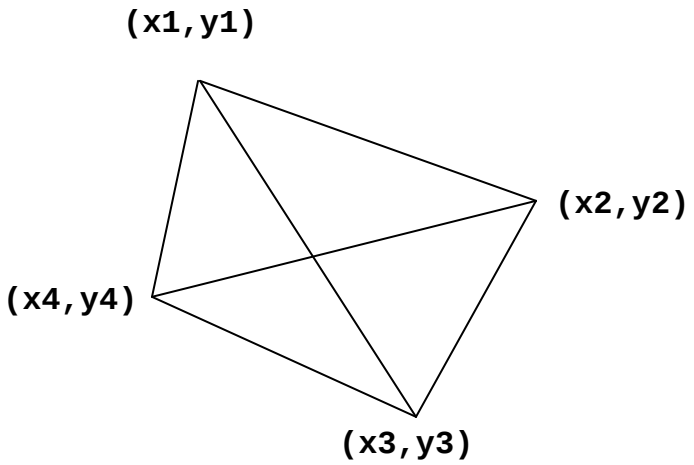
```
4 1
4 5 1 2
4 2
4 5 1 2
0 0
```

Sample Output

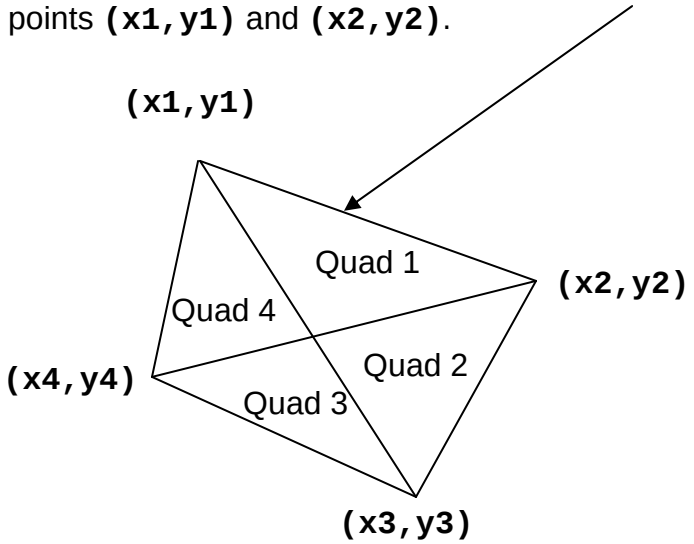
```
17
2
```

D: Shoring Up the Levees

The tiny country of Waterlogged is protected by a series of levees that form a quadrilateral as shown below:



The quadrilateral is defined by four vertices. The levees partition the country into four quadrants. Each quadrant is identified by a pair of vertices representing the outside edge of that quadrant. For example, Quadrant 1 shown below is defined by the points **(x1,y1)** and **(x2,y2)**.



It happens very often that the country of Waterlogged becomes flooded, and the levees need to be reinforced, but their country is poor and they have limited resources. They would like to be able to reinforce those levees that encompass the largest area first, then the next largest second, then the next largest third, and the smallest area fourth.

Help Waterlogged identify which quadrants are the largest, and the length of the levees around them

Input

There will be several sets of input. Each set will consist of eight real numbers, on a single line. Those numbers will represent, in order:

X1 Y1 X2 Y2 X3 Y3 X4 Y4

The four points are guaranteed to form a convex quadrilateral when taken in order – that is, there will be no concavities, and no lines crossing. Every number will be in the range from **`-1000.0`** to **`1000.0`** inclusive. No Quadrant will have an area or a perimeter smaller than **`0.001`**. End of the input will be a line with eight **`0.0`**'s.

Output

For each input set, print a single line with eight floating point numbers. These represent the areas and perimeters of the four Quadrants, like this:

A1 P1 A2 P2 A3 P3 A4 P4

Print them in order from largest area to smallest – so ***A1*** is the largest area. If two Quadrants have the same area when rounded to 3 decimal places, output the one with the largest perimeter first. Print all values with 3 decimal places of precision (rounded). Print spaces between numbers. Do not print any blank lines between outputs.

Sample Input

```
1 2 1 5 5 2 2 0
3.5 2.2 4.8 -9.6 -1.2 -4.4 -8.9 12.4
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

Sample Output

```
5.100 11.459 3.400 9.045 0.900 6.659 0.600 4.876
44.548 38.972 21.982 25.997 20.342 38.374 10.038 19.043
```

E: Combination Lock

A combination lock consists of a circular dial, which can be turned (clockwise or counterclockwise) and is embedded into the "fixed" part of the lock. The dial has **N** evenly spaced "ticks". The ticks are numbered from **0** to **N-1**, increasing in the clockwise direction. The fixed part of the lock has a "mark" which always "points to" a particular tick on the dial. Of course, the mark points to different ticks as the dial is turned.

The lock comes with three code numbers **T1**, **T2**, **T3**. These are non-negative integers and each of them is less than **N**. No two of the three are the same.

The lock is opened in three stages of operations:

1. Turn the dial clockwise exactly two full revolutions, and continue to turn it clockwise until the mark points to tick **T1**.
2. Turn the dial one full revolution counterclockwise and continue to turn it counterclockwise until the mark points to tick **T2**.
3. Turn the dial clockwise until the mark points to tick **T3**. The lock should now open.

You must find the maximum possible number of ticks the dial must be turned in order to open the lock. The number of ticks turned is defined to be the sum of the ticks turned in the three stages outlined above, and is always positive regardless of direction.

Input

The input file consists of a number of test cases, one test case per line. Each line of the input file contains four integers: ***N***, ***T1***, ***T2***, ***T3***, in this order, separated by blank spaces. The integer ***N*** is a multiple of **5**, **25 <= *N* <= 100**. The numbers ***T1***, ***T2*** and ***T3*** satisfy the constraints stated under the description above. The input will be terminated by a line with four blank-separated **0**'s.

Output

For each test case, print the maximum possible number of ticks the dial must be turned in order to open the lock. Print each on its own line. There should be no blank lines between outputs.

Sample Input

```
80 20 40 50
80 10 79 12
0 0 0 0
```

Sample Output

```
409
455
```

F: Fred's Lotto Tickets

Fred likes to play the lotto. Whenever he does, he buys lots of tickets. Each ticket has 6 unique numbers in the range from **1** to **49**, inclusive. Fred likes to "Cover all his bases." By that, he means that he likes for each set of lottery tickets to contain every number from **1** to **49**, at least once, on some ticket. Write a program to help Fred see if his tickets "Cover all the bases."

Input

The input file consists of a number of test cases. Each case starts with an integer ***N*** (**1 <= *N* <= 100**), indicating the number of tickets Fred has purchased. On the next ***N*** lines are the tickets, one per line. Each ticket will have exactly 6 integers, and all of them will be in the range from **1** to **49** inclusive. No ticket will have duplicate numbers, but the numbers on a ticket may appear in any order. The input ends with a line containing only a **0**.

Output

Print a list of responses for the input sets, one per line. Print the word **Yes** if every number from **1** to **49** inclusive appears in some lottery ticket in the set, and **No** otherwise. Print these words exactly as they are shown. Do not print any blank lines between outputs.

Sample Input

```
1
1 2 3 4 5 6
9
1 2 3 4 5 6
10 9 8 7 12 11
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
31 32 33 34 35 36
37 38 39 40 41 42
43 44 45 46 47 48
49 19 34 27 25 13
0
```

Sample Output

```
No
Yes
```

G: A No-Win Situation

Consider a simple variation of the card game Blackjack. In this game, a single player plays against the dealer. The game uses a standard deck of cards, where numbered cards are worth the number of points on the card for the cards numbered 2 to 10, 10 points for the face cards (King, Queen, and Jack) and either 1 or 11 points for the Aces.

The dealer deals the first card in the deck to the player, the second to the dealer, the third to the player, and the fourth to the dealer. The player then may continue to draw cards until s/he decides that the total is as close as possible to 21 and stops voluntarily or until s/he goes over 21. If the player goes over 21, the player loses. Then the dealer must draw cards until s/he reaches 17 or more points (with aces counting as 11 when possible). If the dealer goes over 21, the dealer loses. If neither of them goes over 21, the winner is the one who comes as close as possible to 21. If the player and the dealer have the same total, the player wins.

For example, suppose the first cards in the deck are Queen, 6, 4, 9, and 10. On the initial deal, the player will receive Queen and 4 (for a total of 14) and the dealer will receive 6 and 9 (for a total of 15). If the player does not select a card, the dealer will have to draw (because the dealer's total is less than 17) and will draw the 10, going over, so the player will win. But if the player draws a card (the 10), the player's total will be 24, so the player will lose.

In some situations, it is impossible for the player to win. Consider the case when the cards in the deck are: 10, 3, 4, King, 3, 5. The player will be dealt the cards 10 and 4. The dealer will have 3 and King. The table below illustrates what happens for each number of cards the player might draw:

Cards drawn	Player's hand (Points)	Dealer's hand (Points)
0	10, 4 (14)	3, King, 3, 5 (21)
1	10, 4, 3 (17)	3, King, 5 (18)
2	10, 4, 3, 5 (22)	3, King (13)

No matter how many cards the player draws, the player cannot win.

In this problem, you will analyze decks to determine if they lead to a situation in which the player cannot win.

Input

The input to the program will be one or more decks. Each deck will be represented by a string, on its own line. Each deck will consist of at least 4 cards. where a card is either an integer ***d***, **2** <= ***d*** <= **9**, representing a numbered card, or one of the letters **A**, **K**, **Q**, **J** or **T**, representing Ace, King, Queen, Jack, or Ten, respectively. The letters will be in upper case. There will be no other characters on a line. In particular, there will be no spaces. There will always be enough cards to try all valid draws. End of input is indicated by the word **JOKER**, alone on a line.

Output

Print a list of responses for the input sets, one per line. Print the word **Yes** if there is a number of cards the player can draw and win, and **No** if there is no way for the player to win. Print these words exactly as they are shown. Do not print any blank lines between outputs.

Sample input

```
Q649T
T34K35
AA2T34A5KKQAJ
JOKER
```

Sample Output

```
Yes
No
Yes
```

H: A Walk in the Park

You are responsible for inspecting the trees located in a park, to make sure they remain healthy. The location of each tree is given to you as a point in the two-dimensional plane, distinct from that of every other tree. Due to recently-replanted grass, you are only allowed to walk through the park along a collection of paths. Each path is described by an infinite-length horizontal or vertical line in the two-dimensional plane. No tree lies on any path.

You are concerned that it may not be possible to view all the trees in the park from the paths. In particular, a tree is visible *only* if you can view it by standing on some path while facing in a direction *perpendicular* to that path; there must be no intervening tree that obstructs your view. Given the geometrical configuration of the park, please report the number of visible trees.

Input

There will be multiple input sets. For each input set, the first line will contain two integers, ***N*** and ***M***, (**`0 < N,M <= 100000`**), separated by a space. ***N*** is the number of trees, and ***M*** is the number of paths.

The next ***N*** lines each contain two space-separated integers, ***X*** and ***Y***, specifying the coordinates of a tree. ***X*** and ***Y*** may be any 32-bit integers.

The next ***M*** lines each describe a path (a vertical or horizontal line). They have the form **x=*K*** or **y=*K***, with no spaces. ***K*** may be any 32-bit integer. **x** and **y** will be lower case.

End of the input is signified by a line with two space-separated **`0`**'s.

Output

For each input set, print a single line containing one integer, specifying the number of visible trees. There should be no blank lines between outputs.

Sample Input

```
6 3
-1 3
4 2
6 2
6 3
6 4
4 3
x=0
y=-1
y=5
1 2
2 3
x=5
y=5
0 0
```

Sample Output

```
5
1
```

I: Teleport Out!

You are in a rectangular maze and you would like to leave the maze as quickly as possible. The maze is a rectangular grid of square locations. Some locations are blocked. Some other locations are exits. If you arrive at an exit location, you can immediately leave the maze.

You may walk one step at a time, onto one of the locations adjacent to your current location. Two locations are adjacent if they share a side. That is, you can only move one step North, South, East or West. Of course, you cannot step off the maze, and you cannot step onto a blocked location.

In addition, at any step, you may choose to use your teleport device. This device will send you to a random non-blocked location inside the maze with uniform probability (including, possibly, the one where you currently are standing!). If the device happens to send you onto a spot that is also an exit, then you leave the maze immediately. Hooray!

The only way to leave the maze is by moving onto an exit (either by teleport or walking), you may not walk off the boundary of the maze. Write a program to calculate the expected number of steps you need in order to leave the maze. Assume that you would choose your actions (movements and using teleport device) optimally in order to minimize the expected number of steps to leave the maze. Using the teleport device is considered one step.

Input

There will be multiple test cases. Each test case starts with a line containing two positive integers ***R*** and ***C*** (***R*** `<= 200,` ***C*** `<= 200`). Then, the next ***R*** lines each contain ***C*** characters, representing the locations of the maze. The characters will be one of the following:

E: represents an exit, there will be at least one **E** in every maze.

Y: represents your initial location, there will be exactly one **Y** in every maze.

X: represents a blocked location.

`.`: represents an empty space.

You may move/teleport onto any location that is marked **E**, **Y** or `.`.

The end of input is marked by a line with two space-separated `0`'s.

Output

For each test case, print one line containing the minimum expected number of steps required to leave the maze, given that you make your choices optimally to minimize this value. Print your result to 3 decimal places. Do not print any blank lines between outputs.

Sample Input

```
2 1
E
Y
2 2
E.
.Y
3 3
EX.
XX.
..Y
3 3
EXY
.X.
...
0 0
```

Sample Output

```
1.000
2.000
6.000
3.250
```

J: Worms

Biologists are studying a certain, interesting kind of worm. Each worm can be seen as a line of cells of different types. When a worm is born, it only consists of a single cell. Every day, exactly 1 cell of the entire worm will grow and change into 2 cells. It is rather easy to determine the age of any such worm, since it's simply one less than the number of cells the worm has.

During a worm's growth, a cell does not change into any 2 arbitrary cells; each worm has a set of "growth rules" (encoded in its DNA) that it obeys. A growth rule can be expressed as ***A*→*BC***, where ***A***, ***B*** and ***C*** are uppercase letters (with letters **A**-**T**), representing different types of the worm's cells. The rule ***A*→*BC*** means that in one day, any single cell ***A*** can be grown into the 2 adjacent cells ***BC***, in that order. Note that the rule **I→JK** is different from the rule **I→KJ**. Different worms may have a different set of growth rules.

The worms have now thrown the scientists for a loop. Due to some unknown reason, some worms have mutated into a new kind of specimen. This new kind of worm has the exact same properties, except that during its growth, *multiple* parts of its body can grow at the same time. That is, every day any (at least one, at most all) of its cells can grow; each cell that grows will grow into exactly 2 cells (obeying growth rules similar to their older cousins).

As a result of the mutation, it is no longer trivial to determine the age of a worm. In fact, the exact age of some worms cannot be determined. As a simple example, if a worm has growth rules: **A→BC**, **B→AC**, **C→AB**, and the worm's current cell structure is **ACAB**, the worm can be either 2 or 3 days old (**A→BC→ACAB**, or **A→BC→ACC→ACAB**). Your task is to find out the youngest possible age of any given mutated worm.

Input

There will be multiple worms for examination in the input. Each worm's data set begins with an integer ***N*** (**1 <= *N* <= 80**), the number of growth rules. The next ***N*** lines each contain 3 uppercase letters (with letters **A-T**), representing a growth rule for the current worm. The 1st cell can grow into (and be replaced by) the 2nd and 3rd cells, in order, during the growth process. That is, the line:

ABC

means **A→BC** is a growth rule for the current worm.

The next (and last) line of each worm's data set contains a string of uppercase letters (with letters **A-T**). This line represents the current cell structure of the worm. Every worm in the input will have at least **1** and at most **50** cells.

The last worm will be followed by a line with a single **0**.

Output

For each worm, if the worm can be grown into the given cell sequence with the given growth rule set (starting with any arbitrary single cell), then print the minimum age, in days, of the worm, as an integer on its own line. If the worm cannot be grown into the given cell sequence with the given rule set (starting with any arbitrary single cell), then simply print the number **-1** on its own line. Print no blank lines between outputs.

Sample Input

```
3
ABC
BAC
CAB
ACAB
1
AAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
2
PAA
AAA
AAAAAAAAAAAAAAP
1
BAB
AAAAAAB
0
```

Sample Output

```
2
6
-1
6
```