A. Leapfrog: Ch. 1

This problem statement differs from that of Leapfrog Ch. 2 in only one spot, highlighted in bold below.

A colony of frogs peacefully resides in a pond. The colony is led by a single Alpha Frog, and also includes 0 or more Beta Frogs. In order to be a good leader, the Alpha Frog diligently studies the high art of fractions every day.

There are N lilypads in a row on the pond's surface, numbered 1 to N from left to right, each of which is large enough to fit at most one frog at a time. Today, the Alpha Frog finds itself on the leftmost lilypad, and must leap its way to the rightmost lilypad before it can begin its fractions practice.

The initial state of each lilypad *i* is described by a character L_i, which is one of the following:

> "A": Occupied by the Alpha Frog (it's guaranteed that L_i = "A" if and only if *i* = 1)
> "B": Occupied by a Beta Frog
> ".": Unoccupied

At each point in time, one of the following things may occur:

1) The Alpha Frog may leap over one or more lilypads immediately to its right which are occupied by Beta Frogs, and land on the next unoccupied lilypad past them, if such a lilypad exists. The Alpha Frog must leap over at least one Beta Frog; it may not just leap to an adjacent lilypad. Note that, unlike in Leapfrog Ch. 2, the Alpha Frog may only leap to its right.

2) Any Beta Frog may leap to the next lilypad to either its left or right, if such a lilypad exists and is unoccupied.

Assuming the frogs all cooperate, determine whether or not it's possible for the Alpha Frog to ever reach the rightmost lilypad and begin its daily fractions practice.

Input

Input begins with an integer T, the number of days on which the Alpha Frog studies fractions. For each day, there is a single line containing the length-N string L.

Output

For the *i*th day, print a line containing "Case #*i*: " followed by a single character: "Y" if the Alpha Frog can reach the rightmost lilypad, or "N" otherwise.

Constraints

1 ≤ T ≤ 500

2 ≤ N ≤ 5,000

Explanation of Sample

In the first case, the Alpha Frog can't leap anywhere.

In the second case, the Alpha Frog can leap over the Beta Frog to reach the rightmost lilypad.

In the third case, neither the Alpha Frog nor either of the Beta Frogs can leap anywhere.

In the fourth case, if the first Beta Frog leaps one lilypad to the left, and then the second Beta Frog also leaps one lilypad to the left, then the Alpha Frog can leap over both of them to reach the rightmost lilypad.

Example Input

```
8
A.
AB.
ABB
A.BB
A..BB..B
A.B..BBB.
AB.........
A.B..BBBB.BB
```

Example Output

```
Case #1: N
Case #2: Y
Case #3: N
Case #4: Y
Case #5: N
Case #6: Y
Case #7: N
Case #8: Y
```

B. Leapfrog: Ch. 2

This problem statement differs from that of Leapfrog Ch. 1 in only one spot, highlighted in bold below.

A colony of frogs peacefully resides in a pond. The colony is led by a single Alpha Frog, and also includes 0 or more Beta Frogs. In order to be a good leader, the Alpha Frog diligently studies the high art of fractions every day.

There are N lilypads in a row on the pond's surface, numbered 1 to N from left to right, each of which is large enough to fit at most one frog at a time. Today, the Alpha Frog finds itself on the leftmost lilypad, and must leap its way to the rightmost lilypad before it can begin its fractions practice.

The initial state of each lilypad *i* is described by a character L_i, which is one of the following:

> "A": Occupied by the Alpha Frog (it's guaranteed that L_i = "A" if and only if *i* = 1)
> "B": Occupied by a Beta Frog
> ".": Unoccupied

At each point in time, one of the following things may occur:

1) The Alpha Frog may leap over one or more lilypads immediately to either its left or right which are occupied by Beta Frogs, and land on the next unoccupied lilypad past them, if such a lilypad exists. The Alpha Frog must leap over at least one Beta Frog; it may not just leap to an adjacent lilypad. Note that, unlike in Leapfrog Ch. 1, the Alpha Frog may leap to either its left or right.

2) Any Beta Frog may leap to the next lilypad to either its left or right, if such a lilypad exists and is unoccupied.

Assuming the frogs all cooperate, determine whether or not it's possible for the Alpha Frog to ever reach the rightmost lilypad and begin its daily fractions practice.

Input

Input begins with an integer T, the number of days on which the Alpha Frog studies fractions. For each day, there is a single line containing the length-N string L.

Output

For the *i*th day, print a line containing "Case #*i*: " followed by a single character: "Y" if the Alpha Frog can reach the rightmost lilypad, or "N" otherwise.

Constraints

1 ≤ T ≤ 500

2 ≤ N ≤ 5,000

Explanation of Sample

In the first case, the Alpha Frog can't leap anywhere.

In the second case, the Alpha Frog can leap over the Beta Frog to reach the rightmost lilypad.

In the third case, neither the Alpha Frog nor either of the Beta Frogs can leap anywhere.

In the fourth case, if the first Beta Frog leaps one lilypad to the left, and then the second Beta Frog also leaps one lilypad to the left, then the Alpha Frog can leap over both of them to reach the rightmost lilypad.

Example Input

```
8
A.
AB.
ABB
A.BB
A..BB..B
A.B..BBB.
AB.........
A.B..BBBB.BB
```

Example Output

```
Case #1: N
Case #2: Y
Case #3: N
Case #4: Y
Case #5: Y
Case #6: Y
Case #7: N
Case #8: Y
```

C. Mr. X

"There's nothing more important than x!", laughs Mr. X as he explains a Boolean expression involving a variable x to you and your classmates. He can't go 5 minutes teaching Boolean algebra without making at least one such "joke"...

In Mr. X's class, you've been learning about single-variable Boolean expressions, which are made up of the variable x (and its negation), Boolean constants (True/False), and binary Boolean operators. A valid expression is a string in one of the following two forms:

1) A single term, which is one of the following four characters:

> "x": The variable x
> "X": The negation of the variable x
> "0": The constant False
> "1": The constant True

2) A binary operator joining two valid expressions in the format "`([expression][operator][expression])`", with the operator being one of the following three characters:

> "|": The OR operator (evaluating to True when at least one of its operands is True)
> "&": The AND operator (evaluating to True when both of its operands are True)
> "^": The XOR operator (evaluating to True when exactly one of its operands is True)

For example, the following expressions are valid:

```
"1"
"(x^0)"
"((X&0)|x)"
```

While the following expressions are invalid:

```
"(1)"
"x^0"
"(X&0|x)"
```

An upcoming test will feature a valid expression E in the above format, which must be evaluated for a certain value of x. However, you've been getting tired of Mr. X and his lame jokes about the importance of x, so you're planning on hacking into his test files and changing the expression so as to make x irrelevant! In particular, you'd like to modify as few characters as possible in E such that it ends up still being a valid expression, but such that its overall value doesn't depend on the value of the variable x. You may only change characters in-place into different characters — you may not insert or delete characters.

For example, the expression "`(X|(0&x))`" evaluates to True if x is False, and False if x is True. If it were to be changed into "`((X&0)&1)`" (by modifying its 2nd, 3rd, 4th, 6th, 7th, and 8th characters), then it would evaluate to False regardless of x's value. Though, it's also possible to make its value independent of x by modifying fewer than 6 of its characters.

Given an expression E, what's the minimum number of characters which must be modified? It's possible that no characters may need to be modified at all.

Input

Input begins with an integer T, the number of tests. For each test, there is a line containing the expression E.

Output

For the *i*th test, print a line containing "Case #*i*: " followed by a single integer, the minimum number of characters to modify in E such that the result is a valid expression whose value doesn't depend on the value of x.

Constraints

1 ≤ T ≤ 500

1 ≤ |E| ≤ 300

Explanation of Sample

The first expression can, for example, be changed to "`1`" (and would then always evaluate to True).

The second expression can be left unchanged (as it always evaluates to False).

The third expression can be left unchanged (as it always evaluates to True).

The fourth expression can, for example, be changed to "`((0^(X&X))|x)`" (and would then always evaluate to True).

Example Input

```
4
X
0
(x|1)
((1^(X&X))|x)
```

Example Output

```
Case #1: 1
Case #2: 0
Case #3: 0
Case #4: 1
```

D. Trees as a Service

Carlos is hoping to strike it rich with his new consulting firm, Carlos Structures Industries (CSI). With a solid mission statement and a can-do attitude, he's sure to succeed in the wide world of bespoke data structures.

"Our vision is to synergistically leverage our core competencies to deliver competitive market solutions that assertively meet our clients' needs."

His first client, the well-known translation firm Treehouse, is looking to update their logo. Treehouse wants their logo to be a rooted tree with N nodes numbered 1 through N. They have a rough idea of what the tree should look like, but they want Carlos to finish fleshing it out. In particular, they have M requirements, the *i*th of which states that the lowest common ancestor of nodes X_i and Y_i must be node Z_i.

Carlos's goal is to find any valid tree consistent with all of these requirements if possible, or to determine that no such tree exists.

Input

Input begins with an integer T, the number of tree designs. For each design, there is first a line containing the space-separated integers N and M. Then M lines follow, the *i*th of which contains the space-separated integers X_i, Y_i, and Z_i.

Output

For the *i*th design, print a line containing "Case #*i*: " followed by a description of a valid rooted tree if possible, or the string "Impossible" if no valid tree exists. A tree description is N space-separated integers, the *j*th of which is node *j*'s parent (or 0 if node *j* is the root).

Constraints

$1 \le T \le 100$

$2 \le N \le 60$

$1 \le M \le 120$

$1 \le X_i, Y_i, Z_i \le N$

$X_i \ne Y_i$

Explanation of Sample

In the first case, the LCA of nodes 1 and 2 in the chosen tree is 3, as required. This is the only valid output for this case.

In the third case, "2 3 0 3" would also be accepted, while no other outputs would be.

In the fifth and sixth cases, multiple possible outputs would be accepted.

Example Input

```
6
3 1
1 2 3
3 3
1 2 2
2 3 3
3 1 1
4 2
2 1 2
1 4 3
6 3
2 4 3
6 5 4
1 2 6
7 4
7 3 5
4 1 2
6 3 6
6 4 5
12 9
1 5 7
11 2 6
9 4 12
8 12 6
10 1 7
4 3 12
3 10 6
8 11 8
2 5 10
```

Example Output

```
Case #1: 3 3 0
Case #2: Impossible
Case #3: 3 0 2 3
Case #4: Impossible
Case #5: 2 0 6 5 2 5 5
Case #6: 7 10 12 12 10 0 6 6 12 7 8 6
```