Problem A: Foregone

Someone just won the Code Jam lottery, and we owe them **N** jamcoins! However, when we tried to print out an oversized check, we encountered a problem. The value of **N**, which is an integer, includes at least one digit that is a `4`... and the `4` key on the keyboard of our oversized check printer is broken.

Fortunately, we have a workaround: we will send our winner two checks for positive integer amounts A and B, such that neither A nor B contains any digit that is a `4`, and A + B = **N**. Please help us find any pair of values A and B that satisfy these conditions.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow; each consists of one line with an integer **N**.

Output

For each test case, output one line containing `Case #x: A B`, where `x` is the test case number (starting from 1), and `A` and `B` are positive integers as described above.

It is guaranteed that at least one solution exists. If there are multiple solutions, you may output any one of them. (See "What if a test case has multiple correct solutions?" in the Competing section of the FAQ. This information about multiple solutions will not be explicitly stated in the remainder of the 2019 contest.)

Limits

$1 \le \mathbf{T} \le 100$.
Time limit: 10 seconds per test set.
Memory limit: 1GB.
At least one of the digits of **N** is a 4.

Test set 1 (Visible)

$1 < \mathbf{N} < 10^5$.

Test set 2 (Visible)

$1 < \mathbf{N} < 10^9$.

Solving the first two test sets for this problem should get you a long way toward advancing. The third test set is worth only 1 extra point, for extra fun and bragging rights!

Test set 3 (Hidden)

$1 < \mathbf{N} < 10^{100}$.

Sample

Input	Output
3 4 940 4444	Case #1: 2 2 Case #2: 852 88 Case #3: 667 3777

In Sample Case #1, notice that A and B can be the same. The only other possible answers are `1 3` and `3 1`.

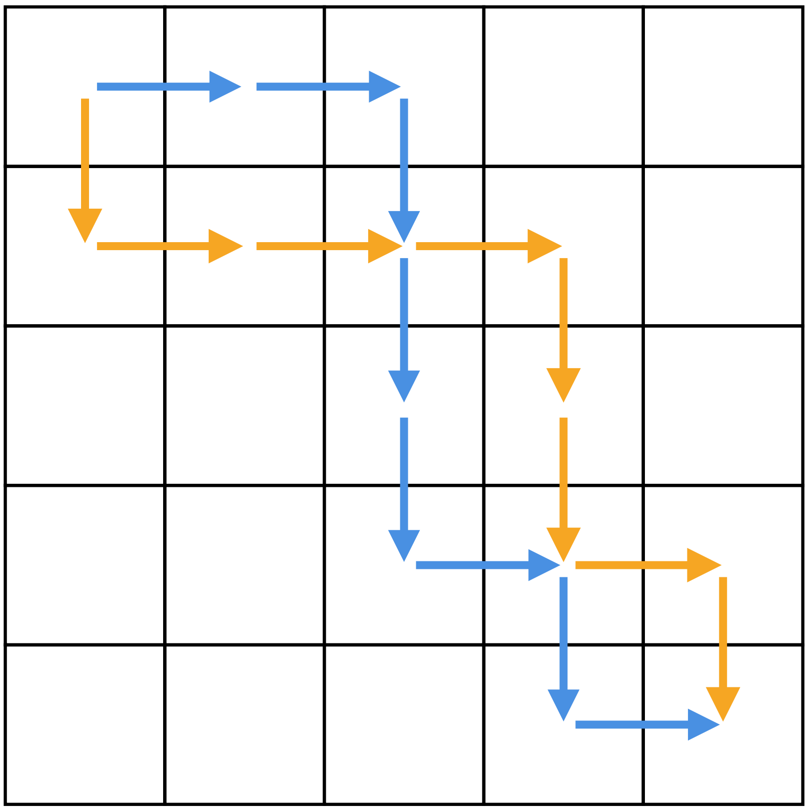
Problem B: You Can Go Your Own Way

You have just entered the world's easiest maze. You start in the northwest cell of an **N** by **N** grid of unit cells, and you must reach the southeast cell. You have only two types of moves available: a unit move to the east, and a unit move to the south. You can move into any cell, but you may not make a move that would cause you to leave the grid.

You are excited to be the first in the world to solve the maze, but then you see footprints. Your rival, Labyrinth Lydia, has already solved the maze before you, using the same rules described above!

As an original thinker, you do not want to reuse any of Lydia's moves. Specifically, if her path includes a unit move from some cell A to some adjacent cell B, your path cannot also include a move from A to B. (However, in that case, it is OK for your path to visit A or visit B, as long as you do not go from A to B.) Please find such a path.

In the following picture, Lydia's path is indicated in blue, and one possible valid path for you is indicated in orange:



Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow; each case consists of two lines. The first line contains one integer **N**, giving the dimensions of the maze, as described above. The second line contains a string **P** of 2**N** - 2 characters, each of which is either uppercase `E` (for east) or uppercase `S` (for south), representing Lydia's valid path through the maze.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is a string of 2**N** - 2 characters each of which is either uppercase `E` (for east) or uppercase `S` (for south), representing your valid path through the maze that does not conflict with Lydia's path, as described above. It is guaranteed that at least one answer exists.

Limits

1 ≤ **T** ≤ 100.
Time limit: 15 seconds per test set.
Memory limit: 1GB.
P contains exactly **N** - 1 `E` characters and exactly **N** - 1 `S` characters.

Test set 1 (Visible)

2 ≤ **N** ≤ 10.

Test set 2 (Visible)

2 ≤ **N** ≤ 1000.

Test set 3 (Hidden)

For at most 10 cases, 2 ≤ **N** ≤ 50000.
For all other cases, 2 ≤ **N** ≤ 10000.

Sample

Input

```
2
2
SE
5
EESSSESE
```

Output

```
Case #1: ES
Case #2: SEEESSES
```

In Sample Case #1, the maze is so small that there is only one valid solution left for us.

Sample Case #2 corresponds to the picture above. Notice that it is acceptable for the paths to cross.

Problem C: Pylons

Our Battlestarcraft Algorithmica ship is being chased through space by persistent robots called Pylons! We have just teleported to a new galaxy to try to shake them off of our tail, and we want to stay here for as long as possible so we can buy time to plan our next move... but we do not want to get caught!

This galaxy is a flat grid of **R** rows and **C** columns; the rows are numbered from 1 to **R** from top to bottom, and the columns are numbered from 1 to **C** from left to right. We can choose which cell to start in, and we must continue to jump between cells until we have visited each cell in the galaxy *exactly* once. That is, we can never revisit a cell, including our starting cell.

We do not want to make it too easy for the Pylons to guess where we will go next. Each time we jump from our current cell, we must choose a destination cell that does not share a row, column, or diagonal with that current cell. Let (i, j) denote the cell in the i-th row and j-th column; then a jump from a current cell (r, c) to a destination cell (r', c') is invalid if and only if any of these is true:

- r = r'
- c = c'
- r - c = r' - c'
- r + c = r' + c'

Can you help us find an order in which to visit each of the **R** × **C** cells, such that the move between any pair of consecutive cells in the sequence is valid? Or is it impossible for us to escape from the Pylons?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line containing two integers **R** and **C**: the numbers of rows and columns in this galaxy.

Output

For each test case, output one line containing `Case #x: y`, where `y` is a string of uppercase letters: either `POSSIBLE` or `IMPOSSIBLE`, according to whether it is possible to fulfill the conditions in the problem statement. Then, if it is possible, output **R** × **C** more lines. The i-th of these lines represents the i-th cell you will visit (counting starting from 1), and should contain two integers r_i and c_i: the row and column of that cell. Note that the first of these lines represents your chosen starting cell.

Limits

Time limit: 20 seconds per test set.
Memory limit: 1GB.

Test set 1 (Visible)

T = 16.
2 ≤ **R** ≤ 5.
2 ≤ **C** ≤ 5.

Test set 2 (Hidden)

1 ≤ **T** ≤ 100.
2 ≤ **R** ≤ 20.
2 ≤ **C** ≤ 20.

Sample

Input	Output

`2`
`2 2`
`2 5`
	`Case #1: IMPOSSIBLE`
	`Case #2: POSSIBLE`
	`2 3`
	`1 1`
	`2 4`
	`1 2`
	`2 5`
	`1 3`
	`2 1`
	`1 5`
	`2 2`
	`1 4`

In Sample Case #1, no matter which starting cell we choose, we have nowhere to jump, since all of the remaining cells share a row, column, or diagonal with our starting cell.

In Sample Case #2, we have chosen the cell in row 2, column 3 as our starting cell. Notice that it is fine for our final cell to share a row, column, or diagonal with our starting cell. The following diagram shows the order in which the cells are visited:

2	4	6	10	8
7	9	1	3	5

Problem D: 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 $\mathbf{L_i}$, which is one of the following:

- "`A`": Occupied by the Alpha Frog (it's guaranteed that $\mathbf{L_i}$ = "`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.

Sample Input

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

Sample 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
```

Problem E: 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 $\mathbf{L_i}$, which is one of the following:

- "`A`": Occupied by the Alpha Frog (it's guaranteed that $\mathbf{L_i}$ = "`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.

Sample Input

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

Sample 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
```

Problem F: 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).

Sample Input

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

Sample Output

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