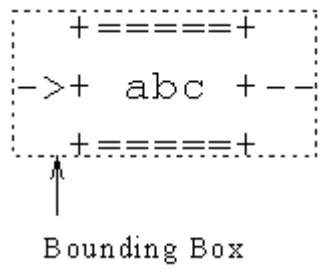
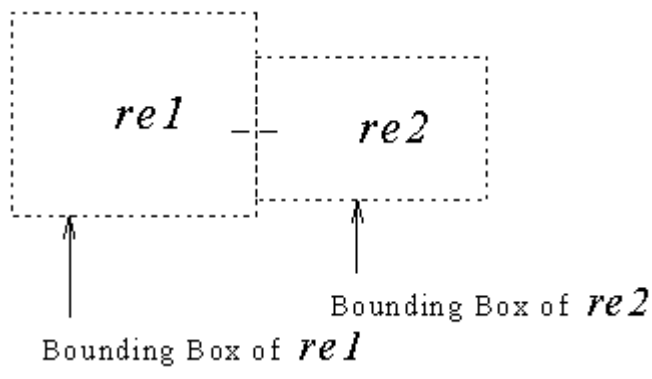
Given a simple syntax for describing regular expressions, one can find a graphical representation for a given regular expression using ASCII characters like '`-`', ' ', '`+`', and '`/`'. The syntax we use can be drawn by using four different patterns:

1. `"abc"` is the terminal string '`abc`', represented as



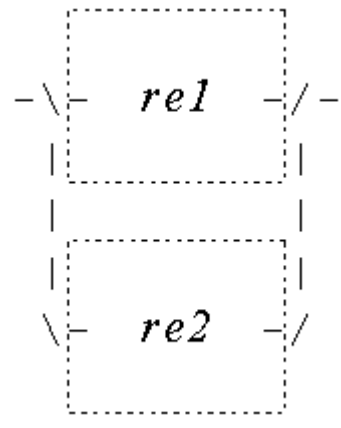
   Note that the graphical representation of every expression has a bounding box. This is the smallest rectangle that surrounds the graphic.

2. ($re1$ $re2$) is a sequence of first expression $re1$, then expression $re2$, represented as



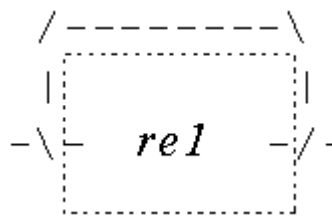
   The two expressions $re1$ and $re2$ have to be concatenated such that the bounding boxes of the two expressions touch and such that the '`-`' on the right of $re1$ matches the '`-`' on the left of $re2$.

3. {$re1$ $re2$} represents alternatives, either $re1$ or $re2$, represented as



   the number of '`|`' characters that has to be added depends on the shapes of $re1$ and $re2$. There has to be exactly one straight blank line between the bounding box of the graphical representation of $re1$ and the bounding box of $re2$. If necessary, also a number of '`-`' characters has to be added on the right side of $re1$ or $re2$, to make the drawing possible. Note that the '`-`' on the left of $re1$ and $re2$ matches the '\' character and that the '`-`' on the right of $re1$ and $re2$ matches the '\' character in the drawing.

4. [$re1$] is a 1-or-more repetition of $re1$, represented as



   Note that the '`-`' on the left of $re1$ matches the '\' character and that the '`-`' on the right of $re1$ matches the '`/`' character in the drawing.

For example the graphical representation for the regular expression {`("f" "bar") ["c"]`} looks like:

```
      +===+    +======+
---\->+ f +--->+ bar +--/-->
   |  +===+    +======+  |
   |                      |
   | /---------\          |
   | |  +===+  |          |
   \-\->+ c +--/--------/
        +===+
```

Write a program that reads syntax rules and prints the size of the graphical representation. For esthetic reasons, the entire graphic has a '`--`' on the left and a '`->`' on the right.

Input

The input consists of a line holding the number of test cases, followed by the input expressions (one per line). The expressions are formatted according to the following grammar:

expression :: sequence — alternatives — repetition — terminal
sequence :: (ws expression expression) ws
alternatives :: { ws expression expression } ws
repetition :: [ws expression] ws
terminal :: " character* " ws
ws :: (¡*space*¿ — ¡*tab*¿)*
character :: ¡*any character except* " *and control-characters (ASCII 0..31)*¿

Note that the grammar is specified according to the following notational conventions:

x y	sequence: x followed by y
$x\|y$	choice: x or y
x^*	repetition: zero or more occurrences of x
<>	used for describing a character

Output

For each expression, output a line of the form 'X`x`Y' with X and Y the width and height of the graphical representation of that expression.

Sample Input

```
1
{("f" "bar") ["c"]}
```

Sample Output

```
28x8
```