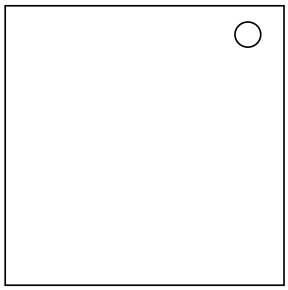
On the Subject of Levenshtein Distance



Close is further than far if you travel by car.

- Two words of lengths 4-8 will be displayed on top two displays.
- There are ten buttons with digits that you can use to type in your answer, which will be displayed on the bottom display. This number cannot have more than two digits.
- There are also three other buttons with labels "SUB" (use to submit your answer), "CLR" (use to clear the bottom display) and "DEL" (use to delete the last digit displayed on the bottom display if there are any).
- Pressing any other button than the one with the label "SUB" will never strike you.
- Pressing the button with the label "SUB" will solve the module if the number displayed on the bottom display is the correct answer.
- The module will always strike if you press the button with the label "SUB" while there's nothing displayed on the bottom display and while the module is not solved yet.
- The module does not reset after striking, the correct answer never changes.
- To calculate the correct answer, follow the steps below.

- First, calculate the Levenshtein Distance of the words displayed on the top two displays. Refer to *Appendix Levenshtein Distance* for instructions. Call the result LD.
- Multiply LD by the number of batteries present on the bomb if there is at least one. Otherwise, multiply LD by 1. You now have X.
- Then multiply the number of port types with the number of indicators present on the bomb and add this product to X to get Y.
- If the sum of serial number digits contains any digit that is present in the serial number, subtract 1 from Y to get Z.
- The correct answer is the result of moduling the number Y by 100.
  - Modulo by 100 is done by repeatedly adding or subtracting 100 until the number is in range from 0 to 99 inclusively.
  - You can keep Z for good luck. It is not be needed to solve the module.

- **However,** if the serial number contains the letters L and D, submit raw LD. You are spared of all the modifications.

Appendix — Levenshtein Distance

Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other.

Calculating Levenshtein Distance using recursion

Levenshtein Distance between two words can be calculated **recursively.**
You can use the following recursive function to calculate Levenshtein Distance between words w_1 and w_2:

- Let's denote the desired result as LD(w_1, w_2). It is yielded by a function LD(w_1, w_2) that contains the following 4 rules:

    1. If w_1 has no letters, then LD(w_1, w_2) is the number of letters of w_2.
    2. If w_2 has no letters, then LD(w_1, w_2) is the number of letters of w_1.
    3. If the first letters of w_1 and w_2 are the same, LD(w_1, w_2) = LD(v_1, v_2).
    4. Otherwise LD(w_1, w_2) is **one more than** the minimum of the set
        - { LD(v_1, w_2); LD(w_1, v_2); LD(v_1, v_2) }.

- In all above statements, v_1 is w_1 without its first letter and v_2 is w_2 without its first letter.

Calculating Levenshtein Distance using dynamic programming

Alternatively, you can use a method of **dynamic programming,** which uses a matrix to keep track of all known results and therefore no recursion is needed.

Here is an example of a matrix used to calculate LD("sitting", "kitten") following that mentioned method:

		k	**i**	**t**	**t**	**e**	**n**
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	**3**

See the next page for detailed description of the process.

Let's have words w_1 and w_2 of lengths m and n respectively.
Follow the steps below to calculate LD(w_1, w_2) using the previous notation:

1. Create a matrix of $m + 2$ rows and $n + 2$ columns.
2. Starting with the third cell from the top, fill in the leftmost column with w_1, letter by letter going top to bottom.
3. Starting with the third cell from the left, fill in the topmost row with w_2, letter by letter going left to right.
4. Starting with the second cell from the top, fill in the second leftmost column by numbers 0, 1, ..., m going top to bottom.
5. Starting with the third cell from the left, fill in the second topmost row by numbers 1, 2, ..., n left to right.

6. Now, starting in the cell that is in the intersection of the third column from the left and the third row from the top, do the following:
    - If the letter that is in the same row as this cell is **the same** as the letter in the same column as this cell, copy into this cell the number from a cell being diagonally adjacent to this cell via its top left corner. Do not add anything.
    - Otherwise find the **minimum** of the numbers in cells
        - one above this cell,
        - one to the left of this cell,
        - one diagonally adjacent to this cell via its top left corner

      and **add one**. Copy this result into this cell.
7. Repeat step 6 until you fill the matrix, going left to right, top to bottom.
8. The number in the **bottom right corner cell** of the matrix is LD(w_1, w_2).

The algorithm descibed here is being demonstrated on the previous page.
See the table for w_1 = "sitting" and w_2 = "kitten" showing that
LD("sitting", "kitten") = 3.

Note: This whole process can be optimized. For example, it is possible to just use two rows of a matrix at a time to save space.

For more reference, see the Wikipedia page.
(https://en.wikipedia.org/wiki/Levenshtein_distance)