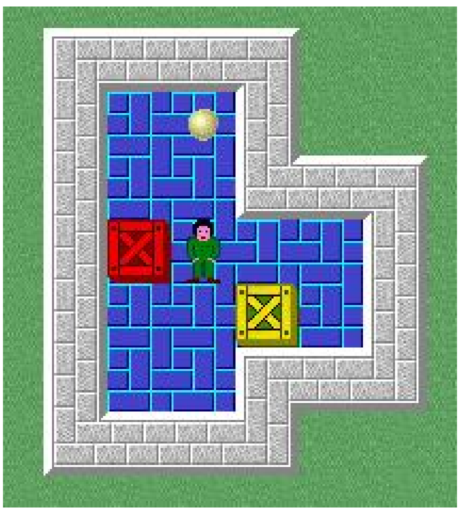
Sokoban is a popular computer game originally created in 1982 by Hiroyuki Imabayashi at the Japanese company Thinking Rabbit, Inc. "Sokoban" is Japanese for "warehouse keeper". The idea is that you are a warehouse keeper trying to push boxes to their proper locations in a warehouse. The following picture depicts the game.



The goal of the game is to help the warehouse keeper to push the boxes to their correct destinations in a crowded warehouse, for which a top-view map is available. The drawback is that our friend is not strong enough to lift one box over another and he isn't strong enough to pull them either. He can only push one box at a time, two boxes at once are already too heavy. Moreover, a passage is just as narrow as a box, so if a box is blocking a path, he cannot go "around" it just by squeezing himself through between the box and the wall, he'll need to find a clear path to reach the other side. And of course, our friend cannot climb over boxes, either.

Therefore, in order to win this game, a player needs to guide the warehouse keeper to move the right way through the labyrinth of walls and boxes and to push all the boxes to the intended target spots in the warehouse.

Sokoban maps with different levels of difficulty are available on the Internet. For instance, the above figure is given by the following map:

```
####
# .#
#  ###
#*@  #
#  $ #
#  ###
####
```

The meaning of each character in this map is the following: '`#`' denotes a wall, space denotes an empty square, '`.`' denotes an empty goal square, '`$`' denotes a box on a square (a yellow box in the picture), '`*`' denotes a box on a goal square (a red box in the picture), '`@`' denotes the warehouse keeper position on an empty square, '`+`' denotes the warehouse keeper position on a goal square.

On these maps, these are the actions that our little friend can perform:

- Move to an adjacent empty square (left, right, up, down, but not in diagonal). Each of these actions is called a "move".
- Push an adjacent box to an empty square (left, right, up, down, but not in diagonal). Each of these actions is called a "push". Remark that in order to push a box in some direction, the destination square must be empty.

For instance, a possible way to solve the previous puzzle with 8 pushes is given in the following 9 diagrams (read them from left to right, and observe that only pushes are shown, moves are not displayed).

```
  ####      ####      ####      ####      ####      ####      ####      ####      ####
  # .#      # .#      # .#      # .#      # .#      # .#      # .#      # .#      # *#
  #  ###    #$ ###    #$ ###    #$ ###    #$ ###    #@ ###    #  ###    # $###    # @###
  #*@  #    #+   #    #.   #    #.$  #    #.@$ #    #* $ #    #*$@ #    #*@  #    #*   #
  #  $ #    #  $ #    # $@ #    # @  #    #    #    #    #    #    #    #    #    #    #
  #  ###    #  ###    #  ###    #  ###    #  ###    #  ###    #  ###    #  ###    #  ###
  ####      ####      ####      ####      ####      ####      ####      ####      ####
```

In fact, there is no way to solve this sokoban map in less than 8 pushes.

Your task is to write a program that, given an integer k and a sokoban map, indicates whether or not it is possible to solve the game with k or less pushes (the number of moves is not taken into account).

Input

Input consists of zero or more test cases. Each test case consists of a line containing an integer k followed by a sokoban map. You can assume that $1 \leq k \leq 30$ and that the size of the map is equal or smaller than 16×16. All lines may not have the same length. A blank line separates two consecutive test cases.

Output

For every test case, use a separate line to print '`YES`' if the map can be solved with k or less pushes, or print '`NO`' otherwise.

Sample Input

```
7
############
#  $  @  . #
############

8
####
# .#
#  ###
#*@  #
#  $ #
#  ###
####
```

Sample Output

```
NO
YES
```