Problem A. Assignment Algorithm

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

A low-budget airline is designing a sophisticated algorithm that will assign more desirable seats to passengers who buy tickets earlier. Their airplane has r rows of seats, where r is an even integer. There are also 3 *exit rows* in the airplane; those rows do not contain any seats but only provide access to the emergency exits. One exit row is in the very front of the airplane (before the first row of seats), one in the very back (behind the last row of seats) and one right in the middle. The rows are numbered with integers 1 through $r + 3$ with row numbers increasing from the front to the back of the airplane. Rows numbered 1, $r/2 + 2$ and $r + 3$ are exit rows while all the other rows are seat rows.

The seating configuration is "3–3–3" — each seat row contains three groups of three seats with the passenger aisles between the groups. Seats in the same row are denoted with consecutive letters left to right corresponding to the pattern "`ABC.DEF.GHI`".

When a passenger purchases a ticket, she is assigned a seat according to the following rules:

1. If there is an empty seat in a row directly after an exit row, all other rows are ignored in the following step (but they are not ignored when balancing the airplane in the last step).
2. First, we select a seat row with the largest number of empty seats. If there are multiple such rows, we select the one closest to an exit row (distance between rows a and b is simply $|a - b|$). If there are still multiple such rows, we select the one with the lowest number.
3. Now, we consider empty seats in the selected row and select one with the highest *priority*. Seat priorities, from highest to lowest are as follows:
   (a) Aisle seats in the middle group (`D` or `F`).
   (b) Aisle seats in the first and third group (`C` or `G`).
   (c) Window seats (`A` or `I`).
   (d) Middle seat in the middle group (`E`).
   (e) Other middle seats (`B` or `H`).

   If there are two empty seats with the same highest priority, we consider the balance of the entire airplane. The airplane's *left side* contains all seats with letters `A`, `B`, `C` or `D`, while the *right side* contains all seats with letters `F`, `G`, `H` or `I`. We select an empty seat in the side with more empty seats. If both sides have the same number of empty seats, we select the seat in the left side of the airplane.

Some of the airplane's seats are already reserved (possibly using a completely different procedure from the one described above). Determine the seats assigned to the next n passengers purchasing a ticket.

Input

The first line contains two integers r and n ($2 \le r \le 50$, $1 \le n \le 26$) — the number of seat rows in the airplane (always an even integer) and the number of new passengers purchasing tickets. The following $r + 3$ lines contain the current layout of the airplane. The j-th line contains exactly 11 characters — the layout of the row j of the airplane. Exit rows and aisles are denoted with the "`.`" characters. The "`#`" character denotes a reserved seat, while the "`-`" character denotes a seat that is currently empty. You may assume there will be at least n empty seats in the airplane.

Output

Output $r + 3$ lines containing the final layout of the airplane. The layout should be exactly the same as in input with the following exception: the seat assigned to the j-th passenger purchasing a ticket should be denoted with the j-th lowercase letter of the English alphabet.

Examples

standard input	standard output
2 17 .... ---.#--.--- .... ---.---.---	.... hnd.#lb.fpj .... kqg.cma.eoi ....
6 26 .... ---.---.### #-#.---.--- ---.###.--- .... ---.###.--- #--.#-#.--# #--.--#.#-#	.... gke.aic.### #-#.mzo.r-v x-p.###.n-t .... fjb.###.dlh #-s.#-#.w-# #-u.qy#.#-# ....