Jigsaw

Input: `jigsaw.in`

He's been at it again. The annoying little kid so well known to contest participants (let's call him Donaldo, although his name is irrelevant), has peeled the pictures from all of his jigsaw pieces. You must write a program that solves the jigsaw puzzle, reassembling the pieces into a rectangle, so that he can draw a new picture on the front.

The shapes of the jigsaw pieces have all been digitized with a low resolution scanner, and the bitmaps converted into easy-to-process text. A complete jigsaw consists of no more than 20 pieces, and although those pieces may be extremely irregular in shape, it is guaranteed that they can be reassembled into a rectangle in only one way. Also, through a stroke of good luck, they are all right side up and correctly oriented, so there is no need to be concerned with possible rotations or reflections of the pieces.

Here is a sample showing how the four pieces of a small puzzle would be represented. The letters give the shapes of the pieces, the dots represent empty space. The minimum number of dots are added to the representation of each shape in order to make it rectangular.

```
.AA.      .BBBBB...      CC..      DDD...      EEEEEEEEEEEE
AAAA      BBB..BB..      .CC.      DD....      ..........EE
AAA.      BB....BB.      ..CC      DD....      ...........E
          BB...BB..      .CCC      DD....
          .BBBBB..B      CC.C      DDD...
          ..BBBBBBB      ...C      DDDD..
          ....BBB..      .CCC      DDDDDD
```

As you can see, the pieces are very irregular, but in this case, knowing that the result should be a 8 row 12 column rectangle, it is quite easy to reassemble them correctly. In this case, the output from your program should be the following reassembled jigsaw rectangle:

```
EEEEEEEEEEEE
DDDBBBBBCCEE
DDBBBAABBCCE
DDBBAAAABBCC
DDBBAAABBCCC
DDDBBBBBCCBC
DDDDBBBBBBBC
DDDDDDBBBCCC
```

Input:

The input will consist of a sequence of jigsaw specifications. Each jigsaw specification begins with a single line containing three integers, the number of rows, *nr*, $0 < nr \leq 100$ and the number of columns, *nc*, $0 < nc \leq 100$, in the completed puzzle, and the number, $0 < p \leq 20$, of puzzle pieces. There follows a sequence of *p* puzzle piece descriptions. Each puzzle piece description begins with a single line containing two integers between 1 and 100, being the number of rows, *r*, and columns, *c*, in the puzzle piece. This is followed by exactly *r* lines consisting of exactly *c* dots or uppercase letters. For each piece description, only one letter is used: the first piece consists of only dots and '`A`'s; the second (if there is one) consists of only dots and '`B`'s; the third consists of only dots and '`C`'s, etc. The pieces are guaranteed to give a unique solution. After the end of the final jigsaw specification, there is a line containing "`0 0 0`" (three zeros) which should not be processed.

Output:

For each input jigsaw specification, your program should print a line "`JIGSAW` <*n*>", where <*n*> is replaced by the problem number (the first being 1), followed by the rectangle of letters representing the reassembled jigsaw. A blank line should follow the output for each puzzle.

Sample Input:

```
4 3 4
3 2
AA
A.
AA
1 2
BB
1 1
C
2 3
..D
DDD
15 24 4
8 21
AAAA.....AAA.........
AAAAA...AAAA.........
AAAAA...AAAA...AAAAA.
AAAA.....AAAAAAAAAAAA
AAA.......AAAAAAAAAAA
AAAA.....AAA...AAAAA.
AAAAA...AAAA.........
AAAAAAAAAAAA.........
7 21
.........BBBBBBBBBBB
.........BBBBBBBBBBB
..BBBB...BBBB....BBBB
BBBBBBBBBBBBB....BBBB
BBBBBBBBBBBBB....BBBB
..BBBB...BBBBB..BBBBB
.........BBBB....BBBB
14 12
CCCCCCCCCCCC
CCCCCCCCCCCC
CCCCC....CCC
CCC.........
CCC.........
CCCCC....CCC
CCCCCCCCCCCC
....CCCCC...
.....CCC....
.....CCC....
....CCCCC...
...CCCCCCC..
....CCCCC...
.....CCC....
13 12
....DDDD....
....DDDD....
....DDDD....
.....DD.....
....DDDD....
DDDDDDDDDDDD
DDDDDDDDDDDD
DDD.....DDDD
.........DDD
.........DDD
DDD.....DDDD
DDDDDDDDDDDD
DDDDDDDDDDDD
0 0 0
```

Output Corresponding to Sample Input:

```
JIGSAW 1
AAC
ABB
AAD
DDD

JIGSAW 2
CCCCCCCCCCCCBBBBBBBBBBBB
CCCCCCCCCCCCBBBBBBBBBBBB
CCCCCBBBBCCCBBBBDDDDBBBB
CCCBBBBBBBBBBBBBDDDDBBBB
CCCBBBBBBBBBBBBBDDDDBBBB
CCCCCBBBBCCCBBBBBDDBBBBB
CCCCCCCCCCCCBBBBDDDDBBBB
AAAACCCCCAAADDDDDDDDDDDD
AAAAACCCAAAADDDDDDDDDDDD
AAAAACCCAAAADDDAAAAADDDD
AAAACCCCCAAAAAAAAAAADDD
AAACCCCCCCAAAAAAAAAADDD
AAAACCCCCAAADDDAAAAADDDD
AAAAACCCAAAADDDDDDDDDDD
AAAAAAAAAAADDDDDDDDDDD
```