TORNJEVI

We are being attacked on a map represented by a rectangular grid of R×S squares. The attackers are barefoot robbers, and we use small cannons on small wooden towers to defend ourselves.

Each tower is equipped with **two cannons**, placed to fire in a 90 degree angle. More precisely, cannons on one tower can be set to fire in one of the following four configurations:

1. fire left and down;
2. fire down and right;
3. fire right and up;
4. fire up and left.

A cannon ball that hits the attacker **destroys him and continues to fly** in the same direction. A cannon ball which hits a castle stops and does no damage to the castle (because castles are big and strong). But, when a cannon ball hits a tower, it destroys it (because towers are small and fragile).

We want to turn the cannons on the towers so that, when we fire exactly one shot from **every cannon**, **we destroy all the attackers**, and **all our towers remain undamaged.**

Input

The first line contains two integers R and S (1 ≤ R, S ≤ 100), the dimensions of the map.

The next R lines contain S characters each, the map.

Each character on the map can be the uppercase letter 'T' (tower), lowercase letter 'n' (attacker), the character '#' (castle) or the character '.' (empty).

Note: There will always be a solution, although not necessarily unique.

Output

Output the map in the same format as in the input, replacing 'T' characters with the orientations of the cannons – each tower should be replaced with one of the digits '1', '2', '3' or '4', corresponding to the four orientations as described above.

Examples

Input:
```
9 13
.............
...........n.
.n.T..nnnn#..
.............
.T#n..n....T.
.............
.n.T..T....n.
.............
```

Input:
```
5 9
.n..T..n.
.T..n....
.n..#..n.
....n..T.
.n..T..n.
```

Output:
```
.n..4..n.
```

Input:
```
9 8
n.Tnnnnn
nnnnnnTn
nTnnnnnn
nnnnTnnn
Tnnnnnnn
..#nnTnn
nnnnnnnT
nnnTn.n.
```

……n……

Output:

……
……n.
.n.3..nnnn#..
……
.4#n..n.…4.
……
.n.1..2.…n.
……
……n……

.2..n.…
.n..#..n.
….n..4.
.n..3..n.

.nTnnnnn

Output:

n.3nnnnn
nnnnnn1n
n2nnnnnn
nnnn1nnn
3nnnnnnn
..#nn4nn
nnnnnnn4
nnn4n.n.
.n3nnnnn