Bunnies

Filename: bunnies

The Problem

Jill, the new lab assistant has made new cages for the lab rabbits, Peter and Cottontail. Unfortunately, these new cages are more like mazes with walls in every direction. Help Arup determine if the rabbits are in the same section of the cage or different sections. Each rabbit can hop one square directly up, down, right or left. They can hop as many times as they want from one free square to another. Each rabbit must stay on the grid.

The Input

Each input will start with a single integer T ($1 \leq T \leq 100$) on the first line. The number T will denote the number of test cases that follow. Each test case will begin with two integers, R and C ($1 \leq R, C \leq 10$) separated by a space. Each of the next R lines will contain C characters of either '_','#', 'P' or 'C' (Quotes for clarity). This will form a grid that represents the cage. A '_' represents a cell free of obstructions, '#' represents a wall, 'P' is Peter's location and 'C' is Cottontail's location. Each grid is guaranteed to have one and only one P and C.

The Output

For each test case output a single line containing "yes" if Peter and Cottontail are in the same section of the cage and "no" if they are not in the same section of the cage.

Sample Input

```
4
2 2
P#
#C
2 2
P_
C_
8 7
__P____
####_##
_____#_
_____#C
##_###_
_____#_
___#_#_
___#___
5 7
__P____
####_##
_____#_
_____#C
##_###_
```

Sample Output

```
no
yes
yes
no
```

Dragon Fire

Filename: `fire`

Andrew was playing the newest turn-based RPG Apples and Dragons. After playing for some hours, he has reached a level full of obstacles (and, of course, dragons!) that has proven to be absurdly difficult. Every now and then, the dragons spew out a torrent of fire which instantly results in death. Frustrated after many attempts, Andrew has started to study the level in depth and has observed a few facts.

The first fact is that these dragons have certain cues in their animation so Andrew can determine when they will spew out their flames of instant death. When they do so, all the dragons will also fire at once. The second fact is that not *all* of the squares on the map will be covered by the fire, meaning it is possible to survive the turn as long as Andrew doesn't have his character standing on a square that will be covered by fire when the dragons let loose! Lastly, Andrew has uncovered the shooting mechanism of dragons and how fire spreads. He has stored the mechanism as well as crucial game rules in the notes below:

1. The map is a two-dimensional rectangular grid composed of squares.

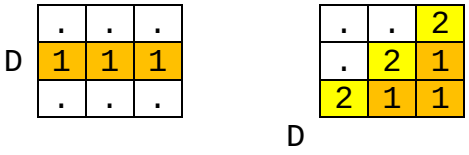
2. Each dragon occupies one square on the map. No two dragons may be on the same square.

3. Each dragon is facing either left, right, up or down at any point in the game.

4. There are two kinds of fire. Both kinds move quickly and spread to all affected squares instantly, up to the edge of the map.

5. The **first** kind of fire will spread only in the direction in which it is fired.

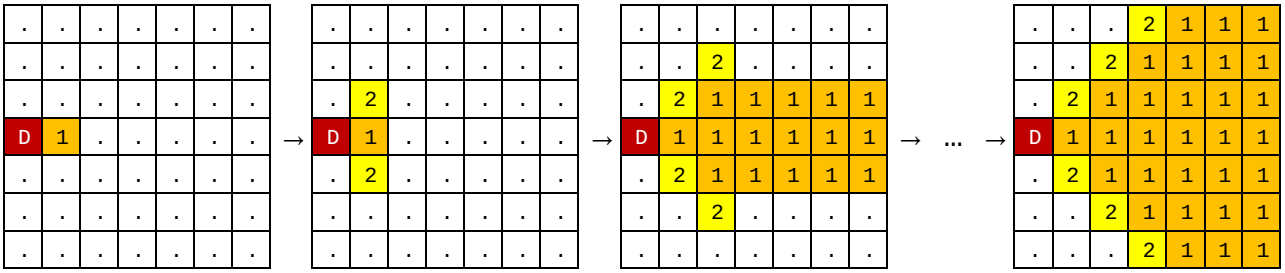
6. The **second** kind of fire will spread in two directions: a *primary* and a *secondary* direction. For each square of fire of the second kind, it will spawn another square of fire of the second kind in the *primary* direction. In addition, it will also spawn a square of fire of the **first** kind in the *secondary* direction.



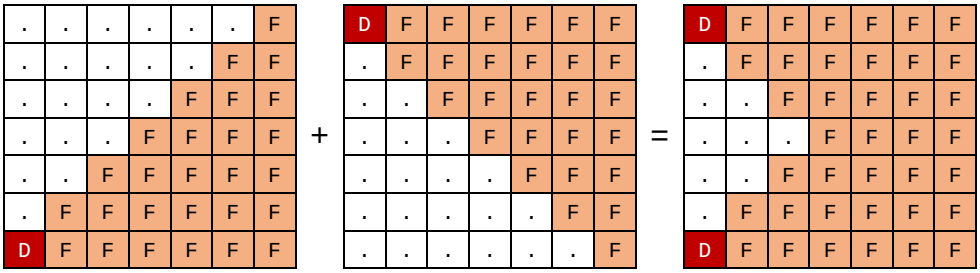
Left Diagram: Fire of the first kind, spreading to the right starting from the "D"

Right Diagram: Fire of the second kind starting at the D in the bottom left, spreading with a primary direction of upper right and secondary direction of right.

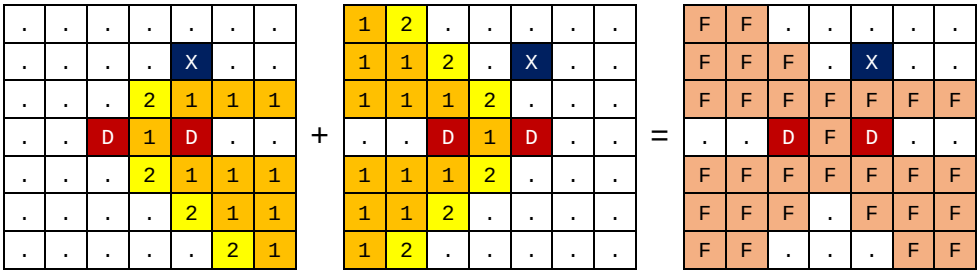
7. When a dragon spews out a stream of fire, it starts by spawning fire of the first kind in the square in front of it. To its relative left and right of that square, the dragon will also attempt to spawn fire squares of the second kind with the primary direction going from the dragon to that square and a secondary direction of the direction the dragon is facing. A dragon will never spawn fire of the second kind with primary and secondary directions other than these. For example, if a dragon is facing right on an empty map, it will spawn fire as such (where `D` represents a dragon, a `1` for a square of fire of the first type, and a `2` for a square of fire of the second type):



8. Fire from a dragon does not interfere with fire from another dragon. Here, two dragons are at the top-left and bottom-left corners, both facing right. In this diagram, all squares covered by *either type* of fire are denoted as `F` and the contribution from each dragon is shown, plus the resulting total fire coverage:



9. Fire of either type cannot spawn at a square that is occupied by another dragon or by an obstacle. This diagram has two dragons facing each other. Each dragon will be in the way of some of the fire of the other dragon. There is also one obstacle (labeled `X`) that will block fire from spreading further (again `F` represents a square of fire of *either* type):



Andrew can determine the moment in time when dragons are about to fire and must find safe squares that will not be touched by fire at all. However, following the notes he has about the map he is playing gives him a headache so he has come to you to write a program to determine which squares on the map are covered by fire.

The Problem:

Given a map, print out the same map updated with the fire spewed out by any and all dragons that appear on it.

The Input:

The input will begin with a positive integer, *m*, representing the number of maps. Each map will start with a line with two positive integers, *h* and *w* ($h \leq 20$; $w \leq 20$), representing the height and the width of the map, respectively. Following this, there will be *h* lines of *w* characters representing the map. Each character will be one of: `X`, `.`, `V`, `^`, `<` or `>`. The characters `<`, `>`, `^` and `V` represents a dragon facing left, right, up and down, respectively; the character `X` is an obstacle and the period (`.`) is an empty square.

The Output:

For each map, first print out "`Map #`*`i`*`:`" on a separate line where *i* is the number of the map in the input (starting at 1). After this, print *h* lines of *w* characters each, representing the map after the dragons spew out fire. If an originally empty square is now covered by any sort of fire from any dragon, print an `F` instead of the original period (`.`). Leave a blank line after the output for each map.

Sample Input:

```
2
4 4
.V..
....
....
...X
3 3
...
..^
...
```

Sample Output:

```
Map #1:
.V..
FFF.
FFFF
FFFX

Map #2:
.FF
..^
...
```

Lex Returns

Filename: lex

The Problem

Lex Luther has liberated Kryptonian crystal technology from that selfish narcissist, Superman. As we all know, Kryptonian technology relies on crystals that can be made to grow into anything simply by adding water. Lex has generously decided to solve the Earth's overpopulation problem by using the crystals to create new landmasses in the ocean. He needs your help to determine how much new land will be created by placing a crystal at a certain point.

The Input

The input will consist of several cases. Each case will describe a section of the planet as a matrix. Each cell (or square) in this matrix will represent 5 square miles of land or water, denoted by '.' Or '~' respectively. There will also be a cell describing the crystal's entry point, 'X', which can be considered a 5 square mile unit of water as well.

The crystal will create land in its initial space of water, as well as any touching space of water. Those spaces will trigger any spaces touching them and so on. Touching in this case is defined as connected by cardinal direction, NOT diagonally.

The first line of the input will contain an integer, *c*, alone on a line to describe the number of cases to analyze. Each case begins with two integers, *m* and *n*, on a line separated by a space. These two integers describe the size of the matrix by number of rows and columns, respectively. The next *m* lines will each contain a string of *n* characters that will only contain the three symbols described above. There will always be one and only one entry point in the input of each case.

You will be guaranteed $0<c<1{,}000$ and $0<m,n<100$.

The Output

For each case, you will output a single integer that describes the square mileage of land created in that case.

Sample Input

```
2
5 5
....~
...~~
....~
...~~
.X~~.
4 6
~~~~~.
~.~~..
~~~X..
..~~~~
```

Sample Output

```
45
80
```

Mines

Filename: `mines`

The Problem:

Acme Mining Co. has been in the mining business for a long time now, and they are one of the biggest in the world. However they are getting worried with all these new companies starting to use computer simulations to determine how much underground volume a blast will clear, they don't want them to gain an unfair competitive advantage, and that is where you come in! Help Acme Mining Co. write a computer simulation of the volume a blast will clear given information about the surrounding structures and spaces.

The simulation will work as follows. The spaces in the mine will be represented as 1's and 0's. A 1 represents solid rock and a 0 represents empty space. When a blast is detonated it will blast away solid rock, but will not do anything in empty space. In this sense, once a blast is detonated it will continue to spread throughout the solid rock of the mine until it hits a pocket(s) of empty space. The blast can only move up, down, left, right, forward, and backward (no diagonals). If a blast is detonated in empty space it does not clear any volume. Each space represents one cubic foot of volume.

The Input:

The first number, *n*, in the input file will be a non-negative integer denoting the dimension of a cubic underground space. Following this will be a single line containing all the info for which three-dimensional coordinates (*x, y, z*) are rock and which are empty space. Valid coordinates for the structure will range from 0 to *n-1* for all three dimensions. The coordinates are ordered in lexicographical order. For example, for a 3 x 3 x 3 space: (0, 0, 0) is the first value, (0, 0, 1) is the next value, (0, 0, 2) is next, (0, 1, 0) is next, ... and the last coordinate would be (2, 2, 2).

Following this is another non-negative integer *t*, which is the number of simulations that will follow. On each of the following *t* lines will be 3 non-negative integers representing the coordinates (*x, y, z*) at which the blast is detonated.

The Output File:

For each simulation, print out a line with the following format:

```
Simulation #k, volume cleared is C cubic feet.
```

where k is the number of the simulation, starting with 1 and C is the number of cubic feet cleared by the blast. Follow the output for each simulation with a blank line.

Sample Input:

```
3
1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 0
2
1 1 0
2 2 2
```

Sample Output:

```
Simulation #1, volume cleared is 16 cubic feet.

Simulation #2, volume cleared is 0 cubic feet.
```