Pacman A*

In the previous game, you performed UCS on the PacMan grid. In this game we use AStar algorithm to reduce the nodes expanded using search using a simple yet efficient heuristic.

AStar on graphs uses the following function

```
cost = d(s,c) + h(c)
```

where s is the source node, c is the node currently expanded and h(c) is the estimation of the cost to reach from c to the destination (food).

In this game, we use manhattan heuristic as an estimate. Given two nodes (r,c) and (r1,c1). The manhattan heuristic is the manhattan distance between the two nodes and is given by

```
|r1 - r | + |c1 - c|
```

Input Format

The first line contains 2 space separated integers which is the position of the PacMan.
The second line contains 2 space separated integers which is the position of the food.
The third line of the input contains 2 space separated integers. Indicating the size of the rows and columns respectively.
This is followed by *row* (r) lines each containing *column* (c) characters. A wall is represented by the character '%' (ascii value 37), PacMan is represented by UpperCase alphabet 'P' (ascii value 80), empty spaces which can be used by PacMan for movement is represented by the character '-' (ascii value 45) and food is represented by the character '.' (ascii value 46)

The top left of the grid is indexed (0,0) and the bottom right of the grid is indexed (r-1,c-1)

The grid is indexed as per matrix convention

For the sake of uniformity across all codes, cost to reach a neighboring node

- 0 if a food is present.
- 1 otherwise.

Output Format

Each cell in the grid is represented by its position in the grid (x,y). PacMan can move only UP, DOWN, LEFT or RIGHT. Your task is to print all the nodes in the shortest path calculated using Astar search between Pacman and Food.

```
 %
%--
 -
```

In the above cell, LEFT and UP are invalid moves. You can either go RIGHT or DOWN. RIGHT is populated first followed by DOWN. i.e., populate the queue UP, LEFT, RIGHT and DOWN order so that UP gets popped first from the queue.

Print the distance 'D' between the source 'P' and the destination '.' calculated using Astar. D+1 lines follow, each line having a node encountered between 'P' and '.' both included. D+1 lines essentially representing the path between source and the destination.

Sample Input

```
35 35
35 1
37 37
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------%-%-%-----------%---%-----%-%
%-%%%%%%%-%-%%%-%-%%%-%%%-%%%%%%%-%-%
%-------%-------%-%-----%-----%-%---%
%%%%%-%%%%%-%%%-%-%-%-%%%-%%%%%-%-%%%
%---%-%-%-%---%-%-%-%---%-%---%-%---%
%-%%%-%-%-%-%%%-%%%%%-%%%-%-%%%-%%%-%
%-------%-----%---%---%-----%-%-%---%
%%%-%%%%%%%%%-%%%%%%%-%%%-%%%-%-%-%-%
%-------------%-------%-%---%-----%-%
%-%-%%%%%-%-%%%-%-%-%%%-%-%%%-%%%-%-%
%-%-%-----%-%-%-%-%-----%---%-%-%-%-%
%-%-%-%%%%%%%-%-%%%%%%%%%-%%%-%-%%%-%
%-%-%-%-----%---%-----%-----%---%---%
%%%-%%%-%-%%%%%-%%%%%-%%%-%%%-%%%%%-%
%-----%-%-%-----%-%-----%-%---%-%-%-%
%-%-%-%-%-%%%-%%%-%%%-%%%-%-%-%-%-%-%
%-%-%-%-%-----------------%-%-%-----%
%%%-%%%%%%%-%-%-%%%%%-%%%-%-%%%-%%%%%
%-------%-%-%-%-----%---%-----%-%---%
%%%%%-%-%-%%%%%%%%%-%%%%%%%%%%%-%-%%%
%---%-%-----------%-%-----%---%-%---%
%-%%%-%%%%%-%%%%%%%%%-%%%%%-%-%-%%%-%
%-%---%------%--------%-----%-------%
%-%-%-%%%%%-%%%-%-%-%-%-%%%%%%%%%%%%%
%-%-%---%-----%-%-%-%-------%---%-%-%
%-%-%%%-%%%-%-%-%-%%%%%%%%%-%%%-%-%-%
%-%---%-%---%-%-%---%-%---%-%-%-----%
%-%%%-%%%-%%%%%-%%%-%-%-%%%%%-%-%%%%%
%-------%---%-----%-%-----%---%-%---%
%%%-%-%%%%%-%%%%%-%%%-%%%-%-%%%-%-%%%
%-%-%-%-%-%-%-%-----%-%---%-%---%-%-%
%-%-%%%-%-%-%-%-%%%%%%%%%-%-%-%-%-%-%
%---%---%---%-----------------%-----%
%-%-%-%-%%%-%%%-%%%%%%%-%%%-%%%-%%%-%
%.%-%-%-------%---%-------%---%-%--P%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Sample Output

sample output

In this example, PacMan is at the position (35,35) and the food is at the position (35,1). The AStar path length between (35,35) and (35,1) is 210. All the nodes encountered between (35,35) and (35,1) both included is printed in the next 211 lines.