Finding a solution of a maze

A maze is simply a text file which has characters which define the “walls” and spaces which define the empty places where a path can be taken. The maze must have a letter S and F specifying the start and finish of the maze path. The program must determine the shortest path between the F and S.

Take the simple maze below:

########################

# #

# F #

# #

################ #

# #

# S #

# #

########################

All of the “white” spaces are cells which can be taken as a path. The solution for this maze is:

########################

# #

# F #

# \*\*\*\*\*\*\*\*\*\* #

################\* #

# \* #

# S\*\*\*\*\* #

# #

########################

The path generated (shown by the \*s) is indeed the shortest path between the S and the F in this case.

Consider the maze as a double array of cells in which only one character can be in a cell at one time. Your program should read the maze into a double array structure like this so it is easier to manipulate.

If you start with the cell that contains the F, you can start putting in numbers that represent the step needed to get to that cell. On all 4 sides of the F cell, you can start by putting a 0 (zero) IF that cell is not already containing a number. Like so:

########################

# 0 #

# 0F0 #

# 0 #

################ #

# #

# S #

# #

########################

If you continue with this method, but increment the step by 1, you can search through your array for all instances of 0’s and fill in the top, bottom, right, and left cells IF they are not already filled with a number or a wall, with the new step like so:

########################

# 101 #

# 10F01 #

# 101 #

################ #

# #

# S #

# #

########################

########################

# 21012 #

# 210F012 #

# 21012 #

################ #

# #

# S #

# #

########################

########################

# 3210123 #

# 3210F0123 #

# 3210123 #

################ #

# #

# S #

# #

########################

…. Continue searching the entire maze, incrementing each time and filling up cells on all sides until you find the S in your travels:

########################

#6543210123456789101112#

#543210F01234567891011 #

#6543210123456789101112#

################101112 #

# 141312111213 #

# S1413121314 #

# 141314 #

########################

In this case, the program found the S on the 14th step.

Now you can clearly see that this maze is solvable because you found the S BEFORE you filled up the entire array with numbers. IF you fill the maze with numbers BEFORE you find an S, then the maze is not solvable, for instance if the wall extended completely to the right, this maze would have stopped at this point. There are no longer any cells that can be filled in.

########################

#6543210123456789101112#

#543210F01234567891011 #

#6543210123456789101112#

########################

# #

# **S** #

# #

########################

Now that you have a clear path from the F to the S, you can start at the S and begin searching backward for each number until you reach 0, which will be where the F is, like so:

########################

#6543210123456789101112#

#543210F01234567891011 #

#654321\*\*\*\*\*\*\*\*\*\*101112#

################\*1112 #

# 141312\*1213 #

# S1\*\*\* 1314 #

# 141314 #

########################

Now, depending on how you implement the algorithm, you may or may not have to remove the numbers from your array and print out the result.

########################

# #

# F #

# \*\*\*\*\*\*\*\*\*\* #

################\* #

# \* #

# S\*\*\*\*\* #

# #

########################