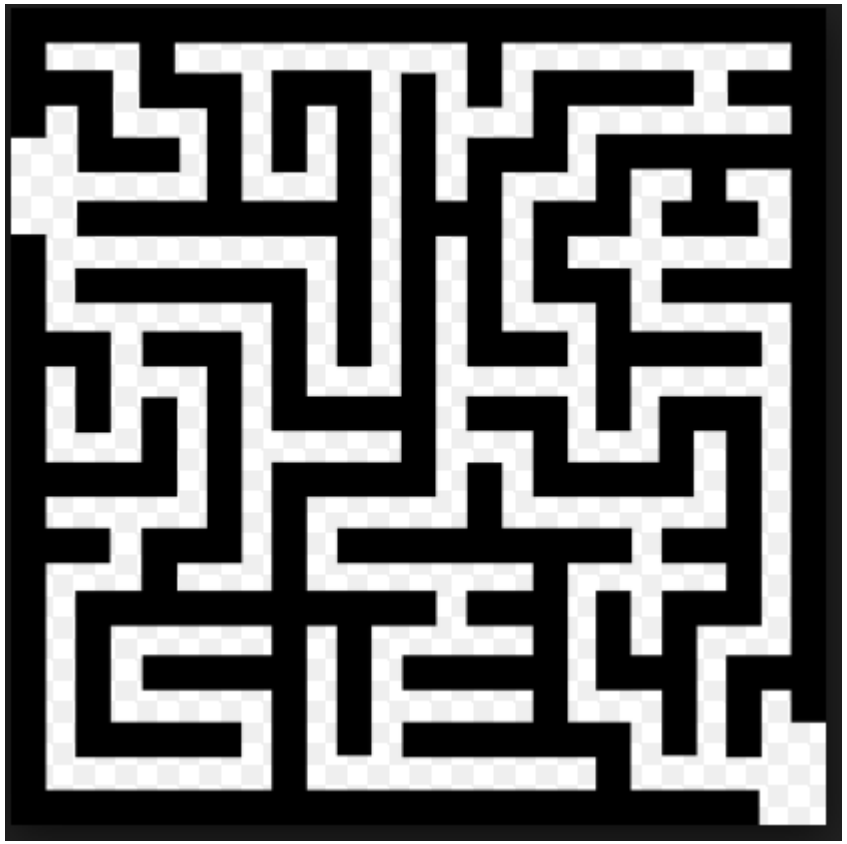
Searches and Mazes..Amazing!!



We will use recursion to solve a maze

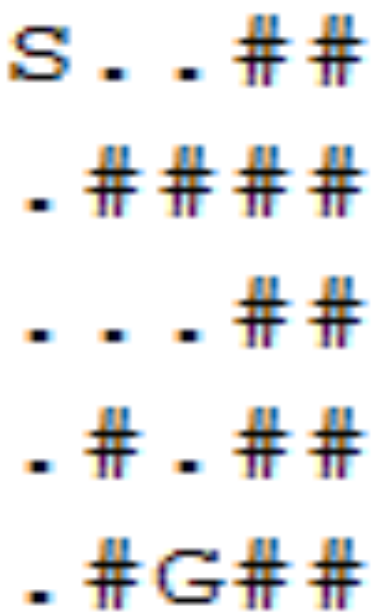
We will simply find a path from the starting node to the goal node

On the next several slides, we solve a maze using depth first search

We will specify a maze as follows

- **We will give the size of the maze (rows and columns)**
- **The letter S will designate the starting cell of the maze**
- **The letter G will designate the goal cell of the maze**
- **A '#' means a cell may not be occupied**
- **A '.' means we can occupy the cell on the way from S to G**

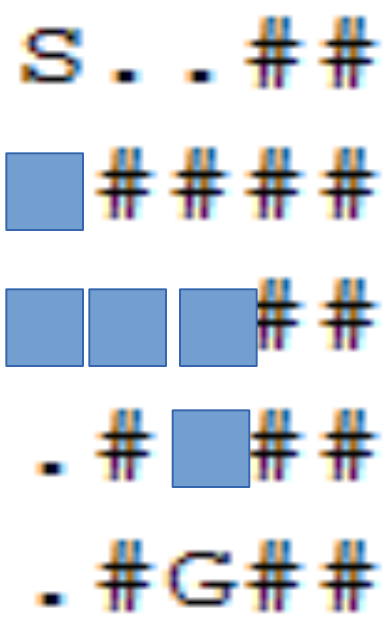
Sample Maze – note that this is NOT an adjacency matrix

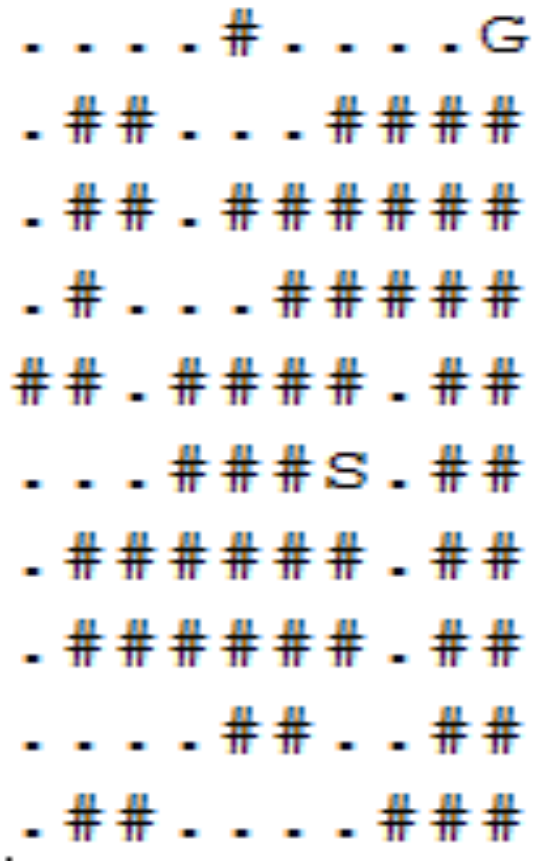


Here is a sample maze

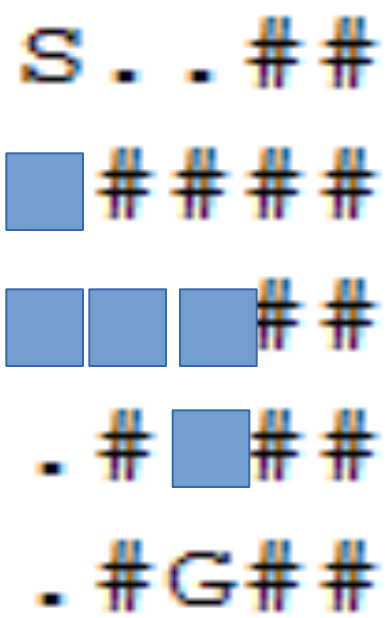
- The size is 5x5
- We start at cell [0][0]
- Our Goal is to reach cell [4][2]
- # indicates the path from S to G cannot pass through the cell
- . indicates we can pass through the cell
- P indicates the cell is on the path from S to G
- ? indicates that the cell was visited but then removed from the solution path via backtracking

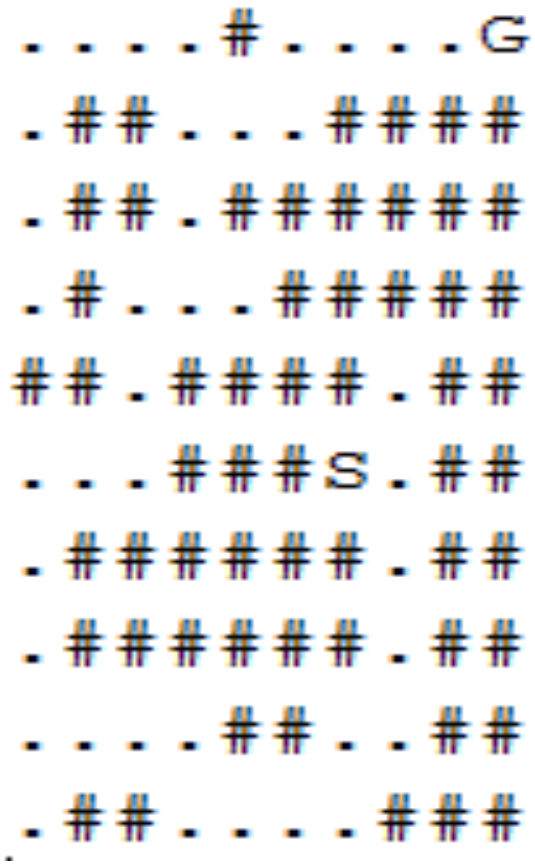
By Inspection we see the solution path in the maze on the left..the one on the right is more complicated





By Inspection we see the solution path in the maze on the left..the one on the right is more complicated





Algorithm to Find Path in A Maze

Begin at start vertex S

if possible, go the vertex, V, *north* of S
if V is the goal, we are done
else
    Recursively call the algorithm from V

if possible, go the vertex, V, east of S
if V is the goal, we are done
else
    Recursively call the algorithm from V
Repeat for south and west

Larger maze with solution by DFS

```
....#....G
.##...####
.##.######
.#...#####
##.####.##
...###S.##
.######.##
.######.##
....##..##
.##....###
```

Why are these 7 ‘?’ here?

```
????#PPPPG
?##PPP####
?##P######
?#PP.#####
##P#####?##
PPP###SP##
P#######P##
P#######P##
PPPP##PP##
.##PPPP###
```

Why is this ‘?’ here?

How we store a maze! Rows and columns followed by maze entries.

```
5 5
S..##
.####
...##
.#.##
.#G##
```

```
FIND-PATH(x, y)

 1. if (x,y outside maze) return false
 2. if (x,y is goal) return true
 3. if (x,y not open) return false
 4. mark x,y as part of solution path
 5. if (FIND-PATH(North of x,y) == true) return true
 6. if (FIND-PATH(East of x,y) == true) return true
 7. if (FIND-PATH(South of x,y) == true) return true
 8. if (FIND-PATH(West of x,y) == true) return true
 9. unmark x,y as part of solution path
10. return false
```

- Suppose we elect to represent mazes in text files as shown at top left. We must be able to read such files into a Java or C program
- The pseudocode at bottom left is easily translatable to Java or C. What is (x,y)?
- Think…..
- What are the coordinates of S? How could your program determine this? What about the goal G?
- What does it mean when the pseudocode says that a given (x,y) is "outside the maze"?
- What point is North of (x,y)? East? South? West?