Applications of Recursion

Backtracking

A common use of recursion is evaluating alternatives, when a problem has more than one possible solution. It provides a mechanism for making successive guesses at a solution. When one solution leads to a dead end, recursion gives the ability to back up and try a different alternative.

An ideal example of backtracking is an algorithm to traverse a maze. A maze consists of walls, and a series of connected spaces which form paths, one of which is a solution that exits the maze. To represent a maze using an array, each element of the array contains a character that represents either a wall or a space. So if we have an 4x4 maze where walls are represented by an 'X' and spaces 'O', it would appear as:

O	O	O	X
O	X	X	X
O	O	O	O
X	X	X	O

Assuming that an individual enters the maze at the first element of the array (element 0,0) the trick is to try cells one at a time, staying within the bounds of the maze, until the exit is reached. For our example we will assume that for an NxN maze, we exit via cell (#rows-1, #cols-1).

Since we have multiple possible paths, we need to mark paths we have already tried (like leaving "bread crumbs" to mark our trail). So when a cell is visited, and we haven't been there before we need to mark it, maybe by placing a character into the cell that isn't a wall or path, a '7' in each cell, thus marking it as part of a possible solution. Additionally, once we reach the end of our maze, we need to return "TRUE" to signify that we have succeeded and mark all cells that are actually part of the solution.. perhaps with a '7'. As we are traversing the maze any cell we determine is not part of the solution is marked as visited but invalid, with perhaps a '3'

All Activations that are part of the solution return true, all others false.

So we begin at cell (0,0), our **traversal** algorithm would be invoked with the cell that is the entrance to our maze. Each activation then visits a "cell" as denoted by a row and column # M & N.

Activation 1
Row = 0
Col = 0
Result =?

Standing in a cell we need to test for 4 possibilities.

1. **Is the cell within the bounds of the array, if not return false, otherwise continue.**
2. **Else does the cell already contain a crumb? If so we've tried this path before so try a different direction.. so return false, this is not part of a possible solution.**
3. **Does the cell contain a wall? if so exit returning false.**
4. **Does the cell represent a "space". Does it contain a space character. If so leave a marker behind, by setting it to '7', and continue.**
5. **If we are in a space we have not previously visited, try to take a step into another cell by going either up, down, left, or right. (invoking our algorithm again)**

We start by stepping into cell (0,0) in activation 1, this cell contains a 0, so we mark it as visited by placing a ‘7’ in it. Next we try to step in another direction…Lets assume we try moving in the order **right,**

7	O	O	X
O	X	X	X
O	O	O	O
X	X	X	O

left, down , and then finally up. (The order your try directions is unimportant as long as all 4 are tried.)

To go “**right**” we move forward one column in the array, so we move from column 0 into column 1, and the row remains the same.. so we try cell (0, 1);

This cell also contains a space, so we mark it as visited and step yet again to the **right**

Activation 2
Row = 0
Col = 1
Result = ?

Another space so mark it.

7	7	7	X
O	X	X	X
O	O	O	O
X	X	X	O

When we try to move “**right**” another space we encounter our first problem

Activation 4
Row = 0
Col = 3
Result = false

Cell (0, 3) contains a wall character, so we can’t move in this direction, so this cell is NOT part of our solution, we need to return “false” and back up to the previous space and try another direction., we exit activation 4, and return to activation 3.

7	7	7	X
O	X	X	X
O	O	O	O
X	X	X	O

Now we need to try another direction, we went "**right**", lets try **down**.. by staying in the same column but increasing the row to cell (1, 2).

Activation 1
Row = 0
Col = 0
Result = ?

Activation 2
Row = 0
Col = 1
Result = ?

Activation 3
Row = 0
Col = 2
Result = ?

Activation 5
Row = 1
Col = 2
Result = false

7	7	7	X
O	X	X	X
O	O	O	O
X	X	X	O

This is also a wall, we can't go into this space, so we exit activation 5, return to the previous activation (3) and try another direction

When we try "**left**" we go back one column in the same row.. so we go to cell (0, 1)

Activation 1
Row = 0
Col = 0
Result = ?

Activation 2
Row = 0
Col = 1
Result = ?

Activation 3
Row = 0
Col = 2
Result = ?

Activation 6
Row = 0
Col = 1
Result = false

We have already visited cell (0, 1) in activation 2, it is marked with a "**marker**" so we simply exit returning false, and activation 6 is destroyed, and we try the final direction in activation 3 which is up.. so we try to go "**up**" one row in the same column this is cell
(-1, 2)

Activation 1
Row = 0
Col = 0
Result = ?

Activation 2
Row = 0
Col = 1
Result = ?

Activation 3
Row = 0
Col = 2
Result = false

Activation 7
Row = -1
Col = 2
Result = false

7	**3**	**7**	**X**
O	X	**X**	X
O	O	O	O
X	X	X	O

This is outside the bounds of the array, so we cant move into this cell, so we simply return **false** to activation 3 and exit.

Now, back in activation 3, we have tried all 4 directions, and none of these are a usable path, so cell (0, 2) in activation 3 is not part of a possible path, we return false, mark it as not part of the solution with a '**3'**, and back up to activation 2, Cell (0,1) and try other directions.

7	**7**	**3**	**X**
O	X	**X**	X
O	O	O	O
X	X	X	O

From activation 2, we try moving **down**.. which is to cell (1, 1)

Activation 1
Row = 0
Col = 0
Result = ?

Activation 2
Row = 0
Col = 1
Result = ?

Activation 8
Row = 1
Col = 1
Result = false

Unfortunately this is a wall, and is not part of our solution

7	**7**	**3**	**X**
O	**X**	**X**	X
O	O	O	O
X	X	X	O

If from activation 2, we try to go "**left**" this is cell (0,0) (activation 9) and we have already visited here, so we return to activation 2, and we try the final direction which is **up**.. we try to move to cell (-1, 1)

Activation 1
Row = 0
Col = 0
Result = ?

Activation 2
Row = 0
Col = 1
Result = ?

Activation 10
Row = -1
Col = 1
Result = false

7	7	3	X
O	X	X	X
O	O	O	O
X	X	X	O

This is outside the bounds of our array, so this should return **false** to activation 2. Activation 2 has exhausted all four directions, this means that it is not part of the solution, and also returns false to activation 1.

And we continue, back in activation 1, now we try to go "**down**" to cell {1, 0} which is a valid path that we have not visited.

We continue cell by cell, in all 4 directions until we find a direction that doesn't return **false**

Finally, after many activations we should reach the end of our maze….

7	3	3	X
7	X	X	X
7	7	7	7
X	X	X	7

To distinguish the correct path from incorrect attempts, we need to mark the correct path. We can set the final cell to 7, and then return **true**,

Each of the previous activations will also return true until all activations terminate. The **final** contents of our "maze" puzzle would look like this :

7	3	3	X
7	X	X	X
7	7	7	7
X	X	X	7

The basic traversal algorithm is:

1. **If the indicated location has not been visited before, AND is not outside the bounds of the array, AND does not contain a WALL character, we mark the current element as visited and part of the potential path.**
2. **IF we are in the LAST element of our array (rows-1, cols-1) We are DONE!!!, and the method should return true.**
3. **Else!! WE try each of the 4 directions.**
    a. **We move down one element**
    b. **If the previous call did not return true try to move one element to the left**
    c. **If the previous call did not return true, try to move one element up.**
    d. **Finally if the previous call did not return true, try moving one element to the right.**
    e. **if All for directions return false, then change the contents of the cell to mark it as not part of the solution, and then return false**
    f.