```
TRY()
   IF (this cell is a wall)
      return(NO)
   ELSE IF (path from this cell in any direction)
      return(YES)
   ELSE
      return(NO);
```

We have now come to something that actually looks difficult — how can we find out whether there is a path from a point to the edge in a particular direction?

The maze can get pretty big, and neither a mouse nor a computer can consider much of it at any one time. Whatever strategy mouse or machine adopts should be a "local" one, in the sense that only a small neighborhood of cells is ever considered at once and that the same strategy is used regardless of where the neighborhood lies in the maze.

A powerful tool for reducing apparent complexity is recursion. In a recursive procedure, the method of solution is defined in terms of itself. That is, each part of the routine handles only a small piece of the strategy, then calls the other parts of the routine as needed to handle the rest. The trick is to reduce each hard case to one that is handled simply elsewhere.

The mouse in a maze problem is a natural for recursion. Imagine that the mouse is sitting somewhere in the middle of the maze wondering if there if a path from where he is. If he is a clever mouse, he will realize that there is a path from where he is to the border if

(1) there is an adjacent accessible cell that he hasn't already looked at, and

(2) there is a path from that cell.

Answering (1) is trivial. And the answer to (2) can be determined in exactly the same way as the original question, except that the mouse is presumably one step nearer to a solution. Thus the solution process is defined in terms of itself: recursion.

Although defining the solution recursively sounds like an infinite loop, it does terminate. Searching stops when the mouse finds a path, or when he has checked out all possibilities without success.

Let us see how recursion applies to path-finding. Suppose we define a recursive function `TRY(i1,j1,i2,j2)` which returns `YES` if there is a path from the point `(i1,j1)` through adjacent point `(i2,j2)` that leads to the edge. If there is no path, `TRY` returns `NO`. How does it work?

If `(i1,j1)` is a wall, there is certainly no path. If we've previously investigated `(i2,j2)` without success or if it's a wall, there is no path. If `(i2,j2)` is an edge cell, there is a path. Otherwise, we simply put `(i1,j1)` on the path tentatively, step over to `(i2,j2)` (making it the current `(i1,j1)`), then look in the four directions from there. If there's a path from one of them, there's a path from `(i1,j1)`, otherwise there isn't.

This is actually clearer in pseudo-code: