A simple and effective way to grow your computer science skills is to master the basics. Knowing the basics sets apart the great coders from the merely intermediate ones. One such basic area in computer science is graph theory which we address in this puzzle.

So first things first: what is a graph? You already know data structures like lists, sets, and dictionaries. These data structures are denoted as complex data structures–not because they’re difficult to understand but because they build upon other data structures.  
A graph is just another complex data structure for relational data.

Relational data consists of edges and vertices. Each vertex stands in one or more relations with other vertices. An example for relational data is the Facebook social graph. Facebook represents users as vertices and friendship relations as edges. Two users are connected via an edge in the graph if they are (Facebook) friends.

How to maintain a graph data structure in the code? The puzzle uses an adjacency matrix as graph data structure G. Each row i in the matrix stores the out-neighbors of vertex i. And each column j stores the in-neighbors of vertex j. Thus, there is an edge from vertex i to vertex j, if G[i][j]==1.

How to determine whether there is a path between two vertices? The function find\_path(graph, v\_start, v\_end, path\_len) checks whether there is a direct or indirect path between two vertices v\_start and v\_end in graph. We know that there is a direct path between v\_start and v\_end if both are already neighbors, i.e., graph[v\_start][v\_end]==1.

However, even if there is not a direct path, there could be an indirect path between vertices v\_start and v\_end. To check this, the algorithm uses a recursive approach. Specifically, there is an indirect path if a vertex v\_nbor exists such that there is a path v\_start –> v\_nbor –> … –> v\_end.

The variable path\_len stores the length of the current path. We increment it in each recursion level as the current path length increases by one. Note that all paths with length $geq n$ consist of at least $n$ vertices. In other words, at least one vertex is visited twice and a cycle exists in this recursion instance. Hence, we skip recursion for paths with length greater or equal than the number of vertices in the graph.

This puzzle asks whether there is a path between 3 and 0. If you understand what the code is doing, it suffices to look at the adjacency matrix G. There is a direct path from vertex 3 to vertices 1 and 2 (and to itself). But neither vertex 1 nor 2 has any out-neighbors. Therefore, there is no path from vertex 3 to any other vertex (besides vertices 1 and 2).