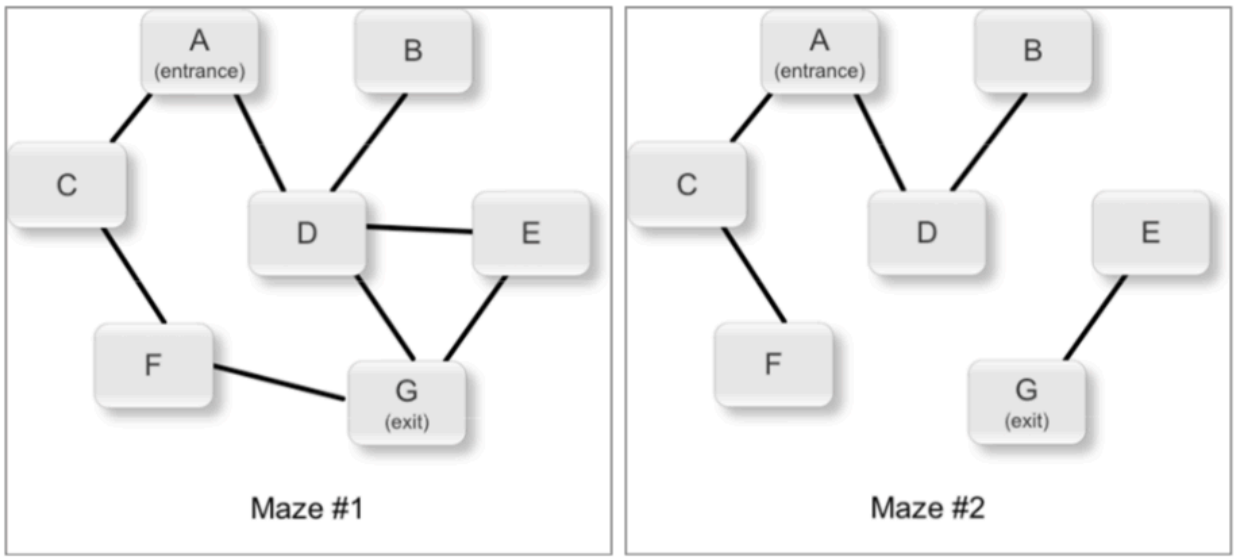
Homework #8

In this assignment you will write an algorithm that determines whether there is a path through a maze. A maze will be represented as an undirected graph with each room represented as a vertex and each corridor represented as a pair of edges. Each room will be uniquely identified by a single character label.

Here are two sample mazes that will be used to test your implementation (notice that maze #1 has a path from the entrance to exit but maze #2 does not):



a) **(9 points)** Implement the following function:

```
int isExitReachable (Graph *pMaze, char entrance, char exit);
```

- This function should return whether a path exists from entrance to exit. A non-zero return value indicates that a path exists; a zero return value indicates that no path exists.

  Demonstrate your function working with mazes #1 & #2. Note, your implement must be generic (i.e. work with any maze) even though you are only required to demonstrate success with mazes #1 & #2.

```c
#include "graph.h"
#include <stdlib.h>
#include <stdio.h>

// Use Depth First Searching to find maze exits
static int DFS(Graph *pMaze, const void *current, const void *exit, Set *visited) {
    // Use a set to keep track of visited verticies

    AdjList *adjlist;                                   // adjecency list of verticies (connected verticies)
    ListElmt *element;

    if (pMaze->match(current, exit)) {                  // if reach exit, return 1 ane exit
        return 1;
    }

    if (graph_adjlist(pMaze, current, &adjlist) != 0) {     // Get adjaency list & check for failure
        return 0;
    }

    if (set_insert(visited, current) != 0) {                // Insert vertex into visited set & check for failure
        return 0;
    }

    // Check each vertex in adjacency list
    for (element = list_head(&adjlist->adjacent); element != NULL; element = list_next(element)) {
        void *neighbor = list_data(element);

        // Skip previously visisted verticies
        if (set_is_member(visited, neighbor)) {
            continue;
        }

        // Use recursion to check for exit in the set
        if (DFS(pMaze, neighbor, exit, visited)) {          // Leave if exit is found
            return 1;                                       // Only true of false result, no need to count number of paths
        }
    }

    return 0;                                           // no path to exit found
}
```

```c
int isExitReachable(Graph *pMaze, char entrance, char exit) {
    Set visited;
    int result;
    char entranceChar = entrance;
    char exitChar = exit;

    // initialize the visited set
    set_init(&visited, pMaze->match, NULL);

    // Perform DFS
    result = DFS(pMaze, &entranceChar, &exitChar, &visited);

    // Clean up
    set_destroy(&visited);

    return result;
}
```

```c
static int match_char(const void *key1, const void *key2) {
    return (*(const char *)key1 == *(const char *)key2);
}
```

```c
int main() {
    Graph maze1;
    Graph maze2;

char vertices[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G'};

    // Initialize both mazes
    graph_init(&maze1, match_char, NULL);
    graph_init(&maze2, match_char, NULL);

    // Graph Maze #1
    for (int i = 0; i < sizeof(vertices)/sizeof(vertices[0]); i++) {
        graph_ins_vertex(&maze1, &vertices[i]);
    }
    graph_ins_edge(&maze1, &vertices[0], &vertices[3]);         // A-D
    graph_ins_edge(&maze1, &vertices[3], &vertices[1]);         // D-B
    graph_ins_edge(&maze1, &vertices[3], &vertices[4]);         // D-E
    graph_ins_edge(&maze1, &vertices[3], &vertices[6]);         // D-G
    graph_ins_edge(&maze1, &vertices[0], &vertices[2]);         // A-C
    graph_ins_edge(&maze1, &vertices[2], &vertices[5]);         // C-F
    graph_ins_edge(&maze1, &vertices[5], &vertices[6]);         // F-G
    graph_ins_edge(&maze1, &vertices[4], &vertices[6]);         // E-G

    // Graph Maze #2
    for (int i = 0; i < sizeof(vertices)/sizeof(vertices[0]); i++) {
        graph_ins_vertex(&maze2, &vertices[i]);
    }
    graph_ins_edge(&maze2, &vertices[0], &vertices[3]);         // A-D
    graph_ins_edge(&maze2, &vertices[3], &vertices[1]);         // D-B
    graph_ins_edge(&maze2, &vertices[0], &vertices[2]);         // A-C
    graph_ins_edge(&maze2, &vertices[2], &vertices[5]);         // C-F
    graph_ins_edge(&maze2, &vertices[4], &vertices[6]);         // E-G

    // Test if exit is reachable
    printf("Maze #1: Exit is %sreachable from entrance\n",
           isExitReachable(&maze1, 'A', 'G') ? "" : "not ");
    printf("Maze #2: Exit is %sreachable from entrance\n",
           isExitReachable(&maze2, 'A', 'G') ? "" : "not ");

    // Clean up
    graph_destroy(&maze1);
    graph_destroy(&maze2);

    return 0;
}
```

```
~/Desktop/DSA/hw8 main* ❯ ./hw8
Maze #1: Exit is reachable from entrance
Maze #2: Exit is not reachable from entrance
~/Desktop/DSA/hw8 main* ❯
```