MAZE program in C

2019320139 Choi Yun Ji

1. Development environment

OS : Window 10

IDE : Visual Studio 2017

2. Explanation of the algorithm and the code

DFS(Depth-First Search) is to fully explore a branch starting from the root node (or any other node) before moving on to the next branch. It is similar to how to find a path in a maze.

Because when you explore a maze, you keep going in one direction as further as possible, and when you can't go any further, you return to the nearest crossroads and explore the other directions.

To make a maze-finding program, we can use a stack to utilize the characteristics of DFS.

And a stack was implemented as a linked stack for easy insertion and deletion.

① First, push a node with entrance index into a stack.

② Pop a node from the stack and examine the possible moves starting from the north clockwise.

③ If next index is judged to be possible, push current index into the stack and move to next index.

④ When visiting an index, record the maze position not to return to a previously tried path.

⑤ If there's no more movement available in a node, delete the node from stack and try again with previous node.

⑥ When next index is equal to exit index, it is a situation in which path is found.

⑦ When the stack is empty, it is a situation in which fail to find a path because it isn't able to move anymore.

A more detailed explanation of the code is given in the next page as annotation.

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MALLOC(p,s) \
    if (!(p = malloc(s))){\
        fprintf(stderr, "Insufficient memory");\
        exit(EXIT_FAILURE); \
    } //MACRO for malloc

typedef struct {
    int row;
    int column;
    int dir;
}position;
//structure for storing a location in a maze

typedef struct stack *stackPointer;
typedef struct stack {
        position p;
        stackPointer link;
    };
//elements of stack are position structure and stackpointer for pointing to next node

bool findpath();
void push(int row, int column, int dir);
position pop();

stackPointer top = NULL;  //first, the top of linked stack is initialized as NULL
int startrow, startcolumn, endrow, endcolumn;
int maze[1001][1001] = { 0, }; //an array for represent a maze
int mark[1001][1001] = { 0, }; //an array for recording the maze positions already checked
```

```c
int main(int argc, char *argv[])
{
/*Program accept command-line arguments for path to input file and then a path to output file
        argv[1] is input file path, and argv[2] is output file path*/

    FILE * fp1 = fopen(argv[1], "r");

    if (!fp1)
    {
        printf("Error! Can not open the file");
        return 0;
    }
    //if fp1 cannot be opened, print error message

    fscanf(fp1, "%d %d %d %d", &startrow, &startcolumn, &endrow, &endcolumn);
    //bring entrance index and exit index from the input file

    MALLOC(top, sizeof(*top));
    char line[1001];
    int row = 1, column = 1;

    for (int i = 0; i < 2; i++)
        fgets(line, 101, fp1);
    //pass two '\n'

    while (fgets(line, 101, fp1) != NULL)
  //read one line from input file and store in char array
    {
        for (column = 0; line[column - 1] != '\n'; column += 2)
        {
            maze[row][column / 2 + 1] = line[column] - '0';
         //Until the string reaches the end, read value(0 or 1) and store in array 'maze'
```

```
        }
        maze[row][0] = 1;
        maze[row][column / 2 + 1] = 1;
        //surround the maze by a border of 1s
        row++;
    }

    for (int i = 0; i <= (column/2 + 1); i++)
    {
        maze[0][i] = 1;
        maze[row][i] = 1;
    }//surround the maze by a border of 1s

    fclose(fp1);

    push(startrow, startcolumn, 1);  //push entrance index in stack
    mark[startrow][startcolumn]  = 1; //check that entrance index is visited

    bool found = findpath();
    //call function for finding path, if path is found return true, else return false

    FILE *fp2 = fopen(argv[2], "w"); // make output file to print path

    if (!fp2)
    {
        printf("Error! Can not open the file");
        return 0;
    }

    if (found)
    {
        stackPointer temp;
        position path[1001];
```

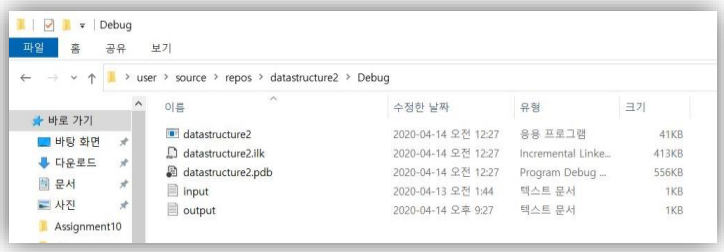
```
            int i = 0;


            for (temp = top; temp != NULL; temp = temp->link, i++)
            {
                path[i] = temp->p; //store position elements of linked stack into array path
            }


            while (i > 0)
            {
                i--;
                fprintf(fp2,"%d %d\n", path[i].row, path[i].column);
     //In output file, positions of linked stack are printed reversely(from entrance to exit)
       using array path
            }
        }
        else
        {
            fprintf(fp2, "%d %d\n", startrow, startcolumn);
            //if there is no path, print entrance index in outputfile
        }


    }
    bool findpath() //function for finding path
    {
        int move[8][2] = { {-1 , 0}, {-1, 1}, {0, 1}, {1, 1}, {1, 0}, {1, -1}, {0, -1}, {-
1, 1} };
        //store possible moves in array move
        bool found = false;


        while (top != NULL && !found) //while stack is not empty -
        {
            position p = pop(); //delete from top of stack
            int row = p.row;
            int column = p.column;
```

```
        int dir = p.dir;

        while (dir < 8 && !found) //while possible moves from current location remain
        {
            int nextrow = row + move[dir][0];
            int nextcolumn = column + move[dir][1];
           //initialize next index to location where haven't moved from current location

            if (nextrow == endrow && nextcolumn == endcolumn) //if reach the exit index
            {
                push(row, column, dir);
                push(nextrow, nextcolumn, 0);
                found = true; //check found
                break;
            }

            else if (!maze[nextrow][nextcolumn] && !mark[nextrow][nextcolumn])
                //if next index is open path and not visited -> next index is movable
            {
                mark[nextrow][nextcolumn] = 1; //check visited
                push(row, column, ++dir); //push current location into stack
                row = nextrow;
                column = nextcolumn;
                dir = 0;
                //move to next index
            }
            else
                dir++; //for trying other moves
        }
    }
    return found;
}
```

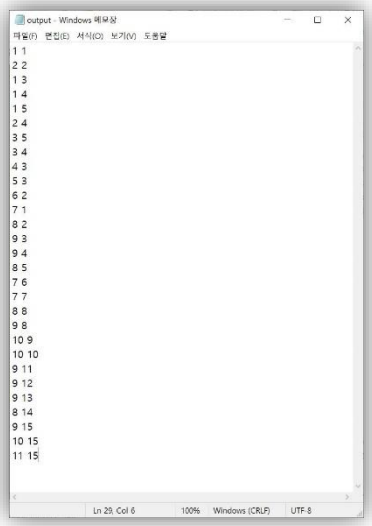
```c
void push(int row, int column, int dir)
{
    position pos = { row, column, dir };
   //make location structure(item of stack) with parameter
    stackPointer temp = NULL; // define stakcpointer variable
    MALLOC(temp, sizeof(*temp)); //malloc memory

    temp->p = pos; //store location structure
    temp->link = top; //connect temp with top (make temp to point to top)
    top = temp; //make temp become top
}

position pop()
{
    stackPointer temp = top;
    position pos;
    if (!temp) //if top is null -> stack is empty
        return;

    pos = temp->p; //store top's location in pos
    top = temp->link; //move top to next node of stack
    free(temp); //delete original top from memory

    return pos; //return location fo original top
}
```

3. Execution Result

I tried to execute the program using cmd to give command-line arguments .



Then, output file has been created like pictures in case of input file being example input.



And in case of input with no path, I could confirm that output included only entrance index.