Hacettepe University
Computer Engineering

BBM 203 – HOMEWORK 1
REPORT

Name: Zekeriya Onur Yakışkan
Student id: 21527539

Problem Definition

A maze has a starting point and an end point. Mazes involves various paths. Generally, most of the paths lead to a dead end. If one can find a path which starts from starting point and ends in end point, that path is one of the solutions of that particular maze.

As a difference from classic maze, our maze will also have keys and Doors. If a door is closed, it behaves like wall. In order to go from a door, at least one place in path must be in the location of its key.

In this homework we are given a square matrix which represents a maze. Our homework is find a valid path solution to given matrix.

Input File

Input file is a txt file and its name will be taken as an argument from command line. Walls of matrix will be represented as 1's and places permitted to go are represented as 0's. Every door is represented by an uppercase letter which can only be opened by the same letters lowercase version(key for the door).

Output File

Output file is a txt file which has to be named as "path.txt". Paths are represented as moves in matrix. A move can be done to east, west, north, south. If one's location is (2,2) in matrix, east is (2,3) , west is (2,1) , north is (1,2) , south is (3,2) . Each move will be represented as their first letters capital(e → E). Path.txt will be "Start path(all moves Exit)".

Algorithm Summary

My algorithm starts from Start point. It marks its location every time before it moves. If it can move it looks which move is a better move to do. If its location is further away from end point horizontally, it make the move which will get it close to end horizontally if it can. If it is further away from end point vertically same goes on. If it can't move in that direction, it moves blindly.

After every move, move is stored in a char array and its location is reassigned with respect to move. If it cant move, it deletes its last move from char array and goes back to its previous location. After that it tries to move again. It is always restricted to go back to a place it had been before.

Every time it is in the same location with a key, it checks whether it has taken this key before. If it is not it resets the maze to initial one. Therefore all the post restriction are gone after it resets maze. Then it starts to move again.

Data Structures

Maze: Maze is a n to n matrix where n = (maze length + 2)**2 . Its length is dynamically found and assigned. Maze is surrounded by 1's, representing mazes borders.

Path: Path is a char array. Size of path is stored in first 4 byte as an integer. Fifth byte is a meaningless char '!'. After that path string starts. Its max size is dynamically assigned as ((maze length) + 1)**

Type term: A term has 3 properties. All of them are ints. One represents column number, one represents row number and one represents value of that row and column.

MazeChars: It is an one dimensional array of terms. First terms col and row number represents actual mazes size. First terms value represents number of terms in that array. Latters are the mazes letters with column and row numbers.

sloc & eloc: one dimensional int arrays where sloc[0] and eloc[0] row of start and end points. Second element is column of start and end points.

Algorithm

Note : Algorithm from where succesfully created my data structers

```
move(maze, mazeChars, path, sloc, eloc )
        here = maze[sloc.row][sloc.col]
        if(here == ‘E’)
                return 1
        if(path.length <  0)
                return 0
        if( ‘a’  <= here <= ‘z’)
                if(!searchPath(path, here))
                        path[ path.len + 5] = here
                        path.len++
                        path[ path.len + 5] = ‘\0’
                        resetMaze(maze , MazeChars)
                else
                        path[ path.len + 5] = here
                        path.len++
                        path[ path.len + 5] = ‘\0’

        if(‘A’ <= here <= ‘Z’)
                path[ path.len + 5] = here
                path.len++
                path[ path.len + 5] = ‘\0’

        dir = getDir(sLoc, eLoc)

        while(count < 4)
                if dir = E
                        if (canE(maze, sLoc, path))
                                loc[0]++
                                path[path.len + 5] = ‘E’
                                path.len++
                                path[path.len + 5] = ‘\0’
                                return move(maze, mazeChars, path, sloc,    eloc )
                        else
                                dir = S

        .       for W , N , S
        .

                count ++
        moveback(path, loc)
        return move(maze, mazeChars, path, sloc,    eloc )
```