**計算機程式Final Project**

學號: B073040031、

姓名: 葉星佑、

**Function prototype and main()**

//linked list storing the maze

typedef struct List list;

void push\_back(list \*first, int \*i, int \*nsew);

Adding data into list

list \*first: root of maze, int \*i: (x,y), \*nsew: (n.s,e,w)

int know\_row(list \*head);

int know\_column(list \*head);

Return the row and column of maze. list \*head: root of maze

//stack storing the solution path

typedef struct Sol sol;

void push(sol \*head, int path);

void pop(sol \*head);

push(): add next step into solution

pop() : delete the last one step from the solution

//execute code

list \*readfile(char \*s);

int \*start\_and\_end(char \*s);

int can\_go(list \*first, int index, int compass);

int next\_step(int index, int direction, int row, int col);

int repeated(sol \*head, int index);

int last(sol \*head);

void find\_path(list \*first, sol \*solution, int index, int end,  int direction, int row, int col);

void output(sol \*head, char \*s);

void find\_path()

list \*first: root of maze (const)

sol \*solution: it add step when forward, it pop when backward

int index: now step(first input is start point)

int end: end point (const)

int direction: 0 1 2 3, n s e w

int row, int col: maze scale (const)

\*\*const means no change in function

//testing function

int size(list \*first);

void size1(sol \*first);

Return size of list and sol

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

//main

int main()

{

    char maze\_file\_name[100];

char in\_and\_out\_file\_name[100];

char output\_file\_name[100];

    printf("Please input the maze format file name: ");

    scanf("%s", maze\_file\_name);

    printf("Please input the in and out file name: ");

    scanf("%s", in\_and\_out\_file\_name);

    printf("Please input the solution\_path file name: ");

    scanf("%s", output\_file\_name);

list \*data\_head = readfile(maze\_file\_name); //maze

    sol \*solution = (sol \*)malloc(sizeof(sol)); //output solution

    solution->i = 0; //initialization default all to zero

    solution->next = NULL;

    int row = know\_row(data\_head);

    int col = know\_column(data\_head);

    int \*sae = start\_and\_end(in\_and\_out\_file\_name); //start\_and\_end

//DFS, recusive

    find\_path(data\_head, solution, sae[0], sae[1], 0, row, col);

    output(solution, output\_file\_name); //output the path to the file

}

3D-maze solving the path with given start and end

input:

      1. maze file name

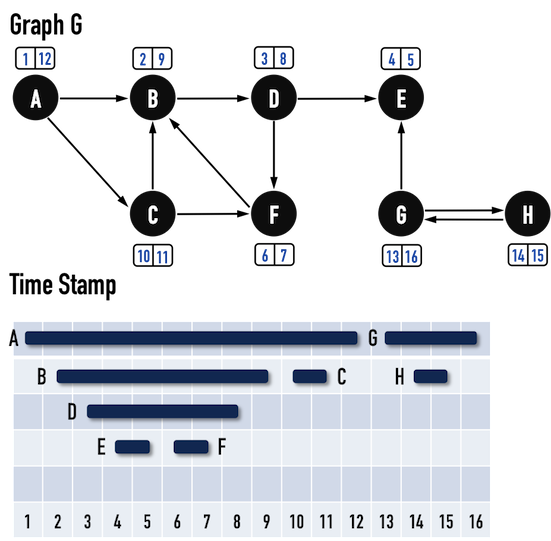
      2. maze start and end name

      3.output file name

output:

store in output\_file\_name.txt

**Algorithm**



**深度優先搜尋演算法**（英語：Depth-First-Search，DFS）是一種用於遍歷或搜尋[樹](https://zh.wikipedia.org/wiki/%E6%A0%91_(%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84))或[圖](https://zh.wikipedia.org/wiki/%E5%9B%BE_(%E6%95%B0%E5%AD%A6))的[演算法](https://zh.wikipedia.org/wiki/%E7%AE%97%E6%B3%95)。沿著樹的深度遍歷樹的節點，盡可能深的搜尋樹的分支。當節點v的所在邊都己被探尋過，搜尋將回溯到發現節點v的那條邊的起始節點。這一過程一直進行到已發現從源節點可達的所有節點為止。如果還存在未被發現的節點，則選擇其中一個作為源節點並重複以上過程，整個行程反覆進行直到所有節點都被存取為止。屬於盲目搜尋。

因為給的迷宮最少有一個解，所以使用此演算法一定找得到。