MAZE ALGORITHM

목차

1. 개요

2019년 3월 12일 알고리즘 응용 수업에서 진행한 미로 찾기 알고리즘을 오른손 법칙을 통해 구현해보고, 해당 루트를 줄일 수 있는 방법을 직접 구현해본다.

- 개발 환경
    - VS code : 문서 작성
    - Visual Studio

2. 문제 정의

1. 출발점에서 탈출구까지 어떤 방식으로 미로를 벗어날 것인가?
2. 해당 방식을 통해 알아낸 루트를 어떻게 줄일 수 있는가?

3. 추상화

1. 탈출 방식
    - 오른손 법칙 활용 : 출구가 있는 모든 미로는 입구에서 벽에 오른손을 짚고, 그 손의 경로를 따라 이동하면 출구에 도달할 수 있다.
        - 벽에 막혔을 경우 : 방향을 전환해서 재확인
            - 회전 방식 : 오른쪽 한번 회전 후 왼쪽 n번 회전으로 일반화 가능하다. 이때, n이란 숫자는 애매해므로 while문을 활용하여 **앞에 벽이 존재하지 않을때까지** 왼쪽으로 회전하도록 한다.
        - 전진 : 방향에 따라 x 및 y 좌표를 더하거나 뺀다. 이때 case문(switch, case), 삼중연산자(? :) 등을 활용하도록 한다.
        - 미로와 시작점, 도착점 : input을 통해 제공한다.
2. 루트 길이 단축 방식
    - 루트 저장 방식
        1. Stack : 두개의 스택에 각각 x좌표와 y좌표를 저장한다. 혹은 하나의 스택에 일정한 알고리즘을 통해 x, y 좌표를 전부 저장한다.
        2. List : x, y, next node를 멤버로 가지는 구조체를 선언, 이를 이어 list를 만든다.

- 필요 없는 부분 선정 : 미로에서 같은 점를 지날경우, 해당 두 node 사이의 이동은 의미 없는 이동이 된다. 따라서 각 node를 비교하며 의미 없는 이동을 선정해낸다.
- 삭제

1. Stack 형태로 루트를 저장한 경우, 같은 점을 지나기는 사이값들을 전부 삭제해 이를 구현할 수 있다.
2. list 형태로 루트를 저장한 경우, 앞 node의 next node를 뒷 node의 next node로 바꿔주면 사이의 데이터를 삭제할 수 있다.

```
        Input : Maze, Maze Size, Start Point(sx, sy), end point
        Output : Maze Route, Shortest Maze Route
```

4. 구현

- 탈출 방식

```
loop -  if in maze :
                if front is wall :
                        turn right
                        while !(front is wall) :
                                turn left
                go front
        goto loop
```

- 루트 길이 단축 방식

```
w/ Stack A                          A                 A
suppose p->(x0,y0)              p  [ x0 ]           p  [ x0 ]
        q->(x1,y1)              p+1[ y0 ]           p+1[ y0 ]
if x0=x1 && y0=y1 :           useless data          q  [ x1 ]
        delete  A p to q        q  [ x1 ]           q+1[ y1 ]
                                q+1[ y1 ]

w/ list A                                             after
suppose p->(x0,y0)              p       |- [   x0,y0   ]-- |    next node : ->
        q->(x1,y1)              p.next  |->[   x1,y1   ]   |
if x0=x1 && y0=y1 :                         useless data   |
        delete  A p to q        q       |- [   xn,yn   ]   |
                                q.next  |->[ xn+1,yn+1 ]<--|
                                        before
```

4. 전체 코드 및 해석

```c
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <windows.h>
#include <time.h>

#define MAZE_SIZE 19
#define MOUSE 2
//각 방향을 shift 연산을 통해 구현
#define UP 1
#define RIGHT 2
#define DOWN 4
#define LEFT 8

typedef struct rec {
        int x;
        int y;
        struct rec* nextnode;
}rec;

int maze[19][19] = { {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
                     {0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1},
                     {1,0,1,0,1,1,1,0,1,1,1,1,1,0,1,0,1,0,1},
                     {1,0,1,0,1,0,0,0,1,0,0,0,0,0,1,0,1,0,1},
                     {1,0,1,0,1,1,1,0,1,1,1,1,1,1,1,0,1,0,1},
                     {1,0,1,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1},
                     {1,0,1,1,1,0,1,0,1,1,1,0,1,0,1,0,1,0,1},
                     {1,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1},
                     {1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1},
                     {1,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1},
                     {1,0,1,1,1,1,1,0,1,0,1,1,1,0,1,0,1,0,1},
                     {1,0,0,0,0,0,0,0,1,0,0,0,1,0,1,0,1,0,1},
                     {1,0,1,1,1,1,1,0,1,0,1,1,1,0,1,0,1,0,1},
                     {1,0,1,0,0,0,0,0,1,0,0,0,1,0,1,0,1,0,1},
                     {1,0,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1},
                     {1,0,1,0,1,0,0,0,0,0,0,0,0,1,0,0,0,1},
                     {1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,1,0,1},
                     {1,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0},
                     {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1} };
void gotoxy(int x, int y) {     //좌표 이동 함수
        COORD Pos = { x,y };
        SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), Pos);
}

rec* init_node() {      //구조체 초기화 함수
        rec* node=(rec*)calloc(1,sizeof(rec));
        return node;
}

int still_in_maze(int x, int y) {       //미로 내에 있는지 확인하는 함수
        if (x > 0 && x <= MAZE_SIZE - 1 && y > 0 && y <= MAZE_SIZE - 1)
                return 1;
        else
```

```
                return 0;
}

int wall_ahead(int m[][MAZE_SIZE], int x, int y, int dir) {// 벽이 앞에 있는지 확인
하는 함수
        x = (dir == LEFT) ? --(x) : (dir == RIGHT) ? ++(x) : x;
        y = (dir == UP) ? --(y) : (dir == DOWN) ? ++(y) : y;
        return m[y][x];
}

void record(rec* node, int x, int y) { //구조체 리스트를 통해 이동 기록을 저장하는 함
수
        rec *tmp = init_node();
        tmp->x = x;
        tmp->y = y;
        node->nextnode = tmp;
}

void forward(int *x, int *y, int dir,rec*node) {        // 앞으로 전진하는 함수
        gotoxy(*x + 1, *y + 1);
        _putch(' ');
        *x = (dir == LEFT) ? --(*x) : (dir == RIGHT) ? ++(*x) : *x;
        *y = (dir == UP) ? --(*y) : (dir == DOWN) ? ++(*y) : *y;
        record(node,*x, *y);
        gotoxy(*x + 1, *y + 1);
        _putch(MOUSE);
}

void turn_right(int *dir) {     // 오른쪽으로 회전하는 함수
        *dir <<= 1;
        *dir = (*dir > LEFT) ? UP : *dir;
}

void turn_left(int *dir) {      // 왼쪽으로 회전하는 함수
        *dir >>= 1;
        *dir = (*dir < UP) ? LEFT : *dir;
}

void right_hand_on_wall(int m[][MAZE_SIZE], int x, int y, int dir, rec* root) { //
오른손 법칙 알고리즘
        rec *pre = init_node();
        pre->x = x;
        pre->y = y;
        root->nextnode = pre;
        gotoxy(x + 1, y + 1);
        _putch(MOUSE);

        forward(&x, &y, dir, pre);
        pre = pre->nextnode;
        while (still_in_maze(x, y)) {
                for (int i = 0; i < 50000000; i++) {}
                turn_right(&dir);
                while (wall_ahead(m, x, y, dir))
                        turn_left(&dir);
```

```
                forward(&x, &y, dir,pre);
                pre = pre->nextnode;
        }
}

rec* delete_way(rec*tmp, rec*node) {    //경로 지우는 함수
        tmp->nextnode = node->nextnode;
        return node->nextnode;
}

void right_hand_delete(rec*node) {      // 짧은 경로 찾는 함수

        rec*tmp = node->nextnode;

        while (tmp->nextnode != NULL) {

                rec* pre = tmp->nextnode;

                while (pre->nextnode != NULL)
                        pre = ((pre->x == tmp->x) && (pre->y == tmp->y)) ?
delete_way(tmp, pre) : (pre->nextnode);

                tmp = tmp->nextnode;

        }
}

void print_maze() {     //출력하는 함수
        int x, y;
        for (y = 0; y < MAZE_SIZE; y++) {
                if (y == 0) gotoxy(0, 1);
                for (x = 0; x < MAZE_SIZE; x++) {
                        if (x == 0) printf(" ");
                        if (maze[y][x] == 1) printf("%%");
                        else printf(" ");
                }
                printf("\n");
        }
}
void main() {
        int dir = LEFT;
        int sx = MAZE_SIZE - 2, sy = MAZE_SIZE - 2;
        rec *root=init_node();
        rec *node = root;

        print_maze();
        right_hand_on_wall(maze, sx, sy, dir, root);
        //right hand on wall

        right_hand_delete(root);
        //get shortest route

        node = node->nextnode;
        while (node->nextnode != NULL) {
```

```
                for (int i = 0; i < 50000000; i++) {}
                gotoxy(node->x + 1, node->y + 1);
                _putch(MOUSE);
                node = node->nextnode;
        }
}
```

- 함수 해석

```
rec* init_node() {
        rec* node=(rec*)calloc(1,sizeof(rec));
        return node;
}
```

  - node 초기화

```
typedef struct rec {
        int x;
        int y;
        struct rec* nextnode;
}rec;
```

  - 함수의 경로를 저장할 때 사용하는 구조체. 위치의 좌표를 저장할 변수와 다음 node의 주소를 저장하는 변수가 필요하다.

```
void gotoxy(int x, int y) {
        COORD Pos = { x,y };
        SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), Pos);
}
```

  - 콘솔창 상에서 좌표를 변경해주는 함수

```
int still_in_maze(int x, int y) {
        if (x > 0 && x <= MAZE_SIZE - 1 && y > 0 && y <= MAZE_SIZE - 1)
                return 1;
        else
                return 0;
}
```

  - 현재 위치가 아직 미로 안인지 확인하는 함수

```
int wall_ahead(int m[][MAZE_SIZE], int x, int y, int dir) {
        x = (dir == LEFT) ? --(x) : (dir == RIGHT) ? ++(x) : x;
```

```
        y = (dir == UP) ? --(y) : (dir == DOWN) ? ++(y) : y;
        return m[y][x];
}
```

- 현재 위치에서 지정된 방향이 벽인지 확인하는 함수

```
void record(rec* node, int x, int y) {
        rec *tmp = init_node();
        tmp->x = x;
        tmp->y = y;
        node->nextnode = tmp;
}
```

- 경로 저장

```
void forward(int *x, int *y, int dir,rec*node) {
        gotoxy(*x + 1, *y + 1);
        _putch(' ');
        *x = (dir == LEFT) ? --(*x) : (dir == RIGHT) ? ++(*x) : *x;
        *y = (dir == UP) ? --(*y) : (dir == DOWN) ? ++(*y) : *y;
        record(node,*x, *y);
        gotoxy(*x + 1, *y + 1);
        _putch(MOUSE);
}
```

- 앞으로 전진하는 함수

```
void turn_right(int *dir) {
        *dir <<= 1;
        *dir = (*dir > LEFT) ? UP : *dir;
}

void turn_left(int *dir) {
        *dir >>= 1;
        *dir = (*dir < UP) ? LEFT : *dir;
}
```

- 오른쪽 및 왼쪽으로 회전하는 함수.

```
void right_hand_on_wall(int m[][MAZE_SIZE], int x, int y, int dir, rec*
root)
{
        rec *pre = init_node();
        pre->x = x;
        pre->y = y;
```

```
        root->nextnode = pre;
        gotoxy(x + 1, y + 1);
        _putch(MOUSE);

        forward(&x, &y, dir, pre);
        pre = pre->nextnode;
        while (still_in_maze(x, y)) {
                for (int i = 0; i < 50000000; i++) {}
                turn_right(&dir);
                while (wall_ahead(m, x, y, dir))
                        turn_left(&dir);
                forward(&x, &y, dir,pre);
                pre = pre->nextnode;
        }
}
```

- 오른손 법칙 알고리즘.

1. 미로 안에 있는지 확인
2. 오른쪽으로 회전 후 길이 보일때까지 왼쪽으로 회전
3. 앞으로 전진하고 좌표를 리스트에 저장.
4. 1~3을 미로에서 빠져나올때까지 반복.

```
rec* delete_way(rec*tmp, rec*node) {
        tmp->nextnode = node->nextnode;
        return node->nextnode;
}
```

- 겹치는 경로가 있을때 호출. 겹치는 노드 사이의 경로를 삭제해주는 함수

```
void right_hand_delete(rec*node) {

        rec*tmp = node->nextnode;

        while (tmp->nextnode != NULL) {

                rec* pre = tmp->nextnode;

                while (pre->nextnode != NULL)
                        pre = ((pre->x == tmp->x) && (pre->y == tmp->y)) ?
delete_way(tmp, pre) : (pre->nextnode);
                tmp = tmp->nextnode;
        }
}
```

- rec리스트를 전부 비교하고 중복되는 좌표의 node가 있으면 delete_way 함수를 호출

```
void print_maze() {
        int x, y;
        for (y = 0; y < MAZE_SIZE; y++) {
                if (y == 0) gotoxy(0, 1);
                for (x = 0; x < MAZE_SIZE; x++) {
                        if (x == 0) printf(" ");
                        if (maze[y][x] == 1) printf("%%");
                        else printf(" ");
                }
                printf("\n");
        }
}
```

- 미로 출력 함수

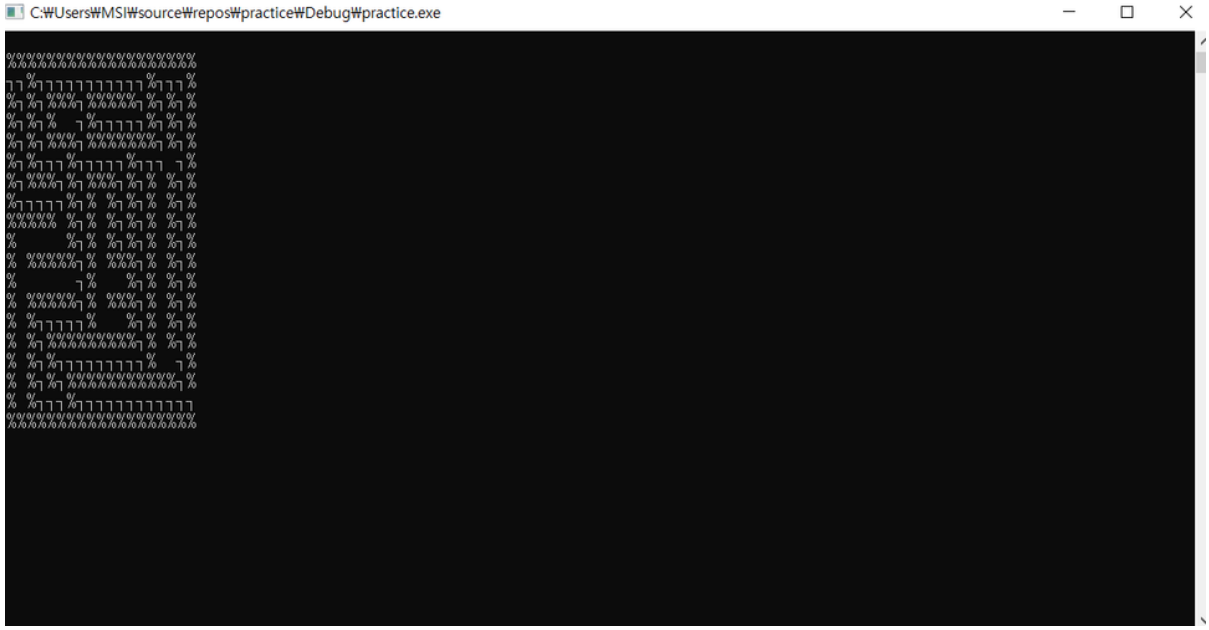
```
void main() {
        int dir = LEFT;
        int sx = MAZE_SIZE - 2, sy = MAZE_SIZE - 2;
        rec *root=init_node();
        rec *node = root;
        print_maze();

        right_hand_on_wall(maze, sx, sy, dir, root);

        right_hand_delete(root);
        node = node->nextnode;
        while (node->nextnode != NULL) {
                for (int i = 0; i < 50000000; i++) {}
                gotoxy(node->x + 1, node->y + 1);
                _putch(MOUSE);
                node = node->nextnode;
        }
}
```

- dir : 방향
- sx : start x
- sy : start y
- root : 리스트 시작 지점
- node : 리스트 출력할때 활용할 변수
    1. 미로 출력
    2. 오른손 알고리즘 실행
    3. root 리스트에 저장된 경로에서 노드를 비교하며 경로 단축
    4. 경로 출력

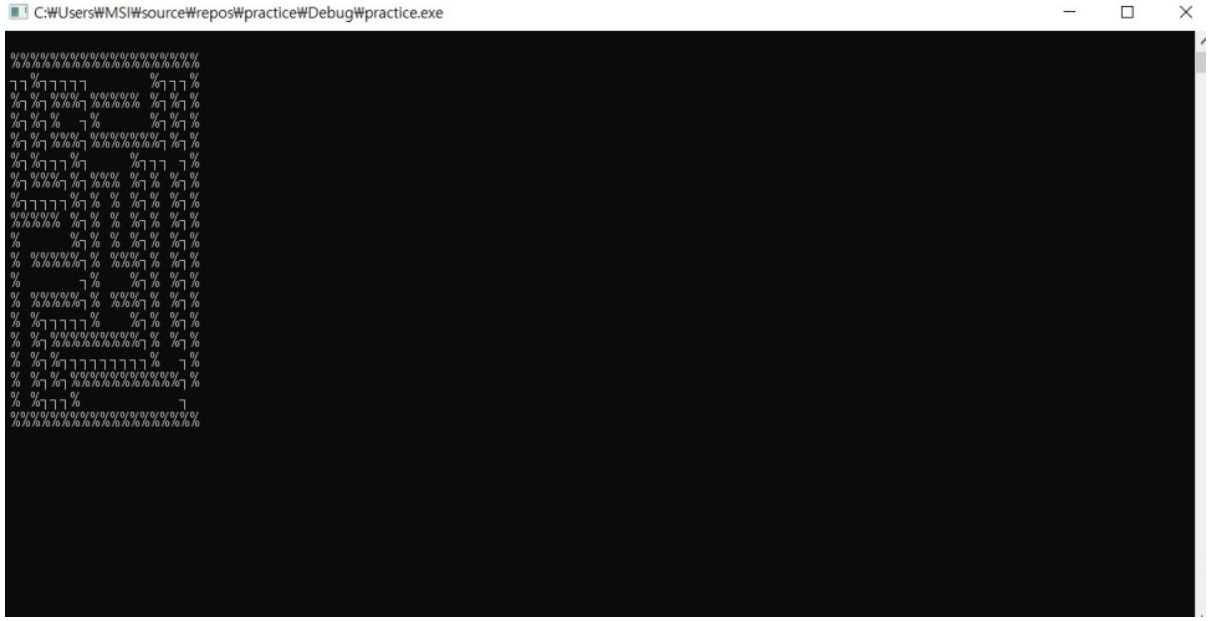
6. 실행 결과 및 공간복잡도

- 실행 결과

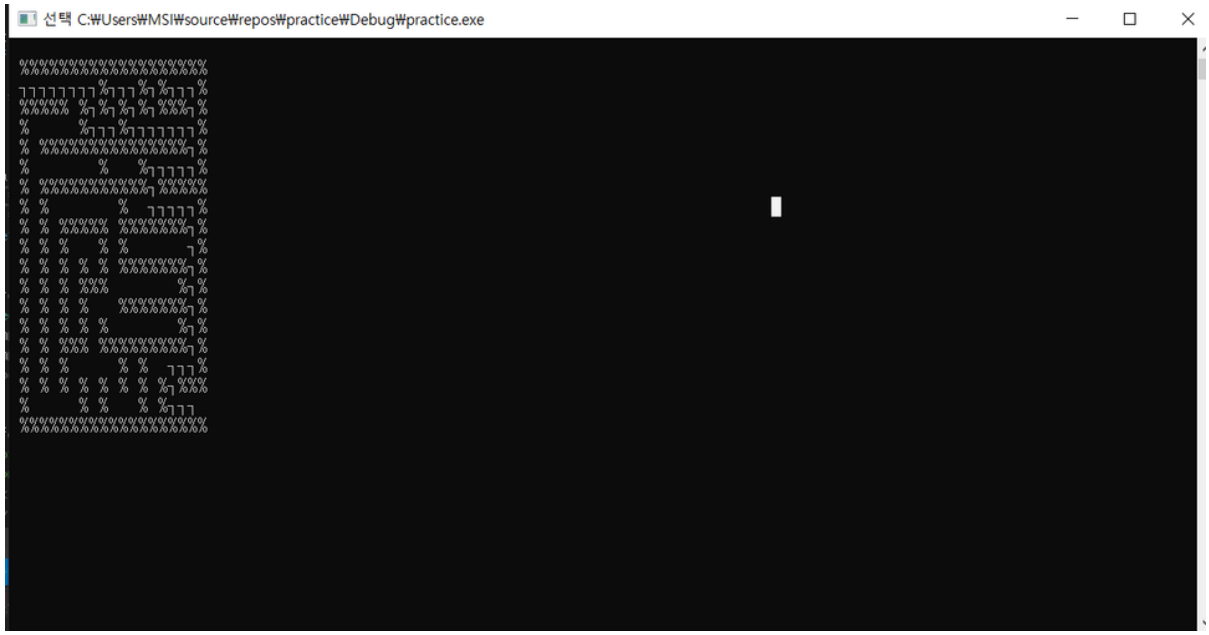
1. Right-hand



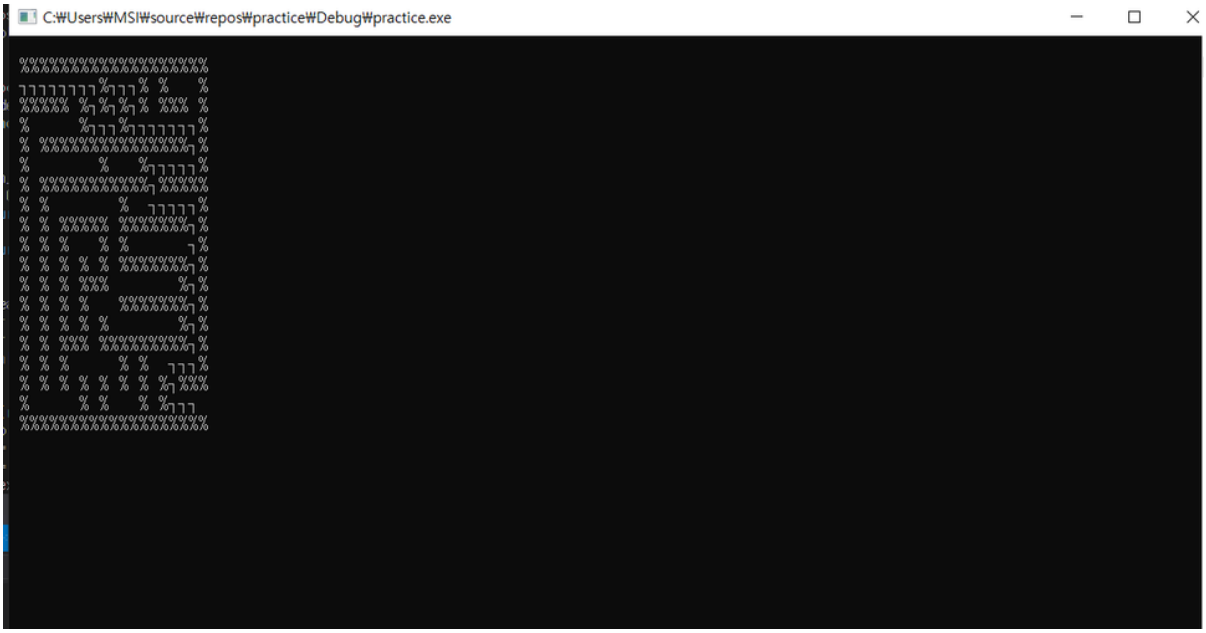
2. Shortest route



3. Right-hand w/ another maze



4. Shortest route w/ another maze



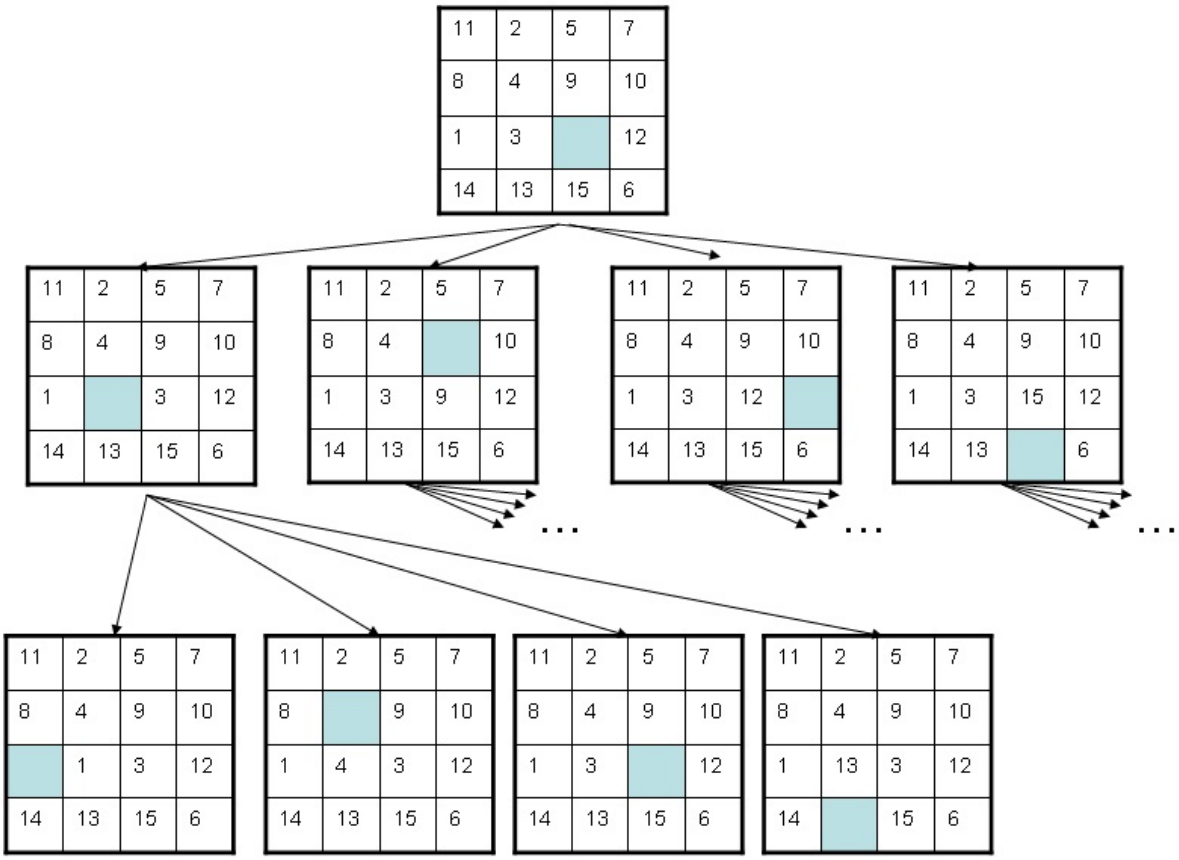
- 공간 복잡도 F(n*(n-1)/2) = O(n^2)

7. 미로 탐색을 구현할 수 있는 알고리즘

1. Random Mouse Solution : 임의로 길을 선택하여 전진. 상당히 느리다.

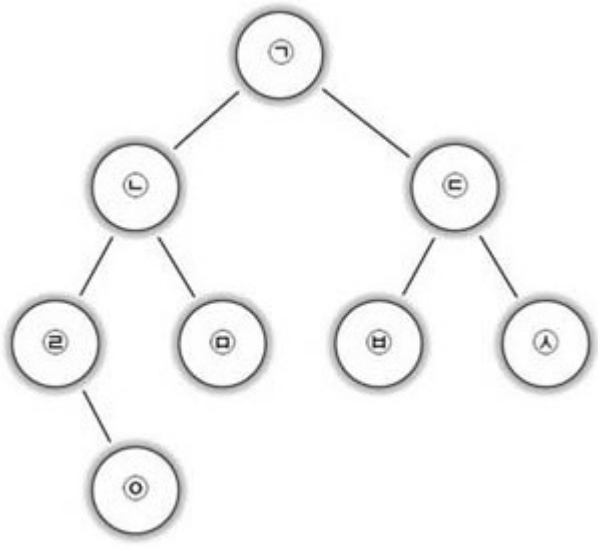
2. Right Hand Soltion(Wall Follower) : 오른쪽 벽을 따라 전진. 대체로 느리다. 벽이 완전히 동떨어져있으면 같은 위치를 빙글빙글 돌 수도 있고, 출구가 두개 있는 경우, 대체로 또 다른 하나의 출구를 찾기 힘들다.

3. A* algorithm : 각 꼭짓점들(이동 및 행동)에서 가중치를 가정하고, 이를 현재의 가중치에 더하는 방식으로 진행



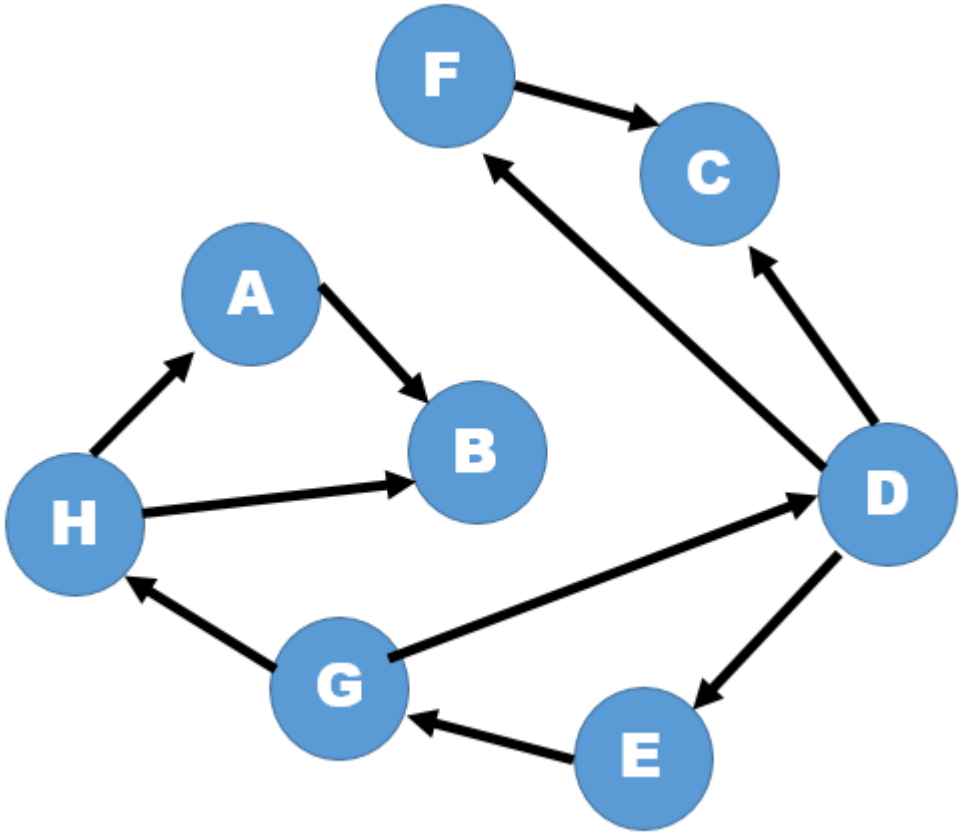
- 빨간색 - 가중치 총 합, 파란색 : 이전 가중치, 초록색 : 예상되는 가중치

4. Breadth-first search : 시작점에서 더 가까운 부분을 먼저 방문.



- 탐색 순서 : ㄱ -> ㄴ -> ㄷ -> ㄹ -> ㅁ -> ㅂ -> ㅅ -> ㅇ

5. Depth-first search : 한 노드의 가지를 계속 따라가다 길이 막히거나 이미 지나온 길을 만나면 가장 최근에 있었던 갈림길로 되돌아가 재시작한다.



- 탐색 순서 : D -> C -> E -> G -> H -> A -> B -> F

8. Reference

쁠로그, [Graph Searching1] 깊이 우선 탐색(Depth First Search : DFS), 2018, https://blog.naver.com/jungeun3814/221321483598 Wikipedia, A* 알고리즘, https://ko.wikipedia.org/wiki/A*_%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98 Wikipedia, Maze Solving Algorithm, https://en.wikipedia.org/wiki/Maze_solving_algorithm