자료구조

[프로그래밍 과제 #3] 최단 거리 미로 탐색

(제출일: 2017년 12월 06일)

담당교수: 김경아

분반 : 1반

학과/학년 : 컴퓨터공학과 / 2

팀명 : Paul Basset

이름 : 여나경(1615039)
이영은(1615051)

최단 거리 미로 탐색

팀명 : Paul Bassett
1615039 여나경
1615051 이영은

1. 문제 기술

스택은 후입 선출(LIFO: Last-In First-Out)의 형식으로 입출력이 일어나는 자료 구조로, 제일 먼저 입력된 데이터가 맨 아래에 쌓이고 가장 최근에 입력된 데이터가 가장 위에 쌓이는 구조이다. 스택을 이용해 미로에 갇힌 생쥐를 출구로 내보내며, 이때 최단 거리 탈출 경로를 탐색한다. 미로는 여러 개의 작은 칸으로 구성되어 있고 입구는 맨 왼쪽 열, 출구는 맨 오른쪽 열에 있다고 가정한다.

2. 입출력의 예

MAZE_SIZE 가 6, 7 인 미로를 입력한다. 입력된 미로를 출력할 때 입구 s, 출구 e, 벽 1, 빈 공간은 0 으로 나타낸다. 예를 들어, 입구의 위치가 (1,0), 출구의 위치가 (4,5) 일 때 입구의 위치가 출구의 위치보다 위쪽에 위치해 있으므로 시계방향일 때 '서-북-동-남'의 순서로 스택에 저장하고 반시계방향일 때 '북-서-남-동'의 순서로 저장한다. 이렇게 pop 된 좌표들의 경로를 생쥐의 이동경로, pop 된 개수를 생쥐의 이동거리로 표현하고 시계방향일 때와 반시계방향일 때의 이동거리의 개수를 비교하여 최단거리의 경우를 최단경로로 출력한다.

3. 문제풀이 방법

MAZE_SIZE 를 6 혹은 7 로 지정하고 미로를 입력한다. 입력된 미로를 출력하고 입·출구 위치를 파악한다. 입구와 출구의 상대적인 위치에 따라 스택에 저장하는 순서를 다르게 지정한다. 또한 시계 방향과 반시계 방향으로 경우를 나누어 두 경우를 출력한다. 입구가 출구보다 위에 있을 경우, 시계 방향으로 스택에 넣을 때 현재 위치를 기준으로 '서-북-동-남'의 순서로 저장한다. 반시계 방향으로 스택에 넣을 때는 현재 위치를 기준으로 '북-서-남-동'의 순서로 저장한다. 입구가 출구보다 아래에 있을 경우 시계 방향으로 스택에 넣을 때 현재 위치를 기준으로 '남-서-북-동'의 순서로 저장하고, 반시계 방향으로 스택에 넣을 때는 현재 위치를 기준으로 '서-남-동-북'의 순서로 저장한다. 이는 입구가 출구보다 위에 있을 때는 오른쪽과 아래의 블록을 마지막에 스택에 저장하고, 입구가 출구보다

아래에 있을 때는 오른쪽과 위의 블록을 마지막에 스택에 저장해야 최단거리를 발견할 수 있기 때문이다.

생쥐가 움직일 수 있는 두 가지의 경로를 모두 출력하고 각각의 이동 거리를 측정(count 변수)한 후 최단 거리를 결정해 해당 경로를 출력한다.

4. 소스 프로그램

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h> //exit 함수 사용

#define MAX_STACK_SIZE 500
#define MAZE_SIZE 7 //6 또는 7을 사용할 계획

char maze1[MAZE_SIZE][MAZE_SIZE]; //시계 방향으로 저장
char maze2[MAZE_SIZE][MAZE_SIZE]; //반시계 방향으로 저장

int count_block = 0;

typedef struct {
        short r;
        short c;
} element;

typedef struct {
        element stack[MAX_STACK_SIZE];
        int top;
} StackType;


element here, start, end; //현재위치, 입구, 출구

//maze 정의
char maze[MAZE_SIZE][MAZE_SIZE] = {
        { '1','1','1','1','1','1','1'},
        { 's','0','1','0','1','1','1'},
        { '1','0','1','0','1','1','1'},
        { '1','0','0','0','1','1','1'},
        { '1','0','1','0','0','1','1'},
        { '1','0','0','1','0','0','e'},
        { '1','1','1','1','1','1','1'},
};


//스택 초기화 함수
void init(StackType *s)
{
        s->top = -1;
        count_block = 0; //이동 거리 초기화
}
```

```
//공백 상태 검출 함수
int is_empty(StackType *s)
{
	return (s->top == -1);
}

//포화 상태 검출 함수
int is_full(StackType *s)
{
	return (s->top == (MAX_STACK_SIZE - 1));
}

//삽입 함수
void push(StackType *s, element item)
{
	if (is_full(s)) {
		fprintf(stderr, "스택 포화 에러₩n");
		return;
	}
	else s->stack[++(s->top)] = item;
}

//삭제 함수
element pop(StackType *s)
{
	if (is_empty(s)) {
		fprintf(stderr, "스택 공백 에러₩n");
		exit(1);
	}
	else return s->stack[(s->top)--];
}

//위치를 스택에 삽입하는 함수
/*1615051 이영은*/
void push_loc_maze1(StackType *s, int r, int c)
{
	if (r < 0 || c < 0) return;
	if (maze1[r][c] != '1' && maze1[r][c] != '>') {
	//벽('1')이 아니고 방문('>')되지 않았으면
		element tmp;
		tmp.r = r;
		tmp.c = c;
		push(s, tmp);
	}

}

/*1615051 이영은*/
void push_loc_maze2(StackType *s, int r, int c)
{
	if (r < 0 || c < 0) return;
	if (maze2[r][c] != '1' && maze2[r][c] != '>') {
	//벽('1')이 아니고 방문('>')되지 않았으면
		element tmp;
```

```
                tmp.r = r;
                tmp.c = c;
                push(s, tmp);
        }
}



//기존의 2차원 배열을 새로운 2차원 배열에 복사하는 함수
/*1615051 이영은*/
void copy(char dst[MAZE_SIZE][MAZE_SIZE], const char src[MAZE_SIZE][MAZE_SIZE], int n) {
        int i, j;
        for (i = 0; i < n; i++) {
                for (j = 0; j < n; j++) {
                        dst[i][j] = src[i][j];
                }
        }
}

//미로를 출력하는 함수
/*1615051 이영은*/
void print_Maze(char maze[MAZE_SIZE][MAZE_SIZE], int n) {
        int i, j;
        for (i = 0; i<n; i++) {
                for (j = 0; j<n; j++) {
                        printf("%c", maze[i][j]);
                }
                printf("₩n");
        }
}

//'1'과 '0'을 출력할때 보기 좋게 만들어 주는 함수
/*1615051 이영은*/
void change_Maze(char maze[MAZE_SIZE][MAZE_SIZE], int n) {
        int i, j;
        for (i = 0; i<n; i++) {
                for (j = 0; j<n; j++) {
                        if (maze[i][j] == '1')
                                maze[i][j] = 127;

                        else if (maze[i][j] == '0')
                                maze[i][j] = 0;

                }
        }
}

//이동거리 count
/*1615039 여나경*/
void count_Maze(char maze[MAZE_SIZE][MAZE_SIZE]) {
        int i, j;
        for (i = 0; i<MAZE_SIZE; i++) {
                for (j = 0; j<MAZE_SIZE; j++) {
                        if (maze[i][j] == '>')
                                count_block++;
```

```
                }
        }
        printf("생쥐가 이동한 거리 : %d₩n", count_block);
        printf("₩n");
}



void main() {
        int count_block1 = 0; //이동한 블럭 개수
        int count_block2 = 0; //이동한 블럭 개수
        int down_s;
        int r, c; //row, column
        int i, j;
        StackType s;

        init(&s);

        //입구(s)와 출구(e) 탐색
        /*1615051 이영은*/
        for (i = 0; i <= MAZE_SIZE; i++)
        {
                for (j = 0; j <= MAZE_SIZE; j++)
                {
                        if (maze[i][j] == 's') {
                                start.r = i;
                                start.c = j;
                        }
                        if (maze[i][j] == 'e') {
                                end.r = i;
                                end.c = j;
                        }
                }
        }

        printf("입구 : (%d,%d)₩n", start.r, start.c);
        printf("출구 : (%d,%d)₩n", end.r, end.c);

        here = start; //현재 위치를 시작점으로 정의

        copy(maze1, maze, MAZE_SIZE);
        copy(maze2, maze, MAZE_SIZE);

        //입, 출구 위치 비교
        /*1615039 여나경*/
        if (start.r <= end.r) {
                down_s = 0; //입구가 출구보다 위에 있는 경우
        }
        else down_s = 1; //입구가 출구보다 아래에 있는 경우

        //미로 출력(벽:1, 빈 공간:0, 입구:s, 출구:e)
        /*1615039 여나경*/
        for (i = 0; i<MAZE_SIZE; i++) {
                for (j = 0; j<MAZE_SIZE; j++) {
                        printf("%c", maze1[i][j]);
```

```
                }
                printf("₩n");
        }
        printf("₩n");

        //stack 저장 순서 정하기
        /*1615039 여나경*/
        switch (down_s) {
        case 0:  //입구가 위에 있을 경우
                //stack에 시계 방향으로 저장
                printf("₩n<시계 방향으로 스택에 넣을 때>₩n");
                while (maze1[here.r][here.c] != 'e') {
                        r = here.r;
                        c = here.c;
                        maze1[r][c] = '>';

                        push_loc_maze1(&s, r, c - 1); //서
                        push_loc_maze1(&s, r - 1, c); //북
                        push_loc_maze1(&s, r, c + 1); //동
                        push_loc_maze1(&s, r + 1, c); //남

                        if (is_empty(&s)) {
                                printf("스택이 비어있습니다.");
                                return;
                        }
                        else here = pop(&s);
                }

                change_Maze(maze1, MAZE_SIZE);
                print_Maze(maze1, MAZE_SIZE); //이동 경로 출력
                count_Maze(maze1); //이동 거리 count
                count_block1 = count_block;

                //반시계 방향
                //스택, 좌표 초기화
                init(&s);
                here = start;

                printf("₩n<반시계 방향으로 스택에 넣을 때>₩n");
                while (maze2[here.r][here.c] != 'e') {
                        r = here.r;
                        c = here.c;
                        maze2[r][c] = '>';
                        push_loc_maze2(&s, r - 1, c); //북
                        push_loc_maze2(&s, r, c - 1); //서
                        push_loc_maze2(&s, r + 1, c); //남
                        push_loc_maze2(&s, r, c + 1); //동


                        if (is_empty(&s)) {
                                printf("스택이 비어있습니다.");
                                return;
                        }
                        else here = pop(&s);
                }
```

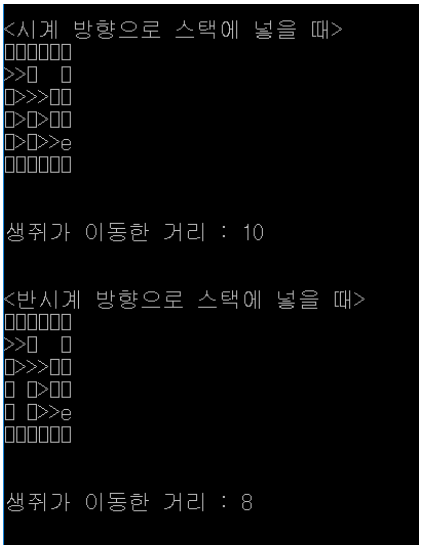
```
                change_Maze(maze2, MAZE_SIZE);
                print_Maze(maze2, MAZE_SIZE); //이동 경로 출력
                count_Maze(maze2); //이동 거리 count
                count_block2 = count_block;

                /*1615051 이영은*/
                printf("₩n<결과>₩n");
                if (count_block1<count_block2) {
                        printf("-> 시계 방향으로 스택에 넣는 경우가 생쥐의
최단경로입니다.₩n");
                        print_Maze(maze1, MAZE_SIZE);
                }
                else if (count_block1>count_block2) {
                        printf("-> 반시계 방향으로 스택에 넣는 경우가 생쥐의
최단경로입니다.₩n");
                        print_Maze(maze2, MAZE_SIZE);
                }
                else if (count_block1 == count_block2) {
                        printf("-> 모든 경우가 생쥐의 최단경로입니다.");
                        print_Maze(maze1, MAZE_SIZE);
                        print_Maze(maze2, MAZE_SIZE);
                }
                break;

                /*1615039 여나경*/
        case 1: //입구가 아래에 있을 경우
                //시계 방향
                printf("<시계 방향으로 스택에 넣을 때>₩n");
                while (maze1[here.r][here.c] != 'e') {
                        r = here.r;
                        c = here.c;
                        maze1[r][c] = '>';

                        push_loc_maze1(&s, r + 1, c); //남
                        push_loc_maze1(&s, r, c - 1); //서
                        push_loc_maze1(&s, r - 1, c); //북
                        push_loc_maze1(&s, r, c + 1); //동

                        if (is_empty(&s)) {
                                printf("스택이 비어있습니다.");
                                return;
                        }
                        else here = pop(&s);

                }

                change_Maze(maze1, MAZE_SIZE);
                print_Maze(maze1, MAZE_SIZE);  //이동 경로 출력
                count_Maze(maze1); //이동 거리 count
                count_block1 = count_block;

                //반시계 방향
                //스택, 좌표 초기화
                init(&s);
```

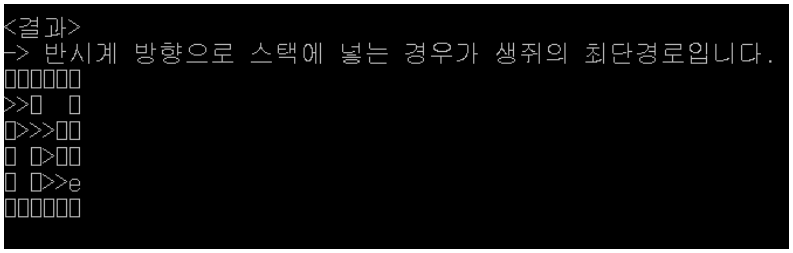
```
                here = start;

                printf("₩n<반시계 방향으로 스택에 넣을 때>₩n");
                while (maze2[here.r][here.c] != 'e') {
                        r = here.r;
                        c = here.c;
                        maze2[r][c] = '>';
                        push_loc_maze2(&s, r, c - 1); //서
                        push_loc_maze2(&s, r + 1, c); //남
                        push_loc_maze2(&s, r, c + 1); //동
                        push_loc_maze2(&s, r - 1, c); //북

                        if (is_empty(&s)) {
                                printf("스택이 비어있습니다.");
                                return;
                        }
                        else here = pop(&s);
                }

                change_Maze(maze2, MAZE_SIZE);
                print_Maze(maze2, MAZE_SIZE);  //이동 경로 출력
                count_Maze(maze2); //이동 거리 count
                count_block2 = count_block;

                /*1615051 이영은*/
                printf("₩n<결과>₩n");
                if (count_block1<count_block2) {
                        printf("-> 시계 방향으로 스택에 넣는 경우가 생쥐의
최단경로입니다.₩n");
                        print_Maze(maze1, MAZE_SIZE);
                }
                else if (count_block1>count_block2) {
                        printf("-> 반시계 방향으로 스택에 넣는 경우가 생쥐의
최단경로입니다.₩n");
                        print_Maze(maze2, MAZE_SIZE);
                }
                else if (count_block1 == count_block2) {
                        printf("-> 모든 경우가 생쥐의 최단경로입니다.₩n");
                        print_Maze(maze1, MAZE_SIZE);
                        print_Maze(maze2, MAZE_SIZE);
                }
                break;
        }
}
```

5. 수행 결과

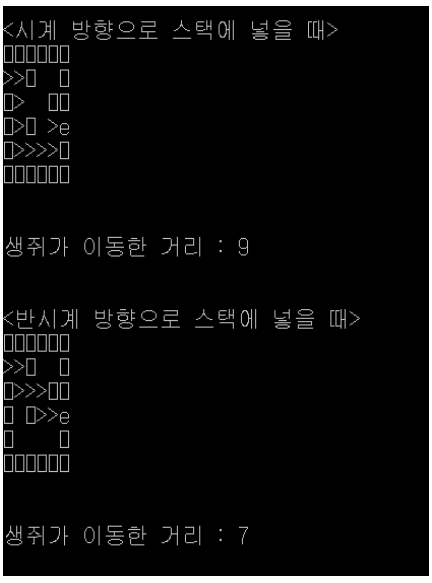
1) MAZE_SIZE 가 6 일 때

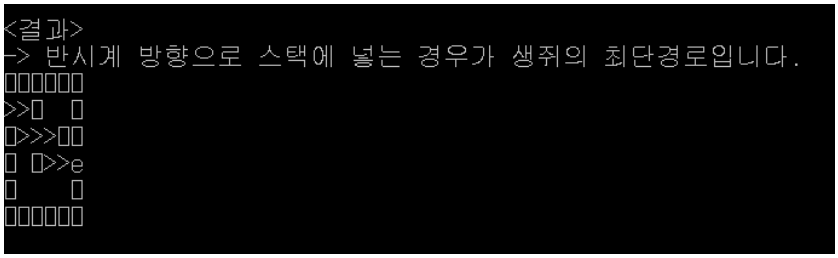
① 반시계방향으로 저장(1)



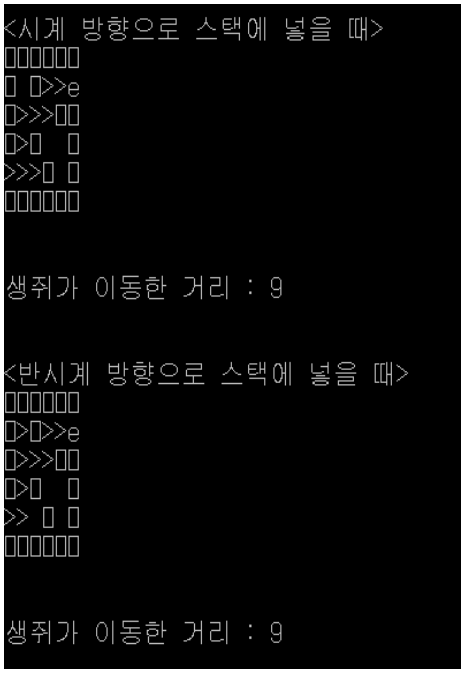


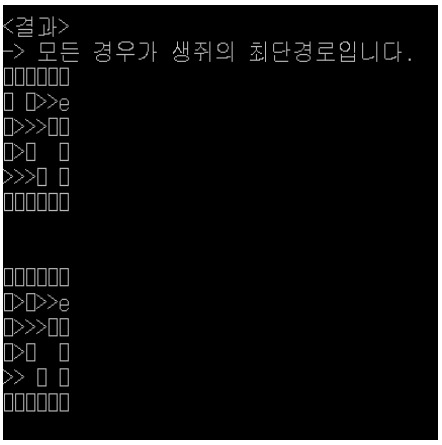
② 반시계방향으로 저장(2)



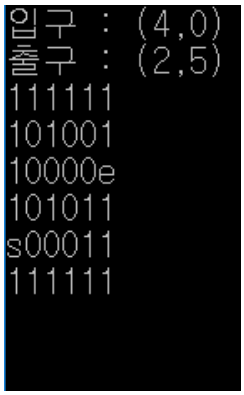


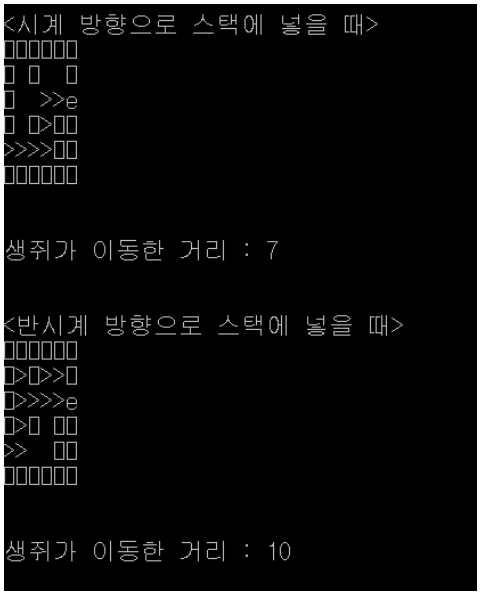
③ 모든 경우

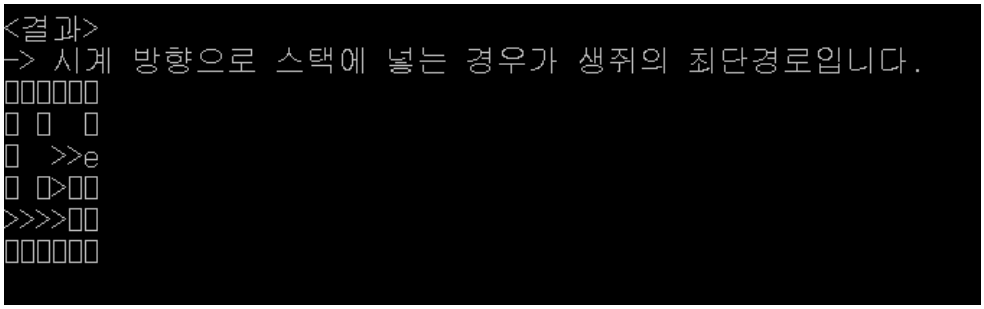




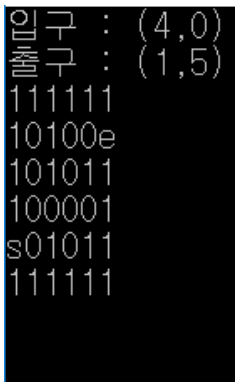
④ 시계방향으로 저장(1)

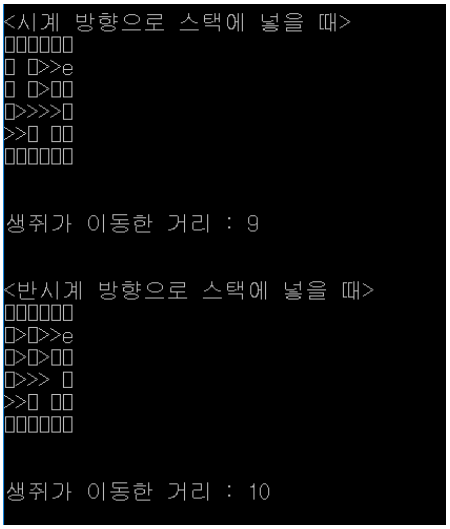


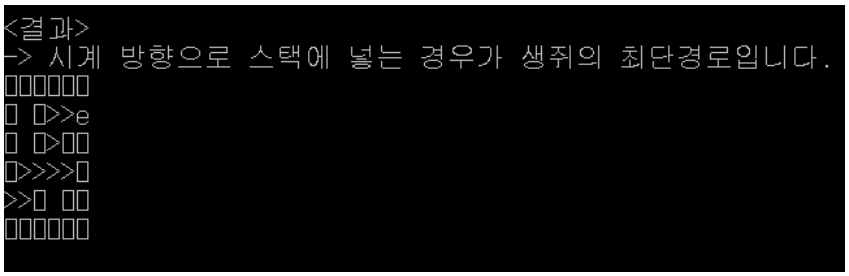




⑤ 시계방향으로 저장(2)

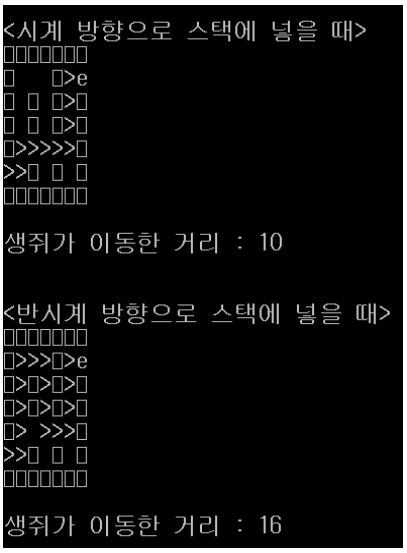


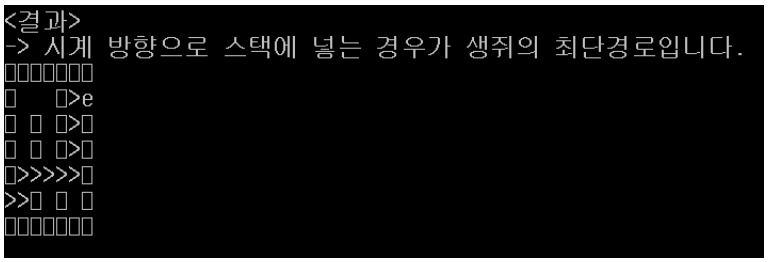




2) Maze size 가 7 일 때

① 시계방향으로 저장(1)





② 시계방향으로 저장(2)

```
입구 : (5,0)
출구 : (1,6)
1111111
100010e
1010001
1010101
1010001
s000101
1111111
```

```
<시계 방향으로 스택에 넣을 때>
□□□□□□□
□   □>e
□ □  >□
□ □ □>□
□ □>>>□
>>>>□ □
□□□□□□□

생쥐가 이동한 거리 : 10


<반시계 방향으로 스택에 넣을 때>
□□□□□□□
□>>>□>e
□>□>>>□
□>□ □ □
□>□   □
>>  □ □
□□□□□□□

생쥐가 이동한 거리 : 12
```

```
<결과>
-> 시계 방향으로 스택에 넣는 경우가 생쥐의 최단경로입니다.
□□□□□□□
□   □>e
□ □  >□
□ □ □>□
□ □>>>□
>>>>□ □
□□□□□□□

계속하려면 아무 키나 누르십시오 . . .
```

③ 반시계방향으로 저장 (1)

```
입구 : (1,0)
출구 : (5,6)
1111111
s000001
1010101
1010101
1010001
101010e
1111111
```

```
<시계 방향으로 스택에 넣을 때>
□□□□□□□
>>>>  □
□>□>□ □
□>□>□ □
□>□>>>□
□>□>□>e
□□□□□□□

생쥐가 이동한 거리 : 15


<반시계 방향으로 스택에 넣을 때>
□□□□□□□
>>>>>>□
□ □ □>□
□ □ □>□
□ □  >□
□ □ □>e
□□□□□□□

생쥐가 이동한 거리 : 10
```

```
<결과>
-> 반시계 방향으로 스택에 넣는 경우가 생쥐의 최단경로입니다.
□□□□□□□
>>>>>>□
□ □ □>□
□ □ □>□
□ □  >□
□ □ □>e
□□□□□□□

계속하려면 아무 키나 누르십시오 . . .
```

④ 반시계방향으로 저장(2)

```
입구 : (1,0)
출구 : (5,6)
1111111
s010111
1010111
1000111
1010011
100100e
1111111
```

```
<시계 방향으로 스택에 넣을 때>
□□□□□□□□
>>□ □□□
□>□ □□□
□>>>□□□
□>□>>□□
□>>□>>e
□□□□□□□□

생쥐가 이동한 거리 : 13


<반시계 방향으로 스택에 넣을 때>
□□□□□□□□
>>□ □□□
□>□ □□□
□>>>□□□
□ □>>□□
□  □>>e
□□□□□□□□

생쥐가 이동한 거리 : 10
```

```
<결과>
-> 반시계 방향으로 스택에 넣는 경우가 생쥐의 최단경로입니다.
□□□□□□□□
>>□ □□□
□>□ □□□
□>>>□□□
□ □>>□□
□  □>>e
□□□□□□□□

계속하려면 아무 키나 누르십시오 . . .
```

⑤ 모든 경우

```
입구 : (5,0)
출구 : (1,6)
1111111
100000e
1010101
1010101
1010001
s000101
1111111
```

```
<시계 방향으로 스택에 넣을 때>
□□□□□□□□
□    >e
□ □ □>□
□ □ □>□
□ □>>>□
>>>>□ □
□□□□□□□□

생쥐가 이동한 거리 : 10


<반시계 방향으로 스택에 넣을 때>
□□□□□□□□
□>>>>>e
□>□ □ □
□>□ □ □
□>□   □
>>  □ □
□□□□□□□□

생쥐가 이동한 거리 : 10
```

```
<결과>
-> 모든 경우가 생쥐의 최단경로입니다.
□□□□□□□□
□    >e
□ □ □>□
□ □ □>□
□ □>>>□
>>>>□ □
□□□□□□□□

□□□□□□□□
□>>>>>e
□>□ □ □
□>□ □ □
□>□   □
>>  □ □
□□□□□□□□

계속하려면 아무 키나 누르십시오 . . .
```

6. 결과 분석 및 토의

각 console창에서 입구의 위치와 출구의 위치, 입력한 배열, 시계방향과 반시계방향으로 스택에 넣었을 때의 이동경로, 이동한 거리와 둘 중 최단거리를 비교하여 출력하는 결과가 잘 출력되었다.

```
{ '1','1','1','1','1','1','1'},
{ 's','0','1','0','1','1','1'},
{ '1','0','1','0','1','1','1'},
{ '1','0','0','0','1','1','1'},
{ '1','0','1','0','0','1','1'},
{ '1','0','0','1','0','0','e'},
{ '1','1','1','1','1','1','1'}
```

입력한 배열을 위의 행렬로 예를 들어 생각해보겠다. print_maze 함수를 이용하여 입력한 배열을 먼저 1과 0으로 출력하였다. 그 다음 change_maze 함수를 이용하여 1을 '□'로 바꾸어 출력하고 생쥐가 이동한 0을 '>'로 바꾸어 출력하고 생쥐가 이동하지않은 0을 공백으로 바꾸어 출력한다. 그 다음 '>'로 바꾼 문자의 개수를 세서 count라는 변수로 정의한 후 시계방향일 때와 반시계방향일 때의 count 개수를 비교하여 count 개수가 더 적은 경로를 최단거리로 결정하고 최단거리의 미로를 한번 더 출력한다(5.2).④ 참고).

이영은 : 현재 위치를 스택에 삽입하는 push_loc_maze1·2함수, copy함수, print_Maze, 1과 0으로 표현되었던 미로를 특수문자로 나타내는 change_Maze함수 등 기본적인 미로 탐색 코드, main 함수에서는 입구와 출구를 탐색해서 해당 좌표를 출력하는 코드, 최단 경로를 결정하고 출력하는 코드.

여나경 : count_Maze함수, 입구와 출구의 상대적인 위치를 판단하는 코드, 입구와 출구의 위치에 따라 스택에 저장하는 순서를 지정하는 코드 등 최단 거리를 탐색하는 코드. count_Maze함수 등을 이용해 이동 경로를 비교하고 최단 거리를 결정하는 코드

수업 시간에 깊이 다루지 않은 미로 탐색을 스택을 이용해 직접 코드로 구현하면서 스택에 대해서 더 깊이 있고 자세하게 배울 수 있었다. 그리고 책에서는 1과 0으로만 표현한 미로를 특수문자를 이용하여 나타냈다. 또한 효율적으로 미로를 탐색하는 방법을 구상하고 이를 스택으로 구현하며 우리가 정한 주제에 흥미를 느꼈다.