자료구조 과제 5

* MAZE –

20151523

김동현

- 실행결과

<maze 1>개체, 시계이(가) 표시된 사진

자동 생성된 설명 <path 1>그리기이(가) 표시된 사진

자동 생성된 설명

<maze 2>개체, 시계이(가) 표시된 사진

자동 생성된 설명 <path 2>그리기이(가) 표시된 사진

자동 생성된 설명

-코드 및 알고리즘 분석

#include <stdio.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 0

: 강의자료를 참고하여 제작하였고, path 함수에서 사용하는 변수들은 전역변수로 선언하여 사용한다고 강의자료에 명시되었기 때문에 필요한 변수들을 전역변수로 선언하고, 동적할당이 필요한 경우에는 main에서 할당해주었다.

typedef struct {

short int row;

short int col;

short int dir;

} element;

element \*stack;

: 스택 자료구조를 구현할 구조체 element를 정의하였다. 포인터로 선언한 이유는 이후에 입력받을 maze의 크기를 토대로 max 사이즈를 동적 할당할 것이기 때문이다.

typedef struct {

short int vert;

short int horiz;

} offsets;

offsets move[8];

: 그리고 방향에 대한 값을 가지는 move 구조체 배열을 정의하였다. 8방향이기 때문에 8개의 원소를 갖는 배열이다.

int EXIT\_ROW, EXIT\_COL;

int top = -1;

int\*\* maze;

int\*\* mark;

: maze와 mark는 2차원 배열로 사용할 변수이므로 더블 포인터로 지정, EXIT\_ROW와 EXIT\_COL은 maze의 종료 지점인데 이는 maze의 실제 크기와 같다고 볼 수 있으므로 추후에 다양한 용도로 쓰이게 된다. Top은 스택에 아무것도 없을 때를 기준으로 -1로 초기화 해주었다.

void move\_set() {

move[0].vert = -1;

move[0].horiz = 0;

move[1].vert = -1;

move[1].horiz = 1;

move[2].vert = 0;

move[2].horiz = 1;

move[3].vert = 1;

move[3].horiz = 1;

move[4].vert = 1;

move[4].horiz = 0;

move[5].vert = 1;

move[5].horiz = -1;

move[6].vert = 0;

move[6].horiz = -1;

move[7].vert = -1;

move[7].horiz = -1;

}

: move\_set 함수는 그냥 단순하게 move 배열의 멤버들 값을 지정해주는 함수이다. 방향의 상수화 라고 보면 될 것이다.

void push(element item)

{

if (top >= (EXIT\_COL\*EXIT\_ROW) / 2 - 1)

printf("stack FULL !\n");

stack[++top] = item;

}

: 스택의 push 함수. 만약 스택 허용 최대 크기를 넘으면 경고문을 출력한다. 하지만 maze 프로그램에서 스택이 초과하는 경우는 있을 수 없기 때문에 코드 에러나 디버깅할 때 확인하는 용도로 설정해두었다.

element pop()

{

if (top == -1)

printf("stack EMPTY !\n");

return stack[top--];

}

: 스택의 pop 함수. 만약 스택이 비어있는데 pop이 호출된 경우, 경고문을 출력한다. 하지만 maze 프로그램에서 스택이 비어있는데 pop을 호출하는 경우는 있을 수 없기 때문에 코드 에러나 디버깅할 때 확인하는 용도로 설정해두었다.

void path(void)

{

int i, row, col, nextRow, nextCol, dir, found = FALSE;

element position;

mark[1][1] = 1; top = 0;

stack[0].row = 1; stack[0].col = 1; stack[0].dir = 1;

: 처음 시작 위치인 (1,1)의 mark를 1로 칠해주고, 방향에 해당하는 스택을 1개 추가하여서 top도 0으로 해주었다.

while (top > -1 && !found) {

: 아직 도착하지 않았고, 스택이 empty가 아닐 경우 아래의 지시를 수행.

position = pop();

row = position.row; col = position.col, dir = position.dir;

아래 반복문에서 스택에 쌓아줬던 위치(지나왔던 위치)를 pop하여 받아서 저장해준다. 막다른 길에 왔을 때 수행하게 될 내용.

while (dir < 8 && !found) {

nextRow = row + move[dir].vert;

nextCol = col + move[dir].horiz;

: 그 다음 가는 좌표를 현재 좌표 + 방향 상수를 통해 설정.

if (nextRow == EXIT\_ROW && nextCol == EXIT\_COL)

found = TRUE;

: 그 다음 가는 좌표가 도착 위치일시 found flag에 TRUE 저장.

else if (!maze[nextRow][nextCol] && !mark[nextRow][nextCol]) {

: 다음 가는 곳이 0(갈수 있는 곳), 가본적 없는 곳인 경우.

mark[nextRow][nextCol] = 1;

position.row = row; position.col = col;

position.dir = ++dir;

push(position);

: 왔다는 표시를 해주고, position 구조체에 현재의 좌표와 다음 방향을 저장한 뒤, 스택에 쌓아둔다. (경로 저장)

row = nextRow; col = nextCol; dir = 0;

}

else ++dir;

: 갈수 없는 곳인 경우, 현재의 위치에서 방향만 바꿔준다.

}

}

if (found) {

printf("The path is : \n");

printf(" row col \n");

for (i = 0; i <= top; i++)

printf(" (%d, %d)\n", stack[i].row, stack[i].col);

printf(" (%d, %d)\n", row, col);

printf(" (%d, %d)\n", EXIT\_ROW, EXIT\_COL);

}

: 도착했을 경우, 차례대로 스택에 쌓인 경로들을 출력해준다. 그리고 현재 위치, 마지막으로 도착 위치를 출력해주면 처음 위치에서 도착 위치까지 거쳤던 좌표가 차례대로 나오게 된다.

else printf("The maze does not have a path \n");

: 도착하지 못하고 path가 끝나게 될 경우 찾지못했다는 메시지를 출력해준다.

}

int main() {

int i, j;

int tmp;

FILE\* fp;

fp = fopen("maze.txt", "r");

fscanf(fp, "%d %d", &EXIT\_ROW, &EXIT\_COL);

: maze.txt 파일을 파일포인터에 지정한 뒤, maze의 row와 col을 각각 EXIT\_ROW, EXIT\_COL 변수에 저장해주었다.

stack = (element\*)malloc(sizeof(element) \* (EXIT\_COL\*EXIT\_ROW) / 2);

maze = (int\*\*)malloc(sizeof(int\*) \* (EXIT\_ROW + 2));

for (i = 0; i < EXIT\_ROW + 2; i++) {

maze[i] = (int\*)malloc(sizeof(int) \* (EXIT\_COL + 2));

}

mark = (int\*\*)malloc(sizeof(int\*) \* (EXIT\_ROW + 2));

for (i = 0; i < EXIT\_ROW + 2; i++) {

mark[i] = (int\*)malloc(sizeof(int) \* (EXIT\_COL + 2));

}

: 스택 배열의 크기는 sizeof(element) \* (EXIT\_COL\*EXIT\_ROW) / 2 라고 잡아주었는데, 그 이유는 해당 크기의 maze에서 가장 긴 루트의 길이가 (EXIT\_COL\*EXIT\_ROW) / 2 이기 때문이다. 그리고 maze와 mark는 2차원 배열을 동적할당으로 생성해주었는데 총 크기가 (EXIT\_ROW + 2) \* (EXIT\_COL + 2) 이다. 왜냐하면 기본 주어진 maze를 둘러싸는 테두리에도 1을 할당해주어서 실제 미로로 가정하면 벽을 만들어 주기 위함이다.

move\_set();

: move\_set 함수를 호출하여 move[8]에 각각 direction에 해당하는 key를 멤버들에게 저장하였다.

i = 1;

j = 1;

: 벽을 제외하고 시작하는 위치인 (1,1)을 i와 j 인덱스 변수에 저장해주었다.

while (!feof(fp)) {

fscanf(fp, "%d", &tmp);

maze[i][j] = tmp;

mark[i][j] = 0;

j++;

if (j > EXIT\_COL) {

j = 1;

i++;

}

}

: maze.txt 파일에서 maze 데이터를 tmp에 하나씩 불러와서 tmp를 maze[i][j]에 저장해주었고, mark는 0으로 초기화 해주었다. 2차원 배열이므로 j가 한 줄을 채우고 나면 다시 1로 초기화 되고, row counter인 i가 증가하게 끔 하였다.

for (i = 0; i < EXIT\_ROW+2; i++) {

maze[i][0] = 1;

maze[i][EXIT\_COL + 1] = 1;

}

for (i = 0; i <= EXIT\_COL+2; i++) {

maze[0][i] = 1;

maze[EXIT\_ROW + 1][i] = 1;

}

: maze를 둘러싼 벽을 설정해주었다.

path();

: path 함수 호출.

for (i = 0; i < EXIT\_ROW + 2; i++) {

free(mark[i]);

}

free(mark);

free(stack);

fclose(fp);

: 동적 할당된 변수들의 메모리 해제와, fp 파일 포인터를 해제해주었다.

return 0;

}