Linked stack을 이용한 Maze

중앙대학교 소프트웨어학과 20186889 권용한

●소스코드 설명

```
typedef struct {
    short int row;
    short int col;
    short int dir;
}element;

typedef struct stack *stackPtr;
typedef struct stack{
    element map;
    stackPtr link;
}stack;
stackPtr top = NULL;
```

stack을 구현하는 코드로, stack의 data로는 row와 col, dir를 인수로 가지는 구조체인 element형의 map을 가지고, link list형식으로 스택을 사용하기 때문에 link역할(다음 데이터의 주소를 저장)을 위해 stackPtr형의 link를 가지고 있다. top은 NULL로 초기화 시키고 이후 push가 일어나면 top은 가장 나중에 들어온 값을 가리키게 된다.

```
void stack_empty( ) {
    printf("stack is empty, cannot delete element");
    exit(EXIT_FAILURE);
}
void push(element item) {
    stackPtr temp;
    temp = (stackPtr)malloc(sizeof(stack));
    temp->map = item;
    temp->link = top;
    top = temp;
}
element pop() {
    stackPtr temp = top;
    element item;
    if (!temp)
        stack_empty( );
    item = temp->map;
    top = temp->link;
    free(temp);
    return item;
}
```

stack을 연산하는 코드로, linked list형식을 쓰기 때문에, 컴퓨터의 메모리 사용 공간이 모자라서 malloc이 실패하지 않는다면 stack이 full이 되는 경우가 없다고 볼 수 있다. 때문에 일반적으로 배열로 선언된 stack과 비교하였을 때 stackfull은 정의하지 않은 것을 볼 수 있다.

stack이 linked list 형식이기 때문에 push와 pop이 배열로 선언되었을 때와는 다른데, pop의 경우 stack이 empty하지 않다면 가장 나중에 들어온 값을 item에 저장한 후, top을 두 번째로 늦게 들어온 것을 가리키게 한 후 가장 늦게 들어온 부분을 free처리하는 것을 확인할 수 있다. top이 두 번째로 늦게 들어온 것을 가리키게 한 다음엔 가장 늦게 들어온 값에 접근할 수 없기 때문에 먼저 item에 값을 저장해야한다.

push의 경우엔 stack의 크기 만큼을 동적으로 할당받고, 할당 받은 곳에 입력된 item을 저장하고, 이전에 가장 늦게 들어온 값이 저장된 곳을 link에 저장한 후, top이 자신을 가리키게 한다. 만일 top이 자신을 먼저 가리키게 한다면, 그 이전에 들어온 값들에 접근할 방법이 사라지기 때문에 순서가 틀리지 않도록 유의해야한다.

```
void setup_maze() {
    short int maze0[numRow][numCol] = { // 여기서 줄 바꾸어 주세요(1).
    { 0,1,1,1,1 },
    { 1,0,1,1,1 },
    { 1,0,0,1,1 },
    { 0,0,1,1,1 },
    { 1,0,0,0,0 } // 여기서 줄 바꾸어 주세요(2).
    };

    for (int i = 0; i < numRow + 2; i++){
        maze[0][i] = 1;
        maze[numRow + 1][i] = 1;
        maze[i][0] = 1;
        maze[i][numCol + 1] = 1;
    }
    for (int i = 0; i < numRow; i++){
        for (int j = 0; j < numCol; j++){
            maze[i + 1][j + 1] = maze0[i][j];
        }
    }
    // 2차원 배열 maze0[numRow][numCol]로부터
    // 전역변수인 2차원 배열 maze[numRow + 2][numCol + 2]를 설정 완료하는 코드
}
```

maze를 초기화 하는 함수이다. 이때 우리는 외벽부분(row와 col이 0 또는 numRow(numCol)+1)을 제외한 부분이

초기화된 maze0 배열이 있기 때문에, 이것을 고려하여 maze를 우리가 필요한 형태에 맞게 저장해야 할 필요가 있다. 때문에 첫 번째 for문에서는 외벽부분을 1로 저장하여 maze 이외의 부분엔 접근하지 못하도록 한다.
두 번째 for문은 maze0의 값을 우리가 실제로 사용하는 maze배열에 저장하는 코드로, maze에서는 시작지점이 1,1인 반면 maze0에서는 시작지점이 0,0이다. 또한 끝지점 또한 maze는 numRow,numCol인 반면 maze0의 끝지점은 numRow-1,numCol-1이다. 때문에 이것을 고려하여 maze[i+1][j+1]=maze0[i][j]를 해준다.

```
void drawmaze(int row,int col,bool found) {
    for (int i = 0; i < numRow; i++)
    {
        for (int j = 0; j < numCol; j++)
        {
            if (maze[i + 1][j + 1] == 1)
                mazech[i][j] = 'x';
            else
                mazech[i][j] = ' ';
        }
    }

    for (int i = 0; i < numRow; i++)
    {
        stackPtr temp = NULL;
        if (top != NULL) {
            temp = top;
            while(temp != NULL){
                mazech[(*temp).map.row - 1][(*temp).map.col - 1] = '*';
                temp = temp->link;
            }
        }
    }
    mazech[row-1][col-1] = '*';
    if (found == true)
        mazech[numRow - 1][numCol - 1] = '*';
    printf("row/col  1  2  3  4  5₩n");
    for(int i=0;i<numRow;i++)
        printf("  %d     %c  %c  %c  %c  %c₩n",i+1, mazech[i][0],mazech[i][1] , mazech[i][2], mazech[i][3], mazech[i][4]);
}
```

maze가 실행되면서 각 실행과정을 확인할 수 있도록 코드를 작성하기 때문에 그 각 과정을 눈으로 확인하기 위해서 maze의 접근 불가능한 부분(벽으로 x로 표현됨)과 이전까지의 이동경로와 현재 위치(*로 표현됨) 등이 표현할 수 있어야한다. 첫 번째 이중 for문은 이동 가능한 지역은 공백으로 남기고, 접근이 불가능한 부분(maze에서 1로 저장된 부분)은 x로 저장한다.
두 번째 for문은 이전의 이동 경로를 *로 표현하는 코드로, 이때 stack을 이용하게 된다. top이 null, 즉 이제 막 시작한 경우엔 stack에 아무 값도 없기 때문에 아무것도 실행하지 않고, stack에 값이 있다면 stackPtr로 선언된 temp변수를 통해 stack 내부를 검사하여 이전의 이동 경로를 *로 저장한다. top에서 바로 접근하게 된다면 한 번의 실행 이후 top이 null값이 될 것이고, 이전의 경로가 그 다음번엔 접근할 수 없게 되기 때문에 top으로 바로 접근하는 것이 아니라 temp라는 stackPtr형 변수에 top의 주소 값을 받고 temp를 통해 top의 값에 변화 없이 그 이후 값들에 접근할 수 있다.
그 밑에 mazech[row-1][col-1]='*'은 현재 위치를 표현하는 것이고, 그 밑에 if(found == true)는 maze가 탈출에 성공하게 된다면 그때의 현재 위치는 새로 업데이트되지 않고, stack도 업데이트 되지 않기 때문에 그 이전과 같은 값을 가지게 된다. 하지만 numRow,numCol에 도달한 것과 도달하지 않은 것을 구분해야 하기 때문에 found로 그 상태를 확인하고, 만일 found가 true라면 도착 지점에도 *을 표시한다.
그 밑의 for문은 이 모든 과정을 통해 저장된 maze를 print하는 코드이다.

```
mark[1][1] = 1;
row = 1;
col = 1;
dir = 1;

drawmaze(row, col, found);
printf("continue(Y/N)? : ");
```

```
while (top != NULL && !found)
{
    position = pop( );
    row = position.row;
    col = position.col;
    dir = position.dir;
    drawmaze(row, col, found);
    printf("continue(Y/N)? : ");
```

backtracking을 하지 않을 경우 backtracking을 하였을 경우
여기서부터는 maze의 경로를 탐색하는 path함수이다.
backtracking을 하지 않고, 목적지인 5,5까지 한번에 갈 수 있는 maze만을 다루기 때문에 일반적인 maze의 코드부분에서 while(top!=NULL&&!found)부분이 없어도 된다. stack에서 pop을 하지 않기 때문에 stack의 top이 NULL값이 될 수 없으며, 5,5에 도달하기 전까진 found가 false인 상황이고 원래의 maze코드에서 while문 내부에 또 다른 while문에 속해서 목적지에 도달하기 전까지 계속 loop를 돌기 때문에 두 번째 while을 탈출할 일이 없다. 때문에 원래의 maze코드의 첫 번째 while loop가 생략된다고 해도 프로그램이 문제없이 돌아간다. 그리고 backtracking을 하

지 않기 때문에 처음에 pop을 실행할 필요도 없어졌다.
이것과 비교하기 위해서 backtracking을 하였을 때의 코드부분의 사진을 같이 첨부하였다. stack에 저장되어 있던 첫 지점을 pop한 후, 첫 지점에 도착한 것을 그린다.(drawmaze) 그리고 만일 dir가 8을 넘어 backtracking이 필요하다면 이 부분에서 backtracking이 일어나게 된다.

```
do {
    scanf("%c", &c);
    if (c == 'N' || c == 'n')
    {
        printf("NO! 입력되어서 프로그램을 종료합₩n");
        exit(EXIT_FAILURE);
    }
    else if (c == 'Y' || c == 'y')
    {
        a = 0;
    }
    else {
        printf("Y나 N를 입력해 주십시오₩n");
        printf("continue(Y/N)? :");

        a = 1;
    }
    c = getchar();
} while (a);
```

continue? 부분에 y 또는 n가 입력되지 않는다면 다시 입력하도록 하고, y가 입력된다면 loop를 탈출해 그 다음과정을 진행하고, n이 입력된다면 그 이후 과정을 진행하지 않는다고 판단하여 프로그램을 종료하도록 코드를 구현하였다. 우선 한번은 실행하고 판단한 뒤에 loop를 실행하기 때문에 do while문이 적합하다 판단하여 do while문을 사용하였다.

```
while (dir < 8 && !found)
{
    nextRow = row + move[dir].vert;
    nextCol = col + move[dir].horiz;
    if (nextRow == numRow && nextCol == numCol) {
        found = true;
        drawmaze(row, col, found);
    }
    else if (!maze[nextRow][nextCol] && !mark[nextRow][nextCol]) {
        mark[nextRow][nextCol] = 1;
        position.row = row;
        position.col = col;
        position.dir = ++dir;
        push(position);
        row = nextRow;
        col = nextCol;
        dir = 0;
        drawmaze(row, col, found);
        printf("continue(Y/N)? :");
```

이 부분은 그 다음 지점이 legal path인 부분으로 그 지점이 목적지이거나 목적지가 아닐 때의 상황을 표현한 코드이다. 책의 내용과 일치하며, 중간 실행동안 maze를 그려야하기 때문에 stack에 값을 저장하고, 현재 위치를 업데이트 한 이후에 drawmaze함수를 호출하고, 그 다음 과정을 할지 묻는다. 만약 그 지점이 목적지라면 현재위치가 업데이트 되지 않지만 found를 같이 drawmaze에 넘겨줘서 그 이전까지의 이동 경로와 목적지의 바로 전 위치(row,col) 그리고 목적지 까지 *로 표현하게 된다.

```
int main() {

    element free;//maze가 결과를 찾은 이후 동적할당된 top 변수들을 free하기 위한 변수

    initial_mark();
    setup_maze();
    path();

    while (top != NULL){
        free = pop();
    }

    return 0;
}
```

이 코드의 main함수 부분으로 처음에 mark를 다 0으로 초기화 한 이후 maze도 초기화를 한다. 그런 다음 path함

수를 통해 목적지까지 경로를 찾고 그 과정을 시각적으로 확인한 다음, 마지막으로 stack에 동적으로 할당된 변수를 해제한다.

●출력결과

```
short int maze0[numRow][numCol] = { // 여기서 줄 바꾸어 주세요(1).
{ 0,1,1,1,1 },
{ 1,0,1,0,1 },
{ 1,0,0,1,0 },
{ 0,0,1,0,1 },
{ 1,0,0,1,0 } // 여기서 줄 바꾸어 주세요(2).
};
```

```
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x     x     x
   3     x        x
   4           x     x
   5     x        x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x     x
   3     x        x
   4           x     x
   5     x        x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x     x
   3     x     *  x
   4           x     x
   5     x        x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x  *  x
   3     x     *  x
   4           x     x
   5     x        x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x  *  x
   3     x     *  x  *
   4           x     x
   5     x        x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x  *  x
   3     x     *  x  *
   4           x  *  x
   5     x        x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x  *  x
   3     x     *  x  *
   4           x  *  x
   5     x        x  *
계속하려면 아무 키나 누르십시오 . . .
```

위 결과는 과제 pdf에서 주어진 예시 maze의 결과 값으로 backtrack이 없는 maze 중 입구부터 출구까지 경로가 존재하는 것만을 대상으로 하기 때문에 실행 과정 중 backtracking 없이 목적지(5,5)에 도달하는 것을 확인 할 수 있다.

1,1>2,2>3,3>2,4>3,5>4,4>5,5

```
short int maze0[numRow][numCol] = { // 여기서 줄 바꾸어 주세요(1).
{ 0,1,0,1,1 },
{ 1,0,1,0,1 },
{ 1,0,0,1,1 },
{ 0,0,1,0,1 },
{ 1,0,0,1,0 } // 여기서 줄 바꾸어 주세요(2).
};
```

```
row/col  1  2  3  4  5
   1     *  x     x  x
   2     x     x     x
   3     x        x  x
   4           x     x
   5     x        x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x     x  x
   2     x  *  x     x
   3     x        x  x
   4           x     x
   5     x        x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  *  x  x
   2     x  *  x     x
   3     x        x  x
   4           x     x
   5     x        x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  *  x  x
   2     x  *  x  *  x
   3     x        x  x
   4           x     x
   5     x        x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  *  x  x
   2     x  *  x  *  x
   3     x     *  x  x
   4           x     x
   5     x        x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  *  x  x
   2     x  *  x  *  x
   3     x     *  x  x
   4           x  *  x
   5     x        x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  *  x  x
   2     x  *  x  *  x
   3     x     *  x  x
   4           x  *  x
   5     x        x  *
계속하려면 아무 키나 누르십시오 . . .
```

이것은 위에 주어진 maze를 바탕으로 한 실행 결과이다. maze 조건은 앞의 경우와 동일하고 북쪽에서 시작하여 시계방향으로 이동 가능한 방향 중 legal한 곳만 지나는 것을 확인할 수 있다.

1,1>2,2>1,3>2,4>3,3>4,4>5,5

```
short int maze0[numRow][numCol] = { // 여기서 줄 바꾸어 주세요(1).
{ 0,1,1,1,1 },
{ 1,0,1,1,1 },
{ 1,0,0,1,1 },
{ 0,0,1,1,1 },
{ 1,0,0,0,0 } // 여기서 줄 바꾸어 주세요(2).
};
```

```
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x     x  x  x
   3     x        x  x
   4           x  x  x
   5     x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x  x  x
   3     x        x  x
   4           x  x  x
   5     x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x  x  x
   3     x     *  x  x
   4           x  x  x
   5     x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x  x  x
   3     x     *  x  x
   4        *  x  x  x
   5     x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x  x  x
   3     x  *  *  x  x
   4        *  x  x  x
   5     x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x  x  x
   3     x  *  *  x  x
   4     *  *  x  x  x
   5     x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x  x  x
   3     x  *  *  x  x
   4     *  *  x  x  x
   5     x  *
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x  x  x
   3     x  *  *  x  x
   4     *  *  x  x  x
   5     x  *  *
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x  x  x
   3     x  *  *  x  x
   4     *  *  x  x  x
   5     x  *  *  *
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x  x  x
   3     x  *  *  x  x
   4     *  *  x  x  x
   5     x  *  *  *  *
계속하려면 아무 키나 누르십시오 . . .
```

이것은 위의 maze의 결과 값으로 maze 조건은 앞의 것들과 동일하며, 북쪽부터 이동 가능한지 조사하기 때문에 (4,2)에서 (3,2)로 이동한 뒤 계속 실행되는 것을 확인할 수 있다.

1,1>2,2>3,3>4,2>3,2>4,1>5,2>5,3>5,4>5,5

```
   1     *  x  x  x  x
   2     x     x     x
   3     x        x
   4           x     x
   5     x        x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x     x
   3     x        x
   4           x     x
   5     x        x
continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x     x
   3     x     *  x
   4           x     x
   5     x        x
continue(Y/N)? :e
Y나 N을 입력해 주십시오continue(Y/N)? :y
row/col  1  2  3  4  5
   1     *  x  x  x  x
   2     x  *  x  *  x
   3     x     *  x
   4           x     x
   5     x        x
continue(Y/N)? :n
NO이 입력되어서 프로그램을 종료함
계속하려면 아무 키나 누르십시오 . . .
```

만일 continue에서 y나 n이 아닌 다른 문자가 입력된다면 다시 문자를 받고, y가 입력된다면 출력을 계속하고, n이 입력된다면 출력을 멈추고 프로그램을 종료한다.