12주차 결과보고서

전공: 철학과 학년: 3학년 학번: 20180032 이름: 남기동

1. 미로 2주차 함수들의 pseudo code

1) readFile 함수

우선 파일의 정보를 한번 쭉 읽어서 미로의 높이와 너비를 구한다.

```
ofBuffer buffer(file);

for (first line to last line) {
        HEIGHT++;
        WIDTH = line.size();
}
```

2차원 배열 maze에 미로의 정보를 넣을 것이므로 높이와 너비를 참고하여 메모리를 동적 할당한다

```
maze = new int* [HEIGHT];
for (i = 0; i < HEIGHT; i++) {
        maze[i] = new int[WIDTH];
}
```

파일의 내용을 읽어서 maze에 저장한다. 모서리는 1, 가로벽은 2, 세로벽은 3, 없어진 가로벽 혹은 세로벽은 4, 미로의 방은 0으로 간주한다.

```
for (first line to last line) {
        for (j = 0; j < WIDTH; j++) {
                if( 모서리 ) maze[i][j] = 1
                else if (가로벽) maze[i][j] = 2;
                else if (세로벽) maze[i][j] = 3;
                else if (없어진 세로벽) maze[i][j] = 4;
                else if (없어진 가로벽) maze[i][j] = 4;
                else maze[i][j] = 0;
        }
}
```

Maze를 이용하여 graph를 만들기 위해 필요한 작업들을 한다. Graph는 미로의 방을 기준으로 정보를 저장했기 때문에 새로운 미로의 높이와 너비를 설정한다. 그리고 graph에 동적 할당한다.

```
int number = 0;
height_graph = HEIGHT / 2;
```

```
width_graph = WIDTH / 2;
num_of_terms = height_graph * width_graph;
graph = new node_pointer[num_of_terms];
```

maze의 미로의 방마다 graph의 노드로 변환하는 작업을 한다. 이때 아직 link는 설정하지 않고 모두 NULL로 초기화한다. 왜냐하면 위에서부터 차례대로 노드를 읽어서 만드는데 아직 못 만든 노드를 가리키는 경우도 있을 수 있기 때문이다.

```
for (i = 0; i < HEIGHT; i++) {
        for (j = 0; j < WIDTH; j++) {
                if (미로의 방인 경우) {
                        node_pointer temp = new node;
                        temp->row = i; temp->col = j; //새로만든 node의 기본 정보 초기화
                        temp->index = number;
                        temp->link[0] = NULL; temp->link[1] = NULL;
                        temp->link[2] = NULL; temp->link[3] = NULL;
                        graph[number++] = temp;
                }
        }
}
```

마지막으로 초기화된 graph의 노드들을 차례대로 읽어가면서 열려 있는 방향에 해당하는 노드와 edge를 만들어준다.

```
for (i = 0; i < num_of_terms; i++) {
        node_pointer ptr = graph[number];
        if (노드 기준 위쪽이 열려 있으면) ptr->link[0] = 위쪽 노드;
        if (노드 기준 오른쪽이 열려 있으면) ptr->link[1] = 오른쪽 노드;
        if (노드 기준 아래쪽이 열려 있으면) ptr->link[2] = 아래쪽 노드;
        if (노드 기준 왼쪽이 열려 있으면) ptr->link[3] = 왼쪽 노드;
        number++; //다음 노드로 넘어감
}
```

2) freeMemory 함수

```
for (i=0; i< HEIGHT; i++)
        delete[] maze[i];
delete[] maze;
delete[] graph;
```

이차원 배열로 할당했던 maze를 먼저 해제해주고 node_pointer의 자료형을 갖고 있으며 1차원

배열인 graph도 해제해준다.

3) draw 함수

파일이 열려 있는 경우에만 graph의 노드들을 하나씩 차례대로 조사하면서 위쪽, 오른쪽, 아래쪽, 왼쪽 각각에 연결된 노드가 없다면 벽인 것을 의미하기 때문에 그런 경우에만 ofDrawLine을 사용하여 벽을 그린다. 한 줄 내에서는 노드를 그리고 다음 노드로 넘어갈 때 y값은 고정이지만 x값은 줄의 길이에 해당하는 line_length만큼 이동하여 그린다. 비슷하게 한 줄을 다 그리고 아래 줄을 그리기 위해 이동할 때는 x값을 처음 시작점으로 옮기고 y값은 line_length만큼 이동하여 노드를 그리기 시작한다.

```
if (isOpen) {
        for (i = 0; i < height_graph; i++) {
                for (j = 0; j < width_graph; j++) {
                        if (graph[number]->link[0] == NULL) ofDrawLine(x, y, x + line_length,
y);
                        if (graph[number]->link[1] == NULL) ofDrawLine(x + line_length, y, x
+ line_length, y + line_length);
                        if (graph[number]->link[2] == NULL) ofDrawLine(x, y + line_length, x
+ line_length, y + line_length);
                        if (graph[number]->link[3] == NULL) ofDrawLine(x, y, x, y +
line_length);
                        x += line_length;
                        number++;
                }
                x = 10;
                y += line_length;
        }
}
```

2. 자료구조와 자료구조를 그리는 방법, 해당 자료구조를 이용한 그래픽 전환 작업의 시간 공간 복잡도, 실험 전과 생각한 방법의 다른 점

미로의 정보를 저장하기 위해 선택한 자료구조는 그래프이다. 그래프는 vertex와 edge로 이루어져 있는 자료구조로 vertex를 노드로 생각하고 edge를 노드의 link로 생각하여 자료구조를 설계했다.

```
typedef struct _node* node_pointer;
typedef struct _node {
```

```
        int row;
        int col;
        int index;
        node_pointer link[4]; //0은 uo, 1은 right, 2는 down, 3은 left를 의미한다
}node;
```

이때 link는 일차원 배열로 0부터 4까지 가능한데 0은 위쪽, 1은 오른쪽, 2는 아래쪽, 3은 왼쪽으로 이동할 수 있는 길이 있는지를 의미하며 벽이 뚫려 있어서 길이 있다면 연결되어 있는 방을 가리키는 노드를 포인터로 가리키게 한다.

그러나 이 자료구조를 구성하기 이전에 파일의 정보를 담는 maze를 따로 사용했는데, maze는 int형 2차원배열이다. 각각의 term은 정수를 갖고 있는데 1은 미로에서 모서리('+')를 의미하고 2는 가로벽(-), 3은 세로벽(|), 4는 가로벽이나 세로벽이 없어진 자리, 0은 미로의 방을 의미한다. Maze로 변환하여 받아온 파일의 정보를 graph의 node로 변환하여 저장하는 방식을 사용했다.

우선 파일을 읽어서 미로의 높이와 너비를 구한 다음 HEIGHT와 WIDTH에 저장한다

```
ofBuffer buffer(file);

for (ofBuffer::Line it = buffer.getLines().begin(), end = buffer.getLines().end(); it != end;
++it) {
        HEIGHT++;
        string line = *it;
        WIDTH = line.size();
}
```

Int형 2차원 배열 maze에 미로의 정보를 넣을 것이기 때문에 maze에 적절한 메모리 크기를 동적으로 할당한다

```
maze = new int* [HEIGHT];
for (i = 0; i < HEIGHT; i++) {
        maze[i] = new int[WIDTH];
}
```

파일의 내용을 읽어서 maze에 저장한다. 이때, 모서리는 1로, 가로벽은 2로, 세로벽은 3으로, 그리고 세로벽과 가로벽들이 있어야 할 위치인데 빈칸인 경우에는 세로벽과 가로벽이 없어진 것을 의미하므로 이를 나타내기 위해 4를 대입하고 나머지는 미로의 방을 나타내기 위해 0을 대입한다.

```
for (ofBuffer::Line it = buffer.getLines().begin(), end = buffer.getLines().end(); it != end;
++it, i++) {
        string line = *it;
        for (j = 0; j < WIDTH; j++) {
                if (line[j] == '+') maze[i][j] = 1;
```

```
                else if (line[j] == '-') maze[i][j] = 2;
                else if (line[j] == '|') maze[i][j] = 3;
                else if ((i % 2 == 1) && (j % 2 == 0) && line[j] == ' ') maze[i][j] = 4;
                else if ((i % 2 == 0) && (j % 2 == 1) && line[j] == ' ') maze[i][j] = 4;
                else maze[i][j] = 0;
        }
}
```

그 다음으로 graph를 만들기 위해 선작업이 필요하다. Graph의 index를 가리킬 number를 0으로 선언하고 height_graph와 width_graph는 각각 HEIGHT와 WIDTH를 2로 나눈 몫을 저장하는데, 이것은 실질적으로 미로의 방만을 따졌을 때의 미로의 높이와 미로의 너비를 의미한다. 그리고 미로의 방의 개수를 num_of_terms로 정의하고 graph에 node_pointer 자료형 크기를 num_of_terms 만큼 곱하여 메모리를 할당한다.

```
int number = 0;
height_graph = HEIGHT / 2;
width_graph = WIDTH / 2;
num_of_terms = height_graph * width_graph;
graph = new node_pointer[num_of_terms];
```

이어서 maze를 읽어서 새로운 자료구조인 graph의 모든 node를 구성하고 초기화한다. Node는 미로에서의 row와 col 값을 갖고 있으며 자신이 몇 번째 노드인지를 나타내는 index와 link[4]를 갖는다. Link[4]는 위, 오른쪽, 아래, 왼쪽에 해당하는 edge를 상징한다. 초기화이기 때문에 link[4]의 값들은 모두 NULL로 저장한다. 나중에 각각의 노드를 읽을 때 link의 값이 NULL인 것은 해당 방향으로 벽이 만들어져 있음을 의미한다.

```
for (i = 0; i < HEIGHT; i++) {
        for (j = 0; j < WIDTH; j++) {
                if ((i % 2 == 1) && (j % 2 == 1)) { //둘다 홀수인 경우가 maze의 방에 해당한다
                        node_pointer temp = new node;
                        temp->row = i; temp->col = j; //새로만든 node의 기본 정보 초기화
                        temp->index = number;
                        temp->link[0] = NULL; temp->link[1] = NULL;
                        temp->link[2] = NULL; temp->link[3] = NULL;
                        graph[number++] = temp;
                }
        }
}
```

마지막으로 초기화된 graph의 모든 노드들을 순서대로 읽어가면서 열려 있는 방향의 link를 열려

있는 방향의 node에 연결시켜서 edge를 만든다.

```
number = 0;
for (i = 0; i < num_of_terms; i++) {
        node_pointer ptr = graph[number];
        if (maze[ptr->row - 1][ptr->col] == 4) ptr->link[0] = graph[ptr->index - width_graph];
        if (maze[ptr->row][ptr->col + 1] == 4) ptr->link[1] = graph[ptr->index + 1];
        if (maze[ptr->row + 1][ptr->col] == 4) ptr->link[2] = graph[ptr->index + width_graph];
        if (maze[ptr->row][ptr->col - 1] == 4) ptr->link[3] = graph[ptr->index - 1];
        number++;
}
```

파일을 읽고 graph의 노드를 만들었으니 draw함수를 사용하여 화면에 미로를 그려야 한다. draw 함수는 파일을 읽으면서 새롭게 정의한 graph의 노드들을 하나씩 읽어가면서 노드의 4방향에 해당하는 link 값이 NULL이면 다른 노드와 연결되어 있지 않다는 것을 의미하기 때문에 벽을 그려주고 그렇지 않다면 빈 공간으로 남겨두는 방식을 사용하였다.

```
char str[256];
ofSetColor(100);
ofSetLineWidth(5);
int i, j; int number = 0;
int x = 10, y = 10; //starting point
int line_length = 35;

if (isOpen) {
        for (i = 0; i < height_graph; i++) {
                for (j = 0; j < width_graph; j++) {
                        if (graph[number]->link[0] == NULL) ofDrawLine(x, y, x + line_length,
y);
                        if (graph[number]->link[1] == NULL) ofDrawLine(x + line_length, y, x
+ line_length, y + line_length);
                        if (graph[number]->link[2] == NULL) ofDrawLine(x, y + line_length, x
+ line_length, y + line_length);
                        if (graph[number]->link[3] == NULL) ofDrawLine(x, y, x, y +
line_length);
                        x += line_length;
                        number++;
                }
                x = 10; //leftmost(starting point)
                y += line_length;
        }
}
```

x, y는 미로를 그릴 때의 시작점이며 line_length는 한 노드를 기준으로 벽을 그릴 때 벽의 길이를 의미한다. 그리고 파일이 열려 있는 경우에만 graph의 노드들이 생성되기 때문에 if(isOpen)을 사용해야 한다. 미로의 높이와 너비를 이중 루프로 돌면서 노드를 차례대로 조사하여 위, 오른쪽,

아래, 왼쪽에 해당하는 방향의 link 값이 NULL인 경우에 벽을 그려준다. 하나의 노드를 다 그렸으면 다음 노드를 그리기 위한 꼭짓점으로 이동하기 위해 x의 값에 line_length를 더하고 graph의 다음 node를 가리키기 위한 인덱스 역할을 하는 number도 1 증가시킨다. 그러다가 한 줄(열)을 다 그렸으면 처음의 x 값으로 돌아오고 y 값은 그 때 line_length만큼 증가시켜서 한 줄 아래 맨 왼쪽에서부터 다시 노드를 조사하고 그릴 수 있도록 한다.

시간복잡도는 미로의 높이를 HEIGHT, 미로의 너비를 WIDTH라고 할 때, O(HEIGHT * WIDTH)이다. 왜냐하면 파일의 정보를 읽어서 maze라는 이차원 배열에 저장하는 것이 HEIGHT와 WIDTH를 이중 루프로 돌기 때문이다. 이후 maze를 이용하여 graph를 만들 때도 HEIGHT와 WIDTH를 이중 루프로 돌아 시간 복잡도가 동일하며 graph의 정보를 처리하고 draw함수에서 graph의 정보를 읽어서 화면에 출력하는 과정은 height_graph와 width_graph를 이중 루프로 도는데 이 값들도 HEIGHT와 WIDTH를 2로 나눈 몫이기 때문에 화면에 미로를 그리기 위한 시간 복잡도는 O(HEIGHT * WIDTH)이다. 공간 복잡도는 maze와 graph 이 두 개를 동적 할당하여 메모리의 크기를 따져봐야 하는데, maze의 크기는 HEIGHT*WIDTH이며 graph의 크기는 height_graph*width_graph이다. 따라서 공간 복잡도도 O(HEIGHT * WIDTH)이다.
예비 보고서에서 생각했던 부분과 크게 달라진 점이 없이 동일하게 실습에서 작성하였고, 프로그램에 원활하게 돌아가는 것을 확인했다.

3. 습득한 내용이나 느낀점

미로 2주차 실습을 하면서 습득한 내용은 크게 두 가지이다. 첫 번째는 윈도우 창을 만들고 윈도우 창에 메뉴와 세부 메뉴, 버튼 등을 만드는 것을 학습했다. 실습에서 실질적으로 윈도우와 메뉴를 만드는 부분을 수정하지는 않았지만 기존에 단순히 화면에 결과를 띄우거나 특정 키를 입력받아, 혹은 마우스 클릭으로 결과를 출력하게끔 하는 것과 달리 윈도우 창에서 메뉴 창을 선택하는 방식을 구현한 것이 새로웠고 해당 부분을 읽고 이해하는 과정에서 배워야 할 부분이 많다는

생각이 들었다.

두 번째로 배운 점은 미로의 자료 구조를 설계할 때 이후에 추가될 기능을 고려하여 graph로 구현하는 과정과 관련되어 있다. 미로 1주차 실습에서는 단순히 int형의 2차원 배열을 활용하여 미로를 만드는 프로그램을 설계했다. 그러나 이번 2주차 실습에서는 3주차에서 수행할 BFS와 DFS라는 알고리즘을 수행하기 위해 적절한 자료구조를 설계하는 것이 관건이었다. 때문에 단순히 2차원 배열을 동일하게 사용하는 것이 아니라 BFS와 DFS를 학습하고 어떤 원리로 작동하는 알고리즘인지 알아보고 그러한 작업을 원활하게, 효율적으로 수행하기 위한 자료 구조를 구상하고 그에 맞게 draw함수를 설계하는 과정을 거쳤다. 이러한 과정에서 단순히 프로그래밍을 할 때, 직관적으로 떠오르고 간단하고 쉬운 자료 구조를 사용하는 것이 아니라 최종적으로 어떤 역할을 수행할지를 생각하고 그에 맞추어 자료구조부터 효율적인 것을 구상하는 과정이 이후 프로그래밍 학습에 큰 도움이 될 경험이었다고 생각한다.