12주차 결과보고서

전공: 신문방송학과 학년: 3학년 학번: 20191150 이름: 전현길

1. 실험시간에 작성한 프로그램에서 자료구조와 구성한 자료구조를 화면에 그리는 방법들을 설명한다. 완성한 자료구조를 이용한 그래픽 전환 작업의 시간 및 공간복잡도를 보이고 실험 전에 생각한 방법과 어떻게 다른지 아울러 기술한다.

구성한 자료구조는 크게 다음과 같다.

1. **char** input**: 미로의 형태를 저장하는 2차원 char 배열

2. **int** visited**: 미로의 방문 가능한 정점을 저장하는 2차원 int 배열

파일을 읽고 메모리를 할당하는 작업은 readFile(), 그래픽 전환 작업은 draw(), 동적 할당된 메모리를 해제하는 작업은 freeMemory()에서 수행했다. 실험 전에 int** visited 배열을 사용해야 하는 것을 감안하지 못했기 때문에, visited 배열에 메모리를 할당하고 적절한 값으로 초기화하는 코드를 추가적으로 작성했다.

DFS, BFS 알고리즘을 위해 이전에 kruskal 알고리즘으로 미로를 생성했을 때처럼 간선을 저장하는 자료구조를 따로 구현할 필요는 없고, visited 배열과 stack 자료구조, queue 자료구조로 대체할 수 있다.

readFile()에서 파일을 읽고 input 2차원 배열에 복사하는 작업, visited 배열과 input 배열에 적절한 값을 초기화하는 작업에 대해 모두 **O(height * width)의 시간 복잡도, 공간 복잡도**가 발생한다.

아래는 각 함수별로 수정한 부분에 대한 설명과 작성한 코드이다.

bool ofApp::readFile()

readFile() 함수는 .maz 확장자의 텍스트 파일을 읽어 input, visited 2차원 배열과 graph 2차원 벡터에 메모리를 동적 할당, 초기화한다. 수정한 부분의 함수의 동작은 다음과 같다. 정상적으로 종료되면 true 값을 반환한다.

1. string 벡터 tMaze()에 파일을 한 줄씩 복사한 뒤, 넓이·높이 변수를 초기화한다.
2. input, visited 배열에 메모리를 높이, 넓이만큼 복사한다.
3. 미로의 형태에 따라 visited 배열을 초기화한다.
   a. 미로가 벽이라면 visited[i][j] = 1, 비어 있다면 visited[i][j] = 0으로 초기화한다.

```
vector<string> tMaze;
HEIGHT = 0;
for (auto line : buffer.getLines()) {
  tMaze.push_back(line);
  WIDTH = line.length();
  HEIGHT++;
}

input = new char*[HEIGHT + 1];
visited = new int*[HEIGHT + 1];
for (int i = 0; i < HEIGHT; i++) {
  input[i] = new char[WIDTH + 1];
  visited[i] = new int[WIDTH + 1];
}

for (int i = 0; i < HEIGHT; i++)
  for (int j = 0; j < WIDTH; j++)
    input[i][j] = tMaze[i][j];

for (int i = 0; i < HEIGHT; i++)
  for (int j = 0; j < WIDTH; j++) {
    visited[i][j] = 1;
    if (input[i][j] == ' ') {
      visited[i][j] = 0;
  }

return true;
}
```

void draw()

자료구조를 활용하여 미로를 그린다. 적절한 크기로 직육면체를 그려주면 된다. '+'는 모서리가 되는 부분이므로 정사각형으로 처리하고, '-'는 가로로 긴 직사각형, '|'는 세로로 긴 직사각형으로 처리한다. i와 j값을 이용해 간단히 좌표 값을 계산해 그릴 수 있다.

미로를 그리는 과정에서 input 배열을 전부 순회하므로 시간 복잡도는 O(HEIGHT * WIDTH)이다.

```
// 저장된 자료구조를 이용해 미로를 그린다.
// add code here
for (i = 0; i < HEIGHT; i++) {
  for (j = 0; j < WIDTH; j++) {
    // 그리기
    char c = input[i][j];
    if (c == ' ') continue;
    else if (c == '+') ofDrawRectangle(i * 15, j * 15, 15, 15);
    else if (c == '-') ofDrawRectangle(i * 15, j * 15, 15, 30);
    else if (c == '|') ofDrawRectangle(i * 15, j * 15, 30, 15);
  }
}
```

bool ofApp::freeMemory()

input 배열, visited 배열에 동적 할당한 메모리를 delete[] 함수를 이용해 해제해 준다.

```
void ofApp::freeMemory() {
  // TO DO: malloc한 memory를 free해주는 함수

  for (int i = 0; i < HEIGHT; i++) {
    delete[] input[i];
    delete[] visited[i];
  }
  delete[] input;
  delete[] visited;
}
```

2. 본 실험을 통해 습득한 내용을 기술하시오.

원래는 2차원 간선 벡터를 구현해서 DFS, BFS 알고리즘을 구현해야 하는지 고민했으나, pair<int, int> 형태의 자료형으로 스택, 큐를 구성하면 된다는 것을 중간에 깨달았다. 문제를 정확히 파악하고 그에 맞는 적절한 자료구조를 선택하는 것이 작업의 효율성을 위해서도, 프로그램의 성능을 위해서도 중요함을 되새길 수 있었다.