12주차 결과보고서

전공: 경영학과 학년: 3학년 학번: 20190963 이름: 한다현

**1.**

이번주차 실험에서 사용한 자료구조는 크게 세 가지이다. 먼저 미로의 모든 문자를 저장하는 maze라는 이차원 배열을 사용하였다. 이차원 배열에서 방에 해당하는 부분은 방의 번호를 저장하였기 때문에 정수형 이차원 배열을 사용하였다. 다음으로, .maz 파일에서 ‘+’ 문자의 좌표를 저장하기 위하여 구조체 배열을 사용하였다. 구조체는 정수형 변수인 x1, y1을 저장하는 구조체이고, 각각은 ‘+’ 문자의 x, y 좌표를 저장한다. 이러한 구조체를 사용하여 dots라는 구조체 배열을 사용하였고, .maz 파일의 모든 ‘+’에 대한 좌표를 저장한다. 마지막으로 사용한 자료구조는 vector 배열이다. 이 배열은 방의 번호, 즉 vertex 간의 연결에 대한 정보를 저장하기 위해 사용하였다. 이 자료구조는 이번주차 실험에서는 필요하지 않지만 다음 주차 실험인 DFS와 BFS에 사용될 것이다.

이번 실험에서는 세 개의 함수를 작성하였다. 먼저 readFile() 함수이다. 이 함수는 .maz 파일을 읽어들여 위에 설명한 세 개의 자료구조에 적절한 데이터를 저장하는 함수이다. 템플릿 코드 대로 파일을 성공적으로 열었다면, 파일을 한 줄씩 읽는다. 파일을 한 줄씩 읽을 때마다 cnt 변수의 값을 1씩 증가시키고, 파일 한 줄의 길이를 WIDTH에 저장한다. 이 반복문이 끝나면 HEIGHT에 cnt를 저장하고 maze\_col를 (WIDHT-1)/2, maze\_row를 (HEIGHT-1)/2로 저장한다. 그 다음으로는 dots 배열에 대한 작업을 실행한다. 먼저 이차원 배열인 dots를 new를 사용해 동적할당한다. dots[i]를 모두 동적할당한 후에는 dots[0][0].x1과 dots[0][0].y1을 0으로 초기화한다. dots의 첫번째 행에 대해 각각의 x, y 좌표를 저장한다. 같은 행에 속한 점들은 y좌표는 동일하고, x 좌표만 이전 ‘+’에 비해 증가할 것이므로 for loop을 통해 dots[0][i].x1 = dots[0][i-1].x1+30, dots[0][i].y1 = 0으로 저장한다. 이 반복문이 끝나면 두번째 행부터 마지막 행까지의 작업을 실행한다. 이는 이중 for loop으로 구성되어 있으며 x 좌표는 이전 행의 같은 열에 해당하는 ‘+’와 같을 것이고, y 좌표는 이전 행의 y 좌표에 비해 증가할 것이므로 dots[i][j].x1 = dots[i - 1][j].x1, dots[i][j].y1 = dots[i - 1][j].y1 + 30 와 같이 저장한다. ‘+’의 좌표를 저장하고 난 후에는 이차원 배열인 maze를 동적할당한다. maze[i]까지 모두 동적할당을 한 후에는 다시 파일을 한 줄씩 읽어들인다. 이 때 maze의 행과 열이 모두 홀수일 때가 maze의 방이라는 것을 의미하기 때문에 maze에 vertex를 저장하고 vertex를 1 증가시킨다. 방이 아닐 때는 읽어들인 문자를 maze에 저장한다. 이렇게 maze에 대한 정보도 모두 저장한 후에는 graph라는 vector 배열을 동적할당한다. graph에 대한 작업은 3중 for loop으로 이루어져 있다. 첫번째 for loop은 i가 1부터 HEIGTH까지 2씩 증가하고 두번째 for loop은 j가 1부터 WIDTH까지 2씩 증가한다. 즉, 이 두 개의 반복문은 maze의 방을 탐색하는 반복문이다. 마지막 for loop은 k가 0부터 3까지 총 4번 반복하는데, 한 방에서 네 방향에 있는 다른 방과 연결되어 있는지 검사하기 위함이다. k가 0이면 위쪽 벽이 막혀있는지 검사하고 그렇지 않다면 push\_back을 사용해 위쪽 방과 연결되어 있음을 나타낸다. k가 1일 때는 오른쪽 벽이 막혀있는지 검사하고 그렇지 않다면 오른쪽 방과 연결되어 있음을 graph에 저장한다. 이와 같은 과정을 아래쪽과 왼쪽 벽에도 동일하게 실행한다.

draw() 함수는 저장된 자료구조를 이용해 미로를 그리는 함수이다. 미로를 그릴 때는 이중 for loop을 사용하는데 두 개의 반복문은 각각 HEIGHT와 WIDTH까지 반복한다. maze[i][j]가 ‘-‘ 일 때 두 개의 점을 이용해 선을 그린다. ‘-‘와 인접한 두 개의 ‘+’의 x, y 좌표를 찾고, 이를 ofDrawLine 함수의 인자로 사용하여 선을 그린다. maze[i][j]가 ‘|’일 때도 마찬가지로 ‘|’와 인접한 두 개의 ‘+’의 x, y 좌표를 찾고, 이를 ofDrawLine 함수의 인자로 사용하여 미로를 그린다.

마지막으로 freeMemory() 함수는 자료구조에 동적할당한 메모리를 해제하는 함수이다. 이번 실험에서 세 가지 자료구조 maze, dots, graph를 사용하였으므로 세 개의 자료구조 모두 메모리를 해제한다. maze와 dots는 이차원 배열이기 때문에 maze[i]와 dots[i]의 메모리를 delete[]를 사용해 해제하고 maze와 dots도 delete[]를 사용해 메모리를 해제한다. graph는 vector 배열이므로 graph->clear()를 통해 메모리를 해제한다.

이 프로그램은 2중 for loop을 여러 번 사용하고 있다. 3중 반복문을 사용하기도 하지만, 여기서 사용한 3중 반복문에 경우 마지막 for loop이 4번 반복하는 것으로 고정되어 있으므로 시간 복잡도는 O(n^2)이다. 또한 이 프로그램에서 사용한 자료구조는 2차원 배열이므로 공간 복잡도는 O(n^2)이다.

실험 전 예비보고서에는 graph에 대해 생각하지 못했지만 실험을 진행하면서 인접한 vertex가 저장될 자료구조가 필요하다는 것을 느꼈기 때문에 graph 자료구조를 추가하였고, 데이터를 graph에 저장하는 방법도 앞서 사용했던 이차원 배열이 아닌 vector를 사용한 것이 실험 전과의 차이점이다.

**2.**

이번 주차 실험을 통해 c++과 openframework에 대해 배울 수 있었다. 특히 buffer를 통해 파일의 내용을 입력 받는 방법을 새로 익힐 수 있었고, c++의 동적 할당과 openframework의 ofDrawLine에 대해 복습할 수 있었던 시간이었다. 또한 c++의 벡터에 대해 이해할 수 있었다. c언어에서는 다루지 않는 자료형이기 때문에 처음 이해하고자 할 때 어려움이 있었지만 vector 배열을 정의하고, 동적으로 메모리를 할당하고, push\_back 메소드를 통해 수월하게 인접한 노드와의 연결 관계를 저장하는 방법을 학습하였다.

vector 배열을 동적으로 할당 받기 위해서는 먼저 vector<int>\* graph 형태로 정의해야 한다. 동적으로 할당 받지 않을 때는 vector<int> graph[10]과 같은 형식으로 벡터 배열을 선언할 수 있지만 동적으로 할당 받는 경우에는 graph의 크기가 달라질 수 있으므로 vector<int>\* graph로 정의한다. graph의 크기가 정해지고, 그 크기의 graph의 메모리를 할당 받기 위해서 graph = new vector<int>[size]와 같은 형식으로 작성하면 graph에 대한 동적 할당이 완료된다. graph[u].push\_back(v)와 같이 배열의 u 인덱스에 v를 저장한다. 작업이 종료되고 벡터 배열의 메모리를 해제할 때는 graph->clear()를 사용한다. 만약 벡터 배열이 아니라, 벡터의 메모리를 해제할 때는 graph.clear()를 사용하지만 실험에서는 벡터 배열을 사용했기 때문에 graph->clear()를 사용한다.