12주차 결과보고서

전공: 경제학과 학년: 4학년 학번: 20180501 이름: 김연수

1.

11주차 Eller’s algorithm을 이용해 완전미로를 생성했다. 생성한 완전미로를 openframework를 통해 화면에 그리는 방법을 설명하겠다.

아래와 같이 구조체를 간단하게 정의해준다. 차후 등장할 다른 기능을 위해 구현하는 구조체와는 별개로 maze를 화면에 표시하기 위한 구조체를 선언해주도록 한다. Maze를 화면에 표시하는 기능만을 위해서는 int형 변수 4개를 담은 구조체로 충분하다. 각 변수는 a부터 차례대로 x1,y1,x2,y2를 뜻한다. 선을 긋기 위해 필요한 점 (x1,y1), (x2,y2)이다. Maze 입력파일을 훑으며, ‘-‘ 또는 ‘|’를 마주치는 순간마다 vector에 각 ‘-‘ 또는 ‘|’의 각 끝 지점을 정보로 전달해준다.

텍스트, 폰트, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

ofApp클래스의 public으로 아래와 같이 DRAW\_LINE 데이터타입을 담는 vextor를 생성해주도록한다.



추가적으로 dfs를 구현하기 위한 그래프를 생성하기 위해서, 다음과 같이 구조체를 선언했다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

이와 관련하여 자세한 코드와 구현 설명은 다음 실험에서 하도록 하겠다.

다음으로 ofApp.cpp의 readfile()에서 maze 입력파일을 읽어, vector에 저장하는 코드를 살펴본다.

아래 코드에서 주목해야 할 부분은 if(line[j] == ‘|’) 과 if(line[j] == ‘-‘)이다. 코드의 나머지 부분은 dfs or bfs 수행을 목적으로 graph를 구현하기위해 정보를 temp에 저장해주는 역할을 한다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

DRAW\_LINE m 선언 후, 미로를 탐색하다가 ‘|’ 또는 ‘-‘를 만난다면 그 좌표 값을 m의 field에 저장해주고, maze.push\_back(m)함수를 이용해 vector에 밀어 넣는다. 넣는 순서, 또는 vector에서 빼내 오는 순서는 관계없다. 모든 line이 화면에 표시되는 것이 목적이다.

Draw()함수에서는 이렇게 저장된 maze vector에서 하나씩 원소를 꺼내 가며 화면에 그려준다. 자세한코드는 아래와 같다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

freememory함수는 아래와 같이 ‘q’를 입력하는 경우에 호출하도록 설계했다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

freememory함수의 자세한 코드는 아래와 같다. Maze.clear()를 호출해 위에서 사용한 maze를 free해주도록 한다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

위의 자료구조를 이용해서 maz 파일을 화면에 그리는 작업은 다음의 시공간 복잡도를 가진다.

시간 복잡도 : maz 파일을 탐색해 maze vector에 담는 과정의 시간 복잡도는 WIDTH \* HEIGHT이다. while문은 maz파일의 미로 HEIGHT 만큼 실행되며, while loop 안의 for loop은 maze의 WIDTH만큼 실행된다. 여기서 각각 WIDTH는 ‘2 \* width + 1’을, HEIGHT는 ‘2 \* height + 1’을 의미한다. 그리고 화면에 그릴 때 vector에서 원소를 빼내 오는 작업이 width \* height만큼 수행된다. 원소의 개수가 width \* height만큼 vector에 담기기 때문이다.

이 둘의 시간 복잡도를 종합하여 고려해봤을 때, 위 작업은 o(n^2)의 시간 복잡도를 가진다고 할 수 있다.

공간 복잡도 : maze는 4개의 field로 구성되고, maze로 push\_back되어서 들어오는 원소의 개수는 WIDTH \* HEIGHT 만큼이라고 할 수 있다. 이를 합치면 4 \* WIDTH \* HEIGHT 가 되는데, 따라서 공간 복잡도는 O(n^2)로 설명할 수 있다.

실험 전 고안했던 것은 그래프와 화면에 그리는 좌표의 정보를 모두 하나의 구조체에 담아 처리하는 방식이었다. 하지만, 직접 구현하는 과정에서 후에 dfs와 bfs의 구현까지 고려했을 때, 이와 같은 방식으로는 다소 어려움이 있을 것으로 판단했다.

2,

본 실험에서 한 번도 여태 사용해보지 않았던 vector를 사용했다. Vector의 사용법은 다음과 같다. 선언할 때 “vector<int> maze;” 와 같은 방식으로 선언할 수 있다. <>안에는 원소의 데이터 타입을 지정해준다. 원소를 집어넣는 방법은 push\_back() 함수를 호출하면 된다. 원소를 빼낼 때는 인덱싱을 이용해 빼낼 수 있다. 본 실험에서는 draw함수에서 vector에서 원소를 빼낼 때, for(const auto& m : maze)와 같은 방식을 사용했다. 이 방식도 처음 접해본 방식이었다. 편리하게 원소를 하나씩 꺼낼 수 있었다.

자료구조를 직접 구상해서 문제를 해결하는 방식에도 익숙해질 수 있었다. 단순하게 int배열을 4개 선언해서 문제를 해결하는 것이 아니라, 구조체 내에 int field를 4개 선언해서 사용하면 코드도 간결해지고, 가독성도 향상시킬 수 있었다. 후에 복잡한 문제 해결을 위해서 string와 int 결합해 사용할 수도 있고, struct내에 다른 struct를 선언한다든가, struct내에 union field를 선언할 수도 있을 것이다. 그리고 그 구조체를 이용해서 linked list를 구현할 것인지, 2차원 배열을 이용할 것인지와 같이 자료구조를 구상하는 연습을 통해서 시공간 복잡도를 향상시킬 수 있는 방식을 고민하는 습관이 길러졌다.