13주차 결과 보고서

전공: 아트앤테크놀로지학과 학년: 3학년 학번: 20201116 이름: 이수빈

* + **DFS**

이는 예비 보고서에는 미로의 각 방을 정점으로 하고 두 방을 있는 벽을 간선으로 하는 그래프를 생성하고 인접 리스트로 이 간선을 저장하는 방식으로 구현하였다. 그러나 실습을 하며 미로의 특성상, DFS를 구현하기 더 간단한 방법을 고안하여 아래와 같이 구현하였다.

Qr code

Description automatically generated

우선, 미로의 모양을 살피면서 만약 미로가 빈칸이라면, 즉 비어 있다면, maze\_check를 0으로 설정해주고 나머지의 경우에는 모두 1로 설정해준다.



그리고 DFS를 구현하기 위해 stack을 정의한다. stack은 First in First Out의 형태를 갖는 자료 구조이다.

DFS 함수를 호출하면 우선 시작 노드 (maze는 시작하는 가장 첫번째 셀)을 stack에 넣어준다. 그 후 stack이 빌 때 까지 아래의 과정을 반복한다.

우선 stack의 가장 위에 있는 원소를 pop을 해를 받고 그 원소에 인접한, 즉 미로에서는 길이 이어져 있는 원소들을 stack에 넣는다. 이는 해당 노드의 상하좌우를 순서대로 살피면 된다. 이때, maze\_check이 1이 아닌 원소들만 stack에 넣어준다. 그 후 stack에 넣은 원소를 maze\_check를 1로 바꿔준다. 이는 이 셀은 이미 방문했다는 의미로 나중에 다시 stack에 넣지 않게 한다. 이때, 위에서 pop한 원소 중 적어도 하나의 원소를 stack에 집어 넣었으면 check를 1로 설정해준다. 이 check는 while문을 반복할 때마다 0으로 초기화를 해주는데 이 값은 탐색한 길 말고 목표 지점까지의 길을 추적하는데 사용된다.

Text

Description automatically generated

위에서 얻은 path 정보를 바탕으로 draw함수에서 path를 그려주면 본 실험에서 원하는 화면을 얻을 수 있다.

DFS는 worst case에서는 모든 maze의 cell에 대해서 수행해야한다. 따라서 복잡도를 다음과 같이 표현할 수 있다.

시간 복잡도 : O(HEIGHT \* WIDTH)

공간 복잡도 : O(HEIGHT \* WIDTH)

* + **BFS**

BFS는 DFS와 다르게 stack 대신 queue를 사용한다. 그렇기 때문에 마지막에 넣은 원소부터 탐색하는 DFS와는 다르게 가장 먼저 넣은 원소 먼저 탐색한다. 그러나 이를 구현하기 위해서는 더 많은 배열을 정의해 줘야한다.

시간 복잡도 : O(HEIGHT \* WIDTH)

공간 복잡도 : O(HEIGHT \* WIDTH)

* + **장단점**

DFS는 깊이를 우선으로 탐색하기 때문에, 목표 지점이 depth가 큰 곳에 있을 경우 비교적 빠르다. 이에 반하 BFS는 넓이를 우선으로 탐색하는 방법으로 찾고자 하는 목표 지점이 depth가 얕은 곳에 있을 때 유리하다. 즉, 미로에 대입해 생각해 보면, DFS는 높이가 넓이보다 더 큰 형태일때, BFS는 넓이가 높이보다 더 큰 경우 유리하다.

본 프로젝트에서는 DFS가 BFS보다 더 좋은 형태를 지닌다. 우선 구현하기 더 간단하고 공간 복잡도도 더 작기 때문이다.