**학과: 컴퓨터공학과 학년: 2 학번: 20191657 이름:최세은**

1. **실습 및 숙제로 작성한 프로그램의 알고리즘과 자료구조를 요약하여 기술한다. 완성한 알고리즘의 시간 및 공간 복잡도를 보이고 실험 전에 생각한 방법과 어떻게 다른지 아울러 기술한다.**

이번 실습에서는 미로에서 길을 찾기 위해 알고리즘으로 DFS를 사용하였고 이를 구현하기 위해 stack 자료구조를 사용하였다.

DFS는 깊이 우선 탐색으로, 더 이상 이동할 수 없을 때까지 이동하면서 도착점을 찾아간다. 이를 구현하기 위해 우선 해당 지점(노드)를 방문했는지 표시하는 visited 배열을 선언해 0으로 초기화 시켰다. 그리고 시작점의 좌표를 가지는 노드를 stack에 삽입하고 반복문을 수행하는데, 이 반복문은 stack에 노드가 하나도 없을 때까지 반복된다. vec 배열을 이용해 현재 좌표에서 상하좌우를 살펴, visited 배열을 이용해 한 번도 방문한 적이 없으며, input 배열에서 빈 칸(‘ ‘)인, 즉 이동할 수 있는 지점을 찾아 해당 지점으로 이동하며 이동할 때는 stack에 해당 좌표를 담은 노드를 삽입한다. 그리고 더 이상 이동할 수 없게 되었다면 stack의 노드를 pop 하면서 마지막으로 이동 할 수 있는 좌표로 돌아온 다음에 다른 곳으로 이동을 시작한다. 이를 반복문을 통해 계속 반복하다가 도착점에 도달하면 return 1로 도착했음을 표시하고 함수를 종료하고, stack에 노드가 없어 끝나면 도착점을 못 찾았으므로 return 0을 해준다.

그리고 dfsdraw 함수로 깊이 우선 탐색을 하여 도착점에 도달 할 때까지 이동한 경로와, 해당 경로 외에도 이동했었던 다른 경로들을 화면에 그려준다. 우선 DFS 함수를 실행하고 난 후, stack에 출발부터 도착할 때까지의 경로가 순서대로 저장되므로 이를 이용해 화면에 선으로 경로를 그려준다. 그리고 다른 색으로 시도했지만(이동했지만) 막혔던 경로도 화면에 그려주는데, 이는 dfs 함수에서 pop한 노드들의 좌표들 이므로 pop할 때 따로 다른 stack에 저장해준 후 마찬가지로 화면에 선으로 경로를 그려준다.

우선 DFS 함수의 시간 복잡도는 최악의 경우 모든 곳을 탐색하고 도착하므로 O(n) (n은 간선과 방의 개수의 합)이며 공간 복잡도는 2차원 배열을 사용했으므로 O(n\*m)이다. 그리고 dfsdraw 함수의 시간 복잡도는 stack에 저장된 노드들을 하나씩 건너 뛰면서 모두 검사하므로 O(n) (n은 노드 개수), 공간 복잡도는 linkedlist를 사용하므로 O(n) 이다.

실험 전에 생각했던 내용처럼, dfs 라는 알고리즘을 사용해서 구현했으므로 생각과 거의 똑같이 구현했다. 다만 dfs 함수에서 pop할 때, 실패했던 경로들도 모두 화면에 그려야 했기에 pop한 노드들을 따로 저장해준 건 실험 전에는 미처 생각하지 못했던 부분이었다.

1. **자신이 설계한 프로그램을 실행하여 보고 DFS, BFS 알고리즘을 서로 비교한다. 각각의 알고리즘은 어떤 장단점을 가지고 있는지, 자신의 자료구조에는 어떤 알고리즘이 더 적합한지 등에 대해 관찰하고 설명한다**

DFS는 무한히 넓은 미로에서 탐색할 때 효과적이다. 또한 현재 경로상의 노드들만 stack에 저장하므로 그래프의 높이 만큼의 공간만을 요구한다. 그러나 막힐 때까지 계속 이동하므로, 엉뚱한 곳에서 오래 머물 수 있다. BFS는 출발한 후 도착할 때까지의 최단 길이의 경로를 찾아준다. 그러나 최악의 경우 모든 노드들에 대한 정보를 queue에 저장해야 할 수 있다.

내가 사용한 자료구조는 linkedlist와 2차원 배열이고, dfs 알고리즘이 더 적합하다고 생각한다. 이유는 현재 생성되는 미로는 완전 미로로, 출발에서 도착할 때까지의 경로가 단 한 개이기 때문에 무조건 최단 경로가 된다. 그리고 bfs는 현재 노드의 좌표의 바로 이전 이동 좌표에 대한 정보도 필요해서 dfs 알고리즘이 더 적합하다고 생각한다.