**Maze-2 예비보고서**

전공: 수학 학년: 4 학번: 20161255 이름: 장원태

1. **DFS와 BFS 알고리즘에 대해 조사하고 간략히 요약한다.**

DFS 알고리즘은 깊이 우선 탐색(Depth-First Search)으로, 루트 노드(node)나 다른 임의의 노드에서 시작해서 다음 분기(branch)로 넘어가기 전에 해당 분기를 완벽하게 탐색하는 방법이다. 미로의 길 찾기를 예로 들면, 한 방향으로 갈 수 있을 때까지 계속 가다가, 더 이상 갈 수 없게 될 경우 가장 가까운 갈림길로 돌아와서, 그 곳에서 다른 방향으로 같은 탐색을 반복하는 방법이다. 이를 그래프로 보면, 특정 꼭지점(vertex)에서 시작하여, 인접한 노드 하나를 선택하여 그 노드와 연결된 꼭지점으로 이동한다. 그리고, 같은 방식으로 인접한 노드 하나를 선택하여 또 그 노드와 연결된 꼭지점으로 이동한다. 이 과정을 이동할 꼭지점이 더 없는 경우까지 반복하다가, 이동할 꼭지점이 더 없는 경우 이전 꼭지점으로 돌아와 같은 과정을 반복하는 알고리즘이다.

BFS 알고리즘은 너비 우선 탐색(Breadth-First Search)으로, 루트 노드나 다른 임의의 노드에서 시작해서 인접한 노드를 먼저 탐색하는 방법이다. 즉, 시작점에서 가까운 정점을 먼저 방문하고, 멀리 떨어져 있는 정점은 나중에 방문하는 방법이다. 이 방법은 두 노드 사이의 최단 경로나 임의의 경로를 찾고 싶을 때 주로 이용하는 방법이다. 이 알고리즘은 DFS 알고리즘과 다르게 재귀적으로 동작하지 않는다. 그리고, 방문한 노드들을 차례대로 저장한 뒤 그 순서대로 꺼내올 수 있는, 즉 FIFO(First-in-First-Out) 원칙을 가지는 큐(queue)를 자료구조로 채택한다.

1. **미로문제에서 DFS, BFS를 수행하기 위한 자료구조를 설계하고 이의 공간복잡도를 보인다.**

미로문제에서는 DFS를 수행하든 BFS를 수행하든 상관없이 우선 주어진 미로를 그래프의 형태로 구현해야 한다. 1주차에서 수행한 실습에서 언급한 cell 배열의 각 cell들을 구조체로 선언하여, 그 구조체에서 행과 열 정보, 즉 위치정보를 저장한다. 또한, 이 구조체는 인접한 방과의 연결 관계를 표시해야 한다. 상하좌우로 방들이 연결되어 있는지를 확인해야 하는데, 상하 중 하나, 좌우 중 하나만 확인할 수 있어도 모든 방향으로 연결 여부를 확인할 수 있다. 예를 들어 좌우 중 “우”, 즉 오른쪽 방과의 연결 관계를 알 수 있다면, 왼쪽 방과의 연결 관계는 해당 왼쪽 방에서 오른쪽 방과의 연결 관계를 확인하면 알 수 있다. 상하 관계도 이와 같은 맥락으로 확인할 수 있다. 그러므로, 메모리 절약을 위해 좌우 중 우(right), 상하 중 하(down) 관계를 저장할 수 있도록 한다. 즉, 다음과 같은 구조체이다.

Struct node{

Int x;

Int y;

Int right;

Int down;

};

또한, DFS와 BFS 알고리즘에서 중요한 것은 각 방을 방문했는지 하지 않았는지 여부이다. 따라서, visited라는 배열을 미로 cell 크기만큼 설정하여, 방문한 경우 1, 방문하지 않은 경우 0으로 표시할 수 있도록 한다.

이와 같은 자료구조를 설정하면, 미로의 너비(width)와 높이(height)에 따라, node는 int형 데이터를 4개 포함하고 있으며 총 width\*height개가 필요하고, visited 배열 역시 이차원 배열로서 데이터가 총 width\*height개가 필요하므로, 공간 복잡도는 O(width\*height)이 된다.

1. **설계한 자료구조에서 DFS, BFS를 어떻게 수행할지 간략히 보인다.**

각 배열이 다음과 같이 정의되어 있다고 하자.

Struct node maze[height][width];

Int visited[height][width];

그리고 maze에 현재 미로의 정보를 저장하고, visited는 모두 0으로 초기화해준다.

DFS 함수의 경우, 시작 칸의 visited를 1로 set한다. 그 후, 그 칸에서 상하좌우로 이동할 수 있는지를 체크한다. 확인 순서는 (우-하-좌-상) 순서라고 가정하자. “우”와 “하”는 단순히 해당 칸에서 right과 down을 확인하면 된다. “좌”와 “상”같은 경우, 해당하는 다음 방에서 역으로, “좌”에 있던 방이면 그 방의 right 값을, “상”에 있던 방이면 그 방의 down 값을 확인하여 이동할 수 있는지를 판별한다. 각 방향대로 이동할 수 있고 해당하는 다음 방이 visited되어 있지 않다면, 현재 방을 stack에 push하고, 다음 방으로 이동한다. 이동한 후의 방의 visited는 1로 set해주도록 한다. 이와 같은 과정을 반복하다가 더 이상 아무 방향으로도 이동할 수 없다면, stack에서 pop 수행을 통해 이전 방으로 돌아가서 다른 방향을 탐색하며 같은 과정을 반복한다.

BFS 함수의 경우, 시작 칸의 visited를 1로 set한다. 그리고, 시작할 때 queue에 push를 해주고 시작한다. 그 후, queue에서 pop 수행을 통해 방의 정보를 얻어내며(시작할 때는 시작 방을 얻는다), 그 방에서 상하좌우로 이동 가능한지를 체크한다. DFS 함수와 마찬가지로 (우-하-좌-상) 순서로 정하도록 하자. 그 후, 이동할 수 있는 모든 방 정보를 queue에 push하여 저장한다. 그 후, queue에서 pop 수행을 통해 얻어낸 방으로 이동하여 그 칸의 visited를 1로 set하고, 같은 과정을 반복한다.