컴퓨터공학실험1 13주차 예비보고서

전공: 컴퓨터공학과 학년: 2 학번: 20191559 이름: 강상원

1. **DFS와 BFS의 시간 복잡도를 계산하고 그 과정을 설명한다.**

   DFS, BFS 모두 모든 노드를 한 번씩만 방문하는 알고리즘의 종류이다. 그래프를 G=(V, E)로 표현한다면 (V: 노드, E: 간선) 모든 노드를 방문하는 데는 O(V)가 필요하다. 각 노드에 대해 그에 인접한 노드를 탐색해 진행하는데, 사용하는 자료 구조에 따라 시간 복잡도가 상이하다. 만약 인접 리스트를 사용할 경우 인접 노드를 정리하는 시간 복잡도가 O(||E||)인 반면에 인접 행렬을 사용할 경우 O(||V||^2)이다.

   ∴ **전체 시간 복잡도**: 인접 행렬은 O(||V||^2), 인접 리스트는 O(||V||+||E||)이다.

   ＊ ||V||+||E||는 WIDTH*HEIGHT에 비례한다.

2. **자신이 구현한 자료구조 상에서 DFS와 BFS 방법으로 실제 경로를 어떻게 찾는지 설명한다. 특히 DFS 알고리즘을 iterative한 방법으로 구현하기 위한 방법을 생각해보고 제시한다.**

   12주차 (Maze 2주차) 예비보고서에 작성한 자료구조를 사용한다.

   현재 칸에서 움직일 수 있는 방향을 생각한다면 위, 아래, 좌, 우 총 4가지이다. 현재 칸이 point[i][j]라 한다면, point[i-1][j], point[i+1][j], point[i][j-1], point[i][j+1]이다. (주어진 미로에서 대각선으로 움직일 수는 없다.) 따라서 이차원 배열을 만들어서, 이미 방문했는지 여부를 저장하면 된다. 이때, 단순 이차원 배열로 저장하면 상, 하, 좌, 우로의 이동 가능성을 저장하는데 문제가 생긴다. 따라서

```
struct block {
    int up;
    int down;
    int left;
    int right;
};
```

   와 같이 구조체를 만들고, 이 구조체를 담는 2차원 배열을 선언하면 된다.

   (ex: 이동 가능하면 1, 이동 불가 0 저장)

   bool visited[HEIGHT][WIDTH] 이차원 배열을 새로 만들어서 방문 여부를 저장한다. 구조체에 저장된 (이동 가능 방향 종류) – (이미 방문한 칸)을 스택에 집어넣고 탐색한다. DFS, BFS의 Pseudo Code는 하단에 기술되어 있다.

   - DFS

     스택을 정의한 다음, 첫 노드를 넣고 그 노드의 인접 노드 중 아직 방문하지 않은 노드들을 스택에 Push한다. 이후 스택의 첫 원소를 제거하며, 스택의 원소의 개수가 0이 될 때까지 반복한다.

- DFS 알고리즘을 iterative한 방법으로 구현

```
void DFS(v) {          // v는 시작 노드
    initialize a stack Stack
    Stack.push(v)
    v is now visited
    while(Stack != empty) {
            if(Stack.top==target) return;
            if(Stack.top has remaining unvisited near node) {
                    u=an unvisited , near node to Stack.top
                    Stack.push(u)
                    u is now visited
            }
            else Stack.pop()
    }
    return
}
```

- BFS

  큐를 정의한 다음, 큐에 첫 원소를 넣고 그 노드의 인접 노드 중 아직 방문하지 않은 노드를 전부 큐에 추가한다. 이후 큐의 첫 원소를 제거하고, 큐의 원소의 개수가 0이 될 때까지 반복한다.

  시작 노드에서 시작해 연결된 노드로 모두 뻗어나가보아 탐색한다. 큐에서 하나씩 노드를 빼서 연결된 노드로 뻗어나가보는 과정을 반복한다. 이 때, 이미 방문한 노드는 visited[]에 정보를 저장해서 중복 방문을 막는다.