1. DFS와 BFS의 시간 복잡도를 계산하고 그 과정을 설명한다.

DFS, BFS는 그래프의 각 vertex를 한 번씩만 방문한다. 따라서 그래프의 vertex 개수를 V, edge 개수를 E라고 하면 각 vertex의 방문에 필요한 시간복잡도는 O(|V|)이다. 한편, 각 vertex를 방문하면서 해당 vertex와 인접한 vertex를 저장하고 있는 자료구조가 필요하다. 대표적으로 인접 행렬과 인접 리스트가 있다. 만약 인접 행렬을 사용한다면 인접 vertex들을 탐색하는 데 필요한 최악 시간복잡도는 $O(|V|^2)$이며, 만약 인접 리스트를 사용한다면 O(|E|)이다.

따라서 DFS와 BFS의 전체 시간 복잡도는 인접 행렬을 사용할 경우 $O(|V|^2)$이며, 인접 리스트를 사용할 경우 O(|V|+|E|)이다.

미로 프로젝트에서 |V|+|E|는 O(WIDTH * HEIGHT)이다.

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

미로에서 각 칸을 vertex라고 한다면 각 vertex에서 인접한 vertex는 상, 하, 좌, 우에 있다. 물론 벽의 유무에 따라 이동이 불가능하여 인접하지 않은 경우도 있다. 따라서 Maze 2주차에서 저장한 maze 정보에 따라 각 vertex의 이동 가능 여부를 저장하는 자료구조가 필요하다. 상, 하, 좌, 우로 이동이 가능 여부를 수월하게 저장하기 위해 구조체를 사용해 각 vertex가 가질 수 있도록 WIDTH * HEIGHT개 사용하면 될 것이다. 또한 DFS, BFS에서는 방문한 여부를 저장할 자료구조도 필요하다. 이 역시 WIDTH * HEIGHT 만큼의 배열을 사용해 방문 여부를 저장할 수 있다.

- DFS: stack을 사용하는 알고리즘이다.

```
void DFS(starting vertex)
	Initialize stack.
	Initialize all vertexes to non-visited vertex.
	Mark starting vertex visited.
	Push starting vertex.
	while (stack is not empty)
		current vertex := pop from stack
		if current vertex is the destination vertex
			then return
		for adjacent vertexes
			if that vertex is not visited
				then push the vertex, mark the vertex visited.
	return
```

- BFS: queue를 사용하는 알고리즘이다.

```
void BFS(starting vertex)
	Initialize queue.
	Initialize all vertexes to non-visited vertex.
	Mark starting vertex visited.
	Push starting vertex.
	while (queue is not empty)
		current vertex := pop from queue
		if current vertex is the destination vertex
			then return
		for adjacent vertexes
			if that vertex is not visited
				the push the vertex, mark the vertex visited.
	return
```