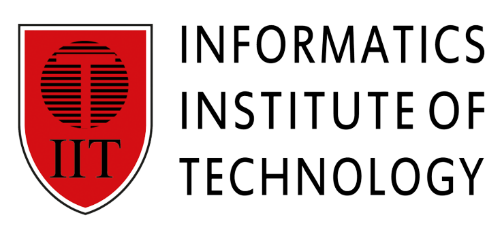
**A close-up of a logo

Description automatically generated**

**Informatics Institute of Technology**

**Algorithms: Theory, Design and Implementation**

**5SENG003C.2**

**Individual Coursework**

**Module Leader: Mr. Sivaraman Ragu**

**Name : M.G. Dilhara Manthilina**

**UOW ID : w1999520**

**IIT ID : 20221643**

# Table of Content

**Contents**

[Table of Content 2](#_Toc164876242)

[1. Data Structure and algorithm 2](#_Toc164876243)

[1.1. Data Structure 2](#_Toc164876244)

[1.2. Algorithm 2](#_Toc164876245)

[2. Analysis 3](#_Toc164876246)

[3. Benchmark example 4](#_Toc164876247)

# Data Structure and algorithm

## Data Structure

To solve the sliding puzzle data structures such as graphs and queues were used. A 2D char array serves as the graph representation. Different symbols each denote different functions such as “@” representing the current player position, ”S” representing starting position, ”F” representing finishing position, “.” Representing floor(ice) and “0” representing obstacles.

## Algorithm

Breadth First Search is implemented using a queue to check cell levels which manages the pattern of the moving starting from start to finish. The procedure frequently takes a cell out of the queue's front, marks it as visited, and then re-enqueues all of the cells that are nearby that have not yet been visited. To begin the process, the start point is put in line. The process continues until the destination is reached or the queue is empty, indicating that there is no path available. Breadth First Search looks into cells based on how far they are from the starting point, making sure that the quickest route will be used in order to get to the final point first.   
A maze's shortest path is found by the computer as quickly as possible thanks to the queue data structure and the BFS algorithm. Through a breadthwise examination of all potential paths, BFS ensures that the shortest option is found first and prevents examining longer paths needlessly.

# Analysis

The performance study of the given code incorporates practical insights and theoretical problems through empirical research. The main focus of this study is on the time complexity, scalability, and application of the Breadth-First Search (BFS) algorithm for solving the sliding puzzle problem.

Time complexity : The number of rows (n) and columns (m) in the map determine the time complexity for parsing the map (`parseMapFromFile(filePath)}) and extracting map information (`parseMapInfo(map)}). This complexity results from reading and processing each map cell iteratively. The integers V and E represent the graph's vertex count (V) and edge count (E), respectively. The time complexity of the BFS traversal {findShortestPath(map, start, finish)} is O(V + E). In this case, every cell represents a vertex, and BFS uses a queue-based approach to look into nearby cells, or edges. By examining nodes level by level, BFS is able to efficiently discover the shortest path because of the grid's consistent move costs and absence of weight.

# Benchmark example

