INFORMATICS INSTITUTE OF TECHNOLOGY

Bsc(hons) Computer Science

Algorithms: Theory, Design and Implementation - 5SENG003C.2

<u>Cw Report - 2024</u>

Name :	Samadhi Ediriwickrama
UoW Number :	w1985625
IIT Number :	20221862
Level :	L5

W1985625_Choice of Data Structure:-

The data structures used in this implementation have been chosen for their scalability and adaptability to express and interact with the grid-based environment. The ice field grid is represented by a 2D grid of Nodes, which is a powerful and efficient way to model and navigate the ice field. This makes it suitable for a wide range of applications that require graph representations and algorithms.

Graph Representation using a 2D Array of nodes: The ice field is a grid-based environment that is represented by a 2D grid. Each node in the grid stores information about the position of the cell (rows and columns) and the terrain type it represents.

IceField class represents the ice field in a 2D grid-like structure. Each node contains information about the position (rows and columns) of the node in the grid and the terrain it represents. This grid-like structure makes it easy to access individual cells in the ice field.

The advantages of this representation are: 2D array structure makes it easy to store and retrieve cell data, Node objects represent cells and encapsulate the data about each cell. In addition, Graph traversal algorithms are easy to implement as the ice field is represented as a network of nodes.

Suitability for graph environments: This representation is suitable for graph environments such as the ice field. It allows you to model terrain and obstacles in a clear and easy-to-understand way. This representation allows me to navigate and explore the grid. This is important for tasks like finding paths from one point to another.

W1985625 Choice of Algorithm:-

Out of all the algorithms we use to solve the problems described in our coursework I choose the Breadth-First Search (BFS) algorithm as the algorithm type for solving the problem. It used to find the shortest way from the starting point to the ending point in an ice field grid. BFS is used because it searches all possible ways from the starting point outward in breadth-first, making sure that the shortest way is found when I get to the end point.

Let's look at the ice field problem from a graph: Each cell is a node, Each adjacent cell is connected by edges, BFS systematically searches all possible ways by visiting adjacent cells in breadth-first so that the first time you reach the end point, it will be through the shortest way.

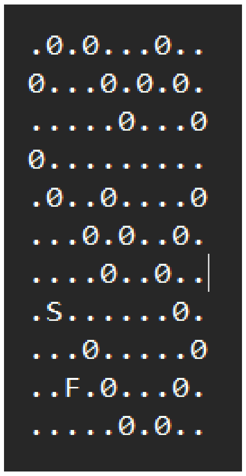
Benefits of using BFS for pathfinding:

- BFS ensures that the shortest path is found from the beginning of the path to the end of the path.
- BFS examines paths systematically and efficiently, making sure that all paths are taken into account.
- BFS works well in unweighted graphs where all edges have equal weight, such as in an ice field.
- BFS can be easily implemented and understood, making it ideal for solving pathfinding issues in grid environments like an ice field.

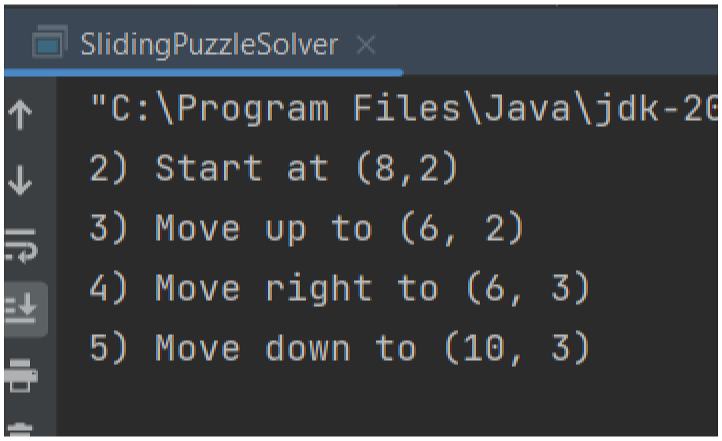
Graph representation and BFS algorithm look like this,

The ice field is graphically represented using a 2D field of Nodes. Each cell in the graph represents a node. Each Node object contains details about its position (rows and columns) and the terrain type it represents. By combining graph representation with the BFS algorithm, we can find the shortest path from the beginning to the end of the ice field grid.

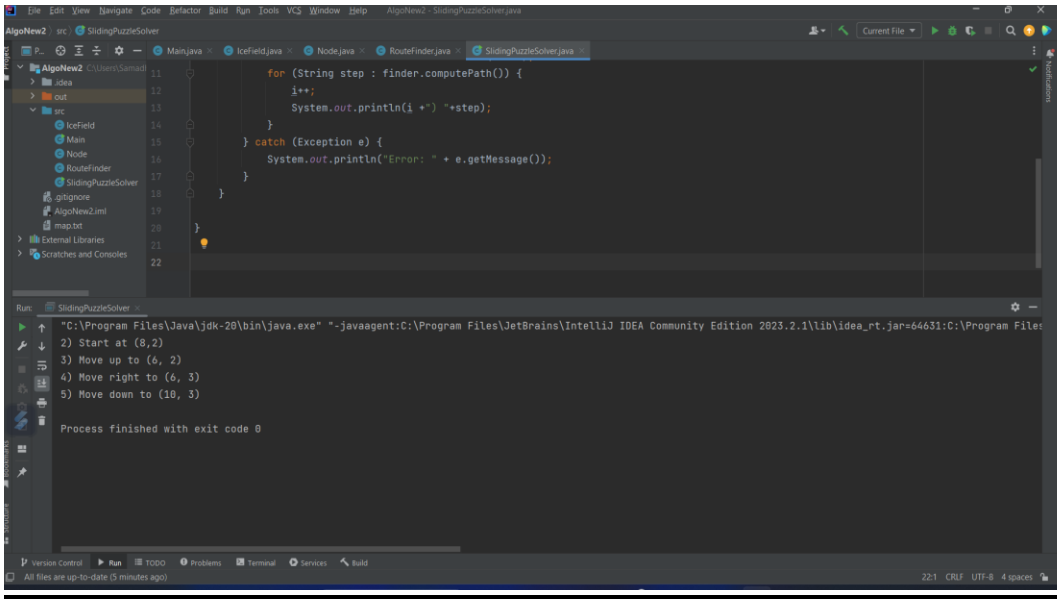
W1985625 Benchmark Example:-



```
.0.0...0..
0...0.0.0.
.....0...0
0.........
.0..0....0
...0.0..0.
....0..0..
.S......0.
...0.....0
..F.0...0.
.....0.0..
```



```
"C:\Program Files\Java\jdk-2
2) Start at (8,2)
3) Move up to (6, 2)
4) Move right to (6, 3)
5) Move down to (10, 3)
```



```
            for (String step : finder.computePath()) {
                i++;
                System.out.println(i +") "+step);
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

}
```

```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.1\lib\idea_rt.jar=64631:C:\Program Files
2) Start at (8,2)
3) Move up to (6, 2)
4) Move right to (6, 3)
5) Move down to (10, 3)

Process finished with exit code 0
```

W1985625 A performance analysis

In worst-case BFS, each node is traversed once and every edge is traversed once. BFS time complexity is O = V + E, where V is number of nodes in the grid and E is number of edges in the grid. In this example, V represents number of cells and E represents number of edges of adjacent cells. Each cell can have 4 adjacent edges (Up, Down, Left, Right) so the number of edge is proportional to number of cells.

Grid traversal: In BFS, each cell can be visited once in the grid. As the algorithm glides through the grid it checks the validity of each cell and updates the route accordingly. This grid traversal also has a time complexity of O = cols.

Path reconstruction: Once the path is found, the algorithm traces the path back from the start node to the end node. In this step, the pathMap has a minimum of O * cols entries.

suggested order-of-growth classification (Big-O notation): O(rows*cols) is the proposed classification of the algorithm according to Big-O notation. The "rows" are the rows of the grid and the "cols" are the columns of the grid. The Big-O notation shows that the time complexity of the algorithm increases linearly as the number of cell in the grid increases. As the size of the grid increases, the algorithm's time complexity increases proportionally.

****