INFORMATICS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF WESTMINSTER

Informatics Institute of Technology

5SENG003C.2 Algorithms

Name : Nammuni Vathila Vilhan De Silva

IIT No : 20200765

UoW No : w1833511

TASK 5

A) The algorithm which is being used in this implementation is the breadth first search. Breath first search is a graph traversal method that starts at a certain node (source or starting node) and traverses the graph layer by layer, visiting the adjacent nodes (nodes which are directly connected to source node) then it must go to the next-level neighbor nodes. The Breadth First Search technique was chosen for several reasons, including its utility in inspecting network nodes and calculating the shortest path from "S" to "F" by traversing the puzzle. A graph might include cycles, which can lead you back to the same node when traversing it. Use of an if condition if (nextPosition != null) to mark the node and to check if the next node is empty or not after it is processed to avoid processing it again. While visiting the nodes in a graph's layer.

```
for (Direction dir : Direction.values()) {     // traversing adjacent nodes while sliding in the direction
    Point nextPosition = move(textDataArray, slideTraverse, currPosition, dir);
    System.out.println("\t" + nextPosition);
    if (nextPosition != null) {
        listQueue.addLast(nextPosition);
        slideTraverse[nextPosition.getY()][nextPosition.getX()] = new Point(currPosition.getX(), currPosition.getY());
        if (nextPosition.getY() == YEndValues && nextPosition.getX() == XEndValues) { //finding the end point

            Point tempValue = nextPosition;
            int count = 0;
            while (tempValue != startPosition) {
                count++;
                tempValue = slideTraverse[tempValue.getY()][tempValue.getX()];
                MazePath.addFirst(tempValue);
            }
            return count;
        }
    }
}
```

In this piece of code, it traverses through the nodes using an if condition which checks if the next position is null or not if it isn't null it travels through the node until a wall is met in the sliding puzzle.

```
public static class Point {
    int x;
    int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() { return x; }

    public int getY() { return y; }
```

A pointer was used to return the co-ordinates of the node.

The data structure which was used to implement the breadth first search algorithm was a queue data structure to find the shortest path, which follows the first in, first out principle. In BFS, one vertex is visited and marked at a time, then its neighbors are visited and placed in the queue.

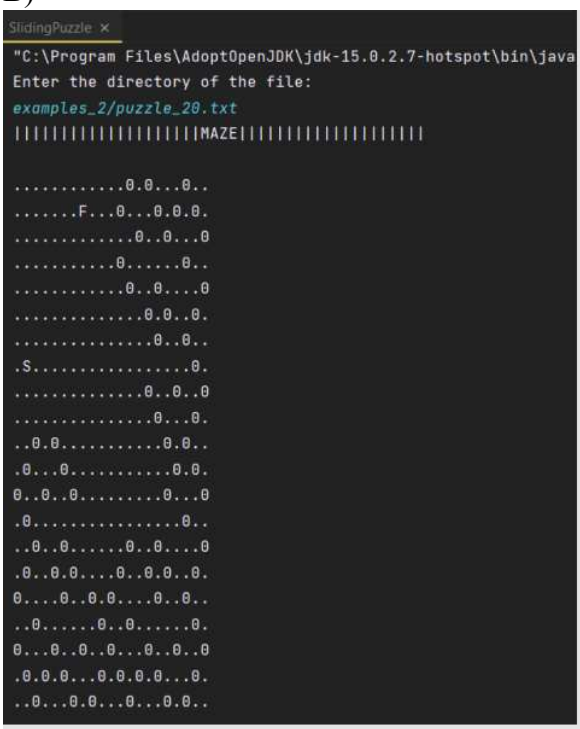
```
Point startPosition = new Point(XStartValues, YStartValues); //starting position to f
LinkedList<Point> listQueue = new LinkedList<>();
Point[][] slideTraverse = new Point[textDataArray.length][textDataArray[0].length]; /
```

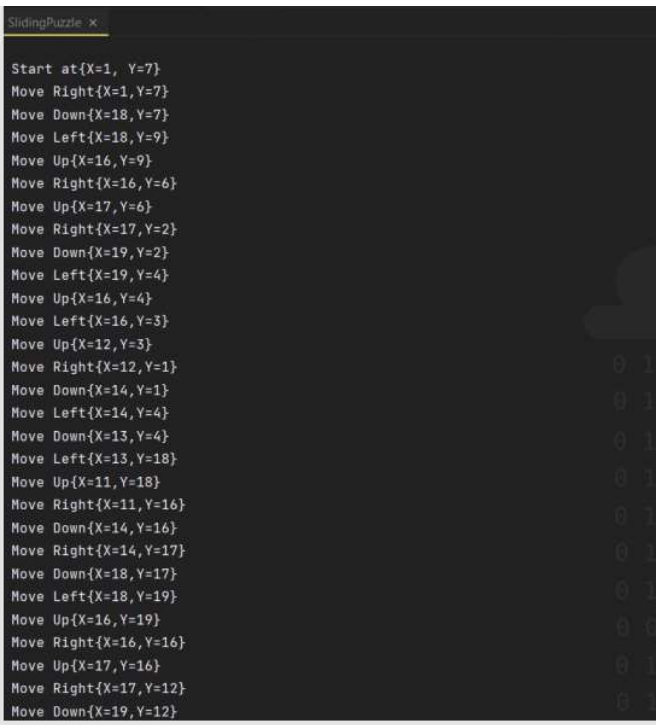
Another LinkedList is used to store the path co-ordinates of the puzzle,

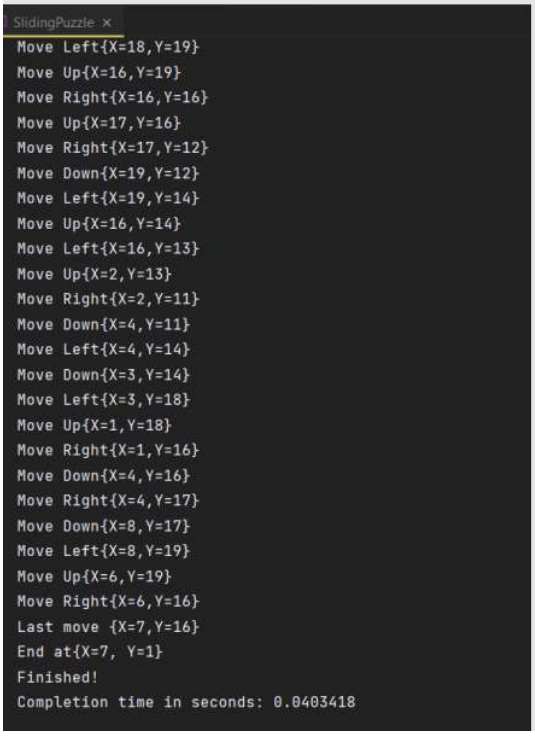
```
static String directionalMovement="";
static LinkedList<Point> MazePath =new LinkedList<>(); //used to store the path(co-ordinates)
static String navigator;
static ArrayList<String> data = new ArrayList<>(); //arraylist created to store the data from text file
static  int countRow = 0; //number of rows in the puzzle
static long timeStart;
```

Advantages of using breadth first search is that BFS will never become lost in a blind alley, resulting in undesired nodes also how breadth first searches optimize is if there are many solutions, it will choose the one that requires the fewest steps.

B)

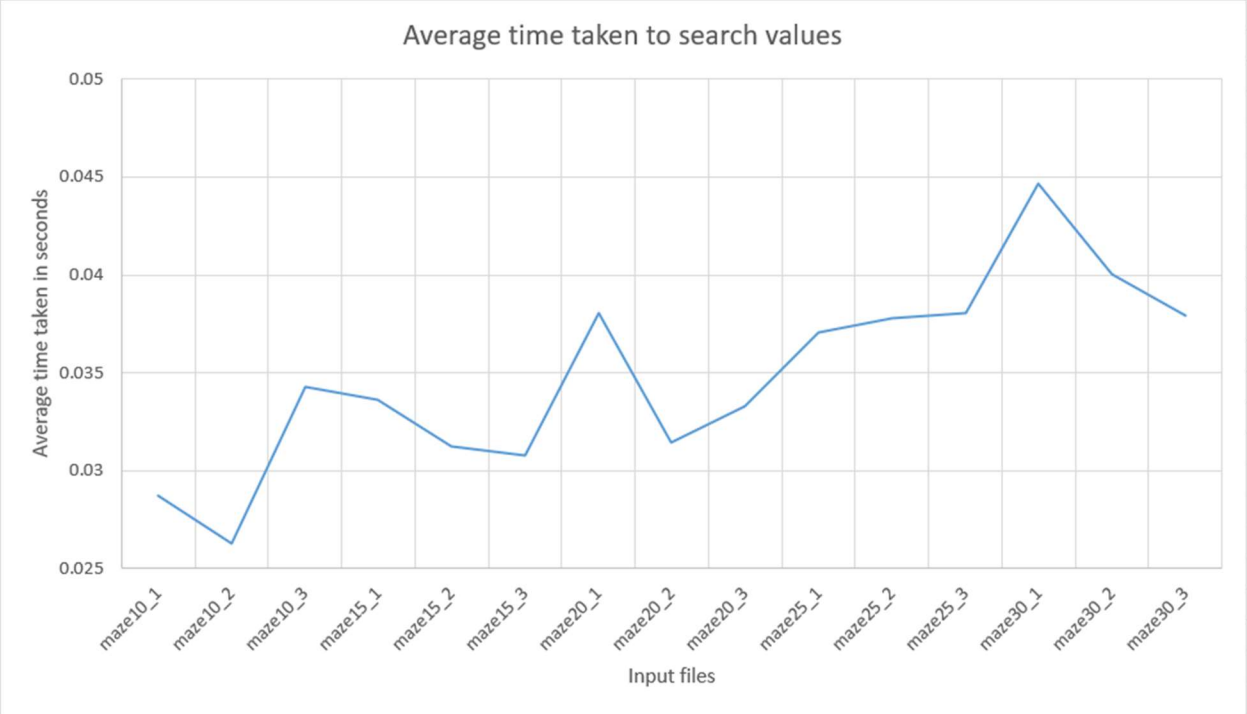

SlidingPuzzle
"C:\Program Files\AdoptOpenJDK\jdk-15.0.2.7-hotspot\bin\java
Enter the directory of the file:
examples_2/puzzle_20.txt
||||||||||||||||||||MAZE||||||||||||||||||||
...........0.0...0..
.......F...0...0.0.0.
............0..0...0
..........0.....0..
............0..0....0
............0.0..0.
............0..0..
.S..............0.
............0..0..0
............0...0.
..0.0..........0.0..
.0...0..........0.0.
0..0..0........0...0
.0..............0..
..0..0.....0..0....0
.0..0.0....0..0.0..0.
0....0..0.0....0..0..
..0.....0..0.....0.
0...0..0..0...0..0..0
.0.0.0...0.0.0.0...0.
..0...0.0...0...0.0..


SlidingPuzzle
Start at{X=1, Y=7}
Move Right{X=1,Y=7}
Move Down{X=18,Y=7}
Move Left{X=18,Y=9}
Move Up{X=16,Y=9}
Move Right{X=16,Y=6}
Move Up{X=17,Y=6}
Move Right{X=17,Y=2}
Move Down{X=19,Y=2}
Move Left{X=19,Y=4}
Move Up{X=16,Y=4}
Move Left{X=16,Y=3}
Move Up{X=12,Y=3}
Move Right{X=12,Y=1}
Move Down{X=14,Y=1}
Move Left{X=14,Y=4}
Move Down{X=13,Y=4}
Move Left{X=13,Y=18}
Move Up{X=11,Y=18}
Move Right{X=11,Y=16}
Move Down{X=14,Y=16}
Move Right{X=14,Y=17}
Move Down{X=18,Y=17}
Move Left{X=18,Y=19}
Move Up{X=16,Y=19}
Move Right{X=16,Y=16}
Move Up{X=17,Y=16}
Move Right{X=17,Y=12}
Move Down{X=19,Y=12}

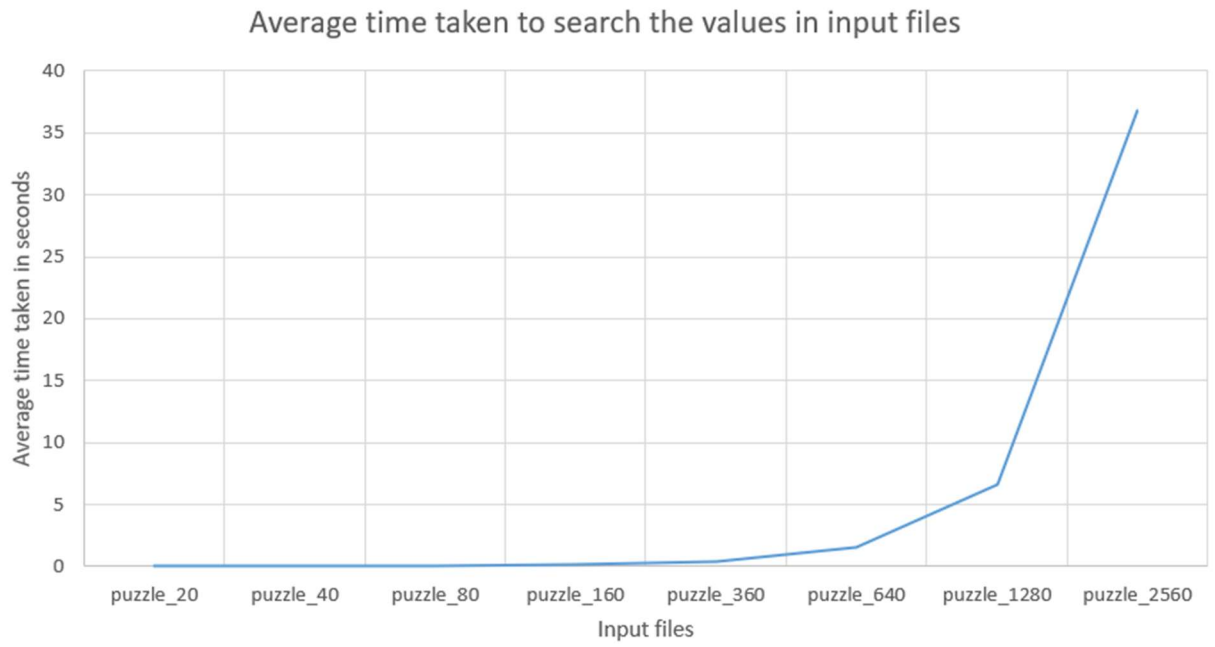

SlidingPuzzle
Move Left{X=18,Y=19}
Move Up{X=16,Y=19}
Move Right{X=16,Y=16}
Move Up{X=17,Y=16}
Move Right{X=17,Y=12}
Move Down{X=19,Y=12}
Move Left{X=19,Y=14}
Move Up{X=16,Y=14}
Move Left{X=16,Y=13}
Move Up{X=2,Y=13}
Move Right{X=2,Y=11}
Move Down{X=4,Y=11}
Move Left{X=4,Y=14}
Move Down{X=3,Y=14}
Move Left{X=3,Y=18}
Move Up{X=1,Y=18}
Move Right{X=1,Y=16}
Move Down{X=4,Y=16}
Move Right{X=4,Y=17}
Move Down{X=8,Y=17}
Move Left{X=8,Y=19}
Move Up{X=6,Y=19}
Move Right{X=6,Y=16}
Last move {X=7,Y=16}
End at{X=7, Y=1}
Finished!
Completion time in seconds: 0.0403418

C)

Input File	Trial 1	Trial 2	Trial 3	Average Time(Seconds)
maze10_1	0.0278156	0.0276264	0.030797801	0.0287466
maze10_2	0.0270123	0.026976299	0.024861199	0.026283266
maze10_3	0.0428536	0.029726	0.030234099	0.034271233
maze15_1	0.0394171	0.0300781	0.0313139	0.033603033
maze15_2	0.029106	0.0311291	0.0335411	0.031258733
maze15_3	0.029048	0.029628	0.0336372	0.030771067
maze20_1	0.0331603	0.0439466	0.0371176	0.038074833
maze20_2	0.0294368	0.031554	0.0332387	0.031409833
maze20_3	0.0324151	0.032772	0.0347735	0.0333202
maze25_1	0.0349478	0.038358	0.0379505	0.037085433
maze25_2	0.0338287	0.0382442	0.0412834	0.037785433
maze25_3	0.0374918	0.0371387	0.0395769	0.038069133
maze30_1	0.0514792	0.043782	0.0386592	0.044640133
maze30_2	0.0376882	0.0439484	0.0384539	0.040030167
maze30_3	0.0394006	0.0379557	0.036456	0.037937433



Input File	Trial 1	Trial 2	Trial 3	Average Time(Seconds)
puzzle_20	0.0438209	0.0311634	0.0354782	0.036820833
puzzle_40	0.0413481	0.0465939	0.0449742	0.0443054
puzzle_80	0.078751	0.0929425	0.0977566	0.0898167
puzzle_160	0.1554928	0.1473333	0.1559815	0.152935867
puzzle_360	0.4071835	0.5057859	0.4361504	0.4497066
puzzle_640	1.5475168	1.5482611	1.6537095	1.583162467
puzzle_1280	6.6850898	6.7024624	6.3220545	6.5698689
puzzle_2560	36.9143996	37.2043045	36.4741908	36.8642983



The graph above demonstrates how the execution time varies with the amount of data in the text files. In the graph above, we can see that as the number of input components grows, time expands linearly. As the number of input elements grows, time nearly quadruples, and n doubles, therefore the graph is written in big o notation as 0(n2).