**Data Structure**

**Maze Solver**

**Project Report**

***By***

**Marina Zakaria**

**Maze Solver Project Report**

**Faculty of Engineering, Alexandria University.**

**Computer & Systems Engineering Dpt.**

**Index**

1. Introduction 4
2. Overview 5
3. Features 5
4. Data Structures 6
5. Functions
6. Sample runs

**Introduction**

This project was assigned in Data Base course in the date: Friday, May 5th, 2017.

And due : Monday,May 22th, 2017.

As the final project of the course, we were assigned this final project individually .

This delivery is by :

*Marina Zakaria Boules Youssef Zakaria*

**Overview**

This program intends to solve a maze given read from a file and the user gets to choose the way of solving either using DFS algorithm or BFS algorithm .

**Features**

*In our implementation,*

The program can:

Read input maze from file

Deduce the maze solution based on DFS algorithm

Deduce the maze solution based on BFS algorithm

**Data Structures**

As we are learning the course Data Structures-1 , we are using our own implementations of Data Structures in this project.

using Singly Linked Lists, Stack and Queue, of my own implementation.

How we used them is described thoroughly in the respective interfaces in the “Functions” section.

Here is the ILinkedList interface:

ILinkedList {

/\*\*

\* Inserts a specified element at the specified position in the

\* list.

\*/

public void add(int index, Object element);

/\*\* Inserts the specified element at the end of the list. \*/

public void add(Object element);

/\*\* Returns the element at the specified position in this list.

\*/

public Object get(int index);

/\*\*

\* Replaces the element at the specified position in this list

\* with the specified element.

\*/

public void set(int index, Object element);

/\*\* Removes all of the elements from this list. \*/

public void clear();

/\*\* Returns true if this list contains no elements. \*/

public boolean isEmpty();

/\*\* Removes the element at the specified position in this list. \*/

public void remove(int index);

/\*\* Returns the number of elements in this list. \*/

public int size();

/\*\*

\* Returns a view of the portion of this list between the

\* specified

\* fromIndex and toIndex, inclusively.

\*/

public ILinkedList sublist(int fromIndex, int toIndex);

/\*\*

\* Returns true if this list contains an element with the same

\* value as the specified element.

\*/

public boolean contains(Object o);

}

The Stack interface

**public** **interface** IStack {

/\*\*

\* Inserts a specified element at the specified position in the list.

\* **@param** index zero-based index

\* **@param** element object to insert

\*/

**public** **void** add(**int** index, Object element);

/\*\*

\* Removes the element at the top of stack and returns that element.

\* **@return** top of stack element, or through exception if empty

\*/

**public** Object pop();

/\*\*

\* Get the element at the top of stack without removing it from stack.

\* **@return** top of stack element, or through exception if empty

\*/

**public** Object peek();

/\*\*

\* Pushes an item onto the top of this stack.

\* **@param** object to insert

\*/

**public** **void** push(Object element);

/\*\*

\* Tests if this stack is empty

\* **@return** true if stack empty

\*/

**public** **boolean** isEmpty();

/\*\*

\* Returns the number of elements in the stack.

\* **@return** number of elements in the stack

\*/

**public** **int** size();

}

And the queue interface

**public** **interface** IQueue {

/\*\*

\* Inserts an item at the queue front.

\*/

**public** **void** enqueue(Object item);

/\*\*

\* Removes the object at the queue rear and returns it.

\*/

**public** Object dequeue();

/\*\*

\* Tests if this queue is empty.

\*/

**public** **boolean** isEmpty();

/\*\*

\* Returns the number of elements in the queue

\*/

**public** **int** size();

}

**Functions**

Read from file method

**public** **static** String[] read(**final** File maze) {

BufferedReader br;

FileReader fr;

**try** {

fr = **new** FileReader(maze);

br = **new** BufferedReader(fr);

String line = br.readLine();

String[] descriotion = **new** String[2];

descriotion = line.split(" ");

*numberRows* = Integer.*parseInt*(String.*valueOf*(descriotion[0]));

*numberColumns* = Integer.*parseInt*(String.*valueOf*(descriotion[1]));

String[] grid = **new** String[*numberRows*];

line = br.readLine();

**int** i = 0;

**while** (line != **null**) {

grid[i++] = line;

line = br.readLine();

}

**return** grid;

} **catch** (Exception e) {

System.***out***.println(e);

**return** **null**;

}

}

}

Solve DFS method

Solve BFS method

**Test cases**

**Test (1)**

5 5

##..S

..#..

.##..

E....

..###

Track using DFS

(0,4)

(1,4)

(2,4)

(3,4)

(3,3)

(3,2)

(3,1)

(4,1)

(4,0)

(3,0)

Solve using BFS

(0,4)

(1,4)

(2,4)

(3,4)

(3,3)

(3,2)

(3,1)

(3,0)

*Thank you.*