Monash University
Department of Electrical & Computer Systems Engineering

**ECE2071 Computer Organisation & Programming**

# Lab VI and Assignment 2: Solving Mazes with the Lee Algorithm

## Contents

## 1 Objectives

In this lab activity and assignment, we will learn how the fundamental data structures such as linked lists, queues and stacks can be used for solving some engineering problems.

## 2 Introduction

One application of computers is the study of algorithms. In this laboratory experiment and the associated assignment, an algorithm, the Lee Algorithm [1, 3] for finding a path between two locations (called the *source* and *target* cells) in a maze is investigated. The Lee Algorithm guarantees to find the shortest path – if it exists – between two points in a maze. It is an extremely effective tool and is universally used, such as computer aided design of printed circuit board layouts.

Details of the Lee Algorithm are provided in Appendix A.

## 3 Programming Assignment

Write a C program (called `ece2071_asg2.c`) that will implement the Lee Algorithm. Your program should

1. Accept, as a command line argument, the name of an ASCII text file that describes the maze. Each row of the maze is stored as a single line consisting of '`#`' or '`.`' characters. In addition, somewhere in the file one '`S`' and one '`T`' representing the start and target

```
# # # # # # # # # # # #
# . . . # . . . . . . #
. . # . # . # # # # . #
# # # . # . . . . # . #
# . T . . # # # . # . .
# # # # . # . # . # . #
# . . # . # . # . # . #
# # . # . # . # . # . #
# . . . . . S . . # . #
# # # # # # . # # # . #
# . . . . . . # . . . #
# # # # # # # # # # # #
```

Figure 1: A maze of 12 by 12 cells. The source cell coordinates are (9,7) and the target cell is at (5,3).

points of the path to be discovered by the algorithm (See Figure 1, which is the same maze we used in our maze solver laboratory experiment).

2. Print, upon completion:
   - if it finds a path from the source to target, it should print a trace of the path as a sequence of matrix coordinates separated by space characters. For example, if we run the program of the maze shown in Figure 1, it should print the following sequence:
     ```
     5,3 5,4 5,5 6,5 7,5 8,5 9,5 9,6 9,7
     ```
   - if no path is found, it should print the following message
     ```
     No path found.
     ```

You are not allowed to use global or `static` arrays. Starting from the source, cells to be visited should be placed in a queue (of which elements are dynamically allocated) and solution path (if ever exists) should be placed in a dynamically allocated stack as explained in Chapters 12.5 and 12.6 of [2].

Note that your program needs to discover the dimensions of the maze by itself (but you can safely assume that we will be dealing with square mazes).

Rename your program as `ece2071_asg2.c`, and submit as a single `.c` file containing only ASCII text characters (no other formats will be accepted). Do not submit any other files. Please read carefully the submission rules and marking procedure provided in Appendix B.

## A The Lee Algorithm

Consider the maze configuration shown in Figure 1. So, as before, we represent the maze as a grid of dots (`.`) and hashes (`#`). The hashes represent the walls of the maze and the dots represent cells in the possible paths through the maze. Moves can only be made to the left, right, up or down from the current location, 1 cell at a time, and diagonal moves are not allowed. Naturally, if a cell contains a `#`, it cannot be moved into. The cells labelled with the characters '`S`' and '`T`' are the source and target cells respectively.

```
# # # # # # # # # # # #   # # # # # # # # # # # #   # # # # # # # # # # # #
# . . . # . . . . . . #   # . . . # . . . . . . #   # . . . # . . . . . . #
. . # . # . # # # # . #   . . # . # . # # # # . #   . . # . # . # # # # . #
# # # . # . . . . # . #   # # # 8 # . . 8 7 # . #   # # # . # . . . . # . #
# . T . . # # # . # . .   # . T 7 6 # # # 6 # . .   # . T 7 6 # # # . # . .
# # # # . # . # . # . #   # # # # 5 # 3 # 5 # . #   # # # # 5 # . # . # . #
# . . # . # . # . # . #   # 7 6 # 4 # 2 # 4 # . #   # . . # 4 # . # . # . #
# # . # . # 1 # . # . #   # # 5 # 3 # 1 # 3 # . #   # # . # 3 # . # . # . #
# . . . . 1 S 1 . # . #   # 5 4 3 2 1 S 1 2 # . #   # . . . 2 1 S . . # . #
# # # # # # 1 # # # . #   # # # # # # 1 # # # . #   # # # # # # 1 # # # . #
# . . . . . . # . . . #   # 7 6 5 4 3 2 # . . . #   # . . . . . . # . . . #
# # # # # # # # # # # #   # # # # # # # # # # # #   # # # # # # # # # # # #
         (a)                         (b)                        (c)
```

Figure 2: Diagrams that sketch the operation of the Lee algorithm while finding the shortest possible path between a source and a target cell: (a) First wavefront generated by the Lee Algorithm. (b) Waveform propagation from the source cell to the target cell. (c) A path connecting the source and target cells.

To find a path starting from '`S`' to '`T`', empty cells adjacent to the source are first labelled 1, indicating the cost of reaching these cells from the source. This is shown in Figure 2(a).

Next, a cost of 2 is placed in each empty cell adjacent to cells labelled 1, and so on. Thus, a diamond shaped expanding wavefront of increasing numbers propagates until the target cell is reached or there are no remaining empty cells adjacent to any of the labelled cells. Figure 2(b) illustrates the complete waveform propagation.

If the target cell has been reached, the algorithm backtracks from the target, moving at each step to any cell with a lower cost. Figure 2(c) shows the algorithm with a backtrack path. Note that, quite often there will be alternative choices for the backtrack path.

## B Submission Rules and Marking Procedure

Please read carefully the following submission rules and marking procedure. They will be *strictly* applied:

1. This is a single person activity. Group submissions will not be accepted. Do not share your code with anybody, and do not use someone else's code. Please read carefully Monash University's information pages about academic integrity, plagiarism and collusion[1].

2. We will compile your program on a Windows 10 command window by using the following command:
   ```
   gcc -std=c17 -pedantic -Werror ece2071_asg2.c  -o ece2071_asg2 -lm
   ```
   If no executable file is generated, we will send you the error output, and invite you to re-submit. But, if you decide to re-submit, we will consider the submission as late, and consequently you will **lose at least 50% of the allocated mark**.

[1] https://www.monash.edu/students/study-support/academic-integrity

Therefore, before submitting, make sure that your program compiles with the above command in a Windows 10 machine command window of one of our department's laboratory computers.

3. Do not use any non-standard libraries. Our system will not have them, and your program will inevitably fail if you use any non-standard libraries.

4. Similarly, do not include any non-standard header ("`.h`") files.

5. We will run the generated executable file with the following command:
   ```
   ece2071_asg2 <mazefilename>
   ```
   where "`mazefilename`" will be one of our test maze files. They will be of increasing dimensions (there will be at least one huge maze). Note that our Windows 10 machine has 16 GB of memory, so be careful about the memory usage limits of your program.

   If your program produces the expected output in a reasonable time for each maze file (the hard-limit is 10 minutes, if your program gets stuck in an infinite loop, the system will terminate it after 10 minutes) you will get at least 60% of the allocated mark.

   We will then rank all the successful programs from shortest run time to the longest. Fastest 25% of the programs will get the full remaining 40%, second fastest group of 25% will get an extra 30%, third fastest group of 25% will receive 20% and final 25% will get an extra 10%.

6. If your program crashes when we run, we will examine the program and its output, and assign a mark between 20% and 60%. If no useful output is generated, we will examine the source code and assign a mark between 0% and 20%.

## References

[1] A. D. Brown and M. Zwolinski. Lee Router Modified for Global Routing. *Computer Aided Design*, 22(5):346–365, June 1990.

[2] P. Deitel and H. Deitel. *C How to Program.* Pearson, (Available on-line via Monash Library), Eighth edition, 2016.

[3] C. Y. Lee. An Algorithm for Path Connection and Its Applications. *IRE Transactions on Electronics and Computers*, pages 346–365, September 1961.