CS201- Data Structures
Programming Assignment No. 2

Problem 1- Optimal Way-Out from a Maze

You are now well aware of how to get a solution rom a maze. The content of the array can be any character {1 (path), 0 (block), **s** (start), **e** (end), ! (visited)}.

The input maze may contain multiple-paths and you need to implement a recursive path finding approach that enumerate all cell of the array that are on the unique path in order of traverse from start s to end e. You need to enumerate all paths. Here is an example of a maze.

s	1	1	1	1	1	0
1	0	0	1	1	0	1
1	0	0	0	1	0	1
1	1	1	e	1	1	1

There are three possible paths:

Path#1= {(0,0),(0,1),(0,2),(0,3),(1,3),(2,3),(3,3)} Cost=7

Path#2= {(0,0),(1,0),(2,0),(3,0),(3,1),(4,1),(4,2),(4,3),(3,3)} Cost=9

Path#3= {(0,0),(1,0),(2,0),(3,0),(4,0),(4,1),(4,2),(4,3),(3,3)} Cost=9

All you need to develop a recursive routine for finding path, the search is only allowed to follow {Left, Right, Up and Down} from any location. The output for this problem is complete path from starting to end location. The validation of input cases required as start and end location must be at the boundary of the maze. There can be several paths for all valid input cases. You need to enumerate all paths from starting to each cell followed till the end. The cost of a path is total number of cell traveled. Hence the shortest path or optimal path is the minimum length path.

Input

The input is from the file, the first line contains two integers n and m, representing the dimension of the maze. The maze is a 2-dimensional array of char. The next n lines contain the rows of the maze; each row contains m columns. Hence there are m characters in each line.

Output

The output file contains paths enumerated as path numbers with cost of each path, starting from the minimum cost path.

Input file							Output file
4 7							Path#1={(0,0),(0,1),(0,2),(0,3),(1,3),(2,3),(3,3)}
s	1	1	1	1	1	0	Cost=7
1	0	0	1	1	0	1	Path#2={(0,0),(1,0),(2,0),(3,0),(3,1),(4,1),(4,2),(
1	0	0	0	1	0	1	4,3),(3,3)} Cost=9
1	1	1	e	1	1	1	Path#3={(0,0),(1,0),(2,0),(3,0),(4,0),(4,1),(4,2),(
							4,3),(3,3)} Cost=9

Problem 2- Infix to Postfix conversion and evaluation

Arithmetic expressions are made of operators (+, -, /, *, ^, etc.) and operands (either numbers, variables or, recursively, smaller arithmetic expressions). The expressions can be written in a variety of notations. In this problem, you will focus on two standard arithmetic expression notations: infix and postfix. In Infix expression, operators are written in-between their operands. This is the usual way we write expressions, for example x+y. Similarly, Postfix notation (also known as "Reverse Polish notation"): x y + Operators are written after their operands. There is a general algorithm that convert an infix expression to postfix using stack (data structure). The algorithm is given below:

Algorithm (Converting an Infix expression to Postfix expression

1) Examine the next element in the input.
2) If it is an operand, output it.
3) If it is opening parenthesis, push it on stack.
4) If it is an operator, then

    i) If stack is empty, push operator on stack.
    ii) If the top of the stack is opening parenthesis, push operator on stack.
    iii) If it has higher priority than the top of stack, push operator on stack. (operator priorities ^, *, /, +, and -.
    iv) Else pop the operator from the stack and output it, repeat step 4.

5) If it is a closing parenthesis, pop operators from the stack and output them until an opening parenthesis is encountered. pop and discard the opening parenthesis.
6) If there is more input go to step 1
7) If there is no more input, unstack the remaining operators to output.

Similarly, you must provide another function that can evaluate any postfix expression by using stack. This can only be called on postfix expression otherwise it gives error message. This function will also use stack for getting operands from the postfix expression, while getting any operator, it will pop the corresponding two (one) operands and perform the operation, finally put back the result on the stack. When you finished the input postfix expression you will get the final evaluated result either on the stack or in the operand. The operator precedence rules are already mentioned in infix to postfix algorithm.

Input/Output: The input file will contain one valid expression in infix form. The output file must contain the valid postfix version of that expression and in the next line it must have its evaluated result.

Problem 3- Simulation of Queuing Systems

The Shop-n-Carry (SC) is a big chain of departmental stores. The SC would like to improve its customer services. There has been a complained for some period that the waiting time to avail Point-of-Sale terminal services is unbearable for some areas in stores. SC is planning to improve the services by means of installing more POS terminals to those stores, which are in heavy use, and customers have to wait for making payments. You have been hired, as a consultant by the SC president to determine whether this idea is feasible or not. The management has decided that if the average waiting time of a user at a POS is greater than 3 minutes to get their services. They will install one more POS terminal machine. It is provided that the arrival rate of POS customer is anytime within the opening hours and the average service time is between 3 to 8 minutes. The POS is operational 18 X 7. Your job is to run a simulation program for SC POS for 18 hours, and calculate the average waiting time for customers on hourly basis. Identify the busy hours. Use the least count of the clock as discrete unit of minutes. Put the entire statistic in the output file which is as follows:

- Number of customers
- Average waiting time (for each hour – in a separate line)
- Average Customers (for each hour – in a separate line)
- POS utilization time (Average Service Time)