CS218- Data Structures
Programming Assignment No. 2
Fall 2020

Instruction

This is the second programming assignment for the course CS218- Data Structures in the offering Fall 2020. The assignment comprises of three problems. It is suggested that you should start working on the assignment at your earliest. This seems a good amount of intellectual work required to complete it. Each question should be solved on Hackerrank platform, invitation to CS218-Programming Assignment will soon be sent to all of you.

Important Note: The assignment is for individual and there should not be any case of cheating. You can have discussion about any problem and approach among yourself but do not share code and instruction for any problem. Try to write the code at hackerrank alone. The hackerrank count down will start from Wednesday November 04, 2020 Morning 9AM and will expire on November 16, 2020 at 9:PM

Due Date: November 16, 2020 21:00PM (Fixed Deadline)

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 {0 (path), 1 (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 cells of the array that are on the unique path in order to traverse from start s to an end e. You need to enumerate all paths. Here is an example of a maze.

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

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 {Right, Down, Left and Up} from any location with the same preferences. 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 file format: 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 file format: The output file contains paths enumerated as path numbers with cost of each path, starting from the minimum cost path. Each path is the sequence of move performed on the grid that is cell by cell transition. See the input output case.

Input File	Output File
4 7 S 0 0 0 0 0 1 0 1 1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 e 0 0 0	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

Problem 2: Process scheduling with Quota

In operating systems, there are several processes with different priority level do exists. One of the process scheduling termed as "Round Robin" is considered all the active processes in a queue and schedule each process as per their allocated quotas. Consider for example that there are three process P1 having 20 units of cpu need and quota of 4 units, P2 having 12 units of cpu and quota of 2 units, and finally P3 24 units of CPU and quota of 8 units. (Assume all processing need is actually divisible by unit quota for the respective process). You need to determining the process completion order for any given instance of scheduling task. This is the order in which each process complete the cpu needs.

Assume all process are available in active queue simultaneously and their natural order is given by the process order in input file. The first process is head of the queue and last process is the end element of the queue.

Input file format: The input is from the file; the first line contains a single integer representing number of process n. The second line gives any permutation of first n numbers. Representing the process in the active queue. The next line gives the unit quota of each process exactly in the given permutation order. The next line gives the actual requirements of CPU time by each process in the same order.

For example:
5
2 3 4 5 1
2 2 2 4 2
12 6 4 4 2

Output file format: The output file contains a single permutation of process which is the processing completion order. In the above given example the output will be

5 1 4 3 2

Input File	Output File
5 2 3 4 5 1 2 2 2 4 2 12 6 4 4 2	5 1 4 3 2

Problem 3: Infix Expression to Pre and Post Fix

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: Prefix and Postfix. You all are very familiar of Infix notations. 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. Similarly, for Prefix notation the output will be + x y

There are general algorithms that convert an infix expression to prefix/postfix using stack (data structure). The algorithms are given below:

Algorithm (Converting an Infix expression to Prefix expression

1)First, reverse the given infix expression.
2) Scan the characters one by one.
3)If the character is an operand, copy it to the prefix notation output.
4)If the character is a closing parenthesis, then push it to the stack.
5)If the character is an opening parenthesis, pop the elements in the stack until we find the corresponding closing parenthesis.
6) If the character scanned is an operator

i) If the operator has precedence greater than or equal to the top of the stack, push the operator to the stack.
ii) If the operator has precedence lesser than the top of the stack, pop the operator and output it to the prefix notation output and then check the above condition again with the new top of the stack.

7) After all the characters are scanned, reverse the prefix notation output.

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.

Input file format: The input for this problem is a string of 80 character comprises of operators and operands. A single line of the input.

Output file format: There are two lines of output. The first line is prefix equalivent of the given expression. The second line is postfix Expression of the same.

Input File	Output File
A + B * C + D	+ + A * B C D A B C * + D +