CS2351 Data Structures

Homework 3

Due: 11:59 pm, May 3, 2010

In this assignment, you may choose to work alone or work in pairs, though the minimum requirements will be different. If you work in pairs, please submit your file together as one copy, and mark clearly your names and your university numbers in the submitted file.

Part I. Written (50%)

1. Consider the mathematical expression $12 \div 6 + 7 \times 5 \times 2 - 4$.

   Note that $\div$ and $\times$ have higher priority than $+$ and $-$, as usual. For instance, $1 + 2 \times 3$ means $1 + (2 \times 3)$ instead of $(1 + 2) \times 3$. Also, all these operators are *left-to-right* operator. For instance, $1 + 2 + 3$ means $(1 + 2) + 3$ instead of $1 + (2 + 3)$.

   (a) Build the expression tree of the expression.

   (b) Write down the prefix notation and the postfix notation of the expression.

   (c) Evaluate postfix notation with a stack, and show the key steps.

2. Assume there are only $+$, $-$, $\times$ and $\div$ in the mathematical expression. Here is an algorithm to transform infix notation to postfix notation of expression by using a stack. (See Tutorial 2 for further explanation.)

Algorithm Transform infix to postfix

```
1: Scan the infix expression from left to right ;
2: x = the operator or the number that is scanned ;
3: if x is a number then
4:    Output x immediately ;
5: else if x is × or ÷ then
6:    y = top of stack ;
7:    if y is × or ÷ then
8:       Pop y from the stack, and output y immediately ;
9:    end if
10:   Push x to the stack ;
11: else if x is + or − then
12:   while stack is not empty do
13:      y = top of stack ;
14:      Pop y from the stack, and output y immediately ;
15:   end while
16:   Push x to the stack ;
17: end if
18: Repeat Steps 2 to 17 until all the expression is scanned ;
19: Pop and output all the operators remain in the stack ;
```

(a) We now introduce a new left-to-right operator $\triangle$, where $\triangle$ has highest priority. Modify the above algorithm to transform infix notation to postfix notation of an expression that may contain $+, -, \times, \div$, and $\triangle$.

(b) Now, we further introduce another operator $\circ$, where $\circ$ has highest priority (even higher than $\triangle$), and $\circ$ is a right-to-left operator. That is, given $a \circ b \circ c$, the expression is equivalent to $a \circ (b \circ c)$. Modify your algorithm in part (a) to transform infix notation to postfix notation of an expression that may contain all the six operators.

3. Let G be a directed graph and M be its adjacency matrix.

(a) Let $M^2 = M \times M$ where the current $\times$ is matrix multiplication. Explain the physical meaning of the value of each entry in M^2. (Hint: Recall that $M[u, v] = 1$ if there is a directed edge from u to v, and $M[u, v] = 0$ otherwise. Now, what does the value in $M^2[u, v]$ mean?)

(b) A *triangle* in G is defined as a sequence of vertices (u, v, w, u) such that $(u, v), (v, w)$, and (w, u) are directed edges found in G.

Design an algorithm to check whether there is a triangle in G? Can you use only *one* matrix multiplication?

Part II. Programming (50%)

In this part, you are asked to implement the two Topological Sort algorithms discussed in the class. Recall that one algorithm requires a queue, and the other requires a stack. If you work alone, you can just implement any one of them. If you work in pairs, you will need to implement both algorithms.

You can develop your code using either C or C++. Your code should be compilable using Dev-C++. We will assess your code by the correctness (35%) and the readability (15%). Note that you should not use any predefined system calls or standard template library calls to perform the queue or stack operations.

Specifications:

Your program of both implementations will expect two arguments when it runs. The first one gives the filename of the input file that contains a directed graph in which the vertices are to be sorted, and the second one gives the filename of the output file where the sorted vertices are stored. For example, suppose your program name is `toposort`, the input filename is `input.txt`, and the output filename is `output.txt`. Then when we run your program, we will type: `./toposort input.txt output.txt` so that your program sorts the numbers in the `input.txt` and store the result in `output.txt`.

The format of the input file is as follows: The first line stores an integer n that indicates how many vertices are to be sorted. We assume that the label of the vertices are from 1 to n. Then the file is followed by n lines, where the ith line lists the neighbors of vertex i, and is ended by 0. For example, the input file may look like:

```
3
2 3 0
3 0
```

```
0
```

The graph represented by the above file contains 3 vertices, with the directed edges (1,2), (1,3), and (2,3). The format of the output file is simply the sequence of the vertices sorted in topological order, each separated by a single space. For example, the output file of the previous example should look like:

```
1 2 3
```

Remarks:

When you implement the topological sort algorithms, you will need to implement your own queue or stack. You may use linked list or array to implement these structures. To help the implementation, you may define some helper functions. For instance, in a simple implementation of a queue using a global array, we may define:

```
// Assume a global queue Q[1...]  contains enough space for the queue,
// with head at location H and the tail at location T
// Also, assume that num stores the number of items in the queue

// The following enqueues the item x to the queue
void enqueue(int x)
{
    if ( num == 0 ) {
       Q[H] = x ; T = H ; num++; }
    else {
       T = T + 1; Q[T] = x ; num++; }
}


// The following dequeues and returns the first item from a non-empty queue
// Thus, we need to check if the queue is empty before calling dequeue
int dequeue()
{
    num--; H = H + 1;
    return Q[H-1];
}


// The following checks if the queue is empty
// It returns 1 if empty, and 0 otherwise
int is_empty()
{
    if ( num > 0 ) return 0;
    else return 1;
}
```