CS 008
Lecture notes
3/21/24

1 Outline

- Big-O notation (again)
- Infix, Prefix, Postfix Notation
- Queues (Circular array implementation)

2 Big-O worksheet

Some of the problems on the worksheet will be proof based:

Definition:

Formal definition of Big-O notation:

$f(n) \in O(g(n))$ if $f(n) \leq kg(n)$ for all $n \geq N$ where k and N represent arbitrary constants.

If we choose a value N that represents a specific number for operations, if we can express $f(N)$ as a scalar multiple of $g(N)$ (multipled by the scalar k), $f(n)$ would be of the same time complexity of $g(n)$.

The formal definition of the Big-O notation means that if we can find a value k that can be multipled by the function $g(N)$ to get the same *(or bigger)* value than $f(N)$, then $f(n)$ has a time complexity of $O(g(n))$.

Example:

Proof: If $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$, then $f(n)$ is $O(h(n))$.

We know $f(n) \leq G_1 g(n)$ for all $n \geq N_1$ and $g(n) \leq G_2 h(n)$ for all $n \geq N_2$.

We want to show that $f(n) \leq G_3 h(n)$ for all $n \leq N_3$

Define: $N_3 = max(N_1, N_2)$
Plugging in for the inequality:

$$f(n) \leq G_1 g(n) \leq G_1 G_2 h(n) \text{ for all } n \geq N_3$$

3 Infix, Prefix, Postfix

Infix notation examples:

- $2 + 2$
- $3 * 4$
- 2^3

Prefix examples:

- $+22$
- $*34$
- $\hat{}23$

Postfix examples:

- $22+$
- $24*$
- $23\hat{}$

3.1 Infix to Postfix conversion

We can convert from infix notation to postfix notation through the use of a stack:

Algorithm:

Consider a regular infix expression (ex: a - b + c) and consider an empty list that we will consider to be our **postfix** expression.

- **Step 1**: Tokenize the expression (split up the numbers, variables, operators and parentheses... etc)
- **Step 2**: Push the parentheses into the stack '(' and ')'
- **Step 3**: Repeat the following steps (4-7) if you find a token (numbers, operators and parentheses)
- **Step 4**: If you find a '(' token, add it to the top of the stack
- **Step 5**: If you find a ')' token, pop the operators that we pushed into the stack (from the top) until we find a '('. Append (add onto the end of the list) the tokens to the postfix expression.
- **Step 6**: If you find an operator, check the top of the stack and if the operator in the stack is of equal or greater precedence than the one found, pop it and append it to the postfix expression. The found operator is then pushed to the top of the stack.
- **Step 7**: If the token is a value (number or variable), append it to the postfix expression.

4 Queue

A **Queue** is an abstract data type similar to a **stack**, however unlike a stack, a queue is **FIFO** *(first in, first out)*.

ADT Definition:

push (dequeue): Add element (to the rear of the queue)
pop (enqueue): Remove an element from the front of the queue
front: View the first element in the queue
empty: Check if the queue is empty
Size: Get the number of elements in the queue

4.1 Circular Arrays

When working with a limited sized queue, one of the issues when popping the beginning of the queue is figuring out what to do with the empty space where the front used to be. We could shift the entire array to have the front at the beginning of the array to fill in the empty space however it is not efficient (because we'd have to have an entire $O(n)$ operation to shift this).

To solve this problem, at the end of the array, we go back to the beginning of the array (not the front of the queue since we're assuming there's still 'space' in the queue). As a result, we don't have to deal with the shifting of the array and we could simply just go back to that empty space we had in the beginning of the array.