Stacks and Queues

Week 3

Required Activities

- Check Announcements regularly (every 2-3 days)
- Read DSA book: Chapter 5 and 6
- Assignment 1 due tomorrow 11/9
- Start working on Assignment 2 due end of week 5 (11/30)
- Watch the example video

Stack

- Abstract data type (ADT) – imagine a stack of plates
- Ordered list where insertion and deletion is done at one end, the *top*.
- The last element inserted is the first one deleted – Last in First out (LIFO)
- Insertion operation is called *push* and deletion is called *pop*
- Empty stack means there are no elements
- Operations:
  - Create empty stack (may allocate storage depending on programming language)
    OR
  - Create empty stack with max capacity
  - Push(item) – insert element on top (need to check for full when using array)
  - Pop() – delete and return top element from the stack (need to check if empty)
  - isEmpty() – returns true if no elements and false otherwise
  - Peek() – optional; returns the value of the top element without deleting
  - isFull() – optional depending on implementation; returns true if cannot add element
- Can be implemented as an array or linked list
- Time efficiency for all above operations is **O(1)** and space complexity is **O(n)**
- If stack has to grow in push (array usually doubles) than time complexity is **O(n)**
- Deleting stack time complexity is **O(1)** for array and **O(n)** for linked list

Some Uses of Stack

- Expression evaluation and conversion between prefix, postfix and infix notations
- Implementation of functions calls to include recursion
- Undo sequence in word processing application
- Back button in web page
- Matching tags in HTML and XML
- Used in algorithms (e.g. tree traversal)

Array versus Linked List

- Most operations are constant time O(1) for both except deleting stack where linked list is O(n)
- Array can be more costly for time complexity when it has to grow (it needs to create new array and copy elements over to that array)
- Linked list needs extra space and time to manage reference to next element

Queue

- Abstract data type (ADT) – imagine a line to cashier in a store
- Ordered list where insertion is done at one end (*rear*) and deletion is done at the other end (*front)*
- The first element inserted is the first one deleted – First in First out (FIFO)
- Insertion operation is called *enQueue* and deletion is called *deQueue*
- Empty queue means there are no elements
- Operations:
  - Create empty queue (may allocate storage depending on programming language) or with max capacity
  - enQueue(item) – insert element at rear (need to check for full when using array)
  - deQueue() – delete and return front element from the queue (need to check if empty)
  - isEmpty() – returns true if no elements and false otherwise
  - Peek() – optional; returns the value of the front element without deleting
  - isFull() – optional depending on implementation; returns true if cannot add element

- Can be implemented as an array or linked list
- Time efficiency for most above operations is **O(1)** and space complexity for enQueue is **O(n)**
- If queue has to grow in enQueue (array usually doubles) then time complexity is **O(n)**
- If move items on deQueue than time complexity is **O(n)**
- Deleting queue time complexity is **O(1)** for array and **O(n)** for linked list

Types of Queues

Array implementation

- **Linear Queue** – moves elements forward on dequeue
- **Circular Queue** – keeps track of front and end where no moving is necessary

Priority Queue

- Element is assigned a priority
- Order of deletion:
  - Highest priority element processed first
  - If same priority, element added first to queue is processed first
- Implemented as: sorted linear linked list, multiple queues per priority, or heap (complete binary tree)

Uses of Queue

- Scheduling of jobs in operating system or application
- Simulation of real queues such as a line at supermarket
- Waiting times of customers at call center
- In other algorithms

Array versus Linked List

- Most operations are constant time O(1) for both except deleting queue where linked list is O(n)
- If need to move elements (linear array) than deQueue is O(n)
- Array can be more costly (need to allocate new array and copy elements over) when it has to grow
- Linked list needs extra space and time to manage reference to next element

Queue and Stack examples

- Since **Queue FIFO**, when elements taken off they are in same order as entered: in queue 1 2 3 and take out 1 2 3
- Since **Stack LIFO**, when elements taken off they are in opposite order than entered: in stack 1 2 3 and take out 3 2 1

- To reverse elements in Queue only using queue and stack operation and no recursion ?

- To copy to another Stack in reverse order ?

- To copy Stack to queue in same order ?

- To reverse first n elements in Queue?

Analysis

To reverse first x elements of Queue q of size n:

create empty stack (assume no storage allocation because using linked list implementation)

Loop x times
    s.push(q.deQueue) // take off x elements from queue and put on stack

Loop x times
    q.enQueue(s.pop) // take off x elements from stack and put back on queue

Loop n - x
    q.enQueue(q.deQueue) // take the remaining elements that are not to be reversed and put back on queue

Time complexity:
First loop: if n == x than loops n times and if x=0 than loops 0 times
Second loop: same as first loop
Third loop: if n == x than loops 0 times and if x=0 than loops n times
Therefore: **n==x:** n+n+0 = 2n and **x=0**: 0 + 0 + n = n so in both cases rate of growth is O(n)

Space complexity:
Space only allocated for stack so at max it will be for size of queue n so O(n)

Questions ?

- Post in the discussions
- Send email to RMcFadden@HarrisburgU.edu
- Respond usually within 48hours