L13 - Queues

November 6, 2019

1 Queues with Linked Lists

1.1 What is a queue?

A queue is a collection in which the elements are maintained in the same order in which they were added. A linear collection is the simplest kind and can be implemented as a singly-linked-list.

Generally, we make a queue as a **f**irst-**in**-**f**irst-**o**ut, or **FIFO**.

1.1.1 Enqueue

- add a new element to the *back* of the queue

1.1.2 Dequeue

- remove an element from the *front* of the queue
- return the value of the element

We have some design options for implementing the queue structure. Consider: * functional requirements * actually doing the job * non-functional requirements * execution speed * memory footprint * scalability

1.2 Queue by singly-linked list - first go

The first node in the list represents the front of the queue. The last node in the list represents the back of the queue.

`dequeue()` will remove and return the first node in the queue, in $O(1)$
`enqueue()` will add the new node to the back of the list in $O(n)$
* this is very inefficient to add a new element, as we must traverse the entire queue to get to the end.

1.3 Queue by singly-linked list - second go

What if we change it so the first node in the list is the *back* of the queue?

`enqueue()` will add the new node to the back of the list in $O(1)$
`dequeue()` will remove and return the first node in the queue, in $O(n)$

All we've done is moved our inefficiency from `dequeue()` to `enqueue()`. Nuts.

1.4 Queue by singly-linked list - third go

What if we add another pointer, `rear`, to point to the back of the line?

`enqueue()`