CMPSC 250
Analysis of algorithms

Dr. Aravind Mohan
Allegheny College

January 31, 2018
Lecture 04: A deeper look at Queue ADT (continuation)
Linked List Implementation

- Dequeue - advance head reference

- Inserting at the head is just as easy
Linked List Implementation (2)

- Enqueue - create a new node at the tail

- chain it and move the tail reference

- How about removing at the tail?
Double-Ended Queue

A double-ended queue, or deque, supports insertion and deletion from the front and back.

The deque supports six fundamental methods:

1. **InsertFirst(Q: ADT, o:element):** ADT - Inserts o at the beginning of the deque
2. **InsertLast(Q:ADT, o:element):** ADT - Inserts o at the end of the deque
3. **RemoveFirst(Q:ADT):** ADT - Removes the first element
4. **RemoveLast(Q:ADT):** ADT - Removes the last element
5. **First(Q:ADT):** element - returns, but does not remove, the first element; an error occurs if the deque is empty.
6. **Last(Q:ADT):** element - returns, but does not remove, the last element; an error occurs if the deque is empty.
Doubly Linked Lists

- Deletions at the end of a singly linked list cannot be done in constant time.
- To implement a deque, we use a doubly linked list with special header and trailer nodes.

A node of a doubly linked list has a next and a prev link.

By using a doubly linked list to, all the methods of a deque have constant (that is, $O(1)$) running time.
Doubly Linked Lists (2)

When implementing a doubly linked lists, we add two special nodes to the ends of the lists: the header and trailer nodes.

- The header node goes before the first list element. It has a valid next link but a null prev link.
- The trailer node goes after the last element. It has a valid prev reference but a null next reference.

The header and trailer nodes are sentinel or "dummy" nodes because they do not store elements.
Stacks with Deques

- Implementing ADTs using implementations of other ADTs as building blocks.

<table>
<thead>
<tr>
<th>Stack Method</th>
<th>Deque Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>size()</td>
<td>size()</td>
</tr>
<tr>
<td>isEmpty()</td>
<td>isEmpty()</td>
</tr>
<tr>
<td>top()</td>
<td>last()</td>
</tr>
<tr>
<td>push(e)</td>
<td>insertLast(e)</td>
</tr>
<tr>
<td>pop()</td>
<td>removeLast()</td>
</tr>
</tbody>
</table>
Queues with Deques

<table>
<thead>
<tr>
<th>Queue Method</th>
<th>Deque Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>size()</td>
<td>size()</td>
</tr>
<tr>
<td>isEmpty()</td>
<td>isEmpty()</td>
</tr>
<tr>
<td>front()</td>
<td>first()</td>
</tr>
<tr>
<td>enqueue()</td>
<td>insertLast(e)</td>
</tr>
<tr>
<td>dequeue()</td>
<td>removeFirst()</td>
</tr>
</tbody>
</table>
The Adaptor Pattern

- Using a deque to implement a stack or queue is an example of the adaptor pattern.
- Adaptor patterns implement a class by using methods of another class.
- In general, adaptor classes specialize general classes.
- Two such applications:
  - Specialize a general class by changing some methods. Ex: implementing a stack with a deque.
  - Specialize the types of objects used by a general class. Ex: Defining an IntegerArrayStack class that adapts ArrayStack to only store integers.
Circular Lists

- No end and no beginning of the list, only one pointer as an entry point.
- Circular doubly linked list with a sentinel is an elegant implementation of a stack or a queue.
Classic Josephus problem

Problem Definition: A group of n people are standing in a circle, numbered consecutively clockwise from 1 to n. Starting with person no. 2, we remove every other person, proceeding clockwise. For example, if n = 6, the people are removed in the order 2, 4, 6, 3, 1, and the last person remaining is no. 5. Let $j(n)$ denote the last person remaining. Find some simple way to compute $j(n)$ for any positive integer $n > 1$.

How to solve this problem using queue or deque?

Discuss how to do it?
– Post your suggestions on Slack so that you can get the Attendance and Class participation points.
– I will give the solution in next class.
Questions