 CMPSC 250
Analysis of algorithms

Dr. Aravind Mohan
Allegheny College

January 24, 2018
Lecture 03: A deeper look at Queue ADT
Short Term Goals and Roadmap for next few weeks

- Finish up ADT’s (Chapter 01).
- Discuss how to analyze algorithms and measure efficiency? (Next week) (Chapter 01).
- Look at existing Sorting Algorithms (2-3 weeks) (Chapter 02).
- Our CMPSC 250 structure this semester, is adapted from other instructors who taught this course many times at Allegheny.
- Quiz 01 on Friday, Feb 9th during class timings.
Finishing up Time Series Problem

- **Recall:** The span $S_i$ of a stock’s price on a certain day $i$ is the maximum number of consecutive days (up to the current day) the price of the stock has been less than or equal to its price on day $i$.

- **Approach:** $S_i$ can be easily computed if we know the closest day preceding $i$, on which the price is greater than the price on $i$. If such a day exist let’s call the day as $h_i$, otherwise we conventionally define $h_i = -1$. 
Finishing up Time Series Problem

• \( h(3) = 2; \ h(5) = 1 \)
• \textbf{What is} \( h(6) \)?
• \textbf{span is now computed as} \( S_i = i - h(i) \)
What are possible values of \( h(7) \)? Can it be 1, 3, or 4?

No. \( h(7) \) can be 2 or 5 or 6

We store indices 2, 5, 6 in the stack

To determine the price of \( h(7) \) we compare the price on day 7 with the day 6, day 5 and day 2 in that order.
Finishing up Time Series Problem

- The first price larger than the day 7 gives $h(7)$. The stack should be updated to reflect the price of day 7. It should now contain 2, 5, 7.
Finishing up Time Series Problem

- **An efficient algorithm**

  Algorithm computeSpans2(P):
  Let D be an empty stack.
  for i <- 0 to n-1 do
  h <- 0; done <- false;
  while not (D is empty() or done) do{
  if P[i] >= P[D.top()] then D.pop()
  else
  done <- true
  }
  if D is empty() then h <- -1
  else h <- D.top()
  S[i] <- i-h
  D.push(i)
  return S
What is a Queue?

- A queue differs from stack in that it insertion and removal routines follows first in first out (FIFO) principal.
- Elements can be inserted at any time, but only the element which has been in the queue longest can be removed.
- Elements are inserted in the rear (enqueued) and removed from the front (dequeued).
Analogy of Queue?
Queue Operations

The queues support 4 fundamental methods

1. new():ADT - Creates a new queue.
2. enqueue(Q:ADT, o:element):ADT - Inserts object o at the rear of the queue Q.
3. dequeue(Q:ADT):ADT - Removes the object from the front of the queue; if the queue is empty an error occurs.
4. front(Q:ADT):element - returns, but does not remove, the front element; an error occurs if the queue is empty.
Queue Operations

- The following support methods could also be defined
  1. `size(Q:ADT):integer` - Returns the number of objects in queue Q.
  2. `isEmpty(Q:ADT):boolean` - Indicates if queue Q is empty.

- The following axioms dictates the scope of the operations in the queue.
  1. `Front(Enqueue(New(),v)) = v`
  2. `Dequeue(Enqueue(New(),v)) = New()`
  3. `Front(Enqueue(Enqueue(Q, w),v)) = Front(Enqueue(Q,w))`
  4. `Dequeue(Enqueue(Enqueue(Q, w),v)) = Enqueue(Dequeue(Enqueue(Q, w)),v)`
An Array Implementation

- Create a queue using an array in a circular fashion.
- A maximum size N is specified.
- The queue consists of an N-element array Q and two integer variables:
  - f, index of the front element (head - for dequeue)
  - r, index of the element after the rear one (tail - for enqueue)
An Array Implementation

- "wrapped around" configuration

- What does $f = r$ mean?
An Array Implementation

Algorithm size()
return \((N-f+r) \mod N\)

Algorithm front()
if isEmpty() then
return QueueEmptyException
else
return Q(f)
An Array Implementation

Algorithm dequeue()
if isEmpty() then
return QueueEmptyException
else{
Q[f] <- null
f <- (f+1) mod N
}

Algorithm enqueue(o)
if size() = N-1 then
return QueueFullException
else {
Q[r] <- o
r <- (r + 1) mod N
}
Implementing queue with Linked List

- Nodes (data, reference link or pointer) connected in a chain by links.
- The head of the list is the front of the queue, the tail of the list is the rear of the queue.
- Why not the opposite?
Questions