Solution Provided

Graded ICA205
Qlist

CS 244

Solution Provided

RECALL

Queue and Singly Linked List

- Main queue operations:
  - enqueue(e): inserts an element at the end of the queue
  - dequeue(): removes and returns the element at the front of the queue
  - front(): returns the element at the front without removing it
  - size(): returns the number of elements stored
  - isEmpty(): returns a Boolean value indicating if there are no elements in the queue

- Singly linked list Operations
  - insertFront(e): inserts an element on the front of the list
  - removeFront(): returns and removes the element at the front of the list
  - insertBack(e): inserts an element on the back of the list
  - removeBack(): returns and removes the element at the end of the list

Solution Provided

Graded In-Class Exercise: Qlist

- Describe how to implement a queue using a singly-linked list
  - Based on previous slide
  - Queue operations:
    - enqueue(x), dequeue(), front(), size(), isEmpty()
  - For each operation, give the running time in Big-Oh
  - Submit your word document / powerpoint slide to the appropriate dropbox on D2L

Solution Provided

Queue as a Singly Linked List

- CLAIM:
  - We can implement a queue with a singly linked list
    - The front element is stored at the head of the list
    - The rear element is stored at the tail of the list
  - The space used is $O(n)$
  - Each operation of the Queue ADT takes $O(1)$ time
    - enqueue, dequeue, front, size, isEmpty each take $O(1)$ time
  - The following slides show how
  - NOTE: we do not have the limitation of the array based implementation on the size of the stack because the size of the linked list is not fixed,
  - i.e. the queue is NEVER full.

Solution Provided

Queue and Singly Linked List

- Main queue operations:
  - enqueue(e): inserts an element at the end of the queue
  - dequeue(): removes and returns the element at the front of the queue
  - front(): returns the element at the front without removing it
  - size(): returns the number of elements stored
  - isEmpty(): returns a Boolean value indicating if there are no elements in the queue

- Singly linked list Operations
  - insertFront(e): inserts an element on the front of the list
  - removeFront(): returns and removes the element at the front of the list
  - insertBack(e): inserts an element on the back of the list
  - removeBack(): returns and removes the element at the end of the list

Solution Provided

Queue and Singly Linked List

- Main queue operations:
  - enqueue(e): inserts an element at the end of the queue
  - dequeue(): removes and returns the element at the front of the queue
  - front(): returns the element at the front without removing it
  - size(): returns the number of elements stored
  - isEmpty(): returns a Boolean value indicating if there are no elements in the queue
- Singly linked list Operations
  - insertFront(e): inserts an element on the front of the list
  - removeFront(): returns and removes the element at the front of the list
  - insertBack(e): inserts an element on the back of the list
  - removeBack(): returns and removes the element at the end of the list

Solution Provided

Queue and Singly Linked List

- Main queue operations:
  - enqueue(e): inserts an element at the end of the queue
  - dequeue(): removes and returns the element at the front of the queue
  - front(): returns the element at the front without removing it
  - size(): returns the number of elements stored
  - isEmpty(): returns a Boolean value indicating if there are no elements in the queue

- Singly linked list Operations
  - insertFront(): inserts an element on the front of the list
  - removeFront(): returns and removes the element at the front of the list
  - insertBack(e): inserts an element on the back of the list
  - removeBack(): returns and removes the element at the end of the list

Solution Provided

Queue and Singly Linked List

- Main queue operations:
  - enqueue(e): inserts an element at the end of the queue
  - dequeue(): removes and returns the element at the front of the queue
  - front(): returns the element at the front without removing it
  - size(): returns the number of elements stored
  - isEmpty(): returns a Boolean value indicating if there are no elements in the queue

- Singly linked list Operations
  - insertFront(): inserts an element on the front of the list
  - removeFront(): returns and removes the element at the front of the list
  - insertBack(e): inserts an element on the back of the list
  - removeBack(): returns and removes the element at the end of the list

Solution Provided

Queue and Singly Linked List

- Main queue operations:
  - enqueue(e): inserts an element at the end of the queue
  - dequeue(): removes and returns the element at the front of the queue
  - front(): returns the element at the front without removing it
  - size(): returns the number of elements stored
  - isEmpty(): returns a Boolean value indicating if there are no elements in the queue
- Singly linked list Operations
  - removeFront(): returns and removes the element at the front of the list
  - insertBack(e): inserts an element on the back of the list

front() would require a minor alteration or addition to LinkedList very similar to removeFront()

Solution Provided

Queue and Singly Linked List

- Main queue operations:
  - enqueue(e): inserts an element at the end of the queue
  - dequeue(): removes and returns the element at the front of the queue
  - front(): returns the element at the front without removing it
  - size(): returns the number of elements stored
  - isEmpty(): returns a Boolean value indicating if there are no elements in the queue

- Singly linked list Operations
  - removeFront(): returns and removes the element at the front of the list
  - insertBack(e): inserts an element on the back of the list

size() and isEmpty() would require the addition of a counter that increments each time enqueue() is called and decrements when dequeue() is called

Solution Provided

Queue as a Singly Linked List

- CONCLUSION:
  - We can implement a queue with a singly linked list
    - The front element is stored at the head of the list
    - The rear element is stored at the tail of the list
  - The space used is $O(n)$
  - Each operation of the Queue ADT takes $O(1)$ time
    - enqueue, dequeue, front, size, isEmpty each take $O(1)$ time
  - NOTE: we do not have the limitation of the static array based implementation on the size of the stack because the size of the linked list is not fixed,
    - i.e. the queue is NEVER full.