Data Structures and Algorithms

Double Ended Queue

Dr. Lina Xu

lina.xu@ucd.ie

School of Computer Science,
University College Dublin

November 12, 2018

Table of Contents

1 The Double-Ended Queue Abstract Data Type

2 Link Based Double-Ended Queue Implementation

Double-Ended Queue

- A double-ended queue is a data structure that supports insertion and deletion at both the front and the rear of the queue.
- Usually called deque
- This is pronounced "deck"

The Double-Ended Queue Abstract Data Type

Concept

The double-ended queue is slightly more complicated than the queue

- A double-ended queue is a container of objects or values
- Insertion and removal based on the the combination of first-in-first-out (FIFO) and last-in-first-out (LIFO)
  - It is like the combination of a Stack and a Queue

The Double-Ended Queue Abstract Data Type

Functional Specification

Operations:

- addFirst(o): Inserts object o onto the front of queue
- addLast(o): Inserts object o onto the back of queue
- removeFirst(): Returns and removes the element in the front of the queue
- removeLast() : Returns and removes the element in the back of the queue
- getFirst(): Returns the object in the front of the queue without removing it
- getLast(): Returns the object in the back of the queue without removing it
- size(): returns the number of elements stored in the queue
- isEmpty(): is the queue empty?

The Double-Ended Queue Abstract Data Type

Java Interface

```
1  public interface Deque {
2    public void addFirst(Object o);
3    public void addLast(Object o);
4    public Object removeFirst() throws
      EmptyDequeException;
5    public Object removeLast() throws
      EmptyDequeException;
6    public Object getFirst() throws
      EmptyDequeException;
7    public Object getLast() throws
      EmptyDequeException;
8    public int size();
9    public boolean isEmpty();
10 }
```

Exceptions for Empty Queue

- If we try and remove or access an element in an empty deque, the following exception should be thrown
- The exception is unchecked because it extends RuntimeException

```
1 public class EmptyDequeException extends
     RuntimeException {
2
3 }
```

The Double-Ended Queue Abstract Data Type

Implementation Strategies

There are two typical implementations of the double-ended queue abstract data type

We are only going to study the link based implementation

- Link based implementations
  - Elements are stored in custom objects called nodes
  - Object references are used to keep track of the order of the items
  - Infinite capacity - Can grow and shrink as more items are added and removed
  - Insertion and removal is always constant time

Table of Contents

1 The Double-Ended Queue Abstract Data Type

2 Link Based Double-Ended Queue Implementation

Link Based Double-Ended Queue Implementation

- Node class similar to doubly linked list

```
1  public class Node implements Position {
2    private Object element;
3    Node next;
4    Node previous;
5
6    public Node(int e) {
7      this.element = e;
8    }
9
10   public Object element() {
11     return element;
12   }
13 }
```

Link Based Double-Ended Queue Implementation

- Create a class called `LinkDeque`
  - Keep reference to the first and last nodes
  - References updated when necessary
    - `private Node first;`
    - `private Node last;`
  - Keep track of number of elements
    - `private int size;`

Link Based Double-Ended Queue Implementation

addFirst(o)

- Insert the value into the front of the queue
  - Create a new Node n containing the object
  - Change the previous reference of first to n
  - Change the next reference of n to first
  - Change the first reference so it points to n
  - Increment the size
- Special Cases:
  - When the deque is empty

Link Based Double-Ended Queue Implementation

addFirst(o)

```
1  Algorithm addFirst(o):
2    Input: An object o to be stored at the
      front of the queue
3    Output: None
4
5  n ← new Node(o)
6  if isEmpty() then
7    first ← n
8    last ← n
9  else
10   first.previous ← n
11   n.next ← first
12   first ← n
13
14 size ← size + 1
```

Link Based Double-Ended Queue Implementation

addLast(o)

- Insert the value into the rear of the queue
  - Create a new Node n containing the object
  - Change the next reference of last to n
  - Change the prev reference of n to last
  - Change the last reference so it points to n
  - Increment the size
- Special Cases:
  - When the deque is empty

Link Based Double-Ended Queue Implementation

addLast(o)

```
1  Algorithm addLast(o):
2    Input: An object o to be stored at the rear
      of the queue
3    Output: None
4
5  n ← new Node(o)
6  if isEmpty() then
7    last ← n
8    first ← n
9  else
10   last.next ← n
11   n.previous ← last
12   last ← n
13
14 size ← size + 1
```

Link Based Double-Ended Queue Implementation

removeFirst()

- Remove the next object from the front of the queue
  - Copy element reference from first as o
  - Change the first reference to the next reference of first
  - Change the previous reference of first to null
  - Decrement the size
  - Return o
- Special Cases:
  - When the deque is empty
  - When the element is the last in the deque

Link Based Double-Ended Queue Implementation

removeFirst()

```
1  Algorithm removeFirst():
2    Input: None
3    Output: The object that was in the front of
      the queue
4
5  if isEmpty then
6    throw empty deque exception
7  o ← first.element
8  first ← first.next
9  if size = 1
10   first = null
11   last = null
12 else
13   first.previous ← null
14 size ← size - 1
15 return o
```

Link Based Double-Ended Queue Implementation

removeLast()

- Remove the next object from the back of the queue
  - Copy element reference from last as o
  - Change the last reference to the previous reference of last
  - Change the next reference of last to null
  - Decrement the size
  - Return o
- Special Cases:
  - When the deque is empty
  - When the element is the last in the deque

Link Based Double-Ended Queue Implementation

removeLast()

```
1  Algorithm removeLast():
2    Input: None
3    Output: The object that was in the rear of
      the queue
4
5  if isEmpty then
6    throw empty deque exception
7  o ← last.element
8  last ← last.previous
9  if size = 1
10   first = null
11   last = null
12 else
13   last.next ← null
14 size ← size - 1
15 return o
```

Further Information and Review

If you wish to review the materials covered in this lecture or get further information, read the following sections in Data Structures and Algorithms textbook.

- 5.3 - Double-Ended Queues