|  |
| --- |
| 0hnQGAuNPo4wojnr77c2x8cWiaT70FGHBZ2h-3__FYQ.png |
| Queue |
| Data Structures Made Easy |
|  |

Dublin city university

***Contents***

[1. *Definition* 3](#_Toc350449914)

[2. *Implementation* 3](#_Toc350449915)

[3. *Example* 3](#_Toc350449916)

[4. *Functions* 4](#_Toc350449917)

[5. *Bounded Queue Pseudocode* 4](#_Toc350449918)

[6. *Unbounded Queue Pseudocode* 5](#_Toc350449919)

[6. *Complexity* 5](#_Toc350449920)

[7. *Advantages of Queues* 6](#_Toc350449921)

[8. *Disadvantages of Queues* 6](#_Toc350449922)

[9. *References* 6](#_Toc350449923)

# 1. *Definition*

A queue is a non-linear data structure that implements the addition of new elements and the deletion of existing elements within the queue. Deletion of an element involves removing the elements from the front of the queue. A queue inherits the FIFO (First In First Out) order.

There are two types of queues:

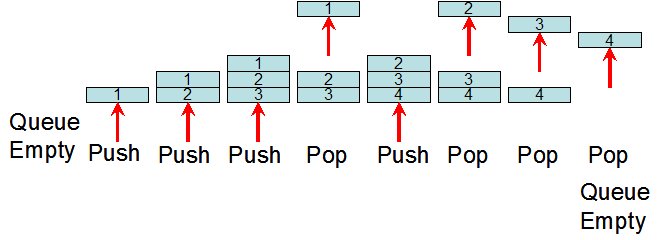
* **Bounded Queue**: A queue that has a maximum size implemented upon it. It contains a sequence of objects that increments and decrements at one end of the queue only.
* **Unbounded Queue:** Similar to a bounded queue, but there is no bound on the queue size.

# 2. *Implementation*

A queue works as follows:

1. Specify the type of data that is desired to be stored.
2. Insert the elements into the queue.
3. Retrieve the element at the front of the queue. That element was originally inserted first into the queue before other elements.

# 3. *Example*



# 4. *Functions*

The implementation of a queue involves the following functions

* **Queue<Object>()**

This constructor creates an empty queue that holds the object type, which is specified upon the creation of the queue.

* **enq(Object obj)**

This enqueue function inserts an object at the back of the queue.

* **deq()**

This dequeue function is used to remove the object from the front of the queue.

# 5. *Bounded Queue Pseudocode*

function enq ( T t ){

if size of array >= length of sequence\_1

sequence\_2 = new sequence [ length of sequence\_1 \* 2 ]

copy sequences

sequence\_1 = sequence\_2

sequence [ tail ] = t

tail = tail + 1 mod length of sequence

increment size

}

function deq (){

if size is 0

return

else

temporary = sequence [ head ]

head = head + 1 mod length of sequence

decrement size

return temp

}

# 6. *Unbounded Queue Pseudocode*

class Node < T >{

T item

Node null

constructor Node

}

function enq ( T t ){

new Node

if tail != null

tail.next = node

else

head = node

tail = node

}

function deq (){

if size is 0

return

T = head.item

head = head.Node

return

}

# 6. *Complexity*

The only considerations to be made in relation to the complexity of a queue are the operations it performs. The operations of enqueue and dequeue possess a time complexity of O(1) on average.

# 7. *Advantages of Queues*

A queue is considered to be a data structure that is easy to implement. A strong advantage of using a queue is that they are used to prioritise operations in operating systems, such as categorizing data for various simulations.

# 8. *Disadvantages of Queues*

A queue is considered to be of minimal use when compared to other optimal data structures. Based on the structure of queues, they suffer from slow access to other items, which can increase the computational time of the application.

# 9. *References*

Morris J. 1998. *Queues* [Online]. Available from: <http://www.cs.auckland.ac.nz/~jmor159/PLDS210/queues.html>

Robert C. 2013. *The Advantages of a Queue in Data Structure* [Online]. Available from:

<http://www.ehow.com/info_8763624_advantages-queue-data-structure.html>