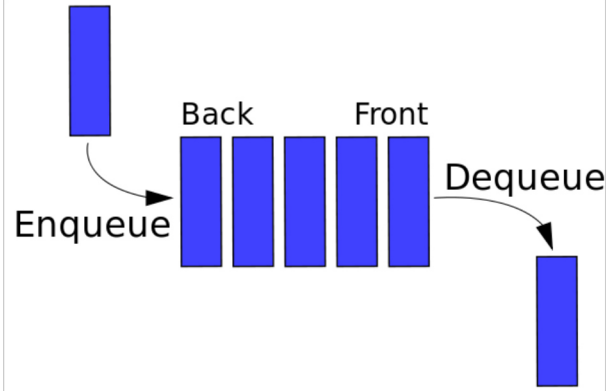
Queue (abstract data type)

From Wikipedia, the free encyclopedia



Representation of a FIFO (first in, first out) queue

In computer science, a **queue** (/ˈkjuː/ *KEW*) is a particular kind of abstract data type or collection in which the entities in the collection are kept in order and the principal (or only) operations on the collection are the addition of entities to the rear terminal position, known as *enqueue,* and removal of entities from the front terminal position, known as *dequeue*. This makes the queue a First-In-First-Out (FIFO) data structure. In a FIFO data structure, the first element added to the queue will be the first one to be removed. This is equivalent to the requirement that once a new element is added, all elements that were added before have to be removed before the new element can be removed. Often a *peek* or *front* operation is also entered, returning the value of the front element without dequeuing it. A queue is an example of a linear data structure, or more abstractly a sequential collection.

Queues provide services in computer science, transport, and operations research where various entities such as data, objects, persons, or events are stored and held to be processed later. In these contexts, the queue performs the function of a buffer.

Queues are common in computer programs, where they are implemented as data structures coupled with access routines, as an abstract data structure or in object-oriented languages as classes. Common implementations are circular buffers and linked lists.

Contents

Queue implementation

Theoretically, one characteristic of a queue is that it does not have a specific capacity. Regardless of how many elements are already contained, a new element can always be added. It can also be empty, at which point removing an element will be impossible until a new element has been added again.

Fixed length arrays are limited in capacity, but it is not true that items need to be copied towards the head of the queue. The simple trick of turning the array into a closed circle and letting the head and tail drift around endlessly in that circle makes it unnecessary to ever move items stored in the array. If n is the size of the array, then computing indices modulo n will turn the array into a circle. This is still the conceptually simplest way to construct a queue in a high level language, but it does admittedly slow things down a little, because the array indices must be compared to zero and the array size, which is comparable to the time

taken to check whether an array index is out of bounds, which some languages do, but this will certainly be the method of choice for a quick and dirty implementation, or for any high level language that does not have pointer syntax. The array size must be declared ahead of time, but some implementations simply double the declared array size when overflow occurs. Most modern languages with objects or pointers can implement or come with libraries for dynamic lists. Such data structures may have not specified fixed capacity limit besides memory constraints. Queue *overflow* results from trying to add an element onto a full queue and queue *underflow* happens when trying to remove an element from an empty queue.

A *bounded queue* is a queue limited to a fixed number of items.[1]

There are several efficient implementations of FIFO queues. An efficient implementation is one that can perform the operations—enqueuing and dequeuing—in O(1) time.

- Linked list
  - A doubly linked list has O(1) insertion and deletion at both ends, so is a natural choice for queues.
  - A regular singly linked list only has efficient insertion and deletion at one end. However, a small modification—keeping a pointer to the *last* node in addition to the first one—will enable it to implement an efficient queue.
- A deque implemented using a modified dynamic array

Queues and programming languages

Queues may be implemented as a separate data type, or may be considered a special case of a double-ended queue (deque) and not implemented separately. For example, Perl and Ruby allow pushing and popping an array from both ends, so one can use **push** and **shift** functions to enqueue and dequeue a list (or, in reverse, one can use **unshift** and **pop**), although in some cases these operations are not efficient.

C++'s Standard Template Library provides a "`queue`" templated class which is restricted to only push/pop operations. Since J2SE5.0, Java's library contains a `Queue (https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html)` interface that specifies queue operations; implementing classes include `LinkedList (https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html)` and (since J2SE 1.6) `ArrayDeque (https://docs.oracle.com/javase/8/docs/api/java/util/ArrayDeque.html)`. PHP has an SplQueue (http://www.php.net/manual/en/class.splqueue.php) class and third party libraries like beanstalk'd and Gearman.

Examples

A simple queue implemented in Ruby:

```ruby
class Queue
  def initialize
    @list = Array.new
  end

  def enqueue(element)
    @list << element
  end

  def dequeue
    @list.shift
  end
end
```

See also

- Circular buffer
- Deque
- Priority queue
- Queueing theory
- Stack – the "opposite" of a queue: LIFO (Last In First Out)

References

1. "Queue (Java Platform SE 7)". Docs.oracle.com. 2014-03-26. Retrieved 2014-05-22.

- Donald Knuth. *The Art of Computer Programming*, Volume 1: *Fundamental Algorithms*, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89683-4. Section 2.2.1: Stacks, Queues, and Deques, pp. 238–243.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 10.1: Stacks and queues, pp. 200–204.
- William Ford, William Topp. *Data Structures with C++ and STL*, Second Edition. Prentice Hall, 2002. ISBN 0-13-085850-1. Chapter 8: Queues and Priority Queues, pp. 386–390.
- Adam Drozdek. *Data Structures and Algorithms in C++*, Third Edition. Thomson Course Technology, 2005. ISBN 0-534-49182-0. Chapter 4: Stacks and Queues, pp. 137–169.

External links

- Queue Data Structure and Algorithm (http://www.studytonight.com/data-structures/queue-data-structure)
- Queues with algo and 'c' programme (http://scanftree.com/Data_Structure/Queues)
- STL Quick Reference (http://www.halpernwightsoftware.com/stdlib-scratch/quickref.html#containers14)
- VBScript implementation of stack, queue, deque, and Red-Black Tree (http://www.ludvikjerabek.com/downloads.html)

Wikimedia Commons has media related to ***Queue data structure***.

Black, Paul E. "Bounded queue". *Dictionary of Algorithms and Data Structures*. NIST.

Categories: Abstract data types
