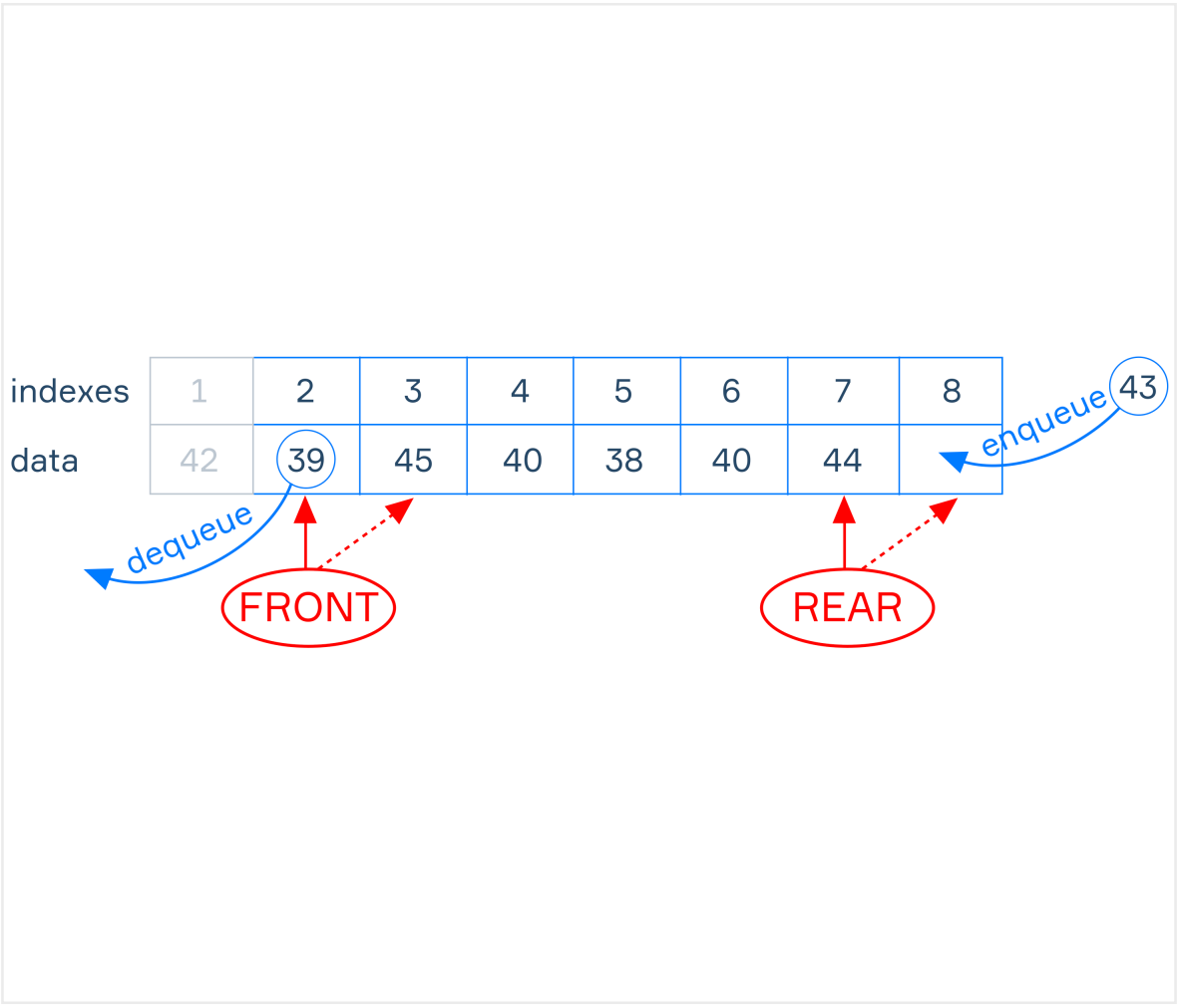
Queue

The first person in the queue gets out first – it is an illustration of the programming principle called **FIFO** (**F**irst **I**n, **F**irst **O**ut), and the data structure that works this way is called a **queue**.

The operation when you insert an element into the queue is called **enqueue**, and the element is added by the REAR pointer. If you remove an element, this operation is called **dequeue**, and the element that was pointed at by FRONT comes out of the queue.



- The enqueue operation inserts an element at the end of the queue and moves the REAR pointer;
- The dequeue operation removes an element from the beginning of the queue, and the FRONT pointer moves.

Implementation of queue

Array

This topic has analysed the simplest implementation of a queue – in an array. In such an implementation, FRONT and REAR can simply be numeric variables that store the indices of the corresponding elements. Then you can access both the first element and the last one in a constant time: the enqueue and dequeue operations will take O(1). However, there is a problem with this implementation: adding and removing elements, you move through the array, which means you need an array of an infinite length.

Linked list

The queue can also be implemented in a linked list, in which each element stores a link to the next one. FRONT and REAR are then pointers to the first and last nodes in the list, which again allows you to do insertion and deletion of elements in O(1). But storing a link to the next item in each item means wasting a lot more memory than the data requires.

Bonus: where is the queue used?

The simple answer is everywhere. If you've ever printed a bunch of documents at once, you know that the printer has its own queue to print – this is our data structure. If you've heard of the breadth-first search algorithm, it too uses a queue (if not, it's just one of the most important algorithms in the world, you'll learn it soon).