Queue

Contents

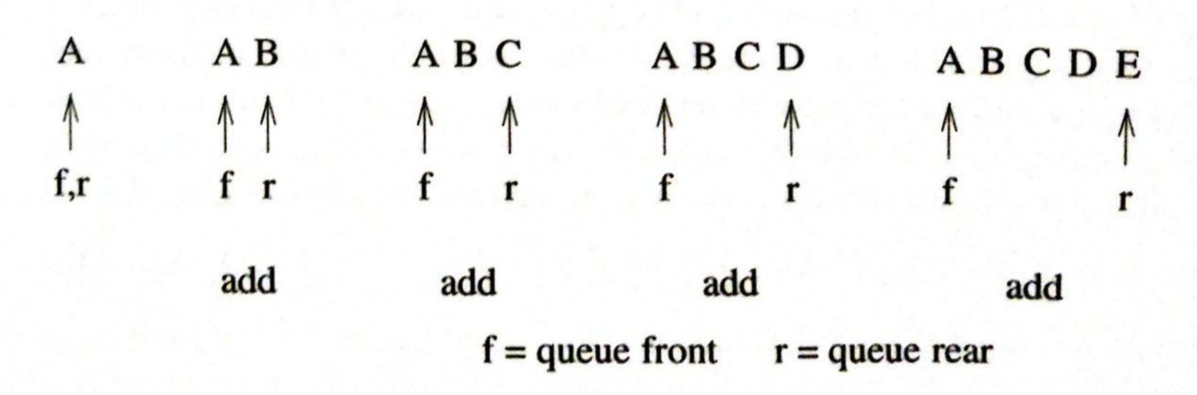
- Definition
- Operations
- Applications
- Queue ADT
- Implementation (Static)
- Circular Queue
- Priority Queue

Queue

- An ordered list in which insertions and deletions take place at different ends
  - Insertions at “rear” end
  - Deletions at “front” end
- First-In-First-Out (FIFO) list
  - The first one inserted is the first one to be removed

Queue

- An ordered list in which insertions and deletions take place at different ends
  - Insertions at “rear” end
  - Deletions at “front” end
- First-In-First-Out (FIFO) list
  - The first one inserted is the first one to be removed



Queue operations

Main queue operations

- **enqueue**: adds a new element at the end of the queue
- **dequeue**: removes and returns the element at the front of the queue

Auxiliary queue operations

- **front**: returns the element at the front without removing it
- **rear**: returns the element at the rear without removing it
- **isEmpty**: indicates whether the queue is empty
- **isFull**: indicates whether there is enough space for a new element

Applications

Direct applications

- Waiting lists
- Job scheduling
- Access to shared resources such as printers etc.

Indirect applications

- Auxiliary data structures for algorithms
- Components of other data structures

Queue ADT

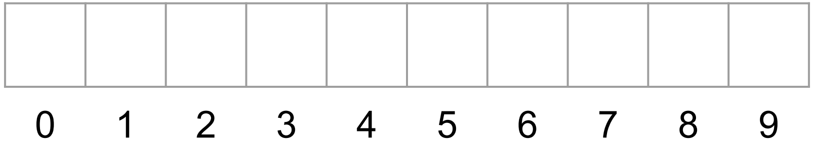
```
ADT Queue is
  Objects: a finite ordered list with zero or more elements
  Functions:
      For all queue ∈ Queue, item ∈ element, maxQueueSize ∈ positive integer
      Queue CreateQ(maxQueueSize) :=
            Create an empty queue whose maximum size is maxQueueSize
      Boolean isFull(queue, maxQueueSize) :=
            if (number of elements in queue == maxQueueSize) return TRUE
            else return FALSE
      Queue Enqueue(queue, item) :=
            if (IsFull(queue)) queueFull
            else insert item at rear of queue and return queue
      Boolean isEmpty(queue) :=
            if (queue == CreateQ(maxQueueSize)) return TRUE
            else return FALSE
      Element Dequeue(queue) :=
            if (IsEmpty(queue)) return
            else remove and return the item at front of queue
```

Static Implementation of a Queue (using arrays)

- The Queue ADT can be implemented using an array.
- To create a queue, we initialize an array of maxQueueSize.

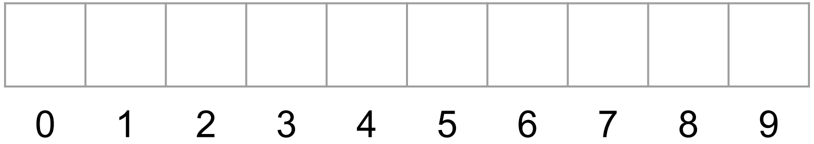
Static Implementation of a Queue (using arrays)

- The Queue ADT can be implemented using an array.
- To create a queue, we initialize an array of maxQueueSize.

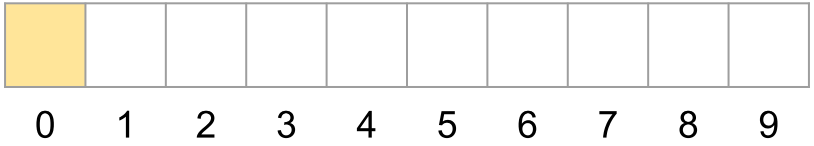


Static Implementation of a Queue (using arrays)

- The Queue ADT can be implemented using an array.
- To create a queue, we initialize an array of maxQueueSize.
- We add elements from left to right (i.e. starting from index 0).



After an enqueue operation



Static Implementation of a Queue (using arrays)

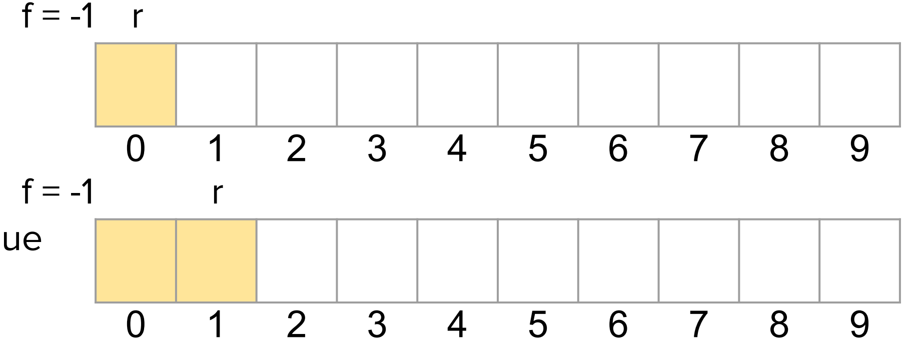
- The Queue ADT can be implemented using an array.
- To create a queue, we initialize an array of maxQueueSize.
- We add elements from left to right (i.e. starting from index 0).
- We use two variables, f and r, to keep track of the index of the front element and that of the rear element.

  Let f and r be - 1 in the beginning. During enqueue, r is increased by 1, and during dequeue, f is increased by 1.

Static Implementation of a Queue (using arrays)

- The Queue ADT can be implemented using an array.
- To create a queue, we initialize an array of maxQueueSize.
- We add elements from left to right (i.e. starting from index 0).
- We use two variables, f and r, to keep track of the index of the front element and that of the rear element.

  Let f and r be - 1 in the beginning. During enqueue, r is increased by 1, and during dequeue, f is increased by 1.

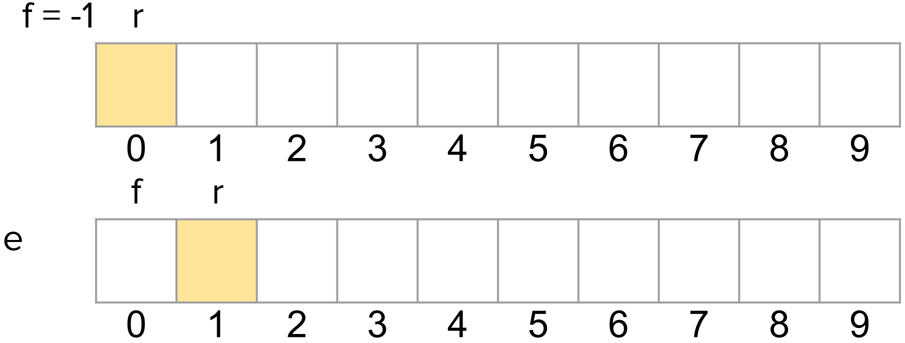


After an enqueue operation

Static Implementation of a Queue (using arrays)

- The Queue ADT can be implemented using an array.
- To create a queue, we initialize an array of maxQueueSize.
- We add elements from left to right (i.e. starting from index 0).
- We use two variables, f and r, to keep track of the index of the front element and that of the rear element.

  Let f and r be - 1 in the beginning. During enqueue, r is increased by 1, and during dequeue, f is increased by 1.



After a dequeue operation

Static Implementation of a Queue (using arrays)

Algorithm: createQueue(maxQueueSize)

Steps:

1. Initialize an array of size maxQueueSize
2. Initialize front and rear to -1

Static Implementation of a Queue (using arrays)

Algorithm: isEmpty()

Steps:

1. If rear == front, return true
2. Else return false

Algorithm: isFull()

Steps:

1. If rear == maxQueueSize - 1, return true
2. Else return false

Static Implementation of a Queue (using arrays)

Algorithm: enqueue(value)
Steps:

1. If queue is not full, increase rear by 1 and store value at index rear of the array
2. Else print Queue overflow message.

Algorithm: dequeue()
Steps:

1. If the queue is not empty, increase front by 1 and return the value at index front of the array.
2. Else print Queue underflow message.

Static Implementation of a Queue (using arrays)

Problem with this implementation

- The queue gradually shifts to the right. Example: suppose maxQueueSize = 4, and operations performed are enqueue(2), enqueue(5), dequeue(), enqueue(4), dequeue(), enqueue(3).

	0	1	2	3	f	r
					-1	-1
enqueue(2)	2				-1	0
enqueue(5)	2	5			-1	1
dequeue()		5			0	1
enqueue(4)		5	4		0	2
dequeue()			4		1	2
enqueue(3)			4	3	1	3
enqueue(7)?						

Static Implementation of a Queue (using arrays)

Problem with this implementation

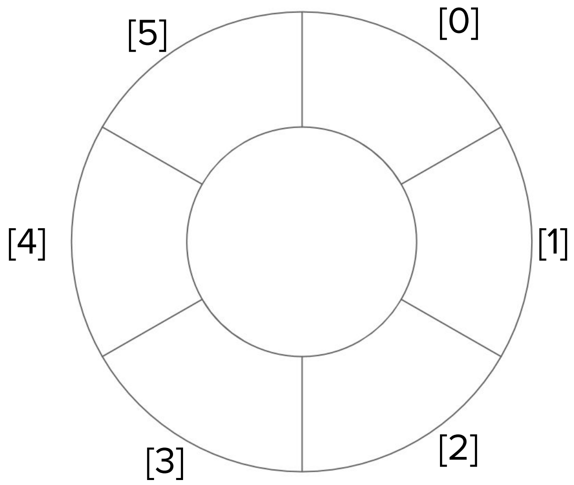
- The queue gradually shifts to the right.

Solution

1. On queue overflow, move all elements to the left so that the queue front is always at 0. However, it is very time consuming.
2. Use circular queue

Circular Queue

- Aka circular buffer, ring buffer
- A FIFO list
- The last position is connected back to the first position to make a circle.
- The position next to maxQueueSize - 1 is 0
- The position that precedes 0 is maxQueueSize - 1
- When the queue rear is at maxQueueSize -1 , the Next element is put into position 0.



Implementation of a circular queue with n-1 space used

Algorithm: createQueue(maxQueueSize)

Steps:

1. Initialize an array of size maxQueueSize
2. Initialize front and rear to 0

Implementation of a circular queue with n-1 space used

Algorithm: isEmpty()

Steps:

1. If rear == front, return true
2. Else return false

Algorithm: isFull()

Steps:

1. If front == (rear + 1) % maxQueueSize, return true
2. Else return false

Implementation of a circular queue with n-1 space used

Algorithm: enqueue(value)

Steps:

```
1. If queue is not full
    a. rear = (rear + 1) % maxQueueSize
    b. data[rear] = value
2. Else
    a. print Queue overflow message.
3. Endif
```

Implementation of a circular queue with n-1 space used

Algorithm: dequeue()

Steps:

```
1. If the queue is not empty
    a. front = (front + 1) % maxQueueSize
    b. return queue[front]
2. Else
    a. print Queue underflow message
3. Endif
```

Priority Queue

- The element to be deleted is the one with the highest or lowest priority
- At any time, an element with arbitrary priority can be inserted into the queue.

Ascending queue

- Only the smallest item can be removed.

Descending queue

- Only the largest item can be removed.