

Indian Institute of Technology Madras
ZANZIBAR CAMPUS

Z5007 · PROGRAMMING AND DATA STRUCTURES

Implementing Queues: Circular Arrays, Linked Lists, Deques and Heaps

Lecture 12 · Wednesday 28 October 2026 · Canning, Broder & Lafore Ch 4 (Queues)

Dr. Innocent Nyalala | MTech Semester 1 | IIT Madras Zanzibar

Today's Plan

IIT Madras Zanzibar

1. (0–4 min) The queue and its promise: O(1) at both ends
2. (4–24 min) Circular array: modulo; empty or full? where is tail? three conventions; trace; code
3. (24–32 min) Linked list: nodes, head and tail pointers, every step drawn, the code
4. (32–37 min) Deque on a circular array; priority queue on a binary heap
5. (37–42 min) Live demo with animation; code-recall questions
6. (42–50 min) Questions and answers; practice problems

Learning Outcomes

IIT Madras Zanzibar

By the end of this session, you will be able to:

Explain	why a queue on a plain array wastes space or time, and how a circular array fixes it
Use	head = (head + 1) % capacity and tail = (tail + 1) % capacity, and say why the % is needed
Trace	enqueue and dequeue on a circular array and on a linked list, step by step
Write	ArrayQueue and LinkedQueue from memory, every operation O(1)
Describe	how a deque and a heap-based priority queue are implemented, and their costs

Key Terms for Today

IIT Madras Zanzibar

Term	Meaning	Example
Queue (FIFO)	first in, first out: items leave in the order they arrived	a bank line
enqueue / dequeue	add at the rear / remove from the front	join / be served
head (front)	index or node of the item that leaves next	head = 2
tail (rear)	where the next item will be added	tail = 0
Capacity	the number of boxes in the array	5
Modulo a % m	the remainder after dividing a by m	5 % 5 = 0, 6 % 5 = 1
Circular array	an array whose last box is followed by box 0	… 3, 4, 0, 1 …
Node	a box holding a value and a reference to the next node	[10 | •] → [20 | •]

Implementing Queues

IIT Madras Zanzibar

→ **The Queue and Its Promise**

A Queue on a Circular Array

A Queue on a Linked List

Deque and Priority Queue

Live Demo and Practice

The Queue's Promise: Every Operation O(1)

IIT Madras Zanzibar

Operation	What it does	Required cost
enqueue(x)	add x at the rear	O(1)
dequeue()	remove and return the front item	O(1)
peek()	look at the front item	O(1)
is_empty(), len()	is it empty? how many items?	O(1)

1 **Bank line**

join at the back, served at the front

2 **Print jobs**

printed in the order sent

3 **Network packets**

a router forwards them in order

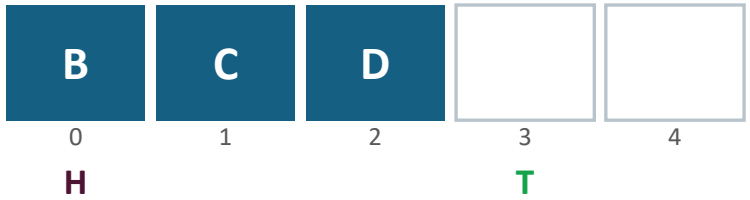
Definition: FIFO (first in, first out): the item that has waited longest leaves first.

? **Think:** with a Python list, enqueue = append and dequeue = pop(0). Which of the two breaks the O(1) promise, and why?

Two Naive Array Queues, and What Goes Wrong

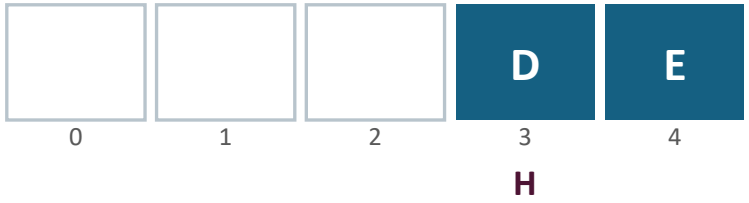
IIT Madras Zanzibar

Attempt 1: shift left on every dequeue: O(n)



dequeue A moved B, C, D one box left: 3 moves

Attempt 2: only move head right: space is wasted



tail has run past box 4: 'full', yet boxes 0, 1, 2 are free

The fix: after the last box, continue at box 0. The array becomes a ring: a circular array.

Implementing Queues

IIT Madras Zanzibar

The Queue and Its Promise

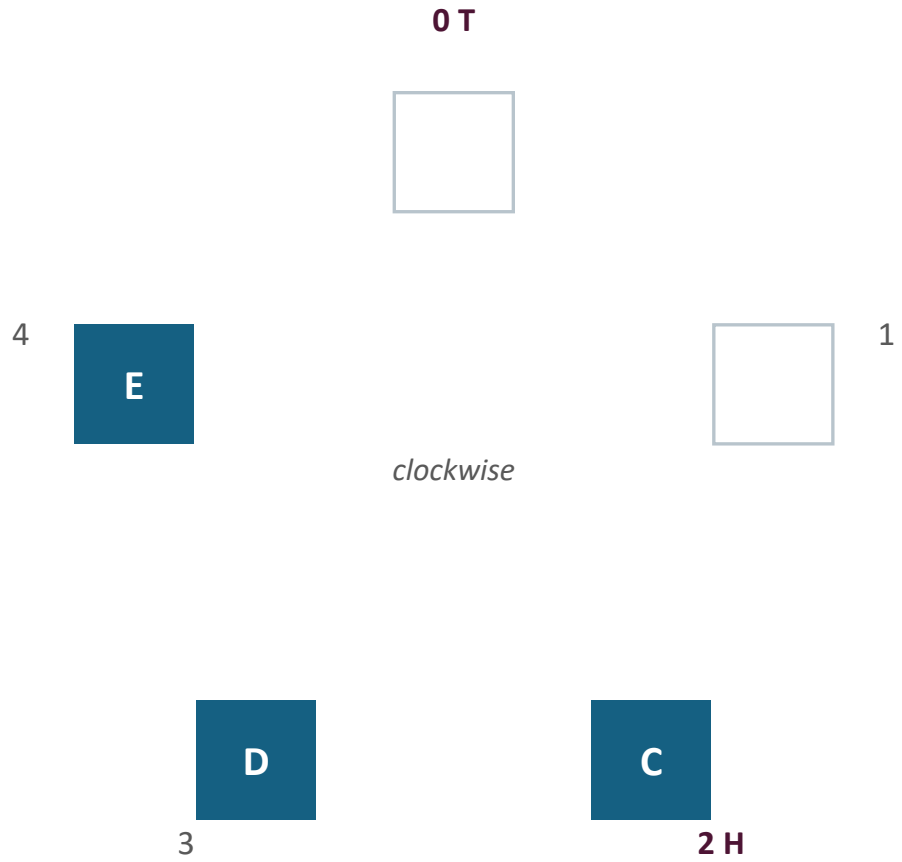
→ **A Queue on a Circular Array**

A Queue on a Linked List

Deque and Priority Queue

Live Demo and Practice

The Idea: Bend the Array into a Ring



How it works

1. The array is still 5 boxes in a row in memory; only the counting changes.
2. The box after index 4 is index 0: the array wraps round.
3. enqueue: write at tail, then tail moves one step clockwise.
4. dequeue: read at head, then head moves one step clockwise.
5. The queue is the run of boxes from head, clockwise, up to (not including) tail.

Definition: **Circular array:** an array used as if the last box were followed by box 0. The 'next' index is computed with % capacity.

Why % capacity? The Modulo Operator

IIT Madras Zanzibar

i	i + 1	(i + 1) % 5	meaning
0	1	1	next box is 1
1	2	2	next box is 2
2	3	3	next box is 3
3	4	4	next box is 4
4	5	0	past the end: back to 0

Read the table

1. a % m = the remainder of a ÷ m. Examples: 7 % 5 = 2, 5 % 5 = 0, 3 % 5 = 3.
2. For i < capacity − 1, (i + 1) % capacity = i + 1: nothing changes.
3. For the LAST box, (i + 1) % capacity = 0: it wraps to the start.
4. Clock: (11 + 1) % 12 = 0.

```
head = (head + 1) % self.cap     # front moves to the next box, wrapping
tail = (tail + 1) % self.cap     # rear moves to the next box, wrapping
```

Definition: Modulo (%): a % m is the remainder of a divided by m; it is always between 0 and m − 1, so it is always a valid index.

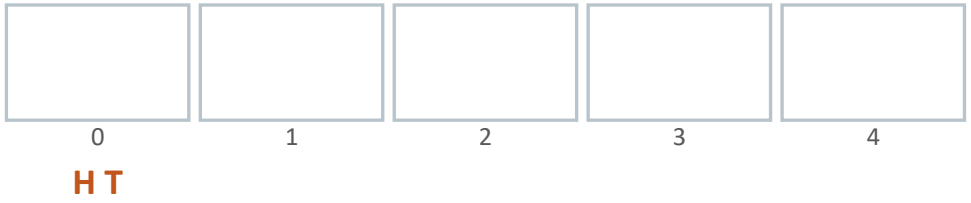
Three Variables: head, tail and count

IIT Madras Zanzibar

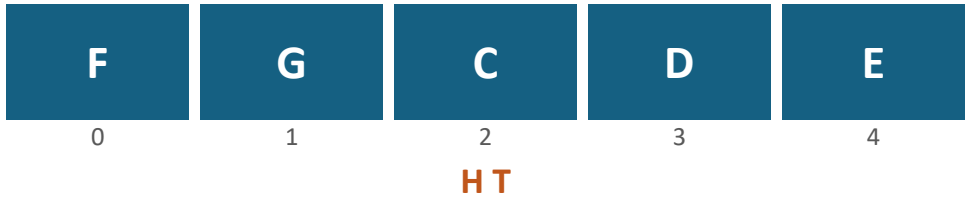
The three variables

1. head = index of the FRONT item (next to leave).
2. tail = index of the next FREE box (where the next item goes).
3. count = number of items stored.
4. Empty: count == 0. Full: count == capacity.

Empty: head = tail = 0, count = 0



Full: head = tail = 2, count = 5



Why count: head == tail in BOTH pictures. Only count tells empty (0) from full (capacity).

Definition: **Invariant:** the items are a[head], a[(head+1) % cap], …, count of them, in arrival order.

Implementing Queues

IIT Madras Zanzibar

The Queue and Its Promise

→ **A Queue on a Circular Array**

A Queue on a Linked List

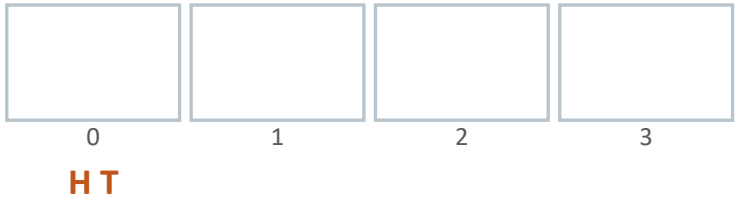
Deque and Priority Queue

Live Demo and Practice

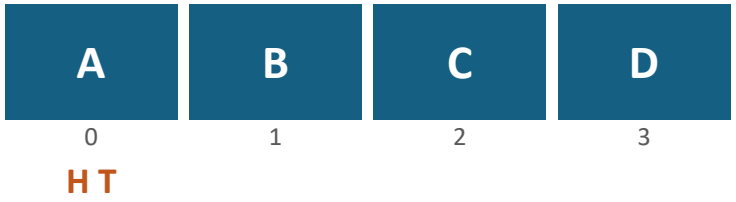
The Puzzle: head == tail, Empty or Full?

IIT Madras Zanzibar

Empty queue, capacity 4: head = 0, tail = 0



After enqueue A, B, C, D: tail = (3 + 1) % 4 = 0



What the two pictures show

1. Both pictures have head = 0 and tail = 0.
2. So 'head == tail' is true when EMPTY and when FULL.
3. The positions alone cannot tell the two apart.
4. We need one extra piece of information. There are three standard choices.

Three ways out: A. keep a count. B. start at −1 and let tail point at the last item. C. never fill the last box.

? **Think:** before the next slide: which single number would remove the confusion?

Three Ways to Start a Circular Queue

IIT Madras Zanzibar

	A: count (this course)	B: start at −1	C: one box left empty
Empty queue starts as	head = 0, tail = 0, count = 0	head = −1, tail = −1	head = 0, tail = 0
tail points to	the next FREE box	the LAST item	the next FREE box
is_empty()	count == 0	head == −1	head == tail
is_full()	count == capacity	(tail + 1) % cap == head	(tail + 1) % cap == head
Items that fit	capacity	capacity	capacity − 1
Special cases	none	first enqueue sets head = 0; last dequeue resets both to −1	none, but one box is wasted

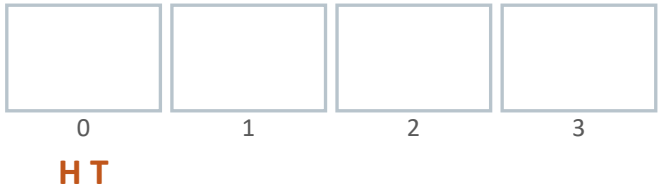
Rule: pick ONE convention and use its own empty and full tests. Mixing them (start at −1, test count) is the usual bug.

Where Is tail? Next Free Box or Last Item?

IIT Madras Zanzibar

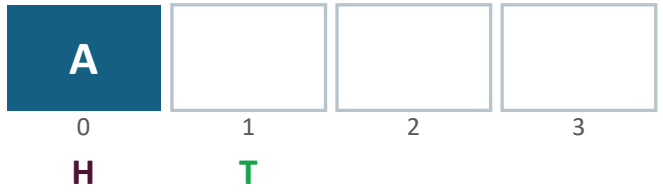
A (this course): tail = NEXT FREE box

empty



head 0, tail 0, count 0

after enqueue A

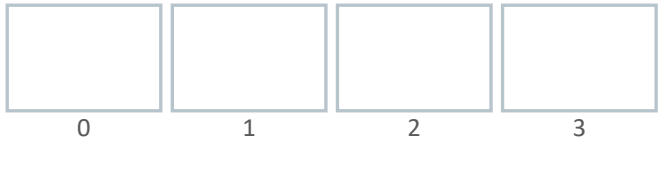


head 0, tail 1, count 1

```
self.a[self.tail] = x                    # write first
self.tail = (self.tail + 1) % self.cap   # then move
```

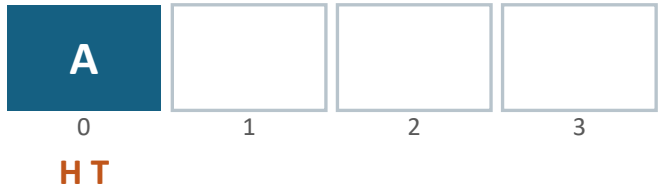
B: tail = LAST item

empty



head −1, tail −1

after enqueue A



head 0, tail 0

```
self.tail = (self.tail + 1) % self.cap   # move first
self.a[self.tail] = x                    # then write
```

Definition: head and tail (convention A): head = index of the FRONT item; tail = index of the next FREE box. With one item at box 0: head = 0, tail = 1.

In both conventions the first item goes into box 0 and head = 0. Only the meaning of tail differs.

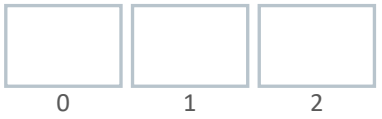
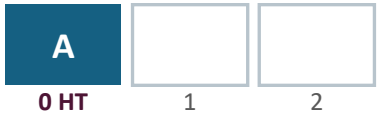
Convention A (Ours): head = tail = 0, count = 0

IIT Madras Zanzibar

start (empty)		head = 0, tail = 0, count = 0	**EMPTY**
enqueue A		head = 0, tail = 1, count = 1	
enqueue B		head = 0, tail = 2, count = 2	
enqueue C		head = 0, tail = 3, count = 3	
enqueue D		head = 0, tail = 0, count = 4	**FULL**
enqueue E: REFUSED (full)		head = 0, tail = 0, count = 4	**FULL**
dequeue → A		head = 1, tail = 0, count = 3	

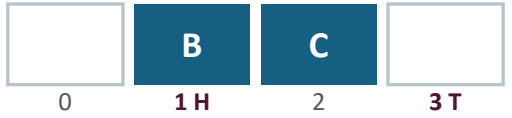
Convention B: head = tail = −1 Means Empty

IIT Madras Zanzibar

start (empty)		head = -1, tail = -1	**EMPTY**
enqueue A		head = 0, tail = 0	
enqueue B		head = 0, tail = 1	
enqueue C		head = 0, tail = 2	**FULL**
dequeue → A		head = 1, tail = 2	
dequeue → B		head = 2, tail = 2	
dequeue → C		head = -1, tail = -1	**EMPTY**

Convention C: Leave One Box Empty (No count)

IIT Madras Zanzibar

start (empty)		head = 0, tail = 0	**EMPTY**
enqueue A		head = 0, tail = 1	
enqueue B		head = 0, tail = 2	
enqueue C		head = 0, tail = 3	**FULL**
enqueue D: REFUSED (full)		head = 0, tail = 3	**FULL**
dequeue → A		head = 1, tail = 3	
dequeue → B		head = 2, tail = 3	

Same Picture, Different Meaning: Summary

IIT Madras Zanzibar

State	A: count	B: −1	C: one empty
head == tail and …	count 0 → empty; count = cap → full	… = −1 → empty; ≥ 0 → one item	always empty
Capacity 4 holds	4 items	4 items	3 items

What to remember

1. This course (lectures, lab, quiz): convention A, head = tail = 0 and a count.
2. In your own code, state your convention in a comment and use its own tests.
3. All three give O(1) enqueue and dequeue.

? **Think:** convention B, capacity 3: enqueue X, enqueue Y, dequeue, dequeue. Give head and tail after each step.

Implementing Queues

IIT Madras Zanzibar

The Queue and Its Promise

→ **A Queue on a Circular Array**

A Queue on a Linked List

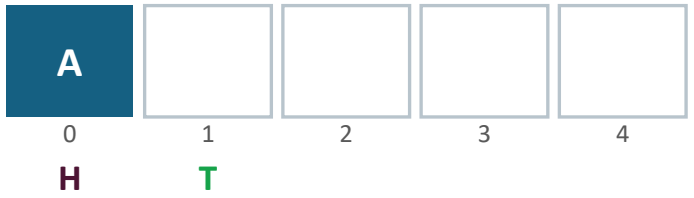
Deque and Priority Queue

Live Demo and Practice

Step by Step on Capacity 5 (1 of 3)

IIT Madras Zanzibar

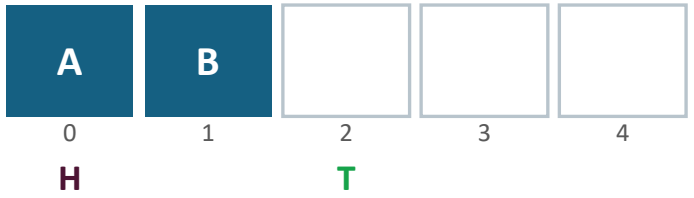
enqueue A



a[0] = A; tail = (0 + 1) % 5 = 1

head = 0, tail = 1, count = 1

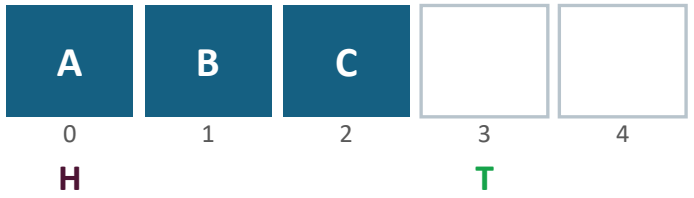
enqueue B



a[1] = B; tail = (1 + 1) % 5 = 2

head = 0, tail = 2, count = 2

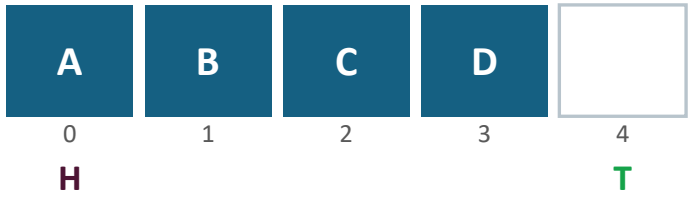
enqueue C



a[2] = C; tail = (2 + 1) % 5 = 3

head = 0, tail = 3, count = 3

enqueue D



a[3] = D; tail = (3 + 1) % 5 = 4

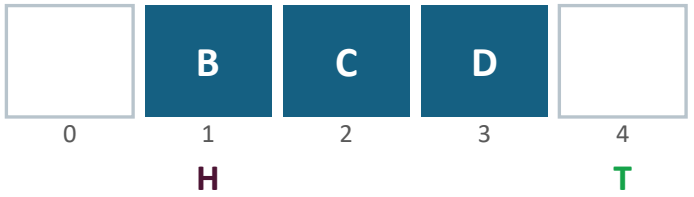
head = 0, tail = 4, count = 4

H = head (front) T = tail (next free box) teal = item white = free box

Step by Step on Capacity 5 (2 of 3)

IIT Madras Zanzibar

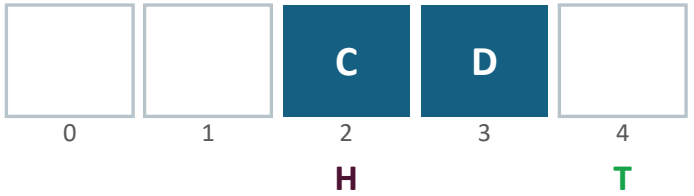
dequeue → A



return a[0] = A; head = (0 + 1) % 5 = 1

head = 1, tail = 4, count = 3

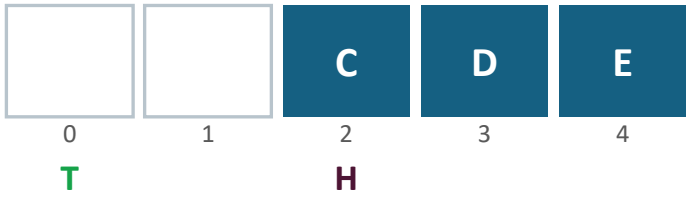
dequeue → B



return a[1] = B; head = (1 + 1) % 5 = 2

head = 2, tail = 4, count = 2

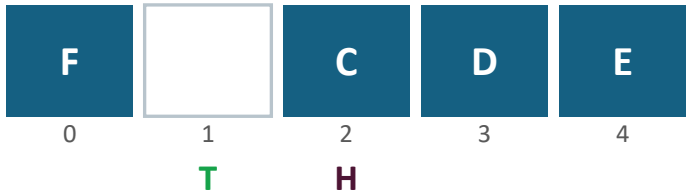
enqueue E



a[4] = E; tail = (4 + 1) % 5 = 0 ← wraps to 0!

head = 2, tail = 0, count = 3

enqueue F



a[0] = F; tail = (0 + 1) % 5 = 1

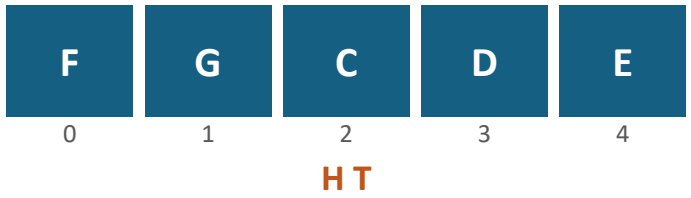
head = 2, tail = 1, count = 4

H = head (front) T = tail (next free box) teal = item white = free box

Step by Step on Capacity 5 (3 of 3)

IIT Madras Zanzibar

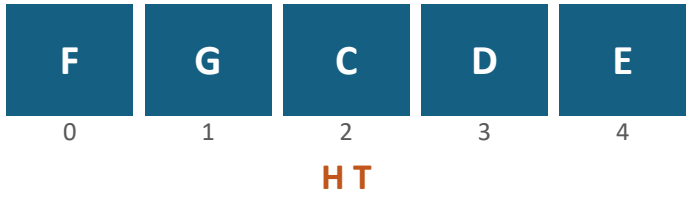
enqueue G



a[1] = G; tail = (1 + 1) % 5 = 2

head = 2, tail = 2, count = 5

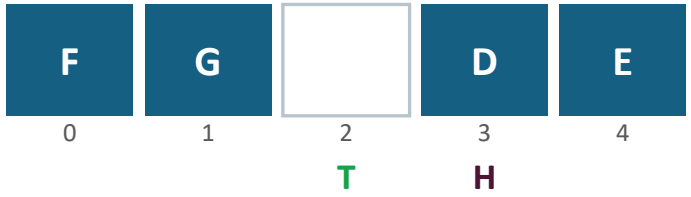
enqueue H



count == capacity: FULL. Raise OverflowError, change nothing.

head = 2, tail = 2, count = 5

dequeue → C

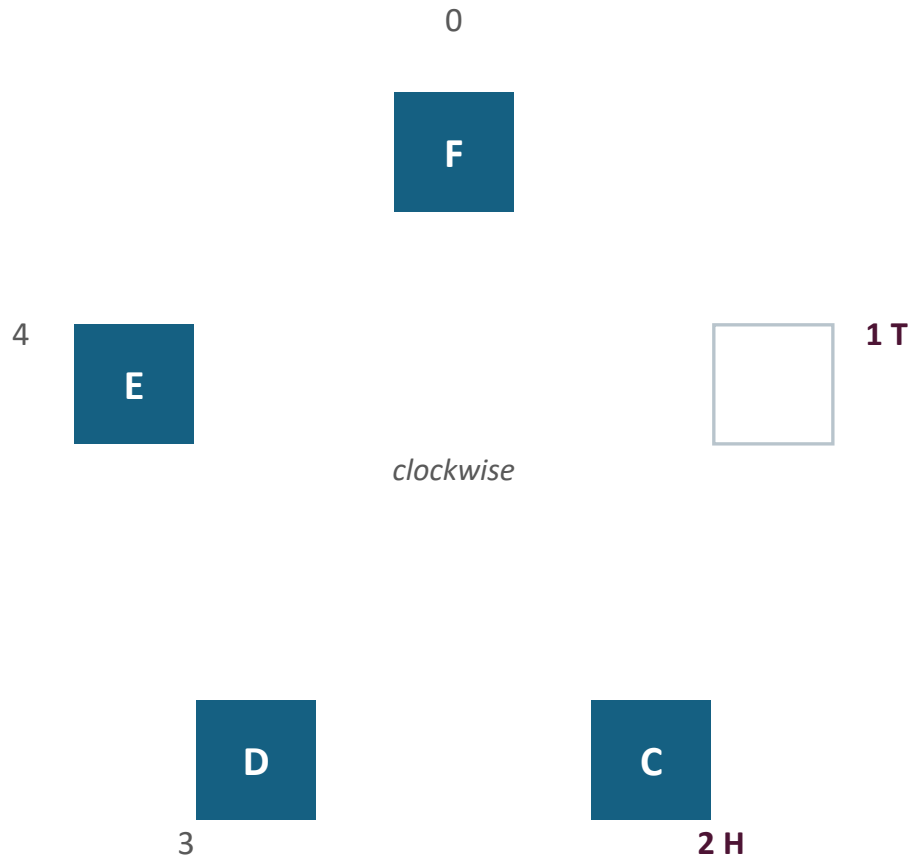


return a[2] = C; head = (2 + 1) % 5 = 3

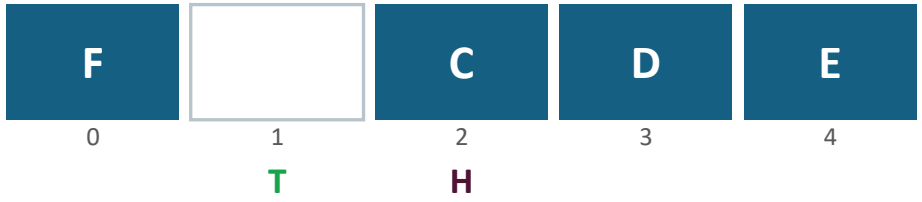
head = 3, tail = 2, count = 4

H = head (front) T = tail (next free box) teal = item white = free box

The Same State, Seen as a Ring



Row view (after enqueue F):



1. Row view: looks split into F | … | C D E.
2. Ring view: one run C → D → E → F, clockwise from H.
3. Front to rear order: C, D, E, F: arrival order.

ArrayQueue, Part 1: The Data and the Tests

```
 1  class ArrayQueue:
 2      def __init__(self, capacity):
 3          self.a = [None] * capacity
 4          self.cap = capacity
 5          self.head = 0
 6          self.tail = 0
 7          self.count = 0
 8
 9      def is_empty(self):
10          return self.count == 0
11
12      def is_full(self):
13          return self.count == self.cap
```

Line	What it does
3	a fixed block of capacity boxes
4	remember the capacity
5	head: index of the front item
6	tail: index of the next free box
7	count: number of items
9–10	empty when count is 0
12–13	full when count equals capacity

ArrayQueue, Part 2: enqueue and dequeue

IIT Madras Zanzibar

```
 1  def enqueue(self, x):
 2      if self.is_full():
 3          raise OverflowError('full')
 4      self.a[self.tail] = x
 5      self.tail = (self.tail + 1) % self.cap
 6      self.count += 1
 7
 8  def dequeue(self):
 9      if self.is_empty():
10          raise IndexError('empty')
11      x = self.a[self.head]
12      self.a[self.head] = None
13      self.head = (self.head + 1) % self.cap
14      self.count -= 1
15      return x
```

Line	What it does
2–3	no room: refuse
4	write into the free box
5	tail to the next box, wrapping
6	one more item
9–10	nothing to remove: refuse
11	read the front item
12	clear its box
13	head to the next box, wrapping
14–15	one fewer item; return it

Implementing Queues

IIT Madras Zanzibar

The Queue and Its Promise

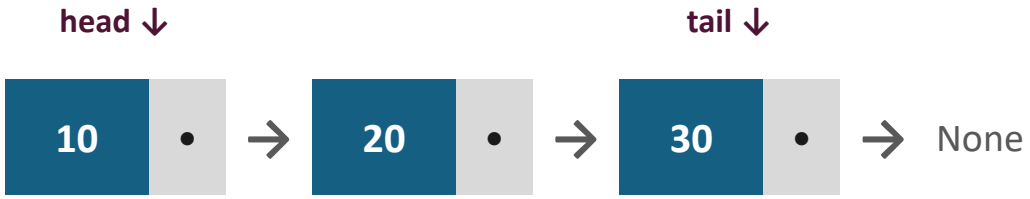
A Queue on a Circular Array

→ **A Queue on a Linked List**

Deque and Priority Queue

Live Demo and Practice

A Queue on a Linked List: Nodes, head and tail



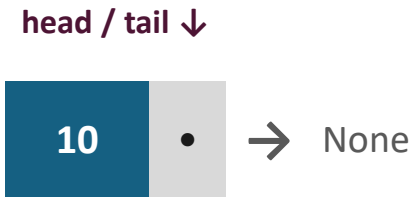
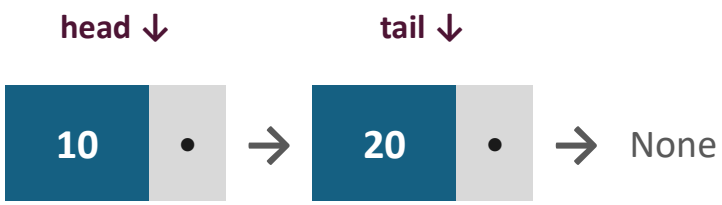
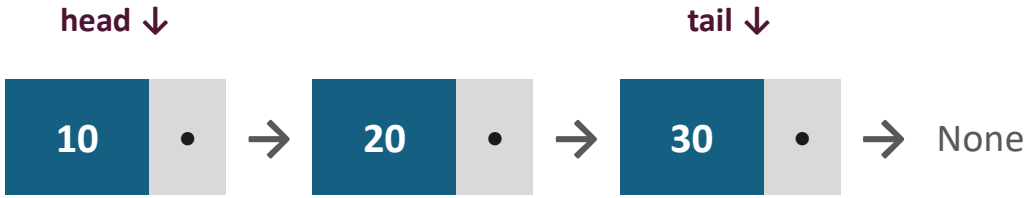
Design

1. Node = value + next. The last node's next is None.
2. head points to the FRONT node (dequeue there); tail points to the REAR node (enqueue there).
3. Why not dequeue at the tail? Removing the last node needs the node before it: O(n) in a singly linked list.
4. No capacity: each enqueue makes a new node.

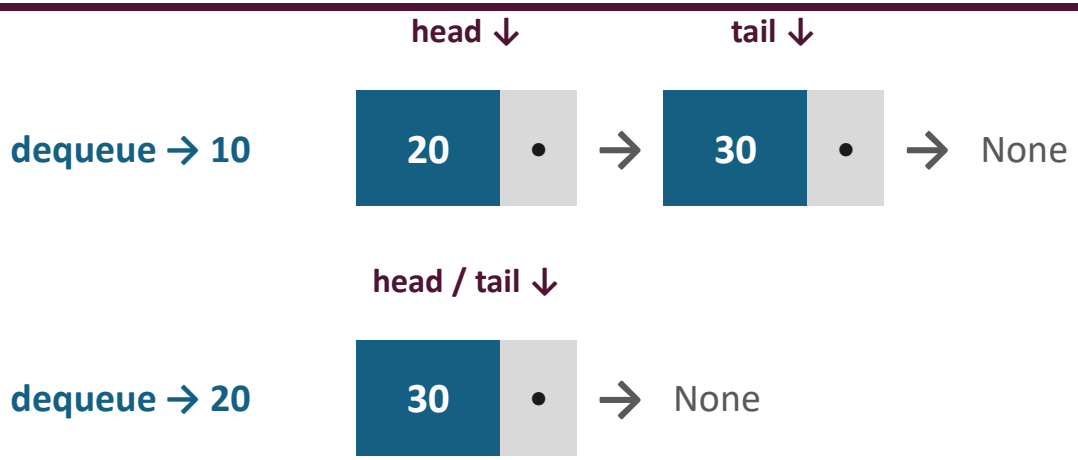
Definition: **Node:** an object with two fields: val (the data) and next (a reference to the next node, or None).

Linked Queue, Step by Step (1 of 2)

IIT Madras Zanzibar

empty	**head = None, tail = None**	head = None, tail = None
enqueue 10		new node; queue was empty, so head = tail = new node
enqueue 20		1. tail.next = new 2. tail = new
enqueue 30		1. tail.next = new 2. tail = new

Linked Queue, Step by Step (2 of 2)



LinkedQueue: The Code, Line by Line

IIT Madras Zanzibar

```
 1  class Node:
 2      def __init__(self, val):
 3          self.val, self.next = val, None
 4
 5  class LinkedQueue:
 6      def __init__(self):
 7          self.head = self.tail = None
 8      def enqueue(self, x):
 9          node = Node(x)
10          if self.tail is None:
11              self.head = self.tail = node
12          else:
13              self.tail.next = node
14              self.tail = node
15      def dequeue(self):
16          if self.head is None: raise IndexError
17          x = self.head.val
18          self.head = self.head.next
19          if self.head is None: self.tail = None
20          return x
```

Line	What it does
1–3	a node: value and next (None)
6–7	empty queue: no front, no rear
9	make the new node
10–11	empty: it is front and rear
13	old rear links to the new node
14	tail moves to the new node
17	read the front value
18	head moves one node along
19	became empty: clear tail too

Implementing Queues

IIT Madras Zanzibar

The Queue and Its Promise

A Queue on a Circular Array

A Queue on a Linked List

→ **Deque and Priority Queue**

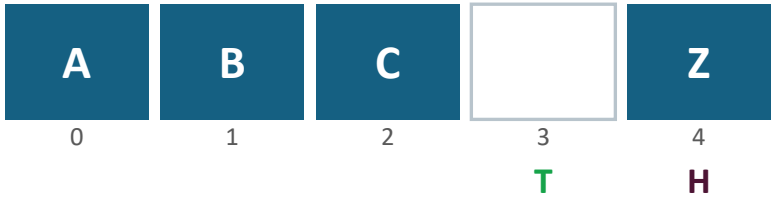
Live Demo and Practice

Deque: A Queue Open at Both Ends

IIT Madras Zanzibar

Operation	Index update	Example (capacity 5)
add_rear(x)	a[tail] = x; tail = (tail + 1) % cap	tail 2 → 3
remove_front()	x = a[head]; head = (head + 1) % cap	head 4 → 0
add_front(x)	head = (head − 1) % cap; a[head] = x	head 0 → (0 − 1) % 5 = 4
remove_rear()	tail = (tail − 1) % cap; x = a[tail]	tail 0 → 4

add_front(Z) when head = 0:



1. head = (0 − 1) % 5 = 4: Z goes into the LAST box.
2. Clockwise from head: Z, A, B, C. Z is at the front.

Definition: Deque (double-ended queue): a sequence with O(1) add and remove at both the front and the rear.

Priority Queue on a Binary Heap (a First Look)

IIT Madras Zanzibar

The idea

1. Every parent ≤ its children, so the root is the smallest.
2. Stored in an array: children of i at 2i+1 and 2i+2; parent at (i−1)//2.
3. push: add at the end, swap UP while smaller than the parent.
4. pop: take the root; move the last item to the root; swap DOWN.
5. Both O(log n): one path of a tree of height log n.

Push 1 into heap [3, 4, 8, 5]:



1 < parent 4 (index 1): swap



1 < parent 3 (index 0): swap. Done.



Definition: **Binary min-heap:** a complete binary tree, stored in an array, where every parent is ≤ its children; the minimum is at index 0.

Python's Ready-Made Versions

IIT Madras Zanzibar

```
from collections import deque
q = deque()
q.append('A')        # enqueue
q.append('B')
q.popleft()          # dequeue -> 'A'
q.appendleft('Z')    # deque: add front
q.pop()              # deque: remove rear

import heapq
pq = []
heapq.heappush(pq, (2, 'Juma'))
heapq.heappush(pq, (1, 'Asha'))
heapq.heappop(pq)    # -> (1, 'Asha')
```

Implementation	enqueue	dequeue	notes
Python list, pop(0)	O(1)*	O(n)	shifts everything
Circular array	O(1)	O(1)	fixed capacity
Linked list	O(1)	O(1)	extra memory per node
collections.deque	O(1)	O(1)	use this in practice
Binary heap (PQ)	O(log n)	O(log n)	smallest first

* amortized

Implementing Queues

IIT Madras Zanzibar

The Queue and Its Promise

A Queue on a Circular Array

A Queue on a Linked List

Deque and Priority Queue

→ **Live Demo and Practice**

Live Demo: Watch the Queue Move

IIT Madras Zanzibar

```
cd ~/PDS/Class_Demos/Week_04
python3 W04_Wed_Oct28_implementing_queues.py animate          # press Enter: one step at a time
python3 W04_Wed_Oct28_implementing_queues.py animate auto     # plays by itself
python3 W04_Wed_Oct28_implementing_queues.py ops 4 "enq A" "enq B" deq "enq C"
python3 W04_Wed_Oct28_implementing_queues.py linked           # or: deque, heap
```

```
enqueue E
   [   ] [   ] [ C ] [ D ] [ E ]
     0     1     2     3     4
     T           H           head=2 tail=0 count=3
   tail = (4 + 1) % 5 = 0
```

Before each Enter: ask the class where head and tail will be, and what the modulo gives.

In-Class Exercise (5 minutes, in pairs)

IIT Madras Zanzibar

IN-CLASS TASK: fill in the blanks from memory

```
def enqueue(self, x):
    if self.count == ______(1)______:
        raise OverflowError
    self.a[______(2)______] = x
    self.tail = ______(3)______
    self.count += 1

class LinkedQueue:
    def dequeue(self):
        x = self.head.val
        self.head = ______(4)______
        if self.head is None:
            ______(5)______
        return x

# ArrayDeque.add_front:   self.head = ______(6)______
```

Code Recall: Solution

IIT Madras Zanzibar

1. **(1) self.cap and (2) self.tail**

   full when count equals capacity; the new item goes into the free box at tail.

2. **(3) (self.tail + 1) % self.cap**

   next box, wrapping to 0 after the last one.

3. **(4) self.head.next and (5) self.tail = None**

   move to the next node; if the queue is now empty, tail must be cleared too.

4. **(6) (self.head - 1) % self.cap**

   step back one box; (0 − 1) % cap wraps to the last box.

In-Class Exercise (5 minutes, in pairs)

IIT Madras Zanzibar

IN-CLASS TASK: trace by hand, show head, tail and count

1. ArrayQueue, capacity 4, empty. enqueue 1, 2, 3; dequeue; dequeue; enqueue 4, 5, 6. Draw the array; give head, tail, count. What does the next dequeue return?

2. LinkedQueue: enqueue 7, 8; dequeue; dequeue. After each step, where do head and tail point?

3. ArrayDeque, capacity 4, empty (head = tail = 0). add_front(A), add_rear(B), add_front(C). Draw the array; head, tail?

4. Why can't a singly linked queue dequeue at the tail in O(1)?

5. Convention B (start at −1), capacity 3: enqueue X, enqueue Y, dequeue, dequeue. Give head and tail after each step.

Solution

IIT Madras Zanzibar

1

1: [5, 6, 3, 4], head 2, tail 2, count 4 (full)

1, 2, 3 go to boxes 0, 1, 2; two dequeues: head = 2; 4 → box 3 (tail wraps to 0), 5 → box 0, 6 → box 1, tail = 2. Next dequeue returns 3.

2

2: pointers

7: head = tail = 7. 8: head 7, tail 8. dequeue: head 8, tail 8. dequeue: head = tail = None.

3

3: [B, _, C, A], head 2, tail 1

add_front(A): head = (0 − 1) % 4 = 3, a[3] = A. add_rear(B): a[0] = B, tail = 1. add_front(C): head = 2, a[2] = C. Front to rear: C, A, B.

4

4: no previous pointer

removing the last node needs the node before it, which takes an O(n) walk from the head.

5

5: (0,0) → (0,1) → (1,1) → (−1,−1)

X: head = 0, tail = 0 (tail = last item). Y: head 0, tail 1. dequeue X: head 1, tail 1 (one item left). dequeue Y: it was the only item, so reset head = tail = −1.

Common Mistakes

IIT Madras Zanzibar

1 **Forgetting % capacity**

tail = tail + 1 eventually gives index = capacity: IndexError

2 **Using head == tail for 'empty'**

it is also true when FULL; keep a count

3 **Linked: not clearing tail**

after the last dequeue, tail still points to a removed node

4 **Linked: wrong order in enqueue**

tail.next = node BEFORE tail = node

Summary and Exam Checkpoints

IIT Madras Zanzibar

Circular array

head and tail move with (index + 1) % capacity; count tells full from empty

Linked list

enqueue at tail, dequeue at head; clear tail when the queue empties

Deque

add_front / remove_rear step BACK: (index − 1) % capacity

Priority queue

binary heap: push and pop O(log n); Python heapq

Next

Thu 29 Oct, 10:00: Linked lists

Thu 29 Oct, 2:00 pm: Lab: implement queues and linked lists

Demo code: Class_Demos/Week_04/W04_Wed_Oct28_implementing_queues.py

References and Further Reading

[1] Canning, J., Broder, A. & Lafore, R. (2022). Data Structures & Algorithms in Python. Addison-Wesley. Chapter 4 (Queues, circular queues, deques, priority queues).

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2022). Introduction to Algorithms, 4th ed. MIT Press. Section 10.1 (queues on arrays); Chapter 6 (heaps, priority queues).

[3] Python documentation: collections.deque, heapq, queue.