Circular Queues, Linked Lists and the LRU Cache

Lecture Script, Worked Traces, Worksheet and Solutions
Z5007 Programming and Data Structures, MTech DSAI | Week 4, Oct 28–30, 2026

1. Circular Array Queues

> Calling `pop(0)` on a Python list shifts every surviving item left in $O(n)$ time. Bending a fixed buffer of capacity C into a ring with modular wrap-around, $tail = (tail + 1) \bmod C$, keeps both `enqueue` and `dequeue` at exact $O(1)$ time while keeping memory contiguous.

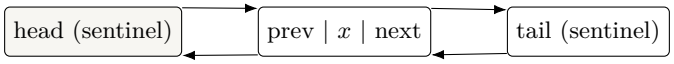
Using a separate integer `count` removes the empty-versus-full ambiguity when `head == tail`: `count == 0` means empty and `count == C` means full.

2. Pointer Surgery: Singly vs. Doubly Linked Lists

In a singly linked list with `head` and `tail`, prepending takes $O(1)$, whereas deleting the tail node still costs $O(n)$ because the node before `tail` cannot be reached backward.

Adding a `prev` reference plus dummy `head` and `tail` sentinels reduces every insert and unlink to uniform pointer updates with zero empty-list boundary branches:

```
def _unlink(node):
    p, q = node.prev, node.next
    p.next = q
    q.prev = p
    node.prev = node.next = None
```



Check question. Why do sentinel nodes eliminate `if head is None` checks during insertion and deletion?

Answer. Every real data node always sits strictly between two valid non-null nodes (`node.prev` and `node.next` are never `None`), so linking and unlinking follow one fixed four-assignment path.

3. $O(1)$ LRU Cache Architecture

An LRU Cache pairs a Python `dict` mapping `key -> node` (giving $O(1)$ key lookup) with a sentinel doubly linked list ordered by recency (giving $O(1)$ promotion to most recent and eviction from the least-recent sentinel).

4. Worksheet (Students Complete Individually)

1. A circular queue of capacity 5 starts with `head=0, tail=0, count=0`. Show `head, tail, count` after enqueue(10, 20, 30), dequeue(), dequeue(), enqueue(40, 50, 60).
2. Simulate a capacity-2 LRU cache on: `put(1,1)`, `put(2,2)`, `get(1)`, `put(3,3)`, `get(2)`, `put(4,4)`, `get(1)`, `get(3)`, `get(4)`.

5. Worksheet Solutions

1. After 3 enqueues: `head=0, tail=3, count=3`; after 2 dequeues: `head=2, tail=3, count=1`; after 3 enqueues: `tail=(3+3)%5=1, head=2, count=4`.

2. `get(1)->1`; `put(3,3)` evicts key 2; `get(2)->-1`; `put(4,4)` evicts key 1; `get(1)->-1`, `get(3)->3`, `get(4)->4`.

References

1. Canning, J., Broder, A. and Lafore, R. (2022). *Data Structures and Algorithms in Python*. Addison-Wesley. Chapters 4–5.

2. Cormen, T. H. et al. (2022). *Introduction to Algorithms*, 4th ed. MIT Press. Chapter 10.