Week 4 Study Summary: Circular Queues, Linked Lists, and LRU Cache

Z5007 Programming and Data Structures · Oct 28–30, 2026 · Dr. Innocent Nyalala

1. FIFO Queues: Circular Array vs. Linked Queue

A naive list queue (`pop(0)`) shifts every remaining element in $O(n)$. A `CircularArrayQueue` tracks `_front` and `_size`, placing incoming items at `(_front + _size) % cap` and advancing `_front = (_front + 1) % cap` on dequeue in $O(1)$ amortized time. A `LinkedQueue` maintains both `head` and `tail` pointers so `enqueue` at tail and `dequeue` at head run in $O(1)$ worst-case time.

2. Pointer Surgery and Floyd Cycle Detection

In-place singly linked list reversal tracks `(prev, cur, nxt)` in $O(n)$ time and $O(1)$ auxiliary memory. Floyd's tortoise-and-hare pointers move slow by 1 step and fast by 2 steps, detecting any cycle in $O(n)$ time and $O(1)$ space.

3. Sentinel Doubly Linked Lists and $O(1)$ LRU Cache

Dummy `head` and `tail` sentinels eliminate empty-list edge cases during node insertion and unlinking. Combining a hash map `key -> node` with a sentinel doubly linked list yields $O(1)$ `get` (move accessed node to front) and $O(1)$ `put` (evict `tail.prev` when full).

Complexity Summary

All operations (`enqueue`, `dequeue`, `get`, `put`) achieve $O(1)$ time; array-backed circular queues resize in $O(n)$ rarely ($O(1)$ amortized).