数据结构

第 8 周 – 队列：先进先出原则、循环队列、双端队列、优先队列

Koczka Ferenc 博士 · 埃斯特哈齐·卡罗伊大学 · 2026

复习

- **栈：**后进先出 – 只能访问栈顶
- Push、Pop、Top – 都是 **O(1)**，用数组时在数组末尾操作
- 应用：后缀表达式求值、括号匹配、撤销、调用栈

今天的问题

- 栈和队列有什么区别?
- 为什么“朴素”的数组队列不好？**循环**队列怎样解决这个问题?
- 什么是双端队列和优先队列?

队列的概念

队列是一种在**队尾**插入、在**队头**删除的数据结构。

FIFO – First In, First Out（先进先出）：先进去的先出来。

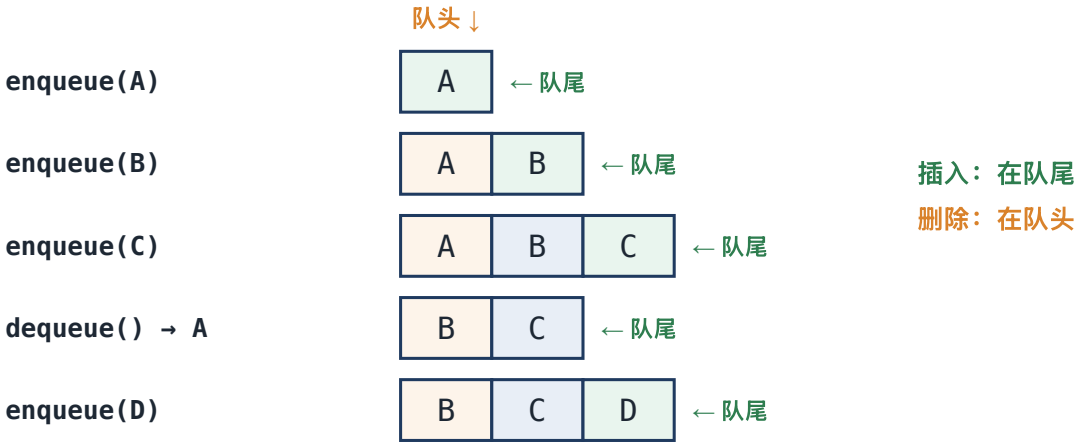
生活中的例子

- 在食堂、收银台排队
- 打印队列
- 客服电话排队
- 键盘缓冲区

性质

元素	同构
关系	线性
访问	**受限** – 队头和队尾
大小	静态或动态

队列的操作



插入：在队尾

删除：在队头

操作	含义	步数
Enqueue	元素加入**队尾**	O(1)
Dequeue	删除**队头**元素	O(1)
Peek / Front	读取队头元素	O(1)
IsEmpty? · IsFull?	状态	O(1)

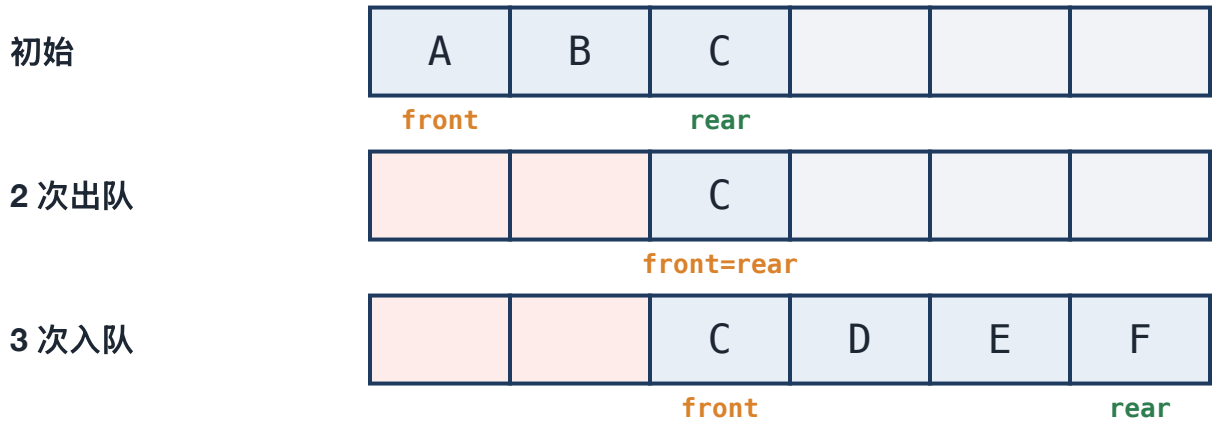
栈与队列的比较

	栈	队列
原则	**后进先出**	**先进先出**
插入	压入栈顶（push）	加入队尾（enqueue）
删除	从栈顶弹出（pop）	从队头删除（dequeue）
访问	一端	两端
数组实现中的指针	`top`	`front`、`rear`
典型应用	撤销、递归、表达式	调度、缓冲、广度优先遍历

用数组实现队列

第一次尝试：朴素的数组队列

`front` 是第一个元素的下标，`rear` 是最后一个元素的下标；入队：`rear ← rear + 1`，出队：`front ← front + 1`



队列向右“漂移”：

rear 到达末尾 → “满”，

尽管前面还有 2 个空位！

改进朴素队列的方法

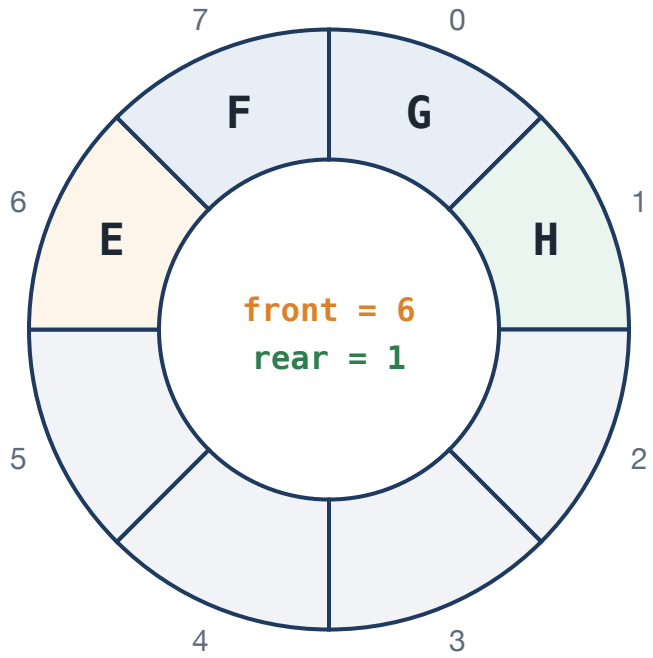
1. 移动元素：每次出队后，把所有元素向前移动一位

- 队头始终在 0，不浪费空间
- 但出队变成了 **O(n) ✗**

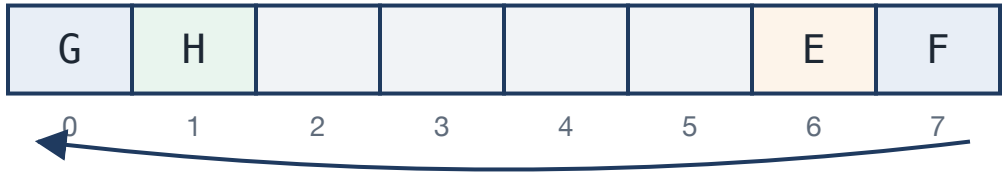
2. 循环队列：把数组看成一个**环** – `rear` 到达数组末尾时，从**开头**继续

- 没有元素移动，每个操作都是 **O(1) ✓**
- 下标“绕回”:**(i + 1) mod n**

循环队列



对应的数组形式：




```
enqueue: rear ← (rear + 1) mod n
dequeue: front ← (front + 1) mod n
空: count = 0
满: count = n
```

每个操作 O(1)，无需移动元素

空还是满？循环队列的陷阱

只用 `front` 和 `rear` 时，**空**队列和**满**队列看起来一样：`front = (rear + 1) mod n`！

解决办法：

1. 记录**元素个数**（`count`）– 空：`count = 0`，满：`count = n` ← **我们采用这种**
2. **始终留一个空位** – 满：`(rear + 1) mod n = front`，但只能放 **n** – 1 个元素
3. 用一个布尔标志（`empty`），由各操作负责设置

> 基于计数器的方法最简单，也最不容易出错。

循环队列 – 伪代码

```
// 数组 T[0..n-1](这里例外地从 0 开始,
// 这样 mod n 比较简单)

Procedure Create
  front ← 0 ; rear ← n − 1 ; count ← 0
End procedure

Function IsEmpty
  Return count = 0
End function

Function IsFull
  Return count = n
End function
```

```
Procedure Enqueue(x)
  If IsFull then ERROR("the queue is full")
  rear ← (rear + 1) mod n
  T[rear] ← x
  count ← count + 1
End procedure

Function Dequeue
  If IsEmpty then ERROR("the queue is empty")
  x ← T[front]
  front ← (front + 1) mod n
  count ← count − 1
  Return x
End function
```

循环队列 – Java 和 Go

Java

```java
class CircularQueue {
    private final int[] data;
    private int front = 0, rear = -1, count = 0;

    void enqueue(int x) {
        if (count == data.length)
            throw new IllegalStateException("full");
        rear = (rear + 1) % data.length;
        data[rear] = x;
        count++;
    }

    int dequeue() {
        if (count == 0)
            throw new IllegalStateException("empty");
        int x = data[front];
        front = (front + 1) % data.length;
        count--;
        return x;
    }
}
```

Go

```go
type CircularQueue struct {
    data               []int
    front, rear, count int
}

func (q *CircularQueue) Enqueue(x int) error {
    if q.count == len(q.data) {
        return errors.New("full")
    }
    q.rear = (q.rear + 1) % len(q.data)
    q.data[q.rear] = x
    q.count++
    return nil
}

func (q *CircularQueue) Dequeue() (int, error) {
    if q.count == 0 {
        return 0, errors.New("empty")
    }
    x := q.data[q.front]
    q.front = (q.front + 1) % len(q.data)
    q.count--
    return x, nil
}
```

内置的队列

Java

```java
Queue<String> q = new ArrayDeque<>();
q.offer("Li");      // 入队
q.offer("Wang");
q.peek();           // "Li"
q.poll();           // "Li" — 出队
q.isEmpty();
```

`ArrayDeque` 内部是一个**循环的、可动态增长的**数组。

Go

```go
q := []string{}
q = append(q, "Li")   // 入队
q = append(q, "Wang")
first := q[0]         // 查看队头
q = q[1:]             // 出队
```

简单，但 `q[1:]` 不会释放开头的空间 — 操作很多时，最好自己写循环队列或使用 `container/list`。

队列的变体

双端队列（deque）



栈和队列合二为一

- **Deque**（**double-ended queue**，读作“deck”）：两端都可以**插入**和**删除**
- 只用一端就是**栈**，用相对的两端就是**队列**
- 用循环数组实现：`addFirst` 时 `front ← (front − 1 + n) mod n`

在 Java 和 Go 中 `−1 % n` 的值是 **−1**，而不是 **n** − 1!所以需要 `+ n` 。

优先队列

每个元素都有一个**优先级**；我们总是取出**最重要**的元素（例如优先级数字最小的） – 而不是最早进入的。

例子：医院急诊、操作系统中的进程调度、路径规划（Dijkstra 算法）

实现	插入	取出最小值
无序数组	**O(1)** – 放在末尾	O(n) – 求最小值
有序数组	O(n) – 移动到正确位置	**O(1)** – 从末尾取
堆（第 15 周）	**O(log n)**	**O(log n)**

内置：Java `PriorityQueue` · Go `container/heap`

应用

- **缓冲：**键盘、网络数据包、视频流 – 平衡生产者和消费者的不同速度
- **调度：**打印队列、操作系统的**时间片轮转**调度
- **模拟：**收银台、交通、服务系统
- **广度优先遍历**（BFS）：树的层序遍历、图中的最短路径
- 分布式系统中的**消息队列**（例如处理网店的订单）

> **时间片轮转：**每个进程最多运行一个时间片，然后回到队尾 – 这样每个进程都能轮到。

小结

- **队列：**先进先出 – 在队尾插入，在队头删除
- 操作：**Enqueue**、**Dequeue**、**Peek**、**IsEmpty?**、**IsFull?** – 都是 **O(1)**
- **朴素数组队列：**队列“漂移”，前面的空间被浪费
- **循环队列：**`(i + 1) mod n` – 绕回，不移动元素；空/满：用**计数器**
- **双端队列：**两端都可以插入和删除
- **优先队列：**最重要的元素先出来 – 用堆实现 O(log n)

下周：动态数据结构 – 动态内存管理模型和单链表。

复习题

1. 输出什么？`enqueue(1); enqueue(2); dequeue(); enqueue(3); enqueue(4); dequeue(); dequeue(); dequeue();`
2. 循环队列，n = 5，初始为空（front = 0,rear = 4）。执行：4 次入队，3 次出队，3 次入队。现在 `front`、`rear` 和 `count` 各是多少？
3. 对 n = 10，下面的代码输出什么？一般地，它做了什么？
   `q.enqueue(0); q.enqueue(1); for (i = 0; i < n; i++) { a = q.dequeue(); print(a); b = q.peek(); q.enqueue(a + b); }`
4. 判断对错：在循环队列中出队是 O(n) 的，因为要把元素向前移动。
5. 下列情况你会用哪种数据结构：a) 浏览器的“后退”按钮，b) 打印任务，c) 医院急诊？

实践课

✎ 示例程序

程序	Java	Go	演示内容
朴素队列	`NaiveQueue.java`	–	“漂移”队列的问题，用移动元素修正
循环队列	`CircularQueue.java`	`w08/circularqueue`	绕回，用计数器判断空/满
双端队列	`CircularDeque.java`	`w08/deque`	两端操作，回文检查
优先队列	`PriorityQueueDemo.java`	`w08/priorityqueue`	用无序数组和用堆实现

✎ 练习

1. **代码阅读：**运行复习题 3 的代码，并解释输出！
2. **约瑟夫问题：**n 个人围成一圈，每数到第 k 个人就淘汰。最后剩下谁？（用队列：把 k – 1 个人送到队尾，第 k 个人淘汰。）
3. **时间片轮转：**模拟调度！进程：P1 = 5,P2 = 2,P3 = 4 个时间单位，时间片 = 2。每个进程什么时候结束？
4. 借助栈**把队列逆序**。
5. **代码阅读：**字母 = 入队，`*` = 出队并输出：`D A T * A * S * T R * U C * T`（与栈的结果比较！）
6. **滑动平均：**每来一个新数据，用 O(1) 步输出最近 k 个数据的平均值！

参考答案：`Exercises08.java`、`w08/exercises`

✎ 课后作业

1. 实现**不用计数器**、始终留一个空位的循环队列！大小为 **n** 的数组能放多少个元素?
2. 扩展循环队列，使它满时把元素复制到一个**两倍大**的数组中（动态循环队列）！注意绕回元素的顺序！
3. 用**两个栈实现队列**（见第 7 周的课后作业）！与循环队列比较！
4. **收银台模拟：**顾客随机到达，一个收银员每分钟服务一位顾客。在不同的到达率下，平均等待时间是多少?