双端队列 – 优先队列

王红元 coderwhy

目录
content

双端队列（Deque）

- 前端我们已经学习了队列（Queue）结构，它是一种受限的线性结构，并且限制非常的严格。
- 双端队列在单向队列的基础上解除了一部分限制：允许在队列的两端添加（入队）和删除（出队）元素。
  - 因为解除了一部分限制，所以在解决一些特定问题时会更加的方便；
  - 比如滑动窗口问题：https://leetcode.cn/problems/sliding-window-maximum/description/

239. 滑动窗口最大值

困难 2.1K

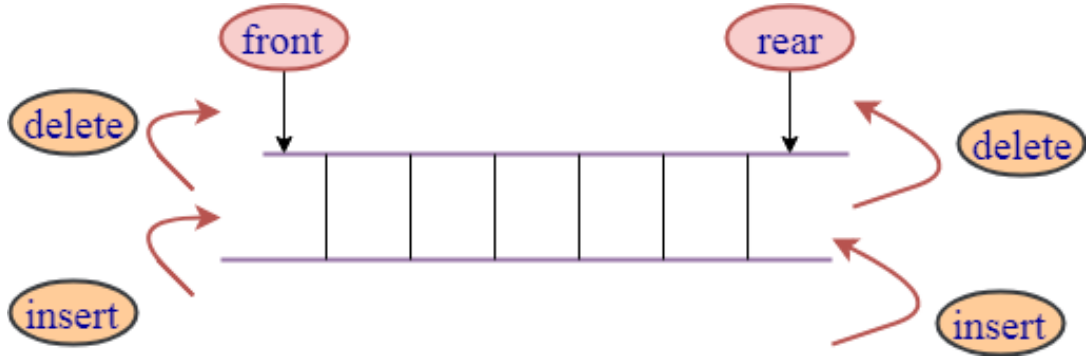
亚马逊 谷歌 Google 字节跳动 ...

给你一个整数数组 `nums`，有一个大小为 `k` 的滑动窗口从数组的最左侧移动到数组的最右侧。你只可以看到在滑动窗口内的 `k` 个数字。滑动窗口每次只向右移动一位。

返回 *滑动窗口中的最大值* 。

示例 1：

```
输入：nums = [1,3,-1,-3,5,3,6,7], k = 3
输出：[3,3,5,5,6,7]
解释：
滑动窗口的位置                最大值
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```



双端队列（Deque）的实现

```
class Deque<T> extends ArrayQueue<T> {
  addFront(value: T) {
    this.data.unshift(value)
  }

  removeBack(): T | undefined {
    return this.data.pop()
  }
}
```

```
const deque = new Deque<number>()
deque.enqueue(10)
deque.enqueue(20)
deque.addFront(30)

console.log(deque.peek())
while (!deque.isEmpty()) {
  console.log(deque.dequeue())
}
```

优先级队列（Priority Queue）

- **优先级队列(Priority Queue)是一种比普通队列更加高效的数据结构。**
  - 它每次出队的元素都是具有最高优先级的，可以理解为元素按照关键字进行排序。
  - 优先级队列可以用数组、链表等数据结构来实现，但是堆是最常用的实现方式。
- **优先级队列的应用**
- 一个现实的例子就是机场登机的顺序
  - 头等舱和商务舱乘客的优先级要高于经济舱乘客。
  - 在有些国家，老年人和孕妇（或带小孩的妇女）登机时也享有高于其他乘客的优先级。
- 另一个现实中的例子是医院的（急诊科）候诊室。
  - 医生会优先处理病情比较严重的患者。
  - 当然, 一般情况下是按照排号的顺序。
- 计算机中, 我们也可以通过优先级队列来重新排序队列中任务的顺序
  - 比如每个线程处理的任务重要性不同, 我们可以通过优先级的大小, 来决定该线程在队列中被处理的次序.

优先级队列的实现（一）

- 优先级队列的实现方式一：创建优先级的节点，保存在堆结构中。

```ts
class PriorityNode<T> extends Node<T> {
  priority: number
  constructor(value: T, priority: number) {
    super(value)
    this.priority = priority
  }

  valueOf() {
    return this.priority
  }
}
```

```ts
class PriorityQueue<T> {
  private heap: Heap<PriorityNode<T>> = new Heap()

  enqueue(element: T, priority: number): void {
    const node = new PriorityNode(element, priority)
    this.heap.insert(node)
  }

  dequeue(): T | undefined {
    return this.heap.delete()?.value
  }

  peek(): T | undefined {
    return this.heap.peek()?.value
  }

  isEmpty() {
    return this.heap.isEmpty()
  }

  size() {
    return this.heap.length
  }
}
```

优先级队列的实现（二）

- 优先级队列的实现方式二：数据自身返回优先级的比较值。

```
class PriorityQueue<T> extends ArrayQueue<T> {
  private heap: Heap<T> = new Heap()

  enqueue(element: T): void {
    this.heap.insert(element)
  }

  dequeue(): T | undefined {
    return this.heap.delete() ?? undefined
  }

  peek(): T | undefined {
    return this.heap.peek() ?? undefined
  }

  isEmpty() {
    return this.heap.isEmpty()
  }

  size(): number {
    return this.heap.length
  }
}
```

```
class Person {
  name: string
  age: number
  constructor(name: string, age: number) {
    this.name = name
    this.age = age
  }

  valueOf() {
    return this.age
  }
}
```