

Datenstrukturen: Queue (Warteschlange)

Das Konzept

Eine *Queue* oder *Warteschlange* ist eine Datenstruktur, in der die Elemente in der gleichen Reihenfolge entnommen werden, in der sie ihr hinzugefügt wurden. Dies wird durch die Formel *First In – First Out* ausgedrückt. Die charakteristischen Methoden sind:

- Ein Element am Ende der Queue hinzufügen: `Q.enqueue(item)`
- Ein Element am Kopf der Queue entnehmen: `Q.dequeue()`

Anwendungen

- Puffer für Tastatur und Maus
- Interprozesskommunikation (IPC)
- Prozesssteuerung

Implementierung als Ringpuffer

Ein Ringpuffer ist eine Möglichkeit, eine Queue effizient als abstrakten Datentyp zu implementieren. Diese Datenstruktur besteht aus einer Liste der Länge n , dem Attribut `Q.head`, das auf das nächste zu entfernende Element zeigt und dem Attribut `Q.tail`, das auf die nächste freie Einfügeposition zeigt.

Abbildung 1 zeigt, wie die Queue `Q` in (a) mit 3 Elementen durch die Operationen `Q.enqueue(7)`, `Q.enqueue(11)` und `Q.enqueue(3)` in den Zustand (b) versetzt wird, wobei das Einfügen am Anfang der Liste fortgesetzt wird, sobald das Listenende erreicht wird. Die Operation `Q.dequeue()` gibt den Schlüssel 2 zurück und versetzt die Queue in den Zustand (c).

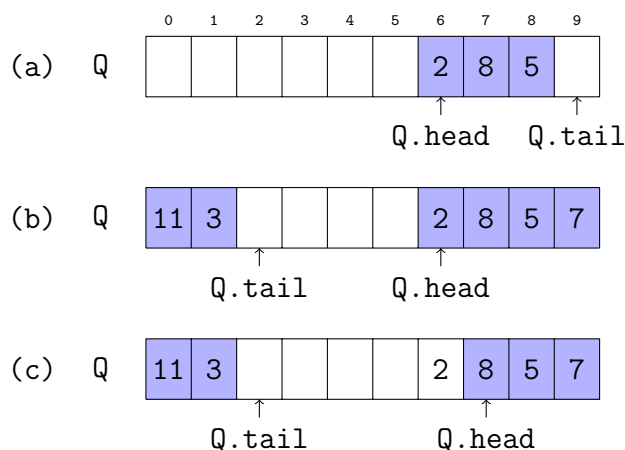


Abbildung 1: Einfüge- und Entnahmeoperationen in einer Queue, die als Ringpuffer implementiert ist

Implementierung in Python

```
1 class Queue:
2
3     def __init__(self, capacity=10):
4         self.cap = capacity
5         self.items = [None for i in range(0, self.cap)]
6         self.tail = 0 # Einfügeposition
7         self.head = 0 # Entnahmeposition
8         self.size = 0
9
10    def __len__(self):
11        return self.size
12
13    def is_empty(self):
14        return self.size == 0
15
16    def enqueue(self, item):
17        if self.size < self.cap:
18            self.items[self.tail % self.cap] = item
19            self.tail = (self.tail+1) % self.cap
20            self.size += 1
21        else:
22            print('Fehler - Queue hat keinen Platz mehr.')
23
24    def dequeue(self):
25        if self.size > 0:
26            item = self.items[self.head]
27            self.head = (self.head+1) % self.cap
28            self.size -= 1
29            return item
30        else:
31            print('Fehler - Queue ist leer.')
32            return False
33
34    def __str__(self):
35        tmp = ['head']
36        for i in range(0, self.size):
37            tmp.append(self.items[(self.head+i)%self.cap])
38        tmp.append('tail')
39        return ' '.join(map(str, tmp))
```