Datenstrukturen: Queue (Warteschlange) Lösung

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 enfernende 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).

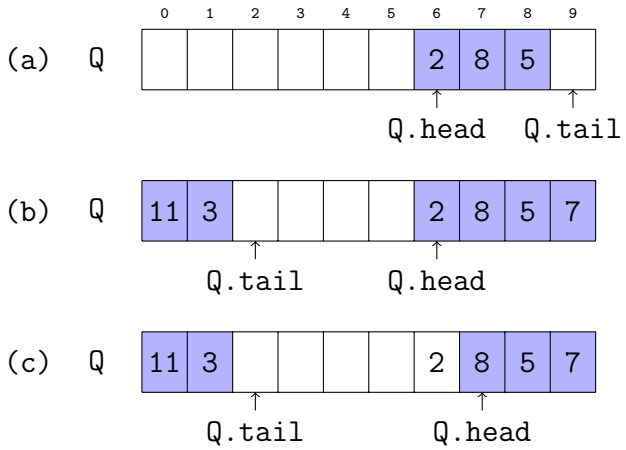

Abbildung 1: Einfüge- und Entnahmeoperationen in einer Queue, die als Ringpuffer implementiert ist

Implementierung in Python

```
class Queue:

    def __init__(self, capacity=10):
        self.cap = capacity
        self.items = [None for i in range(0, self.cap)]
        self.tail = 0 # Einfügeposition
        self.head = 0 # Entnahmeposition
        self.size = 0

    def __len__(self):
        return self.size

    def is_empty(self):
        return self.size == 0


    def enqueue(self, item):
        if self.size < self.cap:
            self.items[self.tail % self.cap] = item
            self.tail = (self.tail+1) % self.cap
            self.size += 1
        else:
            print('Fehler - Queue hat keinen Platz mehr.')

    def dequeue(self):
        if self.size > 0:
            item = self.items[self.head]
            self.head = (self.head+1) % self.cap
            self.size -= 1
            return item
        else:
            print('Fehler - Queue ist leer.')
            return False


    def __str__(self):
        tmp = ['head']
        for i in range(0, self.size):
            tmp.append(self.items[(self.head+i)%self.cap])
        tmp.append('tail')
        return ' '.join(map(str, tmp))
```