Einführung in die Algorithmik - Hausaufgabenserie 3

Nike Pulow, Henri Heyden
stu239549, stu240825

Aufgabe 1.3

Zu betrachten sind A1.3.py, NEW.py und OLD.py.
A1.3.py ist ein Testprogramm um die Queue-Klassen von OLD.py und NEW.py zu vergleichen.
OLD.py und NEW.py sind der Code der Vorlesung (queueClass.py) mit den Beispielen entfernt und unsere Implementation der Queue Klasse beschrieben in Aufgabe 2.
NEW.py und OLD.py sind neben den anderen Dateien auch hochgeladen.

```
Test 1: Initialisation of object
New method faster than old method: True
Time difference difference being: 6.300000000014627e-06s

Test 2: Enqueueing many objects
New method faster than old method: True
Time difference difference being: 0.27276140000000004s

Test 3: Dequeueing many objects
New method faster than old method: False
Time difference difference being: -4.7010468s

Test 4: Enqueueing a big object
New method faster than old method: True
Time difference difference being: 2.8000000007466497e-06s

Test 5: Dequeueing the big object
New method faster than old method: False
Time difference difference being: -1.2800000000368073e-05s
```

Listing 1: Output von A1.3.py unter meiner Maschine

Anscheinend sind wohl beide Klassen in verschiedenen Sachen schneller...
Der Konstruktor von NEW.Queue ist schneller als der Konstruktor von OLD.Queue, was Sinn ergibt, denn self.list = [] muss nur die Adresse dieser Liste irgendwo speichern, währenddessen self.empty = True nicht nur die Referenz speichert sondern auch den Wert True.
Dies ist aber natürlich auch nur ein sehr kleiner Unterschied und irrelevant, da eine Queue meistens auch nur ein mal erstellt werden muss.
Betrachten wir den zweiten Test sehen wir, dass unsere Methode ein bisschen schneller ist, da auf die Liste einfach ein Wert hinzugefügt wird, währenddessen bei der alten Implementierung self.end.fill(value) gecallt werden muss, was ein neues Objekt vom Typ ListElem erstellen muss, welches selbst auch noch weitere Objekte erstellt.
Beim dritten Test ist unsere Implementierung um sehr vieles Langsamer als die alte Implementierung, da die alte Implementierung eine konstante Laufzeit hat, währenddessen die Laufzeit bei uns linear ist, da fast die ganze Liste kopiert werden muss.
Um dieses Problem abzuschwächen, kann man statt self.list = self.list[1:] auch einfach del self.list[0] verwenden, was unsere Implementation um einiges schneller macht, aber die alte Variante ist noch um 0.5764608s schneller (auf meiner Maschine). Die alte Variante ist immer noch schneller, weil sie sich überhaupt nicht um Listen kümmern muss, sondern nur aus self.head.value und self.head lesen muss und auf result schreiben muss. Die Laufzeit ist bei unserer Implementation mit del immer noch linear.
Im vierten Test lässt es sich genau so argumentieren, wie im zweiten Test, jedoch fällt hier eben auf, dass die Größe eines Elements keinen großen Unterschied macht, wie schnell es hinzugefügt wird.
Beim fünften Test fällt wieder auf, dass die Größe, des Elements keinen großen Unterschied macht auf die Laufzeit vom dequeue.
Mit del self.list[0] statt self.list = self.list[1:] ist unsere Implementation wieder ein kleines bisschen schneller, jedoch macht es nahezu keinen Unterschied.
Also insgesamt sollte man die alte Implementierung vorziehen, ich bin gespannt, ob es sich noch effizienter programmieren ließe.