TU Kaiserslautern – FB Informatik
Dr. Patrick Michel

07.01.20

Algorithmen und Datenstrukturen (WS 2019)

Aufgabenblatt 8

zu bearbeiten bis: 13.01.20 - 15.01.20

Aufgabe 8.1 (Verkettete Listen - Theorie)
In der Vorlesung haben Sie eine alternative Implementierung des `List` Interface gesehen, mit Hilfe von einzelt verketteten Listen (`SingleLinkedList`).

- Annotieren Sie an jede Methode von `SingleLinkedList` die best und worst-case Komplexität und vergleichen Sie mit denen von `ArrayList`.

Aufgabe 8.2 (Verkettete Listen - Praxis)
Implementieren Sie nun eine verbesserte Fassung von verketteten Listen, die einige Operationen effizienter macht, aber auch von der Implementierung her gleichförmiger und insgesamt dadurch einfacher ist. Insbesondere soll diese Implementierung folgende Eigenschaften haben:

- Knoten sind **doppelt** verkettet, das heißt sie haben nun auch eine Referenz `prev` auf den Vorgänger. Die Implentierung soll entsprechend `DoubleLinkedList` heißen.
- Wir verwenden einen **Anker-Knoten**, so dass wir keinerlei `null` Referenzen mehr verwenden (und auch nicht mehr auf diese prüfen müssen). Der Anker zeigt mit `next` auf das erste Element der Liste, mit `prev` auf das letzte Element der Liste. Ist die Liste leer, dann zeigt der Anker mit beiden Referenzen auf sich selbst. Damit ist die Liste zu jedem Zeitpunkt ein **Ring** von Knoten und der Anker ist der Einstieg in diesen Ring.
- Die Länge der Liste wird explizit in einem Attribut **size** verwaltet.

Wir gehen bei der Implementierung wiefolgt vor:

- Die beiden Hilfsmethoden `insertNode` und `removeNode` fügen Knoten in den Ring aus Knoten ein oder entfernen einen. Sie sind die einzigen Methoden die an der Verkettung der Liste Änderungen vornehmen.
- Die Hilfsmethode `nodeAt` iteriert durch die Liste zu einem Index und gibt den entsprechenden Knoten zurück.
- Alle anderen Methoden können dann mit Hilfe dieser drei Methoden meist in einer einzigen Zeile implementiert werden.

Implementieren Sie also die Hilfsmethoden und dann die eigentliche Schnittstelle von `List` mit deren Hilfe.

- Implementieren Sie in `DoubleLinkedList` nun die beiden ADTs `Stack` und `Queue`.

Aufgabe 8.3 (Abstrakte Datentypen - Theorie)
Wir betrachten die Eignung der verschiedenen Implementierungen für abstrakte Datentypen (ADTs).

- Bewerten Sie die Eignung von `DoubleLinkedList` als Implementierung für `Queue` und vergleichen Sie mit `ArrayList`.

Eine Variante des Typs `Queue`, mit denselben Operationen, ist die `PriorityQueue`. Sie unterscheidet sich im Verhalten von `Queue` dadurch, dass die `dequeue`-Operation nicht das älteste Element entfernt, sondern das mit der höhesten "Priorität". Die Priorität kann dabei von den Elementen in der Queue ableitbar sein, z.B. das Alter einer Bestellung oder die Priorität eines Netzwerkpakets in einem überlasteten Router. Im letzteren Fall ist die Kapazität der Queue meist beschränkt und es sollen zuerst die weniger wichtigen Pakete bei Überlast verworfen werden.

Eine Implementierung von `PriorityQueue` – egal ob beschränkt oder nicht – braucht also eine Strategie beim Entfernen das maximale Element zu finden.

- Diskutieren Sie wie Sie diesen ADT unter Verwendung einer `DoubleLinkedList` implementieren könnten.
- Welche Komplexität haben dabei die `enqueue` und die `dequeue` / `peek` Operationen.
- Überlegen Sie welche Datenstruktur, die Sie bisher in der Vorlesung kennengelernt haben, sich besser eignen würde und begründen Sie warum.