5. Aufgabenblatt zur Vorlesung

ProInf III: Objektorientierte Programmierung

SoSe 2018

Dozent: Oliver Wiese

Bearbeitung im Tutorium am 04.09.2018

Aufgabe 1 Quicksort

(a) Schreiben Sie eine Variante des Quicksort-Algorithmus aus der Vorlesung, der den Median-Wert aus drei zufällig gewählten Elementen des zu sortierenden Teilarrays berechnet und diesen Wert als Pivot verwendet. Testen und vergleichen Sie Ihre Implementierung mit der aus der Vorlesung für zufällige und für aufsteigend sortierte Listen (worst-case-Eingabe).

(b) Zeigen und erläutern Sie anhand eines konkreten Zahlenbeispiels, dass der Quicksort-Algorithmus aus der Vorlesung nicht stabil ist.

(c) Wann ist der InsertionSort-Algorithmus dem Quicksort-Algorithmus vorzuziehen?

Aufgabe 2 Mergesort

(a) Zeigen Sie die einzelnen Schritte des MergeSort-Algorithmus beim Sortieren der Liste $[5, 1, 3, 6, 8, 0, 4, 7]$.

(b) Programmieren Sie eine iterative (nicht rekursive) Variante des Mergesort-Algorithmus aus der Vorlesung, indem Sie nur ein Hilfsarray verwenden, um die Zwischen-Ergebnisse der Merge-Operationen zu speichern. Keine weiteren Listen oder Arrays sollen während das Sortieren produziert werden.

(c) Testen Sie Ihren Algorithmus mit zufälligen Daten.

Aufgabe 3 Warteschlangen

Mittels Kameraüberwachung im Kassenbereich eines Kaufhaues kann die Anzahl an Artikeln im Einkaufswagen abgeschätzt werden. Der bekannte Hersteller von Kassensystemen möchte nun ein intelligente Kassenleitsystem entwickeln. Die Kerneigenschaft soll sein, dass Kundinnen und Kunden mit wenigen Artikeln im Einkaufswagen bevorzugt werden. Ihre Aufgabe ist die Entwicklung einer passende Datenstruktur mit den Funktionen `anstellen` und `naechster`.

(a) Betrachten Sie die min-heapify Eigenschaft. Diese besagt, dass der Wert in einem Elternknoten kleiner ist als der in den Kinderknoten (also die Umkehrung der max-heapify Eigenschaft aus der Vorlesung). Schreiben Sie eine `min-heapify` Funktion, welche für einen Knoten im Heap die min-heapify Eigenschaft prüft und wiederherstellt. Diese Funktion soll auch rekursiv bei geänderten Kinderknoten die Eigenschaft prüfen und wiederherstellen.

(b) Schreiben Sie eine Funktion `anstellen`. Diese Funktion wird von Kunden aufgerufen und soll ein neues Element einfügen. Als Parameter wird die Anzahl der Artikel übergeben und eine ID.

Hinweis: Neue Elemente sollte hinten angereiht werden und anschließend soll das Element statt nach unten gegebenfalls nach oben wandern. Passen Sie dafür Ihre `min-heapify` Funktion an.

(c) Schreibe Sie eine Funktion `naechster`. Diese Funktion wird vom Personal an der Kassen aufgerufen und gibt die ID der nächsten Person an, die zur Kasse kommen sollen.

Hinweis: Überlegen Sie sich mit welchem Element Sie die Wurzel im Heap ersetzen sollen.

(d) Testen Sie Ihre Datenstruktur sinnvoll.

(e) Analysieren Sie die Laufzeit von `naechster` und `anstellen`. Betrachten sie zusätzlich noch folgende Frage: Kommt jede Person in der Warteschlange auch an die Kasse und kann den Supermarkt verlassen?