Karlsruher Institut für Technologie

Institut für Theoretische Informatik
T.T.-Prof. Dr. Thomas Bläsius
https://scale.iti.kit.edu

Übungsblatt 3

Algorithmen 1 - Sommersemester 2023

Abgabe im ILIAS bis 12.05.2023, 18:00 Uhr
Bitte beschrifte deine Abgabe gut sichtbar mit deinem Namen und deiner Matrikelnummer. Achte insbesondere bei handschriftlichen Abgaben auf Lesbarkeit und genügend Platz für Korrektur-Anmerkungen. Die Abgabe erfolgt über das Übungsmodul in der Gruppe deines Tutoriums im ILIAS. Gib deine Ausarbeitungen in *einer* Datei im PDF-Format ab.
Achte darauf, effiziente Algorithmen zu formulieren, also solche mit möglichst geringer asymptotischer Laufzeit! Wenn du die Korrekheit deines Algorithmus begründen oder dessen Laufzeit analysieren sollst, tue dies getrennt von der Beschreibung deines Algorithmus. **Gib Algorithmen nur in Pseudocode an, wenn dies explizit gefordert ist.**

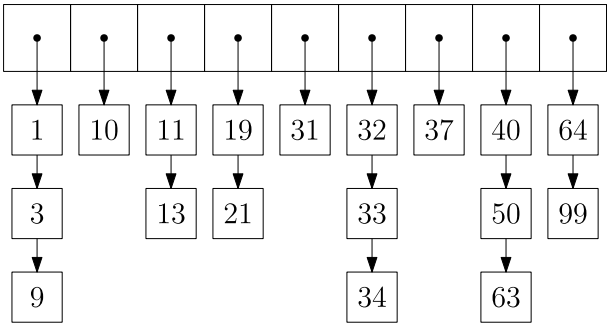
Gesamtpunktzahl: 20 (+ max. 6 Bonuspunkte)

Aufgabe 1 - Amortisierte Analyse (9 Punkte)

Wir betrachten eine Datenstruktur die aus einem Array A besteht, dessen Einträge sortierte Listen sind. Mit ℓ bezeichnen wir die Länge von A, also die Anzahl Listen. Jede Liste $A[i]$ erfüllt zwei Eigenschaften:

- $A[i]$ hat maximal Länge $\sqrt{\ell}$ und
- $\texttt{first}(A[i]) \geq \texttt{last}(A[i-1])$.

Das heißt, wenn wir alle Listen in der Reihenfolge konkatenieren wie sie im Array auftreten, dann enthält die resultierende List höchstens $\ell\sqrt{\ell}$ Elemente in sortierter Reihenfolge. Hier ein Beispiel:

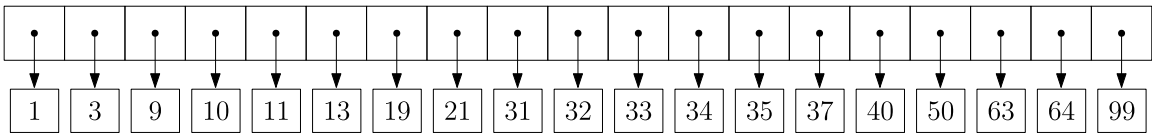


Die Datenstruktur unterstützt nun folgende Operationen:

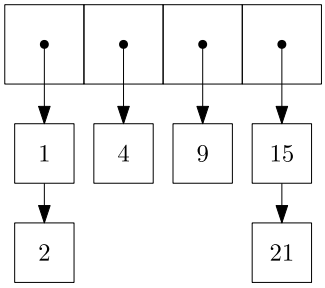
- $\texttt{find}(x)$: Gibt aus, ob ein Element x in unserer Datenstruktur ist.
- $\texttt{add}(x)$: Fügt ein Element x passend sortiert in die Datenstruktur ein, sodass x an das nächstkleinere Element angehängt wird (Du kannst annehmen, dass ein solches kleineres Element immer existiert). Sollte dadurch eine Liste eine Länge von größer als $\sqrt{\ell}$ haben, so ordnen wir die Elemente aller Listen um und erstellen eine neue Datenstruktur, in der jede Liste

Länge 1 hat (und das Array immernoch sortiert ist). Dieser Prozess des Umbauens dauert $\Theta(\ell)$, wobei ℓ nun die neue Anzahl Listen ist.

Als Beispiel ist hier unsere Datenstruktur von oben nach der Operation `add`(35) abgebildet. Da die Liste $A[5]$ zu lang wurde, ist die Datenstruktur umstrukturiert worden.



a) Führe add(x) für die folgenden Werte $[3, 27, 14, 8, 3, 13]$ auf der folgenden gegebenen Datenstruktur aus. Wie sieht die Datenstruktur danach aus? (3 Punkte)



b) Zunächst wollen wir uns mit der `find`(x) Operation beschäftigen. Beschreibe einen Algorithmus, der x als Eingabe erhält und ausgibt, ob sich das Element x in der Datenstruktur befindet. Gib an welche Laufzeit der Algorithmus benötigt. (2 Punkte)

c) Beschreibe, wie wir die Position finden können, an der ein neues Element x eingefügt werden muss, damit die Sortierung erhalten bleibt. Benutze dann eine Methode deiner Wahl, um zu zeigen, dass die Operation `add`(x) amortisiert $\Theta\left(\sqrt{\ell}\right)$ Zeit braucht. (4 Punkte)

*) Für jede weitere Methode der amortisierten Analyse mit der du zeigst, dass `add`(x) in $\Theta\left(\sqrt{\ell}\right)$ liegt, bekommst du zwei Bonuspunkte.

Aufgabe 2 - Oben und Unten und Links und Rechts (6 Punkte)

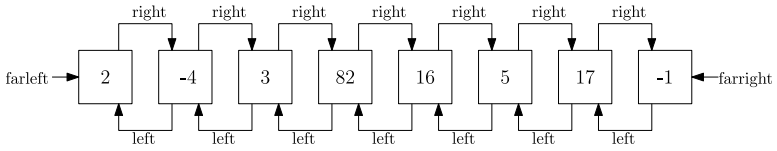
Nun betrachten wir eine Datenstruktur, die aus einer doppelt-doppelt-verketteten Liste besteht. Für den Rest der Aufgabe nennen wir diese Datenstruktur eine **DPAD-List**.

Eine **DPAD-List** hat folgende Eigenschaften:

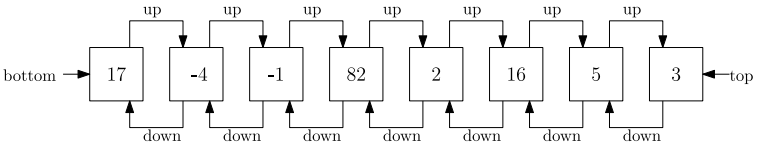
- jeder Knoten enthält eine Zahl, sowie 4 Pointer
  - zwei horizontale Pointer `left` und `right`
  - zwei vertikale Pointer `up` und `down`
- Des Weiteren gibt es 4 Pointer `farleft`, `farright`, `top`, und `bottom`, welche jeweils auf das einzige Element zeigen, dessen `left`, `right`, `up`, oder `down` Pointer `null` ist.

Ein Beispiel für diese Datenstrukur ist hier zu sehen:

Eine Ordnung



Die gleiche **DPAD-List** in einer anderen Ordnung



Nun wollen wir zwei Operationen auf dieser Datenstruktur einführen.

- `add`(x, y, n): Fügt ein Element n zwischen x und x`.right` und zwischen y und y`.down` ein.
- `remove`(n): Entfernt das Element n aus der Datenstruktur und stellt sicher, dass die **DPAD-List** nach wie vor die oben beschriebenen Eigenschaften hat.

Im Folgenden soll nun Pseudocode angegeben werden, der diese Operationen umsetzt. Achte dabei insbesondere darauf, anzugeben, wie sich die Pointer der einzelnen Elemente verändern.

a) Welches Element steht bei `82.up.right.down.left`? (1 Punkt)

b) Gib Pseudocode für einen Algorithmus an, der die Operation `add`(x, y, n) für die **DPAD-List** implementiert. (2 Punkte)

c) Gib Pseudocode für einen Algorithmus an, der die Operation `remove`(n) für die **DPAD-List** implementiert. (3 Punkte)

Aufgabe 3 - Queues (5 Punkte)

Wir wollen nun gemeinsam die Datenstruktur **Queue** modellieren, die die folgenden Operationen unterstützen soll:

- `enqueue`(x): Fügt das Element x der Queue hinzu.
- `dequeue`(): Entfernt das älteste Element aus der Queue und gibt es aus.

Hinweis: In dieser Aufgabe könnte sich Amortisierung verstecken!

a) Beschreibe wie eine solche Queue mit Hilfe einer einfach verketteten Liste modelliert werden kann. Gib an, welche zusätzlichen Daten benötigt werden und wie `enqueue`(x) und `dequeue`() funktionieren. Gib an, welche Laufzeit deine Operationen benötigen und begründe warum.

(2 Punkte)

b) Beschreibe wie eine solche Queue mit Hilfe eines dynamischen Arrays modelliert werden kann. Gib an, welche zusätzlichen Daten benötigt werden und wie `enqueue`(x) und `dequeue()` funktionieren. Gib an, welche Laufzeit deine Operationen benötigen und begründe warum.

(3 Punkte)