Karlsruher Institut für Technologie

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

Übungsblatt 04

Algorithmen I - Sommersemester 2024

Abgabe im ILIAS bis 17.05.2024, 18:00 Uhr

Die Abgabe erfolgt alleine oder zu zweit als eine PDF-Datei über das Übungsmodul in der Gruppe eures Tutoriums im ILIAS. Beschriftet die Abgabe deutlich mit Matrikelnummer und Name.

- Achtet bei handschriftlichen Abgaben auf Lesbarkeit.
- Achtet darauf, ob Algorithmen in Worten oder Pseudocode beschrieben werden sollen.
- Es werden immer asymptotisch möglichst effiziente Algorithmen erwartet, wenn nicht anders angegeben.
- Wenn Korrektheits- oder Laufzeitanalysen gefordert sind, behandelt diese separat von der Algorithmenbeschreibung.

Gesamtpunkte: 20 (+ 2 Bonus)

Aufgabe 1 - Von beiden Enten her (5 Punkte)

Die Biberklonaufsicht (BKA) hat mittlerweile Wind von Dr. Metas geklonten Bibern bekommen. Um Dr. Meta aufzuhalten möchte die BKA herausfinden was Dr. Meta plant. Durch Tarnung zweier Agenten als Mitarbiber in Mr. Metas Geheimlabor, konnte die BKA zwei brisante Dokumente entwenden. Das erste Dokument scheint die Operationen einer Datenstruktur zu beschreiben, während das zweite die Umsetzung derselben durch drei Stacks darstellt. Der Inhalt der beiden Dokumente ist unten angegeben:

Ein **Quack** ist eine Datenstruktur, welche die Eigenschaften von Stacks und Queues vereint. Ein Quack repräsentiert eine Folge von Elementen, die von links nach rechts betrachtet werden.

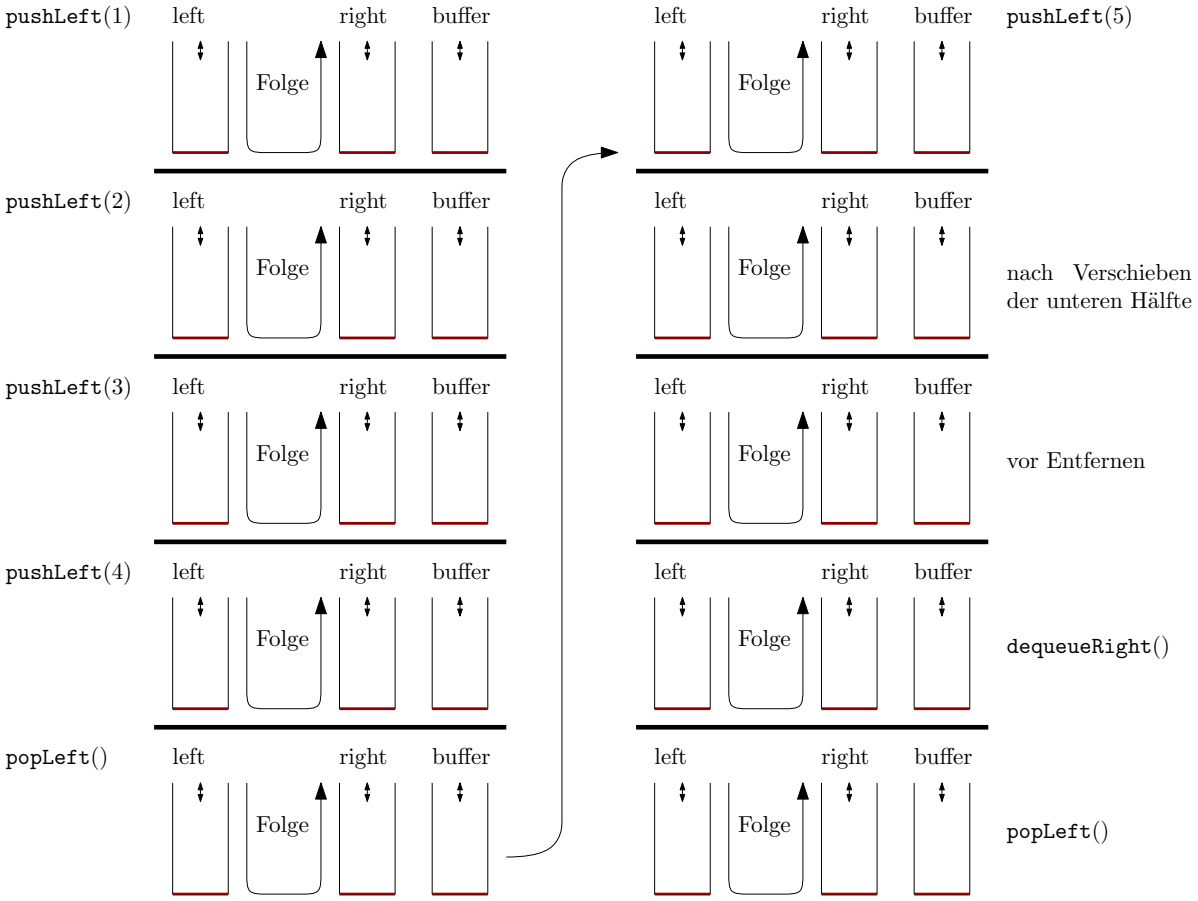
- `pushLeft`(x): Fügt das Element x am linken Ende der Folge ein.
- `popLeft`(): Entfernt das Element am linken Ende der Folge und gibt es zurück.
- `dequeueRight`(): Entfernt das Element am rechten Ende der Folge und gibt es zurück.

Es kann davon ausgegangen werden, dass `popLeft` und `dequeueRight` werden nur aufgerufen, wenn die vom Quack repräsentierte Folge nicht leer ist.

Bauanleitung für einen **Quack**:
Nutze Stacks *left*, *right* und *buffer*. Hierbei repräsentiert *left* den linken Teil der Folge und *right* den rechten. Dabei sind die Elemente von *right* in gespiegelter Reihenfolge angeordnet, so dass das rechteste Element oben auf dem Stack *right* liegt. Jedes Element kommt zu jedem Zeitpunkt nur in einem der drei Stacks vor. Somit können `pushLeft` und `popLeft` am linken Ende der Folge arbeiten, indem sie *left* nutzen. Analog arbeitet `dequeueRight` auf *right* und am rechten Ende. Falls bei `popLeft` oder `dequeueRight` der entsprechende Stack leer ist obwohl die repräsentierte Folge nicht leer ist, wird vor dem Entfernen die Last ausgeglichen. Bei einem Lastausgleich werden abgerundet die Hälfte der Elemente vom noch vollen Stack auf *buffer* verschoben. Anschließend werden die verbleibenden Elemente auf den leeren Stack und die Elemente von *buffer* zurück an ihren Ursprungsplatz verschoben. Nun speichern *left* und *right* je die Hälfte der Elemente. Beachte, dass jedes Verschieben der Elemenente ihre Reihenfolge umkehrt und daher die obigen Vorgaben eingehalten werden.

Das BKA vermutet, dass Dr. Meta einen Beweis dafür hat, dass die amortisierte Laufzeit aller Operationen konstant ist.

a) Hilf der BKA dabei die Datenstruktur zu verstehen. Führe dafür die Operationssequenz `pushLeft`(1), `pushLeft`(2), `pushLeft`(3), `pushLeft`(4), `popLeft`(), `pushLeft`(5), `dequeueRight`(), `popLeft`() auf einem leeren Quack aus und gebe den Zustand der Datenstruktur nach jeder Operation an. Wenn es zu einem Lastausgleich kommt, gebe außerdem die Zustände nach Verschieben der unteren Hälfte der Elemente und direkt vor dem Entfernen des Elementes an. Du kannst deine Ergebnisse in die untenstehende Abbildung eintragen. (2 Punkte)



Darstellung eines leeren Quacks, inklusive freie Felder für Zwischenzustände. Neue Elemente werden stets von oben (kleine Pfeile) auf den Stack gelegt und stapeln sich oberhalb der roten Kante. Zusätzlich ist die Sortierung der Folge über die beiden Stacks hinweg visualisiert.

b) Hilf der BKA den Beweis zu rekonstruieren. Zeige hierfür, dass die Laufzeit für eine beliebige Folge von Operationen auf einem zu Beginn leeren Quack bei $\Theta(1)$ pro Operation liegt. Nutze dafür eine Methode deiner Wahl. (3 Punkte)

Hinweis: Du darfst annehmen, dass die Kosten von `push` und `pop` auf den zugrunde liegenden **Stacks** die Kosten von genau 1 haben.

Aufgabe 2 - I did it ~~my~~ k-way (9 Punkte)

Du hast in der Vorlesung den Algorithmus MERGESORT kennen gelernt, der die Methode MERGE nutzt. Wir wollen nun eine Variante des Algorithmus entwickeln, die das Array in jedem Schritt in mehr als zwei Teile zerteilt.

a) Gib Pseudocode für eine Variante der bekannten MERGE-Methode an, die drei sortierte Arrays als Eingabe erhält und ein sortiertes Array, dass die selben Elemente enthält, zurückgibt.

Gib außerdem die Laufzeit deines Algorithmus in Abhängigkeit von seinen Eingabeparametern an und begründe diese. (3 Punkte)

b) Gib ebenfalls Pseudocode für den Sortieralgorithmus 3-WAY-MERGESORT an, der analog zu MERGESORT funktioniert, aber das Array in drei Abschnitte unterteilt. Welche Laufzeit hat dein Algorithmus (mit Begründung)? (1.5 Punkte)

c) Wir wollen nun für ein festes k in jedem Schritt k sortierte Arrays betrachten. Beschreibe in Worten wie die Algorithmen k-WAY-MERGE und k-WAY-MERGESORT umgesetzt werden können. Gib außerdem die Laufzeit von k-WAY-MERGE und k-WAY-MERGESORT in Abhängigkeit von k und ihrer Eingabe an und begründe diese Laufzeiten. (3 Punkte)

d) Wann und für welche k ist es sinnvoll k-WAY-MERGESORT anstelle des Algorithmus aus der Vorlesung zu nutzen? (1.5 Punkte)

Aufgabe 3 - Qual der Wahl oder Wahl der Qual (6 Punkte)

Gegeben sei ein unsortiertes Array $A : [\mathbb{N}_0;\ n]$, welches n paarweise verschiedene natürliche Zahlen enthält. Wir betrachten nur ungerade n, um die Definition des Median einfach zu halten. Wir wollen das k-kleinste Element in A bestimmen. Hierbei ist k Teil der Eingabe und darf somit in deiner Laufzeitschranke auftauchen.

a) Zur Vorbereitung wollen wir das folgende Problem lösen: An welcher Position würde ein Element, das in A an Index $i \in \{0, \ldots, n-1\}$ steht, in dem Array stehen, dass wir aus A durch Sortieren erhalten? Beschreibe einen Algorithmus, der die gesuchte Position bestimmt, und begründe die Laufzeit deines Algorithmus. (1 Punkt)

b) Beschreibe einen Algorithmus, welcher in $O(n)$ das k-kleinste Element in A bestimmt. Dein Algorithmus darf dabei die Reihenfolge der Einträge in A vertauschen. Du darfst annehmen, dass du den Median eines unsortierten Arrays in $\Theta(n)$ bestimmen kannst. Begründe, warum dein Algorithmus die geforderte Laufzeit hat.

Hinweis: Die folgende Rekurrenz könnte nützlich sein: $T(n) = T(\frac{n}{b}) + \Theta(n)$, für ein geschickt gewähltes b. (5 Punkte)

Aufgabe 4 - Zum Knobeln (0 Punkte + 2 Bonus)

Wir betrachten erneut das Szenario aus Aufgabe 3. Wir nehmen nun nicht mehr an, den Median eines unsortierten Arrays in $\Theta(n)$ bestimmen zu können. Gib einen randomisierten Algorithmus an, welcher das k-kleinste Element von A in erwartet $O(n)$ findet. Begründe die Korrektheit des Algorithmus und der Laufzeit. (2 Punkte)