Karlsruher Institut für Technologie

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

Übungsblatt 09

Algorithmen I – Sommersemester 2024

Abgabe im ILIAS bis 28.06.2024, 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 (+ 3 Bonus)

Aufgabe 1 - Stapelweise Szenarien (4 Punkte)

Bestimme für folgende Situationen, welche möglichst einfache und effiziente Datenstruktur jeweils geeignet ist und gib an, welche Operationen der von dir gewählten Datenstruktur dabei relevant sind.
Hinweis: In der Vorlesung haben wir folgende Datenstrukturen kennengelernt: Doppelt- und einfach-verkettete Listen, (dynamische) Arrays, Hashtabellen, binäre Heaps, (2, 3)-Bäume, Adjazenzliste.

a) In einer Bibliothek sind alle Bücher stets nach dem Titel lexikographisch sortiert. Jedes Buch kann ausgeliehen und zurückgegeben werden. Außerdem soll stets nachgeprüft werden können, ob ein Buch mit gegebenem Titel aktuell verfügbar ist. (1 Punkt)

b) Dr. Meta sammelt Einklebesticker mit Fußballstars der EM. Hierfür kauft er regelmäßig kleine Tüten, die jeweils fünf zufällige Karten enthalten. Er möchte nun wissen, ob es sich lohnt, die neuen Karten zu behalten oder ob er bereits im Besitz einer neu gezogenen Karte ist. (1 Punkt)

c) Der Musikdienstleister Metaify möchte eine neuartige Wiedergabeliste anbieten. Dabei können neue Songs an einer beliebigen Stelle der Liste hinzugefügt oder auch wieder gelöscht werden. Das verwendete Gerät spielt dann stets den ersten Song der Wiedergabeliste. (1 Punkt)

d) Da Dr. Meta beim Jenga[1] spielen stets verliert, hat er einen Algorithmus entworfen, der ihm den besten Stein zum Entfernen empfehlen soll. Der Algorithmus ordnet jedem Stein die Wahrscheinlichkeit zu, dass dieser den Turm zum Einsturz bringt und empfiehlt stets den Stein mit der geringsten Wahrscheinlichkeit. Wird ein Stein in einer Runde entfernt, können sich die Wahrscheinlichkeiten von anderen Steinen, die noch im Spiel sind, ändern. (1 Punkt)

Aufgabe 2 - Bereichsanfragen (11 Punkte)

Für ein Element e, das wir in einem $(2,3)$-Baum verwalten, bezeichnen wir die Anzahl der Elemente, die links von e gespeichert sind, als Rang von e. Für ein Element, das nicht im Baum gespeichert ist, definieren wir den Rang des Elementes als den Rang des nächst größeren, im Baum gespeicherten Elementes. Der Rang entspricht also genau der Anzahl der Elemente im $(2,3)$-Baum, deren Schlüssel echt kleiner sind als der Schlüssel von e.

Wir wollen nun $(2,3)$-Bäume so anpassen, dass wir zusätzlich den Rang eines Elements anfragen können.

a) Beschreibe, welche zusätzlichen Informationen in den inneren Knoten eines $(2,3)$-Baumes gespeichert werden müssen, um in $O(\log(n))$ Zeit den Rang eines Elementes bestimmen zu können. (1 Punkt)

b) Begründe, wieso das asymptotische Laufzeitverhalten von `insert`(e : Element), `find`(e : Element) und `remove`(e : Element) trotz der Ergänzung aus der vorherigen Teilaufgabe gleich bleibt. (2.5 Punkte)

Hinweis: Überlege, wie sich die zusätzlichen Informationen in den Knoten durch Operationen auf dem $(2,3)$-Baum ändern können und wie sie aktuell gehalten werden können.

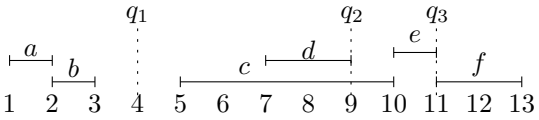
c) Beschreibe einen Algorithmus in Worten, der auf einem $(2,3)$-Baum mit deinen Erweiterungen aus bisherigen Teilaufgaben in Zeit $O(\log n)$ eine Anfrage des Rangs `rank`(e) eines Elements e beantwortet. Begründe, dass dein Algorithmus das geforderte Laufzeitverhalten hat. (2.5 Punkte)

Falls du bisherige Teilaufgaben nicht lösen konntest, darfst du die oben beschriebene Datenstruktur als Blackbox verwenden.

Wir nutzen nun $(2,3)$-Bäume mit Unterstützung von Ranganfragen, um eine Menge I von Intervallen mit Grenzen in $\mathbb{N}$ zu verwalten. Zusätzlich möchten wir *Intervallanfragen*

[1] `https://de.wikipedia.org/wiki/Jenga`

beantworten. Eine Intervallanfrage besteht aus einer natürlichen Zahl x und fragt, ob es ein Intervall in I gibt, welches diese Zahl enthält.



Sechs Intervalle und drei Intervallanfragen $q_1 = 4$, $q_2 = 9$ und $q_3 = 11$, die jeweils mit "Nein", "Ja" und "Ja" zu beantworten sind.

Gesucht ist nun eine Datenstruktur, mit der wir in $O(\log(n))$ Zeit Intervalle einfügen und löschen, sowie Intervallanfragen beantworten können, wobei n die Anzahl verwalteter Intervalle ist.

d) Betrachte das obige Beispiel mit seinen Intervallen. Gib für jede Zahl $x \in \{q_1, q_2, q_3\}$ an, wie viele Intervalle in x oder echt links von x beginnen und wie viele Intervalle echt links von x enden.
Beispiel: Für $x = 5$ beginnen drei Intervalle in x oder echt links davon und zwei Intervalle enden echt links von x. (1 Punkt)

e) Du darfst für diese Teilaufgabe annehmen, dass du für jede natürliche Zahl x weißt, wie viele der verwalteten Intervalle in x oder echt links von x beginnen und wie viele echt links von x enden. Erkläre kurz, wie du damit bestimmen kannst, ob x in einem der Intervalle liegt. (1 Punkt)

Wir nutzen nun zwei $(2,3)$-Bäume T_S und T_E, die Ranganfragen in $O(\log(n))$ unterstützen. Für jedes Intervall, das durch unsere Datenstruktur verwaltet wird, speichern wir seinen Startpunkt in T_S und seinen Endpunkt in T_E. Im Folgenden darfst du annehmen, dass an jedem Punkt höchstens ein Intervall startet und höchstens ein Intervall endet. Wir bezeichnen die Datenstruktur, die aus T_S und T_E besteht, als *Intervallbaum*. Der Intervallbaum soll das Einfügen (`insert`$([a,b]\colon \mathbb{N} \times \mathbb{N})$) und Löschen (`remove`$([a,b]\colon \mathbb{N} \times \mathbb{N})$) von Intervallen, sowie Intervallanfragen (`intervalQuery`$(x\colon \mathbb{N})$) unterstützen.

f) Beschreibe, wie `insert` und `remove` im Intervallbaum umgesetzt wird. (1 Punkt)

g) Beschreibe in Worten einen Algorithmus, der auf einem Intervallbaum in Zeit $O(\log n)$ eine Intervallanfrage `intervalQuery` beantwortet. Begründe, dass dein Algorithmus das geforderte Laufzeitverhalten hat. (2 Punkte)

Aufgabe 3 - Von 2-Bienchen und 3-Blümchen - Wie entstehen Bäume? (5 Punkte)

In dieser Aufgabe wollen wir uns mit der Konstruktion von $(2,3)$-Bäumen beschäftigen.

a) Beschreibe, wie in Linearzeit aus einer gegebenen sortierten Folge ein $(2,3)$-Baum konstruiert werden kann. Begründe, warum dein Algorithmus das geforderte Laufzeitverhalten hat. (3 Punkte)

b) Beschreibe, wie in Linearzeit zwei gegebene $(2,3)$-Bäume zu einem $(2,3)$-Baum zusammengefügt werden können. Begründe, warum dein Algorithmus das geforderte Laufzeitverhalten hat. (2 Punkte)

Hinweis: Achte darauf, dass Werte, die in beiden $(2,3)$-Bäumen vorkommen, nicht verloren gehen.

Aufgabe 4 - Zum Knobeln (0 Punkte + 3 Bonus)

In der Vorlesung wurde erwähnt, dass Suchbäume eine Art eierlegende Wollmilchsau unter den Datenstrukturen sind. In dieser Aufgabe sollst du diese Behauptung unterstreichen, indem du eine Priority-Queue der Größe n mit $(2,3)$-Bäumen umsetzt. In dieser Aufgabe, darfst du nur die in der Vorlesung vorgestellten Methoden `find`(x), `insert`(x) und `remove`(x) auf $(2,3)$-Bäumen aufrufen. Du sollst nun angeben, wie du die Operationen `push`, `popMin` und `decPrio` einer Priority-Queue in Zeit $\mathcal{O}(\log(n))$ umsetzen kannst.

Du darfst davon ausgehen, dass wir die natürlichen Zahlen $\mathbb{N}_0$ als Prioritäten verwenden und nur natürliche Zahlen in der Datenstruktur speichern. Um die Aufgabe einfacher zu halten, darfst du außerdem annehmen, dass gespeicherte Elemente und ihre Prioritäen identisch sind. Somit fügt `push`(x) die Zahl x in die Priority-Queue ein und `decPrio`(x,k) senkt die Priorität der Zahl x um k.

a*) Beschreibe die Umsetzung der Operationen in Worten. (2 Punkte)

b*) Begründe mit einem Satz, warum es weiterhin sinnvoll ist, binäre Heaps einzusetzen. (1 Punkt)