Algorithmen und Datenstrukturen 2

ALGO2 · WiSe-2021/22 · tcs.uni-frankfurt.de/algo2 · 2021-11-09 · *31d0ea4*

Übungen zu Woche 4: Amortisierte Analyse

Das Übungsblatt enthält alle empfohlenen Lernaktivitäten für die aktuelle Woche.

- **Heimarbeit bis Montag 17:00.**
  - Schau die Videos an und lies die Buchkapitel.
  - Bearbeite die 🌱-Aufgabe in Moodle. (Feste Abgabefrist!)
  - Lese den Aufgabentext aller Übungsaufgaben.
- **Heimarbeit.** Bearbeite die Übungsaufgaben soweit möglich. Probier zumindest alle mal!
- **Dienstag/Donnerstag.**
  - **8:00–8:15.** Besprechung im Hörsaal.
  - **8:15–9:15.** Bearbeite jetzt die Übungen, die du noch nicht lösen konntest. Sprich mit anderen Studis! Frag das Vorlesungsteam um Hilfe!
  - **9:15–9:45.** Lösungsspaziergang zu den Aufgaben für heute.
- **Heimarbeit bis Freitag, den 12.11., 17:00.** Gib deine Lösungen zu der ⭐-Aufgabe von diesem Übungsblatt in Moodle ab. (Feste Abgabefrist!)

Dienstag

Aufgabe 4.1 (Splay-Bäume). Gegeben sei ein leerer Splay-Baum.

a) Füge die Schlüssel $41, 38, 31, 12, 19, 8$ in der angegebenen Reihenfolge ein. Wie sieht der resultierende Splay-Baum aus?

b) Wie sieht der Splay-Baum aus, nachdem man Schlüssel 31 entfernt hat?

Aufgabe 4.2 (Amortisierte Analyse). Wir haben eine Datenstruktur mit einer Operation `Foo()`. Die Kosten $T(i)$ der i-ten Ausführung dieser Operation sind gegeben durch:

$$T(i) = \begin{cases} 2i & \text{falls i eine Zweierpotenz ist,} \\ 1 & \text{sonst.} \end{cases}$$

Was ist die amortisierte Laufzeit der Operation `Foo()`? Benutze sowohl die Aggregationsmethode, als auch das Buchhalter-Argument (*accounting method*), um die amortisierte Laufzeit zu analysieren.

Aufgabe 4.3 (Potenzialfunktion: Verdopplung von Arrays). Gegeben sei eine dynamische Tabelle, die nur Einfügungen und keine Löschungen erlaubt, und für die die Verdopplungsmethode genutzt wird. Wenn wir als Potenzialfunktion $\Phi(D_i) = k$ wählen, wobei k die Anzahl an Elementen im Array ist, können wir dann mit dieser Potenzialfunktion zeigen, dass die amortisierten Kosten einer Einfügung $\mathcal{O}(1)$ sind?

Aufgabe 4.4 (Mengenvereinigung). Mengenvereinigung ist eine abstrakte Datenstruktur, die disjunkte Teilmengen von $\{1, \dots, n\}$ verwaltet (ganz ähnlich zu Union-Find). Anfangs sind die Elemente auf n einzelne Mengen aufgeteilt, also $\{1\}, \{2\}, \dots, \{n\}$. Die Datenstruktur soll die folgenden Operationen unterstützen:

- Union(A, B): Führe die beiden Mengen A und B zu einer neuen Menge $C = A \cup B$ zusammen und lösche die alten Mengen.
- SameSet(x, y): Gebe *true* zurück, wenn x und y in derselben Menge liegen, ansonsten gebe *false* zurück.

Diese Operationen können wir folgendermaßen implementieren: Weise jeder Menge am Anfang eine eigene Farbe zu. Um die Union-Operation zu unterstützen, übernimm für alle Elemente in der kleineren Menge die Farbe der größeren Menge (bei Gleichstand wird eine der beiden Farben beliebig gewählt). Um die SameSet-Operation zu beantworten, überprüfe, ob die zwei Elemente dieselbe Farbe haben.

a) Analysiere die *worst-case* Laufzeit der beiden Operationen.
b) Zeige, dass die amortisierten Kosten $\mathcal{O}(\log n)$ für Union und $\mathcal{O}(1)$ für SameSet betragen. Zeige ebenfalls, dass jede Sequenz von m Union-Operationen und l SameSet-Operationen die worst-case Laufzeit $\mathcal{O}(m \log n + l)$ einhält.
   Hinweis: Wie oft kann ein Element die Farbe wechseln?

Donnerstag

Aufgabe 4.5 (Splay-Bäume). Professor Sheldon schlägt die sogenannten Spiel-Bäume als eine einfachere Variante der Splay-Bäume vor. Hierbei verzichten wir innerhalb der *splay(x)*-Methode auf die nervigen *roller-coaster* und *zig-zag* Transformationen, und führen stattdessen nur einfache Rotationen durch—so lange, bis x an der Wurzel landet.

a) Was sind die amortisierten Kosten der Operation *splay(x)*, wenn wir nur einfache Rotationen benutzen? Analysiere die Kosten mit derselben Potenzialfunktion, die wir für Splay-Bäume verwendet haben.
b) Was sind die gesamten (tatsächlichen) Kosten, wenn man zuerst n Elemente mit den Schlüsseln $1, 2, 3, \dots, n$ der Reihenfolge nach in einen Spiel-Baum einfügt und dann in der gleichen Reihenfolge sucht?
c) Professor Sheldon behauptet, dass das Einfügen, Suchen und Löschen in Spiel-Bäumen amortisierte Kosten von $\mathcal{O}(\log n)$ hat. „Du musst nur eine raffiniertere Potenzialfunktion benutzen“, sagt er. Liegt er damit richtig?

Aufgabe 4.6 (Implementierung von dynamischen Tabellen). Implementiere deine eigene dynamische Tabelle für `Integer`-Werte, ohne dabei built-in Methoden zu verwenden. Deine dynamische Tabelle soll folgende Operationen beherrschen:

- Einfügen von Elementen,
- Löschen von Elementen,
- Ausgabe enthaltener Elemente,
- Ausgabe der Größe der Tabelle.

Aufgabe 4.7 (Dynamische Hashtabelle). Erkläre, wie man eine dynamische Hashtabelle mit Einfügungen unter Verwendung der Verdopplungsmethode verwaltet. Deine Lösung soll einen Speicherplatz von $\Theta(n)$ benutzen, wobei n die Anzahl der Elemente in der Hashtabelle ist. Wie viel Zeit benötigt deine Lösung zum Einfügen eines Elements in die Hashtabelle?

Aufgabe 4.8 (Deamortisierung von dynamischen Tabellen). Manchmal ist es möglich, Datenstrukturen zu deamortisieren. Das heißt, man erhält dieselben *worst-case* Schranken, wie im amortisierten Fall, indem die teure Arbeit auf viele Operationen verteilt wird und im Hintergrund ausgeführt wird.

a) Zeige, wie man dynamische Tabellen mit Einfügen (ohne Löschen) so implementieren kann, dass das Einfügen eines Elements im *worst-case* einen konstanten Zeitaufwand benötigt. Hierbei soll zu jedem Zeitpunkt weiterhin nur $\mathcal{O}(n)$ Speicherplatz benutzt werden, wo n die Anzahl der gespeicherten Elemente ist.
   Hinweis: Benutze die Verdopplungsmethode, aber verteile die Arbeit auf alle Einfügungen.

b) Jetzt mach dasselbe für dynamische Tabellen mit Einfügen *und* Löschen.

Aufgabe 4.9 (Puzzle der Woche: Prinzessinnen). Stelle dir vor, du bist ein junger Prinz aus dem fernen Lande Algo und der König des benachbarten Landes Logik hat 3 Töchter. Die Älteste erzählt immer die Wahrheit, die Jüngste lügt immer und die Mittlere lügt und sagt die Wahrheit, wie es ihr gefällt.

Du willst entweder die älteste oder die jüngste Tochter heiraten, da immer lügen genauso gut ist, wie immer die Wahrheit zu sagen. Nur die mittlere Tochter möchtest du nicht heiraten. Allerdings sehen alle Töchter gleich aus, sodass du sie nicht unterscheiden kannst.

Der König ist ein hinterhältiger Mann und erlaubt dir, genau *eine* Frage an genau *eine* der drei Töchter zu stellen. Diese Frage soll mit „Ja“ oder „Nein“ zu beantworten sein. Danach musst du dich entscheiden, welche der drei Prinzessinnen du heiraten möchtest.

Welche Frage solltest du stellen und dann welche der Töchter aussuchen?

Sternaufgabe

Aufgabe 4.10 (⭐: Queues mit zwei Stacks).
Wir erinnern uns, dass ein Stack `S` eine Datenstruktur ist, die die Operationen `S.PUSH(x)`, `S.POP()` und `S.ISEMPTY()` unterstützt. Wir nehmen an, dass diese Operationen in konstanter *worst-case* Zeit implementiert worden sind.

Du sollst für diese Aufgabe eine Queue `Q` mithilfe von zwei Stacks `S1`, `S2` implementieren. Dabei darfst du nur $\mathcal{O}(1)$ zusätzlichen Speicher benutzen. Der *einzige* Zugriff auf die Stacks erfolgt als *black-box* über die Standardoperationen `PUSH`, `POP` und `ISEMPTY`.

a) **Beschreibe mit Pseudocode** eine solche Queue-Implementierung, wobei die amortisierte worst-case Laufzeit für jede `Q.ENQUEUE(x)`, `Q.DEQUEUE()` und `Q.ISEMPTY()` Operation konstant sein muss.

b) **Beweise mithilfe der Potenzialmethode**, dass die amortisierten Kosten deiner Implementierung tatsächlich konstant sind.