Algorithmen und Datenstrukturen 1

ALGO1 · SoSe-2023 · tcs.uni-frankfurt.de/algo1/ · 2023-07-06 · *ed6968a*

Stapel, Warteschlangen, Verkettete Listen und Bäume (Woche 4)

Eigenständige Vorbereitung:
Lies 📖 CLRS Einleitung von Teil III und Kapitel 10, Kapitel 17.4 bis Mitte von 17.4.1 und schau dir das 📺 Video der Woche an. Beantworte dabei folgende Leitfragen:

a) Welche Operationen muss ein Stapel unterstützen und wie implementiert man sie?
b) Wann und warum sind dynamische Arrays nützlich, obwohl sie komplizierter als feste Arrays sind?

Zeichenlegende:

✏️ Schriftliche Aufgabe, die du fristgerecht in Moodle abgibst. In der Klausur wirst du alle Aufgaben schriftlich bearbeiten, daher ist das Feedback der Tutoren wichtig, damit du deine Schreibfähigkeiten verbessern kannst.
👍 Diese Art von Aufgabe musst du sicher können, um die Klausur zu bestehen.
🔑 Diese Art von Aufgabe musst du weitgehend können, um die Klausur zu bestehen.
🔭 Diese Art von Aufgabe musst du können, um eine gute Note zu erhalten.
🌈 Diese Aufgabe ist als Knobelspaß gedacht, der das algorithmische Verständnis vertieft.

Aufgabe 4.1 (Stapel und Warteschlangen). Löse die Teilaufgaben.

a) 👍 Betrachte einen anfangs leeren Stapel, auf dem die Operationen `PUSH(4)`, `PUSH(1)`, `PUSH(3)`, `POP()`, `PUSH(8)`, `POP()` ausgeführt werden. Wie sieht der Stapel nach jeder Operation aus, wenn dieser durch ein festes Feld der Länge 6 implementiert wurde?
b) 🔑 Wie können *zwei* Stapel S_1, S_2 auf *einem* festen Feld A der Länge N implementiert werden? Es darf hierbei zu keinem Überlauf kommen, es sei denn die Anzahl der Elemente in S_1 und S_2 ist größer als N. Die `PUSH` und `POP` Operationen sollen $O(1)$ Zeit benötigen.
c) 👍 Betrachte eine anfangs leere Warteschlange, auf der die Operationen `ENQUEUE(4)`, `ENQUEUE(1)`, `ENQUEUE(3)`, `DEQUEUE()`, `ENQUEUE(8)`, `DEQUEUE()` ausgeführt werden. Wie sieht die Warteschlange nach jeder Operation aus, wenn diese auf einem festen Feld der Länge 6 implementiert wurde?
d) 🔭 Wie können eine Warteschlange Q und ihre Operationen durch zwei Stapel S_1, S_2 implementiert werden? (Zusätzliche Felder oder Objekte sind nicht erlaubt.) Analysiere die Laufzeit jeder Operation.

Aufgabe 4.2 (Nahezu eine ehemalige dänische Klausuraufgabe 👍). Sei S ein Stapel. Führe die folgenden Operationen von links nach rechts aus: Ein Buchstabe i steht hierbei für S.`PUSH(`i`)` und $\star$ steht für S.`POP()`.

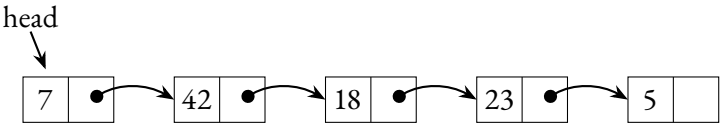
```
IFI*G**OE*THEU**NI
```

Gib die Sequenz der Buchstaben an, die durch die `POP`-Aufrufe ausgegeben werden.

Aufgabe 4.3 (Algorithmen auf verketteten Listen 👍). Betrachte die Algorithmen Foo und Bar und die verkettete Liste darunter.

```
procedure Foo(head)
    x ← head
    c ← 0
    while x ≠ null do
        x ← x.next
        c ← c + 1
    return c
```

```
procedure Bar(x, s)
    if x == null then return s
    else return Bar(x.next, s + x.key)
```



a) Führe Foo(head) händisch aus.
b) Erkläre was Foo berechnet.
c) Führe Bar(head, 0) händisch aus.
d) Erkläre was Bar berechnet.

Aufgabe 4.4 (Implementierung verketteter Listen 👍). Sei x ein Element in einer einfach verketteten Liste wie in den Folien beschrieben. Löse die folgenden Teilaufgaben. *Hinweis: Eine Zeichnung hilft.*

a) Sei x nicht das letzte Element in der Liste. Welchen Effekt hat der folgende Code-Schnipsel?
```
x.next = x.next.next;
```
b) Sei t ein neues Element, das noch nicht in der Liste enthalten ist. Welchen Effekt hat der folgende Code-Schnipsel?
```
t.next = x.next;
x.next = t;
```
c) Voraussetzungen wie in b). Hat der folgende Code-Schnipsel den gleichen Effekt? Wenn nein, welchen hat er?
```
x.next = t;
t.next = x.next;
```

Aufgabe 4.5 (Implementierung von Stapeln und Warteschlangen 👍). Implementiere die folgenden Datenstrukturen in einer Programmiersprache deiner Wahl.

a) Eine verkettete Liste. (Recherchiere z.B. im Internet, wie das in deiner Programmiersprache geht.)
b) Ein Stapel für Integer mittels einer einfach verketteten Liste. (Ohne Recherche.)
c) Eine Warteschlange für Integer mittels einer einfach verketteten Liste. (Ohne Recherche.)

Aufgabe 4.6 (Sortierte verkettete Listen 🔑). Sei L eine einfach verkettete Liste von n Integern in sortierter Reihenfolge. Löse die folgenden Aufgaben.

a) Entwirf einen Algorithmus, der einen neuen Integer in L einfügt, sodass die Liste danach noch sortiert ist.
b) Ein Freund schlägt vor, binäre Suche zu verwenden, um das Suchen in sortierten verketteten Listen zu beschleunigen. Wird das funktionieren? Begründe.

Aufgabe 4.7 (Eine Liste umkehren 🔑). Entwirf einen Algorithmus, der eine einfach verkettete Liste L mit n Elementen als Eingabe erhält und diese umkehrt, das heißt, nach der Ausführung soll die Liste L dieselben Elemente, jedoch in umgekehrter Reihenfolge enthalten. Der Algorithmus soll $O(n)$ Zeit brauchen und höchstens konstant viel zusätzlichen Speicherplatz benötigen.[1]

Aufgabe 4.8 (Lichtschalter im Gefängnis 🌈). 32 Inhaftierte sind zu lebenslänglicher Einzelhaft verurteilt. Der Gefängnisdirektor schlägt den Inhaftierten einen Deal vor, der zwar nicht mit Menschenrechten, jedoch mit einer algorithmischen Fragestellung vereinbar ist. Der Direktor hat eine Schüssel voller Zettel mit den Zellennummern aller Inhaftierten. Jeden Tag zieht der Direktor eine zufällige Nummer und lässt die inhaftierte Person aus der entsprechenden Zelle in das Vernehmungszimmer des Gefängnisses. Anschließend legt er die Zellennummer zurück in die Schüssel. Das Vernehmungszimmer ist leer, abgesehen von k Lichtschaltern. Diese Lichtschalter können von den Inhaftierten an- und ausgeschaltet werden. Der Direktor schlägt folgenden Deal vor: Im Vernehmungszimmer angekommen, darf sich eine inhaftierte Person dazu entscheiden, laut zu sagen, dass alle 32 Inhaftierte mindestens einmal im Zimmer gewesen sein müssen. Liegt die Person mit ihrer Aussage richtig, werden alle Inhaftierten freigelassen. Liegt die Person mit ihrer Aussage falsch, werden alle Inhaftierten hingerichtet. Die Inhaftierten dürfen sich zu Beginn einmal treffen, um sich eine Strategie zu überlegen, bevor sie wieder voneinander isoliert werden. Anfangs sind alle Lichtschalter ausgeschaltet.

a) Ist es möglich für $k = 32$ eine Strategie zu entwickeln, die mit 100%iger Sicherheit funktioniert?
b) Für $k = 5$?
c) (sehr schwer) Für $k = 1$?

Aufgabe 4.9 (Dynamische Felder und Stapel 🚀). Wir wollen Stapel durch dynamische Felder implementieren. Löse die folgenden Teilaufgaben.

a) Verallgemeinere die Implementierung durch dynamische Felder aus den Folien, sodass diese für Stapel sowohl `PUSH` als auch `POP` Operationen unterstützen. Wenn der Stapel anfangs leer ist und n beliebige `PUSH`/`POP` Operationen ausgeführt werden, soll die Gesamtlaufzeit dieser n Operationen $\Theta(n)$ betragen. Außerdem soll jede einzelne `PUSH` und `POP` Operation unabhängig vom Stapelinhalt maximale Laufzeit $O(n)$ haben.
b) (sehr schwer) Betrachte der Einfachheit halber nur die `PUSH` Operation. Zeige, wie man `PUSH` mithilfe dynamischer Felder so implementieren kann, dass die Laufzeit jeder einzelnen Operation $O(1)$ ist, also durch eine Konstante beschränkt. Hierbei soll der Speicherplatz linear in der Anzahl der auf dem Stapel enthaltenen Elemente sein. Wir nehmen für das Kostenmodell in dieser Aufgabe an, dass man ein Feld beliebiger Größe in Zeit $O(1)$ allokieren kann.[2] *Hinweis:* Überlege, wie der Aufwand gleichmäßig über alle Operationen verteilt werden kann.

[1] Das heißt, der zusätzliche Speicherbedarf muss durch eine Konstante (unabhängig von n) beschränkt bleiben, selbst dann, wenn n beliebig wächst.

[2] In C/C++ ist das einigermaßen realistisch, denn mit dem Codeschnipsel `int *array = malloc(n * sizeof(int))` sucht das Betriebssystem nach einem freien Speicherblock und allokiert diesen, ohne ihn zu durchlaufen. In Python stimmt das nicht, denn die ungefähr entsprechende Anweisung `array = [None for _ in range(n)]` läuft in linearer Zeit ab. Wenn Du Python gewohnt bist, kanst du dir für diese Aufgabe also vorstellen, dass diese Python-Anweisung konstante Zeit bräuchte. Beachte ansonsten, dass Du Python-Operationen wie `list.push(x)` oder Ähnliches nicht verwendest, da diese intern nicht unbedingt konstante Zeit brauchen.

Aufgabe 4.10 (Balance ✏️👍). Schreibe ein Programm in C/C++, Java, Python, oder einer anderen üblichen Programmiersprache (kein Pseudocode), welches das folgende Problem in linearer Zeit löst: Entscheide, ob ein gegebener String von zwei verschiedenen Arten von Klammern *balanciert* ist, das heißt, ob die zusammengehörigen öffnenden und schließenden Klammern richtig verschachtelt sind. Zum Beipiel ist `"([])()[]"` balanciert, aber `"(("` und `")("` und `"(]"` nicht.

Um präzise zu sein, hier ist die induktive Definition des Begriffes der Balanciertheit:

(i) der leere String ist balanciert;
(ii) wenn w balanciert ist, dann sind sowohl (w) als auch $[w]$ balanciert; und
(iii) wenn w und x balanciert sind, dann ist auch wx balanciert.

Eingabe. Die Eingabe besteht aus mehreren Zeilen. Jede Zeile ist eine Sequenz w, die nur Zeichen aus dem Alphabet $\{[,], (,)\}$ enthält.

Ausgabe. Für jede Zeile w, die balanciert ist, gib `"1"` aus, und sonst `"0"`.

Beispiel.

Eingabedatei:	Ausgabedatei:
`([(())])[]`	`1`
`)(`	`0`
`[)`	`0`
`((`	`0`
`[(])`	`0`
`[])[])`	`0`
`(`	`0`

Klassifizierte Testfälle. Teste dein Programm! Hier ist ein größeres Beispiel:

- https://files.tcs.uni-frankfurt.de/algo1/balance-example.in
- https://files.tcs.uni-frankfurt.de/algo1/balance-example.out

Stell sicher, dass dein Programm bei Eingabe `balance-example.in` *exakt* die Ausgabedatei `balance-example.out` erzeugt.

Hinweis. Erinnere dich an die Themen der Woche. Welche Datenstruktur eignet sich, um diese Aufgabe zu lösen? Nutze sie!

Hinweise zur Abgabe. Die Abgabe soll wie immer per PDF erfolgen und die grobe Idee, den diesmal echten Code, den Korrektheitsbeweis und die Laufzeitanalyse enthalten. Außerdem: Welche 20 Zeilen (jeweils 0 oder 1) werden bei Eingabe https://files.tcs.uni-frankfurt.de/algo1/balance-secret20.in erzeugt? Die Ausgabe ist in der PDF einzufügen. Weiterhin zu beachten:

- Der Code darf maximal 80 Zeilen lang sein. Möglichst kurz und elegant! (Eine 15-Zeilen Lösung in Python ist möglich. Kommentare zählen nicht dazu.)
- Benutz am Besten deine selbst geschriebene Datenstruktur aus Aufgabe 4.5. Wichtig: Die Datenstruktur darf nur so benutzt werden, wie die in der Vorlesung beschriebene abstrakte Datenstruktur das erlaubt. Falls zum Beispiel eine Warteschlange benutzt wird, dürfen nur die entsprechenden Funktionen enqueue, dequeue und is_empty verwendet werden.