Algorithmen und Datenstrukturen 1

ALGO1 · SoSe-2021 · tcs.uni-frankfurt.de/algo1

Wochenplan: Stapel, Warteschlangen, Verkettete Listen und Bäume

Revision 393abf7

(2021-06-08)

Vorbereitung

Lies CLRS Einleitung von Teil III und Kapitel 10, Kapitel 17.4 bis Mitte von 17.4.1, und schau das Video der Woche.

Dienstag

Aufgabe 1 (Stapel und Warteschlangen). Löse die Teilaufgaben.

a) (einfach) 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) (einfach) 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) (schwer) 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 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 `*` steht für S.`POP()`.

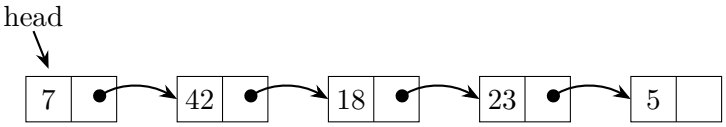
```
IFI*G**OE*THEU**NI
```

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

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

Aufgabe 4 (Implementierung verketteter Listen, einfach). Sei x ein Element in einer einfach verketteten Liste wie in den Folien beschrieben. Löse die folgenden Teilaufgaben.

a) Sei x nicht das letzte Element in der Liste. Worin resultiert der folgende Code-Schnipsel?
```
x.next = x.next.next;
```
b) Sei t ein neues Element, das noch nicht in der Liste enthalten ist. Worin resultiert der folgende Code-Schnipsel? `t.next = x.next;` `x.next = t;`
c) Voraussetzungen wie in b). Bringt der folgende Code-Schnipsel die gleichen Resultate? Wenn nein was sonst? `x.next = t;` `t.next = x.next;`

Aufgabe 5 (Implementierung von Stapeln und Warteschlangen). Implementiere die folgenden Datenstrukturen in einer Programmiersprache deiner Wahl.

a) Ein Stapel für Integer mittels einer einfach verketteten Liste.
b) Eine Warteschlange für Integer mittels einer einfach verketteten Liste.

Donnerstag

Aufgabe 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 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 8. 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 9 (Dynamische Felder und Stapel). Wir wollen Stapel durch dynamische Felder implementieren. Löse die folgenden Teilaufgaben.

a) (schwer) 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 Sie Python gewohnt sind, dürfen Sie sich für diese Aufgabe also vorstellen, dass diese Python-Anweisung konstante Zeit bräuchte. Beachten Sie ansonsten, dass Sie Python-Operationen wie `list.push(x)` oder Ähnliches nicht verwenden, da diese intern nicht unbedingt konstante Zeit brauchen.