UE DGA WS2020-2021

Übungsblatt 8

Aufgabe 43:

Ein bi-parental Heap (oder Beap) ist eine Datenstruktur, wo ein Knoten zwei Eltern hat (außer, es handelt sich um den ersten oder letzten Knoten auf einem Level) und zwei Kinder hat (außer, es ist ein Knoten am untersten Level).

a Sei n die Anzahl an Knoten in einem Beap. Wie groß ist die Höhe des Beaps in etwa? Wie viele Elemente befinden sich auf dem letzten Level (unter der Annahme, dass dieser voll ist)?

b Betrachten wir nun einen MAX-Beap, d.h. einen Beap, bei dem die Einträge jedes Knotens größergleich den Einträgen in den Kindern dieses Knotens sind. Formulieren Sie einen Pseudocode für die Prozedur MAX-BEAPIFY, wobei das Feld A und der Index i gegeben sind, die Teil-Beaps mit den Wurzeln LEFT[i] und RIGHT[i] MAX-Beaps sind, der Eintrag A[i] aber möglicherweise kleiner als die Einträge in den Kindern sein kann.

Aufgabe 44:

Wir können einen Heap bauen, indem wir wiederholt die in der Vorlesung kennengelernte Prozedur EUNFÜGEN aufrufen, um ein Element in den Heap einzufügen. Betrachten Sie die wie folgt geänderte ERZEUGE-HALDE-Prozedur:

ERZEUGE-HALDE'(A)
A.heap-size:=1
for $i = 2$ to A.länge **do**
  EUNFÜGEN($A, A[i]$)
end for

a Erzeugen die Prozeduren ERZEUGE-HALDE und ERZEUGE-HALDE' immer denselben Heap, wenn sie auf das gleiche Eingabefeld angewendet werden? (Beweis oder Gegenbeispiel)

b Zeigen Sie, dass ERZEUGE-HALDE' im worst case eine Laufzeit von $\Theta(n \log n)$ benötigt, um einen Heap mit n Elementen zu erzeugen.

Aufgabe 45:

Wir betrachten "Wechsel-Geld-Problem" (WGK). In einer Währung existieren (unlimitiert viele) Münzen im Wert von $1 = d_1 < d_2 < \cdots < d_k$ Cent. Ziel des WGK ist es, mit möglichst wenigen Münzen einen gegebenen Betrag von n Cent zu wechseln.

a Bestätigen Sie, dass das WGK die optimale Teilstruktur-Eigenschaft erfüllt, d.h., zeigen Sie, dass eine optimale Lösung für n Cent, d.h., $n = \sum_i c_i d_i$ und $\sum_i c_i$ ist minimal, auch eine optimale Lösung für b bzw. $n - b$ Cent darstellt, wenn b ein Anteil von n ist, welcher durch die in der Darstellung von n verwendeten Münzen zustande kommt, d.h., $b = \sum_i c'_i d_i$ und $c'_i \leq c_i$.

b Sei $A(n)$ die Anzahl der Münzen, die zum Wechseln des Betrags n mindestens notwendig sind. Überlegen Sie sich eine Rekursion für $A(n)$.

c Erklären Sie die Funktionsweise des folgenden Algorithmus WECHSEL(d, k, n), welcher für den Wert n und für ein gegebenes Münzen-Array $d = (d_1, \ldots, d_k)$ die optimale Wechsellösung liefert am Beispiel $(d_1, d_2, d_3) = (1, 2, 5)$ und $n = 8$.

```
WECHSEL(d, k, n)
seien C[0..n] und S[0..n] neue Felder
C[0] := 0
for j = 1 to n do
    C[j] := ∞
    for  i = 1 to k do
        if  d_i ≤ j and 1 + C[j − d_i] < C[j] then
            C[j] := 1 + C[j − d_i]
            S[j] := d_i
        end if
    end for
end for
return C und S
```

d Wie groß ist asymptotisch die Komplexität von WECHSEL(d, k, n)?

Aufgabe 46:

Das (diskrete) Rucksack-Problem. Sie haben schon wieder gewonnen! Diesmal dürfen Sie sich unter n Gegenständen (die Sie aber nicht zerteilen dürfen), wobei der i-te Gegenstand v_i Euro wert ist und w_i Kilo wiegt, so viele aussuchen und in Ihren Rucksack hineinpacken, so viel Sie tragen können; wir nehmen an, dass Sie höchstens W Kilo tragen können. Wir nehmen dabei weiters immer an, dass v_i, w_i und W positive ganze Zahlen sind. Die Aufgabe beim "Rucksack-Problem" besteht nun darin, anzugeben, welche Gegenstände Sie mitnehmen sollen, sodass der Gesamtwert der eingepackten Gegenstände möglichst groß ist.
Etwas genauer: Gesucht ist $f(n, W)$, wenn

$$f(m, W') = \max \left\{ \sum_{i=1}^{m} x_i v_i \;\middle|\; \sum_{i=1}^{m} x_i w_i \leq W' \text{ und } x_i \in \{0, 1\} \right\},$$

für $m \in \{1, 2, \ldots, n\}$ und $W' \in \{0, \ldots, W\}$.

a Überlegen Sie sich, dass das Rucksack-Problem die optimale Teilstruktur-Eigenschaft (wie lautet diese hier?) besitzt.

b Geben Sie eine Rekursion für $f(m, W')$, d.h., für den Wert von optimalen Teillösungen des ursprünglichen Problems, an.
Anmerkung: Diese Rekursion hängt nun anders als beim vorigen Beispiel von beiden Parametern m und W' ab.

c Man gebe einen Algorithmus an, der unter Zuhilfenahme von dynamischer Programmierung das Rucksack-Problem löst.

Aufgabe 47:

Nehmen Sie an, Sie wollen die Ausgaben 0 und 1 mit Wahrscheinlichkeit je $\frac{1}{2}$ erhalten. Dazu steht Ihnen die Prozedur BIASED-RANDOM zur Verfügung, die den Wert 1 mit Wahrscheinlichkeit p und den Wert 0 mit Wahrscheinlichkeit $1 - p$ ausgibt ($0 < p < 1$). Sie wissen aber nicht, wie groß p ist. Geben Sie einen Algorithmus (in Pseudocode) an, der BIASED-RANDOM als Unterroutine verwendet und den Wert 0 mit Wahrscheinlichkeit $\frac{1}{2}$ und den Wert 1 ebenfalls mit Wahrscheinlichkeit $\frac{1}{2}$ zurückgibt. Wie groß ist die erwartete Laufzeit ihres Algorithmus als Funktion von p?

Aufgabe 48:

Es gibt zwei Arten von randomisierten Algorithmen: Las-Vegas-Algorithmen nutzen randomisierten Input, um die erwartete Laufzeit zu verringern, produzieren aber immer ein korrektes Ergebnis (man gambelt mit der Laufzeit). Monte-Carlo-Algorithmen hingegen haben deterministische Laufzeit, produzieren aber mit einer gewissen Wahrscheinlichkeit ein falsches oder gar kein Ergebnis (insofern ist die Bezeichung "Algorithmus" auch etwas fragwürdig).
Betrachten wir nun das Problem, in einem Array A der Größe $2n$ bestehend aus n Einträgen mit Wert a und n Einträgen mit Wert b ein a zu finden, sowie einen Las-Vegas- und einen Monte-Carlo-Algorithmus, der dieses Problem löst:

Las-Vegas-Algorithmus:

```
LV-FIND-a(A)
while true do
    i = RANDOM(1, 2n)
    if A[i] = a then
        return i
    end if
end while
```

Dieser Algorithmus liefert immer ein richtiges Ergebnis. Bestimmen Sie seine erwartete Laufzeit.

Monte-Carlo-Algorithmus:

```
MC-FIND-a(A,k)
i=0
while i < k do
    i = RANDOM(1, 2n)
    if A[i] = a then
        return i
    end if
end while
```

Dieser Algorithmus braucht höchstens k Schritte (k fest), hat also konstante Laufzeit. Bestimmen Sie die Wahrscheinlichkeit, dass nach k Durchläufen ein a gefunden wurde.