Diskrete und geometrische Algorithmen
Übung 4

30. Oktober 2023

1. Es gibt n Personen, die einen Raum betreten und nach einer Weile wieder verlassen. Für jedes $i \in \{1, \ldots, n\}$ betritt Person i den Raum zur Zeit a_i und verlässt den Raum zur Zeit b_i (wobei $a_i < b_i$). Alle a_i und b_i seien verschieden. Zu Beginn des Tages ist das Licht im Raum ausgeschalten. Die erste Person, die den Raum betritt, schaltet das Licht ein. Um Strom zu sparen, schaltet Person i beim Verlassen des Raums zur Zeit b_i das Licht aus, wenn sonst niemand im Raum ist. Die nächste Person, die den Raum betritt, schaltet das Licht wieder ein.
   Wir wollen nun für gegebene Werte $(a_1, b_1), \ldots, (a_n, b_n)$ herausfinden, wie oft das Licht eingeschaltet wird. Geben Sie einen Algorithmus mit Laufzeit $O(n \log n)$ an, der berechnet, wie oft das Licht eingeschaltet wird.

2. 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?

3. Wir wollen in einem Array A der Größe $2n$, bestehend aus n Einträgen mit Wert a und n Einträgen mit Wert b, ein a finden. Dazu betrachten wir zwei Algorithmen:

   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)
j=0
while j < k do
    i = RANDOM(1, 2n)
    if A[i] = a then
        return i
    end if
    j = j+1
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.

4. Betrachte die Operation HEAP-DELETE(A, i), welche das Element im Knoten i aus dem Heap A löscht. Geben Sie eine Implementierung von HEAP-DELETE an, die für einen Heap mit n Elementen die Zeit $O(\log n)$ benötigt.

5. Wir können einen Heap bauen, indem wir wiederholt die in der Vorlesung kennengelernte Prozedur MAX-HEAP-INSERT aufrufen, um ein Element in den Heap einzufügen. Betrachten Sie die wie folgt geänderte BUILD-MAX-HEAP-Prozedur:

```
BUILD-MAX-HEAP'(A)
A.heap-size:=1
for i = 2 to A.länge do
    MAX-HEAP-INSERT(A, A[i])
end for
```

   (a) Erzeugen die Prozeduren BUILD-MAX-HEAP und BUILD-MAX-HEAP' immer denselben Heap, wenn sie auf das gleiche Eingabefeld angewendet werden? (Beweis oder Gegenbeispiel)
   (b) Zeigen Sie, dass BUILD-MAX-HEAP' im worst case eine Laufzeit von $\Theta(n \log n)$ benötigt, um einen Heap mit n Elementen zu erzeugen.

6. Geben Sie einen Algorithmus an, der Zeit $O(n \log k)$ benötigt, um k sortierte Listen zu einer einzigen sortierten Liste zusammenzufügen, wobei n die Gesamtanzahl aller Elemente in allen Input-Listen ist.
   Hinweis: Benutzen Sie einen Min-Heap für das k-fache Zusammenfügen.