Datenstrukturen und Algorithmen: Übungsblatt #1

Abgabe am 19. April 2018

Paul Orschau (381085)

15.04.2018

Aufgabe 2

Zeigen Sie, dass die folgenden Aussagen für beliebige $n \in \mathbb{N}^{>0}$ gelten:

1. $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$
2. $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

Lösung

Teil 1

$A(n) \Leftrightarrow \sum_{k=1}^{n} k = \frac{n(n+1)}{2}$

Beweis von $A(n)$ per vollständiger Induktion.

Induktionsanfang: $n_0 = 1$

$$A(1) \Leftrightarrow \sum_{k=1}^{1} k = 1 = \frac{2}{2} = \frac{1(1+1)}{2}$$

Induktionsvoraussetzung: Sei $n \in \mathbb{N}^{>0}$ beliebig, aber fest, und es gelte $A(n)$.

Induktionsschritt: $n \to n+1$

$$A(n+1) \Leftrightarrow \sum_{k=1}^{n+1} k = \sum_{k=1}^{n} k + (n+1) \overset{(I.V.)}{=} \frac{n(n+1)}{2} + (n+1) = \frac{n(n+1)+2n+2}{2} = \frac{n^2+3n+2}{2} = \frac{(n+1)(n+2))}{2}$$

Damit ist $A(n)$ für alle $n \in \mathbb{N}^{>0}$ bewiesen. □

Teil 2

$A(n) \Leftrightarrow \sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

Beweis von $A(n)$ per vollständiger Induktion.

Induktionsanfang: $n_0 = 1$

$$A(1) \Leftrightarrow \sum_{i=0}^{1} 2^i = 1 + 2 = 3 = 4 - 1 = 2^2 - 1 = 2^{1+1} - 1$$

Induktionsvoraussetzung: Sei $n \in \mathbb{N}^{>0}$ beliebig, aber fest, und es gelte $A(n)$.

Induktionsschritt: $n \to n+1$

$$A(n+1) \Leftrightarrow \sum_{i=0}^{n+1} 2^i = \sum_{i=0}^{n} 2^i + 2^{n+1} \overset{(I.V.)}{=} (2^{n+1} - 1) + 2^{n+1} = 2 * (2^{n+1}) - 1 = 2^{n+2} - 1$$

Damit ist $A(n)$ für alle $n \in \mathbb{N}^{>0}$ bewiesen. □

Aufgabe 3

Sei R eine beliebige irreflexive und transitive Relation über einer endlichen Menge X. Zeigen oder widerlegen Sie folgende Aussagen immer gelten:

1. R ist die leere Relation.
2. R ist symmetrisch.
3. R ist anti-symmetrisch.

Lösung

R irreflexiv $\Leftrightarrow \forall x \in X : \neg(xRx)$
R transitiv $\Leftrightarrow \forall x, y, z \in X : (xRy) \wedge (yRz) \implies (xRz)$
Nehme als Beispiel für eine irreflexive und transitive Relation die "Größer als" Relation $>$ und als Beispiel für eine endliche Menge $X = \{1, 2\}$.

Teil 1

Falsch, denn $>$ erfüllt die Anforderungen an R, es handelt sich aber nicht um die leere Relation. □

Teil 2

Falsch, denn $>$ ist nicht symmetrisch, da $\exists x = 1, y = 2 \in X : (x > y) \not\Leftrightarrow (y > x)$ □

Teil 3

Beweis durch Widerspruch.
Seien R und X nun wieder beliebig.
Widerspruchsannahme: R ist nicht anti-symmetrisch
$\Rightarrow \exists x, y \in X : (xRy) \wedge (yRx) \wedge (x \neq y) \overset{R \text{ transitiv}}{\Rightarrow} xRx$
Dies widerspricht der Voraussetzung, dass R irreflexiv ist!
Damit ist die Widerspruchsannahme falsch und somit gezeigt, dass R anti-symmetrisch sein muss. □

Aufgabe 4

Gegeben sei eine Liste L der Länge $n > 2$, mit natürlichen Zahlen als Eintrag.

- L.first beschreibt den ersten Eintrag.
- L.last beschreibt den letzten Eintrag.
- Jeder Eintrag der Liste enthält eine Referenz auf den vorherigen Eintrag, eine Referenz auf den nächsten Eintrag und den Wert des Eintrags.

Schreiben Sie einen Algorithmus, der die zweitgrößte Zahl in der Liste bestimmt, unter Beachtung der folgenden Einschränkung:

1. Der Algorithmus soll iterativ sein (keine Rekursion erlaubt).
2. Der Algorithmus soll rekursiv sein (keine Schleifen erlaubt).

Lösung

Die Lösungen sind in Pseudo-code geschrieben.
Die Idee besteht darin, die Liste zunächst komplett zu sortieren.

Teil 1

```
1  program FindSecondGreatest
2    // Gegeben: Liste L
3    newL = L; // Kopie der Liste, die sortiert wird
4    for (int m=n;m>1;m--) {
5      for (int i=0;i<m-1;i=i+1) {
6        a = newL.first;
7        for (int j=0;j<i;j++) a = a.next;
8        if (a.value < a.next.value) {
9          // Elemente vertauschen
10         b = a;
11         a = a.next;
12         a.next = b;
13       }
14     }
15   }
16   secondGreatest = newL.first.next;
17 end program
```

Nun enthält secondGreatest die geforderte Referenz auf die zweitgrößte Zahl in der Liste.

Aufgabe 4 wird auf nächster Seite fortgesetzt...

Teil 2

```
1  function sorted(first) returns element
2    if (first.next == 0) {
3      // Nur ein Element, also ist Liste bereits sortiert!
4      return first;
5    } else {
6      // Es gibt mindestens zwei Elemente
7      if (first.value < sorted(first.next).value) {
8        // Erstes Element ist kleiner als groesstes Element,
9        // muss also nach hinten wandern
10       b = first;
11       first = first.next;
12       first.next = b;
13       sorted(first.next);
14     }
15     return first;
16   }
17 end function
18
19 program FindSecondGreatest
20   // Gegeben: Liste L
21   newL = L; // Kopie der Liste, die sortiert wird
22   sorted(newL.first);
23   secondGreatest = newL.first.next;
24 end program
```

Nun enthält secondGreatest die geforderte Referenz auf die zweitgrößte Zahl in der Liste.