

Datenstrukturen und Algorithmen: Hausübung 1

Felix Schrader, Jens Duffert, Eduard Sauter

19. Oktober 2015

Aufgabe 1.1

- a) Das folgende Programm berechnet den Binomialkoeffizienten rekursiv:

```
1 // Funktion definieren:
2 function bina(n, k) {
3   // Ergebnisse fuer k=0 und n=k eins setzen, weil bin(n,0)=bin
      (n,n)=1.
4   if(k == 0) return 1;
5   if(n == k) return 1;
6   // Ansonsten die Rekursionsformel benutzen:
7   return bina(n-1, k-1) + bina(n-1, k);
8 }
```

- b) Da die einzigen Startwerte immer eins sind und immer addiert werden um den Binomialkoeffizienten zu berechnen, dient das Ergebnis selber als Counter. Wenn das Programm den Koeffizienten berechnet, so sieht die Formel effektiv so aus:

$$1 + 1 + 1 + \dots = \binom{n}{k}$$

Man kann leicht erkennen, das dafür also

$$N_{Add} = \binom{n}{k} - 1 \quad (1)$$

Additionen nötig sind.

- c) Das folgende Programm verwendet bei der Berechnung Memoisation:

```
1 // Funktion definieren:
2 function binc(n, k) {
3   // (n kreuz k)-Matrix erstellen (alle Elemente sind
      undefined). Darin sollen bekannte Ergebnisse gespeichert
      werden.
```

```

4     B = new Array(n);
5     for(i = 0; i <= n; i++) {
6         B[i] = new Array(k);
7     }
8     // Eintraege mit k=0 und n=k eins setzen:
9     for(nIndex = 0; nIndex <= n; nIndex++) {
10        for(kIndex = 0; kIndex <= k; kIndex++) {
11            if(kIndex == 0) B[nIndex][kIndex]=1;
12            if(nIndex == kIndex) B[nIndex][kIndex]=1;
13        }
14    }
15    // Funktion, die den Binomialkoeffizienten mit Memoisation
16    berechnet, aufrufen:
17    return memobin(n, k, B)
18 }
19 // memobin definieren:
20 function memobin(n, k, B) {
21     // Wenn B[n][k] schon bekannt ist (also typeof(B[n][k])
22     nicht mehr "undefined" ist, soll dieser Wert ausgegeben
23     und die Funktion verlassen werden:
24     if(typeof(B[n][k]) == "number") return B[n][k];
25     // Ansonsten den Wert mit der Rekursionsformel bestimmen
26     und als b speichern:
27     b = memobin(n-1, k-1, B) + memobin(n-1, k, B);
28     // Diesen Wert fuer evtl. spaetere Berechnungen in die
29     Matrix eintragen und als Ergebnis ausgeben:
30     B[n][k] = b;
31     return b;
32 }

```

Aufgabe 1.2

- a) Das folgende Programm berechnet die Quadratwurzel einer positiven Zahl x nach dem Divide-and-Conquer Paradigma.

```

1 function SqrtBisect(x, eps){
2     return SqrtBisectRecurse(x, 0, x + 1, eps * eps);
3 }
4
5 function SqrtBisectRecurse(x, a, b, eps){
6     guess = (b+a)/2;
7     square = guess*guess;
8     if (abs(x - square) < eps)
9         return guess;
10    else if (square < x)
11        return SqrtBisectRecurse(x, guess, b, eps);
12    else

```

```

13         return SqrtBisectRecurse(x, a, guess, eps);
14     }

```

Um die Korrektheit zu zeigen, wird vorausgesetzt ,dass

$$a < b \iff a^2 < b^2 \quad \forall a, b \geq 0 \quad (2)$$

gilt.

SqrtBisect sucht \sqrt{x} im Intervall $[0, x + 1]$. Die Quadratwurzel muss in diesem Bereich liegen, da $a \times a > a \quad \forall a \geq 1$ und $a \times a \leq 1 \quad \forall 0 \leq a \leq 1$. Zu Beginn eines Funktionsaufrufes sei also sichergestellt, dass $\sqrt{x} \in [a, b]$. Aufgrund der oberen Gleichung und Zeilen 10-13 bleibt dies gewährleistet. Die Abbruchbedingung muss somit erreicht werden, da das Suchintervall nach jedem Schritt halbiert wird. An dieser Stelle ist

$$|x - \text{guess}^2| < \varepsilon^2 \quad (3)$$

Es bleibt noch zu zeigen, dass daraus folgt $|\sqrt{x} - \text{guess}| < \varepsilon$.

Dies ergibt sich aus

$$|x - a|^2 \leq |x^2 - a^2|$$

Beweis

(a) Fall $x \geq a$

$$|x - a|^2 = x^2 + a^2 - 2ax \leq x^2 + a^2 - 2a^2 = x^2 - a^2 = |x^2 - a^2|$$

(b) Fall $x < a$

$$|x - a|^2 = x^2 + a^2 - 2ax \leq x^2 + a^2 - 2x^2 = a^2 - x^2 = |x^2 - a^2|$$

Damit ist $|\text{SqrtBisect}(x) - \sqrt{x}| < \varepsilon$.

- b) Im schlimmsten Fall erreicht $b - a$ den Wert von ε^2 . Dies geschieht, falls gleich zu Beginn $a = \sqrt{x} = 0$. Da nach jedem Schritt $b - a \rightarrow (b - a)/2$ ist nach dem n -ten Schritt für den Startwert von $b - a = x + 1$ der Wert $\frac{x+1}{2^n}$. Man erhält als Lösung:

$$\frac{x+1}{2^n} = \varepsilon \implies n = \lg \frac{x+1}{\varepsilon}$$

Aufgabe 1.3

Nimmt man z.B. die Liste $\{1, 19, 21, 22\}$, so arbeitet die Coin-Changing-Methode mit dem Greedy-Algorithmus für die Zerlegung der Zahl 42 nicht optimal. Würde man diesen Algorithmus verwenden, würde zunächst die 22 aus der Liste ausgewählt. Der Rest sind 20. Also würde im nächsten Schritt die 19 ausgewählt. Der Rest 1 wird durch die 1 aufgefüllt. Man braucht also 3 Elemente der Liste.

Optimal wäre allerdings die Zerlegung $42 = 21 + 21$, da dort nur 2 Elemente benötigt werden. Diese Zerlegung findet der Greedy-Algorithmus aber nicht.