# DAP2 – Heimübung 11

Ausgabedatum: 23. 6. 17 — Abgabedatum: Fr. 30. 6. 17 (Mo. 3. 7. für Gruppen 27–32) 12 Uhr

Schreiben Sie unbedingt immer Ihren **vollständigen Namen**, **Ihre Matrikelnummer** und **Ihre Gruppennummer** auf Ihre Abgaben!

**Aufgabe 11.1 (4 Punkte):** (Streaming-Algorithmen)

In der Vorlesung (Foliensatz 14, Folien 3 und 4) wird das Problem der fehlenden Zahl vorgestellt: Das Universum $U$ ist gegeben als die Menge $\{1, \ldots, N\}$ der ersten $N$ natürlichen Zahlen. Im Datenstrom kommt jedes Element des Universums genau ein mal vor, bis auf das gesuchte Element $a$, das im Datenstrom fehlt. Ein Streaming-Algorithmus soll dieses Element mit $\mathcal{O}(\log N)$ Speicherplatz finden.

Gelöst wird die Aufgabe, indem man alle Elemente des Datenstroms aufaddiert. Sei $S = (N(N+1)/2)$ die Summe der ersten $N$ natürlichen Zahlen und $X$ die Summe der Elemente, die im Datenstrom vorhanden sind. Ein Streaming-Algorithmus kann diese Summe $X$ unter Verwendung lediglich einer Variable berechnen, indem er all diese Elemente aus dem Datenstrom aufaddiert. Das fehlende Element $a$ lässt sich dann angeben als $a = S - X$. Die Zahl $X \in \mathcal{O}(N^2)$ kann mit $\mathcal{O}(\log N)$ Bits gespeichert werden, da $X < S \leq N^2$ und damit $\lceil \log X \rceil \leq \log S + 1 \leq 2 \log N + 1$ gilt.

Wir modifizieren das Problem nun und setzen voraus, dass *genau zwei* Elemente $a, b \in U$ im Datenstrom fehlen. Ihre Aufgabe ist es, mit einem Streaming-Algorithmus diese beiden Elemente zu finden und dabei die Platzschranke $\mathcal{O}(\log N)$ beizubehalten.

a) Überlegen Sie, warum es nun nicht mehr ausreicht, lediglich die Summe der Elemente aus dem Datenstrom aufzusummieren.

b) Beschreiben Sie, wie man mit Hilfe einer *zweiten* Variable das Problem lösen kann. Es muss *nicht* explizit ein Algorithmus in Pseudocode formuliert werden, für jeden verwendeten Wert muss jedoch klar sein, wie dieser berechnet werden kann.

c) Beweisen Sie, dass der von ihnen angegebene Algorithmus die Platzschranke $\mathcal{O}(\log N)$ beibehält.

**Lösung:**

a) Gegeben die Summe $X = S - (a+b)$ der Elemente des Stroms, können anhand des Wertes $a + b$ nicht die einzelnen Elemente $a$ und $b$ bestimmt werden.

b) Die Lösung ist nun neben der Summe $X_1$ der Datenstromelemente zusätzlich die Summe $X_2$ ihrer Quadrate zu speichern. Hat ein Algorithmus die Werte $X_1, X_2$ berechnet, so ergibt das folgende Gleichungssystem mit $S_1 = \left(\frac{N(N+1)}{2}\right)$ als Summe der ersten $N$ natürlichen Zahlen und $S_2 = \left(\frac{N(N+1)(2N+1)}{6}\right)$ als Summe ihrer Quadrate:

$$a + b = S_1 - X_1, \quad a^2 + b^2 = S_2 - X_2.$$

Für die Lesbarkeit setzen wir $x = S_1 - X_1$ und $y = S_2 - X_2$. Die erste Gleichung wird nach $b$ aufgelöst und in die erste eingesetzt:

$$a^2 + (x - a)^2 = y, \quad \text{umgeformt: } 2a^2 + x^2 - 2ax - y = 0$$

Anwenden der $p$-$q$-Formel für die quadratische Gleichung führt zu:

$$a_{1,2} = \frac{x}{2} \pm \sqrt{\frac{x^2}{4} + \frac{y - x^2}{2}} = \frac{x \pm \sqrt{2y - x^2}}{2}.$$

Mit $b = x - a$ folgt:

$$b_{1,2} = x - \frac{x \pm \sqrt{2y - x^2}}{2} = \frac{x \mp \sqrt{2y - x^2}}{2}.$$

Die zwei gesuchten Zahlen sind also:

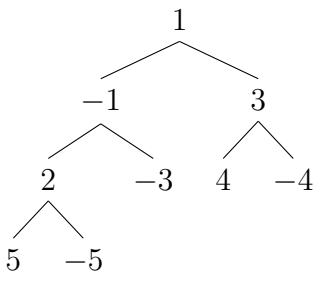
$$a = \frac{x + \sqrt{2y - x^2}}{2} \quad \text{und} \quad b = \frac{x - \sqrt{2y - x^2}}{2}.$$

c) Unser Algorithmus legt zusätzlich eine Variable für die Summe der quadrierten Datenstromelemente an. Diese Variable wird keinen größeren Wert annehmen als die Gesamtsumme $S_2 < N \cdot N^2$ der ersten $N$ Quadratzahlen. Speichern lässt sie sich also mit $\lceil \log N^3 \rceil \in \mathcal{O}(\log N)$ Bits. Für die Variablen $x < 2 \cdot N$ und $y < 2 \cdot N^2$, die der Algorithmus anlegt, um $a$ und $b$ zu berechnen, kann genauso argumentiert werden.

**Aufgabe 11.2 (6 Punkte + 4 Bonuspunkte):** (Teile und Herrsche im Baum II)

Im Folgenden betrachten wir Binärbäume, deren Schlüsseleinträge unterschiedliche ganze Zahlen sind. In einem gegebenen binären Baum $T$ gibt es für zwei Knoten $a_0$ und $a_m$ in $T$ einen eindeutigen Pfad $(a_0, a_1, \ldots, a_m)$ der Länge $m$ von $a_0$ nach $a_m$ (Die Länge ist also die Anzahl der Kanten). Wir betrachten solche Pfade, die aufsteigend sortiert sind, d. h., für deren Schlüssel $S(a_{i-1}) < S(a_i)$, $1 \leq i \leq m$, gilt. Wir suchen die Länge $m$ eines längsten solchen Pfades.

**a) (1 Punkt)** Bestimmen Sie die Länge eines längsten aufsteigend sortierten Pfades im folgenden Baum und geben Sie die Schlüssel der zugehörigen Start- und Zielknoten an.



**b) (5 Punkte)** Entwerfen Sie einen Teile-und-Herrsche-Algorithmus, der bei Eingabe eines binären Baumes $T$ die Länge eines längsten aufsteigend sortierten Pfades in $T$ bestimmt, und beschreiben Sie ihn mit eigenen Worten. Geben Sie eine Implementierung Ihres Algorithmus in Pseudocode an.

**c) (2 Bonuspunkte)** Analysieren Sie die Laufzeit Ihres Algorithmus. Stellen Sie hierzu eine Rekursionsgleichung für die Laufzeit Ihres Algorithmus auf und lösen Sie diese.

**d) (2 Bonuspunkte)** Zeigen Sie die Korrektheit Ihres Algorithmus.

**Lösung:**

a) Der längste aufsteigen sortierte Pfad hat Länge 4, nämlich beginnend im Knoten mit Schlüssel $-3$ und endend im Knoten mit Schlüssel 4.

b) Es bezeichne wieder $T(w)$ den Teilbaum von $T$ mit Wurzel $w$. Ähnlich wie in Präsenzaufgabe 9.1 entwerfen wir einen Teile-und-Herrsche-Algorithmus, der ausgehend von der Wurzel rekursiv jeweils den linken und rechten Teilbaum aufruft.

Ein längster aufsteigender Pfad setzt sich aus folgenden Fällen zusammen:

- er befindet sich im linken Teilbaum,
- er befindet sich im rechten Teilbaum oder
- er enthält die Wurzel.

Im letzten Fall müssen wir unterscheiden, ob der Schlüssel der Würzel $w$ größer oder kleiner als die Schlüssel seiner Kinder ist. Abhängig davon kann der Pfad entweder in $w$ beginnen, enden, von links nach rechts über $w$ laufen, von rechts nach links über $w$ laufen oder nur aus $w$ bestehen. Insbesondere kann $w$ nur einen Pfad fortsetzen, der im jeweiligen Kind beginnt oder endet. Daher bestimmen wir zunächst die Länge $b(w)$ des längsten aufteigenden Pfads, der in $w$ beginnt, und die Länge $e(w)$ des längsten aufteigenden Pfads, der in $w$ endet:

$$b(w) = \begin{cases} 1 + \max\{b(\mathrm{lc}[w]), b(\mathrm{rc}[w])\} & S(w) < S(\mathrm{lc}[w]) \wedge S(w) < S(\mathrm{rc}[w]) \\ 1 + b(\mathrm{lc}[w]) & S(w) < S(\mathrm{lc}[w]) \wedge S(w) > S(\mathrm{rc}[w]) \\ 1 + b(\mathrm{rc}[w]) & S(w) > S(\mathrm{lc}[w]) \wedge S(w) < S(\mathrm{rc}[w]) \\ 0 & S(w) > S(\mathrm{lc}[w]) \wedge S(w) > S(\mathrm{rc}[w]), \end{cases}$$

$$e(w) = \begin{cases} 0 & S(w) < S(\mathrm{lc}[w]) \wedge S(w) < S(\mathrm{rc}[w]) \\ 1 + e(\mathrm{rc}[w]) & S(w) < S(\mathrm{lc}[w]) \wedge S(w) > S(\mathrm{rc}[w]) \\ 1 + e(\mathrm{lc}[w]) & S(w) > S(\mathrm{lc}[w]) \wedge S(w) < S(\mathrm{rc}[w]) \\ 1 + \max\{e(\mathrm{lc}[w]), e(\mathrm{rc}[w])\} & S(w) > S(\mathrm{lc}[w]) \wedge S(w) > S(\mathrm{rc}[w]). \end{cases}$$

Die Maximale Länge setzt sich also zusammen aus

$$m(w) = \max\{m(\mathrm{lc}[w]), m(\mathrm{rc}[w]), k(w)\},$$

wobei für $k(w)$ folgende Kombinationen möglich sind:

$$k(w) = \begin{cases} \max\{b(w), e(w)\} & S(w) < S(\text{lc}[w]) \wedge S(w) < S(\text{rc}[w]) \\ 2 + b(\text{lc[w]}) + e(\text{rc[w]}) & S(w) < S(\text{lc}[w]) \wedge S(w) > S(\text{rc}[w]) \\ 2 + b(\text{rc[w]}) + e(\text{lc[w]}) & S(w) > S(\text{lc}[w]) \wedge S(w) < S(\text{rc}[w]) \\ \max\{b(w), e(w)\} & S(w) > S(\text{lc}[w]) \wedge S(w) > S(\text{rc}[w]). \end{cases}$$

Die Rekursion bricht etwa dann ab, falls ein Baum aus nur einem Knoten $w$ besteht, die Wurzel also gleichzeitig ein Blatt ist. Dann gilt $m(w) = b(w) = e(w) = 0$.

Wird ein leerer Baum aufgerufen, z.B., falls $w$ nur eine Kind hat, bezeichnen wir die Werte mit $-\infty$, sodass sich das Maximum nur aus den jeweils anderen Werten zusammensetzt.

Das ergibt folgenden Pseudocode.

Zunächst rufen wir eine Hilfsfunktion auf:

```
LIPStart(Baum T):
1  w ← Wurzel von T
2  (b(w), e(w), m(w)) ← LIP(T, w)
3  return m(w)
```

```
LIP(Baum T, Wurzel w):
1  if lc[w] ≠ nil then
2      (b_ℓ, e_ℓ, m_ℓ) ← LIP(T, lc[w]))
3  else
4      (b_ℓ, e_ℓ, m_ℓ) ← (−∞, −∞, −∞)                    /* unmögliche Ergebnisse */
5  if rc[w] ≠ nil then
6      (b_r, e_r, m_r) ← LIP(T, rc[w])
7  else
8      (b_r, e_r, m_r) ← (−∞, −∞, −∞)
           /* Rekursive Ergebnisse wurden im linken/rechten Unterbaum berechnet */
9  if (S(w) < S(lc[w]) ∨ lc[w] = nil) ∧ (S(w) < S(rc[w]) ∨ rc[w] = nil) then
10     b ← 1 + max{b_ℓ, b_r}; e ← 0;
11 else if (S(w) > S(lc[w]) ∨ (lc[w] = nil) ∧ (S(w) > S(rc[w]) ∨ rc[w] = nil) then
12     b ← 0; e ← 1 + max{e_ℓ, e_r}
13 else if S(w) < S(lc[w]) ∧ S(w) > S(rc[w]) then
14     b ← 1 + b_ℓ; e ← 1 + e_r; k ← 2 + b_ℓ + e_r
15 else if S(w) > S(lc[w]) ∧ S(w) < S(rc[w]) then
16     b ← 1 + b_r; e ← 1 + e_ℓ; k ← 2 + b_r + e_ℓ
       /* Maxima für b(w) und e(w) wurden berechnet. Daraus setzt sich das folgende
    Maximum zusammen. */
17 if lc[w] = nil ∨ rc[w] = nil ∨ (S(w) < S(lc[w]) ∧ S(w) < S(rc[w])) ∨ (S(w) > S(lc[w]) ∧ S(w) >
    S(rc[w])) then
18     k ← max{b, e}
19 m ← max{m_ℓ, m_r, k}
20 return (b, e, m)
```

c) Die Laufzeit unseres Algorithmus kann für eine Eingabe eines Baums mit $n$ Knoten mit folgender Rekursionsgleichung beschrieben werden:

$$L(n) = \begin{cases} c & \text{falls } n = 0 \vee n = 1 \\ c + L(n') + L(n - n' - 1) & \text{falls } n > 1, \end{cases}$$

wobei $n'$ die Knotenanzahl im linken Teilbaum und $n-n'+1$ die Knotenanzahl im rechten Teilbaum beschreibt. Sowohl im Rekurstionsabbruch als auch im Rekursionsschritt haben alle weiteren Operationen im Worst-Case konstante Laufzeit. Die Konstante $c$ beschreibt die maximale Anzahl dieser Rechenschritte.

Da jeder Knoten genau einmal im Rekursionsabbruch kommt, und es ist konstante Laufzeit benötigt, um die Teilergebnisse zusammenzufassen, ist es leicht zu behaupten, dass die Lösung der rekursive Gleichung, die nicht mit der Einsatzmethode zu lösen ist (wegen unbekanntes $n'$), $L(n) \in \mathcal{O}(n)$ ist.

Vollständigkeitshalber beweisen wir folgende Aussage: es gilt $L(n) \leq 2cn + c$, für alle $n \geq 0$.

I.A. Für $n = 0$ gilt, dass $L(0) = c \leq 2c \cdot 0 + c = c$ ist. Für $n = 1$ gilt, dass $L(1) = c \leq 2c \cdot 1 + c = 3c$ ist.

I.V. Es gelte für alle $\hat{n} < n$, dass $L(\hat{n}) \leq 2c \cdot \hat{n} + c$ ist.

I.S. Da $0 \leq n' < n$ ist, gilt es dass $L(n) \overset{def.}{=} c + L(n') + L(n - n' - 1) \overset{I.V.}{\leq} c + (2c \cdot n' + c) + (2c \cdot (n - n' - 1) + c) = c + 2c \cdot n' + c + 2c \cdot n - 2c \cdot n' - 2c + c = 2c \cdot n + c$ ist, was wir beweisen wollten.

Eine Verständnisfrage für die Leser: warum funktioniert nicht ein solcher induktiver Beweis mit der Behauptung $L(n) \leq c \cdot n$?

d) Wir beweisen die folgende Behauptung per erweiterter Induktion über die Baumhöhe:

**Beh:** Der Algorithmus LIP$(T, w)$ berechnet für den Baum $T$ der Höhe $h$ und mit der Wurzel $w$ die Länge $m(w)$ des längsten aufsteigenden Pfades in $T(w)$ sowie die Länge $b(w)$ des längsten aufsteigenden Pfades, der in $w$ beginnt, sowie die Länge $e(w)$ des längsten aufsteigenden Pfades, der in $w$ endet.

Daraus folgt, dass LIPStart$(T)$ die Länge des insgesamt längsten aufsteigenden Pfades im gesamten Eingabebaum zurückgibt.

**Induktionsanfang:** Wenn $h = 0$ ist, gelten die Else-Fälle in Zeile 3 und Zeile 7, sodass in Zeile 10 $b$ auf $-\infty$ und $e = 0$ gesetzt werden. Wegen $k = 0$ (Zeile 18) wird $m$ in Zeile 22 auf 0 gesetzt und das Tupel $(0, 0, 0)$ zurückgegeben. Dies ist korrekt, da der längste aufsteigende Pfad sowohl in $w$ beginnt als auch anfängt und aus nur einem Knoten besteht, also Länge 0 hat.

**Induktionsvoraussetzung:** Die Aussage gelte für alle Höhen $h$ mit $0 \leq h \leq h_0$ für ein festes $h_0$.

**Induktionsvoraussetzung:** Wir zeigen die Aussage unter dieser Voraussetzung für $h = h_0 + 1 > 0$.

Falls beide Teilbäume von $w$ nicht leer sind, wurden in Zeilen 2 und 6 die Werte $b_\ell$, $e_\ell$ und $m_\ell$ sowie $b_r$, $e_r$ und $m_r$ rekursiv berechnet. Diese sind nach Induktionsvoraussetzung korrekt und jeweils nicht negativ, da beide Teilbäume eine Höhe von mindestens 0 und höchstens $h - 1 = h_0$ haben.

Die Werte $b$, $e$ und $k$ sind jeweils größer oder gleich 0 nach den Berechnungen in Zeilen 10, 12, 14, 16 und 18.

Angenommen es gibt einen aufsteigenden Pfad der Länge $\hat{b} > b$, der in $w$ beginnt. Wegen $b \geq 0$ enthält dieser Pfad mindestens eine Kante. Falls dies die Kante zum linken Teilbaum von $w$ ist, gilt also $S(w) < S(lc[w])$, d. h. $b$ wurde in Zeile 10 oder 14 berechnet, also ist $\hat{b} > 1 + b_\ell$. Damit hat der längste aufsteigende Pfad, der in $lc[w]$ endet, aber eine Länge von $\hat{b} - 1 > b_\ell$; ein Widerspruch dazu, dass $b_\ell$ das Maximum nach Induktionsvoraussetzung ist. Falls dies die Kante zum rechten Teilbaum von $w$ ist, gilt $S(w) < S(rc[w])$, d. h. $b$ wurde in Zeile 10 oder 16 berechnet, also ist $\hat{b} > 1 + b_r$; ein Widerspruch zur Korrektheit von $b_r$. Also ist $b$ die maximale Länge eines aufsteigenden Pfades.

Analog kann für $e$ für absteigende Pfade, also Pfade, die in $w$ enden, argumentiert werden.

Der Wert $k$, die maximale Länge eines Pfades, der die Wurzel enthält, setzt sich nun aus den Möglichkeiten $b$ und $e$ entweder über $w$ fortzusetzen (Zeilen 14 und 16) oder in $w$ zu beginnen oder in $w$ zu enden (Zeile 18) zusammen.

Da die maximale Länge sich entweder auf einen Pfad im linken Teilbaum oder im rechten Teilbaum oder durch die Wurzel bezieht, ist $m = \max\{m_\ell, m_r, k\}$ korrekt.

Falls einer der beiden Teilbäume von $w$ leer ist, wird entweder in Zeile 10 oder in Zeile 12, der Pfad aus dem jeweils anderen Teilbaum absteigend oder aufsteigend fortgesetzt. Für den nicht-leeren Teilbaum der Höhe $h_0$ werden $b_r$ und $e_r$, bzw. $e_\ell$ und $b_\ell$ nach Induktionsvoraussetzung korrekt berechnet. Also wird korrekt entweder $b$ oder $e$ inkrementiert und der jeweils andere Wert auf 0 gesetzt. Dies ist korrekt umgesetzt, da das Maximum aus $-\infty$ und einem nicht-negativen Wert letzterer ist. Wegen $k = \max\{b, e\}0$ (Zeile 18), wird auch $m$ in Zeile 19 korrekt aus den Werten $b$, $e$, $m_\ell$ und $m_r$ berechnet. Letztere beide Werte sind korrekt, da einer der beiden auf $-\infty$ gesetzt wurde und der andere ebenfalls nach Induktionsvoraussetzung korrekt ist.

Insgesamt sind also $b$, $e$ und $m$ korrekt berechnet (und nicht negativ) und werden in Zeile 20 zurückgegeben.