ETH Zürich
Institut für Theoretische Informatik
Prof. Dr. Angelika Steger, Dr. Johannes Lengler
Übungsleitung: Florian Meier

HS 2019

Algorithmen und Komplexität
Lösungsvorschlag zu Übungsblatt 9

Lösungsvorschlag zu Aufgabe 1

Unsere Implementation verwendet zwei Stacks S_1 und S_2. Für die Queue implementieren wir die Operationen ENQUEUE(x) und DEQUEUE$()$ wie folgt:

Algorithm 1 ENQUEUE(x)

```
S_1.PUSH(x)
```

Algorithm 2 DEQUEUE$()$

```
if S_2.isEmpty then
  while not S_1.isEmpty do
    S_2.PUSH(S_1.POP)
  end while
end if
return S_2.POP()
```

Korrektheit: Wir zeigen, dass DEQUEUE$()$ immer jenes Objekt y zurückgibt, welches unter allen Objekten in der Queue als erstes mit ENQUEUE(y) eingefügt wurde. Das ist so, da das oberste Objekt in S_2 immer dieses y ist und der Algorithmus immer das oberste Objekt aus S_2 zurückgibt.

Amortisierte Laufzeit: Wir definieren die Potentialfunktion $\phi(i)$ und die Funktion $t(i)$ wie folgt:

$$\phi(i) := 2 \cdot (\text{Anzahl Objekte in } S_1 \text{ nach den ersten } i \text{ Operationen})$$
$$t(i) := \text{Zeit, die während den ersten } i \text{ Operationen aufgewendet wurde.}$$

Wir nehmen an, dass die Operationen POP und PUSH jeweils nur einen Zeitschritt brauchen. Offensichtlich gilt $\phi(0) = t(0) = 0$ und $\phi(i)$ ist immer positiv. Wir zeigen per Induktion über i, dass $t(i) + \phi(i) \leq 3i$. (Dies ergibt die obere Schranke $t(i) \leq t(i) + \phi(i) \leq 3i$, da $\phi(i) \geq 0$.) Die Induktionsverankerung $i = 0$ ist trivial. Die Induktionshypothese ist $t(i-1) + \phi(i-1) \leq 3(i-1) = 3i - 3$. Falls nun die die i-te Operation ENQUEUE ist, finden wir

$$\begin{aligned} t(i) + \phi(i) &= \big(t(i-1) + 1\big) + \big(\phi(i-1) + 2\big) \\ &= \big(t(i-1) + \phi(i-1)\big) + 3 \leq 3i - 3 + 3 = 3i. \end{aligned}$$

Falls die i-te Operation DEQUEUE ist, unterscheiden wir, ob S_2 leer ist oder nicht. Wenn S_2 nicht leer ist, ergibt sich

$$\begin{aligned} t(i) + \phi(i) &= \big(t(i-1) + 1\big) + \phi(i-1) \\ &= \big(t(i-1) + \phi(i-1)\big) + 1 \leq 3i - 3 + 1 < 3i. \end{aligned}$$

Falls S_2 leer ist, müssen für jedes Objekt in S_1 genau 2 Operationen (ein POP von S_1 und ein PUSH auf S_2) ausgeführt werden. Am Ende wird ein zusätzliches POP auf S_2 aufgerufen. Gemäss Definition von ϕ ist die Anzahl benötigter Operationen genau $\phi(i-1) + 1$ und folglich gilt

$$\begin{aligned} t(i) + \phi(i) &= \big(t(i-1) + \phi(i-1) + 1\big) + 0 \\ &\leq 3i - 3 + 1 + 0 < 3i. \end{aligned}$$

Lösungsvorschlag zu Aufgabe 2

Wir verwenden Induktion. Der Induktionsanfang $n = 1$ und $n = 2$ ist nicht schwierig: einen „entarteten" Baum auf $n = 1$ Knoten erzeugen wir einfach durch Hinzufügen eines beliebigen Schlüssels in einen leeren Heap. Einen „entarteten" Baum mit $n = 2$ Knoten erhalten wir, indem wir zuerst 3 Knoten einfügen und dann EXTRACT-MIN(H) ausführen.

Angenommen, wir haben bereits einen entarteten Baum der Länge n und Wurzel x erzeugt. Wir führen nun folgende Operationen aus:

(i) Füge mit INSERT(H, a), INSERT(H, b), INSERT(H, c) drei neue Knoten mit Schlüsseln

$$\text{key}[a] < \text{key}[b] < \text{key}[c] < \text{key}[\min(H)]$$

ein. Alle drei Knoten werden in die Wurzelliste eingefügt; min[H] zeigt nun auf a.

(ii) Entferne mit EXTRACT-MIN(H) den Knoten a. Die Wurzelliste enthält nun die drei Knoten b, c und x. Da b und c beide Rang 0 haben und key$[b] <$ key$[c]$ gilt, wird mit CONSOLIDATE c an b angehängt. Nun sind noch b und x in der Wurzelliste, und beide haben Rang 1. Wegen key$[b] <$ key$[x]$ hängt CONSOLIDATE nun x (und damit den ganzen entarteten Baum der Länge n) an b an. b ist der einzige verbleibende Knoten in der Wurzelliste und hat Rang 2.

(iii) Weise mit DECREASE-KEY(H, c, k) dem Knoten c einen Wert $k <$ key$[b]$ zu. Dies führt dazu, dass c mit CUT(c) abgeschnitten und in die Wurzelliste eingefügt wird.

(iv) EXTRACT-MIN(H) entfernt nun c aus dem Heap. Zurück bleibt ein entarteter Baum der Länge $n + 1$ mit Wurzel b.

Beachten Sie dass dieser Induktionsschritt für $n = 2$ nicht funktioniert, da CONSOLIDATE im Schritt (ii) c an b anhängt und dann stoppt, da x Rang 0 hat. Wir *müssen* diesen Fall deshalb separat beweisen.

Lösungsvorschlag zu Aufgabe 3

Wir wollen einen Algorithmus mit dem Prinzip der dynamischen Programmierung konstruieren. Die folgenden Eigenschaften von längsten gemeinsamen Teilsequenzen sind die Basis dafür.

Seien $x^{(i)} = x_1 x_2 \dots x_i$ und $y^{(j)} = y_1 \dots y_j$ Präfixe von x bzw. y, und sei $z^{(k)} = z_1 \dots z_k$ eine längste gemeinsame Teilsequenz von $x^{(i)}$ und $y^{(j)}$. Dann gilt:

(1) Wenn $x_i = y_j$ gilt, dann gilt auch $x_i = y_j = z_k$, und $z^{(k-1)}$ ist eine längste gemeinsame Teilsequenz von $x^{(i-1)}$ und $y^{(j-1)}$.

(2) Wenn $x_i \neq y_j$ und $z_k \neq x_i$ gilt, dann ist $z^{(k)}$ eine längste gemeinsame Teilsequenz von $x^{(i-1)}$ und $y^{(j)}$.

(3) Wenn $x_i \neq y_j$ und $z_k \neq y_j$ gilt, dann ist $z^{(k)}$ eine längste gemeinsame Teilsequenz von $x^{(i)}$ und $y^{(j-1)}$.

Diese Eigenschaften sagen uns, dass wir unser ursprüngliches Problem rekursiv lösen können, indem wir jeweils ein oder zwei Unterprobleme lösen: Wenn $x_i = y_j$ gilt, müssen wir eine längste gemeinsame Teilsequenz von $x^{(i-1)}$ und $y^{(j-1)}$ finden. Wenn wir x_i an diese anhängen, erhalten wir eine längste gemeinsame Teilsequenz von $x^{(i)}$ und $y^{(j)}$. Falls $x_i \neq y_j$ gilt, müssen wir eine längste gemeinsame Teilsequenz von $x^{(i-1)}$ und $y^{(j)}$ sowie von $x^{(i)}$ und $y^{(j-1)}$ finden. Die längere von beiden ist auch eine längste gemeinsame Teilsequenz von $x^{(i)}$ und $y^{(j)}$.

Damit können wir eine Rekursion für dieses Problem aufstellen. Es bezeichne $\ell_{i,j}$ die Länge einer längsten gemeinsamen Teilsequenz von $x^{(i)}$ und $y^{(j)}$. Wir haben

$$\ell_{i,j} = \begin{cases} 0 & \text{falls } i = 0 \text{ oder } j = 0, \\ \ell_{i-1,j-1} + 1 & \text{falls } i,j > 0 \text{ und } x_i = y_j, \\ \max\{\ell_{i,j-1}, \ell_{i-1,j}\} & \text{falls } i,j > 0 \text{ und } x_i \neq y_j. \end{cases}$$

Wir können die $\ell_{i,j}$ also in "bottom up"-Weise gemäss dem Prinzip der dynamischen Programmierung bestimmen, indem wir die $n \times m$-Matrix $L = (\ell_{i,j})_{1 \leq i \leq n, 1 \leq j \leq m}$ Zeile für Zeile ausfüllen. Der Eintrag $\ell_{n,m}$ ist dann die Länge der längsten gemeinsamen Teilsequenz von x und y. Da der Wert von $\ell_{i,j}$ nur von x_i, x_j, $\ell_{i-1,j-1}$, $\ell_{i,j-1}$ und $\ell_{i-1,j}$ abhängt, können wir jeden Eintrag in konstanter Zeit berechnen. Also hat dieses Vorgehen Laufzeit $\mathcal{O}(nm)$.

Aus der Matrix L lässt sich auch eine längste gemeinsame Subsequenz von x und y herauslesen: Wir beginnen beim Matrixeintrag $\ell_{n,m}$ rechts unten und durchlaufen einen Pfad zum Eintrag $\ell_{1,1}$ links oben, wobei wir in jedem Schritt nach links, oben, oder schräg links oben gehen. Dabei können wir jeweils aus den Matrixeinträgen $\ell_{i,j}$ ablesen, "woher wir beim Berechnen gekommen sind".

Konkret verfahren wir für den Matrixeintrag $\ell_{i,j}$ wie folgt:

Solange es einen links oder oben angrenzenden Matrixeintrag $\ell_{i,j-1}$ bzw. $\ell_{i-1,j}$ von gleichem Wert gibt, gehen wir zu diesem Eintrag und tun nichts. Falls dies nicht möglich ist, gilt nach der rekursiven Berechnungsvorschrift $\ell_{i,j} = \ell_{i-1,j-1} + 1$. Wir geben das Zeichen $x_i = y_j = z_k$ aus und gehen zum Eintrag $\ell_{i-1,j-1}$ schräg links oben. Im Laufe dieser Durchquerung von L wird in Zeit $\mathcal{O}(m+n)$ eine längste gemeinsame Teilsequenz z von x und y rückwärts ausgegeben.

Algorithm 3 LÄNGSTE GEMEINSAME TEILSEQUENZ(Strings $x[1..m], y[1..n]$)

```
L[0..n, 0..m] := new Array of Integers
P[0..n, 0..m] := new Array of Characters and Pointers (char, (i, j))
for i = 0 to n do
   L[i, 0] := 0;      P[i, 0] := (nil, nil)
end for
for j = 0 to m do
   L[0, j] := 0;      P[0, j] := (nil, nil)
end for
for i = 1 to n do
   for j = 1 to m do
      if x[i] = y[j] then
         L[i, j] := L[i − 1, j − 1] + 1;      P[i, j] := (x[i], (i − 1, j − 1))
      else
         if L[i, j − 1] ≥ L[i − 1, j] then
            L[i, j] := L[i, j − 1];      P[i, j] := P[i, j − 1]
         else
            L[i, j] := L[i − 1, j];      P[i, j] := P[i − 1, j]
         end if
      end if
   end for
end for
```

Durch Verwenden von Pointern kann man die Zeit fürs Auslesen auf $\mathcal{O}(h)$ verbessern, wobei h die Länge von z bezeichnet: Zu den Matrixeinträgen $\ell_{i,j}$ speichert man jeweils einen Pointer zum letzten Matrixeintrag, wo sich die Länge von z um 1 erhöht hat. So kann man jeweils direkt (d.h. in konstanter Zeit) vom auszugebenden Zeichen z_k zum auszugebenden Zeichen z_{k-1} hüpfen.

Diese Verbesserung macht vor allem dann Sinn, wenn wir nicht nur eine, sondern viele Sequenzen aus L auslesen wollen – z.B. weil wir auch an längsten gemeinsamen Subsequenzen von beliebigen Präfixen $x^{(i)}, y^{(j)}$ interessiert sind, oder weil wir *alle* längsten gemeinsamen Teilsequenzen von x und y ausgeben wollen (hierbei kann es allerdings geschehen, dass auch mit dieser Verbesserung alleine das Ausgeben der Lösungen mehr als $\mathcal{O}(nm)$ Zeit benötigt). Im Pseudocode-Algorithmus LÄNGSTE GEMEINSAME TEILSEQUENZ ist dies so implementiert, dass $P[i,j]$ jeweils das letzte Zeichen der aktuell gefundenen längsten gemeinsamen Teilsequenz und die Koordinaten des Matrixeintrags direkt bevor dieses Zeichen hinzugefügt wurde enthält. Man beginnt fürs Auslesen also bei $P[n,m]$, gibt für $P[n,m] = (\texttt{a},(i,j))$ das Zeichen `a` aus und findet in $P[i,j]$ das nächste Zeichen sowie einen Pointer zum übernächsten Zeichen. Wie vorher wird so eine längste gemeinsame Teilsequenz rückwärts ausgegeben.

Lösungsvorschlag zu Aufgabe 4

We solve the problem as in the hint.

1. By the *p-q*-formula that you learned at school, the solutions of $x^2 + px + q = 0$ are $-\frac{p}{2} \pm \frac{1}{2}\sqrt{p^2 - 4q}$. Here we have $p = q = -1$.
2. We just multiply both sides of $x^2 = x + 1$ with x^{n-2}.
3. Let $f, g \in V$ and $\alpha \in \mathbb{R}$. Then $f + g \in V$ because $(f+g)(n) = f(n) + g(n) \overset{f,g\in V}{=} f(n-1) + f(n-2) + g(n-1) + g(n-2) = (f+g)(n-1) + (f+g)(n-2)$.

   Likewise, $\alpha f \in V$ because $\alpha \cdot f(n) \overset{f\in V}{=} \alpha \cdot (f(n-1) + f(n-2)) = \alpha \cdot f(n-1) + \alpha \cdot f(n-2)$.

   Therefore, V is a vector space. It has dimension 2 because (being slightly sloppy) a function in V is determined by the first 2 values.[1] G and H are in V by items 1 and 2, and they are independent because they are not multiples from each other. This can be argued in many ways, for example: if they were multiples of each other, then $G = \Theta(H)$ would have to hold. However, this does not hold since $G(n) \to \infty$ and $H(n) \to 0$ (since $|\psi| < 1$) for $n \to \infty$.
4. This follows since any two independent elements of a vector space of dimension 2 form a basis.
5. By definition of F, G, H, we have $F(0) = 0$, $F(1) = 1$, $G(0) = 1$, $G(1) = \phi$, $H(0) = 1$, $H(1) = \psi$. Hence, a and b must satisfy

   $$a + b = 0,$$
   $$a \cdot \phi + b \cdot \psi = 1.$$

   This is a linear system of equation, and you have learned in linear algebra how to solve them systematically. Here, we can shortcut by rewriting the first equation as $b = -a$, and plugging this into the second equation. This gives $a \cdot (\phi - \psi) = 1$, or $a = 1/(\phi - \psi) = 1/\sqrt{5}$, and thus $b = -a = -1/\sqrt{5}$.
6. This is just a summary of what we have shown: we have $F = a \cdot G + b \cdot H$, which is in other words that for all $n \geq 0$ we have

   $$F(n) = a \cdot G(n) + b \cdot H(n) = \frac{1}{\sqrt{5}} \left(\phi^n - \psi^n\right).$$

Finally, since $H(n) \to 0$ and $G(n) \to \infty$ for $n \to \infty$, we have $F = \Theta(G) = \Theta(\phi^n)$.

[1] A rigorous mathematical argument would be to consider the mapping $P : V \to \mathbb{R}^2; f \mapsto (f(0), f(1))$. It is straightforward to check that this is a homomorphism. Moreover, it is injective (any two functions in V which coincide on 0 and 1 agree everywhere, since they satisfy the same recurrence relation) and surjective (for every $x, y \in \mathbb{R}$ we can construct a function in V with $f(0) = x$ and $f(1) = y$). Hence, P is an isomorphism, and thus V has the same dimension as $\mathbb{R}^2$.