Berechnung der Verschiebefunktion

$$h_i := \max \{ k \mid 1 \leq k < i,\ p_1 \dots p_{k-1} = p_{i-k+1} \dots p_{i-1} \} \cup \{0\}$$

für $i = 1, \dots, n$, für das Muster $p_1 p_2 \dots p_n$, $n \geq 1$. Durch diese Gleichung ist auch h_{n+1} definiert, obwohl es für das Teilwortproblem nicht benötigt wird. Mit der Notation $P_j := p_1 p_2 \dots p_j$ für $0 \leq j \leq n$ lässt sich die Definition auch so ausdrücken:

$$h_i = \max \{ k \mid 1 \leq k < i,\ P_{k-1} \text{ ist Suffix von } P_{i-1} \} \cup \{0\}, \qquad (*)$$

wobei gilt:

$$P_j \text{ ist Suffix von } P_l \iff j \leq l \wedge p_1 \dots p_j = p_{l-j+1} \dots p_l,$$

für alle $0 \leq j, l \leq n$.

```
    i := 1; j := 0; h[1] := 0;
    while i < n do
        // Berechne h_{i+1}:
        while j > 0 and p_i ≠ p_j do
1           j := h[j];
        end while;
2       h[i + 1] := j + 1; i++; j++;
    end while
```

Die Korrektheit ergibt sich aus folgenden Schleifeninvarianten:

(I1) $0 \leq j < i \leq n$. (Zusätzlich ist $i < n$ am Beginn der Schleife.)

(I2) $h[1], \dots, h[i]$ sind bereits korrekt bestimmt, das heißt, $h[l] = h_l$ für $l = 1, \dots, i$.

(I3) $h_{i+1} \leq j + 1$.

(I4) Falls $j > 0$ ist, dann ist P_{j-1} Suffix von P_{i-1}.

Die Gültigkeit der Invarianten zu Beginn prüft man leicht nach. (I2) und (I3) ergeben sich dabei aus $h_1 = 0$ und $h_2 = 1$, was direkt aus der Definition folgt.

Die Invariante (I1) ergibt sich direkt durch Betrachtung aller Fälle im Algorithmus, zusammen mit der Eigenschaft $0 \leq h_l < l$ für $1 \leq l \leq n$, die unmittelbar aus der Definition $(*)$ folgt. Außerdem erhält man damit, dass das Programm terminiert: Die äußere *while*-Schleife ist eigentlich eine *for*-Schleife, in der i von 1 bis $n-1$ läuft. In der inneren Schleife nimmt j streng monoton ab.

Die Invariante (I2) ist für die Zeile zu beweisen, wo $h[i+1]$ gesetzt wird und i um 1 erhöht wird (Zeile 2). Wenn Zeile 2 erreicht wird, gilt entweder $j = 0$ oder $j > 0 \wedge p_i = p_j$. Nach (I3) ist in jedem Fall $h_{i+1} \leq j + 1$.

Falls $j = 0$ ist, bedeutet das $h_{i+1} \leq 1$, und $h_{i+1} \geq 1$ ergibt sich für $i \geq 1$ direkt aus der Definition $(*)$, und somit ist $h[i+1] = j + 1 = 1$ der korrekte Wert.

Im Fall $j > 0 \wedge p_i = p_j$ wissen wir aus (I4), dass P_{j-1} Suffix von P_{i-1} ist. Wegen $p_j = p_i$ können wir daraus schließen, dass P_j Suffix von P_i ist. Somit ist $j + 1$ ein möglicher Wert für das Maximum k in Definition $(*)$ für h_{i+1}, und daher ist $h_{i+1} \geq j + 1$. Zusammen mit (I3) ergibt sich $h_{i+1} = j + 1$, also wird $h[i + 1]$ korrekt bestimmt.

Die Eigenschaften (I1) und (I2) werden im Folgenden ohne besondere Erwähnung verwendet, und es wird nicht zwischen $h[l]$ und h_l unterschieden.

Die Invariante (I3) muss für jede Zeile der Programmes bewiesen werden, weil entweder i verändert wird (Zeile 2) oder j verkleinert wird (Zeile 1). Im ersten Fall ist $j = h_i$ (am Ende der Schleife) und die Gültigkeit von (I3) ergibt sich aus der Beziehung $h_{i+1} \leq h_i+1$, die ich in der Vorlesung hergeleitet habe, und die ich hier noch einmal beweise: Es sei $s := h_{i+1} - 1$; Nach Definition ist dann P_s Suffix von P_i. Falls $s = 0$ ist, folgt $1 \leq h_i + 1$ trivialerweise. Andernfalls können wir den letzten Buchstaben $p_s = p_i$ weglassen, und daher ist P_{s-1} Suffix von P_{i-1}. Somit ist s ein möglicher Wert für das Maximum k in Definition $(*)$ für h_i, und daher ist $h_i \geq s$.

Der Fall von Zeile 1 ist etwas aufwendiger. Wir müssen zeigen, dass die Werte $k = h_j + 2, \ldots, j + 1$ (für den alten Wert von j) nicht als Werte für h_{i+1} in Frage kommen. Dies folgt aus dem folgenden Lemma:

Lemma 1. *Aus den vier Invarianten und $j > 0$, $p_i \neq p_j$ folgt $h_{i+1} \leq h_j + 1$.*

Beweis. Es sei $s := h_{i+1} - 1$. Aus der Definition von h_{i+1} folgt die Beziehung

$$P_s \text{ ist Suffix von } P_i. \qquad (**)$$

Wir wollen zeigen, dass $s \leq h_j$ ist. Nach (I3) ist $s \leq j$. Der Fall $s = j$ fällt weg, weil die letzen Buchstaben in $(**)$ verschieden sind: $p_j \neq p_i$. Also ist $s < j$. Für $s = 0$ ist nichts zu zeigen. Andernfalls lassen wir den letzten Buchstaben in $(**)$ weg und erhalten: P_{s-1} ist Suffix von P_{i-1}. Auf Grund von (I4) ist P_{j-1} ebenfalls ein Suffix fon P_{i-1}. Da $s - 1 < j - 1$ ist, muss dann das kürzere Wort P_{s-1} ein Suffix fon P_{j-1} sein. Also ist s ein möglicher Wert für das Maximum k in Definition $(*)$ für h_j, und $h_j \geq s$, was zu zeigen war. □

Die Invariante (I4) ist leicht nachzuprüfen: Wenn am Anfang von Zeile 2 $j = 0$ ist, gilt danach $j = 1$ und (I4) ist trivialerweise erfüllt. Im anderen Fall gilt $j > 0 \wedge p_i = p_j$, und aus

$$P_{j-1} \text{ ist Suffix von } P_{i-1}$$

und $p_i = p_j$ ergibt sich unmittelbar:

$$P_j \text{ ist Suffix von } P_i.$$

Das heißt, dass (I4) auch nach dem Inkrementieren von i und j erfüllt bleibt.

Im Fall von Zeile 1 gilt nach Definition von $k := h_j$: Falls $k > 0$ ist, dann ist P_{k-1} Suffix von P_{j-1}. Auf Grund von (I4) ist P_{j-1} wiederum Suffix von P_{i-1}. Die Suffixrelation ist transitiv, und deshalb ist P_{k-1} Suffix von P_{i-1}. Somit gilt (I4) für den neuen Wert k von j.

Die *Laufzeit* kann man durch die Potentialfunktion $\Phi(i, j) := 2i - j$ abschätzen. Die Größe $\Phi(i, j)$ nimmt in jeder Zeile des Programms um mindestens eins zu, und sie beginnt bei 2 und kann höchstens $2n$ werden; daher ist die Laufzeit $O(n)$.