Beispiel (Forts.)

In der Tabelle auf der vorigen Folie fehlen 9 Einträge, die wir nun berechnen:

$g(3,\{2\}) = m_{3,2} + m_{2,1} = 13 + 5 = 18$
$g(4,\{2\}) = m_{4,2} + m_{2,1} = 8 + 5 = 13$
$g(2,\{3\}) = m_{2,3} + m_{3,1} = 9 + 6 = 15$
$g(4,\{3\}) = m_{4,3} + m_{3,1} = 9 + 6 = 15$
$g(2,\{4\}) = m_{2,4} + m_{4,1} = 10 + 8 = 18$
$g(3,\{4\}) = m_{3,4} + m_{4,1} = 12 + 8 = 20$
$g(4,\{2,3\}) = \min\{m_{4,2} + g(2,\{3\}), m_{4,3} + g(3,\{2\})\} = \min\{23,27\} = 23(2)$
$g(3,\{2,4\}) = \min\{m_{3,2} + g(2,\{4\}), m_{3,4} + g(4,\{2\})\} = \min\{31,25\} = 25(4)$
$g(2,\{3,4\}) = \min\{m_{2,3} + g(3,\{4\}), m_{2,4} + g(4,\{3\})\} = \min\{29,25\} = 25(4)$

	$\emptyset$	$\{2\}$	$\{3\}$	$\{4\}$	$\{2,3\}$	$\{2,4\}$	$\{3,4\}$	$\{2,3,4\}$
$i=1$	–	–	–	–	–	–	–	35(2)
$i=2$	5	–	15	18	–	–	25(4)	–
$i=3$	6	18	–	20	–	25(4)	–	–
$i=4$	8	13	15	–	23(2)	–	–	–

Beispiel (Abschluss)

Bitte überprüfen Sie selbst, dass der (hier nicht vorgerechnete) Eintrag rechts oben (35(2)) tatsächlich korrekt ist.

Wie interpretieren wir die Ergebnisse aus der Tabelle?

Der Eintrag 35(2) ganz rechts oben besagt, dass der kürzeste Rundweg die Länge 35 hat und von Stadt 1 zunächst zu Stadt 2 führt. Nun haben wir die Aufgabe von Stadt 2 durch die Städte 3 und 4 zurück zu Stadt 1 zu gelangen, d.h. wir müssen im Eintrag $g(2, \{3, 4\})$ nachsehen, wo wir den Eintrag 25(4) finden. Der Weg führt uns also von Stadt 2 zu Stadt 4, von wo wir dann nur noch Stadt 3 zu besuchen haben, bevor der Ausgangspunkt Stadt 1 wieder angesteuert wird.

Die Lösung ist also $1 \to 2 \to 4 \to 3 \to 1$, mit der Länge

$$m_{1,2} + m_{2,4} + m_{4,3} + m_{3,1} = 10 + 10 + 9 + 6 = 35.$$

String Matching

Das String-Matching-Problem ist das folgende:

Gegeben ist ein *Text* $T \in \Sigma^n$ und ein Muster (*pattern*) $P \in \Sigma^m$, wobei $m \leq n$.

Gesucht sind alle Verschiebungswerte (*shifts*) s, für die gilt:

$$T[s+1, \ldots, s+m] = P[1, \ldots, m]$$

In Worten: Wir suchen alle Vorkommen des Musters P im Text T.

Eine naheliegende Variante wäre, das *kleinste* s, oder auch einfach nur *irgendein* s mit dieser Eigenschaft zu suchen.

Wir wollen hier beim allgemeinsten Problem bleiben, nämlich alle Werte für s zu finden.

Erster Ansatz für String Matching

Die naive Methode, das String-Matching Problem zu lösen, wäre die folgende:

Prüfe für alle $s \in \{0, \ldots, n-m\}$, ob s eine Lösung ist.

Dazu kann man den folgenden Algorithmus verwenden:

```
FOR s := 0 TO n − m DO
        found := TRUE;
        FOR i := 1 TO m DO
                IF T[s + i] ≠ P[i] THEN found := FALSE ENDIF
        ENDFOR
        IF found THEN OUTPUT(s) ENDIF
ENDFOR
```

Und welche Komplexität erhält man auf diese Weise?

Zusätzlicher Speicherplatz wird nicht benötigt.

Als Zeitkomplexität erhält man offenbar $\Theta(m \cdot n)$.

String Matching nach Rabin und Karp

Der Rabin-Karp Algorithmus zum Pattern Matching arbeitet mit einer einfach zu berechnenden sogenannten *Hashfunktion*. Diese soll jedem Pattern der Länge m einen *Hashwert* aus einem vergleichsweise kleinen Zahlenbereich zuordnen. Dann kann man das Matching Problem so lösen:

```
H := Hashwert von P;
FOR s := 0 TO n − m DO
   IF (Hashwert von T[s + 1 ... s + m]) = H THEN Test(s) ENDIF;
```

Die Prozedur `Test` vergleicht wieder $T[s+i]$ mit $P[i]$ für alle i von 1 bis m.

Zur Definition eines Hashwerts für jedes m-buchstabige Pattern über dem Alphabet Σ fassen wir dieses als m-stellige Zahl im Zahlensystem mit Basis $|\Sigma|$ auf und berechnen den Wert dieser Zahl modulo einer geeigneten natürlichen Zahl q.

Rabin-Karp: Berechnung des Hashwerts

Der entscheidende Trick beim Rabin-Karp Algorithmus ist die sehr effiziente Berechnung der Hashwerte durch sukzessive Adaption.

Wir verdeutlichen das Prinzip an einem Beispiel:

Es sei $m = 4$ und $|\Sigma| = 3$. Als Modulus wählen wir $q = 11$. Der Text enthalte das Teilwort $a_5a_4a_3a_2a_1$. Dann ist der Hashwert erst $27a_5 + 9a_4 + 3a_3 + a_2 \mod 11$. Im nächsten Schritt soll daraus $27a_4 + 9a_3 + 3a_2 + a_1 \mod 11$ werden.

Wie geht das effizient?

Wir subtrahieren $27a_5$ (bzw. $5a_5$, weil wir modulo 11 rechnen), multiplizieren dann mit 3 (also mit $|\Sigma|$), und addieren a_1. Der Aufwand für diese Anpassung ist unabhängig von Text- und Patternlänge immer derselbe - also *konstant.*

Jetzt sind wir bereit für einen Pseudo-Code des Algorithmus.