Rabin-Karp: Pseudo-Code

Zunächst initialisieren wir die benötigte Zahl u und berechnen den Hashwert p für das Pattern $P[1 \dots m]$, sowie t für die ersten m Zeichen des Textes $T[1 \dots n]$. Dann folgt der Hauptteil:

```
u := |Σ|^(m−1) mod q;
p := 0;  t := 0;                       /* Initiale             */
FOR i := 1 TO m DO                     /* Hashwertberechnung */
   p := (|Σ| · p + P[i]) mod q;
   t := (|Σ| · t + T[i]) mod q
ENDFOR;
                                       /* Hauptteil */
FOR s := 0 TO n − m DO
   IF p = t THEN test(s) ENDIF;        /* ggfs. Ausgabe s */
   IF s < n − m THEN
      t := ((t − T[s + 1] · u) · |Σ| + T[s + m + 1]) mod q
   ENDIF
ENDFOR
```

Rabin-Karp: Analyse

Die Anzahl der Hashwertberechnungen in beiden Teilen zusammen ist $m + n$, der Aufwand hierfür also in $O(m + n)$. Dazu kommt für jede *Kollision*, d.h. für jedes s, für das $p = t$ erfüllt ist, ein Vergleich mit dem Pattern P, wofür $O(m)$ Schritte nötig sind.

Bei k Kollisionen erhalten wir also eine Gesamtrechenzeit von $O(m + n) + k \cdot O(m)$.

Unter der realistischen Annahme, dass $q > m$ ist und dass die Hashwerte zufällig verteilt im Bereich $0, \ldots, q - 1$ liegen, erhalten wir damit eine lineare Gesamtrechenzeit.

Theoretisch kann für den Rabin-Karp Algorithmus ein worst case im Bereich $\Theta(m \cdot n)$ auftreten – in der Realität erweist sich dieses Verfahren aber als sehr schnell.

Rabin-Karp: Beispiel

Wir suchen das Pattern adac in folgendem Text:

cdcbadccbaadcaadddccaaadacbaab

Dazu schreiben wir den Text zweimal untereinander, aber um $|\Sigma| = 4$ Stellen versetzt, und beachten, dass der Hashwert von adac $0 \cdot 64 + 3 \cdot 16 + 0 \cdot 4 + 2 \cdot 1 \bmod 11 = 50 \bmod 11 = 6$ ist.

```
    cdcbadccbaadcaadddccaaadacbaab
cdcbadccbaadcaadddccaaadacbaab
    984132X11314X48181673163331X
```

An den beiden markierten Stellen werden die Muster ccaa bzw. adac getestet.

Die Werte der Ziffern sind der Reihe nach 64, 16, 4 und 1, und alle Rechnungen werden mod 11 durchgeführt.

Die angegebenen Hashwerte berechnen wir an der Tafel.

String-Matching mit endlichem Automat

Wir erinnern uns an eine alte Aufgabe:

Finde DEA für Menge der Wörter $uababcv$ mit $u, v \in \{a, b, c\}^*$.

Wenn nach dem zweiten b wieder ein a kommt, springt man nicht an den Anfang zurück, sondern nur um 2 Buchstaben.

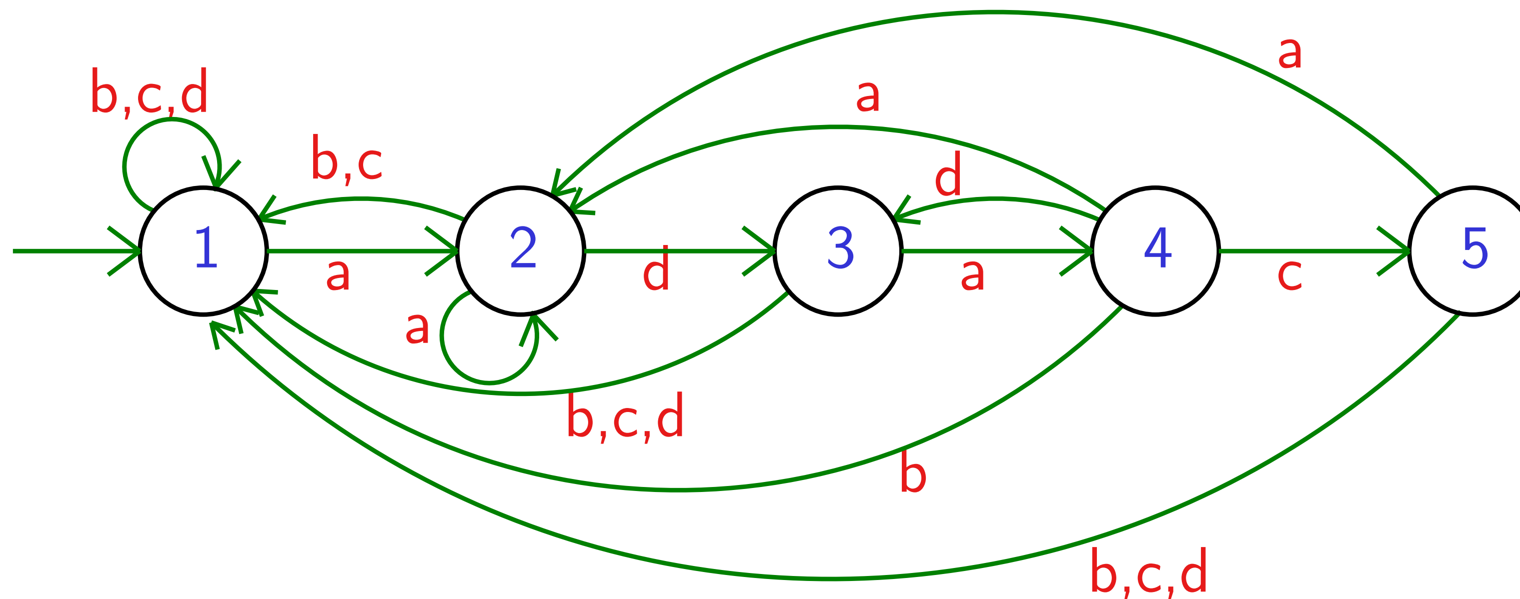
Diesen Gedanken verwenden wir auch jetzt wieder.

Der zugehörige Algorithmus ist denkbar einfach:

```
Berechne Übergangsfunktion δ;
z := 0;
FOR i := 1 TO n DO
   z := δ(z, T[i]);
   IF z = m THEN OUT(i − m) ENDIF
ENDFOR
```

String-Matching mit endlichem Automat (2)

Wie sieht der Automat für unser Beispiel adac aus?



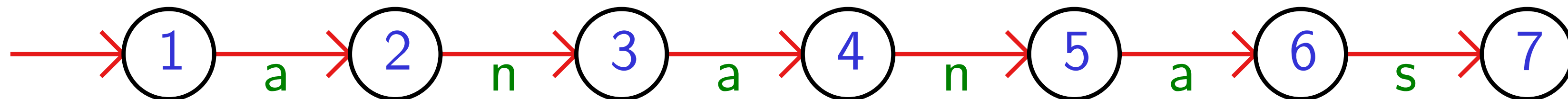
Falls es noch weitere Buchstaben, z.B. σ, im Alphabet Σ gibt (außer a,b,c,d), ist $\delta(q, \sigma) = 1$ für alle $q \in \{1, 2, 3, 4, 5\}$.

Bitte schreiben Sie sich selbst die Funktion δ vollständig in Tabellenform auf!

String-Matching mit endlichem Automat (3)

Wie sieht der Automat für das Standardbeispiel ananas aus?

Das *Skelett* des Automaten ist das folgende:



Bitte ergänzen Sie die fehlenden Übergänge und schreiben Sie sich auch hier die Funktion δ vollständig in Tabellenform auf.