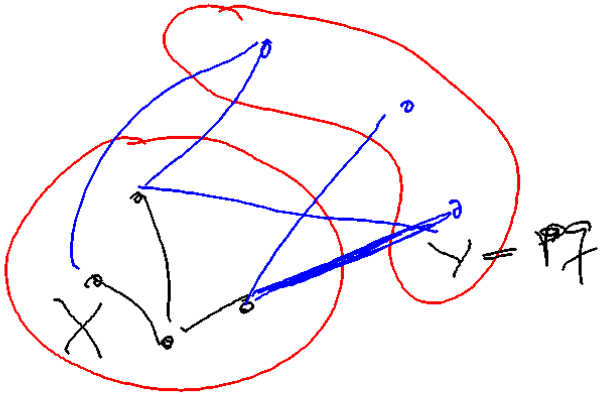
Script **generated by TTT**

Title: Seidl: GAD (29.06.2016)

Date: Wed Jun 29 13:22:32 CEST 2016

Duration: 44:20 min

Pages: 23

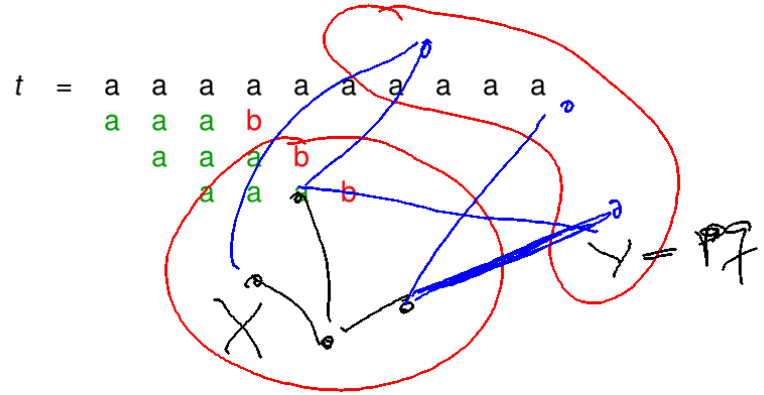


Algorithmus Jarník-Prim: findet minimalen Spannbaum

Eingabe : $G = (V, E)$, $c : E \mapsto \mathbb{R}_+$, $s \in V$
Ausgabe : Minimaler Spannbaum in Array *pred*

```
d[v] = ∞ for all v ∈ V \ s;
d[s] = 0;   pred[s] = ⊥;
pq = ⟨⟩;   pq.insert(s, 0);
while ¬pq.empty() do
    v = pq.deleteMin();
    forall {v, w} ∈ E do
        newWeight = c(v, w);
        if newWeight < d[w] then
            pred[w] = v;
            if d[w] == ∞ then pq.insert(w, newWeight) ;
            else
                if w ∈ pq then pq.decreaseKey(w, newWeight) ;
            d[w] = newWeight;
```


Naiver Algorithmus: Beispiele



Naiver Algorithmus: Implementation

Funktion NaiveSearch(char $t[\,]$, int n, char $s[\,]$, int m)

```
int i := 0, j := 0;
while (i ≤ n − m) do
    while (t[i + j] = s[j]) do
        j++;
        if (j = m) then
            return TRUE;
    i++;
    j := 0;
return FALSE;
```


Analyse des naiven Algorithmus

- zähle Vergleiche von Zeichen,
- äußere Schleife wird $(n - m + 1)$-mal durchlaufen,
- die innere Schleife wird maximal m-mal durchlaufen.
- maximale Anzahl von Vergleichen: $(n - m + 1)m$,
- Laufzeit: $O(nm)$

Bessere Idee

- frühere erfolgreiche Zeichenvergleiche ausnutzen
- Idee:

  Suchwort so weit nach rechts verschieben, dass in dem Bereich von t, in dem bereits beim vorherigen Versuch erfolgreiche Zeichenvergleiche durchgeführt wurden, nun nach dem Verschieben auch wieder die Zeichen in diesem Bereich übereinstimmen

Rand und eigentlicher Rand

Definition

Ein Wort r heißt Rand eines Wortes w, wenn r Präfix und Suffix von w ist. (Für jedes Wort w ist das leere Wort ε ein Rand von w, genau wie w selbst.)

Ein Rand r eines Wortes w heißt eigentlicher Rand, wenn $r \neq w$ und wenn es außer w selbst keinen längeren Rand gibt.

Rand und eigentlicher Rand

Beispiel

Das Wort $w =$ aabaabaa besitzt folgende Ränder:

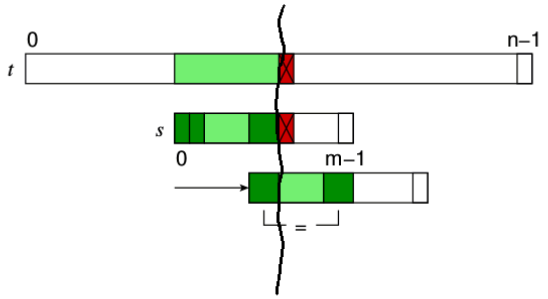
- ε
- a
- aa
- aabaa
- aabaabaa $= w$.

Der eigentliche Rand ist aabaa.

Beachte: bei der Darstellung des Rands im Wort können sich Präfix und Suffix in der Wortmitte überlappen.

Shift-Idee

- Pattern s so verschieben, dass im bereits gematchten Bereich wieder Übereinstimmung herrscht.
- Dazu müssen überlappendes Präfix und Suffix dieses Bereichs übereinstimmen.



Shifts und sichere Shifts

Definition

Eine Verschiebung der Anfangsposition i des zu suchenden Wortes (also eine Indexerhöhung $i \to i'$) heißt Shift.

Ein Shift von $i \to i'$ heißt sicher, wenn s nicht als Teilwort von t an der Position $k \in [i+1 : i'-1]$ vorkommt, d.h., $s \neq t_k \cdots t_{k+m-1}$ für alle $k \in [i+1 : i'-1]$.

- Sinn eines sicheren Shifts:
  dass man beim Verschieben des Suchworts kein eventuell vorhandenes Vorkommen von s in t überspringt

Sichere Shifts

Definition

Sei $\partial(s)$ der eigentliche Rand von s und sei

$$border[j] = \begin{cases} -1 & \text{für } j = 0 \\ \left|\partial(s_0 \cdots s_{j-1})\right| & \text{für } j \geq 1 \end{cases}$$

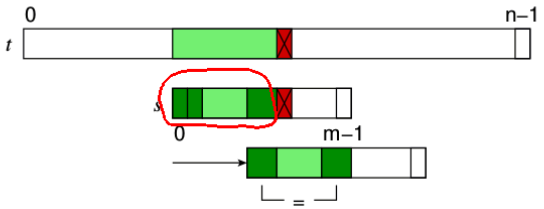
die Länge des eigentlichen Rands des Präfixes der Länge j.

Lemma

Ist das Präfix der Länge j gematcht (also gilt $s_k = t_{i+k}$ für alle $k \in [0 : j-1]$) und haben wir ein Mismatch an der nächsten Position j ($s_j \neq t_{i+j}$), dann ist der Shift $i \to i + j - border[j]$ sicher.

Shift-Idee

- Pattern s so verschieben, dass im bereits gematchten Bereich wieder Übereinstimmung herrscht.
- Dazu müssen überlappendes Präfix und Suffix dieses Bereichs übereinstimmen.



Sichere Shifts

Definition

Sei $\partial(s)$ der eigentliche Rand von s und sei

$$border[j] = \begin{cases} -1 & \text{für } j = 0 \\ |\partial(s_0 \cdots s_{j-1})| & \text{für } j \geq 1 \end{cases}$$

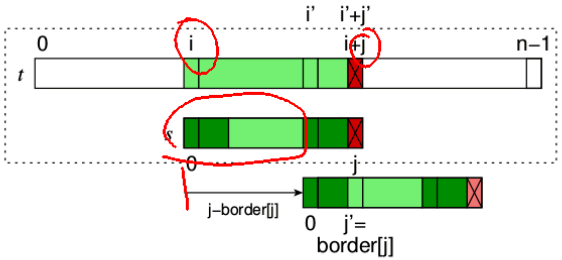
die Länge des eigentlichen Rands des Präfixes der Länge j.

Lemma

Ist das Präfix der Länge j gematcht (also gilt $s_k = t_{i+k}$ für alle $k \in [0 : j-1]$) und haben wir ein Mismatch an der nächsten Position j ($s_j \neq t_{i+j}$), dann ist der Shift $i \to i + j - border[j]$ sicher.

Sichere Shifts

Shift um $j - border[j]$



Sichere Shifts

Beweis.

- (siehe Skizze)

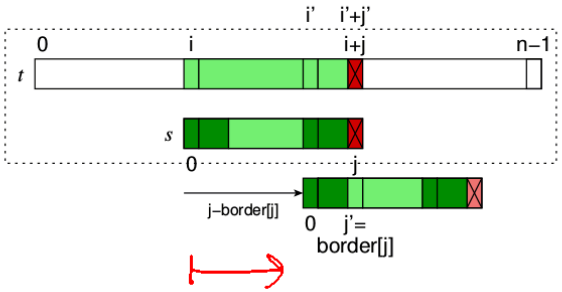
$$\begin{aligned} s_0 \cdots s_{j-1} &= t_i \cdots t_{i+j-1}, \\ s_j &\neq t_{i+j} \end{aligned}$$

- Der eigentliche Rand von $s_0 \cdots s_{j-1}$ hat die Länge $\mathrm{border}[j]$.
- Verschiebt man s um $j - \mathrm{border}[j]$ nach rechts, so liegt der linke Rand von $s_0 \cdots s_{j-1}$ nun genau da, wo vorher der rechte Rand lag, d.h. im Präfix/Suffix-Überlappungsbereich besteht Übereinstimmung zwischen Präfix, Suffix und Text.
- Da es keinen längeren Rand von $s_0 \cdots s_{j-1}$ als diesen gibt (außer $s_0 \cdots s_{j-1}$ selbst), ist dieser Shift sicher.

□

Sichere Shifts

Shift um j – border[j]



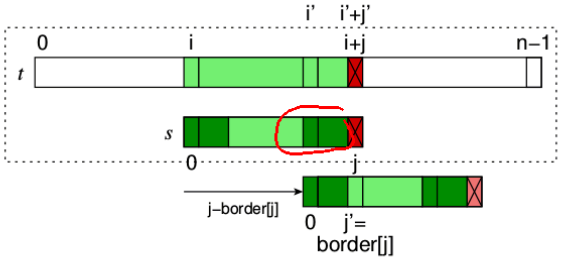
KMP-Algorithmus

Funktion KMP($t[\,]$, n, $s[\,]$, m)

```
int border[m + 1];
computeBorders(border, m, s);
int i := 0, j := 0;
while i ≤ n − m do
    while t[i + j] = s[j] do
        j++;
        if j = m then
            return TRUE;
    i := i + (j − border[j]) ;          // Es gilt j − border[j] > 0
    j := max{0, border[j]};
return FALSE;
```


Sichere Shifts

Shift um j – border[j]



Sichere Shifts

Beweis.

- (siehe Skizze)

$$\begin{aligned} s_0 \cdots s_{j-1} &= t_i \cdots t_{i+j-1}, \\ s_j &\neq t_{i+j} \end{aligned}$$

- Der eigentliche Rand von $s_0 \cdots s_{j-1}$ hat die Länge $\mathrm{border}[j]$.
- Verschiebt man s um $j - \mathrm{border}[j]$ nach rechts, so liegt der linke Rand von $s_0 \cdots s_{j-1}$ nun genau da, wo vorher der rechte Rand lag, d.h. im Präfix/Suffix-Überlappungsbereich besteht Übereinstimmung zwischen Präfix, Suffix und Text.
- Da es keinen längeren Rand von $s_0 \cdots s_{j-1}$ als diesen gibt (außer $s_0 \cdots s_{j-1}$ selbst), ist dieser Shift sicher.

□

KMP-Algorithmus

Funktion KMP($t[\,]$, n, $s[\,]$, m)

```
int border[m + 1];
computeBorders(border, m, s);
int i := 0, j := 0;
while i ≤ n − m do
    while t[i + j] = s[j] do
        j++;
        if j = m then
            return TRUE;
    i := i + (j − border[j]) ;          // Es gilt j − border[j] > 0
    j := max{0, border[j]};
return FALSE;
```


Pattern Matching | Knuth-Morris-Pratt-Algorithmus

KMP-Algorithmus

Funktion KMP($t[\,]$, n, $s[\,]$, m)

```
int border[m + 1];
computeBorders(border, m, s);
int i := 0, j := 0;
while i ≤ n − m do
    while t[i + j] = s[j] do
        j++;
        if j = m then
            return TRUE;
    i := i + (j − border[j]) ;          // Es gilt j − border[j] > 0
    j := max{0, border[j]};
return FALSE;
```

H. Seidl (TUM) GAD SS'16 575

Pattern Matching | Knuth-Morris-Pratt-Algorithmus

Laufzeit des KMP-Algorithmus: erfolglose Vergleiche

Nach erfolglosem Vergleich (Mismatch) wird $(i + j)$ nie kleiner:

- Seien dazu i und j die Werte vor einem erfolglosen Vergleich und i' und j' die Werte nach einem erfolglosen Vergleich.
- Wert vor dem Vergleich: $i + j$
- Wert nach dem Vergleich:
  $i' + j' = (i + j - border[j]) + (max\{0, border[j]\})$.
- Fallunterscheidung: $border[j]$ negativ oder nicht.
  - $border[j] < 0$, also $border[j] = -1$, dann muss $j = 0$ sein.
    Das bedeutet $i' + j' = i' + 0 = (i + 0 - (-1)) + 0 = i + 1$.
  - $border[j] \geq 0$, dann gilt $i' + j' = i + j$
- Also wird $i + j$ nach einem erfolglosen Vergleich nicht kleiner.

Pattern Matching | Knuth-Morris-Pratt-Algorithmus

Laufzeit des KMP-Algorithmus

- Nach jedem erfolglosen Vergleich wird $i \in [0 : n - m]$ erhöht.
- i wird nie verkleinert.

$\Rightarrow$ maximal $n - m + 1$ erfolglose Vergleiche

- Nach einem erfolgreichen Vergleich wird $i + j$ um 1 erhöht.
- maximal n erfolgreiche Vergleiche, da $i + j \in [0 : n - 1]$.

- insgesamt maximal $2n - m + 1$ Vergleiche