Suchen in Texten

- Naives Suchen
- Verfahren von Knuth-Morris-Pratt
- Verfahren von Boyer-Moore
- Ähnlichkeitssuchen
- Editierdistanz

Textsuche

Gegeben ist ein Zeichensatz (Alphabet) Σ.
Für einen Text $T \in \Sigma^n$ und ein Wort $w \in \Sigma^m$ mit $n >> m$ stellt sich die Frage:
Wie oft / Wo kommt Wort w in Text T vor?

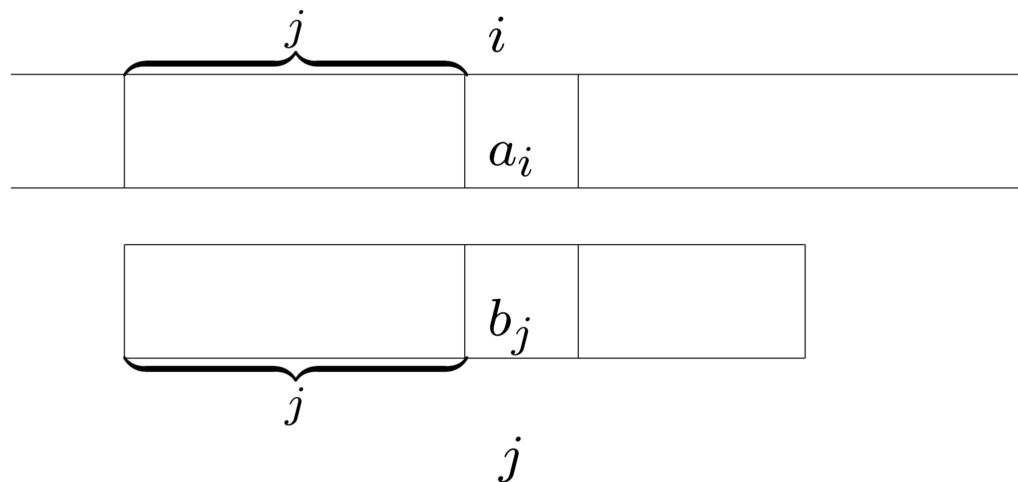
Dabei dynamischer Text, d.h. Vorbereitung durch Abspeichern in geeigneten Datenstrukturen zu aufwändig.

```
int naivSuch(char [] text, char [] wort)
        {
        int zaehler = 0;
        for (int i = 0;i <= text.length - wort.length;i++)
           { boolean passt = true;
             for (int j=0; j< wort.length; j++)
                 {
                 if (text[i+j] != wort[j]) passt = false;
                 }
            if (passt) zaehler++;
           }
        return zaehler;
        }
```

Aufwand: $\mathcal{O}(m * n)$

Verfahren von Knuth-Morris-Pratt

Idee: Information aus vorherigen Vergleichen wiederverwerten.
Keine bekannten Zeichen des Textes erneut vergleichen.
Wort ggf. um mehrere Stellen verschieben.



Betrachte $w' = b_0, \ldots, b_{j-1}$

Gesucht: größter (echter) Suffix von w', der selbst Präfix von w ist.

Beispiel:

j	0	1	2	3	4	5	6
$char[j]$	A	N	A	N	A	S	
$next[j]$	-1	0	0	1	2	3	0

```
int kmpSuche(char [] text,  char [] wort)
     {
     int zaehler = 0;
     int i = 0;  int j =0;
     int [] next = initnext(wort);
             while ( i < text.length)
             {
                 if (text[i] == wort[j] && j < wort.length -1)
                 {i++; j++;}
                 else {
                     if (text[i] == wort[j] && j== wort.length -1)
                         { zaehler ++;}
                     j = next[j];
                     if (j == -1) {i++; j++;}
                 }
             }
     return zaehler;
     }
```

Aufwandsabschätzung:

- i wird nie kleiner.
- $next[j]$ ist immer kleiner als j.
- i und j werden immer gemeinsam inkrementiert.
- j kann nur so oft kleiner werden, wie es vorher inkrementiert wurde.

$\Rightarrow j \;=\; next[j]$ wird höchstens n-mal angewandt.

$\Rightarrow$ Aufwand in $\mathcal{O}(n)$

Dabei wird $next[j]$ als bekannt vorausgesetzt.

Berechnung der next[j] Werte

```
int [] initnext( char[] wort)
{
    int[] next = new int[wort.length+1];
    next[0] = -1;
    next[1] = 0;
    int i = 1; int j = 0;
    while (i < wort.length){
        if (wort[i] != wort[j])
            { j = next[j];
              if (j == -1) {i++; j++; next[i] = j;}
        }
        else {i++; j++;next[i] = j;}
    }
return next;
}
```

Aufwand in $\mathcal{O}(m)$

Verfahren von Boyer-Moore

Wort $w = w_1 \dots w_m$ wird von links nach rechts gegen Text T verschoben
Aber: Vergleich läuft von w_m nach w_1.
Für alle Zeichen c des Alphabets Σ:

$$delta - 1(c) = \begin{cases} m, & \text{falls } c \text{ nicht in } w_1 \dots w_m \text{ vorkommt.} \\ m - j, & \text{falls } c = w_j \text{ und } c \neq w_k \text{ für } j < k \leq m \end{cases}$$

Aufwand im schlechtesten Fall $\mathcal{O}(n * m)$.
Für grosses Alphabet/kurze Muster $\mathcal{O}(n/m)$.

Verbesserung: Funktion $delta - 2(j)$ zur Rechtsverschiebung.
Konstruktion ähnlich der $next[j]$ Funktion beim KMP-Verfahren.

Verhindert Verschieben nach links $\Rightarrow$ worst-case $\mathcal{O}(n + m)$.

k-Mismatch-Problem

Gegeben: Text $T \in \Sigma^n$, Muster $w \in \Sigma^m$.

Gesucht: Abschnitt $b \in \Sigma^m$ in T, so das sich b und w an höchstens k Positionen unterscheiden.

Für $k = 0$: Textsuchproblem

Naive Lösung: Abwandlung der naiven Lösung für Textsuche.

```
int naivSuch(char [] text, char [] wort, int k)
        {
        int zaehler = 0;
        for (int i = 0;i <= text.length - wort.length;i++)
           { int passt = 0;
             for (int j=0; j< wort.length; j++)
                 {
                 if (text[i+j] != wort[j]) passt++;
                 }
            if (passt <= k) zaehler++;
           }
        return zaehler;
        }
```

Editieroperationen: Löschen, Einfügen, Ändern

Seien $\alpha, \beta \in \Sigma$, ε das leere Wort. Dann ist eine Editieroperation eine Anwendung einer Regel der Form $\alpha \rightarrow \varepsilon$, $\varepsilon \rightarrow \alpha$ oder $\alpha \rightarrow \beta$.

Kostenfunktion c weist jeder Operation $\alpha \rightarrow \beta$ Kosten $c(\alpha \rightarrow \beta)$ zu.

Einheitskostenmodell: Für $a, b \in \Sigma, a \neq b : c(a \rightarrow b) = c(a \rightarrow \varepsilon) = c(\varepsilon \rightarrow b) = 1$ und $c(a \rightarrow a) = 0$.

Für Zeichenketten $A = a_1 \ldots a_m$ und $B = b_1 \ldots b_n$ definiere als Editierdistanz $D(A, B)$ die minimalen Kosten einer Folge von Editieroperationen, die A in B überführt.

Berechnung der Editierdistanz

- Optimale Folge von Editieroperationen ändert jedes Zeichen höchstens einmal.

- Jede Zerlegung einer optimalen Spur (geordnete Folge von Operationen) führt zu optimalen Spuren der entsprechenden Teilsequenzen.

$\Rightarrow$ Konstruktion der optimalen Gesamtlösung durch rekursive Kombination von Lösungen für Teilprobleme.

Berechne für $A = a_1 \dots a_m$ und $B = b_1 \dots b_n$ die Editierdistanz $D(A, B)$.

$\Leftarrow$ Berechne für jedes Paar (i, j) mit $0 \leq i \leq m$ und $0 \leq j \leq n$ die Editierdistanz $D_{i,j} \stackrel{\text{def}}{=} D(a_1 \dots a_i, b_1 \dots b_j)$.

$D_{0,j} \stackrel{\text{def}}{=} D(\varepsilon, b_1 \dots b_j) = j$ für $1 \leq j \leq n$(Nur Einfügen).

$D_{i,0} \stackrel{\text{def}}{=} D(a_1 \dots a_i, \varepsilon) = i$ für $1 \leq i \leq m$ (Nur Löschen).

$D_{0,0} = 0$.

$D(A, B) = D_{m,n}$.

Berechne $D_{i,j} \stackrel{\text{def}}{=} D(a_1 \dots a_i, b_1 \dots b_j)$.

Fall 1) Löschen von a_i: Dann ist $D_{i,j} = D(a_1 \dots a_{i-1}, b_1 \dots b_j) + 1 = D_{i-1,j} + 1$.

Fall 2) Einfügen von b_j: Dann ist $D_{i,j} = D(a_1 \dots a_i, b_1 \dots b_{j-1}) + 1 = D_{i,j-1} + 1$.

Fall 3a) $a_i = b_j$: Dann ist $D_{i,j} = D(a_1 \dots a_{i-1}, b_1 \dots b_{j-1}) = D_{i-1,j-1}$.

Fall 3b) $a_i \neq b_j$: Dann ist $D_{i,j} = D(a_1 \dots a_{i-1}, b_1 \dots b_{j-1}) + 1 = D_{i-1,j-1} + 1$.

$$D_{i,j} = min\{D_{i-1,j-1} + \begin{cases} 1, & \text{für } a_i \neq b_j \\ 0, & \text{für } a_i 0 b_j \end{cases}, D_{i-1,j} + 1, D_{i,j-1} + 1$$

Editierdistanz $\surd$

Wie sieht die optimale Folge von Editieroperationen aus?

Konstruiere Spurgraph:

Knoten: Positionen (i, j) mit Wert $D_{i,j}$.

bewertete Kanten: Entsprechend der Konstruktion der Werte, also z.B. $c(D_{i-1,j}, D_{i,j}) = 1$ im Fall 1).

Finde kürzesten Weg von $D_{0,0}$ nach $D_{m,n}$.

Im optimalen Pfad ist der Wert eines Knoten $D_{i,j}$ gleich der Summe der Gewichte von $D_{0,0}$ nach $D_{i,j}$.

Weitere Fragestellungen

Gegeben Text T und Wort w sowie $k \geq 0$. Suche alle Vorkommen von Zeichenreihen B in T, so das $D(w, B) \leq k$ ist.

.

.

.