Textsuche mit Don't-Care Zeichen

Don't Cares im Muster

Geg.:

Alphabet Σ, $|\Sigma| = r$
Text $x = x_1x_2\ldots x_n$, $x_i \in \Sigma$, $|x| = n$
Muster $y = \Phi^{k_0} y_1 \Phi^{k_1} y_2 \Phi^{k_2} \ldots\ y_{p-1} \Phi^{k_{p-1}} y_p \Phi^{k_p}; k_0, k_p \in \mathbb{N}_0; k_1 \ldots k_{p-1} \in \mathbb{N}$; $y_i \in \Sigma^+$;
$|y| = m$, $\Phi^t \in \{\Phi\}^+$, $\forall e \in \Sigma : \Phi \Leftrightarrow e$

Ges.:

Menge aller Treffer des Musters y in x.
Genauer: $T = \{\ i \mid x_i x_{i+1} \ldots x_{i+m-1} = y\ ,\ 1 \leq i \leq\ n-m+1\}$

Idee:

Durchlaufe den Text von Position 1 bis n.
Lege ein Integer Array der Größe m an, und führe eine Strichliste (Hochzählen) darüber, wie oft ein Segment y_i des Musters gefunden wurde, das zur selben möglichen Anfangsposition eines Treffers im Text gehört. Diese Anfangspositionen werden zyklisch Modulo m betrachtet. Hat ein Feldelement dieses Arrays den Wert p erreicht - mit 0 initialisiert - , so liegt ein Treffer vor. Nach jedem Hochsetzen des Positionszeigers im Text werden zuallererst veraltete Zählerstände im Integerarray durch Reinitialisieren mit 0 bereinigt, damit keine Fehltreffer entstehen. Dann erst erfolgt ein eventueller Stricheintrag.

Algorithmus von Pinter:

m[1..m] = 0;
dy[1..p] = <Offset vom letzten Zeichen des Mustersegments y_q zum Musteranfang y>

```
For i = 1 to n
Begin
        imod =  ( (i-1) mod m ) +1;
        m[imod] = 0;
        For q = 1 to p
        Begin
                If ( match( x[ i ], y[q] ) )
                Begin
                        j = ( ( i – dy[q] – 1) mod m ) +1;
                        m[ j ]++;
                        If( m[ j ] == p)
                        Begin
                                startpos = ⌊(i – j) / m⌋•m + j ;
                                T = T ∪ {startpos};
                        End
                End
        End
End
```

Don't Cares im Muster und im Text

<u>Geg.:</u>

Alphabet Σ, $|\Sigma| = r$, $\Phi \in \Sigma$, $\forall e \in \Sigma : \Phi = e$
Text $x = x_0 x_1 \ldots x_n$, $x_i \in \Sigma$, $|x| = n+1$
Muster $y = y_0 y_1 \ldots y_m$, $y_i \in \Sigma$, $|y| = m+1$

<u>Ges.:</u>

Menge aller Treffer des Musters y in x.
Genauer: $T = \{ i \mid x_i x_{i+1} \ldots x_{i+m-1} = y \,,\ 1 \le i \le n-m+1 \}$

<u>Ansatz:</u>

Das Linearprodukt zweier Vektoren lässt sich schreiben

$$\begin{pmatrix} X_0 \\ X_1 \\ \vdots \\ X_{m+n} \end{pmatrix} \begin{matrix} \otimes \\ \oplus \end{matrix} \begin{pmatrix} Y_0 \\ Y_1 \\ \vdots \\ Y_{m+n} \end{pmatrix} = X \overset{\otimes}{\underset{\oplus}{}} Y = Z = \begin{pmatrix} Z_0 \\ Z_1 \\ \vdots \\ Z_{m+n} \end{pmatrix} = Z_0 Z_1 \ldots Z_{m+n}$$

$$Z_k = \oplus_{i+j=k} X_i \otimes Y_j \quad , k = 0, \ldots, m+n$$

Wenn wir definieren: $\oplus := \wedge$ und $\otimes := \equiv$, dann ergibt

$$Z_k = \wedge_{i+j=k} X_i \equiv Y_j$$

Wenn wir nun X und Y folgendermaßen belegen

$$X = \begin{pmatrix} X_0 \\ X_1 \\ \vdots \\ X_n \\ \Phi^m \end{pmatrix} \quad , \quad Y = \begin{pmatrix} Y_0 \\ Y_1 \\ \vdots \\ Y_m \\ \Phi^n \end{pmatrix} ,$$

dann können wir einen Text der Länge n+1 und ein Muster der Länge m+1 als Vektoren gleicher Länge n+m+1 schreiben.
Außerdem kürzt sich für $m \le k \le n$, welches das Intervall der möglichen Endpositionen des Musters im Text darstellt, folgendes:

$$[X_{k-m} \ldots X_k] \equiv [Y_m \ldots Y_0] = \left(X \overset{\equiv}{\underset{\wedge}{}} Y \right)_k \quad ; m \le k \le n$$

Hieraus ergibt sich, dass man bei $\left(X \overset{\equiv}{\underset{\wedge}{}} Y \right)_k = true$ einen Treffer bei Endposition k im Text gefunden hat, jedoch für das Muster Y^R. Wenn man also Treffer für ein Muster W haben möchte, so muß man $X \overset{\equiv}{\underset{\wedge}{}} W^R$ rechnen.

Betrachten wir jetzt den Fall, dass unser Text x nur aus 2 verschiedenen Zeichen aufgebaut ist, sagen wir a und das Don't Care Symbol Φ. Analog sei das Muster y nur aus dem Zeichen b≠a und Φ aufgebaut. Diesen entsprechend existieren die Vektoren X und Y.

Sei $H_c(X_i) = \begin{cases} 1, wenn\ \mathrm{X_i} = c \\ 0, sonst \end{cases}$.

Wenn wir nun wieder mehr als 2 Symbole für Text und Muster erlauben, müssen wir für alle 2-Variationen (geordnete 2er Mengen) von Zeichen aus Σ die obige Rechnung machen und die Einzelergebnisse geeignet verknüpfen und damit interpretieren.

$$\neg \vee_{a\neq b;a,b\in\Sigma} H_a(X) \underset{\vee}{\overset{\wedge}{}} H_b(Y) = \neg \vee_{a\neq b;a,b\in\Sigma} \left(\vee H_a(X) \wedge H_b(Y)\right)$$

$$= \wedge_{a\neq b;a,b\in\Sigma} \neg \left(\vee H_a(X) \wedge H_b(Y)\right)$$

$$= \begin{pmatrix} \wedge_{a\neq b;a,b\in\Sigma} \neg \left(\vee_{i+j=0} H_a(X_i) \wedge H_b(Y_j)\right) \\ \wedge_{a\neq b;a,b\in\Sigma} \neg \left(\vee_{i+j=1} H_a(X_i) \wedge H_b(Y_j)\right) \\ \vdots \\ \wedge_{a\neq b;a,b\in\Sigma} \neg \left(\vee_{i+j=n+m} H_a(X_i) \wedge H_b(Y_j)\right) \end{pmatrix}$$

Die Schreibweise mit dem großen ∧ am Anfang, gibt zum Ausdruck, dass für alle r*(r-1) Buchstabenvariationen, das Ergebnis ein Treffer darstellen muss bzgl. des gewählten Buchstabentupels, damit das ∧ true liefert.

<u>Idee (Fischer und Paterson):</u>

Mustersuche umformulieren in das Berechnen des Linearproduktes von Vektoren.

Das Ergebnis des Booleschen Produktes kann aus dem Polynomprodukt gewonnen werden, welches sich ergibt, wenn man für $\oplus$ $+$ einsetzt und für $\otimes$ $\bullet$.
Dazu bildet man aus den aus 0 und 1 bestehenden Vektoren der Gleichung
$\neg \vee_{a\neq b;a,b\in\Sigma} H_a(X) \underset{\vee}{\overset{\wedge}{}} H_b(Y) = \neg \vee_{a\neq b;a,b\in\Sigma} H_a(X) \underset{\oplus}{\overset{\otimes}{}} H_b(Y)$, welche pro Zeichenpaar (a,b) existieren, entsprechende Vektoren $X(2^r) = \sum_{i=0}^{n} X_i \bullet 2^{r\cdot i}$ bzw. Y(2 ^{r}), mit $2^r > m+1; r \in \mathbb{N}$. Mit diesen Vektoren lässt sich eine Ganzzahlmultiplikation durchführen und das Ergebnis Rücktransformieren, wenn man Summen gößer 0 auf true abbildet und die 0 als false interpretiert.

<u>Laufzeit:</u>

Der Algorithmus von Shönhage-Strassen benötigt zur Multiplikation von zwei Zahlen der Größe N und M eine Laufzeit von O(N*logM*loglogM) für N>M,durch Benutzung einer Mehr-Band Turingmaschine.
Für unser Problem des Stringvergleiches ergubt sich aus N=nr und M=m*r eine Laufzeit von

O(n*log ^{2}m*log log m)

LCS (Längster gemeinsamer Teilstring // Longest Common Subsequences)

Definitionen:

- [i,j] ist ein Match wenn $y_i = x_j$
- Rang eines Matches ist die Länge einer LCS zwischen des Strings $y_1 \ldots y_i$ und $x_1 \ldots x_j$
- Ein Match [i,j] ist k-dominant wenn es
  - Rang k hat und
  - Für alle [i‘,j‘] mit Rang k gilt:
    - $i' \leq i$ und $j' > j$ oder
    - $i' > i$ und $j' \leq j$
- Ordnungsrelation R: eine Match [i,j] ist der Vorgänger von [i‘,j‘] wenn gilt: i<i‘ und j<j‘. Dies erzeugt eine partiell geordnete Menge (Poset).
- Kette: Eine Menge von Matches in der jeweils zwei in Relation zueinander stehen.
- Antikette: Eine Menge von Matches, in der niemand zu niemandem in Relation steht.
- Eine Zerlegung des Posets in Antiketten heißt minimal, wenn die Anzahl der Antiketten minimal ist.

Folgerungen:

- „Finde LCS“ wird übersetzt in „Suche nach einer Kette maximaler Länge im Poset“.
- Theorem:
  Gegeben ein Poset P. Eine Kette maximaler Länge enthält aus einer minimalen Antikettenzerlegung aus jeder Antikette ein Element.

- Gegeben: Ein String y[1:m] und ein String x[1:n].
- Gesucht: eine LCS

Dynamische Berechnung des LCS

FOR $1 \leq j \leq |y|$

  FOR $1 \leq i \leq |x|$

    IF (x_i=y_j) THEN L[i,j] = L[i-1,j-1]+1
    ELSE L[i,j] = MAX{L[i,j-1],L[i-1,j]}

L[i,j] ist dann die Länge eines LCS zwischen $x_1 \ldots x_i$ und $y_1 \ldots y_j$.

Laufzeit: $\Theta(m * n)$. Speicherplatz: Wenn nur die Länge der LCS interessiert linear. Wenn auch die LCS interessiert müssen die Entscheidungen gespeichert werden.

Hirschbergalgorithmus:

- findet schrittweise die Antiketten
- Annahme: [0,0] ist ein 0-dominantes Match
- Des weiteren seien alle (k-1)-dominanten Matches gefunden.
- Und der Teil der Matrix der über bzw. links neben der (k-1) Antikette liegt ist bereits bearbeitet.
- Finden der k`ten Antikette.
  - Scannen des unbearbeiteten Bereiches der Matrix (von oben nach unten, von rechts nach links) bis ein Strom von Matches gefunden wird.
  - Das Linkeste dieser Matches ist das k-dominante Match.
  - Der Scan geht in der nächsten Reihe nach Links weiter.

Wenn effizient implementiert: Laufzeit: $O(nl + n\log s)$. Speicherplatz: $O(d + n)$.

Wobei :

d = # dominanten Matches || l = Länge der LCS || s = Mächtigkeit des Alphabetes.
Linearer Speicherplatz reicht, wenn nur die Länge der LCS interessiert.

Hunt-Szymanskialgorithmus:

- Benötigt:
  - integer Array THRESH[0:m]
  - list Array MATCHLIST[1:m]
  - Pointer Array LINK[1:m]

Phase 1:

- Füllen der MATCHLIST
  - MATCHLIST[i] = $\{j_1, j_2, \ldots, j_v\}$. wobei:
    - $y_i = x_w$
    - $j_1 > j_2 > \ldots > j_v$
- Füllen des THRESH
  - THRESH[i] = n+1 und THRESH[0] = 0
- LINK[0] = null

Phase 2:

- FOR i=1:m
  - FOR j $\in$ MATCHLIST[i]
    - Finde ein k, so dass gilt THRESH[k-1] $< j \leq$ THRESH[k]
    - IF j<THRESH[k] DANN
      - THRESH[k] = j
      - LINK[k] = neuer Knoten(i,j,LINK[k-1])

Phase 3:

- $\hat{k} =$ das größte k, für dass gilt: $THRESH[k] \neq n+1$
- Pointer = LINK[$\hat{k}$]
- Solange Pointer $\neq$ null
  - Out [i,j]
  - Verfolge LINK

Laufzeit: $O((r+m)\log n + n \log s)$ Speicherplatz: $O(d+n)$ Wobei:
r = #Matches || m bzw. n = |x| bzw. |y| || s = Mächtigkeit des Alphabetes || d = # dominanter Matches.

Stringsuche mit Fehlern:

- Gegeben: ein Text x und ein Muster y
- Ziel: alle Stellen im Text finden die mit dem Muster übereinstimmen. Bis zu k Abweichungen sind erlaubt.

Realisierung mittels dynamischem Programmieren

Initialisierungswerte:

$C(0,0)=0 \quad C(i,0)=i \quad C(0,j)=0$

$C(i,j)=\min\{C(i-1,j-1)+S(y_i,x_j), C(i-1,j)+1, C(i,j-1)+1\}$

Effekt: die Kosten, ein Präfix beliebiger Länge vom Text abzuschneiden, sind 0.

Laufzeit: $\Theta(m*n)$ Speicherplatz: linear Als Ausgabe ist die gesamte letzte Zeile interessant.