Technische Universität München
Institut für Informatik
Lehrstuhl für Effiziente Algorithmen
Volker Heun

SS 2000
Lösungsblatt 4
22. Juni 2000

Algorithmen der Bioinformatik

Abgabetermin: 6. Juni

Aufgabe 1

Sei Σ ein Alphabet mit $- \notin \Sigma$, $\Sigma^- := \Sigma \cup \{-\}$ und $\mathbf{R}_0^+ = \{x \in \mathbf{R} : x \geq 0\}$.

Die Funktion $d : (\Sigma^-)^2 \to \mathbf{R}_0^+$ ist ein *Abstandsmaß*, wenn folgende Bedingungen gelten:

- d ist symmetrisch, d.h. $d(x, y) = d(y, x)$.
- d erfüllt die Dreiecksungleichung, d.h. $d(x, y) \leq d(x, z) + d(z, y)$.
- Für $A := \max\{d(a, a) : a \in \Sigma\}$ und $B := \min\{d(-, -), d(a, b) : a \neq b \in \Sigma^-\}$ gilt $A \leq B$.

Die Funktion $s : (\Sigma^-)^2 \to \mathbf{R}$ ist ein *Ähnlichkeitsmaß*, wenn folgende Bedingungen gelten:

- s ist symmetrisch, d.h. $s(a, b) = s(b, a)$
- $s(a, a) \geq 0$ für $a \in \Sigma$.
- $s(a, b) \leq 0$ für $a \neq b \in \Sigma$.
- $s(-, -) \leq 0$.

1) Zeigen Sie, daß man zu jedem Abstandsmaß d ein Ähnlichkeitsmaß s definieren kann, so daß $s(a, b) = C - d(a, b)$ für ein geeignetes $C \in \mathbf{R}_0^+$.

Lösungsvorschlag:
Da für ein Ähnlichkeitsmaß die Werte $s(a, a)$ bzw. $s(a, b)$ für $a \neq b$ gerade durch die 0 getrennt sein sollen, bietet sich an, $C := A$ zu setzen. Die Symmetrie bleibt natürlich vom Abstandsmaß her erhalten, und es gilt gerade $s(a, a) = \max_{a' \in \Sigma} d(a', a') - d(a, a) \geq 0$ und $s(a, b) = \max_{a' \in \Sigma} d(a', a') - d(a, b) \leq A - B \leq 0$. Letzteres gilt ebenso für $a = b =$ '$-$'.

2) Zeigen Sie, daß man zu jedem Ähnlichkeitsmaß s ein Abstandsmaß d definieren kann, so daß $d(a, b) = D - s(a, b)$ für ein geeignetes $D \in \mathbf{R}_0^+$.

Lösungsvorschlag:
Hier ist die Wahl von D etwas weniger intuitiv: Man setzt $D := \max_{a,b\in\Sigma} |s(a,b)|$. Dann gilt sowohl die Symmetrie als auch die geforderte Ungleichung $A \leq B$. Denn es ist

$$A = \max_{a\in\Sigma} d(a,a) = \max_{a\in\Sigma}(\max_{a',b'\in\Sigma} |s(a',b')| - \underbrace{s(a,a)}_{\geq 0}) \leq \max_{a',b'\in\Sigma} |s(a',b')|,$$

und

$$B = \min_{a\neq b} d(a,b) = \min_{a\neq b}(\max_{a',b'\in\Sigma} |s(a',b')| - \underbrace{s(a,b)}_{\leq 0}) \geq \max_{a',b'\in\Sigma} |s(a',b')|.$$

Für die Dreiecksungleichung $d(x,z) + d(z,y) \geq d(x,y)$ ist eine Fallunterscheidung für paarweise Gleichheit von x, y und z auszuführen. Gilt $x = y = z$, so ist die Ungleichung trivial. Für den Fall $x = y \neq z$ gilt $d(x,z) + d(z,y) = 2D - s(x,z) - s(z,y) = 2(D - s(x,z)) \geq 2D$, aber $d(x,y) = d(x,x) = D - s(x,x) \leq D$. Ähnlich sehen auch die anderen Fälle aus.

3) Kann man auf die Bedingung der Symmetrie bei der Definition eines Distanzmaßes bzw. Abstandsmaßes verzichten?

Lösungsvorschlag:
Formal kann man die Symmetrie einfach fortlassen. Was dann entsteht, ist aber (im Falle des Distanzmaßes) keine Metrik mehr.

Aufgabe 2

Modifizieren Sie den Algorithmus von Hirschberg aus der Vorlesung, so daß er auch für das Sequenz-Alignment mit affinen Gap-Funktionen eingesetzt werden kann.

Lösungsvorschlag:

kommt nächste Woche

Aufgabe 3

Implementieren Sie in `C` den Algorithmus zum globalen Sequenz-Alignment mittels Dynamischer Programmierung. Sie dürfen dabei nur linear viel Speicherplatz verwenden. Verwenden Sie also die Erweiterung von Hirschberg.

Lösungsvorschlag:
Aus Gründen der Lesbarkeit präsentieren wir hier keine C-Implementierung des

Hirschberg-Algorithmus, sondern einen Pseudocode. Dieser ist jedoch sehr nahe an der Sprache, so daß eine direkte Übertragung nach C keine zu großen Probleme bereiten sollte.
Seien $S^1_{1\dots m}, S^2_{1\dots n} \in \Sigma^*$ sowie die Scorefunktion $\delta : (\Sigma \cup \text{'–'})^2 \to \mathbf{R}$ gegeben und ein optimales globales Alignment gesucht. Zunächst beschreiben wir eine Funktion, die den Score eines optimalen Alignments zwischen $S^1_{l\dots l'}$ und $S^2_{r\dots r'}$ bis zu einer gegebenen Position h in S_2 berechnet. Das bedeutet, es wird die Dynamic Programming Tabelle für das Alignment der beiden Teilstrings bis zur Zeile h berechnet. Die Funktion berechnet nicht das Alignment selbst, denn sie soll mit linearem Platzbedarf implementiert werden. Was aber berechnet und zurückgegeben wird, ist zumindest die Information, auf welche Weise in den optimalen Alignments von $S^2_{r\dots h}$ und jedem Präfix von $S^1_{l\dots l'}$ die Position h in $S^2_{r\dots h}$ erreicht wird. Das heißt, es wird für die h-te Zeile der Tabelle die Pointer-Information gespeichert und zurückgegeben.

compute-row(l, l', r, r', h) **returns array** $C[\,]$, pointers[ ]

$m := l' - l;$
$P[0] := 0;$
allocate $C[\,]$, pointers[ ]
for $j := 1$ to m **do**
  $P[j] := P[j-1] + \delta(\text{'–'}, S^1_j);$
end for
for $i := 1$ to h **do**
  for $j := 1$ to m **do**
    $C[j] := \max\{P[j]+\delta(S^2_{r+i}, \text{'–'}), C[j-1]+\delta(\text{'–'}, S^1_{l+j}), P[j-1]+\delta(S^2_{r+i}, S^1_{l+j})\};$
    pointers$[j] = \emptyset$;
    if $C[j] = P[j] + \delta(S^2_{r+i}, \text{'–'})$ **then** pointers$[j] \cup=$ '↑';
    elsif $C[j] = C[j-1] + \delta(\text{'–'}, S^1_{l+j})$ **then** pointers$[j] \cup=$ '←';
    elsif $C[j] = P[j-1] + \delta(S^2_{r+i}, S^1_{l+j})$ **then** pointers$[j] \cup=$ '↖';
  end for
  $P := C;$
end for
return $C[\,]$, pointers[ ]

Wir definieren zusätzlich eine Funktion `compute-row`^{r}, die sich wie `compute-row` verhält, aber auf den umgedrehten Versionen der Strings S^1 und S^2 arbeitet.
Man beachte, daß das Array pointers[ ] hier immer wieder überschrieben wird. Nur die Werte am Ende des Aufrufs sind interessant.

Der nun zu implementierende Hirschberg-Algorithmus ist rekursiv und wird dementsprechend auch als rekursiv sich selbst aufrufende Funktion implementiert. Während seines Ablaufs berechnet er den (globalen) Wert C_{opt} eines optimalen Alignments und speichert die zugehörigen Edit-Operationen in der Liste `Pointers`.

Hirschberg(l, l', r, r') **returns sequence** Pointers

Pointers:=λ
— *Base Cases*
if $l < l'$ **then**
  for ($k := r$ **to** r') **do**
    Pointers:= '↑' ∘ Pointers;
    $C_{opt} += \delta(S^2_k)$
  end for
  return Pointers
end if
if $r < r'$ **then**
  for ($k := l$ **to** l') **do**
    Pointers:= '←' ∘ Pointers;
    $C_{opt} += \delta(S^2_k)$
  end for
  return Pointers
end if
— *Punkt k^* in Zeile h der Tabelle und Übergang*
— *zu den benachbarten Zeilen berechnen:*
— *in den Strings S^1 und S^2*
$h := (r + r')/2$;
$(C, \text{pointers}) := \text{compute-row}(l, l', r, r', h)$;
$(C^r, \text{pointers}^r) := \text{compute-row}^r(l, l', r, r', h)$;
$k^* := k'$ mit $C[k'] + C^r[m - k'] = \max\{C[k] + C^r[m - k] \mid 0 \leq k \leq m\}$;
for ($j = k^*$; '←' $\in$ pointers[j]; $j--$) **do**
  Pointers:= '←' ∘ Pointers;
  $C_{opt} += \delta('-', S^1_{l+j})$;
end for
Pointers:= pointers[j] ∘ Pointers;
$C_{opt} += \delta(\text{pointers}[j])$;
if pointers[j] $\neq$ '↑' **then** $j--$ **fi**;
— *und in ihren Umkehrungen*
for ($j^r = m - k^*$; '←' $\in$ pointers^r[j]; j^r--) **do**
  Pointers:=Pointers ∘ '←';
  $C_{opt} += \delta('-', S^1_{l+m-j^r})$;
end for
Pointers:=Pointers ∘ pointers^r[j^r];
$C_{opt} += \delta(\text{pointers}^r[j^r])$;
if pointers^r[j^r] $\neq$ '↑' **then** j^r-- **fi**;
free $C[\,], C'[\,]$, pointers[ ], pointers^r[ ];
Pointers:=**Hirschberg**($r, h-1, l, j$) ∘ **Pointers;**
Pointers:=Pointers ∘ **Hirschberg**($h+1, r', j^r, l'$)**;**
return Pointers

Der erste Aufruf lautet **Hirschberg**$(1, m, 1, n)$. Die durch globale Seiteneffekte belegte Variable C_{opt} und die rekursiv übergebene Sequenz `Pointers` enthalten am Schluß den Score eines optimalen Alignments bzw. das Alignment selbst.