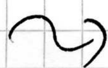
11 : i
8 : ippi
5 : issippi
2 : ississippi
1 : Mississippi
10 : pi
9 : ppi
7 : sippi
4 : sissippi
6 : ssippi
3 : ssissippi

Die Zahl links vom Suffix gibt seine Anfangsposition im String an.

◊

Idee:

Entwickle eine Datenstruktur, die die binäre Suche unterstützt.



Ein Suffixarray für den String x ist ein Feld Pos der Anfangspositionen der Suffixe von x in lexikographischer Ordnung.

Beispiel:

Suffixarray für Mississippi:

Pos = [11, 8, 5, 2, 1, 10, 9, 7, 4, 6, 3]

◊

Eigenschaften: ($n := |x|$)

1) Speicherplatz:

   - Hängt nicht von der Alphabetgröße ab.
   - Wesentlich geringer als für den Suffixbaum.

2) Konstruktionszeit:

   - direkte Konstruktion: $O(n \cdot \log n)$
   - unter Verwendung des Suffixbaumes: $O(n)$

3) Musterstringsuche: (alle Vorkommen)

   - naiv: $O(m \cdot \log n)$, wobei $m := |y|$ ← Musterstring
   - einfach beschleunigt:
     in Praxis $O(m + \log n)$ aber
     im worst case $O(m \cdot \log n)$
   - klever beschleunigt:
     im worst case $O(m + \log n)$.

Wir werden uns zunächst überlegen, wie wir in $O(n)$ Zeit das Suffixarray mit Hilfe eines gegebenen Suffixbaumes konstruieren können. Die Idee hierbei ist, dass wir während der Vorbereitungsphase mehr Speicherplatz verwenden dürfen. Der für den Suffixbaum benötigte Speicherplatz wird nach der Konstruktion des Suffixarrays wieder freigegeben.

Annahme:

Suffixbaum $T(x)$ ist gegeben.

Idee:

Führe auf $T(x)$ eine lexikographische Tiefensuche durch; d.h., von inneren Knoten v aus betrete als nächste Kante diejenige, deren Markierung unter allen noch nicht betretenen ausgehenden Kanten von v mit dem lexikographisch kleinsten Buchstaben beginnt.

Übung

a) Arbeiten Sie die Konstruktion des Suffixarrays $Pos(x)$ eines Textes x mittels lexikographischer Tiefensuche auf einem Suffixbaum $T(x)$ aus.
b) Beweisen Sie die Korrektheit Ihres Verfahrens.

Als nächstes werden wir Verfahren zur Musterstringsuche in einem Text x unter Verwendung des Suffixarrays $Pos(x)$ entwickeln.

Annahme:

Musterstring y, $|y| = m$ soll im Textstring x, $|x| = n$ gesucht werden, wobei das Suffixarray $Pos(x)$ gegeben ist.

Beobachtung:

Die Anfangspositionen aller Vorkommen von y in x sind in $Pos(x)$ aufeinanderfolgend in einem Block gruppiert.

Ziel:

Ermittlung des kleinsten Index min und des größten Index max, so dass y Präfix des im Pos [min] sowie des in Pos [max] stehenden Suffixes ist.

Idee:

Suche zunächst min und dann max mittels binärer Suche auf dem Suffixarray Pos.

Durchführung:

- Bezeichne stets L und R die linke und die rechte Grenze des aktuellen Suchintervalls. D.h., zu Beginn gilt

$$L = 1 \quad \text{und} \quad R = n.$$

- Bezeichne $\leq_{lex}$ die lexikographische Ordnungsrelation.

- Vergleiche in jedem Schritt y und Pos [M], wobei $M := \lceil \frac{R+L}{2} \rceil$ bezüglich ihrer lexikographischen Ordnung.

if $y <_{lex}$ Pos [M]

then

$$R := \lceil \frac{R+L}{2} \rceil$$

else

if Pos [M] $<_{lex}$ y

then $L := \lceil \frac{R+L}{2} \rceil + 1$

else $\quad max := \left\lceil \frac{\ell + L}{2} \right\rceil$

fi

fi.

Übung:
Arbeiten Sie die Berechnung von min und max mittels Binärsuche auf Pos aus.

benötigte Zeit:

- worst case : $O(m \cdot \log n)$
- Diese Zeit wird benötigt, wenn die mit y verglichenen Suffixe meistens lange Präfixe mit y gemeinsam haben.

Frage: Können unnötige Vergleiche vermieden werden?

Bezeichne ℓ bzw. r stets die Länge des gemeinsamen Präfixes von Pos[L] und y bzw von Pos[R] und y.

$\Rightarrow$

Pos[M] und y haben einen gemeinsamen Präfix der Länge $\geq \min\{\ell, r\}$.

$\Rightarrow$

Der Vergleich von Pos[M] mit y könnte in der Position $\min\{\ell, r\} + 1$ begonnen werden.

benötigte Zeit:

im Praxis: $O(m + \log n)$
worst case: $\Omega(m \log n)$.

Ziel:
Modifikation der obigen Idee, dass auch im worst case die Zeit $O(m + \log n)$ benötigt wird.

Die Überprüfung eines Symbols in y heißt redundant, falls dieses Symbol bereits vorher überprüft worden ist. Falls es gelingt die Anzahl der redundanten Überprüfungen pro Iteration der binären Suche auf eins zu begrenzen, dann erreichen wir obiges Ziel.

Für i und j bezeichne $Lcp(i,j)$ die Länge des längsten gemeinsamen Präfixes von $Pos[i]$ und $Pos[j]$.

Beispiel:

x = Mississippi

$lcp(3,4) = 4$, da

$Pos[3]$ = issippi
$Pos[4]$ = ississippi

Frage:

Welche Überprüfungen sind bei einfacher Beschleunigung redundant?

Antwort:

Die Überprüfung von Symbolen in den Positionen von $\min\{l, r\} + 1$ bis $\max\{l, r\}$ sind alle redundant.

Idee:

Verwende $Lcp(L, M)$ und $Lcp(M, R)$ für jedes auftretenden Tripel (L, M, R) um diese Überprüfungen zu vermeiden.

Wir nehmen an, dass jeder der benötigten Werte in konstanter Zeit verfügbar ist. Später werden wir uns überlegen, wie wir dies erreichen.

Durchführung:

- Falls $l = r$, dann können derartige redundante Überprüfungen nicht stattfinden, so dass der Vergleich zwischen y und $Pos[M]$ in Position

$$\min\{l, r\} + 1 = l + 1 = r + 1$$

gestartet werden kann.

- Annahme: $l \neq r$.

Zunächst betrachten wir den Fall $l > r$.

Zur Ermittlung der neuen unteren (und oberen) Grenze des Such=intervals vergleichen wir Lcp(L, M) mit l. Drei Fälle können eintreten.

Lcp(L,M) > l:

D.h., dass das gemeinsame Präfix von Pos[L] und Pos[M] länger ist als das gemeinsame Präfix von y und Pos[L].

$\Longrightarrow$

gemeinsame Präfix von y und Pos[M] hat Länge l.

Da $y >_{lex}$ Pos[L] und Pos[L] und Pos[M] auch in der $l+1$-ten Position gleich sind, folgt

$$\text{Pos}[M] <_{lex} y$$

$\Longrightarrow$

M ist die neue untere Grenze des Such=intervalls. l und r bleiben unverändert.

Lcp(L,M) < l:

D.h., das gemeinsame Präfix von Pos[L] und Pos[M] ist kürzer als das gemeinsame Präfix von y und Pos[L].

$\Longrightarrow$

$$y <_{lex} \text{Pos}[M]$$

$\Longrightarrow$

M ist die neue obere Grenze des Suchintervalls.
l bleibt unverändert. r wird zu $Lcp(L, M)$.

$\underline{Lcp(L, M) = l}$:

D.h., das gemeinsame Präfix von y und Pos[M] hat Länge $\geq l$.

$\leadsto$

Beginnend in Position $l+1$ vergleicht der Algorithmus lexikographisch y und Pos[M].

In Abhängigkeit des Resultats ändert sich die untere oder obere Grenze des Suchintervalls und dementsprechend der korrespondierende Wert l oder r.

Der Fall $l < r$ geht analog.

Übung:
Arbeiten Sie obigen Algorithmus aus.

Satz 2.2

Musterstringsuche mit "clevere" Beschleunigung hat Laufzeit $O(m + \log n)$.

Beweis:

- Konstruktion $\Rightarrow$ l und r wachsen monoton.
- Jede Iteration der binären Suche
  - terminiert die Suche oder
  - überprüft kein Symbol von y oder
  - endet nach dem ersten Mismatch.

Nur in den Fällen $l = r$ und $Lcp(L, M) = l$ werden Symbole in y und $Pos[M]$ lexikographisch miteinander verglichen.

Gestartet wird in Position $\max\{l, r\} + 1$.

Annahme:

In der aktuellen Iteration werden k Symbole überprüft.

$\Rightarrow$

$k-1$ der Überprüfungen waren Matches

$\Rightarrow$

$\max\{l, r\}$ erhöht sich um $k-1$, da entweder l oder r zu diesem Wert übergeht.

$\Rightarrow$

Zu Beginn jeder Iteration gilt:

Das Symbol in Position $\max\{l, r\} + 1$ in y wurde bereits überprüft (mit Ausgang Mismatch). Jedes in y nachfolgende Symbol wurde bisher nicht überprüft.

$\Rightarrow$

Pro Iteration wird höchstens eine redundante Überprüfung durchgeführt.

$\Rightarrow$ Anzahl der redundante Überprüfungen insgesamt:

$$\leq \log n.$$

Weiterhin gilt:

i) Gesamtaufwand pro Iteration in den anderen Fällen : $O(1)$

ii) Anzahl der nicht redundanten Überprüfungen insgesamt : $\leq m$

$\Rightarrow$

Gesamtaufwand: $O(m + \log n)$

■

Als nächstes werden wir uns eine effiziente Berechnung der benötigten Lcp-Werte überlegen.

<u>Frage:</u>

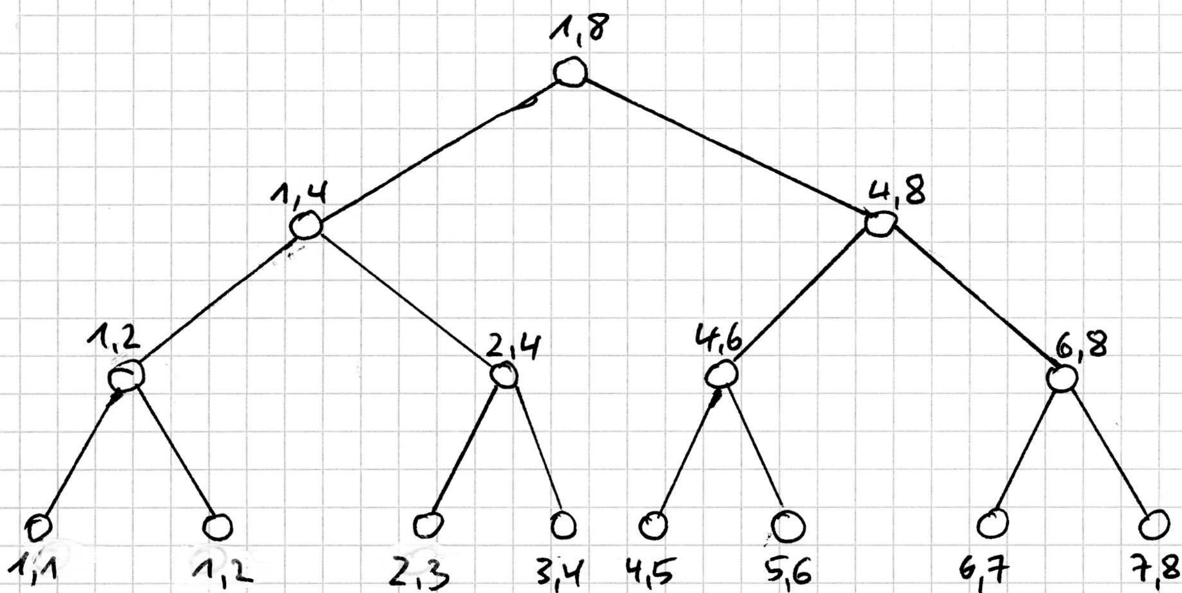
Welche Lcp-Werte werden eventuell benötigt?

Für die Beantwortung dieser Frage nehmen wir der Einfachheit halber an, dass n eine Potenz von zwei ist.

Sei B ein vollständig binärer Baum mit n Blättern. Jeder Knoten in B ist mit einem Paar (i,j), $1 \leq i \leq j \leq n$ wie folgt markiert:

- Markierung der Wurzel: $(1, n)$
- Ein innerknoten mit Markierung (i,j) hat zwei Söhne mit Markierungen $(i, \lfloor \frac{i+j}{2} \rfloor)$ und $(\lfloor \frac{i+j}{2} \rfloor, j)$.

Beispiel: $n = 8$



Die Markierungen der Knoten geben jeweils die untere und die obere Grenze des korrespondierenden Such= intervalls $[L, R]$ an.

Da die Anzahl der Knoten $2n-1$ ist, werden höchstens die zu den Markierungen dieser Knoten korrespondierenden Lcp-Werte benötigt.

Annahme:

An jedem inneren Knoten v in $T(x\$)$ ist $|m(v)|$ gespeichert.

Beobachtung:

- Bezeichne $v(i,j)$ denjenigen inneren Knoten in $T(x\$)$, in dem sich die Pfade von dem zu $Pos[i]$ bzw. $Pos[j]$ korrespondierenden Blatt zur Wurzel treffen. Dann gilt:

$$Lcp(i,j) = |m(v(i,j))|.$$

$\Rightarrow$

Die benötigten Lcp-Werte können während der lexikographischen Tiefensuche auf $T(x\$)$ mit berechnet werden.

Übung:

Erweitern Sie die lexikographische Tiefensuche, so dass auch die benötigten Lcp-Werte mit berechnet werden.

3. Approximatives Stringmatching

Ziel:

Durchführung von approximatives String=matching, d.h., finden von zu einem Musterstring "ähnliche" Teilstrings in einem Textstring.

weitere Literatur:

N. Blum, Speeding up dynamic pro=gramming without omitting any optimal solution and some applications in mole=cular biology, J. of Algorithms 35 (2000), 129 - 168.

Approximatives Stringmatching sucht nicht notwendigerweise exakte Vorkommen eines Musterstrings im Textstring, sondern nahezu exakte Vorkommen.

Es stellt sich nun folgende Frage:

Was bedeutet "nahezu exakt"?

<u>Ziel</u>:
Präzisierung von "nahezu exakt" nebst exakter Formulierung der Aufgabenstellung.

3.1 Die Editierdistanz zweier Sequenzen

Seien $x = x_1 x_2 \ldots x_n$ und $y = y_1 y_2 \ldots y_m$ zwei Strings über einem endlichen Alphabet Σ.

<u>Beispiel:</u>

Alphabete in genetischen Sequenzen

- $\Sigma = \{A, G, C, T\}$,

  falls x eine DNA-Sequenz ist.

- $\Sigma = \{$ alanine, arginine, ..., valine $\}$,

  falls x eine Proteinsequenz ist.

◇

Genetische Sequenzen können in der evolutionären Geschichte durch <u>Mutationen</u> geändert werden. Dies möchten wir nun modellieren.

<u>Modell:</u>

Mutation $\hat{=}$ $(x,y) \in \Sigma^+ \times \Sigma^+$

<u>Interpretation:</u> Sequenz x wird durch Sequenz y ersetzt.

$M \subseteq \Sigma^+ \times \Sigma^+$ Menge der betrachteten Mutationen

$c: M \to \mathbb{R}$ Kostenfunktion.

Eine Folge $S := s_1, s_2, \ldots, s_t$ heißt genau dann <u>Mutationenfolge über M</u>, wenn es Strings $z_0, z_1, \ldots, z_t \in \Sigma^+$ gibt, so dass $(z_{i-1}, z_i) = s_i \in M$ für alle $1 \leq i \leq t$. Wir sagen, <u>S transformiert z_0 nach z_t</u>. $z_0, z_1, \ldots, z_t$ heißt <u>M-Ableitung von z_t aus z_0</u>

$$c(S) := \sum_{i=1}^{t} c(s_i)$$

sind die <u>Kosten der Mutationenfolge S.</u>

M heißt genau dann <u>vollständig</u>, wenn für alle $(x,y) \in \Sigma^+ \times \Sigma^+$ eine M-Ableitung von y aus x existiert.

Der <u>evolutionäre Abstand</u> oder auch <u>Editier-abstand</u> $d_M(x,y)$ der Sequenzen x und y ist definiert durch

$$d_M(x,y) := \min\{ c(S) \mid S \text{ transformiert } x \text{ nach } y \}.$$

<u>Frage:</u> Was ist eine vernünftige Wahl für M und c?

<u>Anforderungen an M:</u>

1) M muss Anforderungen aus der Molekular=biologie genügen, d.h., die Wirklichkeit mög=lichst exakt widerspiegeln.
2) M muss Anforderungen aus der Informatik genügen, d.h., die resultierenden Probleme müssen praktisch mit dem Computer lösbar sein.

Üblicherweise werden nur Mutationen betrachtet, die als lokale Operationen beschreibbar sind. Dann ist es möglich, die Menge aller betrachteten Muta=tionen durch eine endliche Menge von Operationen zu beschreiben.

<u>M_1, eine erste Wahl für M:</u>

M_1 enthält folgende vier Typen von Operationen:

a) <u>Streichen</u> eines Symbols aus x.
b) <u>Einfügen</u> eines Symbols in x.
c) <u>Ersetzen</u> eines Symbols durch ein anderes in x.
d) <u>Substitution</u> eines Symbols durch sich selbst in x.

Obwohl eine Substitution keine Mutation modelliert, ist es aus Gründen der Darstellung sinnvoll, sie auch in die Operationenmenge aufzunehmen. Da wir Mutationen positive Kosten zuordnen, haben Substitutionen Kosten ≤ 0.

Formal ist eine Operation in M_1 ein Paar $(a,b) \in (\Sigma \cup \{-\})^2 \setminus \{(-,-)\}$, $- \notin \Sigma$, wobei

$$(a,b) = \begin{cases} \text{Streichen von } a & \text{falls } a \in \Sigma,\ b = - \\ \text{Einfügen von } b & \text{falls } a = -,\ b \in \Sigma \\ \text{Ersetzen von } a \text{ durch } b & \text{falls } a \in \Sigma,\ b \in \Sigma \setminus \{a\} \\ \text{Substitution von } a \text{ für } a & \text{falls } a = b \in \Sigma \end{cases}$$

<u>Bemerkung:</u>

- M_1 ist vollständig
- Die Kostenfunktion c ist nur sinnvoll, falls

$$c(a,b) < c(a,-) + c(-,b) \quad \forall a \in \Sigma,\ b \in \Sigma \setminus \{a\}$$

Eine in der Molekularbiologie übliche Darstellungsweise einer Ableitung einer Sequenz y aus einer Sequenz x ist das korrespondierende, nachfolgend definierte Alignment $A(x,y)$.

Ein <u>Alignment</u> $A(x,y)$ zwischen x und y besteht aus einer zweizeiligen Matrix, so dass gilt:

a) Die erste Zeile besteht aus x, möglicherweise mit Nullsymbolen – durchsetzt.

b) Die zweite Zeile besteht aus y möglicherweise mit Nullsymbolen – durchsetzt.

c) Es gibt keine Spalte, die nur aus Nullsymbolen – besteht.

Eine Spalte $\begin{bmatrix} a \\ b \end{bmatrix}$ beschreibt die Operation (a,b). Die Kosten $c_{M_1}(A(x,y))$ eines Alignments $A(x,y)$ sind definiert durch

$$c_{M_1}(A(x,y)) := \sum_{\begin{bmatrix} a \\ b \end{bmatrix} \in A(x,y)} c_{M_1}(a,b),$$

wobei $c_{M_1}: M_1 \to \mathbb{R}$ die Kostenfunktion bezüglich der Operationenmenge M_1 ist. Dabei werden gleiche Spalten in unterschiedlichen Positionen des Alignments in obiger Summe als verschiedene Spalten behandelt.

Beobachtung:

Ein Alignment $A(x,y)$ beschreibt eine Menge von Ableitungen von y aus x. Zwei dieser Ableitungen unterscheiden sich lediglich in der Anordnung der Operationen. Demzufolge haben alle diese Ableitungen dieselben Kosten $c_{M_1}(A(x,y))$.

Ziel:

Entwicklung eines Verfahrens zur Berechnung der Editierdistanz zweier gegebenen Strings.

Sei $x = x_1 x_2 \dots x_n$ ein String über ein Alphabet Σ. Dann bezeichnet

$$x^j := \begin{cases} \varepsilon & \text{falls } j = 0 \\ x_1 x_2 \dots x_j & \text{falls } j > 0. \end{cases}$$

Folgender Satz liefert uns eine Rechenvorschrift zur Berechnung der Editierdistanz zweier Strings.

<u>Satz 3.1</u>

Seien $x = x_1 x_2 \dots x_n$ und $y = y_1 y_2 \dots y_m$ zwei Strings. Dann gilt für $0 \le j \le n$, $0 \le i \le m$:

$$d_{M_1}(x^j, y^i) = \min \begin{cases} d_{M_1}(x^{j-1}, y^i) + c_{M_1}(x_j, -) \\ d_{M_1}(x^j, y^{i-1}) + c_{M_1}(-, y_i) \\ d_{M_1}(x^{j-1}, y^{i-1}) + c_{M_1}(x_j, y_i) \end{cases}$$

<u>Beweis</u>:

Oben haben wir uns überlegt, dass für die Kosten eines Alignments die Reihenfolge, in der die Operationen durchgeführt werden, keine Rolle spielt. Ein optimales Alignment von x^j und y^i kann die Operation (x_j, y_i) enthalten oder nicht.

Ein bestes Alignment, das (x_j, y_i) enthält, erhalten wir, indem wir ein optimales Alignment für x^{j-1} und y^{i-1} nehmen und als letzte Spalte die Spalte $\begin{bmatrix} x_j \\ y_i \end{bmatrix}$ hinzufügen.

Falls ein optimales Alignment (x_j, y_i) nicht enthält, dann muss dieses die Operation $(x_j, -)$ oder die Operation $(-, y_i)$ enthalten.

Obige Minimumsbildung wählt unter diesen drei Alternativen eine optimale aus.

$\Rightarrow$

Behauptung.

■

Obiger Satz impliziert die Korrektheit des folgenden Algorithmus zur Berechnung der Editierdistanz zweier Strings.

<u>Algorithmus</u> EdDisM_1

<u>Eingabe:</u> Strings $x = x_1 x_2 \dots x_n$, $y = y_1 y_2 \cdots y_m \in \Sigma^+$, Kostenfunktion c_{M_1}.

<u>Ausgabe:</u> Editierdistanz $d_{M_1}(x, y)$

<u>Methode:</u>

$e(x^0, y^0) := 0$;
<u>for</u> j <u>from</u> 1 <u>to</u> n
<u>do</u>
$e(x^j, y^0) := e(x^{j-1}, y^0) + c_{M_1}(x_j, -)$
<u>od</u>;
<u>for</u> i <u>from</u> 1 <u>to</u> m
<u>do</u>
$e(x^0, y^i) := e(x^0, y^{i-1}) + c_{M_1}(-, y_i)$
<u>od</u>;

for j **from** 1 **to** n

  do

    for i **from** 1 **to** m

      do

$$e(x^j, y^i) := \min \begin{cases} e(x^{j-1}, y^i) + c_{M_1}(x_j, -) \\ e(x^j, y^{i-1}) + c_{M_1}(-, y_i) \\ e(x^{j-1}, y^{i-1}) + c_{M_1}(x_j, y_i) \end{cases}$$

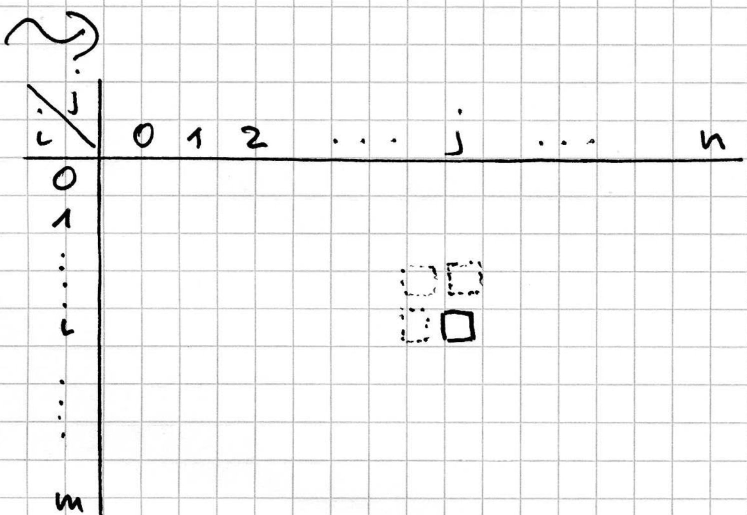
      od

  od ;

$d_{M_1}(x, y) := e(x^n, y^m)$.

Fragen:

- Wie implementieren wir den Algorithmus $EDDis_{M_1}$? Welche Datenstruktur verwenden wir?
- Wie erhalten wir ein oder alle zu optimalen Ableitungen korrespondierenden Alignments?
- Was ist die Zeit- und die Platzkomplexität des Algorithmus?



$(m+1) \times (n+1)$ - Matrix Γ

In Position $[i,j]$ der Matrix wird der Wert $e(x^j, y^i)$ eingetragen.

Ziel:

Berechnung aller zu den optimalen Ableitungen korrespondierenden Alignments.

Beobachtung:

Es genügt, sich zusätzlich zum Wert $e(x^j, y^i)$ genau diejenigen der drei Alternativen zu merken, die zum Minimum führen.

Hierzu betrachten wir den zur obigen Matrix korrespondierenden Editiergraphen $\mathcal{E}_{\alpha_1}(x,y) := (V,E)$, wobei

$$V := \{ [i,j] \mid 0 \leq i \leq m,\ 0 \leq j \leq n \} \text{ und}$$

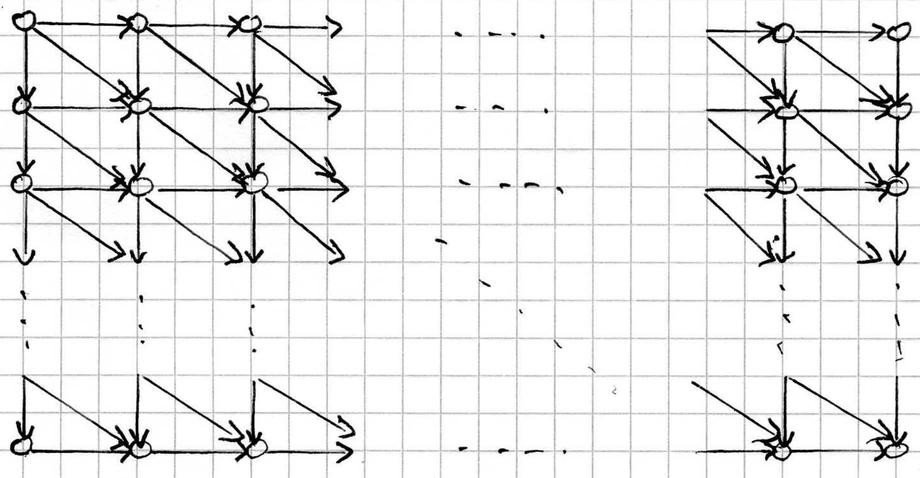
$$E := \{ ([i,j],[i,j+1]), ([i,j],[i+1,j]), ([i,j],[i+1,j+1]) \mid 0 \leq i < m,\ 0 \leq j < n \}$$
$$\cup \{ ([m,j],[m,j+1]) \mid 0 \leq j < n \}$$
$$\cup \{ ([i,n],[i+1,n]) \mid 0 \leq i < m \}.$$

Jede Kante $e := ([i,j],[i',j'])$ korrespondiert auf folgende Art und Weise zu einer Editieroperation $op(e)$:

$$op(e) = \begin{cases} \text{Streichen von } x_{j+1} & \text{falls } i' = i,\ j' = j+1 \\ \text{Einfügen von } y_{i+1} & \text{falls } i' = i+1,\ j' = j \\ \text{Ersetzen bzw. Substitution von } x_{j+1} \text{ durch } y_{i+1} & \text{falls } i' = i+1,\ j' = j+1 \end{cases}$$

Mit der Kante e assoziieren wir die Kosten $c_{M_1}(op(e))$. Die Kosten $c_{M_1}(P)$ eines Pfades P in $\mathcal{E}_{M_1}(x,y)$ sind definiert durch

$$c_{M_1}(P) := \sum_{e \in P} c_{M_1}(op(e)).$$



Editiergraph $\mathcal{E}_{M_1}(x,y)$.

Der <u>Distanzgraph</u> $D_{M_1}(x,y) = (V, E_d)$ ist dann definiert durch

$$E_d := \{([i,j],[i',j']) \in E \mid e(x^{j'}, y^{i'}) = e(x^j, y^i) + c_{M_1}(op(([i,j],[i',j'])))\}.$$

Übung:

a) Zeigen Sie, dass es eine Bijektion zwischen den Pfaden vom Knoten $[0,0]$ zum Knoten $[m,n]$ in $D_{M_1}(x,y)$ und den Alignments $A(x,y)$ mit $c_{M_1}(A(x,y)) = d_{M_1}(x,y)$ gibt.

b) Bauen Sie in den Algorithmus EDDISM_1 die Konstruktion des Distanzgraphen ein.

c) Analysieren Sie die Zeit- und die Platzkomplexität des unter b) erweiterten Algorithmus EDDISM_1.

Insgesamt haben wir somit folgenden Satz bewiesen:

Satz 3.2

Seien $x = x_1 x_2 \dots x_n$ und $y = y_1 y_2 \dots y_m$ zwei Strings. Dann kann der Editierabstand $d_{M_1}(x,y)$ von x und y und der Distanzgraph $D_{M_1}(x,y)$ in Zeit $O(m \cdot n)$ und Platz $O(m \cdot n)$ berechnet werden.

Übung:

Zeigen Sie, dass $d_{M_1}(x,y)$ auch in $O(\min\{m,n\})$ Platz und $O(m \cdot n)$ Zeit berechnet werden kann.

M_2, eine zweite Wahl für M:

In der evolutionären Geschichte genetischer Sequenzen tritt das Streichen bzw. Einfügen eines

Strings der Länge ≥ 2 als einzelne Mutation auf. M_1 erlaubt jedoch die Behandlung einer solchen Mutation nur als Folge von k Einzeloperationen, deren Kosten sich addieren. Diese Vorgehensweise wird von Molekularbiologen als nicht sinnvoll betrachtet. Daher erweitern wir M_1 durch Hinzufügen dieser Operationen zu einer Operationenmenge M_2.
M_2 enthält demnach folgende Typen von Operationen:

a) <u>Streichen</u> eines Strings der Länge k, $k \geq 1$ aus x.
b) <u>Einfügen</u> eines Strings der Länge k, $k \geq 1$ in x.
c) <u>Ersetzen</u> eines Symbols durch ein anderes in x.
d) <u>Substitution</u> eines Symbols durch sich selbst in x.

Somit ist eine Operation in M_2 ein Paar
$(a,b) \in (\Sigma^+ \times \{-\}) \cup (\{-\} \times \Sigma^+) \cup (\Sigma \times \Sigma)$,
wobei

$$(a,b) = \begin{cases} \text{Streichen von } a & \text{falls } a \in \Sigma^+, b = - \\ \text{Einfügen von } b & \text{falls } a = -, b \in \Sigma^+ \\ \text{Ersetzen von } a \text{ durch } b & \text{falls } a \in \Sigma, b \in \Sigma \setminus \{a\} \\ \text{Substitution von } a \text{ für } a & \text{falls } a = b \in \Sigma \end{cases}$$

Üblicherweise sind die Kosten einer Einfüge- bzw. Streicheoperation derart definiert, dass diese nur von der Länge des Operanden und nicht vom Operanden selbst abhängen. D.h., die Kosten sind definiert durch eine Funktion

$w_{id}: \mathbb{N} \rightarrow \mathbb{R}^+$ und eine Konstante $d > 0$.

Mit jeder Operation (a,b) assoziieren wir die Kosten $c_{M_2}(a,b)$, wobei

$$c_{M_2}(a,b) := \begin{cases} d + w_{id}(|a|) & \text{falls } b = - \\ d + w_{id}(|b|) & \text{falls } a = - \end{cases}$$

und ansonsten wie $c_{M_1}(a,b)$ definiert ist.

d ist die "Strafe" dafür, dass überhaupt eine Streiche- bzw. Einfügeoperation erfolgt und $w_{id}(|a|)$ bzw. $w_{id}(|b|)$ ist die "Strafe" für die Länge des Operanden.

Übung:

Erweitern Sie die bzgl. M_1 entwickelten Methoden auf M_2. Genauer:

a) Definieren Sie evolutionäre Distanz, Alignment, Editier- und Distanzgraph für M_2.

b) Formulieren Sie bzgl. M_2 einen zu Satz 2.1 analogen Satz und beweisen Sie diesen.

c) Modifizieren Sie den Algorithmus EDDisM_1, so dass die evolutionäre Distanz und der Distanzgraph bzgl. M_2 in Zeit und Platz $O(m^2n + mn^2)$ berechnet werden.

M_3, eine dritte Wahl für M

Üblicherweise definiert man $c_{M_2}(a,a) := 0 \quad \forall a \in \Sigma$. Mitunter ist es nützlich, wenn die Belohnung für gleiche Teilsequenzen mit deren Länge wächst. Da die Strafe für Mutationen positiv ist, muss die Belohnung für Gleichheit negativ sein. Analog zum Streichen oder Einfügen eines Strings der Länge > 1 betrachten wir die Substitution eines Strings der Länge > 1. Daher erweitern wir wie folgt die Operationenmenge M_2 zur Operationenmenge M_3:

a) <u>Streichen</u> eines Strings der Länge k, $k \geq 1$ aus x.
b) <u>Einfügen</u> eines Strings der Länge k, $k \geq 1$ in x.
c) <u>Ersetzen</u> eines Symbols durch ein anderes in x.
d) <u>Substitution</u> eines Teilstrings der Länge $k \geq 1$ durch sich selbst in x.

Wir definieren die Kosten einer Substitution derart, dass die Belohnung nur von der Länge des Operanden und nicht vom Operanden selbst abhängt. D.h., die Belohnung ist definiert durch eine Funktion

$$w_m : \mathbb{N} \to \mathbb{R}^- \text{ und einer Konstanten } 0 < g < |w_m(1)|.$$

Mit jeder Operation $(a,a) \in M_3$ assoziieren wir die Belohnung $c_{M_3}(a,a)$, wobei

$$c_{M_3}(a,a) := g + w_m(|a|).$$

Biologisch sinnvoll scheint, dass der Anteil an

Belohnung mit der Länge des substituierten Teil=strings wächst. Die Konstante g stellt sicher, dass eine lange Substitution stets günstiger, als zwei aufeinanderfolgende kürzere Substitutionen ist.

<u>Übung:</u>
Erweitern Sie die bzgl. M_2 entwickelten Methoden auf M_3.

<u>Biologisch relevante Kostenfunktionen</u>

Als biologisch relevant wird von Molekularbiologen angesehen, dass der Kostenzuwachs für Streiche- bzw. Einfügeoperationen mit der Länge des Operan=den abnimmt. Solche Funktionen heißen in der Mathematik <u>konkav</u>. Formaler

$w: \mathbb{N} \to \mathbb{R}$ heißt <u>konkav</u>, falls $\forall k \geq 1$

$$w(k+1) - w(k) \geq w(k+2) - w(k+1).$$

Da Belohnung negativ ist, wäre dann eine Funk=tion w_m, die den Anteil an Belohnung mit der Länge des substituierten Teilstrings wachsen lässt, konkav.

Konkave Kostenfunktionen lassen effizientere Algorithmen zu.

3.2. Der Algorithmus von Galil und Giancarlo

Z. Galil, R. Giancarlo, Speeding up dynamic programming with applications to molecular biology, TCS 64 (1989), 107 – 118.

Wir betrachten nachfolgend nur konkave Kostenfunktionen.

Ziel:
Beschleunigung der Algorithmen aufgrund der Konkavität der Kostenfunktionen.

Betrachten wir noch einmal die oben definierte $(m+1) \times (n+1)$ - Matrix Γ. $v(i,j)$ bezeichnet die minimale Kosten für das zur Position $[i,j]$ in Γ korrespondierende Teilproblem.

Für $0 \le i \le m$, $0 \le j \le n$ erhalten wir

$$v(i,j) = \begin{cases} 0 & \text{falls } i=j=0 \\ \min\{rmin(i,j), cmin(i,j), dmin(i,j)\} & \text{sonst,} \end{cases}$$

wobei

- $rmin(i,j) := \min\limits_{0 \le k < j} \{ v(i,k) + c([i,k],[i,j]) \}$,

- $cmin(i,j) := \min\limits_{0 \le \ell < i} \{ v(\ell,j) + c([\ell,j],[i,j]) \}$

und

- $dmin(i,j) := \min_{1 \le q \le \min\{i,j\}} \{ v(i-q, j-q) + c([i-q, j-q], [i,j]) \}$.

Dabei bezeichnet $c([i',j'],[i,j])$ die zusätzlichen Kosten für eine Lösung des Teilproblems $[i,j]$, gestartet mit einer Lösung des Teilproblems $[i',j']$.

Für $1 \le i \le m$ sei

$$RMIN(i) := [rmin(i,0), rmin(i,1), \ldots, rmin(i,n)].$$

Im folgenden betrachten wir i fest.

Jeder Pfad P im Distanzgraphen, der für $RMIN(i)$ von Relevanz ist, hat eine letzte Kante e, so dass beide Endknoten von e in Zeile i sind. Ein derartiger Pfad heißt <u>Zeilenlösung</u>.

<u>Annahme</u>:
Für $0 \le k < p$ sind die Kosten $v(i,k)$ eines optimalen Pfades vom Knoten $[0,0]$ zum Knoten $[i,k]$ im Distanzgraphen berechnet.

<u>Ziel</u>:
Berechnung einer optimalen Zeilenlösung P mit letzten Knoten $[i,p]$ auf P.

<u>1. Lösungsmethode</u>:

- Betrachte alle $0 \le k < p$ und berechne die Kosten

$v(i,k) + c([i,k],[i,p])$. Nimm eine Lösung mit minimalen Kosten.

Aufwand: $O(p)$

Gesamtaufwand für RMIN(i): $\sum_{p=1}^{n} p = O(n^2)$.

Ziel:
Für konkave Kostenfunktionen c möchten wir ein Verfahren zur Berechnung von RMIN(i) entwickeln, welches lediglich die Zeit $O(n \log n)$ benötigt.

Idee:
Beschränke die Anzahl der Kandidaten $k < p$, die betrachtet werden müssen, auf eins.

Betrachten wir hierzu diejenige optimale Zeilenlösung nach $[i,p-1]$ mit längster letzten Kante $([i,k],[i,p-1])$.

$\Rightarrow$

Für jede andere optimale Zeilenlösung nach $[i,p-1]$ gilt $k' > k$, wobei $[i,k']$ der Anfangsknoten der letzten Kante $([i,k'],[i,p-1])$ dieser Zeilenlösung ist.

Konkavität der Kostenfunktion c $\Rightarrow$

Eine beste Zeilenlösung, die die Kante $([i,k],[i,p])$ verwendet, kann nicht schlechter sein, als eine Zeilenlösung, die die Kante $([i,k'],[i,p])$, $k < k' < p-1$ verwendet.

$\Rightarrow$

Der Knoten $[i,k]$ <u>dominiert</u> alle Knoten $[i,k']$ mit $k < k' < p-1$ <u>bezüglich $[i,p]$</u>.

<u>Beobachtung:</u>
Falls auf jeden optimalen Pfad von $[0,0]$ nach $[i,p-1]$ als letzte Kante eine Spalten- oder Diagonalenkante liegt und auch

$$v(i,p-1) + c([i,p-1],[i,p]) < v(i,k) + c([i,k],[i,p]),$$

dann gilt:

Eine Zeilenlösung, die die Kante $([i,p-1],[i,p])$ verwendet, ist kostengünstiger, als jede Zeilenlösung, die die Kante $([i,k],[i,p])$ verwendet.

Dann sagen wir, $[i,p-1]$ <u>dominiert</u> $[i,k]$ bzgl. $[i,p]$.

Da die Länge der Kante $([i,k],[i,p])$ größer ist, als die Länge der Kante $([i,p-1],[i,p])$, stellt sich unmittelbar folgende Frage:

Kann es sein, dass zwar $[i,p-1]$ den Knoten $[i,k]$ bzgl. $[i,p]$ dominiert, jedoch für ein $p' > p$ gilt: $[i,k]$ dominiert $[i,p-1]$ bzgl. $[i,p']$?

Zur Beantwortung dieser Frage betrachten wir nun die Knoten $[i,p+1]$, $[i,p+2]$, ... und die Kanten $([i,p-1],[i,p+1])$, $([i,p-1],[i,p+2])$, ... sowie

die Kanten $([i,k],[i,p+1]), ([i,k],[i,p+2]), \ldots$

Beobachtung:

Zunächst können auch die Kanten $([i,p-1],[i,p+1])$, $([i,p-1],[i,p+2]), \ldots$ eine Zeit lang zu besseren Zeilenlösungen führen, als die Kanten $([i,k],[i,p+1])$ $([i,k],[i,p+2]), \ldots$.

Aber:

Aufgrund der Konkavität der Kostenfunktion c kann $p' > p$ existieren, so dass $([i,k],[i,p'])$ zu einer Zeilenlösung führt, die nicht schlechter ist, als eine Zeilenlösung, die die Kante $([i,p-1],[i,p'])$ verwendet.

Konkavität $\Rightarrow$ Für $p'' \geq p'$ würde dies dann immer der Fall sein.

$\Rightarrow$

Das kleinste $p' > p$ mit der Eigenschaft, dass $([i,k],[i,p'])$ zu einer Zeilenlösung führt, die nicht schlechter ist, als eine Zeilenlösung, die $([i,p-1],[i,p'])$ verwendet, ist von Interesse.

Betrachten wir nun $k' < k$. Dann gilt aufgrund der Wahl des Knotens $[i,k]$:

$[i,k]$ dominiert $[i,k']$ bzgl $[i,p-1]$

$\Rightarrow$

$[i,k']$ dominiert genau dann $[i,k]$ bzgl. $[i,p]$, wenn p der kleinste Wert ist, so dass $([i,k'],[i,p])$ zu einer Zeilenlösung führt, die nicht schlechter ist, als eine Zeilenlösung, die die Kante $([i,k],[i,p])$ verwendet.

Somit führt die gesamte Betrachtung zu folgender Definition:

Ein Knoten $[i,k]$ p-dominiert genau dann den Knoten $[i,k']$, wenn eine der folgenden zwei Bedingungen erfüllt ist:

1) $k < k'$ und $v(i,k) + c([i,k],[i,p]) \leq v(i,k') + c([i,k'],[i,p])$

2) $k > k'$ und $v(i,k) + c([i,k],[i,p]) < v(i,k') + c([i,k'],[i,p])$.

Beobachtung 1:
Falls $[i,k]$ einen Knoten $[i,k']$ p-dominiert und $k < k'$, dann p'-dominiert $[i,k]$ den Knoten $[i,k']$ für alle $p' > p$.

Wir kombinieren die beiden Fälle in obiger Definition wie folgt:

Sei $0 \leq k < j \leq p \leq n$. Der Knoten $[i,k]$ heißt genau dann **$\langle j,p \rangle$ - dominant**, wenn für alle $k' < j$ der Knoten $[i,k]$ den Knoten $[i,k']$ p-dominiert.

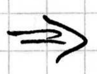
Die Konkavität der Kostenfunktion c impliziert folgende Beobachtung:

<u>Beobachtung 2:</u>

a) $\forall\ 0 < j \leq p \leq n$ existiert genau ein Knoten $[i,k]$, der $\langle j,p \rangle$-dominant ist. Bezeichne $dom(j,p)$ diesen eindeutigen Knoten.

b) Sei j fest und $[i,k_p] := dom(j,p)$. Dann gilt: $\quad q > p \Rightarrow k_q \leq k_p$.

c) $rmin(i,j) = v(i,k) + c([i,k],[i,j])$, wobei $[i,k] := dom(j,j)$.



Zu dem Zeitpunkt, wenn wir eine optimale Zeilen=lösung nach $[i,p]$ berechnen möchten, genügt die Kenntnis des eindeutigen $\langle p,p \rangle$ - dominanten Knotens, um $rmin(i,p)$ in konstanter Zeit zu be=rechnen.

<u>Frage:</u>
Was ist eine vernünftige Datenstruktur zur Verwal=tung derjenigen Knoten, die eventuell irgendwann dominant werden?

Oben haben wir uns überlegt, dass ein Knoten $[i,k]$ dominant werden kann und daher der bisherig dominante Knoten $[i,k']$ diese Eigenschaft verliert. Wenn der Knoten $[i,k']$ wieder dominant werden sollte, dann wird danach $[i,k]$ niemals wieder dominant werden.

$\Rightarrow$

Ein Keller scheint eine vernünftige Datenstruktur zu sein.

<u>Idee</u>

Anstatt für $[i,k]$ jedes $[i,p]$ mit der Eigenschaft, dass $[i,k]$ $\langle j,p\rangle$-dominant ist, zu speichern, genügt es, den Knoten $[i,p]$ mit <u>kleinsten</u> $p \geq j$ und $[i,k]$ ist $\langle j,p\rangle$-dominant zu speichern.

Daher definieren wir für $0 \leq k < j \leq n$

$$p_j(k) := \min_{j \leq p \leq n} \{ p \mid [i,k] = dom(j,p) \}.$$

Falls $p_j(k)$ definiert ist, dann nennen wir die Kante $([i,k],[i,p_j(k)])$ <u>$\langle k,j\rangle$-wesentlich</u>.

<u>Durchführung:</u>

Zur Berechnung von RMIN(i) betrachten wir die i-te Zeile der Matrix Γ von links nach rechts.

<u>Annahme:</u>

Zum Zeitpunkt, wenn der Knoten $[i,j]$ betrachtet

wird, sind in einem Keller $S(j)$ genau die $\langle k,j\rangle$-wesentlichen Kanten bzgl. k monoton steigend vom Kellerboden zum obersten Kellerelement abgelegt.

$\Longrightarrow$

Die zweiten Komponenten der Kantenköpfe sind monoton fallend abgelegt.

<u>Bezeichnungen:</u>

$TOP(S(j))$	oberste Kante in $S(j)$
$SEC(S(j))$	zweitoberste Kante in $S(j)$
$S(0)$	leerer Keller

<u>Annahme:</u> $TOP(S(j)) = (dom(j,j), [i,j])$

Zusammen mit Beobachtung 2c folgt somit, dass $rmin(i,j)$ in konstanter Zeit berechnet werden kann.

Sei

$$S(j) = [([i,k_1],[i,p_1]), ([i,k_2],[i,p_2]), \ldots, \underset{\text{TOP}\downarrow}{([i,k_r],[i,p_r])}].$$

Dann gilt:

1) $p_r = j$
2) $k_r = j-1 \iff$
   - Jeder optimale Pfad von $[0,0]$ nach $[i,j-1]$ hat eine Spalten- oder eine Diagonalen-Kante als letzte Kante und
   - $[i,j-1]$ j-dominiert $[i,k_{r-1}]$.

<u>Frage:</u>

Wie erhalten wir nach der Betrachtung des Knotens $[i,j]$ den Keller $S(j+1)$?

Hierzu modifizieren wir den Keller $S(j)$.

<u>Modifikation von $S(j)$:</u>

- Da die Kante $([i,k_r],[i,j])$ nicht $\langle k_r, j+1\rangle$-wesentlich sein kann, verfahren wir wie folgt:

$$\begin{cases} \text{Ersetze den Kopf dieser Kante durch } [i,j+1] & \text{falls } kopf(SEC(S_j)) \neq [i,j+1] \\ \text{Führe Pop-Operation aus} & \text{sonst.} \end{cases}$$

Sei

$$S'(j) = [([i,k_1],[i,p_1]),([i,k_2],[i,p_2]),\ldots,([i,k_t],[i,p_t])]$$

der auf diese Art und Weise modifizierte Keller.

$\Rightarrow$

$$p_t = j+1.$$

Falls $v(i,j)$ bezüglich einer Zeilenkante definiert wurde, dann gilt $S(j+1) = S'(j)$ und wir sind fertig.

<u>Annahme:</u>
$v(i,j)$ wurde nicht bezüglich einer Zeilenkante definiert.

$\Rightarrow$

Oben auf $S'(j)$ sind möglicherweise einige Kanten $([i,k_e],[i,p_e])$, die nicht $\langle k_e, j+1\rangle$ - wesentlich sind.

<u>Ziel</u>:

Entfernung dieser Kanten aus dem Keller.

Beobachtungen 1 und 2 $\Rightarrow$

Wir können den Keller von oben nach unten inspizieren und entscheiden, welche Kanten zu entfernen sind. Dies sieht man folgendermaßen ein:

Sei

$$([i,k_s],[i,p_s])$$

die oberste Kante in $S'(j)$, die $\langle k_s, j+1\rangle$ - wesentlich ist.

$\Rightarrow$

$([i,k_s],[i,p_s])$ ist die oberste Kante, so dass der Knoten $[i,k_s]$ nicht durch $[i,j]$ p_s - dominiert wird.

$\Rightarrow$

Folgende Bedingungen sind erfüllt:

1) $v(i,k_s) + c([i,k_s],[i,p_s]) \leq v(i,j) + c([i,j],[i,p_s])$

und

2) Falls $s < t$, dann gilt

$$v(i, k_{s+1}) + c([i,k_{s+1}],[i,p_{s+1}]) > v(i,j) + c([i,j],[i,p_{s+1}]).$$

$\Longrightarrow$

i) Genau die Kanten oberhalb $([i,k_s],[i,p_s])$ in $S'(j)$ sind zu entfernen.

ii) Falls $s < t$, dann kann eine der Kanten mit Anfangsknoten $[i,k_{s+1}]$ und Kopf $[i,p]$, $p_{s+1} < p < p_s$ $\langle k_{s+1}, j+1\rangle$ - wesentlich sein.

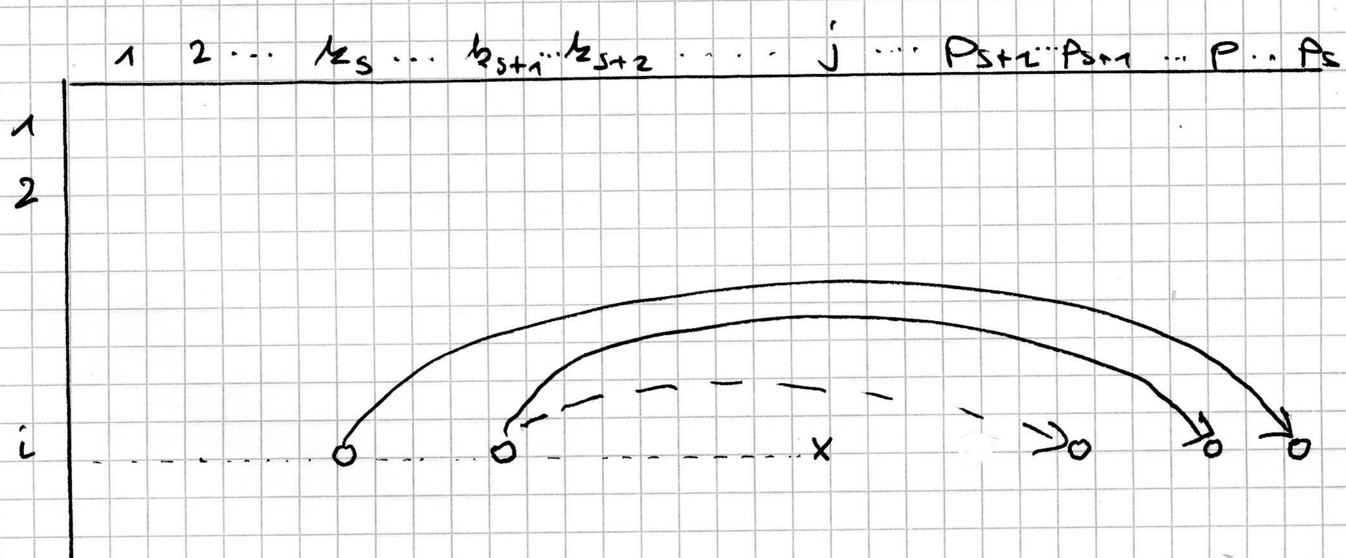
iii) Sei $p \in \{p_{s+1}+1, p_{s+1}+2, \dots, p_s - 1\}$ minimal, so dass $([i,k_{s+1}],[i,p])$ $\langle j+1, p\rangle$ - dominant.

Beobachtungen 1 und 2 $\Longrightarrow$

$([i,k_{s+1}],[i,p])$ ist die einzige $\langle k_{s+1}, j+1\rangle$ - wesentliche Kante.

Falls p nicht existiert, dann gibt es keine solche Kante.

Situation:



Die Kante $([i,j],[i,j+1])$ ist genau dann $\langle j,j+1\rangle$-wesentlich, wenn $s<t$. Dann muss diese noch auf den Keller abgelegt werden.

Insgesamt wird somit $S(j+1)$ durch folgenden Algorithmus aus $S(j)$ berechnet:

<u>Algorithmus</u> $S(j) \leadsto S(j+1)$

<u>Eingabe</u>: $S(j)$
<u>Ausgabe</u>: $S(j+1)$
<u>Methode</u>:

(1) <u>if</u> Kopf$(SEC(S(j))) = [i,j+1]$
  <u>then</u>
   POP
  <u>else</u>
   Ersetze Kopf$(TOP(S(j)))$ durch $[i,j+1]$
 <u>fi</u>; (* Sei t die Höhe des resultierenden Kellers *)

(2) Berechne größtes $s \le t$ mit

$$v(i,k_s) + c([i,k_s],[i,p_s]) \le v(i,j) + c([i,j],[i,p_s]);$$

 <u>if</u> $s<t$
  <u>then</u>
   Entferne alle $([i,k_q],[i,p_q])$ mit $q>s$
 <u>fi</u>;

(3.) **if** $s < t$

then

Berechne $p \in \{p_{s+1}+1, \dots, p_s - 1\}$ mit

$$v(i, k_{s+1}) + c([i, k_{s+1}], [i, p-1])$$

$$> v(i,j) + c([i,j], [i, p-1])$$

und

$$v(i, k_{s+1}) + c([i, k_{s+1}], [i, p])$$

$$\leq v(i,j) + c([i,j], [i,p])$$

fi;

if p definiert

then

PUSH$(([i, k_{s+1}], [i, p]))$

fi;

(4) **if** $s < t$

then

PUSH$(([i,j], [i, j+1]))$

fi.

Zeitanalyse:

Bezeichne GZ die für die Konstruktion von RMIN(i) benötigte Gesamtzeit und KZ diejenige Zeit, die zur Manipulation des Kellers benötigt wird.

Dann gilt: $GZ = O(n) + KZ$.

<u>Ziel:</u> Abschätzung von KZ.

<u>Beobachtung:</u>
Nach jeder Betrachtung eines Knotens der Matrix werden maximal zwei Kanten auf den Keller abgelegt.

$\Rightarrow$

Die für PUSH- und POP-Operationen benötigte Gesamtzeit ist $O(n)$.

Falls wir im Schritt (3) binäre Suche verwenden, dann genügt für eine Ausführung von Schritt (3) $O(\log n)$ Zeit.

$\Rightarrow$

$KZ = O(n \cdot \log n)$.

Bezüglich den Spalten und den Diagonalen des Distanzgraphen verfahren wir analog.

<u>Übung:</u>
Entwickeln Sie einen Algorithmus, der bei konkaven Kostenfunktionen für $0 \le i \le m$, $0 \le j \le n$ alle $v(i,j)$ in $O(m \cdot n \log(m+n))$ Zeit berechnet.

Um eine kompakte Repräsentation des Distanzgraphen in $O(m \cdot n \cdot \log(m+n))$ Zeit zu berechnen muss noch einiges an Arbeit investiert werden.
m

N. Blum, Speeding up dynamic programming...

ist folgender Satz bewiesen:

Satz 3.3

Seien $x := x_1 x_2 \dots x_n$ und $y = y_1 y_2 \dots y_m$ zwei Sequenzen. Dann kann bei konkaven Kostenfunktionen eine kompakte Repräsentation des Distanzgraphen $D_{M_3}(x,y)$ in $O(m \cdot n \cdot \log(m+n))$ Zeit und $O(m \cdot n)$ Platz berechnet werden.

3.3 Algorithmen zur Mustererkennung in Sequenzen

Seien $x = x_1 x_2 \cdots x_n$ und $y = y_1 y_2 \cdots y_m$ zwei Strings über ein endliches Alphabet Σ. Ein Alignment zwischen y und einem Teilstring von x heißt lokales Alignment. Ein Alignment zwischen einem Teilstring von y und einem Teilstring von x heißt Subalignment.

Folgende Probleme werden wir betrachten:

Problem 1:

Gegeben zwei Sequenzen $x = x_1 x_2 \dots x_n$, $y = y_1 y_2 \cdots y_m \in \Sigma^+$, $n \gg m$, eine Operationenmenge M und eine Kostenfunktion c_M sollen alle Teilsequenzen $\tilde{x}$ von x mit lokal minimaler Distanz von y und für jedes $\tilde{x}$ alle korrespondierende lokal optimale lokale Alignments berechnet werden.

Problem 2:

Gegeben zwei Sequenzen $x := x_1 x_2 \ldots x_n$, $y := y_1 y_2 \ldots y_m$ $\in \Sigma^+$, eine Operationenmenge M und eine Kosten=funktion c_M sollen alle Paare $(\check{x}, \check{y})$ von Teil=sequenzen von x und von y mit lokal maximaler Ähnlichkeit und für jedes dieser Paare alle korres=pondierenden lokal optimale Subalignments be=rechnet werden.

Wir betrachten in der Vorlesung nur die Operationen=menge M_1. Zunächst müssen wir den Lokalitäts=begriff formal definieren. Die erste solche Definition wurde 1980 von Sellers gegeben:

Eine Teilsequenz $\check{x}$ von x hat <u>schwach lokal minimale Distanz</u> von y, falls für alle Teil=sequenzen x' von x mit $x' \subset \check{x}$ oder $\check{x} \subset x'$ gilt:

$$c_{M_1}(\check{x}, y) \leq c_{M_1}(x', y).$$

Da wir nur die Operationenmenge M_1 zugrunde legen, lassen wir ab sofort den Index M_1 weg.

Ein lokales Alignment $A(\check{x}, y)$ mit $c(A(\check{x}, y)) = c(\check{x}, y)$ heißt <u>schwach lokal optimal</u>.

<u>Kritik</u>:

Folgende Situation kann eintreten:

Eine Teilsequenz $\check{x}$ von x hat schwach lokal minimale Distanz von y, obwohl ein Teilstring

x' von x existiert mit:

a) $\tilde{x} \cap x' \neq \emptyset$. D.h., $\tilde{x}$ und x' überlappen.
b) $c(x', y) < c(\tilde{x}, y)$.

Dies bedeutet insbesondere, dass in einer sinnvollen lokalen Region von x die Teilsequenz $\tilde{x}$ nicht lokal minimale Distanz von y hat, obwohl $\tilde{x}$ schwach lokal minimale Distanz von y hat.

<u>Lösung</u>: Verbot der oben beschriebenen Situation.

Dies führt zur folgenden Definition:

Eine Teilsequenz $\tilde{x}$ von x hat <u>lokal minimale Distanz</u> von y, falls für alle Teilsequenzen x' von x mit $\tilde{x} \cap x' \neq \emptyset$ gilt:

$$c(\tilde{x}, y) \leq c(x', y).$$

Ein lokales Alignment $A(\tilde{x}, y)$ mit $c(A(\tilde{x}, y)) = c(\tilde{x}, y)$ heißt <u>lokal optimal</u>.

Bezüglich Subalignments ist folgende analoge Definition sinnvoll:

Ein Paar $(\tilde{x}, \tilde{y})$ von Teilsequenzen von x und y hat <u>lokal maximale Ähnlichkeit</u>, falls für alle Paare (x', y') von Teilsequenzen von x und y mit $x' \cap \tilde{x} \neq \emptyset$ oder $y' \cap \tilde{y} \neq \emptyset$ gilt:

$$c(\tilde{x}, \tilde{y}) \leq c(x', y').$$

Ein Subalignment $A(\tilde{x},\tilde{y})$ mit $c(A(\tilde{x},\tilde{y})) = c(\tilde{x},\tilde{y})$ heißt dann <u>lokal optimal</u>.

Genauso wie früher definieren wir den Editiergraphen $\mathcal{E}_{M_1}(x,y) = (V,E)$. Analog zu früher lässt sich auch zur Lösung von Problem 1 sowie auch zur Lösung von Problem 2 ein korrespondierender Distanzgraph $D'_{M_1}(x,y)$ bzw. $D''_{M_1}(x,y)$ definieren und auch berechnen.

<u>Übung</u>:
Definieren Sie $D'_{M_1}(x,y)$ und $D''_{M_1}(x,y)$ und modifizieren Sie den Algorithmus EDDISM$_1$ derart, dass $D'_{M_1}(x,y)$ bzw. $D''_{M_1}(x,y)$ berechnet wird. Was ist die Zeitkomplexität Ihres Algorithmus?

Zunächst werden wir uns mit Problem 1 beschäftigen. Hierzu sind folgende Definitionen nützlich:

Ein Pfad P in $D'_{M_1}(x,y) = (V, E_d)$ heißt <u>(schwach) lokal optimal</u>, falls P zu einem (schwach) lokal optimalen Alignment korrespondiert.

Seien $\mathcal{E}_{se}(x,y) := (V_{se}, E_{se})$ und $\mathcal{E}_e(x,y) := (V_e, E_e)$ definiert durch:

$$E_{se} := \{ e \in E_d \mid e \text{ liegt auf einem schwach lokal optimalen Pfad in } D'_{M_1}(x,y) \}$$

$V_{se} := \{ v \in V \mid \exists w \in V : (v,w) \in E_{se} \text{ oder } (w,v) \in E_{se} \}$.

und

$$E_e := \{ e \in E_d \mid e \text{ liegt auf einem lokal optimalen Pfad } P \text{ in } D'_{M_1}(x,y) \}$$

$$V_e := \{ v \in V \mid \exists w \in V : (v,w) \in E_e \text{ oder } (w,v) \in E_e \}.$$

Bezeichne $x(P)$ denjenigen Teilstring von x, der zum Pfad P korrespondiert. $y(P)$ ist analog definiert. Sei

$$e(x^j, y^i) := \begin{cases} 0 & \text{falls } i = 0 \\ \min\{ c(\tilde{x}, y^i) \mid \tilde{x} \text{ ist Suffix von } x^j \} & \text{falls } i > 0 \end{cases}$$

Folgendes Lemma ist bei der Lösung von Problem 1 hilfreich:

<u>Lemma 3.1</u>

Sei $P := P_1, [i,j], P_2$ ein (schwach) lokal optimaler Pfad. Dann gilt

$$c(P_1, [i,j]) = e(x^j, y^i).$$

<u>Beweis:</u>

Definition $\Rightarrow$

$$c(P_1, [i,j]) \geq e(x^j, y^i).$$

Annahme: $c(P_1, [i,j]) > e(x^j, y^i)$.

Dann existiert ein Pfad P', der in Zeile 0 beginnt, mit

$$c(P', [i,j]) < c(P_1, [i,j]).$$

Betrachten wir

$$\tilde{P} := P', [i,j], P_2.$$

Es gilt: $c(\tilde{P}) < c(P)$.

Sei $[m,s]$ der letzte Knoten auf P und somit auch auf $\tilde{P}$.

$\Rightarrow$

$\tilde{P}$ korrespondiert zu einem Alignment zwischen einem Suffix von $x_1 x_2 \ldots x_s$ und y.

Wegen $c(\tilde{P}) < c(P)$ folgt somit:

P ist nicht (schwach) lokal optimal.

Dies ist ein Widerspruch zur Voraussetzung des Lemmas.

$\Rightarrow$

Annahme ist falsch, d.h., $c(P_1, [i,j]) = e(x^j, y^i)$.

■

Lemma 3.1 impliziert direkt folgendes Lemma:

Lemma 3.2

$$\mathcal{E}_{se}(x,y) \subseteq \mathcal{D}'_{M_1}(x,y).$$

Aus den Definitionen von schwach lokal optimal und lokal optimal folgt direkt

$$\mathcal{E}_e(x,y) \subseteq \mathcal{E}_{se}(x,y).$$

Bezeichne P_{lk} einen Pfad vom Knoten $[0,l]$ zum Knoten $[m,k]$ in $D'_{M_1}(x,y)$. In den nächsten drei Lemmata beweisen wir einige Eigenschaften von $\mathcal{E}_{se}(x,y)$ und von $\mathcal{E}_e(x,y)$.

Lemma 3.3

Seien $P_{l_1k_1}$ und $P_{l_2k_2}$ zwei Pfade in $D'_{M_1}(x,y)$ mit $l_1 \le l_2$ und $k_2 \le k_1$. Falls $P_{l_ik_i}$, $i \in \{1,2\}$ schwach lokal optimal ist, dann gilt:

$$e(x^{k_i}, y^m) = \min\{ e(x^{k_1}, y^m), e(x^{k_2}, y^m) \}.$$

Beweis:

Wegen $l_1 \le l_2$ und $k_2 \le k_1$ gilt:

$$x(P_{l_2k_2}) \subseteq x(P_{l_1k_1}).$$

Also folgt die Behauptung direkt aus der Definition von schwach lokal optimal. ∎

Sei

$$e_{l,k} := \min_{1 \le j \le n} \{ e(x^j, y^m) \mid \exists \text{ Pfad von } [l,k] \text{ nach } [m,j] \text{ in } D'_{M_1}(x,y) \}$$

Dann gilt

$$e_{\ell,k} = \min \{ e_{\ell',k'} \mid ([\ell,k],[\ell',k']) \in E_d \}$$

Aus Lemma 3.3 folgt direkt folgendes Lemma:

<u>Lemma 3.4</u>

Sei $([r,k],[r',k']) \in E_d$. Falls $e_{r,k} < e_{r',k'}$, dann gilt $([r,k],[r',k']) \notin E_{se}$.

Aus Lemma 3.3 und Lemma 3.4 folgt folgendes Lemma:

<u>Lemma 3.5</u>

Sei CC eine Zusammenhangskomponente von $\mathcal{E}_{se}(x,y)$ oder $\mathcal{E}_e(x,y)$. Dann haben alle Pfade P_{ek} in CC dieselben Kosten.

Folgender Satz besagt, dass es zur Lösung von Problem 1 genügt $\mathcal{E}_{se}(x,y)$ bzw. $\mathcal{E}_e(x,y)$ zu berechnen.

<u>Satz 3.4</u>

Es existiert eine Bijektion zwischen den Pfaden von Zeile 0 zur Zeile m in

a) $\mathcal{E}_{se}(x,y)$ und den schwach lokal optimalen Pfaden.

b) $\mathcal{E}_e(x,y)$ und den lokal optimalen Pfaden.

Beweis:

Konstruktion $\Rightarrow$

a) $\mathcal{E}_{se}(x,y)$ enthält alle schwach lokal optimale Pfade.

b) $\mathcal{E}_{e}(x,y)$ enthält alle lokal optimale Pfade.

Also genügt es zu zeigen, dass der Graph jeweils keine weitere Pfade von Zeile 0 zur Zeile m enthält.

Sei $P_{rk} := e_1, e_2, \dots, e_s$ ein beliebiges Pfad in $\mathcal{E}_{se}(x,y)$ bzw. $\mathcal{E}_{e}(x,y)$. Sei

$$e_s := ([m-1, e], [m, k]).$$

Lemmata 3.1 und 3.2 $\Rightarrow$

$$c(P_{rk}) = e(x^k, y^m).$$

Falls P_{rk} ein Pfad in $\mathcal{E}_{se}(x,y)$ ist, dann folgt die Behauptung direkt aus Lemma 3.5, der Definition von $\mathcal{E}_{se}(x,y)$ und der Tatsache, dass e_s auf P_{rk} liegt.

__Annahme__:

P_{rk} ist ein Pfad in $\mathcal{E}_e(x,y)$, der nicht lokal optimal ist.

Dann existiert ein Pfad $P_{r'k'}$ in $D'_{\sigma_1}(x,y)$, so dass

i) $c(P_{r'k'}) < c(P_{rk})$ und

ii) $x(P_{rk})$ und $x(P_{r'k'})$ überlappen. D.h., $r' \leq r \leq k'$ oder $r \leq r' \leq k$.

Da $e_s \in E_\ell$, liegt e_s auf mindestens einem lokal optimalen Pfad.

Somit folgt aus i) und der Definition der lokalen Optimalität:

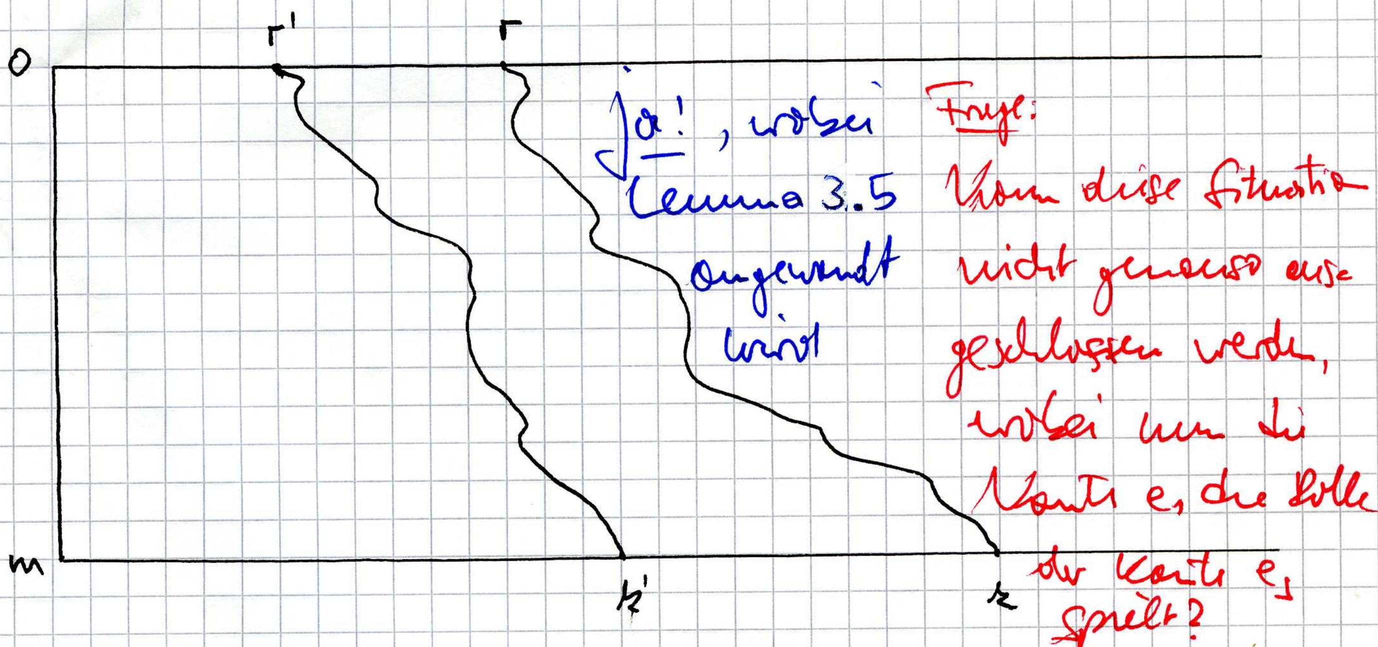
- Das linke Ende von $x(P_{r\ell})$ überlappt das rechte Ende von $x(P_{r'\ell'})$.

Ansonsten wäre $e_s \notin E_\ell$.

$\Rightarrow$

$$r' \leq r \leq \ell'.$$

Somit liegt folgende Situation vor:



Definition der lokalen Optimalität impliziert somit:

Für jeden lokal optimalen Pfad $P_{r\ell''}$, der e_1 enthält, gilt: $\quad c(P_{r\ell''}) \leq c(P_{r'\ell'})$.

Sei e_t die erste Kante auf $P_{r\ell}$, die auf einem lokal optimalen Pfad

$P_{\bar{r}\bar{z}} := Q_1, e_t, Q_2$ mit $c(P_{\bar{r}\bar{z}}) > c(P_{r'z'})$

liegt. Da e_s diese Eigenschaft hat, existiert e_t.

$\Rightarrow$

$\exists$ Pfad $P_{rz''} = e_1, e_2, \ldots, e_{t-1}, R$ mit

$$c(P_{rz''}) \leq c(P_{r'z'}).$$

Betrachten wir den Pfad $T := Q_1, R$. Dann gilt:

i) T ist ein Pfad von $[0, \bar{r}]$ nach $[m, z'']$ in $D'_{M_1}(x,y)$ und $c(T) = c(P_{rz''})$.

ii) $x(T)$ ist ein Teilstring von $x(P_{\bar{r}\bar{z}})$ oder umgekehrt.

Wegen

$$c(P_{\bar{r}\bar{z}}) > c(P_{r'z'}) \geq c(P_{rz''}) = c(T)$$

ist dies ein Widerspruch zur lokalen Optimalität von $P_{\bar{r}\bar{z}}$.

Also war unsere Annahme falsch. D.h., jeder Pfad von Zeile 0 zur Zeile m in $\mathcal{E}_e(x,y)$ ist lokal optimal.

∎

<u>Ziel</u>:

Berechnung von $\mathcal{E}_{se}(x,y)$ und dann von $\mathcal{E}_e(x,y)$.

Berechnung von $\mathcal{E}_{se}(x,y)$:

Idee:

(1) (Anwendung von Lemma 3.4)
Führe eine Rückwärts-topologische Suche auf $D'_{M_1}(x,y)$ durch und entferne dabei alle Kanten $([r,k],[r',k'])$ mit $e_{r,k} < e_{r',k'}$.

(2) Streiche alle Kanten $([i,j],[i',j'])$ mit $i>0$ und Eingangsgrad$([i,j]) = 0$, bis keine solche Kante mehr existiert.

Im folgenden Algorithmus verwenden wir die Variable num$([i,j])$, um die Anzahl der nicht betrachteten direkten Nachfolger des Knotens $[i,j]$ zu speichern. Zu Beginn ist die Anzahl der direkten Nachfolger des Knotens $[i,j]$ der Wert der Variablen num$([i,j])$. Nach Betrachten des Knotens $[i,j]$ setzen wir die Variable num$([i,j])$ auf den Wert 4. Damit soll ausgedrückt werden, dass der Knoten $[i,j]$ bereits betrachtet ist.

Algorithmus BERECHNUNG $\mathcal{E}_{se}(x,y)$

Eingabe: Distanzgraph $D'_{M_1}(x,y) = (V, E_d)$ nebst Werte $e(x^j, y^i)$ $\forall [i,j] \in V$.

Ausgabe: $\mathcal{E}_{se}(x,y) = (V_{se}, E_{se})$.

Methode:

```
(1)  K := ∅;
(2)  for j from 1 to n
        do
           for i from 0 to m-1
              do
                 e_{i,j} := ∞ ;
                 num([i,j]) := outdeg_{E_d}([i,j])
              od;
           e_{m,j} := e(x^j, y^m) ;
           num([m,j]) := 0
        od;
(3)  while ∃ [r,k] mit num([r,k]) = 0
        do
           Wähle solch einen Knoten [r,k] ;
           num([r,k]) := 4 ;
           for alle ([r,k], [r̄,k̄]) ∈ E_d
              do
                 if e_{r,k} = e_{r̄,k̄}
                   then
                      K := K ∪ {([r,k], [r̄,k̄])}
                 fi
              od;
           for alle ([r',k'], [r,k]) ∈ E_d
              do
                 e_{r',k'} := min{e_{r',k'}, e_{r,k}};
                 num([r',k']) := num([r',k']) - 1
              od
        od;
```

(4) <u>while</u> $\exists\, ([i,j],[i',j']) \in K$ mit $i > 0$ und $\text{indeg}_K([i,j]) = 0$

<u>do</u>

$K := K \setminus \{([i,j],[i',j'])\}$

<u>od</u>;

(5) $E_{se} := K$;

(6) $V_{se} := \{ v \in V \mid \exists\, w \in V : (v,w) \in E_{se} \text{ oder } (w,v) \in E_{se} \}$.

<u>Übung:</u>

Beweisen Sie die Korrektheit des Algorithmus BERECHNUNG $\mathcal{E}_{se}(x,y)$ und entwickeln Sie Datenstrukturen für die Implementierung. Was ist die von dem Algorithmus benötigte Zeit?

Insgesamt haben wir folgenden Satz bewiesen:

<u>Satz 3.5</u>

Sei M_1 die Operationenmenge und sei $D'_{M_1}(x,y)$ gegeben. Dann kann $\mathcal{E}_{se}(x,y) = (V_{se}, E_{se})$ in $O(m \cdot n)$ Zeit berechnet werden, wobei $|x| = n$ und $|y| = m$.

<u>Berechnung von $\mathcal{E}_e(x,y)$:</u>

<u>Idee:</u>

Ausgehend vom Graphen $\mathcal{E}_{se}(x,y)$ erhalten wir $\mathcal{E}_e(x,y)$, indem wir aus V_{se} alle Knoten in $V_{se} \setminus V_e$ und dann aus E_{se} alle Kanten in $E_{se} \setminus E_e$ entfernen.

Für die Durchführung benötigen wir einige Bezeichnungen.

Sei CC eine Zusammenhangskomponente in $\mathcal{E}_{se}(x,y)$. Dann bezeichnet

- $c(CC)$ die eindeutig bestimmten Kosten $c(P_{rk})$ für $P_{rk} \in CC$.

- $Left(CC)$ die erste Position von CC bzgl. x. D.h.,
$$Left(CC) := \min_{1 \le k \le n} \{ k \mid [0, k-1] \in CC \}.$$

- $Right(CC)$ die letzte Position von CC bzgl. x. D.h.,
$$Right(CC) := \max_{1 \le k \le n} \{ k \mid [m, k] \in CC \}.$$

Als nächstes werden wir mittels einigen Lemmata exakt diejenigen Knoten charakterisieren, die aus V_{se} zu streichen sind.

<u>Lemma 3.6</u>

Sei CC eine Zusammenhangskomponente in $\mathcal{E}_{se}(x,y)$. Sei $[r,k] \in CC$. Falls es eine Zusammenhangskomponente CC' in $\mathcal{E}_{se}(x,y)$ mit $c(CC') < c(CC)$, $Left(CC') \le k$ und $Right(CC') \ge k$ gibt, dann ist $[r,k] \notin V_e$.

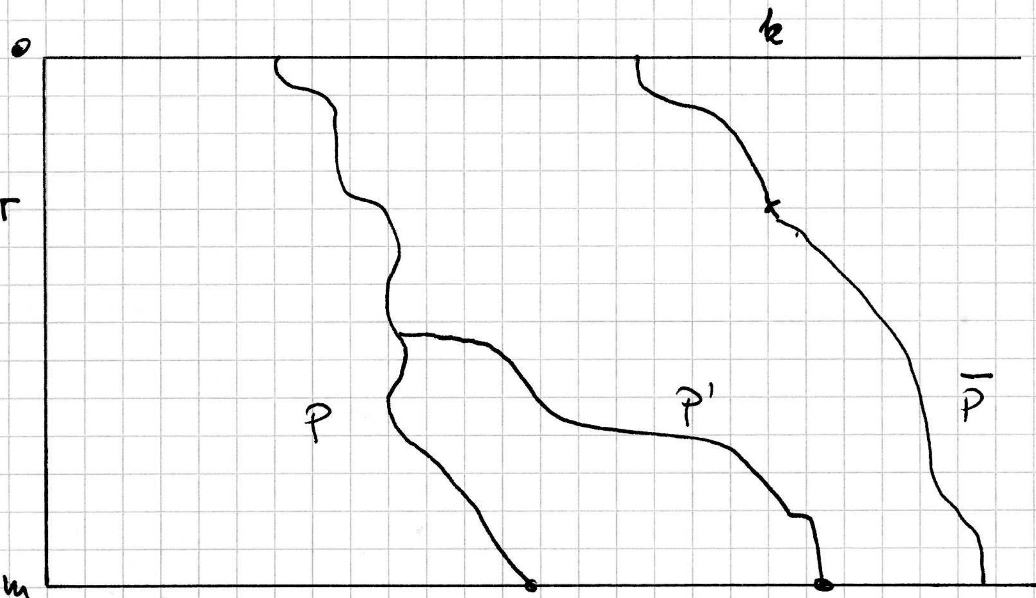
<u>Beweis</u>:

Die Behauptung folgt direkt aus der Definition eines lokal optimalen Pfades und der Struktur von $\mathcal{E}_{se}(x,y)$.

∎

Lemma 3.6 charakterisiert Knoten $[r,k] \in CC$, die nicht in V_e sein können, da x_k von einem Teilstring von x überlappt wird, der zu y eine kleinere Distanz als $c(CC)$ hat.

Folgendes Beispiel beschreibt eine weitere Situation, in der ein Knoten $[r,k] \in CC$ nicht in V_e sein kann, da x_k von einem Teilstring von x überlappt wird, der zu y eine kleinere Distanz als $c(CC)$ hat.



$$c(P) < c(P') < c(\overline{P})$$

Der zu P' korrespondierende Teilstring von x wird von $c(P)$ ausgeschlossen, ist also nicht in der Zusammenhangskomponente, die P enthält.

<u>Aber:</u> $x(P')$ eliminiert $[r,k]$ aus V_e!

Wir sagen, ein Knoten $[r,k] \in CC$ wird <u>direkt ausgeschlossen</u>, falls x_k von einem Teilstring

von x überlappt wird, der zu y eine kleinere Distanz als $c(CC)$ hat.

Des Weiteren müssen wir alle Knoten $[r,k]$ ausschließen, die die Eigenschaft besitzen, dass für alle Pfade P in $\mathcal{E}_{se}(x,y)$ mit $[r,k] \in P$ gilt: $x(P)$ wird von einem Teilstring von x überlappt, welcher kleinere Distanz als $c(CC)$ von y hat. Daher definieren wir induktiv, dass $[i,j] \in V_{se}$ <u>ausgeschlossen</u> wird, falls einer der folgenden drei Fälle erfüllt ist:

1. $[i,j]$ wird direkt ausgeschlossen.
2. Alle direkte Vorgänger von $[i,j]$ in $\mathcal{E}_{se}(x,y)$ werden ausgeschlossen.
3. Alle direkte Nachfolger von $[i,j]$ in $\mathcal{E}_{se}(x,y)$ werden ausgeschlossen.

Folgendes Lemma charakterisiert exakt die Knoten in $V_{se} \setminus V_e$.

<u>Lemma 3.7</u>

Sei $[i,j] \in V_{se}$. Dann gilt genau dann $[i,j] \notin V_e$, wenn $[i,j]$ ausgeschlossen wird.

Beweis:

Es ist klar, dass $[i,j] \notin V_e$, falls $[i,j]$ ausgeschlossen wird.

Zum Beweis der anderen Richtung betrachten wir

$$[i,j] \in V_{se} \setminus V_e.$$

Sei CC diejenige Zusammenhangskomponente in $E_{se}(x,y)$, die den Knoten $[i,j]$ enthält.

$[i,j] \notin V_e \Rightarrow$

Alle Teilstrings x' von x mit $x' = x(P)$ für einen Pfad $P = P_1, [i,j], P_2 \in CC$ von Zeile 0 nach Zeile m werden von einem Teilstring von x überlappt, dessen Distanz zu y kleiner als $c(CC)$ ist.

$\Rightarrow$

Auf all diesen Pfaden P gibt es einen Knoten, der direkt ausgeschlossen wird.

Annahme:

Es existieren Pfade $Q = Q_1, [i,j], Q_2$ und $R = R_1, [i,j], R_2$ in CC, so dass kein Knoten auf $Q_1, [i,j]$ und kein Knoten auf $[i,j], R_2$ direkt ausgeschlossen wird.

$\Rightarrow$

$x(Q_1, [i,j], R_2)$ wird nicht von einem Teilstring von x überlappt, dessen Distanz zu y

kleiner als $c(CC)$ ist.

$\Longrightarrow$

$[i,j] \in V_e$, Widerspruch.

Also ist mindestens einer der folgenden zwei Fälle erfüllt:

i) Auf allen Pfaden $Q_1, [i,j]$ gibt es einen Knoten, der direkt ausgeschlossen wird.

ii) Auf allen Pfaden $[i,j], R_2$ gibt es einen Knoten, der direkt ausgeschlossen wird.

In beiden Fällen kann leicht mittels Induktion bewiesen werden, dass $[i,j]$ ausgeschlossen wird.

Übung:
Durchführung des Induktionsbeweises.

■

Mit Hilfe des Lemmas 3.7 ist es nun einfach, $\mathcal{E}_e(x,y) = (V_e, E_e)$ zu berechnen. Wir erhalten folgenden Algorithmus:

<u>Algorithmus</u> BERECHNUNG $\mathcal{E}_e(x,y)$

<u>Eingabe:</u> $x = x_1x_2\cdots x_n$, $y = y_1y_2\cdots y_m \in \Sigma^+$,
Kostenfunktion c_{M_1}, $D'_{M_1}(x,y)$.

<u>Ausgabe:</u> $\mathcal{E}_e(x,y) = (V_e, E_e)$.

Methode:

(1) Berechne gleichzeitig $\mathcal{E}_{se}(x,y) = (V_{se}, E_{se})$, die Zusammenhangskomponenten $CC_1, CC_2, \ldots$ und für jede Zusammenhangskomponente CC_p die benötigte Information $c(CC_p)$, $Left(CC_p)$ und $Right(CC_p)$.

(2) Berechne alle Knoten, die direkt ausge= schlossen werden. Sei K_0 die Menge dieser Knoten.

(3) Mittels einer topologischen Suche auf $\mathcal{E}_{se}(x,y)$, wobei wir in den Knoten, die direkt "von links" ausgeschlossen werden, starten, berech= ne alle Knoten $[i,j]$, die ausgeschlossen werden, da alle direkte Vorgänger von $[i,j]$ ausgeschlossen werden. Sei K_1 die Menge dieser Knoten.

(4) Mittels einer Rückwärts-topologischen Suche auf $\mathcal{E}_{se}(x,y)$, wobei wir in den Knoten, die direkt "von rechts" ausgeschlossen werden, starten, berechne alle Knoten $[i,j]$, die ausge= schlossen werden, da alle direkte Nachfolger von $[i,j]$ ausgeschlossen werden. Sei K_2 die Menge dieser Knoten.

(5) $V_e := V_{se} \setminus (K_0 \cup K_1 \cup K_2)$.

(6) $E_e := E_{se} \cap (V_e \times V_e)$.

Übung:

Arbeiten Sie den Algorithmus BERECHNUNG $E_e(x,y)$ aus und analysieren Sie die benötigte Zeit.

Insgesamt erhalten wir somit folgenden Satz:

Satz 3.6

Sei M_1 die Operationenmenge und sei $D'_{M_1}(x,y)$ gegeben. Dann kann $E_e(x,y) = (V_e, E_e)$ in $O(m \cdot n)$ Zeit, wobei $|x| = n$ und $|y| = m$, berechnet werden.

Übung:

Modifizieren Sie die bzgl. der Operationenmenge M_1 entwickelte Lösung von Problem 1, so dass Problem 2 gelöst wird.

(Hinweis: Definieren Sie analog zu Left(CC) und Right(CC) Upper(CC) und Lower(CC) und modifizieren Sie Lemma 3.7 entsprechend).

Bemerkung:

Eine Lösung von Problem 1 und von Problem 2 bzgl. der Operationenmenge M_3 kann in

N. Blum, Speeding up dynamic programming...

gefunden werden.