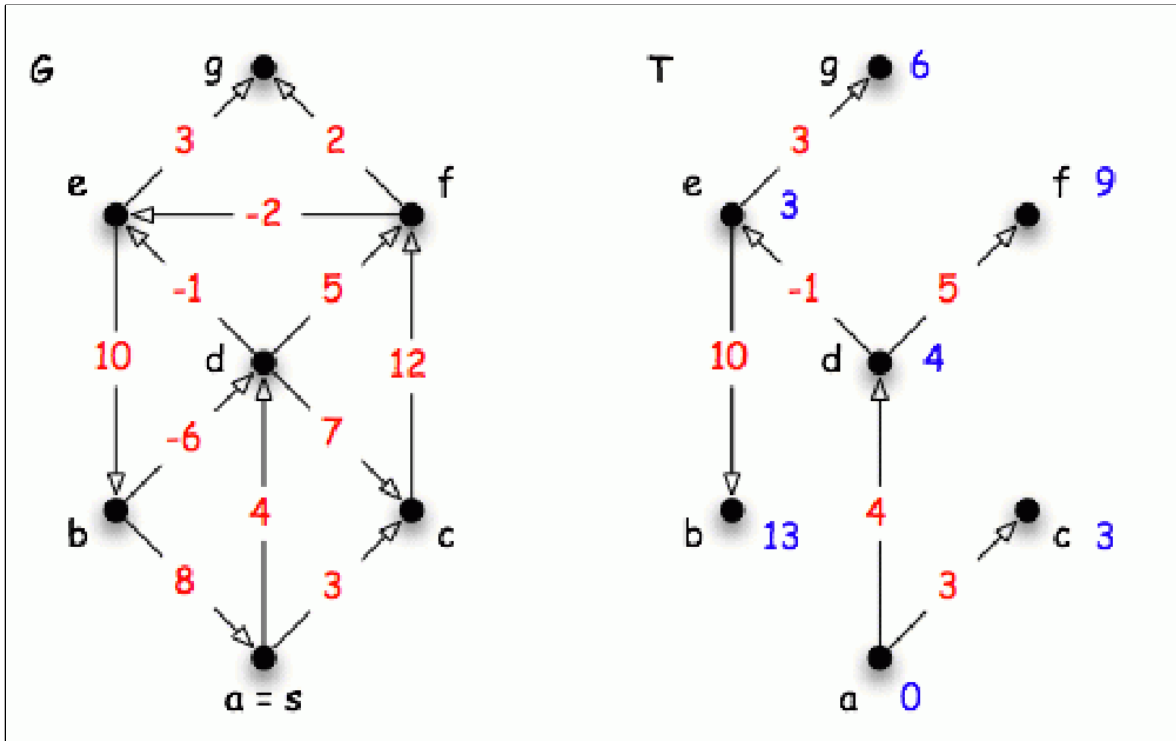


↳ G.W. Algorithmen

↳ Sequence Alignment



Bellman-Ford-Algorithmus

```
int n = a.length;
u = new double[n][n];
tree = new int[n][n];
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
    {
        u[i][j] = a[i][j];
        if (a[i][j] != Double.POSITIVE_INFINITY)
            tree[i][j] = i;
        else
            tree[i][j] = -1;
    }
for (int m = 2; m <= n; ++m)
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < n; ++k)
                if (u[i][j] > u[i][k] + a[k][j])
                {
                    u[i][j] = u[i][k] + a[k][j];
                    tree[i][j] = k;
                }
```

- $O(n^3)$
(kann man noch beschleunigen)
- Verbesserung durch "Active-Set"-Strategie
(*"Shortest Path II"*: VL)

• notwendig: konservative Kantengewichte
(keine negative Zyklen)

⇒ negative Kantengewichte erlaubt!

Dijkstra-Algorithmus

• verwaltet Distanz-Labels d an den Knoten

• verwaltet Priority-Queue mit "aktiven Knoten"

$$d(s) := 0;$$

$$d(v) := \infty \quad \forall v \neq s;$$

$$A := V$$

while ($A \neq \emptyset$) {

$v \in A$ mit $d(v)$ minimal

$\forall (v, w) \in E$ {

 if ($d(v) + c(v, w) < d(w)$)

$d(w) := d(v) + c(v, w)$

$\text{pred}(w) := v;$

 A.decreasekey($w, d(w)$);

 }

}

- Laufzeit $O(m + n \log n)$
(hängt stark von Implementation ab!)
 - Voraussetzung: Kantengewichte nicht negativ!
-

Idee decreaseKey:

- Zugriff auf beliebiges Element in der Queue!

↳ standardmäßig nicht unterstützt!

insert

```
public PriorityQueue.Position<AnyType> insert(AnyType x)
```

Insert into the priority queue, and return a Position that can be used by decreaseKey. Duplicates are allowed.

Parameters:

x - the item to insert.

Returns:

the node containing the newly inserted item.

- insert hat Rückgabewert

PriorityQueue.Position

⇒ beim Einfügen von v zurückgegebener Wert ermöglicht Zugriff auf v .

decreaseKey

```
public void decreaseKey(PriorityQueue.Position<AnyType> pos,  
    AnyType newVal)
```

Change the value of the item stored in the pairing heap.

Parameters:

`pos` - any Position returned by insert.

`newVal` - the new value, which must be smaller than the currently stored value.

Throws:

`java.lang.IllegalArgumentException` - if `pos` is null or if new value is larger than old.

- hat Parameter PriorityQueue.Position und neuer (niedrigere) Wert.

- Vorgehen:
- Speicher beim Einfügen von Knoten ihre Position
 - Verwende Position beim Aufruf von decreaseKey.

Sequence Alignment

The screenshot shows a Google search interface. At the top, there are navigation links for 'Web', 'Bilder', 'Groups', 'News', 'Products', and 'Mehr'. The search bar contains the text 'Felix Konegt' and a 'Suche' button. Below the search bar, there are options for 'Suche: Das Web', 'Seiten auf Deutsch', and 'Seiten aus Deutschland'. The search results section is titled 'Web' and shows a suggestion: 'Meinten Sie: Felix Kunert'. Below this, there are two messages: 'Es wurden keine Standard-Webseiten gefunden, in denen alle eingegebenen Suchbegriffe vorkommen.' and 'Es wurden keine mit Ihrer Suchanfrage - Felix Konegt - übereinstimmenden Dokumente gefunden.' At the bottom, there is a 'Vorschläge:' section with four bullet points: 'Vergewissern Sie sich, dass alle Wörter richtig geschrieben sind.', 'Probieren Sie andere Suchbegriffe.', 'Probieren Sie allgemeinere Suchbegriffe.', and 'Probieren Sie weniger Suchbegriffe.'

Web

Meinten Sie: [Felix König](#)

Es wurden keine Standard-Webseiten gefunden, in denen alle eingegebenen Suchbegriffe vorkommen.

Es wurden keine mit Ihrer Suchanfrage - **Felix Känek** - übereinstimmenden Dokumente gefunden.

Vorschläge:

- Vergewissern Sie sich, dass alle Wörter richtig geschrieben sind.
- Probieren Sie andere Suchbegriffe.
- Probieren Sie allgemeinere Suchbegriffe.
- Probieren Sie weniger Suchbegriffe.

Wann sind zwei Wörter ähnlich?

Bsp.:

occurrence

ocurrance

o c c u r r e n c e
o c u r r a n c e

=> 1 "Lücke"
1 "Fehler"

o c c u r r e n c e
o c u r r a n c e

=> 3 Lücken, 0 Fehler

• Viele Möglichkeiten, Wörter zu vergleichen

→ Sind 3 Lücken beim als 1
Lücke und 1 Fehler?

andere Bsp: a b b b a a a b a b b a b ⇒ ?
a b a b a a b b a a b a b

Anwendung: Molekularbiologie

DNA ↔ lang Wörter über Alphabet

{A, C, G, T}

geg: DNA-Strings zweier Bakterien X, Y

Frage: Sind X, Y ähnlich?

Definition: Seien $X = x_1 x_2 \dots x_m$ und
 $Y = y_1 y_2 \dots y_n$ zwei Strings
über dem gleichen Alphabet.

Ein Alignment von X und Y ist
ein Matching M zwischen X und Y mit

$(i, j), (i', j') \in M, i < i'$

$\Rightarrow j < j'$

(keine Paare über Kreuz)

Bsp.:

stop
| / |
top s

Alignment



~~stop
x / x
top s
kein~~

$$M = \{(1,1), (3,2), (4,3)\}$$

Seien Parameter

$\delta :=$ "Strafe für Lücke"

$\alpha_{pq} :=$ "Strafe falls Buchstabe p mit Buchstabe q nicht matcht"

und seien zu einem Alignment M seine Kosten $c(M)$ die Summe dieser Strafen.

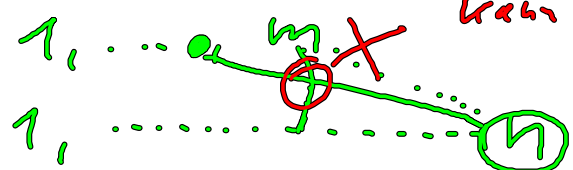
Definition: Der Abstand zwischen zwei Wörtern sind die Kosten eines kostenminimalen Alignments.

⇒ Frage: Wie finde wir ein kostenminimales Alignment?

... durch Dynamische Programmierung!
(Ausnutzung optimaler Substruktur)

Beobachtungen: Sei M^* ein optimales Alignment

- entweder $(m, n) \in M^*$ oder $(m, n) \notin M^*$
- falls $(m, n) \notin M^*$, so bleibt m oder n ungematcht.

Bsp.: 

Proposition: In einem optimalen Alignment gilt wenigstens eins der folgenden:

- $(m, n) \in M^*$
- m nicht gematcht
- n nicht gematcht

Seien $OPT(i, j)$ die Kosten eines optimalen Alignments von $x_1 x_2 \dots x_i$ und $y_1 y_2 \dots y_j$, dann gilt eins von

$$(i) \quad OPT(m, n) = \alpha_{x_n y_n} + OPT(m-1, n-1)$$

$$(ii) \quad OPT(m, n) = \delta + OPT(m-1, n)$$

$$(iii) \quad OPT(m, n) = \delta + OPT(m, n-1)$$

Lemma: Für $i \geq 1, j \geq 1$ gilt

$$a) \quad OPT(i, j) = \min \{ \alpha_{x_i y_j} + OPT(i-1, j-1); \\ \delta + OPT(i-1, j); \\ \delta + OPT(i, j-1); \}$$

b) $(i, j) \in M^* \Leftrightarrow$ Minimum oben wird von erstem Term angenommen.

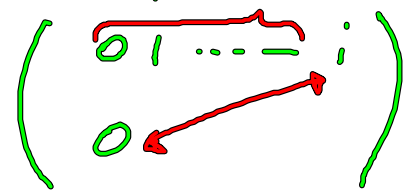
ALGO: Array $[0, \dots, m][0, \dots, n]$ $OPT; i\delta$ Strafe

$$OPT[i][0] = i\delta;$$

$$OPT[0][j] = j\delta;$$

for ($i = 1, \dots, m$) {

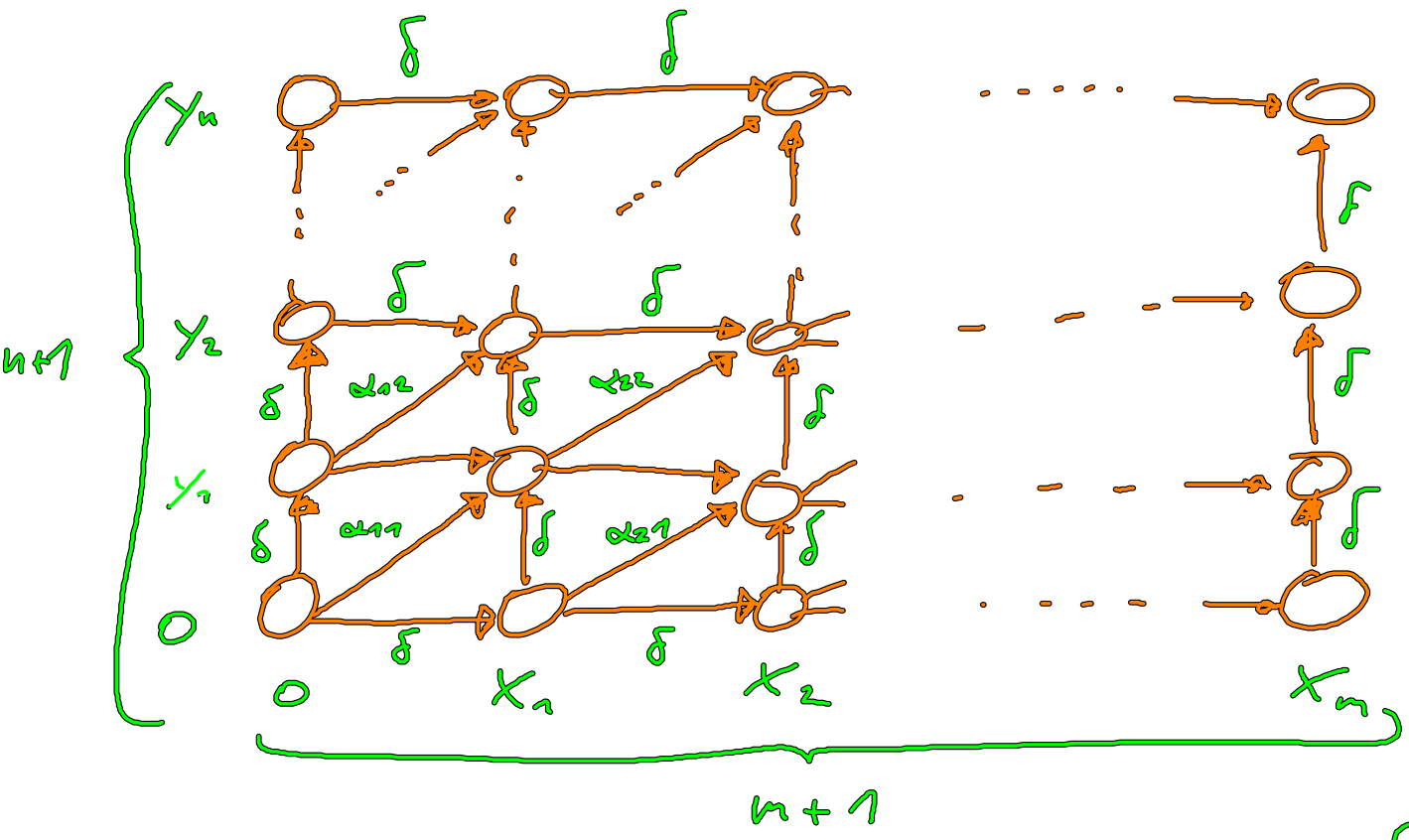
 for ($j = 1, \dots, n$) {

$$OPT[i][j] := \min \{ \dots \};$$


if (Min. vom ersten Term an genommen)
 $M := M \cup \{(i, j)\};$

Alternativ: Graphenalgorithmus

Definiere G wie folgt:



Gewichte: vertikale/horizontale Kante: δ

• Kante von $(i-1, j-1)$ nach (i, j) : α_{x_i, y_j}

Lemma: Sei $f(i, j)$ die Länge eines kürzesten Weges von $(0, 0)$ nach (i, j) . Dann

gilt:

$$F(i,j) = \text{OPT}(i,j)$$

Bew.: IA: $F(0,0) = 0 = \text{OPT}(0,0) \quad \checkmark$

IV: Aussage gilt für i', j' mit
 $i' + j' < i + j$

IS: Letzte Kante auf einem kürzesten Weg von
 $(0,0)$ nach (i,j) ist entweder

- vertikal
- horizontal
- diagonal

$$\Rightarrow F(i,j) = \min \left\{ \begin{aligned} &\delta + F(i, j-1); \\ &\delta + F(i-1, j); \\ &\alpha_{x,y_i} + F(i-1, j-1) \end{aligned} \right\}$$

$$\stackrel{(IV)}{=} \min \left\{ \begin{aligned} &\alpha_{x,y_i} + \text{OPT}(i-1, j-1); \\ &\delta + \text{OPT}(i, j-1); \\ &\delta + \text{OPT}(i-1, j) \end{aligned} \right\}$$

$$= \text{OPT}(i,j) \quad \square$$

