Informatik II - SS 2018

(Algorithmen & Datenstrukturen)

Vorlesung 19 (27.6.2018)

Dynamische Programmierung III

Fabian Kuhn

Algorithmen und Komplexität

UNI FREIBURG

Dynamische Programmierung

DP $\approx$ Rekursion + Memoization

Memoize: *Speichere* Lösungen zu *Teilproblemen*, verwende die gespeicherten Lösungen, falls das gleicht Teilproblem wieder auftaucht.

- Bei den Fibonacci-Zahlen sind die Teilprobleme $F_1, F_2, F_3, \ldots$

Laufzeit $=$ #Teilprobleme $\cdot$ Zeit pro Teilproblem

Editierdistanz

Given: Two strings $A = a_1 a_2 \dots a_m$ and $B = b_1 b_2 \dots b_n$

Goal: Determine the minimum number $D(A, B)$ of edit operations required to transform A into B

Edit operations:

a) **Replace** a character from string A by a character from B

b) **Delete** a character from string A

c) **Insert** a character from string B into A

```
m a - t h e m - - a t i c i a n
m u l t i p l i c a t i o - - n
```

Editierdistanz - Kostenmodell

- Cost for **replacing** character a by b: $\boldsymbol{c(a,b) \geq 0}$

- Capture insert, delete by allowing $a = \varepsilon$ or $b = \varepsilon$:
  - Cost for **deleting** character a: $\boldsymbol{c(a,\varepsilon)}$
  - Cost for **inserting** character b: $\boldsymbol{c(\varepsilon,b)}$

- **Triangle inequality**:

$$c(a,c) \leq c(a,b) + c(b,c)$$

→ each character is changed at most once!

- **Unit cost model**: $c(a,b) = \begin{cases} 1, & \text{if } a \neq b \\ 0, & \text{if } a = b \end{cases}$

Rekursive Struktur

- Optimal "alignment" of strings (unit cost model)

  `bbcadfagikccm` and `abbagflrgikacc`:

```
- b b c a g f a - g i k - c c m
a b b - a d f l r g i k a c c -
```

- Consists of optimal "alignments" of sub-strings, e.g.:

```
-bbcagfa      -gik-ccm
abb-adfl      rgikacc-
```

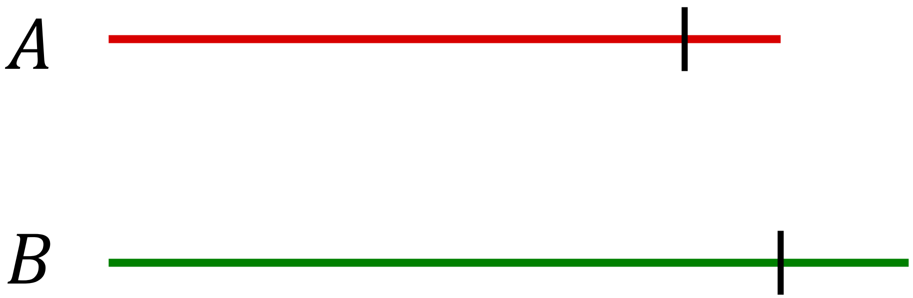
- Edit distance between $A_{1,m} = a_1 \ldots a_m$ and $B_{1,n} = b_1 \ldots b_n$:

$$D(A,B) = \min_{k,\ell}\left\{D\left(A_{1,k}, B_{1,\ell}\right) + D\left(A_{k+1,m}, B_{\ell+1,n}\right)\right\}$$

Berechnen der Editierdistanz

Let $A_k := a_1 \dots a_k$, $B_\ell := b_1 \dots b_\ell$, and

$$D_{k,\ell} := D(A_k, B_\ell)$$



Berechnen der Editierdistanz

Three ways of ending an "alignment" between A_k and B_ℓ:

1. a_k is replaced by b_ℓ:

   $D_{k,\ell} = D_{k-1,\ell-1} + c(a_k, b_\ell)$

2. a_k is deleted:

   $D_{k,\ell} = D_{k-1,\ell} + c(a_k, \varepsilon)$

3. b_ℓ is inserted:

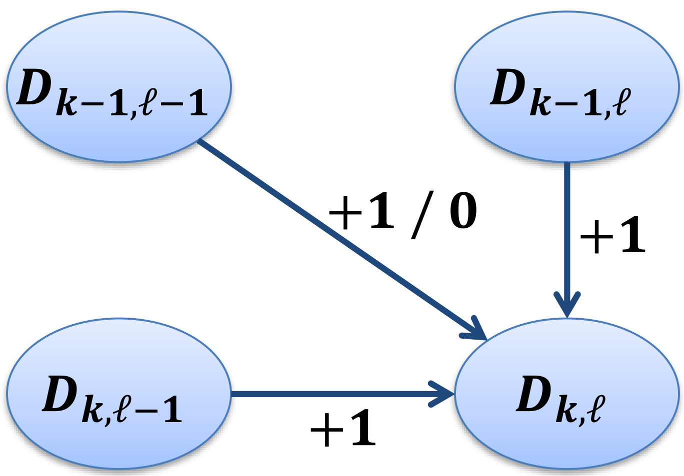
   $D_{k,\ell} = D_{k,\ell-1} + c(\varepsilon, b_\ell)$

Berechnen der Editierdistanz

- Recurrence relation (for $k, \ell \geq 1$)

$$D_{k,\ell} = \min \begin{Bmatrix} D_{k-1,\ell-1} & + c(a_k, b_\ell) \\ D_{k-1,\ell} & + c(a_k, \varepsilon) \\ D_{k,\ell-1} & + c(\varepsilon, b_\ell) \end{Bmatrix} = \underbrace{\min \begin{Bmatrix} D_{k-1,\ell-1} & + 1 \;/\; 0 \\ D_{k-1,\ell} & + 1 \\ D_{k,\ell-1} & + 1 \end{Bmatrix}}_{\textbf{unit cost model}}$$

- Need to compute $D_{i,j}$ for all $0 \leq i \leq k$, $0 \leq j \leq \ell$:



Rekursionsgleichung Editierdistanz

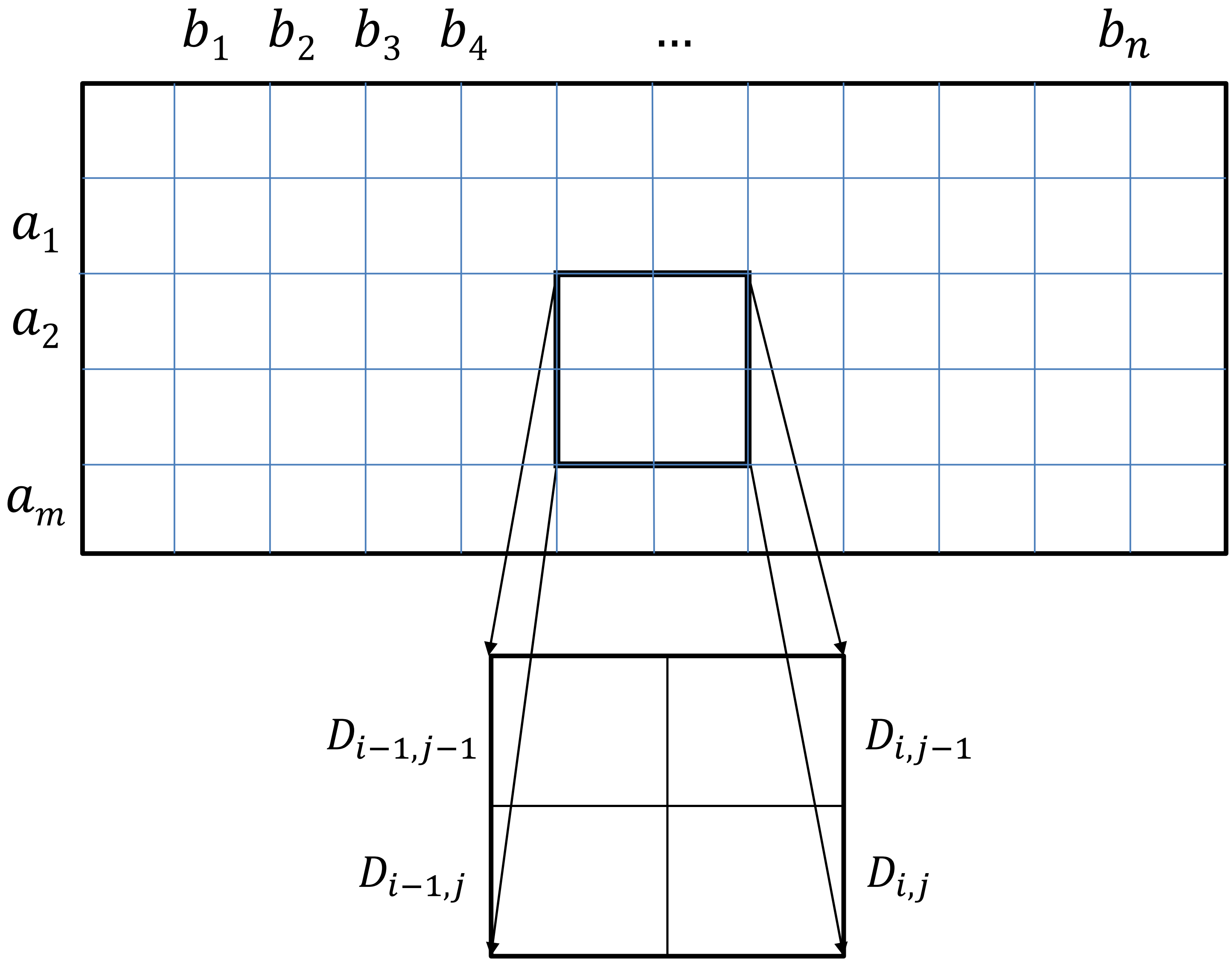
Base cases:

$$
\begin{aligned}
D_{0,0} &= D(\varepsilon,\varepsilon) = 0\\
D_{0,j} &= D(\varepsilon, B_j) = D_{0,j-1} + c(\varepsilon, b_j)\\
D_{i,0} &= D(A_i, \varepsilon) = D_{i-1,0} + c(a_i, \varepsilon)
\end{aligned}
$$

Recurrence relation:

$$
D_{i,j} = \min \begin{Bmatrix} D_{k-1,\ell-1} & + c(a_k, b_\ell) \\ D_{k-1,\ell} & + c(a_k, \varepsilon) \\ D_{k,\ell-1} & + c(\varepsilon, b_\ell) \end{Bmatrix}
$$

Reihenfolge der Teilprobleme



Pseudocode

Algorithm *Edit-Distance*

Input: 2 strings $A = a_1 \ldots a_m$ and $B = b_1 \ldots b_n$

Output: matrix $D = \left(D_{ij}\right)$

```
D[0,0] := 0;
for i := 1 to m do D[i,0] := i;
for j := 1 to n do D[0,j] := j;
for i := 1 to m do
    for j := 1 to n do
```

$$D[i,j] := \min \begin{Bmatrix} D[i-1,j] & +1 \\ D[i,j-1] & +1 \\ D[i-1,j-1] & + c\left(a_i, b_j\right) \end{Bmatrix};$$

Beispiel

		a	b	c	c	a
b						
a						
b						
d						
a						

Editieroperationen

		a	b	c	c	a
	0	1	2	3	4	5
b	1	1	1	2	3	4
a	2	1	2	2	3	3
b	3	2	1	2	3	4
d	4	3	2	2	3	4
a	5	4	3	3	3	3

Editieroperationen – Pseudocode

Algorithm *Edit-Operations*(i, j)
Input: matrix D (already computed)
Output: list of edit operations

if $i = 0$ **and** $j = 0$ **then return** empty list

if $i \neq 0$ **and** $D[i,j] = D[i-1,j] + 1$ **then**
    return *Edit-Operations*$(i-1, j) \circ$ „delete a_i“

else if $j \neq 0$ **and** $D[i,j] = D[i,j-1] + 1$ **then**
    return *Edit-Operations*$(i, j-1) \circ$ „insert b_j“

else // $D[i,j] = D[i-1,j-1] + c(a_i, b_j)$
    if $a_i = b_i$ **then return** *Edit-Operations*$(i-1, j-1)$
    else return *Edit-Operations*$(i-1, j-1) \circ$ „replace a_i by b_j“

Initial call: *Edit-Operations*(*m*,*n*)

Editieroperationen

		a	b	c	c	a
	0	1	2	3	4	5
b	1	1	1	2	3	4
a	2	1	2	2	3	3
b	3	2	1	2	3	4
d	4	3	2	2	3	4
a	5	4	3	3	3	3

Editierdistanz – Zusammenfassung

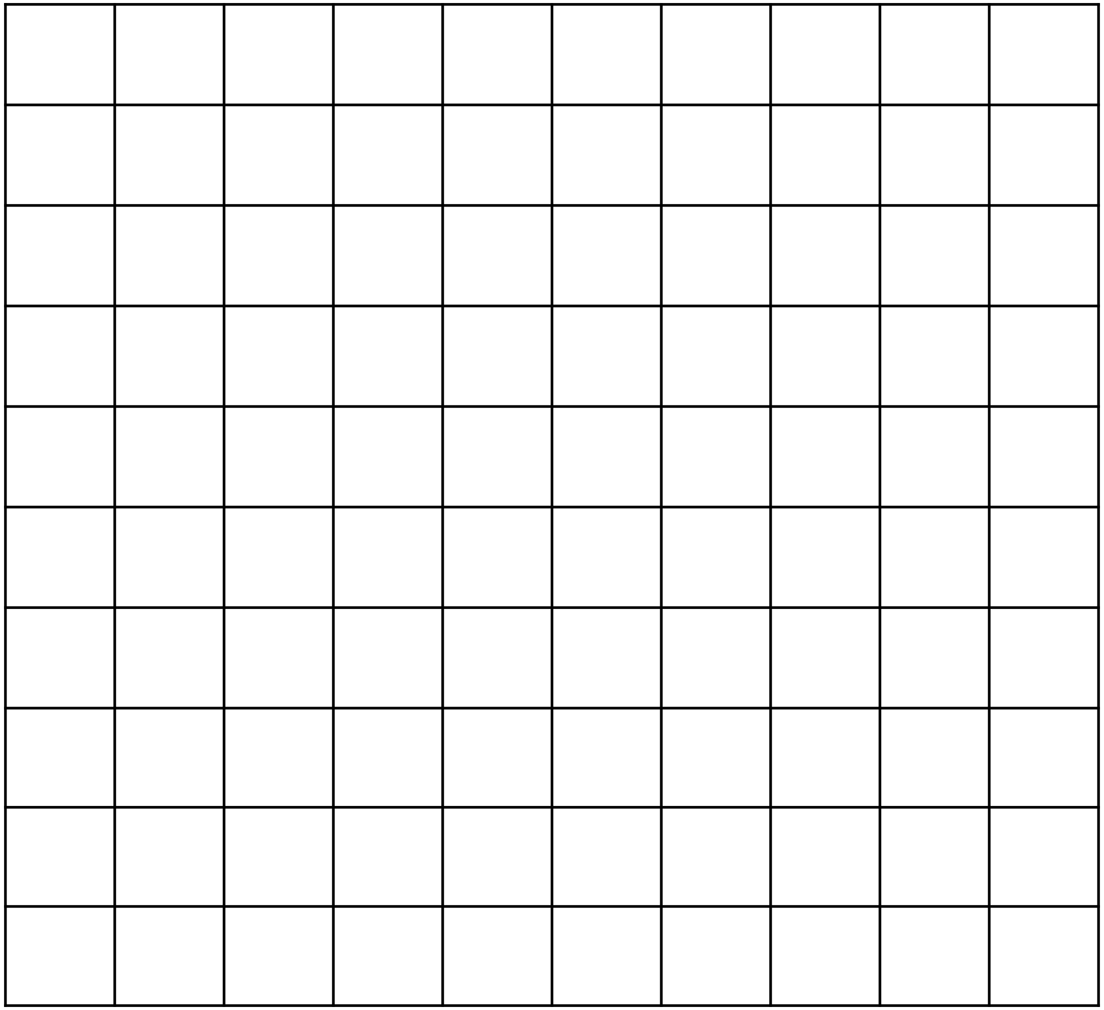
- Edit distance between two strings of length m and n can be computed in $O(mn)$ time.

- Obtain the edit operations:
  - for each cell, store which rule(s) apply to fill the cell
  - track path backwards from cell (m, n)
  - can also be used to get all optimal "alignments"

- Unit cost model:
  - interesting special case
  - each edit operation costs 1

Editierdistanz schneller berechnen?

- Wie klein können die Lösungen der Teilprobleme sein?



Editierdistanz schneller berechnen?

- Falls wir wissen, dass die Editierdistanz $\leq \delta$ ist?

0	1	2	3	4	5	6	7	8	9
1	0	1	2	3	4	5	6	7	8
2	1	0	1	2	3	4	5	6	7
3	2	1	0	1	2	3	4	5	6
4	3	2	1	0	1	2	3	4	5
5	4	3	2	1	0	1	2	3	4
6	5	4	3	2	1	0	1	2	3
7	6	5	4	3	2	1	0	1	2
8	7	6	5	4	3	2	1	0	1
9	8	7	6	5	4	3	2	1	0

Editierdistanz schneller berechnen?

Falls wir wissen, dass die Editierdistanz $\leq \delta$ ist:

- Wir benötigen nur die Hauptdiagonale und die δ ersten Nebendiagonalen (auf beide Seiten)
- Das sind $O(\delta \cdot n)$ Einträge (falls beide Strings Länge $O(n)$ haben)

Laufzeit + Speicher: $\boldsymbol{O(\delta \cdot n)}$

- Was tun, falls δ nicht bekannt ist?

  1. Wir raten δ und probieren, bis wir ein genug grosses δ gewählt haben (falls wir δ zu klein raten, dann entdecken wir das)

  2. Wir versuchen die Tabelle direkt entlang der Diagonale zu füllen und verwenden die bekannten unteren Schranken für die Einträge
     - Werte in den Nebendiagonalen erst berechnen, wenn man sie braucht
     - Immer zuerst die fehlenden Werte berechnen, welche näher bei der Hauptdiagonale liegen

Editierdistanz schneller berechnen?

Den Parameter δ raten?

- Annahme: Editierdistanz $= D$
- Versuche mit Parameter $\delta \geq 1$
  - Algorithmus hat Laufzeit $O(n \cdot \delta)$
  - Falls $D \leq \delta$, dann finden wir die Editierdistanz (und die Edit.-op.)
  - Falls $D > \delta$, dann finden wir das heraus

Versuch 1, wähle $\boldsymbol{\delta = 1, 2, 3, 4, 5}$, …

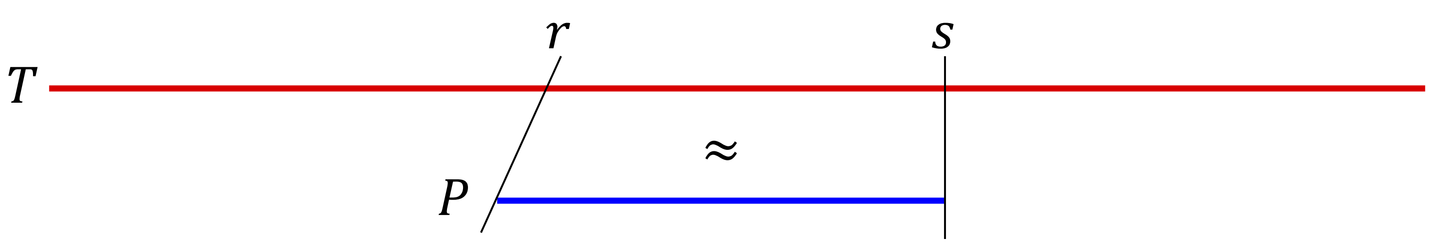
Versuch 2, wähle $\boldsymbol{\delta = 1, 2, 4, 8, 16}$, …

Approximate String Matching

Given: strings $T = t_1 t_2 \dots t_n$ (text) and $P = p_1 p_2 \dots p_m$ (pattern).

Goal: Find an interval $[r, s]$, $1 \leq r \leq s \leq n$ such that the sub-string $T_{r,s} := t_r \dots t_s$ is the one with highest similarity to the pattern P:

$$\arg\min_{1 \leq r \leq s \leq n} D\left(T_{r,s}, P\right)$$



Approximate String Matching

Naive Lösung:

for all $1 \leq r \leq s \leq n$ **do**

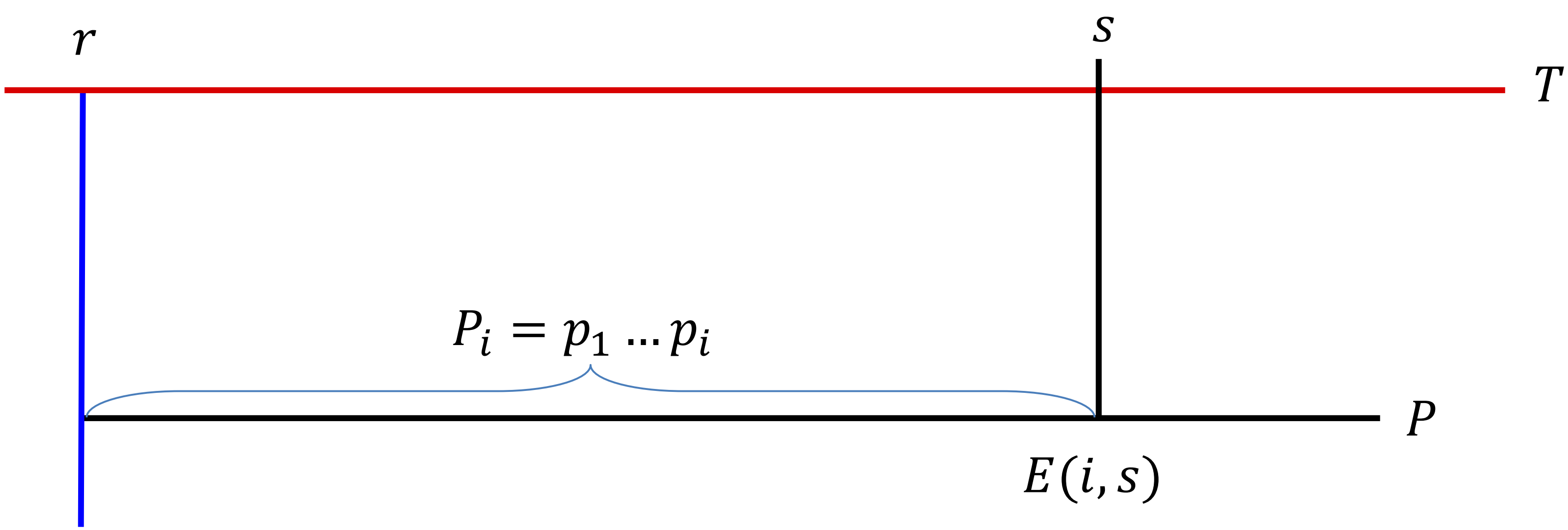
compute $D(T_{r,s}, P)$

choose the minimum

Approximate String Matching

A related problem:

- For each position s in the text and each position i in the pattern compute the minimum edit distance $E(i, s)$ between $P_i = p_1 \dots p_i$ and any substring $T_{r,s}$ of T that ends at position s.



Approximate String Matching

Three ways of ending optimal alignment between T_b and P_i:

1. t_b is replaced by p_i:

   $$E_{b,i} = E_{b-1,i-1} + c(t_b, p_i)$$

2. t_b is deleted:

   $$E_{b,i} = E_{b-1,i} + c(t_b, \varepsilon)$$

3. p_i is inserted:

   $$E_{b,i} = E_{b,i-1} + c(\varepsilon, p_i)$$

Approximate String Matching

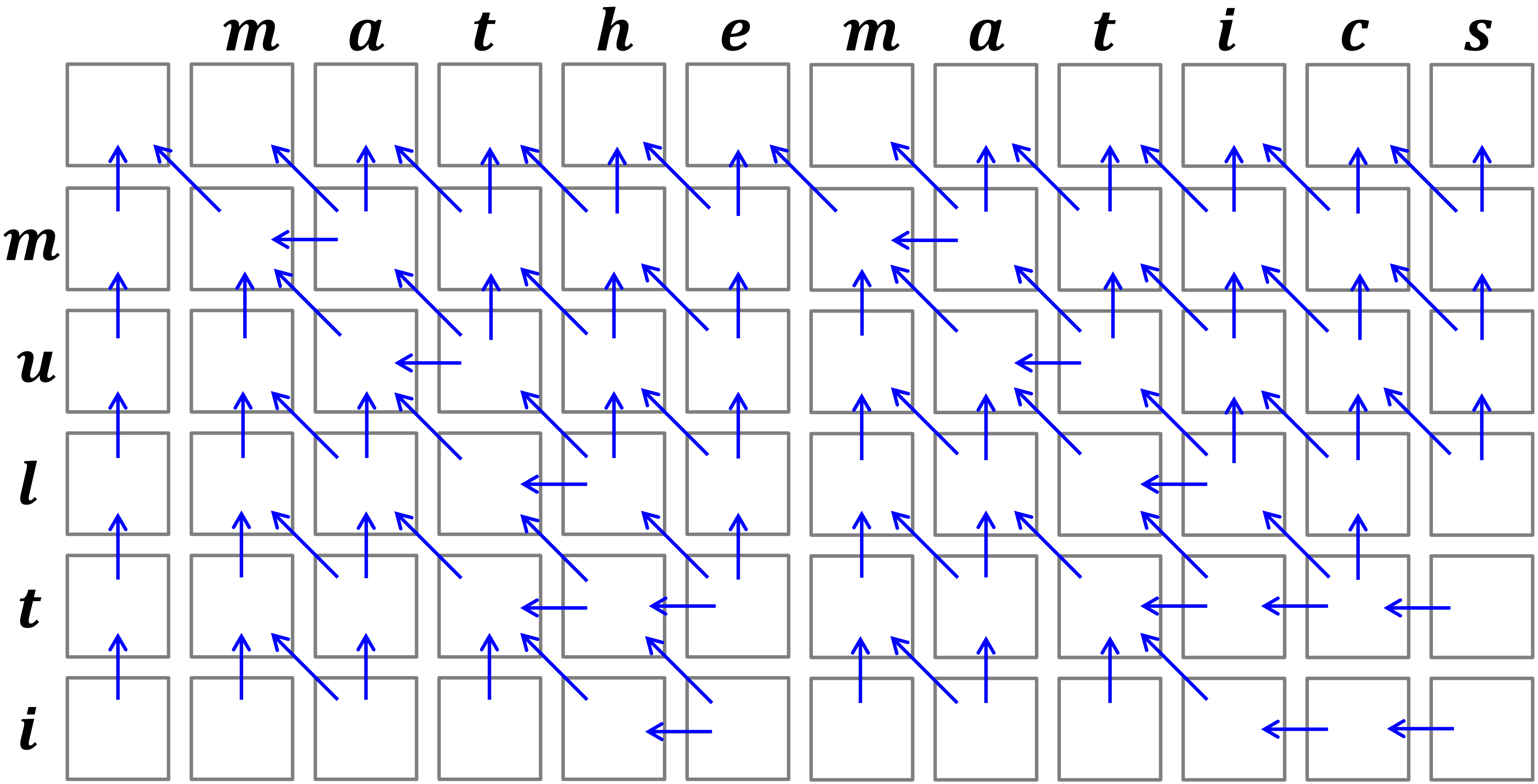
Recurrence relation (unit cost model):

$$E_{b,i} = \min \begin{Bmatrix} E_{b-1,i-1} + 1 \\ E_{b-1,i} \quad + 1 \\ E_{b,i-1} \quad + 1 \end{Bmatrix}$$

Base cases:

$$E_{0,0} = 0$$
$$E_{0,i} = i$$
$$E_{i,0} = 0$$

Beispiel



Approximate String Matching

- Optimal matching consists of optimal sub-matchings
- Optimal matching can be computed in $O(mn)$ time
- Get matching(s):
  - Start from minimum entry/entries in bottom row
  - Follow path(s) to top row
- Algorithm to compute $E(b, i)$ identical to edit distance algorithm, except for the initialization of $E(b, 0)$

Verwandte Problem in der Bioinformatik

Sequence Alignment:

Find optimal alignment of two given DNA, RNA, or amino acid sequences.

```
G A – C G G A T T A G
G A T C G G A A T – G
```

Global vs. Local Alignment:

- *Global alignment*: find optimal alignment of 2 sequences
- *Local alignment*: find optimal alignment of sequence 1 (patter) with sub-sequence of sequence 2 (text)

Eier-Rätsel

Gegeben: Ein Haus mit $n \geq 1$ Stockwerken und $k \geq 1$ Eier

Finde mit möglichst wenigen Versuchen heraus, von welchen Stock-werken ein Ei ganz bleibt, wenn man es fallen lässt.

Wieviele Versuche braucht es im Worst Case für gegebene n und k?

Annahmen:

- Die Stockwerke sind von 1 (EG) bis n nummeriert
- Alle Eier sind gleich
- Falls ein Ei ganz bleibt, dann verhält es sich danach, wie ein Ei, dass noch nie fallen gelassen wurde
- Falls ein Ei denn Fall von Stockwerk s überlebt, dann überlebt es auch den Fall von Stockwerken $s' < s$

Eier-Rätsel

Gegeben: Ein Haus mit $n \geq 1$ Stockwerken und $k \geq 1$ Eier

Wieviele Versuche braucht es im Worst Case für gegebene n und k?

Einfache Fälle:

- $k = 1$ (es steht nur 1 Ei zur Verfügung)
  - Man muss der Reihe nach die Stockwerke $1, 2, 3, \dots, n$ ausprobieren
- $k = \infty$ (es stehen beliebig viele Eier zur Verfügung)
  - Binäre Suche
- Kennt jemand die Lösung für $k = 2$?
  - Wieviele Versuche braucht es z.B. für 100 Stockwerke mit 2 Eiern?

Eier-Rätsel

Gegeben: Ein Haus mit $n \geq 1$ Stockwerken und $k \geq 1$ Eier

Wieviele Versuche braucht es im Worst Case für gegebene n und k?

Rekursive Formulierung:

- $\text{attempts}(n, k)$: Anzahl Versuche bei n Stockwerken und k Eiern

Nächste Woche

- Ich werde nächste Woche nicht in Freiburg sein. Da wir beim Vorlesungsstoff genug weit sind, wird nächste Woche keine Vorlesung stattfinden.
- Der Montagstermin wird ganz ausfallen
- Am Mittwoch wird im Vorlegunssaal (106-00-036) eine Übungs-, bzw. Fragestunde stattfinden, in der die Assistenten zusätzliche Übungen und/oder eine alte Klausur besprechen.

- Für die aktuelle Übung haben sie 2 Wochen Zeit (Abgabe: 9.7.)
- Danach gibt es noch eine reguläre Übung
- In der letzten Woche wird es noch ein Bonus-Übungsblatt geben
  - damit diejenigen, welche knapp unter 50% sind, noch auf die nötigen Punkte für die Klausurzulassung kommen können