Chapter 3
Dynamic Programming

Algorithm Theory
WS 2019/20

Fabian Kuhn

Dynamic Programming (DP)

DP $\approx$ Recursion + Memoization

Recursion: Express problem *recursively* in terms of (a 'small' number of) *subproblems* (of the same kind)

Memoize: *Store* solutions for *subproblems* reuse the stored solutions if the same subproblems has to be solved again

Weighted interval scheduling: subproblems $\underline{W(1)}, \underline{W(2)}, W(3), \ldots$

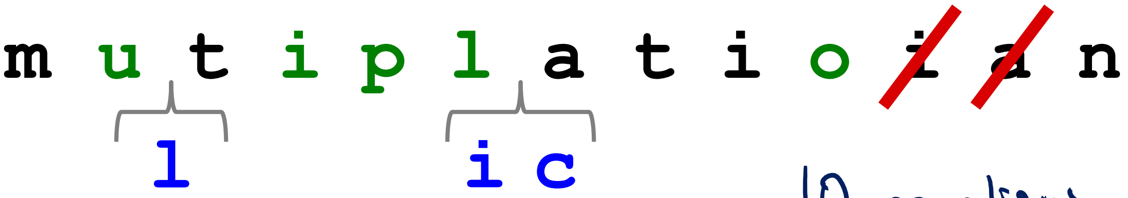
runtime $=$ $\underline{\underline{\text{\#subproblems} \cdot \text{time per subproblem}}}$

String Matching Problems

Edit distance:

- For two given strings A and B, efficiently compute the **edit distance $D(A, B)$** (# edit operations to transform A into B) as well as a minimum sequence of edit operations that transform A into B.

- **Example:** mathematician → multiplication:



10 operations

Edit Distance

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

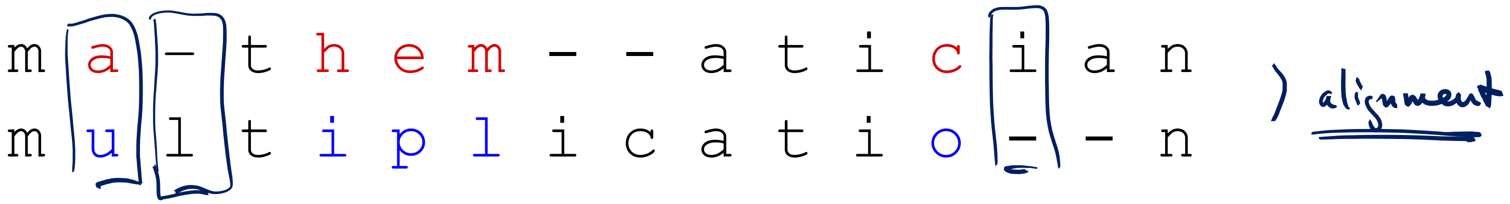
Goal: Determine the minimum number $D(A, B)$ of edit operations required to transform A into B

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
```

} alignment

Edit Distance – Cost Model

$c(a,a) = 0$

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

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

- **Triangle inequality**:

$$\underline{c(a,c) \leq c(a,b) + c(b,c)}$$

→ each character is changed at most once!

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

Recursive Structure

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

  `bbcadfagikccm` **and** `abbagflrgikacc:`

```
               k
- 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 -
               ℓ
```

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

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

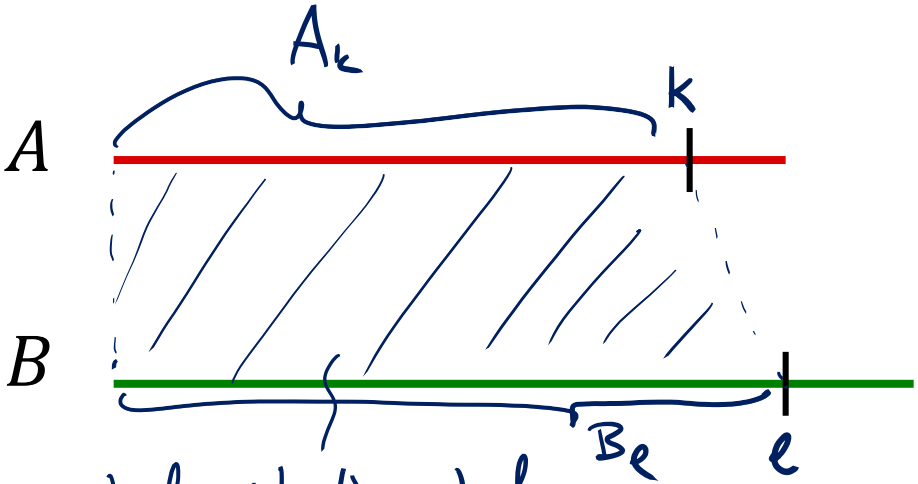
- Edit distance between $A_{1,m} = a_1 \dots a_m$ and $B_{1,n} = b_1 \dots b_n$:

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

Computation of the Edit Distance

Let $\underline{A_k} := a_1 \ldots a_k$, $\underline{B_\ell} := b_1 \ldots b_\ell$, and

$A_{1,k}$ $B_{1,\ell}$ $\underline{D_{k,\ell}} := \underline{D(A_k, B_\ell)}$



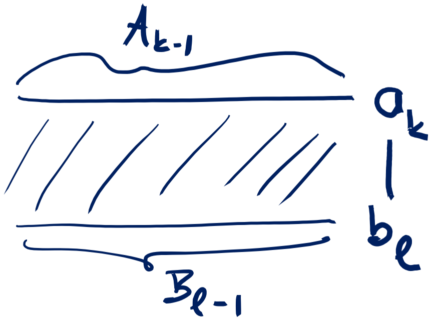
$$\underbrace{\text{cost of an opt. alignment of } A_k \text{ \& } B_\ell}_{\underline{D_{k,\ell}}}$$

Computation of the Edit Distance

Three ways of ending an “alignment” between A_k and B_ℓ:

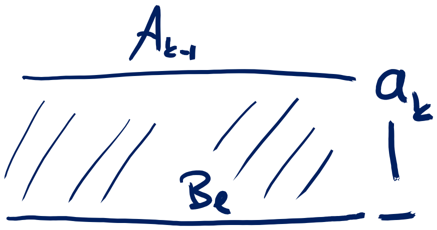
1. a_k is replaced by b_ℓ:

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



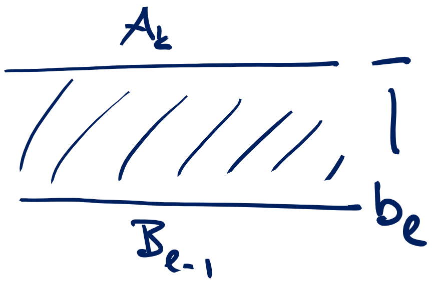
2. a_k is deleted:

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



3. b_ℓ is inserted:

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

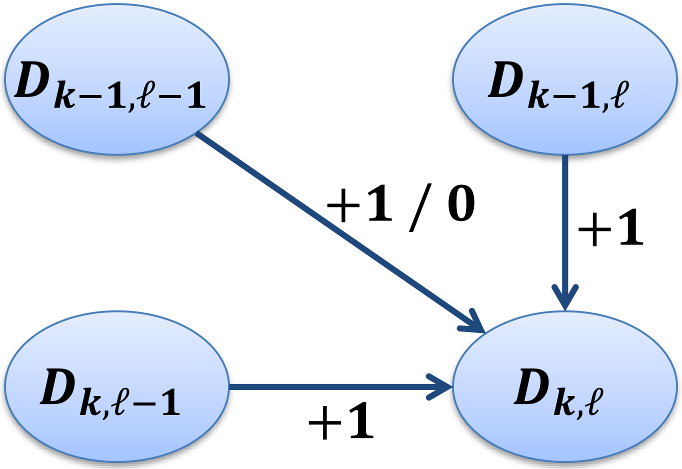


Computing the Edit Distance

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

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

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



Recurrence Relation for the Edit Distance

Base cases:

unit cost

$$\underline{D_{0,0}} = D(\varepsilon, \varepsilon) = \underline{0}$$

$$D_{\underline{0,j}} = D(\varepsilon, B_j) = D_{0,j-1} + c(\underline{\varepsilon, b_j})$$

$$D_{i,0} = D(A_i, \varepsilon) = D_{i-1,0} + c(a_i, \varepsilon)$$

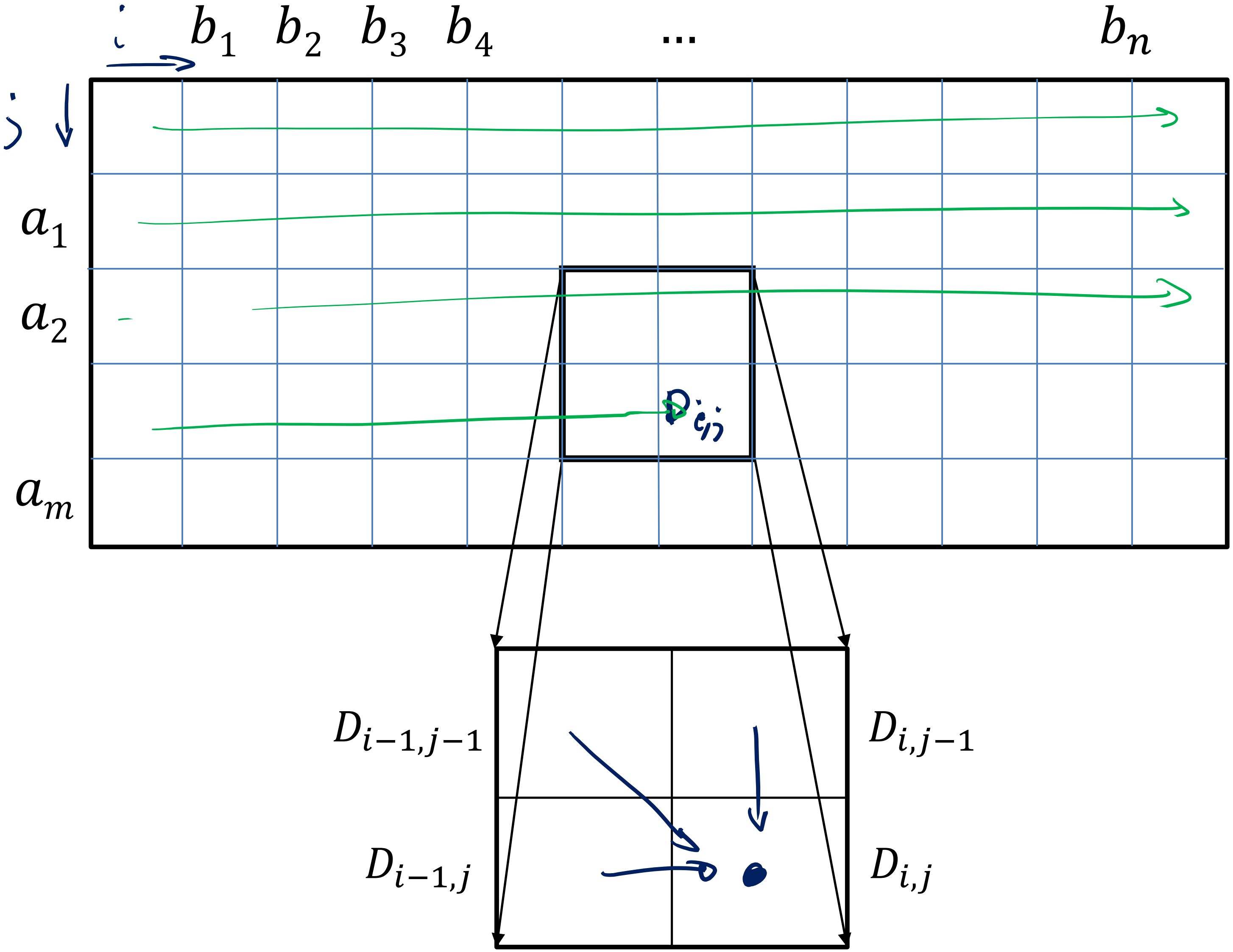
$D_{0,j} = j$

$D_{i,0} = i$

Recurrence relation:

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

Order of solving the subproblems



Algorithm for Computing the Edit Distance

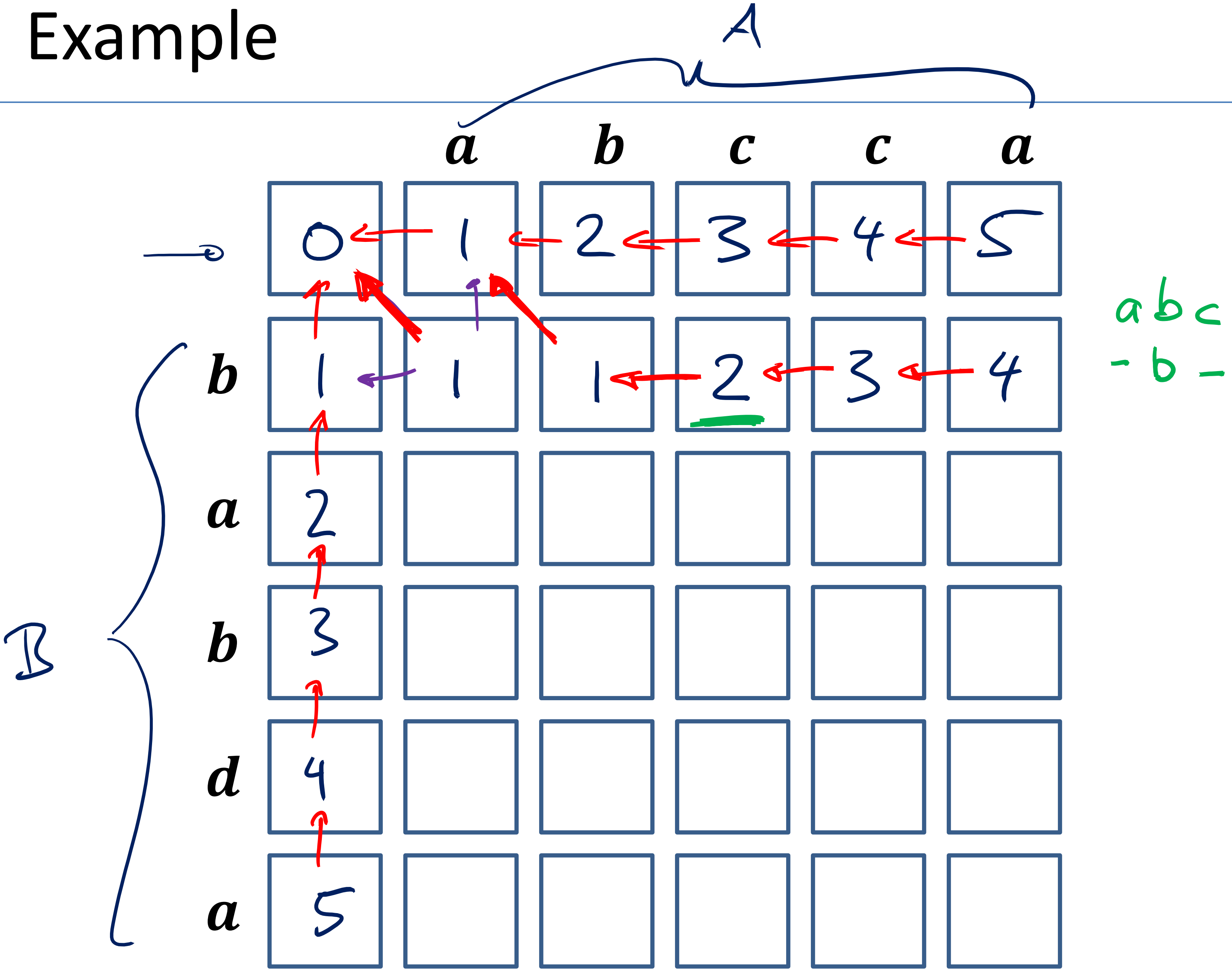
Algorithm *Edit-Distance*

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

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

Example



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

abc
-b-

Edit Operations

		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

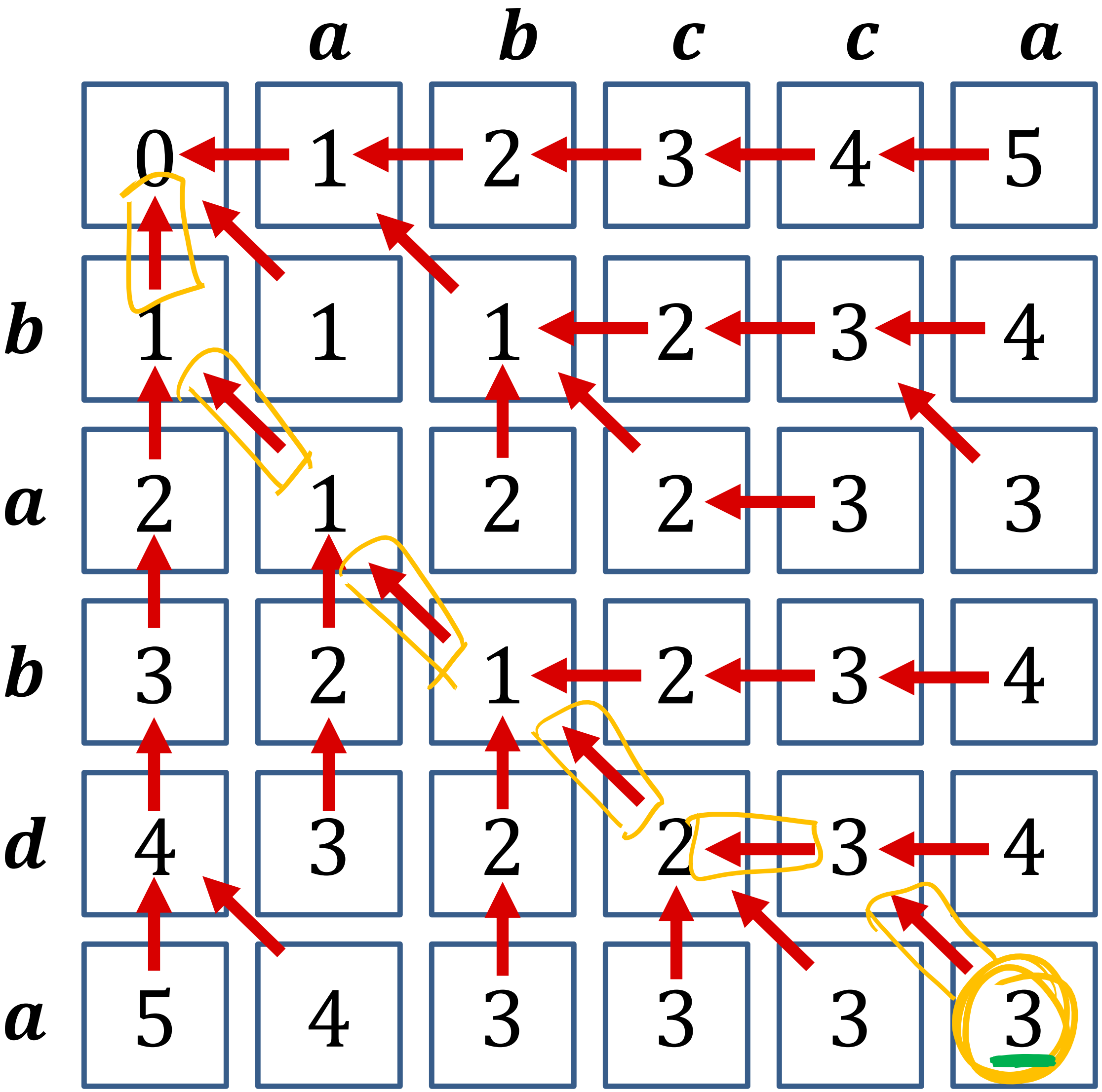
Computing the Edit Operations

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*)

Edit Operations



		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

- a b c c a

b a b d - a

Edit Distance: Summary

- Edit distance between two strings of length m and n can be computed in $\underline{O(mn)}$ 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

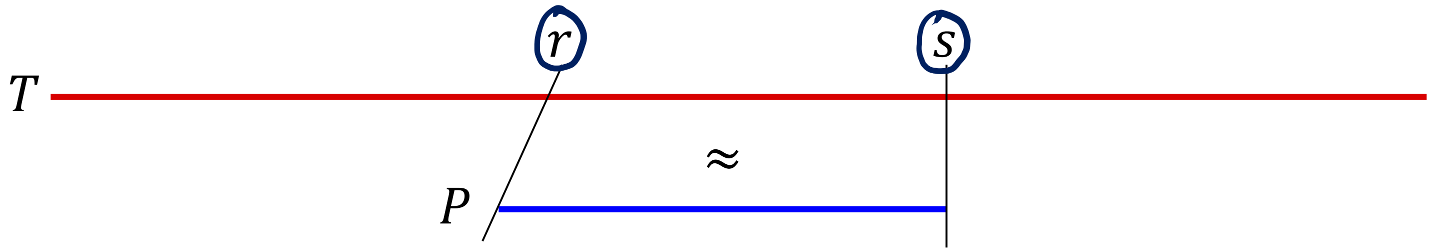
Approximate String Matching

$m \ll n$

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

$$\underset{1 \le r \le s \le n}{\arg\min}\, D\big(T_{r,s}, P\big)$$



Approximate String Matching

Naive Solution:

#combinations : $O(n^2)$

```
for all $1 \le r \le s \le n$ do
    compute $D(T_{r,s}, P)$
choose the minimum
```

<u>cost:</u> $O((s-r) \cdot m) = O(m \cdot n)$

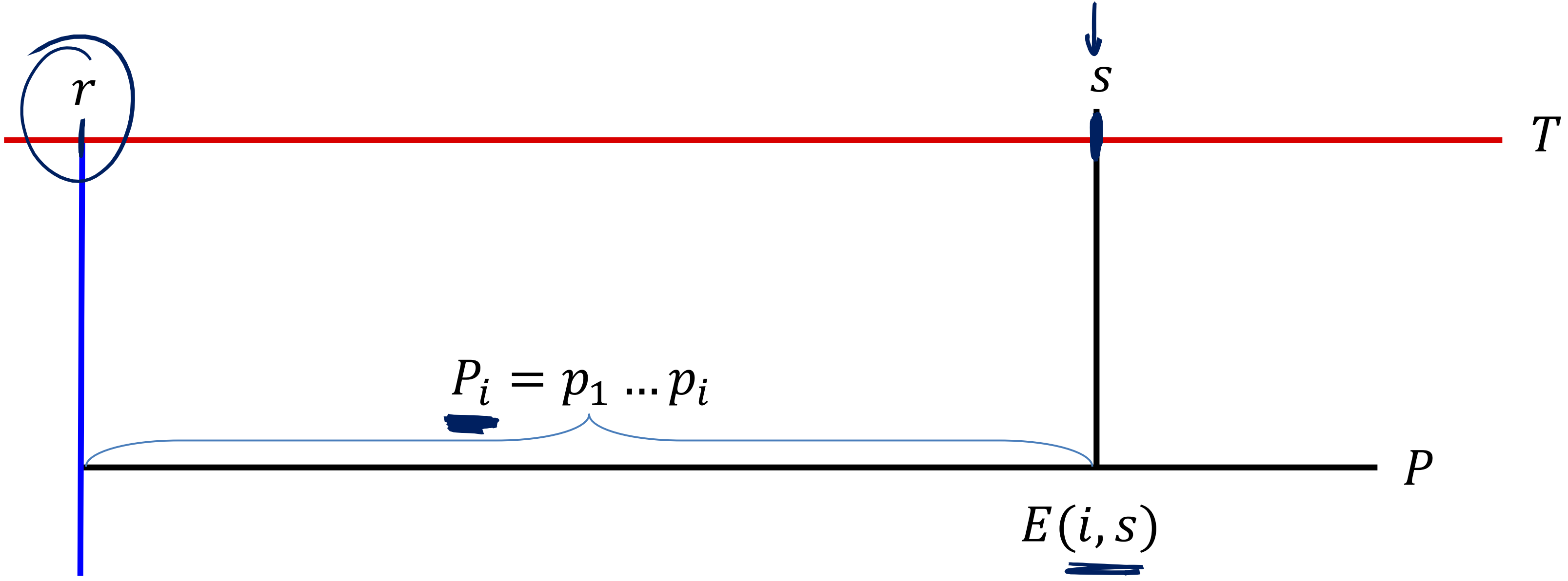
<u>overall:</u> $O(m \cdot n^3)$

unit cost, can be more clever : $O(n \cdot m^3)$

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.

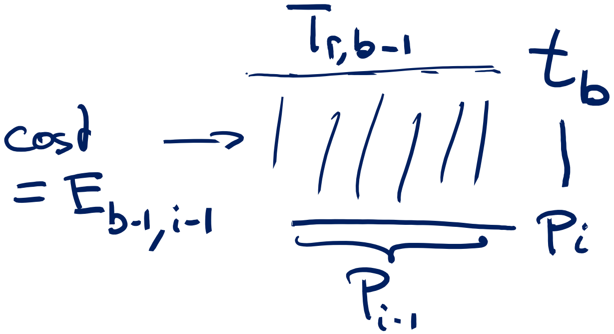


Approximate String Matching

Three ways of ending optimal alignment between T_b and P_i:

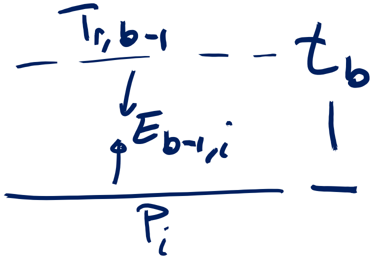
1. t_b is replaced by p_i:

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



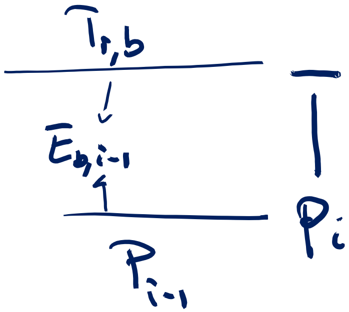
2. t_b is deleted:

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



3. p_i is inserted:

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



Approximate String Matching

Recurrence relation (unit cost model):

$$E_{b,i} = \min \begin{Bmatrix} \underline{E_{b-1,i-1}} + 1 \,/\, \underline{0} \\ E_{b-1,i} + 1 \\ E_{b,i-1} + 1 \end{Bmatrix}$$

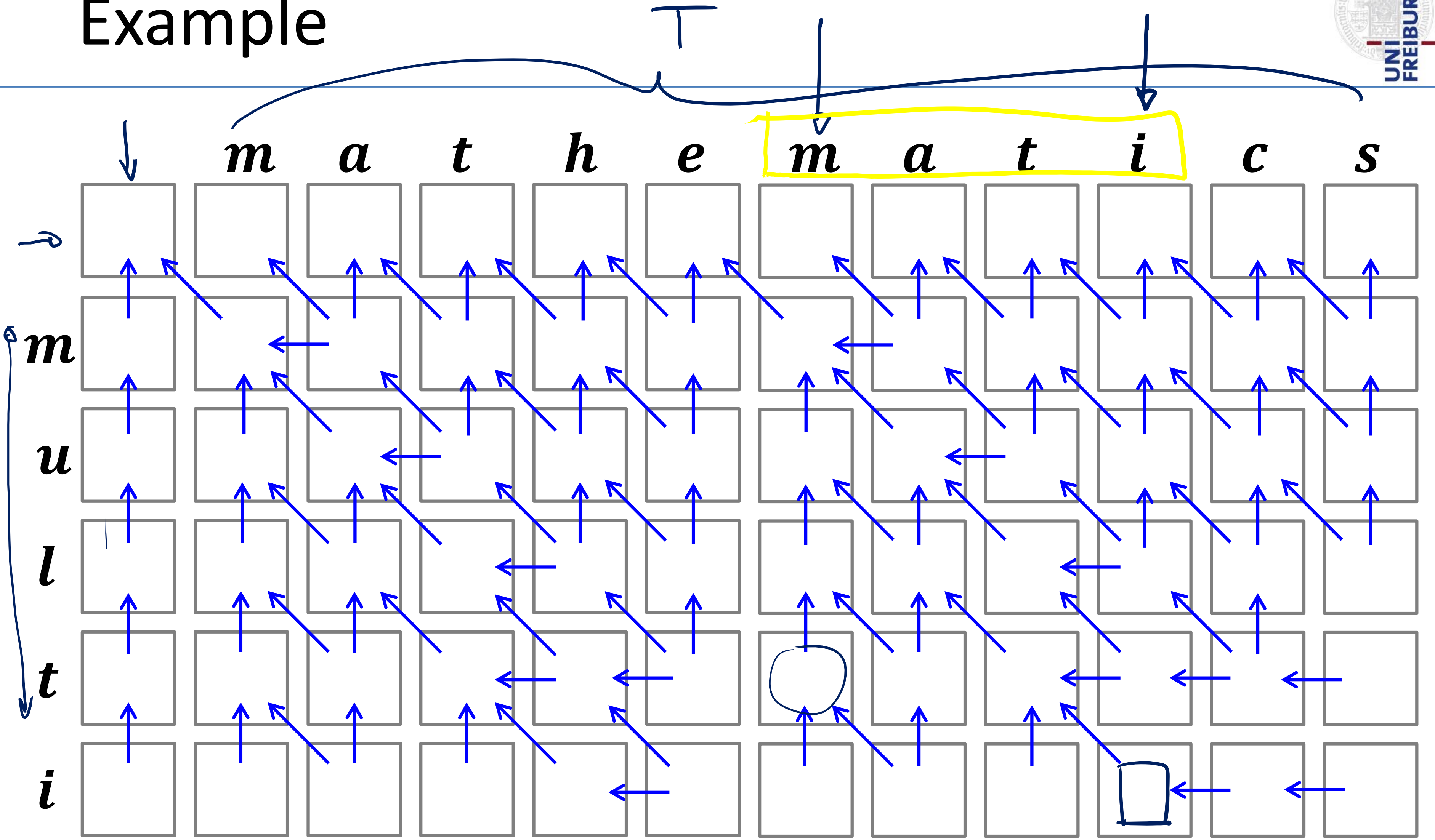
Base cases:

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

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

$$\underline{{\color{red}E_{i,0} = 0}}$$

Example



m a - t i
| | | | |
m u l t i

Approximate String Matching

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

Related Problems in Bioinformatics

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)