Theory I
Algorithm Design and Analysis

(11 - Edit distance and approximate string matching)

Prof. Dr. Th. Ottmann

Dynamic programming

- Algorithm design technique often used for optimization problems
- Generally usable for recursive approaches if the same partial solutions are required more than once
- Approach: store partial results in a table
- Advantage: improvement of complexity, often polynomial instead of exponential

Two different approaches

Bottom-up:

+ controlled efficient table management, saves time
+ special optimized order of computation, saves space
- requires extensive recoding of the original program
- possible computation of unnecessary values

Top-down: (Note-pad method)

+ original program changed only marginally or not at all
+ computes only those values that are actually required
- separate table management takes additional time
- table size often not optimal

Problem: similarity of strings

Edit distance

For two given *A* and *B*, compute, as efficiently as possible, the edit distance *D(A,B)* and a minimal sequence of edit operations which transforms *A* into *B*.

i	n	f	-	-	-	o	r	m	a	t	i	k	-
i	n	t	e	r	p	o	l	-	a	t	i	o	n

Problem: similarity of strings

Approximate string matching

For a given text *T*, a pattern *P*, and a distance *d*, find all substrings *P´* in *T* with $D(P,P') \leq d$

Sequence alignment

Find optimal alignments of DNA sequences

G A G C A - C T T G G A T T C T C G G
\- - - C A C G T G G - - - - - - - - -

Edit distance

Given: two strings $A = a_1a_2 .... a_m$ and $B = b_1b_2 ... b_n$

Wanted: minimal cost $D(A,B)$ for a sequence of edit operations to transform A into B.

Edit operations:

1. Replace one character in A by a character from B
2. Delete one character from A
3. Insert one character from B

Edit distance

Cost model:

$$c(a,b) = \begin{cases} 1 & \text{if } a \neq b \\ 0 & \text{if } a = b \end{cases}$$

$$a = \varepsilon,\ \ b = \varepsilon \ \text{ possible}$$

We assume the triangle inequality holds for *c*:

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

→ Each character is changed at most once

Edit distance

Trace as representation of edit sequences

```
A =     b a a c a a b c
B =   a b a c b c a c
```

or using indels

```
A =   - b a a c a - a b c
B =   a b a - c b c a - c
```

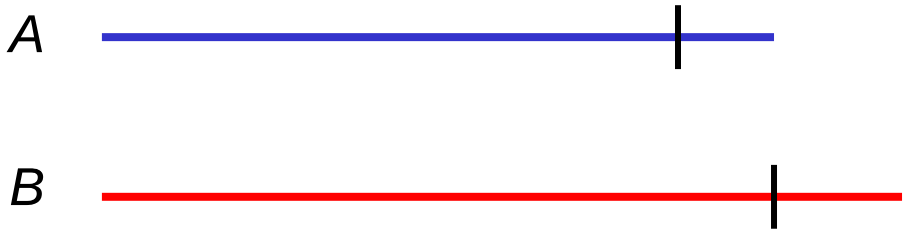
Edit distance (cost): 5

Division of an optimal trace results in two optimal sub-traces
→ dynamic programming can be used

Computation of the edit distance

Let $A_i = a_1 \ldots a_i$ and $B_j = b_1 \ldots . b_j$

$$D_{i,j} = D(A_i, B_j)$$



Computation of the edit distance

Three possibilities of ending a trace:

1. a_m is replaced by b_n :
   $D_{m,n} = D_{m\text{-}1,n\text{-}1} + c(a_m, b_n)$

2. a_m is deleted: $D_{m,n} = D_{m-1,n} + 1$

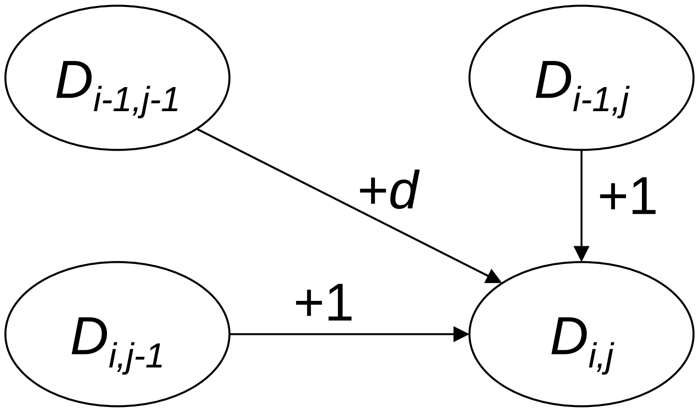
3. b_n is inserted: $D_{m,n} = D_{m,n-1} + 1$

Computation of the edit distance

Recurrence relation, if $m,n \geq 1$:

$$D_{m,n} = \min \begin{Bmatrix} D_{m-1,n-1} & + & c(a_m, b_n), \\ D_{m-1,n} & + & 1, \\ D_{m,n-1} & + & 1 \end{Bmatrix}$$

→ Computation of all $D_{i,j}$ is required, $0 \leq i \leq m$, $0 \leq j \leq n$.



Recurrence relation for the edit distance

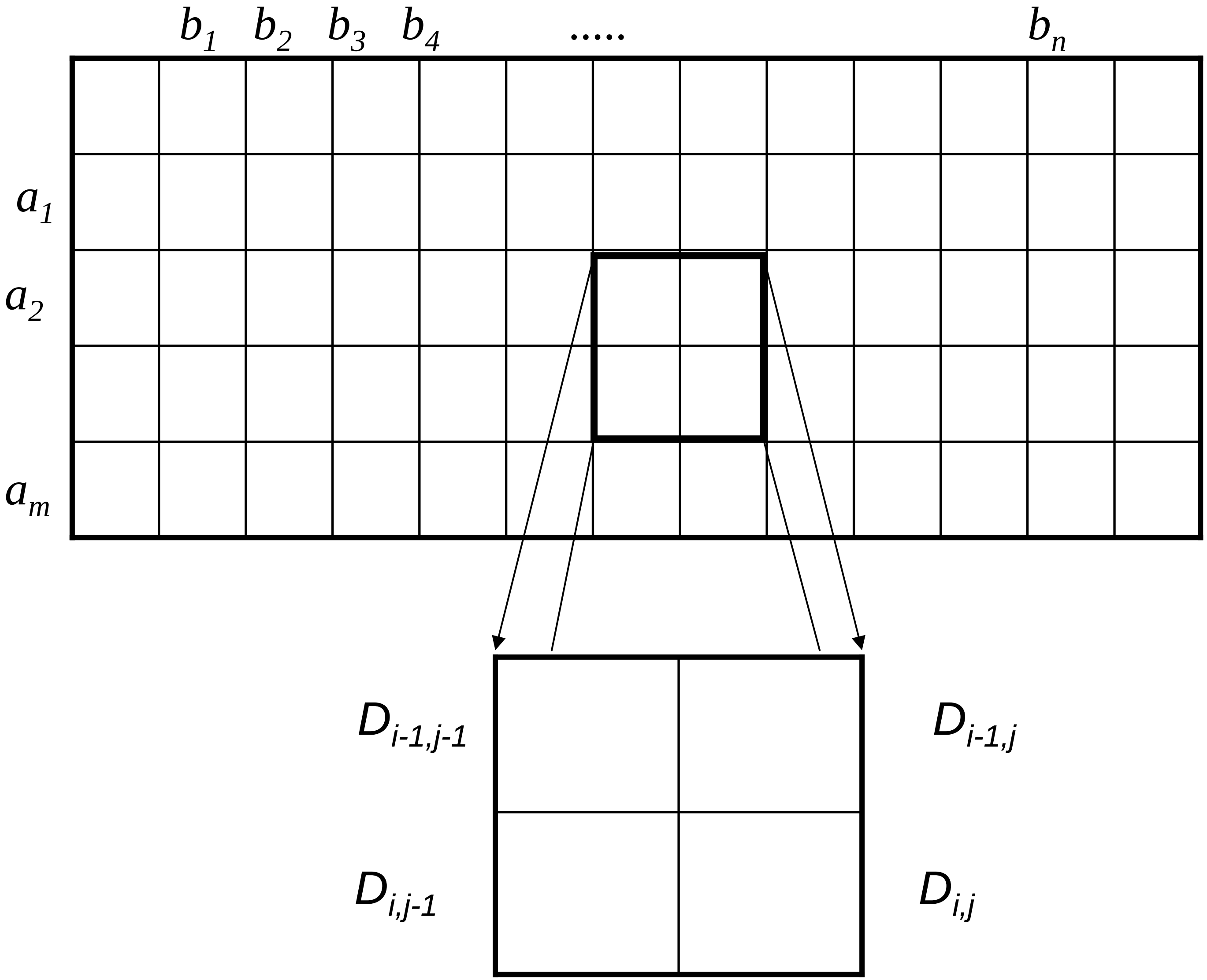
Base cases:

$$D_{0,0} = D(\varepsilon, \varepsilon) = 0$$
$$D_{0,j} = \mathrm{D}(\varepsilon, B_j) = j$$
$$D_{i,0} = D(A_i, \varepsilon) = i$$

Recurrence equation:

$$D_{i,j} = \min \begin{Bmatrix} D_{i-1,j-1} & + & c(a_i, b_j) \\ D_{i-1,j} & + & 1, \\ D_{i,j-1} & + & 1 \end{Bmatrix}$$

Order of computation for the edit distance



Algorithm for the edit distance

Algorithm edit_distance

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

Output: the matrix $D = (D_{ij})$

$D[0,0] := 0$

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

    $D[i,j] := \min(D[i-1,j] + 1,$

      $D[i,j-1] + 1,$

      $D[i-1, j-1] + c(a_i,b_j))$

Example

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

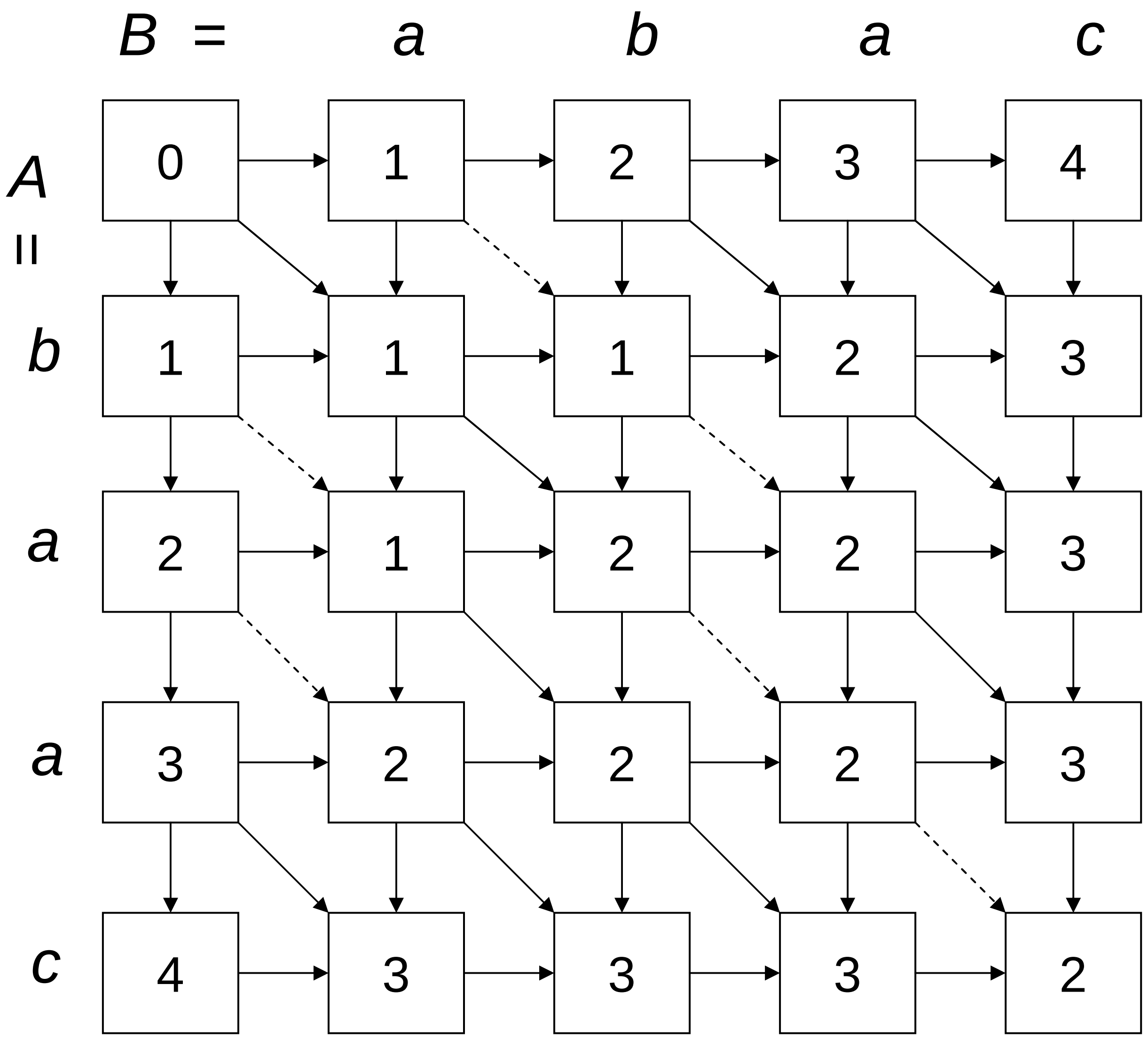
Computation of the edit operations

Algorithm edit_operations (i,j)
Input: matrix D (computed)

if $i = 0$ **and** $j = 0$ **then return**
if $i \neq 0$ **and** $D[i,j] = D[i-1, j] + 1$
  then „delete $a[i]$“
    edit_operations ($i-1, j$)
else if $j \neq 0$ and $D[i,j] = D[i, j-1] + 1$
  then „insert $b[j]$“
    edit_operations ($i, j-1$)
else
  /* $D[i,j] = D[i-1, j-1] + c(a[i], b[j])$ */
    „replace $a[i]$ by $b[j]$ “
    edit_operations ($i-1, j-1$)

Initial call: edit_operations(m,n)

Trace graph of the edit operations



Sub-graph of the edit operations

Trace graph: Overview of all possible traces for the transformation of A into B, directed edges from vertex (i, j) to ($i + 1, j$), ($i, j + 1$) and ($i + 1, j + 1$).
Weights of the edges represent the edit costs.

Costs are monotonic increasing along an optimal path.

Each path with monotonic increasing cost from the upper left corner to the lower right corner represents an optimal trace.

Approximate string matching

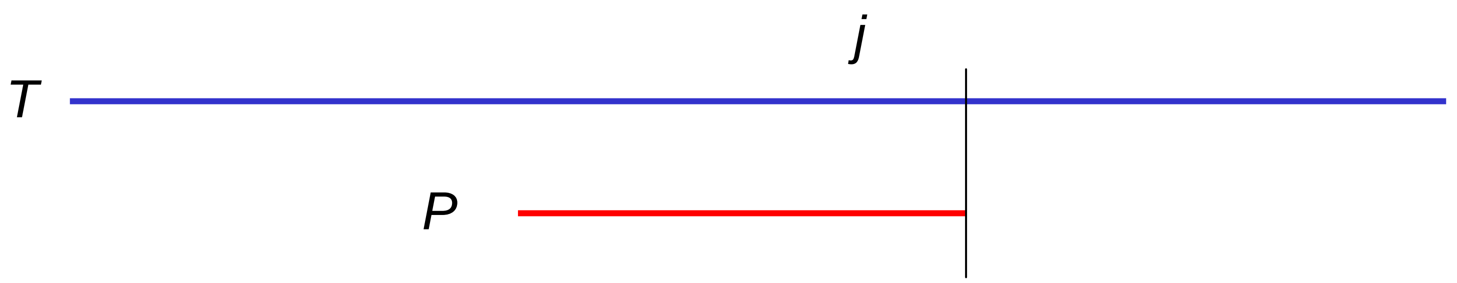
Given: two strings $P = p_1 p_2 \ldots p_m$ (pattern) and

$T = t_1 t_2 \ldots t_n$ (text)

Wanted: an interval $[j', j]$, $1 \leq j' \leq j \leq n$, such that the substring

$T_{j', j} = t_{j'} \ldots t_j$ of T is the one with the greatest similarity to pattern P, i.e. for all other intervals $[k', k]$, $1 \leq k' \leq k \leq n$:

$$D(P, T_{j', j}) \leq D(P, T_{k', k})$$



Approximate string matching

Naïve approach:

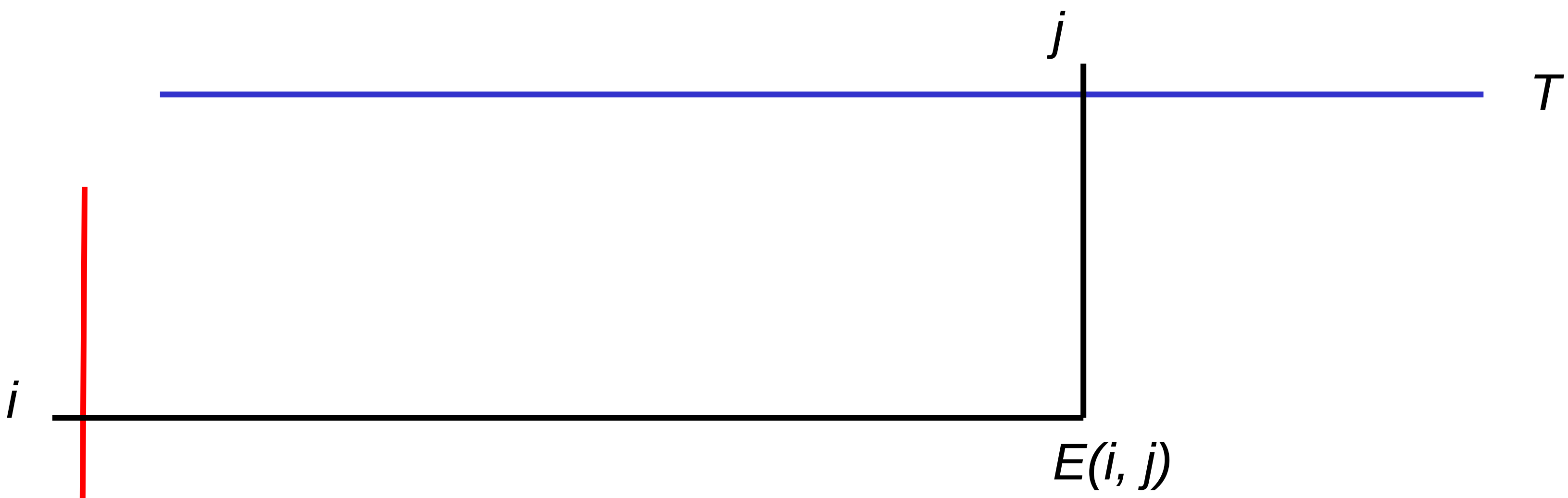
for all $1 \leq j' \leq j \leq n$ **do**

compute $D(P,T_{j',j})$

choose minimum

Approximate string matching

Consider a related problem:



For each text position j and each pattern position i compute the edit distance of the substring $T_{j',j}$ of T ending at j which has the greatest similarity to P_i.

Approximate string matching

Method:

for all $1 \le j \le n$ ***do***

compute j' such that $D(P, T_{j',j})$ is minimal

For $1 \le i \le m$ and $0 \le j \le n$ let:

$$E_{i,j} = \min_{1 \le j' \le j+1} D(P_i, T_{j',j})$$

Optimal trace:

$P_i =$ b a a c a a b c

$T_{j',j} =$ b a c b c a c

Approximate string matching

Recurrence relation:

$$E_{i,j} = \min \begin{Bmatrix} E_{i-1,j-1} + c(p_i, t_j), \\ E_{i-1,j} + 1, \\ E_{i,j-1} + 1 \end{Bmatrix}$$

Remark:

j´ can be completely different for $E_{i-1, j-1}$, $E_{i-1,j}$ and $E_{i, j-1}$.

A subtrace of an optimal trace is an optimal subtrace.

Approximate string matching

Base cases:

$$E_{0,0} = E(\varepsilon, \varepsilon) = 0$$
$$E_{i,0} = \mathrm{E}(P_j, \varepsilon) = i$$

but

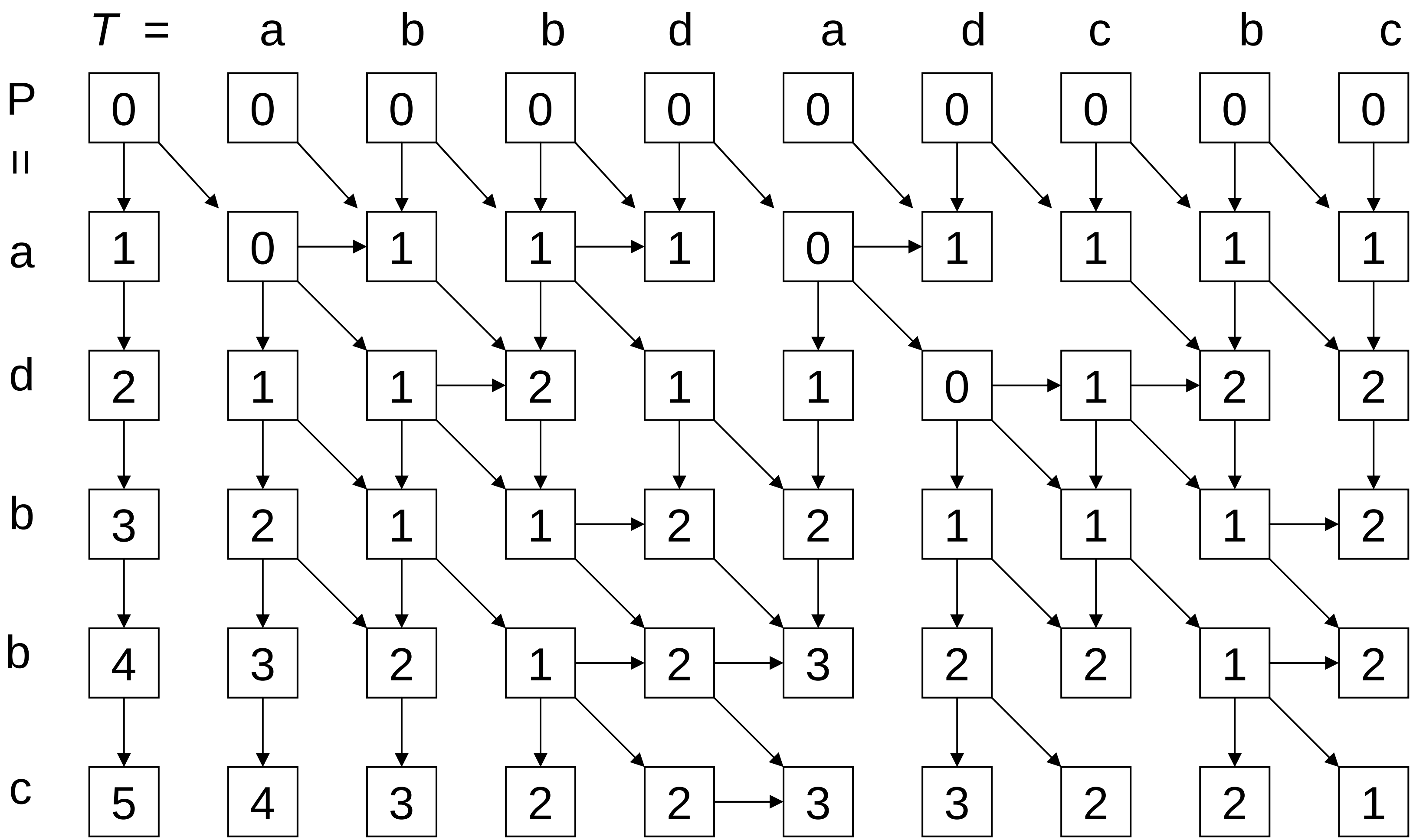
$$E_{0,j} = E(\varepsilon, T_j) = 0$$

Observation:

The optimal edit sequence from P to $T_{j',j}$ does not start with an insertion of $t_{j'}$.

Approximate string matching

Dependency graph



Approximate string matching

Theorem

If there is a path from $E_{0,\, j'-1}$ to $E_{i,\, j}$ in the dependency graph, then $T_{j',\, j}$ is a substring of T ending in j with the greatest similarity to P_i and

$$D(P_i,\ T_{j',j}) = E_{i,\, j}$$