



# Chapter 3
# Dynamic Programming

## Algorithm Theory
## WS 2015/16

Fabian Kuhn

# Dynamic Programming

*„Memoization“* for increasing the efficiency of a recursive solution:

- Only the *first time* a sub-problem is encountered, its solution is computed and then stored in a table. Each subsequent time that the subproblem is encountered, the value stored in the table is simply looked up and returned
  (without repeated computation!).

- *Computing the solution*: For each sub-problem, store how the value is obtained (according to which recursive rule).

# Dynamic Programming

Dynamic programming / memoization can be applied if

- Optimal solution contains optimal solutions to sub-problems (recursive structure)

- Number of sub-problems that need to be considered is small

# Knapsack

- $n$ items $1, \dots, n$, each item has weight $w_i$ and value $v_i$
- Knapsack (bag) of capacity $W$

- Goal: pack items into knapsack such that total weight is at most $W$ and total value is maximized:

$$\max \sum_{i \in S} v_i$$

$$\text{s.t. } S \subseteq \{1, \dots, n\} \text{ and } \sum_{i \in S} w_i \leq W$$

- E.g.: jobs of length $w_i$ and value $v_i$, server available for $W$ time units, try to execute a set of jobs that maximizes the total value

# Recursive Structure?

- Optimal solution: $\mathcal{O}$
- If $n \notin \mathcal{O}$: $\mathrm{OPT}(n) = \mathrm{OPT}(n-1)$
- What if $n \in \mathcal{O}$?
  - Taking $n$ gives value $v_n$
  - But, $n$ also occupies space $w_n$ in the bag (knapsack)
  - There is space for $W - w_n$ total weight left!

$$\mathrm{OPT}(n) = w_n + \text{optimal solution with first } n-1 \text{ items and knapsack of capacity } W - w_n$$

# A More Complicated Recursion

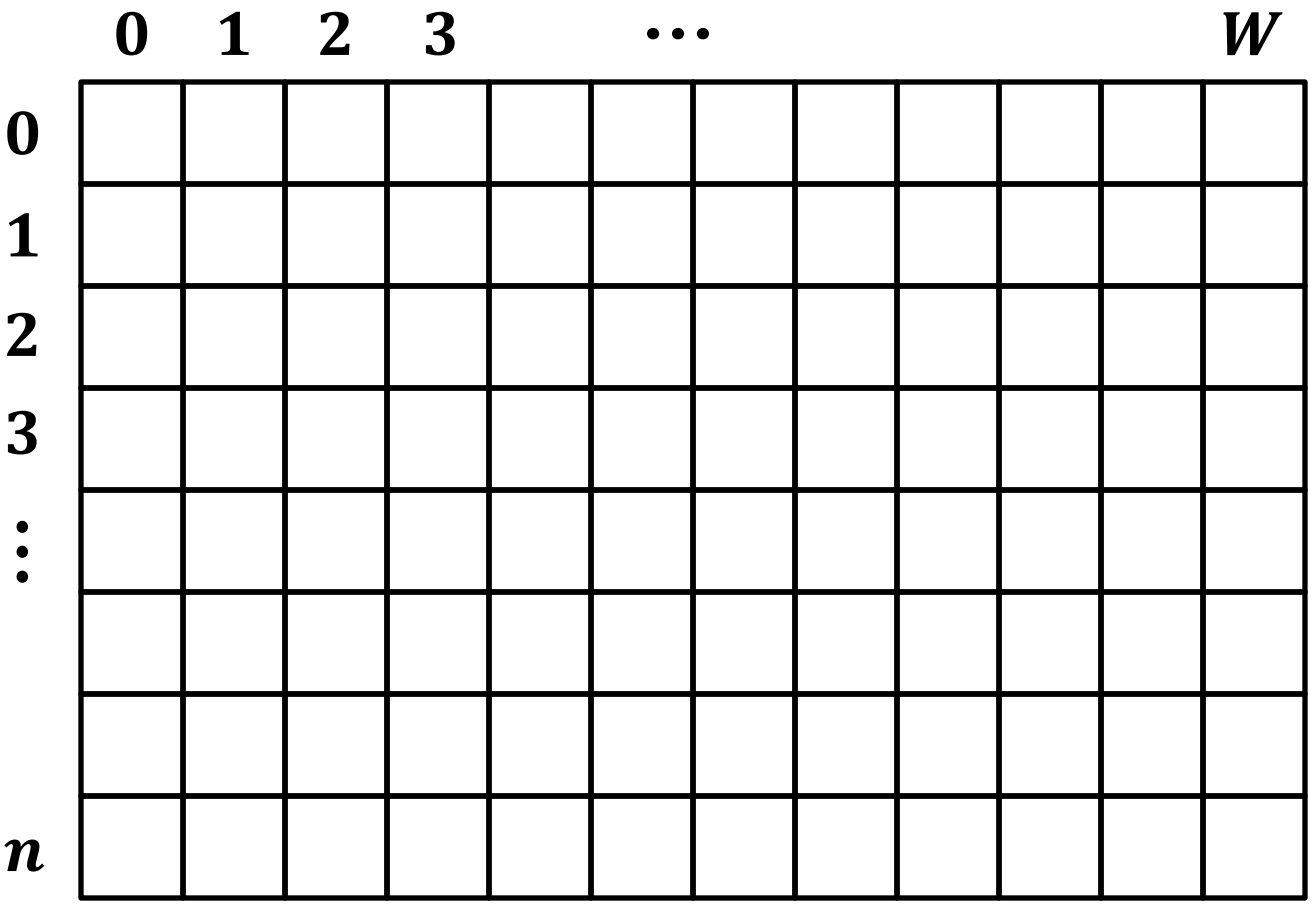
$\mathbf{OPT}(k, x)$: value of optimal solution with items $1, \dots, k$
and knapsack of capacity $x$

**Recursion:**

# Dynamic Programming Algorithm

Set up table for all possible $\mathrm{OPT}(k, x)$-values

- Assume that all weights $w_i$ are integers!




**Row $i$, column $j$:**

$OPT(i, j)$

# Example

- 8 items: $(3,2), (2,4), (4,1), (5,6), (3,3), (4,3), (5,4), (6,6)$
  Knapsack capacity: 12

weight value

- $\boldsymbol{OPT(k,x) = \max\{OPT(k-1,x), OPT(k-1,x-w_k) + v_k\}}$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | |

# Running Time of Knapsack Algorithm

- **Size of table:** $O(n \cdot W)$
- Time per table entry: $O(1)$ → **overall time:** $\boldsymbol{O(nW)}$
- Computing solution (set of items to pick):
  Follow $\leq n$ arrows → $O(n)$ time (after filling table)
- Note: Time depends on $W$ → can be exponential in $n$…
- And it is problematic if weights are not integers.

# String Matching Problems

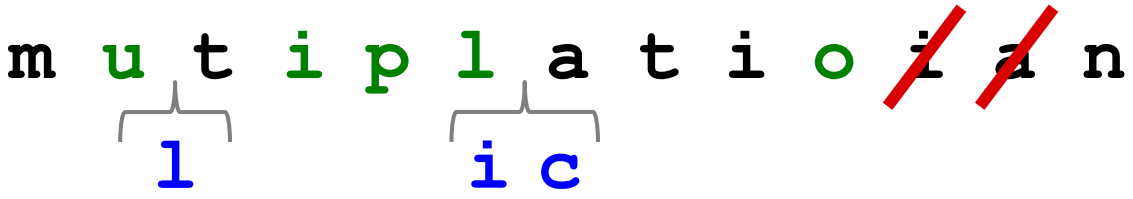
## Edit distance:

- 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:



# 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
```

# Edit Distance – Cost Model

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

# Recursive Structure

- 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   and  rgikacc-
```

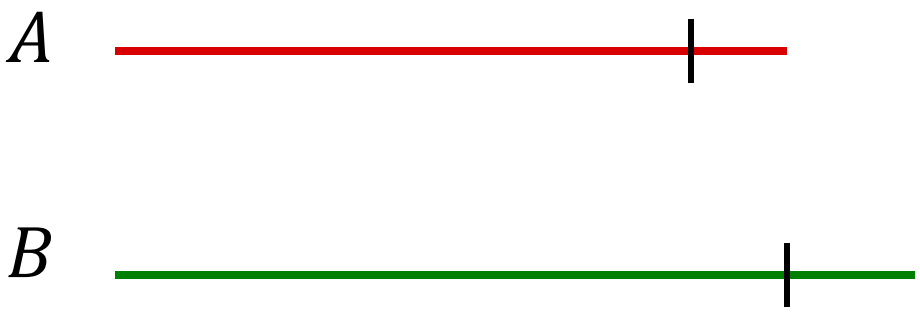
- Edit distance between $A_{1,m} = a_1 \dots a_m$ and $B_{1,n} = b_1 \dots b_n$:

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

# Computation of the Edit Distance

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

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



# Computation of the Edit Distance

Three ways of ending an "alignment" between $A_k$ and $B_\ell$:

1. $a_k$ is replaced by $b_\ell$:

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

2. $a_k$ is deleted:

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

3. $b_\ell$ is inserted:

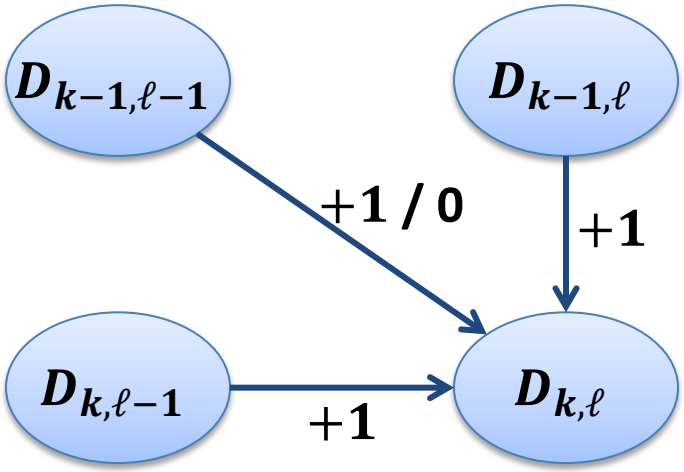
   $$D_{k,\ell} = D_{k,\ell-1} + c(\varepsilon, b_\ell)$$

# Computing the Edit Distance

- 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 $0 \leq i \leq k$, $0 \leq j \leq \ell$:



# Recurrence Relation for the Edit Distance

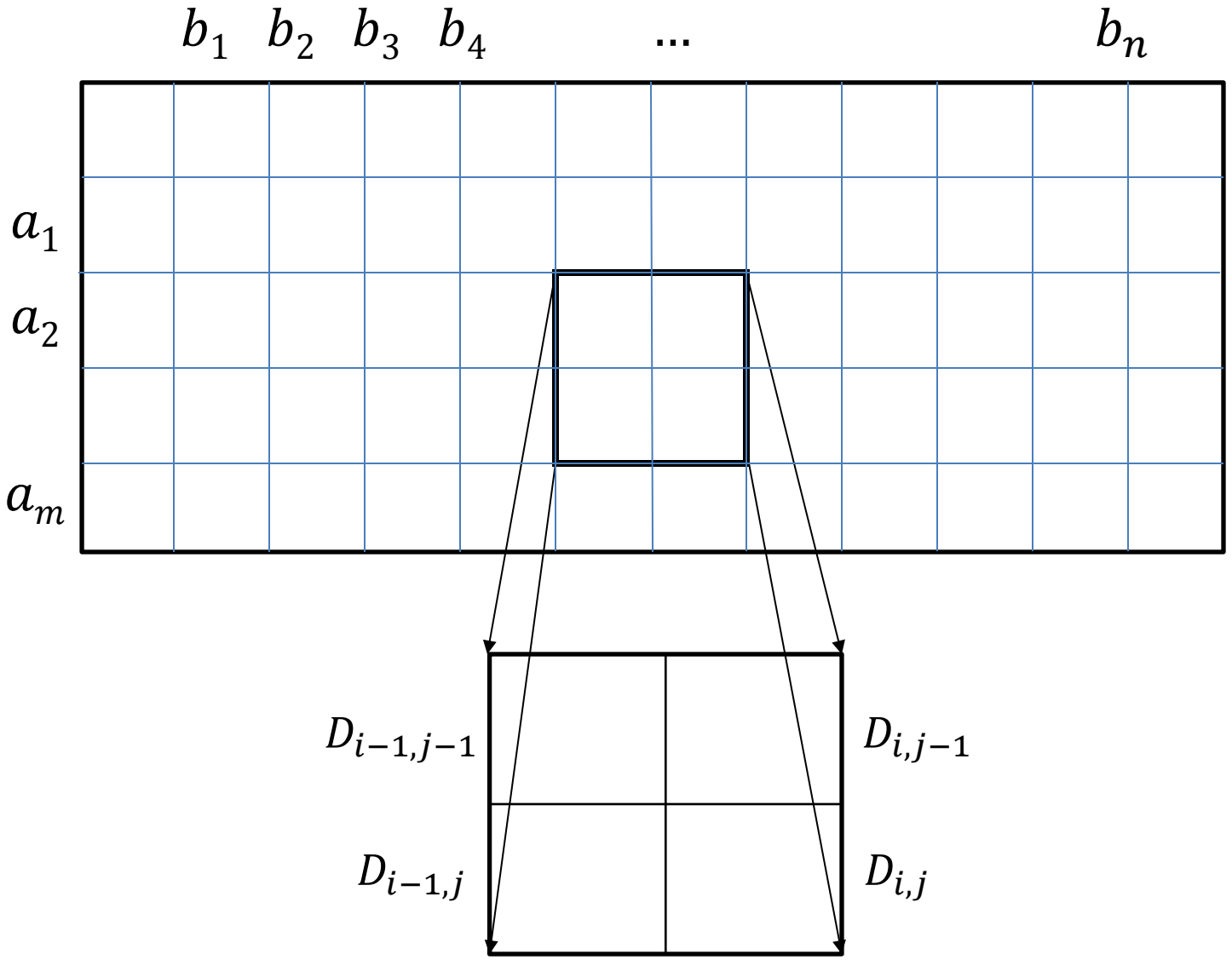
**Base cases:**

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

**Recurrence relation:**

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

# 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 $D = (D_{ij})$

$1\ D[0,0] \coloneqq 0;$

2 **for** $i \coloneqq 1$ **to** $m$ **do** $D[i,0] \coloneqq i;$

3 **for** $j \coloneqq 1$ **to** $n$ **do** $D[0,j] \coloneqq j;$

4 **for** $i \coloneqq 1$ **to** $m$ **do**

5 **for** $j \coloneqq 1$ **to** $n$ **do**

6 $$D[i,j] \coloneqq \min \begin{Bmatrix} D[i-1,j] & +1 \\ D[i,j-1] & +1 \\ D[i-1,j-1] & + c(a_i, b_j) \end{Bmatrix};$$