Algorithmen und Datenstrukturen

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Felix Kuhr (Übungen)

sowie viele Tutoren

Gemeinsame Teilsequenz (Common Subsequence)

- Teilsequenz einer Zeichenkette: Zeichenkette mit 0 oder mehr ausgelassenen Zeichen
- Gemeinsame Teilsequenz von zwei Zeichenketten
  - Teilsequenz von beiden Zeichenketten
- Beispiel:
  - x = <A B C B D A B>, y = <B D C A B A>
  - <B C> und <A A> sind gemeinsame Teilsequenzen von x und y

Längste gemeinsame Teilsequenz

Gegeben sei ein Alphabet Σ und zwei Sequenzen x[1..m] und y[1..n] in denen jeder Buchstabe aus Σ vorkommt.
Aufgabe: Bestimme eine längste gemeinsame Teilsequenz (longest common subsequence, LCS)

- NB: „eine“ längste, nicht die längste

x: A B C B D A B

y: B D C A B A

$\left.\right\}$ BCBA = LCS(x, y)

Funktionale Notation
(aber keine Funktion)

Brute-Force-Algorithmus?

Prüfe jede Teilsequenz von x[1..m] und prüfe, ob es sich auch um eine Teilsequenz von y[1..n] handelt

Analyse:

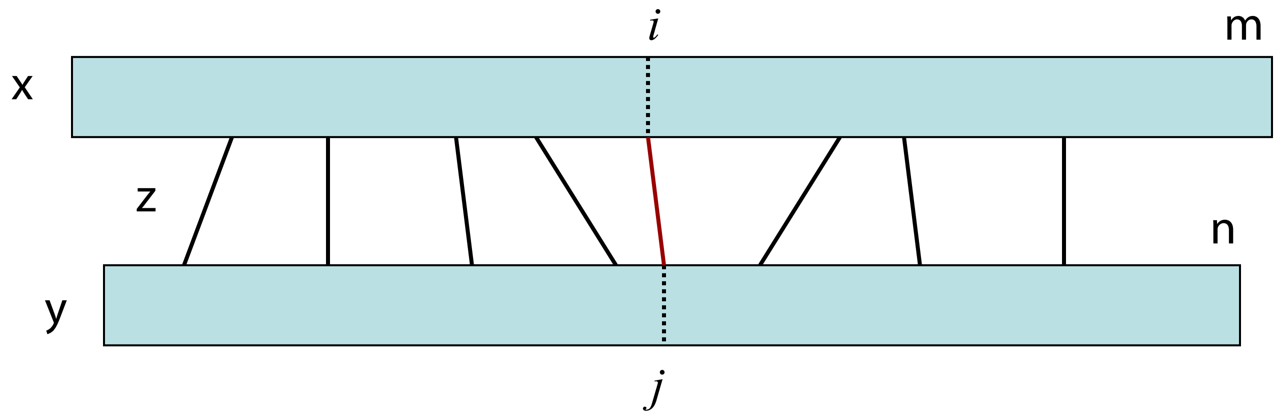
- 2^m Teilsequenzen in x vorhanden (jeder Bitvektor der Länge m bestimmt unterschiedliche Teilsequenz)
- Die Zeitfunktion dieses Algorithmus wäre in $\Theta(2^m)$, der Algorithmus also exponentiell (--> nicht praxistauglich)

Auf dem Weg zu einer besseren Strategie:

- Ansatz der dynamischen Programmierung
  - Bestimme opt. Substruktur, überlappende Teilprobleme
- Zunächst: Bestimmung der Länge eines LCS, dann Bestimmung eines LCS

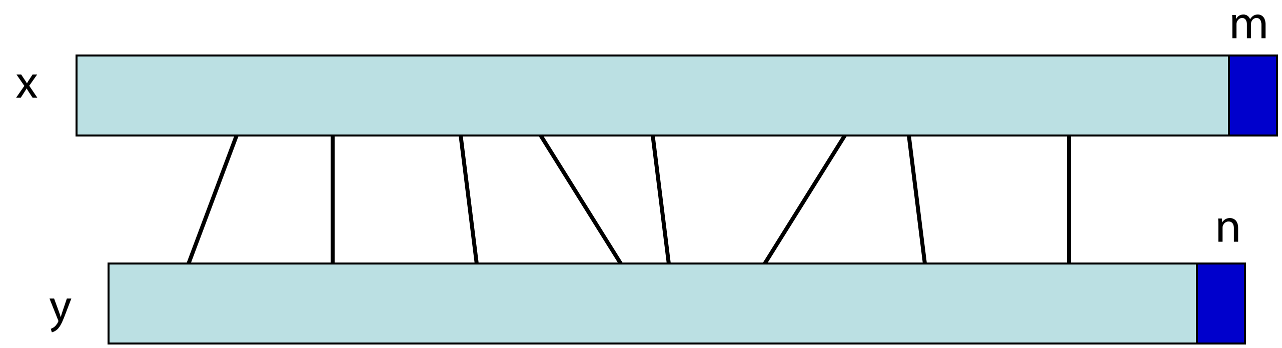
Optimale Substruktur

- Falls z=LCS(x, y), dann gilt für jeden Präfix u:
  uz ist ein LCS von ux und uy



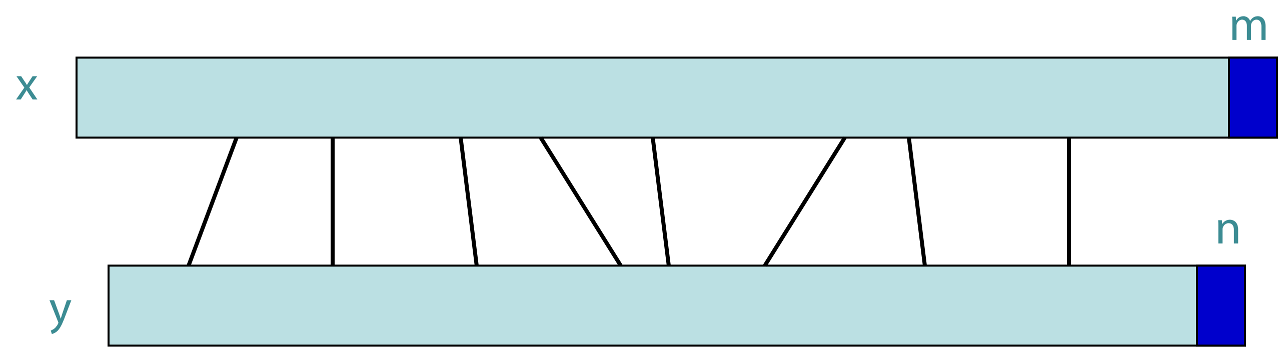
- Teilprobleme: Finde LCS von Präfixen von x und y

Rekursiver Ansatz



- Fall 1: x[m]=y[n]: Es gibt **einen** optimalen LCS in dem x[m] mit y[n] abgeglichen wird → Finde LCS (x[1..m-1], y[1..n-1])
- Fall 2: x[m]≠ y[n]: Einer könnte in LCS sein
  - Fall 2.1: x[m] nicht in LCS → Finde LCS (x[1..m-1], y[1..n])
  - Fall 2.2: y[n] nicht in LCS → Finde LCS (x[1..m], y[1..n-1])

Rekursiver Ansatz



- Fall 1: $x[m]=y[n]$
  - $LCS(x, y) = LCS(x[1..m-1], y[1..n-1]) \,||\, x[m]$

Reduziere beide Sequenzen um 1 Zeichen

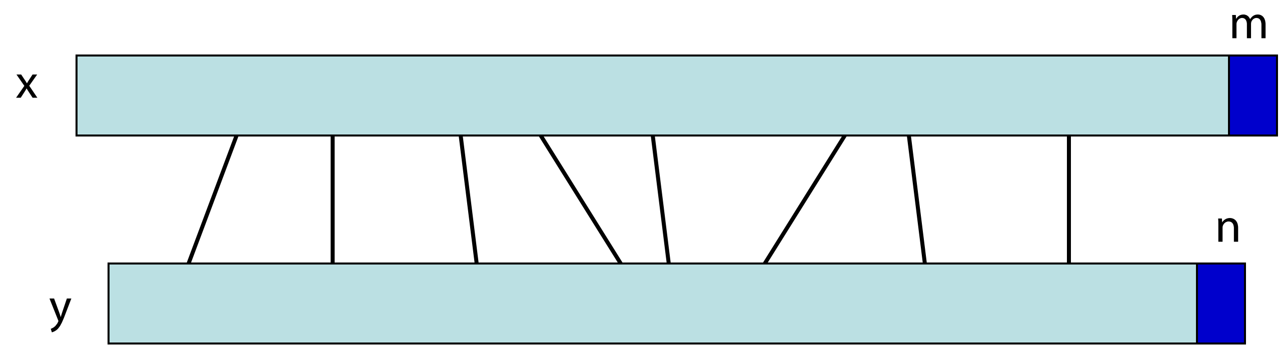
Konkatenierung

- Fall 2: $x[m] \neq y[n]$
  - $LCS(x, y) = LCS(x[1..m-1], y[1..n])$ oder

    $LCS(x[1..m], y[1..n-1])$

Reduziere eine der Sequenzen um 1 Zeichen

Finden der Länge eines LCS



- Sei c[i, j] die Länge von LCS(x[1..i], y[1..j])

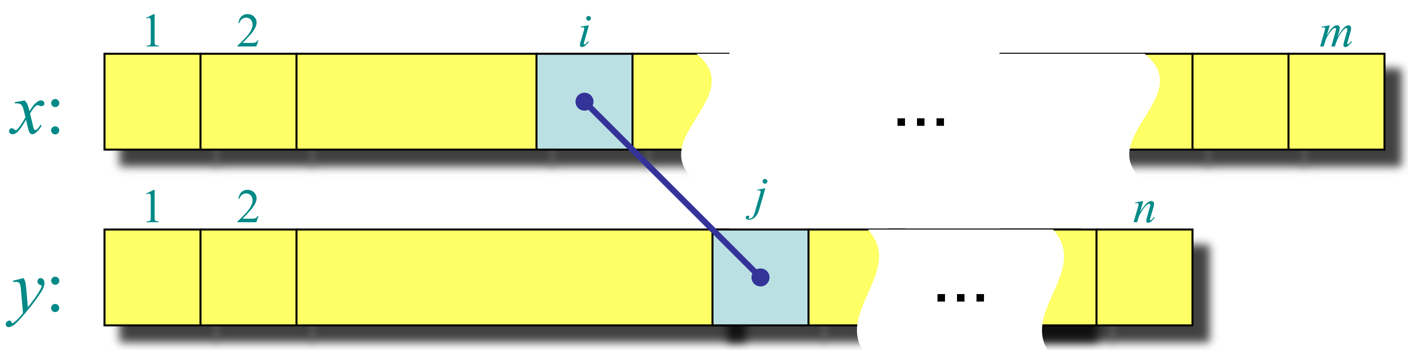
  dann ist c[m, n] die Länge von LCS(x, y)
- Falls x[m] = y[n] dann

  c[m, n] = c[m-1, n-1] + 1
- Falls x[m] ≠ y[n] dann

  c[m, n] = max({ c[m-1, n], c[m, n-1] })

Generalisierung: Rekursive Formulierung

$$c[i, j] = \begin{cases} c[i-1, j-1] + 1 & \text{falls } x[i] = y[j], \\ \max\{c[i-1, j], c[i, j-1]\} & \text{sonst} \end{cases}$$



Rekursiver Algorithmus für LCS

procedure LCS(x, y, i, j):

  if $x[i] = y[j]$

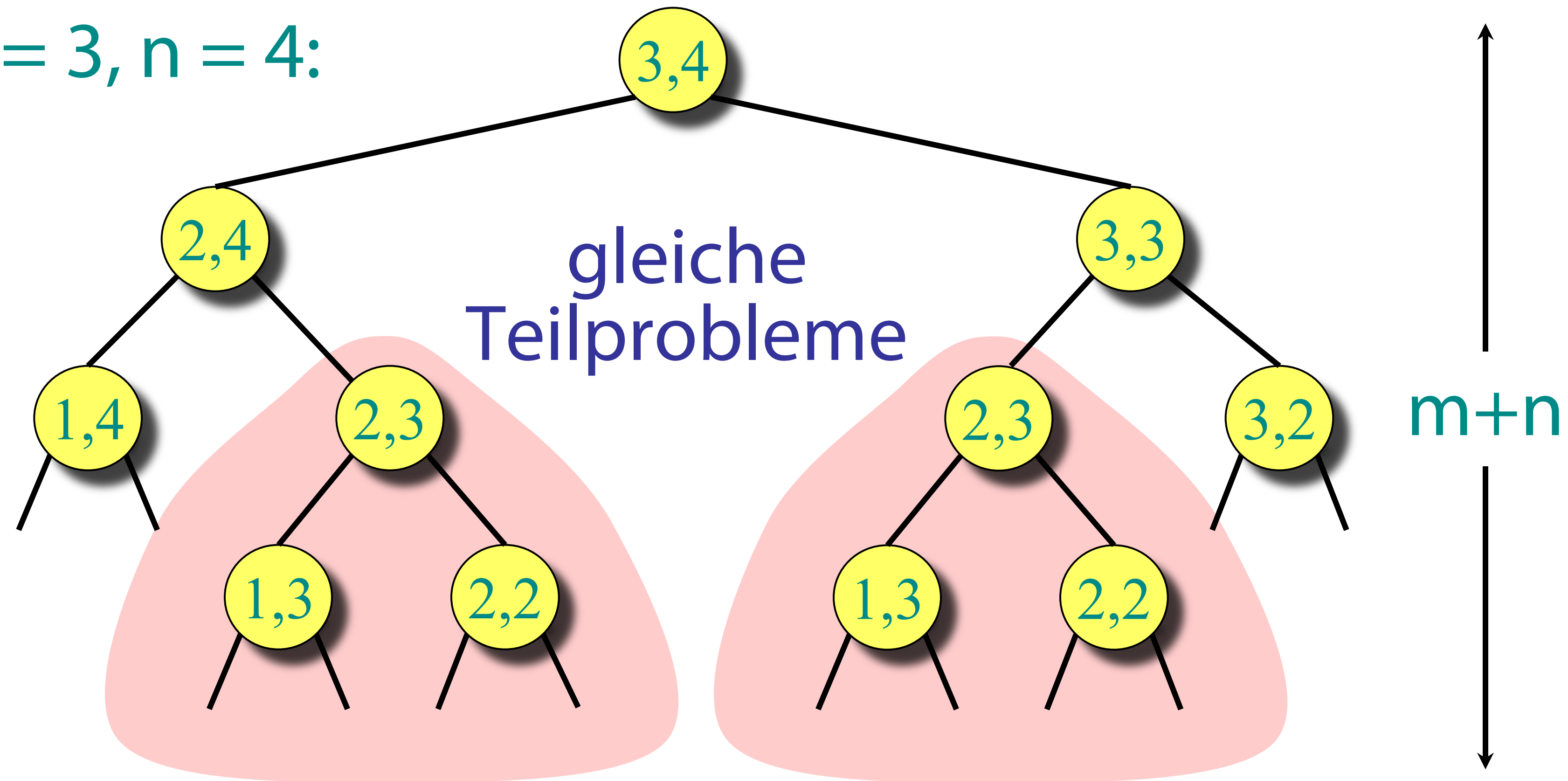
    then $c[i, j] :=$ LCS$(x, y, i-1, j-1) + 1$

    else $c[i, j] := \max(\{$ LCS$(x, y, i-1, j)$,
                    LCS$(x, y, i, j-1)\})$

Schlimmster Fall: $x[i] \neq y[j]$
dann zwei Subprobleme, jedes mit nur einer Dekrementierung (um 1)

Rekursionsbaum

$m = 3, n = 4$:

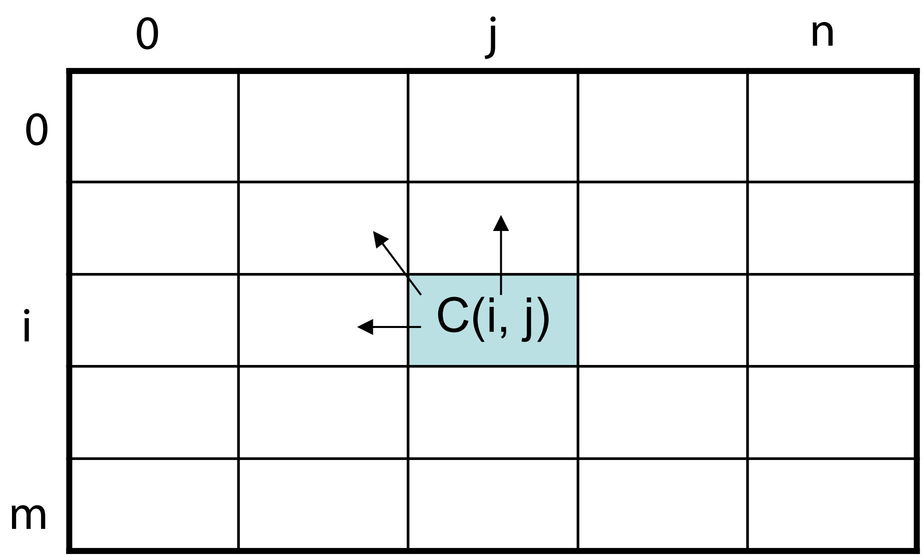


Höhe $= m + n \Rightarrow$ potentiell exponentieller Aufwand mit wiederholter Lösung gleicher Teilprobleme!

Dynamische Programmierung

- Finde richtige Anordnung der Teilprobleme
- Gesamtanzahl der Teilprobleme: $m \cdot n$

$$c[i, j] = \begin{cases} c[i-1, j-1] + 1 & \text{falls } x[i] = y[j], \\ \max(\{c[i-1, j], c[i, j-1]\}) & \text{sonst.} \end{cases}$$



Algorithmus LCS

```
Function LCS-length(X, Y):
 m := length(X); n  := length(Y)
 c: Array [0..m, 0..n] of Integer
 for i from 1 to m do c[i,0] := 0       // Sonderfall: Y[0]
 for j from 1 to n do c[0,j] := 0       // Sonderfall: X[0]
 for i from 1 to m                      // für alle X[i]
   for j from 1 to n                    // für alle Y[j]
     if X[i] = Y[j] then
       c[i,j] := c[i-1,j-1] + 1
     else c[i,j] := max( { c[i-1,j], c[i,j-1] } )
 return c
```

LCS Anwendungsbeispiel

- X = ABCB
- Y = BDCAB

Was ist der LCS von X und Y?

LCS(X, Y) = BCB

X = A **B** **C** **B**

Y = **B** D **C** A **B**

LCS Beispiel (0)

ABCB
BDCAB

i	j	0	1	2	3	4	5
		Y[j]	**B**	**D**	**C**	**A**	**B**
0	X[i]						
1	**A**						
2	**B**						
3	**C**						
4	**B**						

X = ABCB; m = |X| = 4
Y = BDCAB; n = |Y| = 5
Alloziere Array c[5,6]

LCS Beispiel (1)

ABCB
BDCAB

		j	0	1	2	3	4	5
i			Y[j]	**B**	**D**	**C**	**A**	**B**
0	X[i]		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**					
2	**B**		**0**					
3	**C**		**0**					
4	**B**		**0**					

for i from 1 to m   c[i,0] := 0
for j from 1 to n   c[0,j] := 0

LCS Beispiel (2)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Y[j]	B	D	C	A	B
0	X[i]	**0**	**0**	**0**	**0**	**0**	**0**
1	A	**0**	**0**				
2	B	**0**					
3	C	**0**					
4	B	**0**					

if $X_i = Y_j$ then
$c[i,j] := c[i-1,j-1] + 1$
else $c[i,j] := \max(c[i-1,j], c[i,j-1])$

LCS Beispiel (3)

ABCB

BDCAB

	j	0	1	2	3	4	5
i		Y[j]	**B**	**D**	**C**	**A**	**B**
0	X[i]	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**		
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

if $X_i = Y_j$ then

$c[i,j] := c[i-1,j-1] + 1$

else $c[i,j] := \max(c[i-1,j], c[i,j-1])$

LCS Beispiel (4)

ABCB

BDCAB

	j	0	1	2	3	4	5
i		Y[j]	B	D	C	A	B
0	X[i]	0	0	0	0	0	0
1	A	0	0	0	0	1	
2	B	0					
3	C	0					
4	B	0					

if $X_i = Y_j$ then
c[i,j] := c[i-1,j-1] + 1
else c[i,j] := max(c[i-1,j], c[i,j-1])

LCS Beispiel (5)

ABCB

BDCAB

	j	0	1	2	3	4	5
i		Y[j]	B	D	C	A	B
0	X[i]	**0**	**0**	**0**	**0**	**0**	**0**
1	A	**0**	**0**	**0**	**0**	**1** →	**1**
2	B	**0**					
3	C	**0**					
4	B	**0**					

if $X_i = Y_j$ then

$c[i,j] := c[i-1,j-1] + 1$

else $c[i,j] := \max(c[i-1,j], c[i,j-1])$

LCS Beispiel (6)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Y[j]	**B**	**D**	**C**	**A**	**B**
0	X[i]	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	1				
3	**C**	**0**					
4	**B**	**0**					

if $X_i = Y_j$ then
$c[i,j] := c[i-1,j-1] + 1$
else $c[i,j] := \max(c[i-1,j], c[i,j-1])$

LCS Beispiel (7)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Y[j]	B	D	C	A	B
0	X[i]	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	
3	C	0					
4	B	0					

if $X_i = Y_j$ then
$c[i,j] := c[i-1,j-1] + 1$
else $c[i,j] := \max(c[i-1,j], c[i,j-1])$

LCS Beispiel (8)

ABCB
BDCAB

		j	0	1	2	3	4	5
i			Y[j]	B	D	C	A	B
0	X[i]		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0					
4	B		0					

if $X_i = Y_j$ then
$c[i,j] := c[i-1,j-1] + 1$
else $c[i,j] := \max(c[i-1,j], c[i,j-1])$

LCS Beispiel (9)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Y[j]	B	D	C	A	B
0	X[i]	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	2
3	C	0	1	1			
4	B	0					

if $X_i = Y_j$ then
$c[i,j] := c[i-1,j-1] + 1$
else $c[i,j] := \max(c[i-1,j], c[i,j-1])$

LCS Beispiel (10)

ABCB
BDCAB

		j	0	1	2	3	4	5
i			Y[j]	**B**	**D**	**C**	**A**	**B**
0	X[i]		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**	**0**	**0**	**0**	**1**	**1**
2	**B**		**0**	**1**	**1**	**1**	**1**	**2**
3	**C**		**0**	**1**	**1**	**2**		
4	**B**		**0**					

if $X_i = Y_j$ then
$c[i,j] := c[i-1,j-1] + 1$
else $c[i,j] := \max(c[i-1,j], c[i,j-1])$

LCS Beispiel (11)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Y[j]	**B**	**D**	**C**	**A**	**B**
0	X[i]	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**B**	**0**					

if $X_i = Y_j$ then
$c[i,j] := c[i-1,j-1] + 1$
else $c[i,j] := \max(c[i-1,j], c[i,j-1])$

LCS Beispiel (12)

ABCB

BDCAB

		j	0	1	2	3	4	5
i			Y[j]	B	D	C	A	B
0	X[i]		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2	2	2
4	B		0	1				

if $X_i = Y_j$ then
$c[i,j] := c[i-1,j-1] + 1$
else $c[i,j] := \max(c[i-1,j], c[i,j-1])$

LCS Beispiel (13)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Y[j]	B	D	C	A	B
0	X[i]	**0**	**0**	**0**	**0**	**0**	**0**
1	A	**0**	**0**	**0**	**0**	**1**	**1**
2	B	**0**	**1**	**1**	**1**	**1**	**2**
3	C	**0**	**1**	**1**	**2**	**2**	**2**
4	B	**0**	**1**	**1**	**2**	**2**	

if $X_i = Y_j$ then
$c[i,j] := c[i-1,j-1] + 1$
else $c[i,j] := \max(c[i-1,j], c[i,j-1])$

LCS Beispiel (14)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			Y[j]	**B**	**D**	**C**	**A**	**B**
0	X[i]		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**	**0**	**0**	**0**	**1**	**1**
2	**B**		**0**	**1**	**1**	**1**	**1**	**2**
3	**C**		**0**	**1**	**1**	**2**	**2**	**2**
4	**B**		**0**	**1**	**1**	**2**	**2**	3

if $X_i = Y_j$ then
$c[i,j] := c[i-1,j-1] + 1$
else $c[i,j] := \max(c[i-1,j], c[i,j-1])$

LCS-Algorithmus: Analyse

- Der LCS-Algorithmus bestimmt die Werte des Feldes c[m,n]
- Laufzeit?

O(m·n)

Jedes c[i,j] wird in konstanter Zeit berechnet, und es gibt m·n Elemente in dem Feld

Wie findet man den tatsächlichen LCS?

- Für c[i, j] ist bekannt wie es hergeleitet wurde:

$$c[i,j] = \begin{cases} c[i-1, j-1]+1 & \text{falls } x[i] = y[j], \\ \max(c[i,j-1], c[i-1,j]) & \text{sonst} \end{cases}$$

- Match liegt nur vor, wenn erste Gleichung verwendet
- Beginnend von c[m,n] und rückwärtslaufend, speichere x[i] wenn c[i,j] = c[i-1, j-1]+1.

2	2
2	3

Zum Beispiel hier
c[i,j] = c[i-1,j-1] +1 = 2+1=3

Finde LCS   Zeit für Rückverfolgung: O(m+n)

	j	0	1	2	3	4	5
i		Y[j]	B	D	C	A	B
0	X[i]	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	2
3	C	0	1	1	2	2	2
4	B	0	1	1	2	2	3

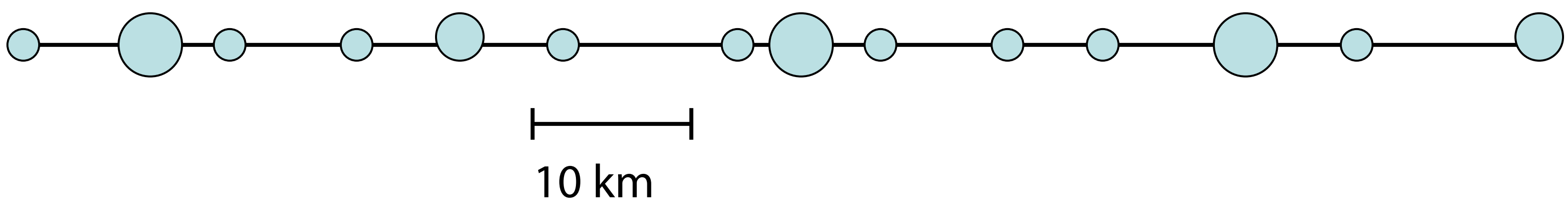
LCS (umgekehrt): B C B

LCS (richtig dargestellt): **B C B**

(ein Palindrom)

Dynamische Programmierung: Restaurant-Platzierung

- Städte $t_1, t_2, \ldots, t_n$ an der Autobahn
- Restaurants in t_i haben von der Größe der Stadt abhängigen geschätzten jährlichen Profit p_i
- Restaurants mit Mindestabstand von 10 km aufgrund von Vorgaben
- Ziel: Maximierung des Profits – großer Bonus

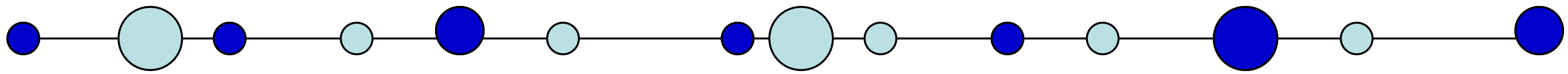


Brute-Force-Ansatz

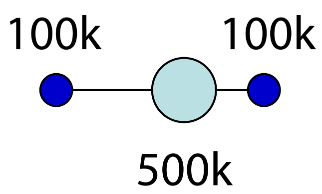
- Jede Stadt wird entweder gewählt oder nicht
- Testen der Bedingungen für 2^n Teilmengen
- Eliminiere Teilmengen, die Einschränkungen nicht erfüllen
- Berechne Gesamtprofit für jede übrigbleibende Teilmenge
- Wähle Teilmenge von Städten mit größtem Profit

- $\Theta(n \cdot 2^n)$

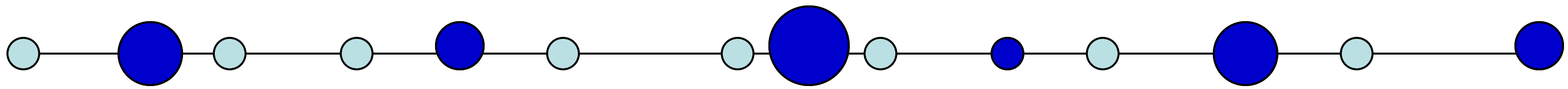
Natürlich-gierige Strategie 1



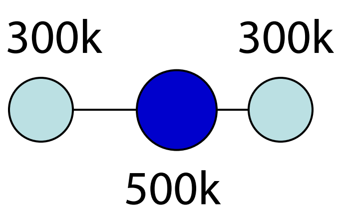
- Nehme erste Stadt. Dann nächste Stadt mit Entfernung >= 10 km
- Können Sie ein Beispiel angeben, bei dem nicht die richtige (beste) Lösung bestimmt wird?



Natürlich-gierige Strategie 2

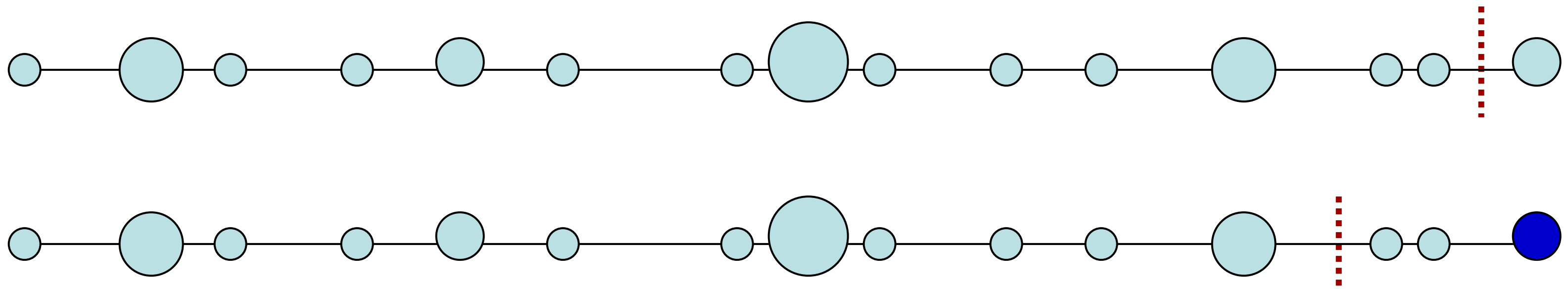


- Nehme Stadt mit höchstem Profit und dann die nächsten, die nicht <10 km von der vorher gewählten Stadt liegen
- Können Sie ein Beispiel angeben, bei dem nicht die richtige (beste) Lösung bestimmt wird?



Formulierung über dynamische Programmierung

- Nehmen wir an, die optimale Lösung sei gefunden
- Entweder enthält sie t_n oder nicht
- Fall 1: t_n nicht enthalten
  - Beste Lösung identisch zur besten Lösung von $t_1, \ldots, t_{n-1}$
- Fall 2: t_n enthalten
  - Beste Lösung ist p_n + beste Lösung für $t_1, \ldots, t_j$, wobei $j < n$ der größte Index ist, so dass $dist(t_j, t_n) \geq 10$



Formulierung als Rekurrenz

- Sei S(i) der Gesamtprofit der optimalen Lösung, wenn die ersten i Städte betrachtet, aber nicht notwendigerweise ausgewählt wurden
  - S(n) ist die optimale Lösung für das Gesamtproblem

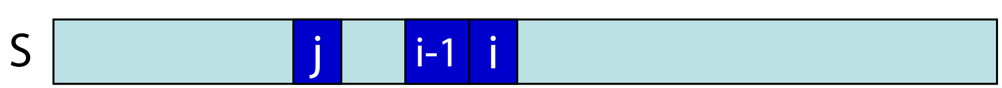
$$S(n) = \max \begin{cases} S(n-1) & \\ S(j) + p_n & j < n \;\&\; \text{dist}\,(t_j, t_n) \geq 10,\ j \text{ maximal} \end{cases}$$

$$\Big\downarrow \text{ Generalisiere}$$

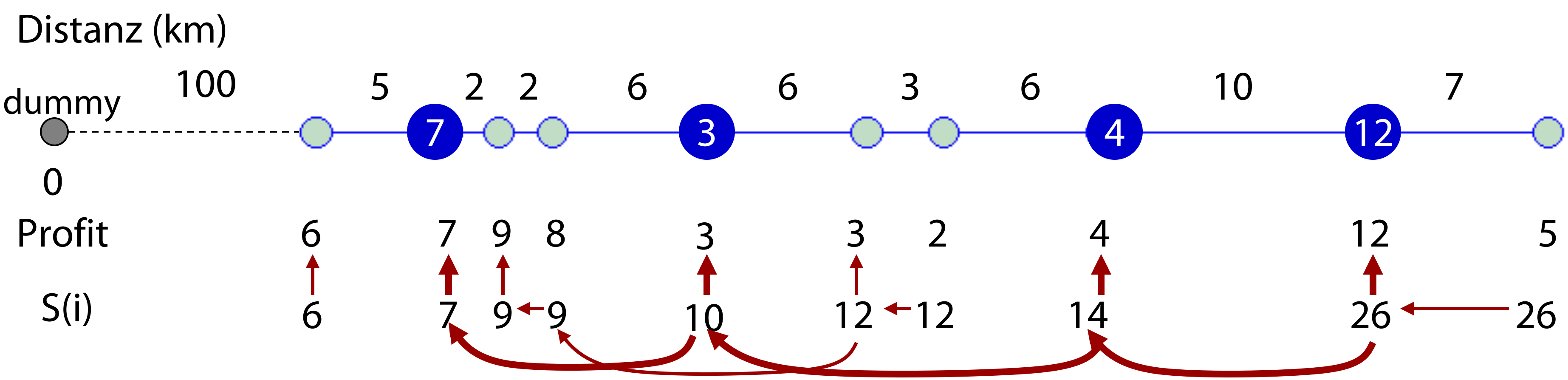
$$S(i) = \max \begin{cases} S(i-1) & \\ S(j) + p_i & j < i \;\&\; \text{dist}\,(t_j, t_i) \geq 10,\ j \text{ maximal} \end{cases}$$

Anzahl der Teilprobleme: n.   Grenzfall: $S(0) = 0$.

Abhängigkeiten:



Beispiel



Optimal: 26

$$S(i) = \max \begin{cases} S(i-1) & \\ S(j) + p_i & j < i \;\&\; \text{dist}(t_j, t_i) \geq 10 \end{cases}$$

- Natürlich-gierig 1: 6 + 3 + 4 + 12 = 25
- Natürlich-gierig 2: 12 + 9 + 4 = 25

Aufwandsanalyse

- Zeit: $\Theta(nk)$, wobei k die maximale Anzahl der Städte innerhalb von 10km nach links zu jeder Stadt ist
  - Im schlimmsten Fall $\Theta(n^2)$
  - Kann durch Vorverarbeitung verbessert werden zu $\Theta(n)$
- Speicher: $\Theta(n)$