20. Dynamische Programmierung

Gierige Algorithmen:

- Berechne Lösung schrittweise
- In jedem Schritt mache lokal optimale Wahl

Anwendbar:

- Wenn optimale Lösung durch lokal optimale Wahl zu einer optimalen Lösung erweiterbar ist.

Algorithmen:

- Scheduling Probleme
- Optimale Präfix-Kodierung (Huffman Codes)

20. Dynamische Programmierung

Divide&Conquer:

- Teile Problem in Teilprobleme auf.
- Kombiniere optimale Lösungen der Teilprobleme zu einer optimalen Lösung für das Ausgangsproblem.

Anwendbar:

- Wenn Problem in Teilprobleme zerlegbar ist, deren Lösungen zu einer Lösung des Gesamtproblems kombinierbar sind.

Algorithmen:

- Mergesort
- Strassens Methode der Matrixmultiplikation

20. Dynamische Programmierung

Divide&Conquer:

- Teile Problem in Teilprobleme auf.
- Kombiniere optimale Lösungen der Teilprobleme zu einer optimalen Lösung für das Ausgangsproblem.

Anwendbar:

- Wenn Problem in Teilprobleme zerlegbar ist, deren Lösungen zu einer Lösung des Gesamtproblems kombinierbar sind.

Problem:

- Es kann bei rekursiven Aufrufen vorkommen, dass Teilprobleme mehrfach gelöst werden!

20. Dynamische Programmierung

Divide&Conquer:

- Teile Problem in Teilprobleme auf.
- Kombiniere optimale Lösungen der Teilprobleme zu einer optimalen Lösung für das Ausgangsproblem.

Anwendbar:

- Wenn Problem in Teilprobleme zerlegbar ist, deren Lösungen zu einer Lösung des Gesamtproblems kombinierbar sind.

Lösung: dynamische Programmierung

- Speichern einmal berechneter Lösungen in einer Tabelle

Dynamische Programmierung

Typische Anwendung für dynamisches Programmieren: *Optimierungsprobleme*

Eine optimale Lösung für das Ausgangsproblem setzt sich aus *optimalen* Lösungen für kleinere Probleme zusammen.

Definition:

- Seien $X=(x_1,\ldots,x_m)$ und $Y=(y_1,\ldots,y_n)$ zwei Folgen, wobei $x_i, y_j \in A$ für ein endliches Alphabet A.
- Dann heißt Y Teilfolge von X, wenn es aufsteigend sortierte Indizes $i_1,\ldots,i_n$ gibt mit $x_{i_j} = y_j$ für $j = 1,\ldots,n$.

Dynamische Programmierung
Längste gemeinsame Teilfolge

Definition:

- Seien $X=(x_1,\dots,x_m)$ und $Y=(y_1,\dots,y_n)$ zwei Folgen, wobei x_i, $y_j \in A$ für ein endliches Alphabet A.
- Dann heißt Y Teilfolge von X, wenn es aufsteigend sortierte Indizes $i_1,\dots,i_n$ gibt mit $x_{i_j} = y_j$ für $j = 1,\dots,n$.

Beispiel:

Folge Y

B	C	A	C

Folge X

A	B	A	C	A	B	C

Dynamische Programmierung
Längste gemeinsame Teilfolge

Definition:

- Seien $X=(x_1,...,x_m)$ und $Y=(y_1,...,y_n)$ zwei Folgen, wobei $x_i, y_j \in A$ für ein endliches Alphabet A.
- Dann heißt Y Teilfolge von X, wenn es aufsteigend sortierte Indizes $i_1,...,i_n$ gibt mit $x_{i_j} = y_j$ für $j = 1,...,n$.

Beispiel:

Folge Y	B	C	A	C			
Folge X	A	B	A	C	A	B	C

Dynamische Programmierung
Längste gemeinsame Teilfolge

Definition:

- Seien $X=(x_1,\dots,x_m)$ und $Y=(y_1,\dots,y_n)$ zwei Folgen, wobei $x_i, y_j \in A$ für ein endliches Alphabet A.
- Dann heißt Y Teilfolge von X, wenn es aufsteigend sortierte Indizes $i_1,\dots,i_n$ gibt mit $x_{i_j} = y_j$ für $j = 1,\dots,n$.

Beispiel:

Folge Y

B	C	A	C

Folge X

A	B	A	C	A	B	C

Dynamische Programmierung
Längste gemeinsame Teilfolge

Definition:

- Seien $X=(x_1,\dots,x_m)$ und $Y=(y_1,\dots,y_n)$ zwei Folgen, wobei $x_i, y_j \in A$ für ein endliches Alphabet A.
- Dann heißt Y Teilfolge von X, wenn es aufsteigend sortierte Indizes $i_1,\dots,i_n$ gibt mit $x_{i_j} = y_j$ für $j = 1,\dots,n$.

Beispiel:

Folge Y

B	C	A	C

Folge X

A	B	A	C	A	B	C

Dynamische Programmierung
Längste gemeinsame Teilfolge

Definition:

- Seien $X=(x_1,...,x_m)$ und $Y=(y_1,...,y_n)$ zwei Folgen, wobei x_i, $y_j \in A$ für ein endliches Alphabet A.
- Dann heißt Y Teilfolge von X, wenn es aufsteigend sortierte Indizes $i_1,...,i_n$ gibt mit $x_{i_j} = y_j$ für $j = 1,...,n$.

Beispiel:

Folge Y	B	C	A	C			
Folge X	A	B	A	C	A	B	C

- Y ist Teilfolge von X, wähle $(i_1,i_2,i_3,i_4) = (2,4,5,7)$

Dynamische Programmierung
Längste gemeinsame Teilfolge

Definition:

- Seien X, Y, Z Folgen über A.
- Dann heißt Z gemeinsame Teilfolge von X und Y, wenn Z Teilfolge sowohl von X als auch von Y ist.

Definition:

- Seien X, Y, Z Folgen über A.
- Dann heißt Z gemeinsame Teilfolge von X und Y, wenn Z Teilfolge sowohl von X als auch von Y ist.

Beispiel:

Folge Z

B	C	A	C

Folge X

A	B	A	C	A	B	C

Folge Y

B	A	C	C	A	B	B	C

- Z ist gemeinsame Teilfolge von X und Y

Dynamische Programmierung
Längste gemeinsame Teilfolge

Definition:

- Seien X, Y, Z Folgen über A.
- Dann heißt Z gemeinsame Teilfolge von X und Y, wenn Z Teilfolge sowohl von X als auch von Y ist.

Beispiel:

Folge Z	B	C	A	C				
Folge X	A	B	A	C	A	B	C	
Folge Y	B	A	C	C	A	B	B	C

- Z ist gemeinsame Teilfolge von X und Y

Definition:

- Seien X, Y, Z Folgen über A.
- Dann heißt Z gemeinsame Teilfolge von X und Y, wenn Z Teilfolge sowohl von X als auch von Y ist.

Beispiel:

Folge Z	B	C	A	C				
Folge X	A	B	A	C	A	B	C	
Folge Y	B	A	C	C	A	B	B	C

- Z ist gemeinsame Teilfolge von X und Y

Dynamische Programmierung
Längste gemeinsame Teilfolge

Definition:

- Seien X, Y, Z Folgen über A.
- Dann heißt Z längste gemeinsame Teilfolge von X und Y, wenn Z gemeinsame Teilfolge von X und Y ist und es keine andere gemeinsame Teilfolge von X und Y gibt, die größere Länge als Z besitzt.

Dynamische Programmierung
Längste gemeinsame Teilfolge

Definition:

- Seien X, Y, Z Folgen über A.
- Dann heißt Z längste gemeinsame Teilfolge von X und Y, wenn Z gemeinsame Teilfolge von X und Y ist und es keine andere gemeinsame Teilfolge von X und Y gibt, die größere Länge als Z besitzt.

Beispiel:

Folge X	A	B	A	C	A	B	C	
Folge Y	B	A	C	C	A	B	B	C

Dynamische Programmierung
Längste gemeinsame Teilfolge

Definition:

- Seien X, Y, Z Folgen über A.
- Dann heißt Z längste gemeinsame Teilfolge von X und Y, wenn Z gemeinsame Teilfolge von X und Y ist und es keine andere gemeinsame Teilfolge von X und Y gibt, die größere Länge als Z besitzt.

Beispiel:

Folge X	A	B	A	C	A	B	C	
Folge Y	B	A	C	C	A	B	B	C
Folge Z_1	B	C	A	C				

Dynamische Programmierung
Längste gemeinsame Teilfolge

Definition:

- Seien X, Y, Z Folgen über A.
- Dann heißt Z längste gemeinsame Teilfolge von X und Y, wenn Z gemeinsame Teilfolge von X und Y ist und es keine andere gemeinsame Teilfolge von X und Y gibt, die größere Länge als Z besitzt.

Beispiel:

Folge X

A	B	A	C	A	B	C

Folge Y

B	A	C	C	A	B	B	C

Längste Teilfolge hat Länge 6

Folge Z_1

B	C	A	C

Folge Z_2

B	A	C	A	C

Dynamische Programmierung
Längste gemeinsame Teilfolge

Eingabe:

- Folge $X=(x_1,\ldots,x_m)$
- Folge $Y=(y_1,\ldots,y_n)$

Ausgabe:

- Längste gemeinsame Teilfolge Z (**L**ongest **C**ommon **S**ubsequence)

Dynamische Programmierung
Längste gemeinsame Teilfolge

Eingabe:

- Folge $X=(x_1,...,x_m)$
- Folge $Y=(y_1,...,y_n)$

Ausgabe:

- Längste gemeinsame Teilfolge Z
  (**L**ongest **C**ommon **S**ubsequence)

Beispiel:

Folge X

A	B	C	B	D	A	B

Folge Y

B	D	C	A	B	A

Längste gemeinsame Teilfolge

Algorithmus:

- Erzeuge alle möglichen Teilfolgen von X
- Teste für jede Teilfolge von X, ob auch Teilfolge von Y
- Merke zu jedem Zeitpunkt bisher längste gemeinsame Teilfolge

Laufzeit:

- 2^m mögliche Teilfolgen
- Exponentielle Laufzeit!

Dynamische Programmierung
Längste gemeinsame Teilfolge

Satz 20.1:

Seien $X=(x_1,...,x_m)$ und $Y=(y_1,...,y_n)$ beliebige Folgen und sei $Z=(z_1,...,z_k)$ eine längste gemeinsame Teilfolge von X und Y. Dann gilt

1. Ist $x_m = y_n$, dann ist $z_k = x_m = y_n$ und $(z_1,...,z_{k-1})$ ist eine längste gemeinsame Teilfolge von $(x_1,...,x_{m-1})$ und $(y_1,...,y_{n-1})$.

Dynamische Programmierung
Längste gemeinsame Teilfolge

Satz 20.1:

Seien $X=(x_1,\dots,x_m)$ und $Y=(y_1,\dots,y_n)$ beliebige Folgen und sei $Z=(z_1,\dots,z_k)$ eine längste gemeinsame Teilfolge von X und Y. Dann gilt

1. Ist $x_m = y_n$, dann ist $z_k = x_m = y_n$ und $(z_1,\dots,z_{k-1})$ ist eine längste gemeinsame Teilfolge von $(x_1,\dots,x_{m-1})$ und $(y_1,\dots,y_{n-1})$.
2. Ist $x_m \neq y_n$ und $z_k \neq x_m$, dann ist Z eine längste gemeinsame Teilfolge von $(x_1,\dots,x_{m-1})$ und Y.

Satz 20.1:

Seien $X=(x_1,...,x_m)$ und $Y=(y_1,...,y_n)$ beliebige Folgen und sei $Z=(z_1,...,z_k)$ eine längste gemeinsame Teilfolge von X und Y. Dann gilt

1. Ist $x_m = y_n$, dann ist $z_k = x_m = y_n$ und $(z_1,...,z_{k-1})$ ist eine längste gemeinsame Teilfolge von $(x_1,...,x_{m-1})$ und $(y_1,...,y_{n-1})$.
2. Ist $x_m \neq y_n$ und $z_k \neq x_m$, dann ist Z eine längste gemeinsame Teilfolge von $(x_1,...,x_{m-1})$ und Y.
3. Ist $x_m \neq y_n$ und $z_k \neq y_n$, dann ist Z eine längste gemeinsame Teilfolge von X und $(y_1,...,y_{n-1})$.

Dynamische Programmierung
Längste gemeinsame Teilfolge

Satz 20.1:

Seien $X=(x_1,...,x_m)$ und $Y=(y_1,...,y_n)$ beliebige Folgen und sei $Z=(z_1,...,z_k)$ eine längste gemeinsame Teilfolge von X und Y. Dann gilt

1. Ist $x_m = y_n$, dann ist $z_k = x_m = y_n$ und $(z_1,...,z_{k-1})$ ist eine längste gemeinsame Teilfolge von $(x_1,...,x_{m-1})$ und $(y_1,...,y_{n-1})$.
2. Ist $x_m \neq y_n$ und $z_k \neq x_m$, dann ist Z eine längste gemeinsame Teilfolge von $(x_1,...,x_{m-1})$ und Y.
3. Ist $x_m \neq y_n$ und $z_k \neq y_n$, dann ist Z eine längste gemeinsame Teilfolge von X und $(y_1,...,y_{n-1})$.

Dynamische Programmierung
Längste gemeinsame Teilfolge

Lemma 20.2:

Sei C[i][j] die Länge einer längsten gemeinsamen Teilfolge von $(x_1,\dots,x_i)$ und $(y_1,\dots,y_j)$. Dann gilt:

$$C[i][j] = \begin{cases} 0 & \text{falls } i = 0 \text{ oder } j = 0 \\ & \\ & \end{cases}$$

Dynamische Programmierung
Längste gemeinsame Teilfolge

Lemma 20.2:

Sei C[i][j] die Länge einer längsten gemeinsamen Teilfolge von $(x_1,\ldots,x_i)$ und $(y_1,\ldots,y_j)$. Dann gilt:

$$C[i][j] = \begin{cases} 0 & \text{falls } i = 0 \text{ oder } j = 0 \\ C[i-1][j-1]+1 & \text{falls } i, j > 0 \text{ und } x_i = y_j \\ & \end{cases}$$

Dynamische Programmierung
Längste gemeinsame Teilfolge

Lemma 20.2:

Sei C[i][j] die Länge einer längsten gemeinsamen Teilfolge von $(x_1,\dots,x_i)$ und $(y_1,\dots,y_j)$. Dann gilt:

$$C[i][j] = \begin{cases} 0 & \text{falls } i = 0 \text{ oder } j = 0 \\ C[i-1][j-1]+1 & \text{falls } i, j > 0 \text{ und } x_i = y_j \\ \max\{C[i-1][j], C[i][j-1]\} & \text{falls } i, j > 0 \text{ und } x_i \neq y_j \end{cases}$$

Dynamische Programmierung Längste gemeinsame Teilfolge

Lemma 20.2:

Sei C[i][j] die Länge einer längsten gemeinsamen Teilfolge von $(x_1,...,x_i)$ und $(y_1,...,y_j)$. Dann gilt:

$$C[i][j] = \begin{cases} 0 & \text{falls } i = 0 \text{ oder } j = 0 \\ C[i-1][j-1]+1 & \text{falls } i, j > 0 \text{ und } x_i = y_j \\ \max\{C[i-1][j], C[i][j-1]\} & \text{falls } i, j > 0 \text{ und } x_i \neq y_j \end{cases}$$

Beobachtung:

Rekursive Berechnung der C[i][j] würde zu Berechnung immer wieder derselben Werte führen. Dieses ist ineffizient. Berechnen daher die Werte C[i][j] iterativ, nämlich zeilenweise.

Dynamische Programmierung
Längste gemeinsame Teilfolge

```
LCS-Länge(Array X, Y)
m ← length[X]
n ← length[Y]
new array C[0,..,m][0,..,n]
for i ← 0 to m do C[i][0] ← 0
for j ← 0 to n do C[0][j] ← 0
for i ← 1 to m do
    for j ← 1 to n do
        ➢ Längenberechnung(X, Y, C, i, j)
return C
```

Dynamische Programmierung
Längste gemeinsame Teilfolge

```
LCS-Länge(Array X, Y)
    m ← length[X]
    n ← length[Y]
    new array C[0,..,m][0,..,n]
    for i ← 0 to m do C[i][0] ← 0
    for j ← 0 to n do C[0][j] ← 0
    for i ← 1 to m do
        for j ← 1 to n do
            ➢ Längenberechnung(X, Y, C, i, j)
    return C
```

```
LCS-Länge(Array X, Y)
    m ← length[X]
    n ← length[Y]
    new array C[0,..,m][0,..,n]
    for i ← 0 to m do C[i][0] ← 0
    for j ← 0 to n do C[0][j] ← 0
    for i ← 1 to m do
        for j ← 1 to n do
            ➢ Längenberechnung(X, Y, C, i, j)
    return C
```

Tabelle für die C[i][j] Werte anlegen.

Dynamische Programmierung
Längste gemeinsame Teilfolge

LCS-Länge(Array X, Y)

1. m ← length[X]
2. n ← length[Y]
3. **new** array C[0,..,m][0,..,n]
4. **for** i ← 0 **to** m **do** C[i][0] ← 0
5. **for** j ← 0 **to** n **do** C[0][j] ← 0
6. **for** i ← 1 **to** m **do**
7.   **for** j ← 1 **to** n **do**
8.   ➢ Längenberechnung(X, Y, C, i, j)
9. **return** C

Erste Spalte der Tabelle auf 0 setzen.

Dynamische Programmierung
Längste gemeinsame Teilfolge

LCS-Länge(Array X, Y)

1. m ← length[X]
2. n ← length[Y]
3. **new** array C[0,..,m][0,..,n]
4. **for** i ← 0 **to** m **do** C[i][0] ← 0
5. **for** j ← 0 **to** n **do** C[0][j] ← 0
6. **for** i ← 1 **to** m **do**
7. **for** j ← 1 **to** n **do**
8. ➢ Längenberechnung(X, Y, C, i, j)
9. **return** C

Erste Reihe der Tabelle auf 0 setzen.

Dynamische Programmierung
Längste gemeinsame Teilfolge

```
LCS-Länge(Array X, Y)
m ← length[X]
n ← length[Y]
new array C[0,..,m][0,..,n]
for i ← 0 to m do C[i][0] ← 0
for j ← 0 to n do C[0][j] ← 0
for i ← 1 to m do
    for j ← 1 to n do
        ➢ Längenberechnung(X, Y, C, i, j)
return C
```

Dynamische Programmierung
Längste gemeinsame Teilfolge

Längenberechnung(Array X, Y, C, i, j)

1. **if** $x_i = y_j$ **then** C[i][j] $\leftarrow$ C[i-1][j-1] +1
2. **else**
3. **if** C[i-1][j] $\geq$ C[i][j-1] **then** C[i][j] $\leftarrow$ C[i-1][j]
4. **else** C[i][j] $\leftarrow$ C[i][j-1]

$$C[i][j] = \begin{cases} 0 & \text{falls } i = 0 \text{ oder } j = 0 \\ \boxed{C[i-1][j-1]+1 \qquad \text{falls } i, j > 0 \text{ und } x_i = y_j} & \\ \max\{C[i-1][j], C[i][j-1]\} & \text{falls } i, j > 0 \text{ und } x_i \neq y_j \end{cases}$$

Dynamische Programmierung
Längste gemeinsame Teilfolge

Längenberechnung(Array X, Y, C, i, j)

1. **if** $x_i = y_j$ **then** C[i][j] $\leftarrow$ C[i-1][j-1] +1
2. **else**
3. **if** C[i-1][j] $\geq$ C[i][j-1] **then** C[i][j] $\leftarrow$ C[i-1][j]
4. **else** C[i][j] $\leftarrow$ C[i][j-1]

$$C[i][j] = \begin{cases} 0 & \text{falls } i = 0 \text{ oder } j = 0 \\ C[i-1][j-1]+1 & \text{falls } i, j > 0 \text{ und } x_i = y_j \\ \boxed{\max\{C[i-1][j], C[i][j-1]\} \quad \text{falls } i, j > 0 \text{ und } x_i \neq y_j} \end{cases}$$

Dynamische Programmierung
Längste gemeinsame Teilfolge

LCS-Länge(Array X, Y)

1. m ← length[X]
2. n ← length[Y]
3. **new** array C[0,..,m][0,..,n]
4. **for** i ← 0 **to** m **do** C[i][0] ← 0
5. **for** j ← 0 **to** n **do** C[0][j] ← 0
6. **for** i ← 1 **to** m **do**
7.   **for** j ← 1 **to** n **do**
8.   ➢ Längenberechnung(X, Y, C, i, j)
9. **return** C

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i							
1	A							
2	B							
3	C							
4	B							
5	D							
6	A							
7	B							

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0						
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0					
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0					
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0					
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0				
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0			
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0			
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

i	j	0	1	2	3	4	5	6
	y_j		B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0			
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1		
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1		
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1		
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1					
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1				
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1			
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0						
6	A	0						
7	B	0						

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Dynamische Programmierung
Längste gemeinsame Teilfolge

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Dynamische Programmierung
Längste gemeinsame Teilfolge

i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Dynamische Programmierung
Längste gemeinsame Teilfolge

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Dynamische Programmierung
Längste gemeinsame Teilfolge

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Dynamische Programmierung
Längste gemeinsame Teilfolge

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Dynamische Programmierung
Längste gemeinsame Teilfolge

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Dynamische Programmierung
Längste gemeinsame Teilfolge

Lemma 20.3:

Der Algorithmus LCS-Länge hat Laufzeit O(nm), wenn die Folgen X,Y Länge n und m haben.

Lemma 20.4:

Die Ausgabe der längsten gemeinsamen Teilfolge anhand der Tabelle hat Laufzeit O(n+m), wenn die Folgen X,Y Länge n und m haben.

Dynamische Programmierung
Längste gemeinsame Teilfolge

1. Bestimme rekursive Struktur einer optimalen Lösung.

2. Entwerfe rekursive Methode zur Bestimmung des Wertes einer optimalen Lösung.

3. Transformiere rekursive Methode in eine iterative Methode zur Bestimmung des Wertes einer optimalen Lösung.

4. Bestimme aus dem Wert einer optimalen Lösung und den in 3. berechneten Zusatzinformationen eine optimale Lösung.

Fraktionales Rucksack-Problem

- Gegeben sind n Gegenstände. Der i-te Gegenstand besitzt Wert v_i und Gewicht g_i. Ausserdem ist eine Gewichtsschranke W gegeben.

- Zulässige Lösungen sind Zahlen $a_i \in [0,1]$ mit

$$\sum_{i=1}^{n} a_i g_i \leq W.$$

- Gesucht ist eine zulässige Lösung $a_1,\ldots,a_n$ mit möglichst großem Gesamtwert

$$\sum_{i=1}^{n} a_i v_i.$$

Gierige Lösung des fraktionalen Rucksack-Problems

Algorithmus Gieriges-Einpacken:

1. Sortiere die Verhältnisse v_i/g_i absteigend. Sei

$$v_{\pi(1)}/g_{\pi(1)} \geq v_{\pi(2)}/g_{\pi(2)} \geq \cdots v_{\pi(n)}/g_{\pi(n)}$$

   für Permutation π auf (1,…,n).

2. Bestimme maximales k, so dass noch gilt $\sum_{i=1}^{k} g_{\pi(i)} \leq W$.

3. Setze $a_{\pi(1)} = a_{\pi(2)} = \cdots = a_{\pi(k)} = 1$ und setze

$$a_{\pi(k+1)} = \frac{W - \sum_{i=1}^{k} g_{\pi(i)}}{g_{\pi(k+1)}}.$$

   Alle anderen a_i setze auf 0.

Gieriges Einpacken ist optimal

Satz 20.5: Gieriges Einpacken löst das fraktionale Rucksackproblem optimal.

Beweis:

- Sei (g,v,W) ein beliebiges fraktionales Rucksackproblem mit $g:\{1,\dots,n\}\to\mathbb{N}$, $v:\{1,\dots,n\}\to\mathbb{N}$ und $W\in\mathbb{N}$.
- Betrachte das Rucksackproblem (g',v',W), für das für jeden Gegenstand i im originalen Rucksackproblem g_i Stücke mit Gewicht $g'_i=1$ und Wert $v'_i=v_i/g_i$ existieren. Sei $n'=\Sigma_i\, g_i$ die daraus resultierende Anzahl an Gegenständen. Offensichtlich sind die optimalen Werte für beide Probleme identisch.
- Sei $S'\subseteq\{1,\dots,n'\}$ eine optimale Lösung für (g',v',W). Dann gilt für alle $j\in\{1,\dots,n'\}\backslash S'$, dass $v'_j \leq \min_{i\in S} v'_i$ ist. (Sonst würde ein Austausch den Wert verbessern.)
- Da der Wert von S' dem des gierigen Einpackens für (g,v,W) entspricht, folgt Satz 20.5.

Dynamisches Programmieren – Rucksack

Das Rucksackproblem:

- Rucksack mit begrenzter Kapazität
- Objekte mit unterschiedlichem Wert und unterschiedlicher Größe
- Wir wollen Objekte von möglichst großem Gesamtwert mitnehmen

Dynamisches Programmieren – Rucksack

Beispiel:

- Rucksackgröße 6

Größe	5	2	1	3	7	4
Wert	11	5	2	8	14	9

Dynamisches Programmieren – Rucksack

Beispiel:

- Rucksackgröße 6

Größe	5	2	1	3	7	4
Wert	11	5	2	8	14	9

- Objekt 1 und 3 passen und haben Gesamtwert 13
- Optimal?

Dynamisches Programmieren – Rucksack

Beispiel:

- Rucksackgröße 6

Größe	5	2	1	3	7	4
Wert	11	5	2	8	14	9

- Objekt 1 und 3 passen und haben Gesamtwert 13
- Optimal?
- Objekt 2, 3 und 4 passen und haben Gesamtwert 15 !

Das Rucksackproblem (Optimierungsversion):

- Eingabe: n Objekte {1,…,n};
  Objekt i hat ganzz. pos. Größe g[i] und Wert v[i];
  Rucksackkapazität W

- Ausgabe: Menge $S \subseteq \{1,\ldots,n\}$ mit $\sum_{i \in S} g[i] \leq W$ und
  maximalem Wert $\sum_{i \in S} v[i]$

Dynamisches Programmieren – Rucksack

Herleiten einer Rekursion:

- Sei O optimale Lösung
- Bezeichne Opt(i,w) den Wert einer optimalen Lösung aus Objekten 1 bis i bei Rucksackgröße w

Unterscheide, ob Objekt n in O ist:

- Fall 1(n nicht in O):
  Opt(n,W) = Opt(n-1,W)
- Fall 2 (n in O):
  Opt(n,W) = v[n] + Opt(n-1,W-g[n])

Dynamisches Programmieren – Rucksack

Rekursion:

- $Opt(i,0) = 0$ für $0 \leq i \leq n$
- $Opt(0,i) = 0$ für $0 \leq i \leq W$
- Wenn $w < g[i]$ dann $Opt(i,w) = Opt(i-1,w)$
- Sonst,
  $Opt(i,w) = \max\{Opt(i-1,w), v[i] + Opt(i-1,w-g[i])\}$

Dynamisches Programmieren – Rucksack

Rekursion:

- Opt(i,0)= 0 für 0≤i≤n

Kein Objekt passt in den Rucksack

- Opt(0,i)= 0 für 0≤i≤W
- Wenn w<g[i] dann Opt(i,w) = Opt(i-1,w)
- Sonst,
  Opt(i,w) = max{Opt(i-1,w), v[i] + Opt(i-1,w-g[i])}

Dynamisches Programmieren – Rucksack

Kein Objekt steht zur Auswahl

Rekursion:

- Opt(i,0)= 0 für 0≤i≤n
- Opt(0,i)= 0 für 0≤i≤W
- Wenn w<g[i] dann Opt(i,w) = Opt(i-1,w)
- Sonst,
  Opt(i,w) = max{Opt(i-1,w), v[i] + Opt(i-1,w-g[i])}

Dynamisches Programmieren – Rucksack

Rekursion:

- Opt(i,0)= 0 für $0 \leq i \leq n$
- Opt(0,i)= 0 für $0 \leq i \leq W$
- Wenn w<g[i] dann Opt(i,w) = Opt(i-1,w)
- Sonst,
  Opt(i,w) = max{Opt(i-1,w), v[i] + Opt(i-1,w-g[i])}

Passst aktuelles Objekt in den Rucksack?

Dynamisches Programmieren – Rucksack

Rekursion:

- Opt(i,0)= 0 für $0 \leq i \leq n$
- Opt(0,i)= 0 für $0 \leq i \leq W$
- Wenn w<g[i] dann Opt(i,w) = Opt(i-1,w)
- Sonst,
  Opt(i,w) = max{Opt(i-1,w), v[i] + Opt(i-1,w-g[i])}

Sonst, verwende Rekursion

Dynamisches Programmieren – Rucksack

Rucksack(n,W)

1. Initialisiere Feld A[0,..,n][0,..,W] mit A[0,i] =0 für alle $0 \le i \le n$ und A[j,0]=0 für alle $0 \le i \le W$
2. **for** i $\leftarrow$ 1 **to** n **do**
3.   **for** j $\leftarrow$ 1 **to** W **do**
4.   Berechne A[i,j] nach Rekursion
5. **return** A[n,W]

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0								
	0								
	0								
	0								
	0								
1	0								
0	0	0	0	0	0	0	0	0	0
	0	1							W

Größe Wert

	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0								
	0								
	0								
	0								
	0								
1	0	0	0	0	0				
0	0	0	0	0	0	0	0	0	0
	0	1							W

g[1]>W:
Also Opt(i,w) = Opt(i-1,w)

Größe Wert

	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0								
	0								
	0								
	0								
	0								
1	0	0	0	0	0	2			
0	0	0	0	0	0	0	0	0	0
	0	1							W

Opt(i,w) = max{Opt(i-1,w),
v[i] + Opt(i-1,w-g[i])}

Größe Wert

	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0								
	0								
	0								
	0								
	0								
1	0	0	0	0	0	2	2		
0	0	0	0	0	0	0	0	0	0
	0	1							W

Größe, Wert

	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0								
	0								
	0								
	0								
	0								
1	0	0	0	0	0	2	2	2	
0	0	0	0	0	0	0	0	0	0
	0	1							W

Größe Wert

	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0								
	0								
	0								
	0								
	0								
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

Größe Wert

	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0								
	0								
	0								
	0								
	0	0	0						
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

Größe Wert

	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0								
	0								
	0								
	0								
	0	0	0	4	4				
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

	Größe	Wert
	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0								
	0								
	0								
	0								
	0	0	0	4	4	4	4	4	
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

Größe, Wert

	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

	0	1							W
n	0								
	0								
	0								
	0								
	0								
	0								
	0	0	0	4	4	4	4	4	6
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0

Größe Wert

	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0								
	0								
	0								
	0	1	1						
	0	0	0	4	4	4	4	4	6
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

Größe Wert

	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0								
	0								
	0								
	0	1	1	4					
	0	0	0	4	4	4	4	4	6
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

	Größe	Wert
	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0								
	0								
	0								
	0	1	1	4	5	5	5	5	
	0	0	0	4	4	4	4	4	6
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

	Größe	Wert
	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0								
	0								
	0								
	0	1	1	4	5	5	5	5	6
	0	0	0	4	4	4	4	4	6
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

Größe Wert

	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0								
	0								
	0	1	3	4	5	7	8	8	8
	0	1	1	4	5	5	5	5	6
	0	0	0	4	4	4	4	4	6
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

	Größe	Wert
	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0								
	0	2	3	5	6	7	9	10	10
	0	1	3	4	5	7	8	8	8
	0	1	1	4	5	5	5	5	6
	0	0	0	4	4	4	4	4	6
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

	Größe	Wert
	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0								
	0	2	3	5	6	7	9	10	10
	0	2	3	5	6	7	9	10	10
	0	1	3	4	5	7	8	8	8
	0	1	1	4	5	5	5	5	6
	0	0	0	4	4	4	4	4	6
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

Größe Wert

	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0								
	0	2	3	5	7	9	10	12	13
	0	2	3	5	6	7	9	10	10
	0	2	3	5	6	7	9	10	10
	0	1	3	4	5	7	8	8	8
	0	1	1	4	5	5	5	5	6
	0	0	0	4	4	4	4	4	6
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

	Größe	Wert
	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0	2	3	5	7	9	10	12	13
	0	2	3	5	7	9	10	12	13
	0	2	3	5	6	7	9	10	10
	0	2	3	5	6	7	9	10	10
	0	1	3	4	5	7	8	8	8
	0	1	1	4	5	5	5	5	6
	0	0	0	4	4	4	4	4	6
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

	Größe	Wert
	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

n	0	2	3	5	7	9	10	12	13
	0	2	3	5	7	9	10	12	13
	0	2	3	5	6	7	9	10	10
	0	2	3	5	6	7	9	10	10
	0	1	3	4	5	7	8	8	8
	0	1	1	4	5	5	5	5	6
	0	0	0	4	4	4	4	4	6
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

Optimaler Lösungswert für W=8

	Größe	Wert
	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamisches Programmieren – Rucksack

Beispiel:

	0	1							W
n	0	2	3	5	7	9	10	12	13
	0	2	3	5	7	9	10	12	13
	0	2	3	5	6	7	9	10	10
	0	2	3	5	6	7	9	10	10
	0	1	3	4	5	7	8	8	8
	0	1	1	4	5	5	5	5	6
	0	0	0	4	4	4	4	4	6
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0

Optimaler Lösungswert für W=8

Größe, Wert

	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Satz 20.6

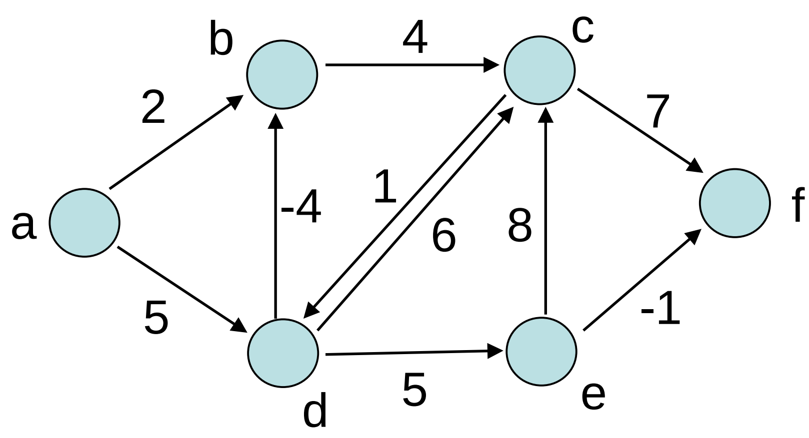
Algorithmus Rucksack berechnet in $\Theta(nW)$ Zeit den Wert einer optimalen Lösung, wobei n die Anzahl der Objekte ist und W die Größe des Rucksacks.

Dynamische Programmierung - APSP

All Pairs Shortest Path (APSP):

- Eingabe: Gewichteter Graph G=(V,E)
- Ausgabe: Für jedes Paar von Knoten u,v∈V die Distanz von u nach v sowie einen kürzesten Weg

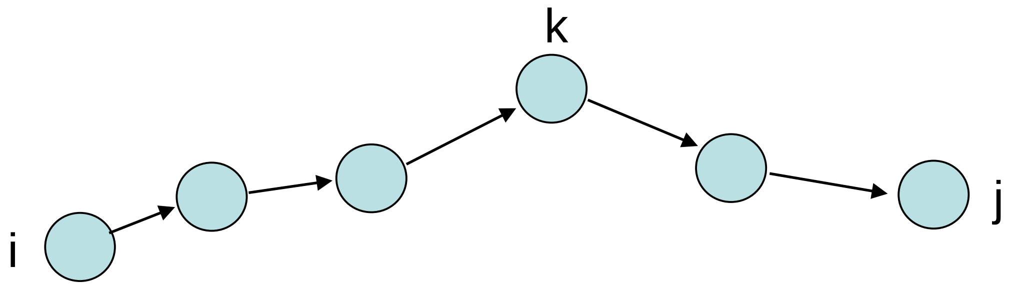
	a	b	c	d	e	f
a	0	1	5	5	10	9
b	∞	0	4	5	10	9
c	∞	-3	0	1	6	5
d	∞	-4	0	0	5	4
e	∞	5	8	9	0	-1
f	∞	∞	∞	∞	∞	0



Dynamische Programmierung - APSP

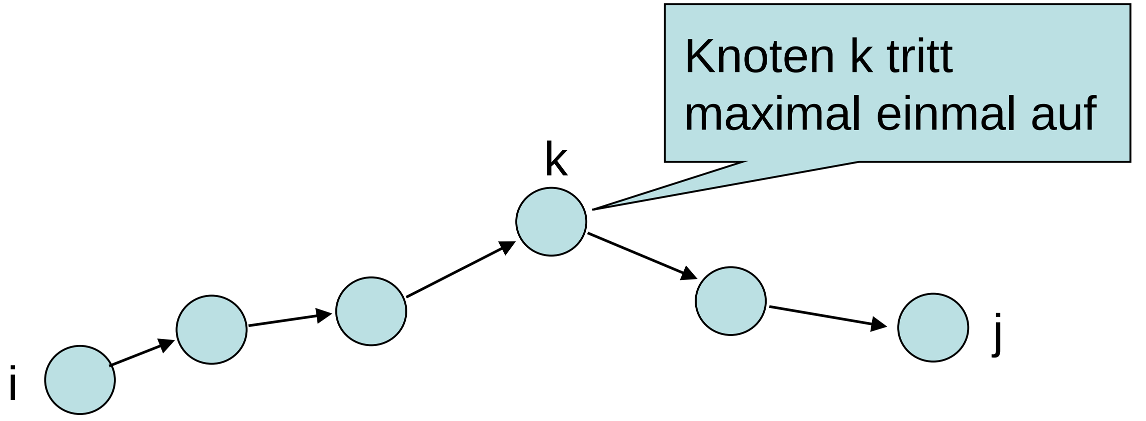
Zur Erinnerung:

- Sei G ein Graph ohne negative Zyklen und sei j von i aus erreichbar. Dann gibt es einen kürzesten i-j-Weg, der keinen Knoten doppelt benutzt.
- Wir können also annehmen, dass jeder Knoten in jedem Weg maximal einmal vorkommt
- Betrachte i-j-Weg, der nur über Knoten aus {1,…,k} läuft:



Zur Erinnerung:

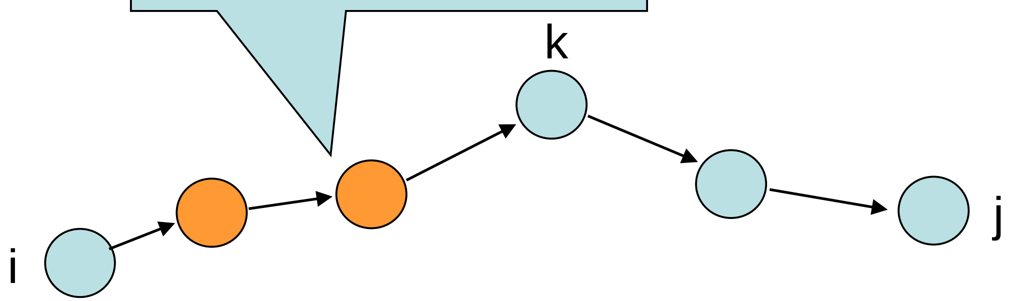
- Sei G ein Graph ohne negative Zyklen und sei j von i aus erreichbar. Dann gibt es einen kürzesten i-j-Weg, der keinen Knoten doppelt benutzt.
- Wir können also annehmen, dass jeder Knoten in jedem Weg maximal einmal vorkommt
- Betrachte i-j-Weg, der nur über Knoten aus {1,…,k} läuft:



Zur Erinnerung:

- Sei G ein Graph ohne negative Zyklen und sei j von i aus erreichbar. Dann gibt es einen kürzesten i-j-Weg, der keinen Knoten doppelt benutzt.
- Wir können also annehmen, dass jeder Knoten in jedem Weg maximal einmal vorkommt
- Betracht … über Knoten aus {1,…,k} läuft:

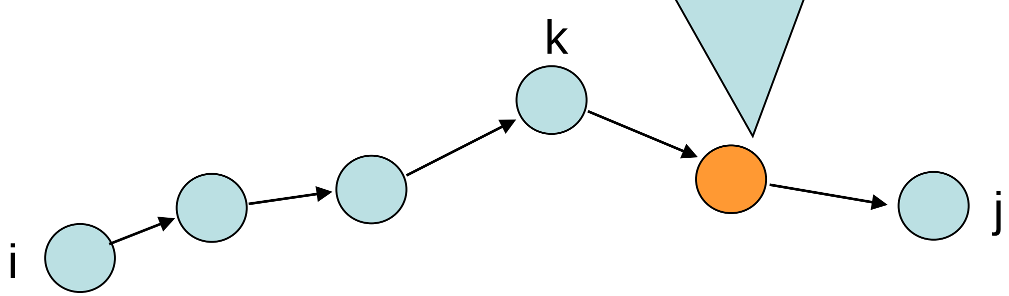
Weg von u nach k
führt nur über Knoten
aus {1,…,k-1}



Zur Erinnerung:

- Sei G ein Graph ohne negative Zyklen und sei j von i aus erreichbar. Dann gibt es einen kürzesten i-j-Weg, der keinen Knoten doppelt benutzt.
- Wir können also annehmen, dass jeder Knoten in jedem Weg maximal einmal vorkommt
- Betrachte i-j-Weg, der n… s {1,…,k} läuft:

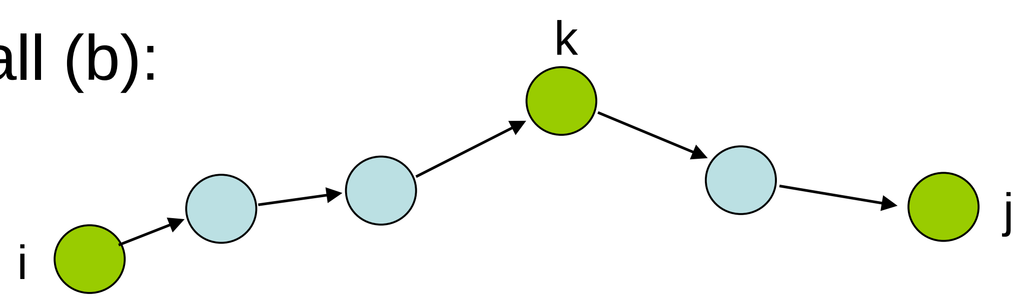
Weg von k nach v führt nur über Knoten aus {1,…,k-1}



Dynamische Programmierung - APSP

- Kürzester i-j-Weg über Knoten aus {1,…,k} ist
- (a) kürzester i-j-Weg über Knoten aus {1,…,k-1} oder
- (b) kürzester i-k-Weg über Knoten aus {1,…,k-1} gefolgt von kürzestem k-j-Weg über Knoten aus {1,…,k-1}

Fall (b):



Dynamische Programmierung - APSP

Die Rekursion:

- Sei $d_{ij}^{(k)}$ die Länge eines kürzesten i-j-Wegs mit Knoten aus {1,…,k}

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{, falls } k=0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right) & \text{, falls } k \geq 1 \end{cases}$$

- Matrix $D^{(n)} = (d_{ij}^{(n)})$ enthält die gesuchte Lösung

Dynamische Programmierung - APSP

Floyd-Warshall(W,n)

1. $D^{(0)} \leftarrow W$
2. **for** $k \leftarrow 1$ **to** n **do**
3.   **for** $i \leftarrow 1$ **to** n **do**
4.   **for** $j \leftarrow 1$ **to** n **do**
5. $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
6. **return** $D^{(n)}$

Dynamische Programmierung

Generelle Vorgehensweise:

1. Aufstellung der Rekursionsgleichung
   – Initialfall (z.B. OPT(i,0)=OPT(0,j)=0)
   – Rekursion (z.B. OPT(i,j)=…)
   – Optimaler Wert (z.B. OPT = OPT(n,W))

2. Formulierung des dynamischen Programms
   Wichtig: tabellarische Berechnung so durchführen, dass auf bereits berechnete Werte zurückgegriffen werden kann!

Dynamische Programmierung

Beispiel 1: Matrixketten-Multiplikation

Gegeben: Matrizen $A_1,\ldots,A_n$, und Werte $p_0,\ldots,p_n \in \mathbb{N}$, wobei Matrix A_i eine $p_{i-1} \times p_i$-Matrix ist.

Gesucht: Minimale Anzahl an Multiplikationen, um $A_1 \cdot \ldots \cdot A_n$ zu berechnen.

Beobachtung:

- Sei $A_{i..j} = A_i \cdot \ldots \cdot A_j$
- Dann kostet das Matrixprodukt $A_{i..k} \cdot A_{k+1..j}$ $p_{i-1} \cdot p_k \cdot p_j$ viele Multiplikationen (mit der naiven Methode)

Dynamische Programmierung

- $m[i,j]$: minimale Anzahl an Multiplikationen, um $A_{i..j}$ zu berechnen

Initialfall:

$m[i,i]=0$ für alle $i \in \{1,\dots,n\}$.

Rekursion:

$m[i,j] = \min_{i \le k < j} m[i,k] + m[k+1,j] + p_{i-1} \cdot p_k \cdot p_j$ für alle $i<j$

Optimaler Wert:

$m[1,n]$

Berechnung:

- Führe erst Initialfall aus
- Berechne $m[i,j]$ für alle $i<j$ mit $|j-i|=d$, angefangen mit $d=1$

Beispiel 2: Optimaler binärer Suchbaum

Gegeben: Schlüssel $k_1,\ldots,k_n$, mit Zugriffswahrscheinlichkeiten $p_1,\ldots,p_n \in [0,1]$, so dass $\Sigma_{i=1}^{n} p_i = 1$.

Gesucht: Binärer Suchbaum T mit minimaler erwarteter Suchzeit, d.h. $\Sigma_{i=1} p_i \cdot (\text{Tiefe}_T(k_i)+1)$ ist minimal.
($\text{Tiefe}_T(k)$: Tiefe des Knotens mit Schlüssel k in T
(Tiefe der Wurzel ist 0))

Dynamische Programmierung

- $m[i,j]$: minimale erwartete Suchzeit für einen Binärbaum mit den Schlüsseln k_i bis k_j

Initialfall:

- $m[i,i-1]=0$ für alle $i \in \{1,\dots,n\}$
- $m[i,i]=p_i$ für alle $i \in \{1,\dots,n\}$

Rekursion:

$m[i,j] = \min_{i \le k \le j} m[i,k-1] + m[k+1,j] + \sum_{l=i}^{j} p_l$ für alle $i<j$

Optimaler Wert:

$m[1,n]$

Berechnung:

- Führe erst Initialfall aus
- Berechne $m[i,j]$ für alle $i<j$ mit $|j-i|=d$, angefangen mit $d=1$
- Gib am Ende $m[1,n]$ aus

Beispiel 3: Längster einfacher Weg in einem gerichteten azyklischen Graph

Gegeben: Gerichteter azyklischer Graph $G=(V,E)$ mit $V=\{1,\ldots,n\}$, wobei die Knoten gemäß ihrer Nummern topologisch sortiert sind, d.h. für alle $(v,w)\in E$ gilt $v<w$.

Gesucht: Länge des längsten einfachen (d.h. kreisfreien) Weges in G.

Dynamische Programmierung

- $L[i,j]$: Länge des längsten einfachen Weges von Knoten i nach Knoten j in G
- $A[i,j] \in \{0,1\}$: ist 1 genau dann wenn $(i,j) \in E$

Initialfall:

- $L[i,i]=0$ für alle $i \in \{1,\ldots,n\}$

Falls die Menge leer ist, ist max{…}=0.

Rekursion:

$L[i,j] = \max \{ \max\{L[i,k] + L[k,j] \mid i<k<j$ und $L[i,k]>0$ und $L[k,j]>0\},$
$A[i,j] \}$ für alle $i<j$

Optimaler Wert:

$\max_{i<j} L[i,j]$ (=0: es gibt keine Kanten in G)

Berechnung:

- Führe erst Initialfall aus
- Berechne $L[i,j]$ für alle $i<j$ mit $|j-i|=d$, angefangen mit $d=1$
- Gib am Ende $\max_{i<j} L[i,j]$ aus

Beispiel 4: Fahrstuhloptimierung

Gegeben: Flure $f_1,\ldots,f_n \in \mathbb{N}$, bei denen die n Kunden aussteigen wollen ($f_1 \leq \ldots \leq f_n$), und maximale Anzahl Haltepunkte $k \in \mathbb{N}$.

Gesucht: Minimale Anzahl an Fluren, die die Kunden bei maximal k Haltepunkten zu Fuß gehen müssen, um ihren Zielflur zu erreichen, wenn der Fahrstuhl bei Flur 1 startet.

Dynamische Programmierung

- $m[i,j]$: minimale Anzahl an Fluren, die die Kunden zu Fuß gehen müssen, wenn der Fahrstuhl genau j-mal hält und der höchste Halt Flur i ist.
- $flure[i,j]$: Minimale Anzahl an Fluren, die Kunden mit Aussteige-wunsch f mit $i \le f < j$ laufen müssen, wenn der Fahrstuhl nur bei Flur i und j aber nicht dazwischen hält. (Kann direkt berechnet werden.)

Initialfall:

$m[i,1] = flure[1,i]+flure[i,\infty]$ für alle $i \in \{2,\ldots,f_n\}$

Rekursion:

$m[i,j+1] = \min_{j+1 \le l < i} m[l,j]-flure[l,\infty]+flure[l,i]+flure[i,\infty]$ für alle $i>j+1,\ j \ge 1$

Optimaler Wert:

$\min_{k+1 \le i \le f_n} m[i,k]$

Dynamische Programmierung

Teile & Herrsche:

- Aufteilen der Eingabe in mehrere Unterprobleme
- Rekursives lösen der Unterprobleme
- Zusammenfügen

Gierige Algorithmen:

- Konstruiere Lösung Schritt für Schritt
- In jedem Schritt optimiere einfaches, lokales Kriterium

Dynamische Programmierung:

- Formuliere Problem rekursiv
- Vermeide mehrfache Berechnung von Teilergebnissen
- Verwende „bottom-up“ Implementierung