

2.3 Dynamische Programmierung

Dynamische Programmierung

Gegeben sei **Instanz \mathcal{I} eines Problems**.

Löse \mathcal{I} wie folgt:

1. **Zerlege \mathcal{I} in Teilinstanzen $\mathcal{I}_1, \dots, \mathcal{I}_k$ für ein $k \geq 1$.**
2. **Löse die Teilinstanzen $\mathcal{I}_1, \dots, \mathcal{I}_k$.**
3. **Kombiniere die Lösungen von $\mathcal{I}_1, \dots, \mathcal{I}_k$ zu Gesamtlösung von \mathcal{I} .**

2.3 Dynamische Programmierung

Dynamische Programmierung

Gegeben sei **Instanz \mathcal{I} eines Problems**.

Löse \mathcal{I} wie folgt:

1. **Zerlege \mathcal{I} in Teilinstanzen $\mathcal{I}_1, \dots, \mathcal{I}_k$ für ein $k \geq 1$.**
2. **Löse die Teilinstanzen $\mathcal{I}_1, \dots, \mathcal{I}_k$.**
3. **Kombiniere die Lösungen von $\mathcal{I}_1, \dots, \mathcal{I}_k$ zu Gesamtlösung von \mathcal{I} .**

Unterschied zu Divide-and-Conquer: **Speichere Lösungen der Teilinstanzen.**

2.3 Dynamische Programmierung

Fibonacci-Zahlen: Für $n \in \mathbb{N}_0$ sei die n -te Fibonacci-Zahl f_n wie folgt definiert:

$$f_n = \begin{cases} 0 & \text{falls } n = 0, \\ 1 & \text{falls } n = 1, \\ f_{n-1} + f_{n-2} & \text{falls } n \geq 2. \end{cases}$$

2.3 Dynamische Programmierung

Fibonacci-Zahlen: Für $n \in \mathbb{N}_0$ sei die n -te Fibonacci-Zahl f_n wie folgt definiert:

$$f_n = \begin{cases} 0 & \text{falls } n = 0, \\ 1 & \text{falls } n = 1, \\ f_{n-1} + f_{n-2} & \text{falls } n \geq 2. \end{cases}$$

FIBREK(int n)

```
1  if ( $n == 0$ ) return 0;  
2  else if ( $n == 1$ ) return 1;  
3  else return FIBREK( $n - 1$ )+FIBREK( $n - 2$ );
```

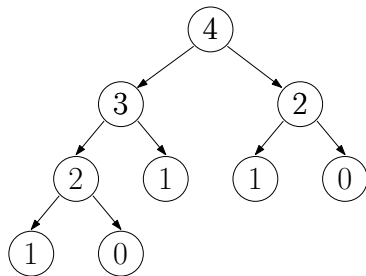
2.3 Dynamische Programmierung

Fibonacci-Zahlen: Für $n \in \mathbb{N}_0$ sei die n -te Fibonacci-Zahl f_n wie folgt definiert:

$$f_n = \begin{cases} 0 & \text{falls } n = 0, \\ 1 & \text{falls } n = 1, \\ f_{n-1} + f_{n-2} & \text{falls } n \geq 2. \end{cases}$$

FIBREK(int n)

```
1  if ( $n == 0$ ) return 0;  
2  else if ( $n == 1$ ) return 1;  
3  else return FIBREK( $n - 1$ )+FIBREK( $n - 2$ );
```



2.3 Dynamische Programmierung

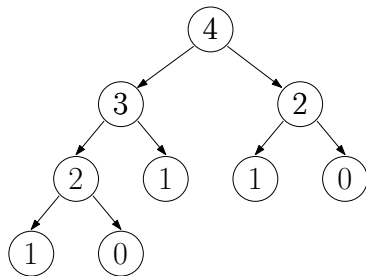
Fibonacci-Zahlen: Für $n \in \mathbb{N}_0$ sei die n -te Fibonacci-Zahl f_n wie folgt definiert:

$$f_n = \begin{cases} 0 & \text{falls } n = 0, \\ 1 & \text{falls } n = 1, \\ f_{n-1} + f_{n-2} & \text{falls } n \geq 2. \end{cases}$$

FIBREK(int n)

```
1  if (n == 0) return 0;  
2  else if (n == 1) return 1;  
3  else return FIBREK(n - 1) + FIBREK(n - 2);
```

Exponentielle Laufzeit, da Teilprobleme mehrfach gelöst werden.



2.3.1 Berechnung optimaler Zuschnitte

Zuschnittproblem

Eingabe: **Länge** $n \in \mathbb{N}$

Preise $p_1, \dots, p_n \in \mathbb{N}$

2.3.1 Berechnung optimaler Zuschnitte

Zuschnittproblem

Eingabe: Länge $n \in \mathbb{N}$

Preise $p_1, \dots, p_n \in \mathbb{N}$

Ausgabe: optimaler Zuschnitt eines Brettes der Länge n

D. h. $\ell \in \{1, \dots, n\}$ und Folge $i_1, \dots, i_\ell \in \{1, \dots, n\}$ mit $i_1 + \dots + i_\ell = n$
sodass Gesamterlös $p_{i_1} + \dots + p_{i_\ell}$ größtmöglich ist.

2.3.1 Berechnung optimaler Zuschnitte

Zuschnittproblem

Eingabe: Länge $n \in \mathbb{N}$

Preise $p_1, \dots, p_n \in \mathbb{N}$

Ausgabe: optimaler Zuschnitt eines Brettes der Länge n

D. h. $\ell \in \{1, \dots, n\}$ und Folge $i_1, \dots, i_\ell \in \{1, \dots, n\}$ mit $i_1 + \dots + i_\ell = n$
sodass Gesamterlös $p_{i_1} + \dots + p_{i_\ell}$ größtmöglich ist.

Länge i	1	2	3	4	5
Preis p_i	1	3	4	5	7

$$n = 5$$

2.3.1 Berechnung optimaler Zuschnitte

Zuschnittproblem

Eingabe: Länge $n \in \mathbb{N}$

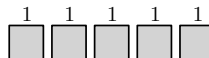
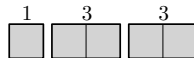
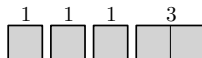
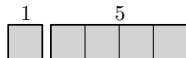
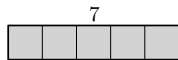
Preise $p_1, \dots, p_n \in \mathbb{N}$

Ausgabe: optimaler Zuschnitt eines Brettes der Länge n

D. h. $\ell \in \{1, \dots, n\}$ und Folge $i_1, \dots, i_\ell \in \{1, \dots, n\}$ mit $i_1 + \dots + i_\ell = n$
sodass **Gesamterlös** $p_{i_1} + \dots + p_{i_\ell}$ **größtmöglich** ist.

Länge i	1	2	3	4	5
Preis p_i	1	3	4	5	7

$n = 5$



2.3.1 Berechnung optimaler Zuschnitte

Zuschnittproblem

Eingabe: Länge $n \in \mathbb{N}$

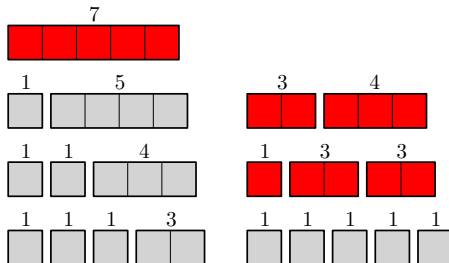
Preise $p_1, \dots, p_n \in \mathbb{N}$

Ausgabe: optimaler Zuschnitt eines Brettes der Länge n

D. h. $\ell \in \{1, \dots, n\}$ und Folge $i_1, \dots, i_\ell \in \{1, \dots, n\}$ mit $i_1 + \dots + i_\ell = n$
sodass **Gesamterlös** $p_{i_1} + \dots + p_{i_\ell}$ **größtmöglich** ist.

Länge i	1	2	3	4	5
Preis p_i	1	3	4	5	7

$n = 5$



2.3.1 Berechnung optimaler Zuschnitte

Für $i \in \{1, \dots, n\}$ sei R_i optimaler Erlös eines Brettes der Länge i .

Es gilt $R_0 = 0$ und $R_1 = p_1$.

2.3.1 Berechnung optimaler Zuschnitte

Für $i \in \{1, \dots, n\}$ sei R_i optimaler Erlös eines Brettes der Länge i .

Es gilt $R_0 = 0$ und $R_1 = p_1$.

Lemma 2.10

Für $i \in \{1, \dots, n\}$ gilt $R_i = \max\{p_j + R_{i-j} \mid j \in \{1, \dots, i\}\}$.

2.3.1 Berechnung optimaler Zuschnitte

Für $i \in \{1, \dots, n\}$ sei R_i optimaler Erlös eines Brettes der Länge i .

Es gilt $R_0 = 0$ und $R_1 = p_1$.

Lemma 2.10

Für $i \in \{1, \dots, n\}$ gilt $R_i = \max\{p_j + R_{i-j} \mid j \in \{1, \dots, i\}\}$.

Beweis:

Sei $j \in \{1, \dots, i\}$.



2.3.1 Berechnung optimaler Zuschnitte

Für $i \in \{1, \dots, n\}$ sei R_i optimaler Erlös eines Brettes der Länge i .

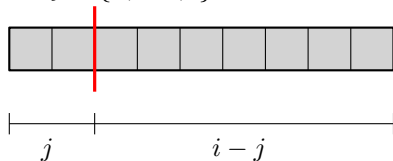
Es gilt $R_0 = 0$ und $R_1 = p_1$.

Lemma 2.10

Für $i \in \{1, \dots, n\}$ gilt $R_i = \max\{p_j + R_{i-j} \mid j \in \{1, \dots, i\}\}$.

Beweis:

Sei $j \in \{1, \dots, i\}$.



2.3.1 Berechnung optimaler Zuschnitte

Für $i \in \{1, \dots, n\}$ sei R_i optimaler Erlös eines Brettes der Länge i .

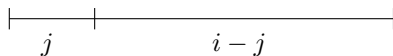
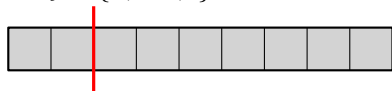
Es gilt $R_0 = 0$ und $R_1 = p_1$.

Lemma 2.10

Für $i \in \{1, \dots, n\}$ gilt $R_i = \max\{p_j + R_{i-j} \mid j \in \{1, \dots, i\}\}$.

Beweis:

Sei $j \in \{1, \dots, i\}$.



$$\Rightarrow R_i \geq \max_j (p_j + R_{i-j})$$

2.3.1 Berechnung optimaler Zuschnitte

Für $i \in \{1, \dots, n\}$ sei R_i optimaler Erlös eines Brettes der Länge i .

Es gilt $R_0 = 0$ und $R_1 = p_1$.

Lemma 2.10

Für $i \in \{1, \dots, n\}$ gilt $R_i = \max\{p_j + R_{i-j} \mid j \in \{1, \dots, i\}\}$.

Beweis:

Sei $j \in \{1, \dots, i\}$.



$$\Rightarrow R_i \geq \max_j (p_j + R_{i-j})$$

Sei i_1, \dots, i_ℓ mit $i_1 + \dots + i_\ell = i$ optimale Aufteilung.



2.3.1 Berechnung optimaler Zuschnitte

Für $i \in \{1, \dots, n\}$ sei R_i optimaler Erlös eines Brettes der Länge i .

Es gilt $R_0 = 0$ und $R_1 = p_1$.

Lemma 2.10

Für $i \in \{1, \dots, n\}$ gilt $R_i = \max\{p_j + R_{i-j} \mid j \in \{1, \dots, i\}\}$.

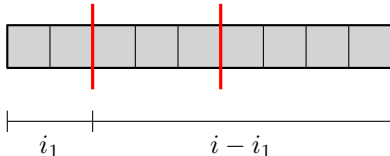
Beweis:

Sei $j \in \{1, \dots, i\}$.



$$\Rightarrow R_i \geq \max_j (p_j + R_{i-j})$$

Sei i_1, \dots, i_ℓ mit $i_1 + \dots + i_\ell = i$ optimale Aufteilung.



2.3.1 Berechnung optimaler Zuschnitte

Für $i \in \{1, \dots, n\}$ sei R_i optimaler Erlös eines Brettes der Länge i .

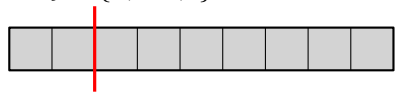
Es gilt $R_0 = 0$ und $R_1 = p_1$.

Lemma 2.10

Für $i \in \{1, \dots, n\}$ gilt $R_i = \max\{p_j + R_{i-j} \mid j \in \{1, \dots, i\}\}$.

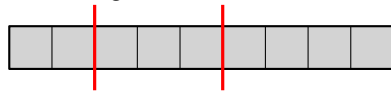
Beweis:

Sei $j \in \{1, \dots, i\}$.



$$\Rightarrow R_i \geq \max_j (p_j + R_{i-j})$$

Sei i_1, \dots, i_ℓ mit $i_1 + \dots + i_\ell = i$ optimale Aufteilung.



$$\Rightarrow R_i = p_{i_1} + \dots + p_{i_\ell} \leq p_{i_1} + R_{i-i_1}.$$

2.3.1 Berechnung optimaler Zuschnitte

Für $i \in \{1, \dots, n\}$ sei R_i optimaler Erlös eines Brettes der Länge i .

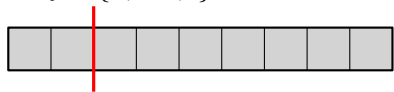
Es gilt $R_0 = 0$ und $R_1 = p_1$.

Lemma 2.10

Für $i \in \{1, \dots, n\}$ gilt $R_i = \max\{p_j + R_{i-j} \mid j \in \{1, \dots, i\}\}$.

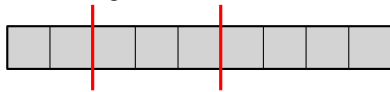
Beweis:

Sei $j \in \{1, \dots, i\}$.



$$\Rightarrow R_i \geq \max_j(p_j + R_{i-j})$$

Sei i_1, \dots, i_ℓ mit $i_1 + \dots + i_\ell = i$ optimale Aufteilung.



$$\Rightarrow R_i = p_{i_1} + \dots + p_{i_\ell} \leq p_{i_1} + R_{i-i_1}$$

$$\Rightarrow R_i \leq \max_j(p_j + R_{i-j})$$

□

2.3.1 Berechnung optimaler Zuschnitte

Lemma 2.10

Für $i \in \{1, \dots, n\}$ gilt $R_i = \max\{p_j + R_{i-j} \mid j \in \{1, \dots, i\}\}$.

```
ZUSCHNITTREK(int i)
```

```
1  if (i == 0) return 0;
```

```
2  R = -1;
```

```
3  for (j = 1; j <= i; j++)
```

```
4      R = max{R, pj + ZUSCHNITTREK(i - j)};
```

```
5  return R;
```

2.3.1 Berechnung optimaler Zuschnitte

Lemma 2.10

Für $i \in \{1, \dots, n\}$ gilt $R_i = \max\{p_j + R_{i-j} \mid j \in \{1, \dots, i\}\}$.

ZUSCHNITT(int n)

```
1   $R_0 = 0$ ;  
2  for ( $i = 1$ ;  $i \leq n$ ;  $i++$ ) {  
3       $R_i = -1$ ;  
4      for ( $j = 1$ ;  $j \leq i$ ;  $j++$ )  
5           $R_i = \max\{R_i, p_j + R_{i-j}\}$ ;  
6  }  
7  return  $R_n$ ;
```

2.3.1 Berechnung optimaler Zuschnitte

Lemma 2.10

Für $i \in \{1, \dots, n\}$ gilt $R_i = \max\{p_j + R_{i-j} \mid j \in \{1, \dots, i\}\}$.

ZUSCHNITT(int n)

```
1   $R_0 = 0$ ;  
2  for ( $i = 1$ ;  $i \leq n$ ;  $i++$ ) {  
3       $R_i = -1$ ;  
4      for ( $j = 1$ ;  $j \leq i$ ;  $j++$ )  
5           $R_i = \max\{R_i, p_j + R_{i-j}\}$ ;  
6  }  
7  return  $R_n$ ;
```

Theorem 2.11

Der Algorithmus ZUSCHNITT berechnet in Zeit $\Theta(n^2)$ den maximal erreichbaren Erlös R_n .

2.3.1 Berechnung optimaler Zuschnitte

Ermittle nicht nur den Wert, sondern auch die Lösung.

```
ZUSCHNITT2(int n)
1   $R_0 = 0$ ;
2  for ( $i = 1$ ;  $i \leq n$ ;  $i++$ ) {
3       $R_i = -1$ ;
4      for ( $j = 1$ ;  $j \leq i$ ;  $j++$ ) {
5          if ( $p_j + R_{i-j} > R_i$ ) {
6               $R_i = p_j + R_{i-j}$ ;
7               $s_i = j$ ;
8          }
9      }
10 }
11 return  $R_n$ ;
```


2.3.1 Berechnung optimaler Zuschnitte

ZUSCHNITT2(int n)

```
1   $R_0 = 0$ ;  
2  for ( $i = 1$ ;  $i \leq n$ ;  $i++$ ) {  
3       $R_i = -1$ ;  
4      for ( $j = 1$ ;  $j \leq i$ ;  $j++$ ) {  
5          if ( $p_j + R_{i-j} > R_i$ ) {  
6               $R_i = p_j + R_{i-j}$ ;  
7               $s_i = j$ ;  
8          }  
9      }  
10 }  
11 return  $R_n$ ;
```

Beispiel

Länge i	1	2	3	4	5	6	7	8	9
Preis p_i	1	3	5	5	9	10	10	11	13
s_i									
R_i									

2.3.1 Berechnung optimaler Zuschnitte

ZUSCHNITT2(int n)

```
1   $R_0 = 0$ ;  
2  for ( $i = 1$ ;  $i \leq n$ ;  $i++$ ) {  
3       $R_i = -1$ ;  
4      for ( $j = 1$ ;  $j \leq i$ ;  $j++$ ) {  
5          if ( $p_j + R_{i-j} > R_i$ ) {  
6               $R_i = p_j + R_{i-j}$ ;  
7               $s_i = j$ ;  
8          }  
9      }  
10 }  
11 return  $R_n$ ;
```

Beispiel

Länge i	1	2	3	4	5	6	7	8	9
Preis p_i	1	3	5	5	9	10	10	11	13
s_i	1								
R_i	1								



2.3.1 Berechnung optimaler Zuschnitte

ZUSCHNITT2(int n)

```
1   $R_0 = 0$ ;  
2  for ( $i = 1$ ;  $i \leq n$ ;  $i++$ ) {  
3       $R_i = -1$ ;  
4      for ( $j = 1$ ;  $j \leq i$ ;  $j++$ ) {  
5          if ( $p_j + R_{i-j} > R_i$ ) {  
6               $R_i = p_j + R_{i-j}$ ;  
7               $s_i = j$ ;  
8          }  
9      }  
10 }  
11 return  $R_n$ ;
```

Beispiel

Länge i	1	2	3	4	5	6	7	8	9
Preis p_i	1	3	5	5	9	10	10	11	13
s_i	1	2							
R_i	1	3							

Diagram illustrating the calculation of R_3 . A blue arrow points from $R_2 = 3$ to R_3 , and a red arrow points from $p_1 = 1$ to R_3 , with a red $+1$ indicating the update $R_3 = 3 + 1 = 4$.



2.3.1 Berechnung optimaler Zuschnitte

ZUSCHNITT2(int n)

```
1   $R_0 = 0$ ;  
2  for ( $i = 1$ ;  $i \leq n$ ;  $i++$ ) {  
3       $R_i = -1$ ;  
4      for ( $j = 1$ ;  $j \leq i$ ;  $j++$ ) {  
5          if ( $p_j + R_{i-j} > R_i$ ) {  
6               $R_i = p_j + R_{i-j}$ ;  
7              si = j;  
8          }  
9      }  
10 }  
11 return  $R_n$ ;
```

Beispiel

Länge i	1	2	3	4	5	6	7	8	9
Preis p_i	1	3	5	5	9	10	10	11	13
s_i	1	2	3						
R_i	1	3	5						

Diagram illustrating the calculation of optimal cuts for length 5. A blue arrow points from $R_3 = 5$ to R_5 . A red arrow points from $R_2 = 3$ to R_5 , labeled $+1$. Another red arrow points from $R_3 = 5$ to R_5 , labeled $+3$.



2.3.1 Berechnung optimaler Zuschnitte

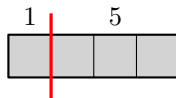
ZUSCHNITT2(int n)

```
1   $R_0 = 0$ ;  
2  for ( $i = 1$ ;  $i \leq n$ ;  $i++$ ) {  
3       $R_i = -1$ ;  
4      for ( $j = 1$ ;  $j \leq i$ ;  $j++$ ) {  
5          if ( $p_j + R_{i-j} > R_i$ ) {  
6               $R_i = p_j + R_{i-j}$ ;  
7              si = j;  
8          }  
9      }  
10 }  
11 return  $R_n$ ;
```

Beispiel

Länge i	1	2	3	4	5	6	7	8	9
Preis p_i	1	3	5	5	9	10	10	11	13
s_i	1	2	3	1					
R_i	1	3	5	6					

Diagram illustrating the calculation of R_4 from the table above. A red arrow points from $R_3 = 5$ to $R_4 = 6$ with a label $+1$. A blue arrow points from $R_1 = 1$ to $R_4 = 6$ with a label $+5$. A red arrow points from $R_2 = 3$ to $R_4 = 6$ with a label $+3$.



2.3.1 Berechnung optimaler Zuschnitte

ZUSCHNITT2(int n)

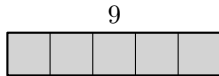
```
1   $R_0 = 0$ ;  
2  for ( $i = 1$ ;  $i \leq n$ ;  $i++$ ) {  
3       $R_i = -1$ ;  
4      for ( $j = 1$ ;  $j \leq i$ ;  $j++$ ) {  
5          if ( $p_j + R_{i-j} > R_i$ ) {  
6               $R_i = p_j + R_{i-j}$ ;  
7              si = j;  
8          }  
9      }  
10 }  
11 return  $R_n$ ;
```

Beispiel

Länge i	1	2	3	4	5	6	7	8	9
Preis p_i	1	3	5	5	9	10	10	11	13
s_i	1	2	3	1	5				
R_i	1	3	5	6	9				

Diagram illustrating the calculation of optimal cuts for $i=5$. Red arrows show the recurrence relation $R_i = p_j + R_{i-j}$ for $j=1, 2, 3$:

- $R_5 = p_1 + R_4 = 1 + 6 = 7$ (labeled +6)
- $R_5 = p_2 + R_3 = 3 + 5 = 8$ (labeled +5)
- $R_5 = p_3 + R_2 = 5 + 3 = 8$ (labeled +3)
- The optimal value is $R_5 = 9$ (labeled +1), achieved by $s_5 = 5$ (indicated by a blue arrow).



2.3.1 Berechnung optimaler Zuschnitte

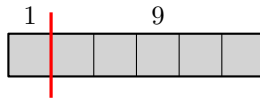
ZUSCHNITT2(int n)

```
1   $R_0 = 0$ ;  
2  for ( $i = 1$ ;  $i \leq n$ ;  $i++$ ) {  
3       $R_i = -1$ ;  
4      for ( $j = 1$ ;  $j \leq i$ ;  $j++$ ) {  
5          if ( $p_j + R_{i-j} > R_i$ ) {  
6               $R_i = p_j + R_{i-j}$ ;  
7              si = j;  
8          }  
9      }  
10 }  
11 return  $R_n$ ;
```

Beispiel

Länge i	1	2	3	4	5	6	7	8	9
Preis p_i	1	3	5	5	9	10	10	11	13
s_i	1	2	3	1	5	1			
R_i	1	3	5	6	9	10			

Diagram illustrating the calculation of optimal cuts for length 6. The table shows the values of p_i , s_i , and R_i for lengths 1 through 9. The diagram shows the calculation of R_6 by adding the price of a piece of length 1 ($p_1 = 1$) to the optimal solution for length 5 ($R_5 = 9$), resulting in $R_6 = 10$. The diagram also shows the calculation of R_5 by adding the price of a piece of length 5 ($p_5 = 9$) to the optimal solution for length 1 ($R_1 = 1$), resulting in $R_5 = 10$. The diagram also shows the calculation of R_4 by adding the price of a piece of length 1 ($p_1 = 1$) to the optimal solution for length 3 ($R_3 = 5$), resulting in $R_4 = 6$. The diagram also shows the calculation of R_3 by adding the price of a piece of length 3 ($p_3 = 5$) to the optimal solution for length 0 ($R_0 = 0$), resulting in $R_3 = 5$. The diagram also shows the calculation of R_2 by adding the price of a piece of length 2 ($p_2 = 3$) to the optimal solution for length 0 ($R_0 = 0$), resulting in $R_2 = 3$. The diagram also shows the calculation of R_1 by adding the price of a piece of length 1 ($p_1 = 1$) to the optimal solution for length 0 ($R_0 = 0$), resulting in $R_1 = 1$.



2.3.1 Berechnung optimaler Zuschnitte

ZUSCHNITT2(int n)

```
1   $R_0 = 0$ ;  
2  for ( $i = 1$ ;  $i \leq n$ ;  $i++$ ) {  
3       $R_i = -1$ ;  
4      for ( $j = 1$ ;  $j \leq i$ ;  $j++$ ) {  
5          if ( $p_j + R_{i-j} > R_i$ ) {  
6               $R_i = p_j + R_{i-j}$ ;  
7              si = j;  
8          }  
9      }  
10 }  
11 return  $R_n$ ;
```

Beispiel

Länge i	1	2	3	4	5	6	7	8	9
Preis p_i	1	3	5	5	9	10	10	11	13
s_i	1	2	3	1	5	1	2		
R_i	1	3	5	6	9	10	12		

Diagram illustrating the calculation of optimal cuts for $n=9$. Red arrows show the recurrence relation $R_i = \max_j (p_j + R_{i-j})$ for $i=7$ to 9 . Blue arrows show the optimal cut s_i for $i=7$ to 9 . The values $+10$, $+9$, $+5$, $+5$, $+3$, and $+1$ are shown below the arrows.



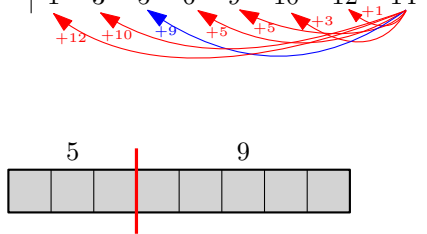
2.3.1 Berechnung optimaler Zuschnitte

ZUSCHNITT2(int n)

```
1   $R_0 = 0$ ;  
2  for ( $i = 1$ ;  $i \leq n$ ;  $i++$ ) {  
3       $R_i = -1$ ;  
4      for ( $j = 1$ ;  $j \leq i$ ;  $j++$ ) {  
5          if ( $p_j + R_{i-j} > R_i$ ) {  
6               $R_i = p_j + R_{i-j}$ ;  
7               $s_i = j$ ;  
8          }  
9      }  
10 }  
11 return  $R_n$ ;
```

Beispiel

Länge i	1	2	3	4	5	6	7	8	9
Preis p_i	1	3	5	5	9	10	10	11	13
s_i	1	2	3	1	5	1	2	3	
R_i	1	3	5	6	9	10	12	14	



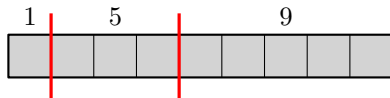
2.3.1 Berechnung optimaler Zuschnitte

ZUSCHNITT2(int n)

```
1   $R_0 = 0$ ;  
2  for ( $i = 1$ ;  $i \leq n$ ;  $i++$ ) {  
3       $R_i = -1$ ;  
4      for ( $j = 1$ ;  $j \leq i$ ;  $j++$ ) {  
5          if ( $p_j + R_{i-j} > R_i$ ) {  
6               $R_i = p_j + R_{i-j}$ ;  
7              si = j;  
8          }  
9      }  
10 }  
11 return  $R_n$ ;
```

Beispiel

Länge i	1	2	3	4	5	6	7	8	9
Preis p_i	1	3	5	5	9	10	10	11	13
s_i	1	2	3	1	5	1	2	3	1
R_i	1	3	5	6	9	10	12	14	15



2.3.2 Rucksackproblem

Rucksackproblem (Knapsack Problem (KP))

Eingabe: Nutzen $p_1, \dots, p_n \in \mathbb{N}$

Gewichte $w_1, \dots, w_n \in \mathbb{N}$

Kapazität $t \in \mathbb{N}$

Ausgabe: $x_1, \dots, x_n \in \{0, 1\}$, sodass Gesamtnutzen $p_1 x_1 + \dots + p_n x_n$ maximal unter der Bedingung $w_1 x_1 + \dots + w_n x_n \leq t$

2.3.2 Rucksackproblem

Rucksackproblem (Knapsack Problem (KP))

Eingabe: Nutzen $p_1, \dots, p_n \in \mathbb{N}$

Gewichte $w_1, \dots, w_n \in \mathbb{N}$

Kapazität $t \in \mathbb{N}$

Ausgabe: $x_1, \dots, x_n \in \{0, 1\}$, sodass Gesamtnutzen $p_1x_1 + \dots + p_nx_n$ **maximal**
unter der Bedingung $w_1x_1 + \dots + w_nx_n \leq t$

Wie sehen geeignete Teilprobleme aus?

Sei $W = \max_{i \in \{1, \dots, n\}} w_i$.

Für jede Kombination aus $i \in \{1, \dots, n\}$ und $w \in \{0, \dots, nW\}$ sei

$$P(i, w) = \max\{p_1x_1 + \dots + p_ix_i \mid w_1x_1 + \dots + w_ix_i \leq w\}.$$

2.3.2 Rucksackproblem

Rucksackproblem (Knapsack Problem (KP))

Eingabe: Nutzen $p_1, \dots, p_n \in \mathbb{N}$

Gewichte $w_1, \dots, w_n \in \mathbb{N}$

Kapazität $t \in \mathbb{N}$

Ausgabe: $x_1, \dots, x_n \in \{0, 1\}$, sodass Gesamtnutzen $p_1x_1 + \dots + p_nx_n$ **maximal**
unter der Bedingung $w_1x_1 + \dots + w_nx_n \leq t$

Wie sehen geeignete Teilprobleme aus?

Sei $W = \max_{i \in \{1, \dots, n\}} w_i$.

Für jede Kombination aus $i \in \{1, \dots, n\}$ und $w \in \{0, \dots, nW\}$ sei

$$P(i, w) = \max\{p_1x_1 + \dots + p_ix_i \mid w_1x_1 + \dots + w_ix_i \leq w\}.$$

Dies ist das Rucksackproblem gegeben durch Objekte $1, \dots, i$ und Kapazität w .

2.3.2 Rucksackproblem

Für jede Kombination aus $i \in \{1, \dots, n\}$ und $w \in \{0, \dots, nW\}$ sei

$$P(i, w) = \max\{p_1x_1 + \dots + p_ix_i \mid w_1x_1 + \dots + w_ix_i \leq w\}.$$

Randfälle:

$$P(1, w) = \begin{cases} 0 & \text{falls } w < w_1 \\ p_1 & \text{falls } w \geq w_1 \end{cases}$$

2.3.2 Rucksackproblem

Für jede Kombination aus $i \in \{1, \dots, n\}$ und $w \in \{0, \dots, nW\}$ sei

$$P(i, w) = \max\{p_1x_1 + \dots + p_ix_i \mid w_1x_1 + \dots + w_ix_i \leq w\}.$$

Randfälle:

$$P(1, w) = \begin{cases} 0 & \text{falls } w < w_1 \\ p_1 & \text{falls } w \geq w_1 \end{cases}$$

Konvention:

$$P(i, 0) = 0 \text{ und } P(i, w) = -\infty \text{ für alle } i \in \{1, \dots, n\} \text{ und } w < 0.$$

2.3.2 Rucksackproblem

Sei für ein $i \geq 2$ und für alle $w \in \{0, \dots, nW\}$ **der Wert $P(i - 1, w)$ bekannt.**

Ziel: Berechnung von $P(i, w)$ für $w \in \{0, \dots, nW\}$.

2.3.2 Rucksackproblem

Sei für ein $i \geq 2$ und für alle $w \in \{0, \dots, nW\}$ **der Wert $P(i - 1, w)$ bekannt.**

Ziel: Berechnung von $P(i, w)$ für $w \in \{0, \dots, nW\}$.

Sei $I \subseteq \{1, \dots, i\}$ mit $\sum_{i \in I} w_i \leq w$ und größtmöglichem Nutzen, d. h. $\sum_{i \in I} p_i = P(i, w)$.

2.3.2 Rucksackproblem

Sei für ein $i \geq 2$ und für alle $w \in \{0, \dots, nW\}$ **der Wert $P(i-1, w)$ bekannt.**

Ziel: Berechnung von $P(i, w)$ für $w \in \{0, \dots, nW\}$.

Sei $I \subseteq \{1, \dots, i\}$ mit $\sum_{i \in I} w_i \leq w$ und größtmöglichem Nutzen, d. h. $\sum_{i \in I} p_i = P(i, w)$.

- Falls $i \notin I$, so ist $I \subseteq \{1, \dots, i-1\}$ mit $\sum_{j \in I} w_j \leq w$.

2.3.2 Rucksackproblem

Sei für ein $i \geq 2$ und für alle $w \in \{0, \dots, nW\}$ **der Wert $P(i-1, w)$ bekannt.**

Ziel: Berechnung von $P(i, w)$ für $w \in \{0, \dots, nW\}$.

Sei $I \subseteq \{1, \dots, i\}$ mit $\sum_{i \in I} w_i \leq w$ und größtmöglichem Nutzen, d. h. $\sum_{i \in I} p_i = P(i, w)$.

- Falls $i \notin I$, so ist $I \subseteq \{1, \dots, i-1\}$ mit $\sum_{j \in I} w_j \leq w$.
 $\Rightarrow P(i, w) = P(i-1, w)$

2.3.2 Rucksackproblem

Sei für ein $i \geq 2$ und für alle $w \in \{0, \dots, nW\}$ **der Wert $P(i-1, w)$ bekannt.**

Ziel: Berechnung von $P(i, w)$ für $w \in \{0, \dots, nW\}$.

Sei $I \subseteq \{1, \dots, i\}$ mit $\sum_{i \in I} w_i \leq w$ und größtmöglichem Nutzen, d. h. $\sum_{i \in I} p_i = P(i, w)$.

- Falls $i \notin I$, so ist $I \subseteq \{1, \dots, i-1\}$ mit $\sum_{j \in I} w_j \leq w$.
 $\Rightarrow P(i, w) = P(i-1, w)$
- Falls $i \in I$, so ist $I \setminus \{i\} \subseteq \{1, \dots, i-1\}$ mit $\sum_{j \in I \setminus \{i\}} w_j \leq w - w_i$.

2.3.2 Rucksackproblem

Sei für ein $i \geq 2$ und für alle $w \in \{0, \dots, nW\}$ **der Wert $P(i-1, w)$ bekannt.**

Ziel: Berechnung von $P(i, w)$ für $w \in \{0, \dots, nW\}$.

Sei $I \subseteq \{1, \dots, i\}$ mit $\sum_{i \in I} w_i \leq w$ und größtmöglichem Nutzen, d. h. $\sum_{i \in I} p_i = P(i, w)$.

- Falls $i \notin I$, so ist $I \subseteq \{1, \dots, i-1\}$ mit $\sum_{j \in I} w_j \leq w$.
 $\Rightarrow P(i, w) = P(i-1, w)$
- Falls $i \in I$, so ist $I \setminus \{i\} \subseteq \{1, \dots, i-1\}$ mit $\sum_{j \in I \setminus \{i\}} w_j \leq w - w_i$.
 $\Rightarrow \sum_{j \in I \setminus \{i\}} p_j = P(i-1, w - w_i)$

2.3.2 Rucksackproblem

Sei für ein $i \geq 2$ und für alle $w \in \{0, \dots, nW\}$ **der Wert $P(i-1, w)$ bekannt.**

Ziel: Berechnung von $P(i, w)$ für $w \in \{0, \dots, nW\}$.

Sei $I \subseteq \{1, \dots, i\}$ mit $\sum_{i \in I} w_i \leq w$ und größtmöglichem Nutzen, d. h. $\sum_{i \in I} p_i = P(i, w)$.

- Falls $i \notin I$, so ist $I \subseteq \{1, \dots, i-1\}$ mit $\sum_{j \in I} w_j \leq w$.
 $\Rightarrow P(i, w) = P(i-1, w)$
- Falls $i \in I$, so ist $I \setminus \{i\} \subseteq \{1, \dots, i-1\}$ mit $\sum_{j \in I \setminus \{i\}} w_j \leq w - w_i$.
 $\Rightarrow \sum_{j \in I \setminus \{i\}} p_j = P(i-1, w - w_i)$
 $\Rightarrow P(i, w) = P(i-1, w - w_i) + p_i$

2.3.2 Rucksackproblem

Sei für ein $i \geq 2$ und für alle $w \in \{0, \dots, nW\}$ **der Wert $P(i-1, w)$ bekannt.**

Ziel: Berechnung von $P(i, w)$ für $w \in \{0, \dots, nW\}$.

Sei $I \subseteq \{1, \dots, i\}$ mit $\sum_{i \in I} w_i \leq w$ und größtmöglichem Nutzen, d. h. $\sum_{i \in I} p_i = P(i, w)$.

- Falls $i \notin I$, so ist $I \subseteq \{1, \dots, i-1\}$ mit $\sum_{j \in I} w_j \leq w$.
 $\Rightarrow P(i, w) = P(i-1, w)$
- Falls $i \in I$, so ist $I \setminus \{i\} \subseteq \{1, \dots, i-1\}$ mit $\sum_{j \in I \setminus \{i\}} w_j \leq w - w_i$.
 $\Rightarrow \sum_{j \in I \setminus \{i\}} p_j = P(i-1, w - w_i)$
 $\Rightarrow P(i, w) = P(i-1, w - w_i) + p_i$

Insgesamt folgt $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$.

2.3.2 Rucksackproblem

DYNKP

```
1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;
```


2.3.2 Rucksackproblem

DYNKP

```
1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;
```

Theorem 2.12

Der Algorithmus DYNKP bestimmt in Zeit $\Theta(n^2 W)$ den maximal erreichbaren Nutzen einer gegebenen Instanz des Rucksackproblems.

2.3.2 Rucksackproblem

DYNKP

1 // Sei $P(i, 0) = 0$ und $P(i, w) = -\infty$ für $i \in \{1, \dots, n\}$ und $w < 0$.

$$2 \quad W = \max_{i \in \{1, \dots, n\}} w_i;$$

```

3   for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;

```

```
4  for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
```

```
5  for ( $i = 2; i \leq n; i++$ )
```

```
6     for ( $w = 1; w \leq nW; w++$ )
```

$$P(i, w) = \max\{P(i-1, w), P(i-1, w-w_i) + p_i\};$$

```

8   return  $P(n, t)$ ;

```

[illegible]

2.3.2 Rucksackproblem

DYNKP

```

1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;

```

[illegible]

2.3.2 Rucksackproblem

DYNKP

```

1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;

```

[illegible]

2.3.2 Rucksackproblem

DYNKP

```

1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w-w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;

```

[illegible]

2.3.2 Rucksackproblem

DYNKP

```

1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;

```

[illegible]

2.3.2 Rucksackproblem

DYNKP

```

1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;

```

[illegible]

2.3.2 Rucksackproblem

DYNKP

1 // Sei $P(i, 0) = 0$ und $P(i, w) = -\infty$ für $i \in \{1, \dots, n\}$ und $w < 0$.

$$2 \quad W = \max_{i \in \{1, \dots, n\}} w_i;$$

```

3   for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;

```

```
4  for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
```

```
5  for ( $i = 2; i \leq n; i++$ )
```

```
6     for ( $w = 1; w \leq nW; w++$ )
```

$$P(i, w) = \max\{P(i-1, w), P(i-1, w-w_i) + p_i\};$$

```

8   return  $P(n, t)$ ;

```

[illegible]

2.3.2 Rucksackproblem

DYNKP

1 // Sei $P(i, 0) = 0$ und $P(i, w) = -\infty$ für $i \in \{1, \dots, n\}$ und $w < 0$.

$$2 \quad W = \max_{i \in \{1, \dots, n\}} w_i;$$

```

3   for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;

```

```
4  for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
```

```
5  for ( $i = 2; i \leq n; i++$ )
```

```
6     for ( $w = 1; w \leq nW; w++$ )
```

$$P(i, w) = \max\{P(i-1, w), P(i-1, w-w_i) + p_i\};$$

```

8   return  $P(n, t)$ ;

```

[illegible]

2.3.2 Rucksackproblem

DYNKP

1 // Sei $P(i, 0) = 0$ und $P(i, w) = -\infty$ für $i \in \{1, \dots, n\}$ und $w < 0$.

$$2 \quad W = \max_{i \in \{1, \dots, n\}} w_i;$$

```

3   for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;

```

```
4  for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
```

```
5  for ( $i = 2; i \leq n; i++$ )
```

```
6    for ( $w = 1; w \leq nW; w++$ )
```

$$P(i, w) = \max\{P(i-1, w), P(i-1, w-w_i) + p_i\};$$

```

8   return  $P(n, t)$ ;

```

[illegible]

2.3.2 Rucksackproblem

DYNKP

1 // Sei $P(i, 0) = 0$ und $P(i, w) = -\infty$ für $i \in \{1, \dots, n\}$ und $w < 0$.

$$2 \quad W = \max_{i \in \{1, \dots, n\}} w_i;$$

```

3   for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;

```

```
4  for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
```

```
5  for ( $i = 2; i \leq n; i++$ )
```

```
6    for ( $w = 1; w \leq nW; w++$ )
```

$$P(i, w) = \max\{P(i-1, w), P(i-1, w-w_i) + p_i\};$$

```

8   return  $P(n, t)$ ;

```

[illegible]

2.3.2 Rucksackproblem

DYNKP

```

1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;

```

[illegible]

2.3.2 Rucksackproblem

DYNKP

1 // Sei $P(i, 0) = 0$ und $P(i, w) = -\infty$ für $i \in \{1, \dots, n\}$ und $w < 0$.

$$2 \quad W = \max_{i \in \{1, \dots, n\}} w_i;$$

```

3   for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;

```

```
4  for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
```

```
5  for ( $i = 2; i \leq n; i++$ )
```

```
6     for ( $w = 1; w \leq nW; w++$ )
```

$$P(i, w) = \max\{P(i-1, w), P(i-1, w-w_i) + p_i\};$$

```

8   return  $P(n, t)$ ;

```

[illegible]

2.3.2 Rucksackproblem

DYNKP

1 // Sei $P(i, 0) = 0$ und $P(i, w) = -\infty$ für $i \in \{1, \dots, n\}$ und $w < 0$.

$$2 \quad W = \max_{i \in \{1, \dots, n\}} w_i;$$

```

3   for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;

```

```
4  for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
```

```
5  for ( $i = 2; i \leq n; i++$ )
```

```
6    for ( $w = 1; w \leq nW; w++$ )
```

$$P(i, w) = \max\{P(i-1, w), P(i-1, w-w_i) + p_i\};$$

```

8   return  $P(n, t)$ ;

```

[illegible]

2.3.2 Rucksackproblem

DYNKP

1 // Sei $P(i, 0) = 0$ und $P(i, w) = -\infty$ für $i \in \{1, \dots, n\}$ und $w < 0$.

$$2 \quad W = \max_{i \in \{1, \dots, n\}} w_i;$$

```

3   for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;

```

```
4  for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
```

```
5  for ( $i = 2; i \leq n; i++$ )
```

```
6    for ( $w = 1; w \leq nW; w++$ )
```

$$P(i, w) = \max\{P(i-1, w), P(i-1, w-w_i) + p_i\};$$

```

8   return  $P(n, t)$ ;

```

[illegible]

2.3.2 Rucksackproblem

DYNKP

1 // Sei $P(i, 0) = 0$ und $P(i, w) = -\infty$ für $i \in \{1, \dots, n\}$ und $w < 0$.

$$2 \quad W = \max_{i \in \{1, \dots, n\}} w_i;$$

```

3   for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;

```

```
4  for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
```

```
5  for ( $i = 2; i \leq n; i++$ )
```

```
6    for ( $w = 1; w \leq nW; w++$ )
```

$$P(i, w) = \max\{P(i-1, w), P(i-1, w-w_i) + p_i\};$$

```

8   return  $P(n, t)$ ;

```

[illegible]

2.3.2 Rucksackproblem

DYNKP

```
1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;
```

			w													
p_i	w_i	i	< 0	0	1	2	3	4	5	6	7	8	9	10	11	12
2	3	1	$-\infty$	0	0	0	2	2	2	2	2	2	2	2	2	2
1	2	2	$-\infty$	0	0	1	2	2	3	3	3	3	3	3	3	3
3	4	3	$-\infty$	0	0	1										

2.3.2 Rucksackproblem

DYNKP

```
1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;
```

			w													
p_i	w_i	i	< 0	0	1	2	3	4	5	6	7	8	9	10	11	12
2	3	1	$-\infty$	0	0	0	2	2	2	2	2	2	2	2	2	2
1	2	2	$-\infty$	0	0	1	2	2	3	3	3	3	3	3	3	3
3	4	3	$-\infty$	0	0	1	2									

2.3.2 Rucksackproblem

DYNKP

```
1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;
```

			w													
p_i	w_i	i	< 0	0	1	2	3	4	5	6	7	8	9	10	11	12
2	3	1	$-\infty$	0	0	0	2	2	2	2	2	2	2	2	2	2
1	2	2	$-\infty$	0	0	1	2	2	3	3	3	3	3	3	3	3
3	4	3	$-\infty$	0	0	1	2	3								

2.3.2 Rucksackproblem

DYNKP

```

1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;

```

			w													
p_i	w_i	i	< 0	0	1	2	3	4	5	6	7	8	9	10	11	12
2	3	1	$-\infty$	0	0	0	2	2	2	2	2	2	2	2	2	2
1	2	2	$-\infty$	0	0	1	2	2	3	3	3	3	3	3	3	3
3	4	3	$-\infty$	0	0	1	2	3	3							

2.3.2 Rucksackproblem

DYNKP

```

1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;

```

			w													
p_i	w_i	i	< 0	0	1	2	3	4	5	6	7	8	9	10	11	12
2	3	1	$-\infty$	0	0	0	2	2	2	2	2	2	2	2	2	2
1	2	2	$-\infty$	0	0	1	2	2	3	3	3	3	3	3	3	3
3	4	3	$-\infty$	0	0	1	2	3	3	4						

2.3.2 Rucksackproblem

DYNKP

```
1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;
```

			w													
p_i	w_i	i	< 0	0	1	2	3	4	5	6	7	8	9	10	11	12
2	3	1	$-\infty$	0	0	0	2	2	2	2	2	2	2	2	2	2
1	2	2	$-\infty$	0	0	1	2	2	3	3	3	3	3	3	3	3
3	4	3	$-\infty$	0	0	1	2	3	3	4	5					

2.3.2 Rucksackproblem

DYNKP

```
1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;
```

			w													
p_i	w_i	i	< 0	0	1	2	3	4	5	6	7	8	9	10	11	12
2	3	1	$-\infty$	0	0	0	2	2	2	2	2	2	2	2	2	2
1	2	2	$-\infty$	0	0	1	2	2	3	3	3	3	3	3	3	3
3	4	3	$-\infty$	0	0	1	2	3	3	4	5	5				

2.3.2 Rucksackproblem

DYNKP

```
1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;
```

			w													
p_i	w_i	i	< 0	0	1	2	3	4	5	6	7	8	9	10	11	12
2	3	1	$-\infty$	0	0	0	2	2	2	2	2	2	2	2	2	2
1	2	2	$-\infty$	0	0	1	2	2	3	3	3	3	3	3	3	3
3	4	3	$-\infty$	0	0	1	2	3	3	4	5	5	6			

2.3.2 Rucksackproblem

DYNKP

```
1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;
```

			w													
p_i	w_i	i	< 0	0	1	2	3	4	5	6	7	8	9	10	11	12
2	3	1	$-\infty$	0	0	0	2	2	2	2	2	2	2	2	2	2
1	2	2	$-\infty$	0	0	1	2	2	3	3	3	3	3	3	3	3
3	4	3	$-\infty$	0	0	1	2	3	3	4	5	5	6	6		

2.3.2 Rucksackproblem

DYNKP

```
1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;
```

			w													
p_i	w_i	i	< 0	0	1	2	3	4	5	6	7	8	9	10	11	12
2	3	1	$-\infty$	0	0	0	2	2	2	2	2	2	2	2	2	2
1	2	2	$-\infty$	0	0	1	2	2	3	3	3	3	3	3	3	3
3	4	3	$-\infty$	0	0	1	2	3	3	4	5	5	6	6	6	6

2.3.2 Rucksackproblem

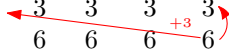
DYNKP

```

1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;

```

			w													
p_i	w_i	i	< 0	0	1	2	3	4	5	6	7	8	9	10	11	12
2	3	1	$-\infty$	0	0	0	2	2	2	2	2	2	2	2	2	2
1	2	2	$-\infty$	0	0	1	2	2	3	3	3	3	3	3	3	3
3	4	3	$-\infty$	0	0	1	2	3	3	4	5	5	6	6	6	6



2.3.2 Rucksackproblem

DYNKP

```
1 // Sei  $P(i, 0) = 0$  und  $P(i, w) = -\infty$  für  $i \in \{1, \dots, n\}$  und  $w < 0$ .
2  $W = \max_{i \in \{1, \dots, n\}} w_i$ ;
3 for ( $w = 1$ ;  $w < w_1$ ;  $w++$ )  $P(1, w) = 0$ ;
4 for ( $w = w_1$ ;  $w \leq nW$ ;  $w++$ )  $P(1, w) = p_1$ ;
5 for ( $i = 2$ ;  $i \leq n$ ;  $i++$ )
6     for ( $w = 1$ ;  $w \leq nW$ ;  $w++$ )
7          $P(i, w) = \max\{P(i-1, w), P(i-1, w - w_i) + p_i\}$ ;
8 return  $P(n, t)$ ;
```

			w													
p_i	w_i	i	< 0	0	1	2	3	4	5	6	7	8	9	10	11	12
2	3	1	$-\infty$	0	0	0	2	2	2	2	2	2	2	2	2	2
1	2	2	$-\infty$	0	0	1	2	2	3	3	3	3	3	3	3	3
3	4	3	$-\infty$	0	0	1	2	3	3	4	5	5	6	6	6	6

Beispiel: Kapazität $t = 8$ $P(3, 8) = 5$ optimale Lösung $\{1, 3\}$

2.3.2 Rucksackproblem

Kann der Algorithmus DYNKP als effizient angesehen werden?

Laufzeit von DYNKP beträgt $\Theta(n^2 W)$.

2.3.2 Rucksackproblem

Kann der Algorithmus DYNKP als effizient angesehen werden?

Laufzeit von DYNKP beträgt $\Theta(n^2 W)$.

Unterschied zu Laufzeiten der bisherigen Algorithmen: Die Laufzeit hängt **nicht nur von der Anzahl der Objekte, sondern zusätzlich von dem maximalen Gewicht W ab.**

2.3.2 Rucksackproblem

Kann der Algorithmus DYNKP als effizient angesehen werden?

Laufzeit von DYNKP beträgt $\Theta(n^2 W)$.

Unterschied zu Laufzeiten der bisherigen Algorithmen: Die Laufzeit hängt **nicht nur von der Anzahl der Objekte, sondern zusätzlich von dem maximalen Gewicht W ab.**

⇒ Bereits Eingaben, die sich mit wenigen Bytes codieren lassen, können zu einer sehr großen Laufzeit führen.