

2.2 Greedy-Algorithmen

Greedy-Algorithmen

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

Ein **Greedy-Algorithmus** löst \mathcal{I} **schrittweise** und trifft in jedem Schritt eine Entscheidung, die **für den aktuellen Schritt optimal** ist.

2.2 Greedy-Algorithmen

Greedy-Algorithmen

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

Ein **Greedy-Algorithmus** löst \mathcal{I} **schrittweise** und trifft in jedem Schritt eine Entscheidung, die **für den aktuellen Schritt optimal** ist.

Wechselgeldproblem

Eingabe: $z \in \mathbb{N}$

Ziel: Setze z Cent mit **möglichst wenigen Münzen** zusammen.

Dafür stehen beliebig viele Münzen mit den Werten 1, 2, 5, 10, 20, 50, 100 und 200 Cent zur Verfügung.

2.2 Greedy-Algorithmen

Greedy-Algorithmen

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

Ein **Greedy-Algorithmus** löst \mathcal{I} **schrittweise** und trifft in jedem Schritt eine Entscheidung, die **für den aktuellen Schritt optimal** ist.

Wechselgeldproblem

Eingabe: $z \in \mathbb{N}$

Ziel: Setze z Cent mit **möglichst wenigen Münzen** zusammen.

Dafür stehen beliebig viele Münzen mit den Werten 1, 2, 5, 10, 20, 50, 100 und 200 Cent zur Verfügung.

Beispiel: $z = 149$



2.2 Greedy-Algorithmen

GREEDYCHANGE(int z)

```
1   $M = \{1, 2, 5, 10, 20, 50, 100, 200\};$   
2   $S = ();$   
3  while ( $z > 0$ ) {  
4       $x = \max\{i \in M \mid i \leq z\};$   
5       $S.append(x);$   
6       $z = z - x;$   
7  }  
8  return  $S;$ 
```

2.2 Greedy-Algorithmen

GREEDYCHANGE(int z)

```
1   $M = \{1, 2, 5, 10, 20, 50, 100, 200\};$   
2   $S = ();$   
3  while ( $z > 0$ ) {  
4       $x = \max\{i \in M \mid i \leq z\};$   
5       $S.append(x);$   
6       $z = z - x;$   
7  }  
8  return  $S;$ 
```

Beispiel: $z = 149$



2.2 Greedy-Algorithmen

GREEDYCHANGE(int z)

```
1   $M = \{1, 2, 5, 10, 20, 50, 100, 200\};$   
2   $S = ();$   
3  while ( $z > 0$ ) {  
4       $x = \max\{i \in M \mid i \leq z\};$   
5       $S.append(x);$   
6       $z = z - x;$   
7  }  
8  return  $S;$ 
```

Beispiel: $z = 149$



Beispiel: $z = 39$



2.2 Greedy-Algorithmen

Theorem 2.4

Für $M = \{1, 2, 5, 10, 20, 50, 100, 200\}$ findet GREEDYCHANGE für jeden Betrag eine Lösung mit der kleinstmöglichen Anzahl an Münzen.

2.2 Greedy-Algorithmen

Theorem 2.4

Für $M = \{1, 2, 5, 10, 20, 50, 100, 200\}$ findet GREEDYCHANGE für jeden Betrag eine Lösung mit der kleinstmöglichen Anzahl an Münzen.

Idee:

- Finde **Struktur der Lösung**, die der Greedy-Algorithmus berechnet.
- Zeige, dass die Struktur eine **eindeutige Lösung** erzeugt.
- Zeige, dass jede **optimale Lösung** diese Struktur hat.

2.2 Greedy-Algorithmen

Behauptung

Für $i \in M$ sei x_i die Anzahl an Münzen mit Wert i in einer Lösung S . Für $i > 1$ gelte

$$\sum_{j \in M, j < i} jx_j < i.$$

Mit $\sum_{i \in M} ix_i = z$ bestimmen diese Ungleichungen eindeutig die Lösung S .

2.2 Greedy-Algorithmen

Behauptung

Für $i \in M$ sei x_i die Anzahl an Münzen mit Wert i in einer Lösung S . Für $i > 1$ gelte

$$\sum_{j \in M, j < i} jx_j < i.$$

Mit $\sum_{i \in M} ix_i = z$ bestimmen diese Ungleichungen eindeutig die Lösung S .

Beweis: (Induktion über z)

2.2 Greedy-Algorithmen

Behauptung

Für $i \in M$ sei x_i die Anzahl an Münzen mit Wert i in einer Lösung S . Für $i > 1$ gelte

$$\sum_{j \in M, j < i} jx_j < i.$$

Mit $\sum_{i \in M} ix_i = z$ bestimmen diese Ungleichungen eindeutig die Lösung S .

Beweis: (Induktion über z)

$z = 1$: Lösung $x_1 = 1$ und $x_i = 0$ für $i > 1$ ist eindeutig.

2.2 Greedy-Algorithmen

Behauptung

Für $i \in M$ sei x_i die Anzahl an Münzen mit Wert i in einer Lösung S . Für $i > 1$ gelte

$$\sum_{j \in M, j < i} jx_j < i.$$

Mit $\sum_{i \in M} ix_i = z$ bestimmen diese Ungleichungen eindeutig die Lösung S .

Beweis: (Induktion über z)

$z = 1$: Lösung $x_1 = 1$ und $x_i = 0$ für $i > 1$ ist eindeutig.

$z > 1$: **Sei $i \in M$ maximal, sodass $i \leq z$.**

2.2 Greedy-Algorithmen

Behauptung

Für $i \in M$ sei x_i die Anzahl an Münzen mit Wert i in einer Lösung S . Für $i > 1$ gelte

$$\sum_{j \in M, j < i} jx_j < i.$$

Mit $\sum_{i \in M} ix_i = z$ bestimmen diese Ungleichungen eindeutig die Lösung S .

Beweis: (Induktion über z)

$z = 1$: Lösung $x_1 = 1$ und $x_i = 0$ für $i > 1$ ist eindeutig.

$z > 1$: **Sei $i \in M$ maximal, sodass $i \leq z$.**

Wegen $\sum_{j \in M, j < i} jx_j < i$ gilt **$x_i \geq 1$** . Wert i kommt also mindestens einmal vor.

2.2 Greedy-Algorithmen

Behauptung

Für $i \in M$ sei x_i die Anzahl an Münzen mit Wert i in einer Lösung S . Für $i > 1$ gelte

$$\sum_{j \in M, j < i} jx_j < i.$$

Mit $\sum_{i \in M} ix_i = z$ bestimmen diese Ungleichungen eindeutig die Lösung S .

Beweis: (Induktion über z)

$z = 1$: Lösung $x_1 = 1$ und $x_i = 0$ für $i > 1$ ist eindeutig.

$z > 1$: **Sei $i \in M$ maximal, sodass $i \leq z$.**

Wegen $\sum_{j \in M, j < i} jx_j < i$ gilt **$x_i \geq 1$** . Wert i kommt also mindestens einmal vor.

Sei $(x'_i)_{i \in M}$ eine Lösung für **Restbetrag $z' = z - i$** , welche die Ungleichungen erfüllt.

2.2 Greedy-Algorithmen

Behauptung

Für $i \in M$ sei x_i die Anzahl an Münzen mit Wert i in einer Lösung S . Für $i > 1$ gelte

$$\sum_{j \in M, j < i} jx_j < i.$$

Mit $\sum_{i \in M} ix_i = z$ bestimmen diese Ungleichungen eindeutig die Lösung S .

Beweis: (Induktion über z)

$z = 1$: Lösung $x_1 = 1$ und $x_i = 0$ für $i > 1$ ist eindeutig.

$z > 1$: **Sei $i \in M$ maximal, sodass $i \leq z$.**

Wegen $\sum_{j \in M, j < i} jx_j < i$ gilt **$x_i \geq 1$** . Wert i kommt also mindestens einmal vor.

Sei $(x'_i)_{i \in M}$ eine Lösung für **Restbetrag $z' = z - i$** , welche die Ungleichungen erfüllt.

Laut **Induktionsannahme** ist die Lösung für **z' eindeutig**.

2.2 Greedy-Algorithmen

Behauptung

Für $i \in M$ sei x_i die Anzahl an Münzen mit Wert i in einer Lösung S . Für $i > 1$ gelte

$$\sum_{j \in M, j < i} jx_j < i.$$

Mit $\sum_{i \in M} ix_i = z$ bestimmen diese Ungleichungen eindeutig die Lösung S .

Beweis: (Induktion über z)

$z = 1$: Lösung $x_1 = 1$ und $x_i = 0$ für $i > 1$ ist eindeutig.

$z > 1$: **Sei $i \in M$ maximal, sodass $i \leq z$.**

Wegen $\sum_{j \in M, j < i} jx_j < i$ gilt **$x_i \geq 1$** . Wert i kommt also mindestens einmal vor.

Sei $(x'_i)_{i \in M}$ eine Lösung für **Restbetrag $z' = z - i$** , welche die Ungleichungen erfüllt.

Laut **Induktionsannahme** ist die Lösung für **z' eindeutig**.

Dann ist $x_j = x'_j$ für $i \neq j$ und $x_i = x'_i + 1$ als Lösung für z eindeutig. □

2.2 Greedy-Algorithmen

Behauptung

Sei $(y_i \in \mathbb{N}_0)_{i \in M}$ optimale Lösung, d. h. $\sum_{i \in M} i y_i = z$ und $\sum_{i \in M} y_i$ kleinstmöglich.
Für $i \in M$ mit $i > 1$ gilt $\sum_{j \in M, j < i} j y_j < i$.

Beweis:

$i = 2$: **Es gilt $y_1 \leq 1$.**

Sonst $2 \times 1\text{-Cent} \rightarrow 1 \times 2\text{-Cent}$.

2.2 Greedy-Algorithmen

Behauptung

Sei $(y_i \in \mathbb{N}_0)_{i \in M}$ optimale Lösung, d. h. $\sum_{i \in M} i y_i = z$ und $\sum_{i \in M} y_i$ kleinstmöglich.

Für $i \in M$ mit $i > 1$ gilt $\sum_{j \in M, j < i} j y_j < i$.

Beweis:

$i = 2$: Es gilt $y_1 \leq 1$.

Sonst $2 \times 1\text{-Cent} \rightarrow 1 \times 2\text{-Cent}$.

$$\Rightarrow 1 \cdot y_1 < 2$$

2.2 Greedy-Algorithmen

Behauptung

Sei $(y_i \in \mathbb{N}_0)_{i \in M}$ optimale Lösung, d. h. $\sum_{i \in M} i y_i = z$ und $\sum_{i \in M} y_i$ kleinstmöglich.
Für $i \in M$ mit $i > 1$ gilt $\sum_{j \in M, j < i} j y_j < i$.

Beweis:

$i = 2$: **Es gilt $y_1 \leq 1$.**

Sonst $2 \times 1\text{-Cent} \rightarrow 1 \times 2\text{-Cent}$.

$$\Rightarrow 1 \cdot y_1 < 2$$

$i = 5$: **Es gilt $y_2 \leq 2$.**

Sonst $3 \times 2\text{-Cent} \rightarrow 1 \times 5\text{-Cent} + 1 \times 1\text{-Cent}$.

2.2 Greedy-Algorithmen

Behauptung

Sei $(y_i \in \mathbb{N}_0)_{i \in M}$ optimale Lösung, d. h. $\sum_{i \in M} i y_i = z$ und $\sum_{i \in M} y_i$ kleinstmöglich.

Für $i \in M$ mit $i > 1$ gilt $\sum_{j \in M, j < i} j y_j < i$.

Beweis:

$i = 2$: **Es gilt $y_1 \leq 1$.**

Sonst $2 \times 1\text{-Cent} \rightarrow 1 \times 2\text{-Cent}$.

$$\Rightarrow 1 \cdot y_1 < 2$$

$i = 5$: **Es gilt $y_2 \leq 2$.**

Sonst $3 \times 2\text{-Cent} \rightarrow 1 \times 5\text{-Cent} + 1 \times 1\text{-Cent}$.

Nicht gleichzeitig $y_1 = 1$ und $y_2 = 2$.

Sonst $1 \times 1\text{-Cent} + 2 \times 2\text{-Cent} \rightarrow 1 \times 5\text{-Cent}$.

2.2 Greedy-Algorithmen

Behauptung

Sei $(y_i \in \mathbb{N}_0)_{i \in M}$ optimale Lösung, d. h. $\sum_{i \in M} i y_i = z$ und $\sum_{i \in M} y_i$ kleinstmöglich.
Für $i \in M$ mit $i > 1$ gilt $\sum_{j \in M, j < i} j y_j < i$.

Beweis:

$i = 2$: **Es gilt $y_1 \leq 1$.**

Sonst $2 \times 1\text{-Cent} \rightarrow 1 \times 2\text{-Cent}$.

$$\Rightarrow 1 \cdot y_1 < 2$$

$i = 5$: **Es gilt $y_2 \leq 2$.**

Sonst $3 \times 2\text{-Cent} \rightarrow 1 \times 5\text{-Cent} + 1 \times 1\text{-Cent}$.

Nicht gleichzeitig $y_1 = 1$ und $y_2 = 2$.

Sonst $1 \times 1\text{-Cent} + 2 \times 2\text{-Cent} \rightarrow 1 \times 5\text{-Cent}$.

$$\Rightarrow 1 \cdot y_1 + 2 \cdot y_2 < 5$$

2.2 Greedy-Algorithmen

Behauptung

Sei $(y_i \in \mathbb{N}_0)_{i \in M}$ optimale Lösung, d. h. $\sum_{i \in M} i y_i = z$ und $\sum_{i \in M} y_i$ kleinstmöglich.
Für $i \in M$ mit $i > 1$ gilt $\sum_{j \in M, j < i} j y_j < i$.

Beweis:

$i = 2$: **Es gilt $y_1 \leq 1$.**

Sonst $2 \times 1\text{-Cent} \rightarrow 1 \times 2\text{-Cent}$.

$$\Rightarrow 1 \cdot y_1 < 2$$

$i = 5$: **Es gilt $y_2 \leq 2$.**

Sonst $3 \times 2\text{-Cent} \rightarrow 1 \times 5\text{-Cent} + 1 \times 1\text{-Cent}$.

Nicht gleichzeitig $y_1 = 1$ und $y_2 = 2$.

Sonst $1 \times 1\text{-Cent} + 2 \times 2\text{-Cent} \rightarrow 1 \times 5\text{-Cent}$.

$$\Rightarrow 1 \cdot y_1 + 2 \cdot y_2 < 5$$

$i = 10$: **Es gilt $y_5 \leq 1$**



2.2 Greedy-Algorithmen

Theorem 2.4

Für $M = \{1, 2, 5, 10, 20, 50, 100, 200\}$ findet GREEDYCHANGE für jeden Betrag eine Lösung mit der kleinstmöglichen Anzahl an Münzen.

2.2 Greedy-Algorithmen

Theorem 2.4

Für $M = \{1, 2, 5, 10, 20, 50, 100, 200\}$ findet GREEDYCHANGE für jeden Betrag eine Lösung mit der kleinstmöglichen Anzahl an Münzen.

Beweis:

- Die Greedy-Lösung $(x_i)_{i \in M}$ erfüllt die Ungleichungen

$$\sum_{j \in M, j < i} jx_j < i \quad \text{und} \quad \sum_{i \in M} ix_i = z$$

2.2 Greedy-Algorithmen

Theorem 2.4

Für $M = \{1, 2, 5, 10, 20, 50, 100, 200\}$ findet GREEDYCHANGE für jeden Betrag eine Lösung mit der kleinstmöglichen Anzahl an Münzen.

Beweis:

- Die Greedy-Lösung $(x_i)_{i \in M}$ erfüllt die Ungleichungen

$$\sum_{j \in M, j < i} jx_j < i \quad \text{und} \quad \sum_{i \in M} ix_i = z$$

- Diese Ungleichungen bestimmen eindeutig die Lösung $(x_i)_{i \in M}$

2.2 Greedy-Algorithmen

Theorem 2.4

Für $M = \{1, 2, 5, 10, 20, 50, 100, 200\}$ findet GREEDYCHANGE für jeden Betrag eine Lösung mit der kleinstmöglichen Anzahl an Münzen.

Beweis:

- Die Greedy-Lösung $(x_i)_{i \in M}$ erfüllt die Ungleichungen

$$\sum_{j \in M, j < i} jx_j < i \quad \text{und} \quad \sum_{i \in M} ix_i = z$$

- Diese Ungleichungen bestimmen eindeutig die Lösung $(x_i)_{i \in M}$
- Jede optimale Lösung $(y_i \in \mathbb{N}_{\geq 0})_{i \in M}$ erfüllt auch die obigen Ungleichungen.

2.2 Greedy-Algorithmen

Theorem 2.4

Für $M = \{1, 2, 5, 10, 20, 50, 100, 200\}$ findet GREEDYCHANGE für jeden Betrag eine Lösung mit der kleinstmöglichen Anzahl an Münzen.

Beweis:

- Die Greedy-Lösung $(x_i)_{i \in M}$ erfüllt die Ungleichungen

$$\sum_{j \in M, j < i} jx_j < i \quad \text{und} \quad \sum_{i \in M} ix_i = z$$

- Diese Ungleichungen bestimmen eindeutig die Lösung $(x_i)_{i \in M}$
- Jede optimale Lösung $(y_i \in \mathbb{N}_{\geq 0})_{i \in M}$ erfüllt auch die obigen Ungleichungen.

$\Rightarrow (y_i)_{i \in M}$ entspricht der Greedy-Lösung $(x_i)_{i \in M}$.



2.2 Greedy-Algorithmen

Greedy ist nicht für jedes Münzsystem optimal:

Sei $M = \{1, 3, 4\}$ und $z = 6$.

Greedy: $4 + 1 + 1$ (3 Münzen)

Optimal: $3 + 3$ (2 Münzen)

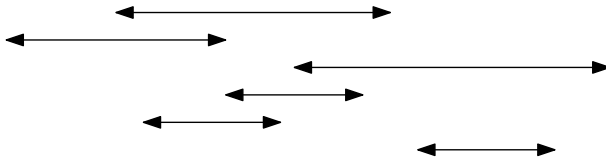
2.2.1 Optimale Auswahl von Aufgaben

Interval Scheduling

Eingabe: Menge $S = \{1, \dots, n\}$ von **Aufgaben**

Startzeitpunkte $s_1, \dots, s_n \geq 0$

Fertigstellungszeitpunkte $f_1, \dots, f_n \geq 0$ mit $f_i > s_i$



2.2.1 Optimale Auswahl von Aufgaben

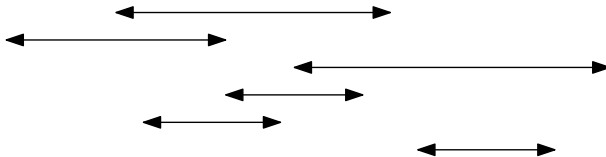
Interval Scheduling

Eingabe: Menge $S = \{1, \dots, n\}$ von **Aufgaben**

Startzeitpunkte $s_1, \dots, s_n \geq 0$

Fertigstellungszeitpunkte $f_1, \dots, f_n \geq 0$ mit $f_i > s_i$

Ausgabe: **größtmögliche Teilmenge** $S' \subseteq S$ von **disjunkten Aufgaben**, d. h.
 $\forall i, j \in S', i \neq j : [s_i, f_i) \cap [s_j, f_j) = \emptyset$



2.2.1 Optimale Auswahl von Aufgaben

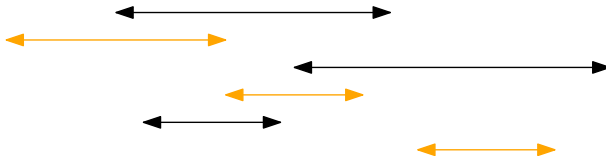
Interval Scheduling

Eingabe: Menge $S = \{1, \dots, n\}$ von **Aufgaben**

Startzeitpunkte $s_1, \dots, s_n \geq 0$

Fertigstellungszeitpunkte $f_1, \dots, f_n \geq 0$ mit $f_i > s_i$

Ausgabe: **größtmögliche Teilmenge** $S' \subseteq S$ von **disjunkten Aufgaben**, d. h.
 $\forall i, j \in S', i \neq j : [s_i, f_i) \cap [s_j, f_j) = \emptyset$



2.2.1 Optimale Auswahl von Aufgaben

GREEDYSTART

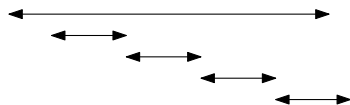
```
1   $S^* = \emptyset$ ;  
2  while ( $S \neq \emptyset$ ) {  
3    Wähle Aufgabe  $i \in S$  mit  
      kleinstem Startzeitpunkt  $s_i$ .  
4     $S^* = S^* \cup \{i\}$ ;  
5    Lösche alle Aufgaben aus  $S$ ,  
      die mit  $i$  kollidieren.  
6  }  
7  return  $S^*$ ;
```


2.2.1 Optimale Auswahl von Aufgaben

GREEDYSTART

```
1   $S^* = \emptyset$ ;  
2  while ( $S \neq \emptyset$ ) {  
3    Wähle Aufgabe  $i \in S$  mit  
      kleinstem Startzeitpunkt  $s_i$ .  
4     $S^* = S^* \cup \{i\}$ ;  
5    Lösche alle Aufgaben aus  $S$ ,  
      die mit  $i$  kollidieren.  
6  }  
7  return  $S^*$ ;
```

im Allgemeinen nicht optimal



2.2.1 Optimale Auswahl von Aufgaben

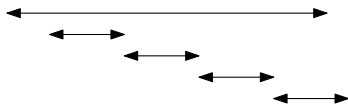
GREEDYSTART

```
1   $S^* = \emptyset$ ;  
2  while ( $S \neq \emptyset$ ) {  
3    Wähle Aufgabe  $i \in S$  mit  
      kleinstem Startzeitpunkt  $s_i$ .  
4     $S^* = S^* \cup \{i\}$ ;  
5    Lösche alle Aufgaben aus  $S$ ,  
      die mit  $i$  kollidieren.  
6  }  
7  return  $S^*$ ;
```

GREEDYDAUER

```
1   $S^* = \emptyset$ ;  
2  while ( $S \neq \emptyset$ ) {  
3    Wähle Aufgabe  $i \in S$  mit der  
      kürzesten Dauer  $f_i - s_i$ .  
4     $S^* = S^* \cup \{i\}$ ;  
5    Lösche alle Aufgaben aus  $S$ ,  
      die mit  $i$  kollidieren.  
6  }  
7  return  $S^*$ ;
```

im Allgemeinen nicht optimal



2.2.1 Optimale Auswahl von Aufgaben

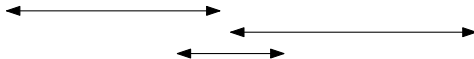
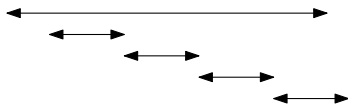
GREEDYSTART

```
1   $S^* = \emptyset$ ;  
2  while ( $S \neq \emptyset$ ) {  
3    Wähle Aufgabe  $i \in S$  mit  
      kleinstem Startzeitpunkt  $s_i$ .  
4     $S^* = S^* \cup \{i\}$ ;  
5    Lösche alle Aufgaben aus  $S$ ,  
      die mit  $i$  kollidieren.  
6  }  
7  return  $S^*$ ;
```

GREEDYDAUER

```
1   $S^* = \emptyset$ ;  
2  while ( $S \neq \emptyset$ ) {  
3    Wähle Aufgabe  $i \in S$  mit der  
      kürzesten Dauer  $f_i - s_i$ .  
4     $S^* = S^* \cup \{i\}$ ;  
5    Lösche alle Aufgaben aus  $S$ ,  
      die mit  $i$  kollidieren.  
6  }  
7  return  $S^*$ ;
```

im Allgemeinen nicht optimal



2.2.1 Optimale Auswahl von Aufgaben

GREEDYENDE

```
1   $S^* = \emptyset$ ;  
2  while ( $S \neq \emptyset$ ) {  
3      Wähle Aufgabe  $i \in S$  mit dem frühesten Fertigstellungszeitpunkt  $f_i$ .  
4       $S^* = S^* \cup \{i\}$ ;  
5      Lösche alle Aufgaben aus  $S$ , die mit  $i$  kollidieren.  
6  }  
7  return  $S^*$ ;
```

2.2.1 Optimale Auswahl von Aufgaben

GREEDYENDE

```
1   $S^* = \emptyset$ ;  
2  while ( $S \neq \emptyset$ ) {  
3      Wähle Aufgabe  $i \in S$  mit dem frühesten Fertigstellungszeitpunkt  $f_i$ .  
4       $S^* = S^* \cup \{i\}$ ;  
5      Lösche alle Aufgaben aus  $S$ , die mit  $i$  kollidieren.  
6  }  
7  return  $S^*$ ;
```

Theorem 2.6

Der Algorithmus GREEDYENDE wählt für jede Instanz eine größtmögliche Menge von paarweise nicht kollidierenden Aufgaben aus.

2.2.1 Optimale Auswahl von Aufgaben

Lemma 2.5

Es sei S eine Menge von Aufgaben und es sei $i \in S$ eine Aufgabe mit dem **frühesten Fertigstellungszeitpunkt** f_i . Dann gibt es eine optimale Auswahl $S' \subseteq S$ von paarweise nicht kollidierenden Aufgaben mit $i \in S'$.

2.2.1 Optimale Auswahl von Aufgaben

Lemma 2.5

Es sei S eine Menge von Aufgaben und es sei $i \in S$ eine Aufgabe mit dem **frühesten Fertigstellungszeitpunkt** f_i . Dann gibt es eine optimale Auswahl $S' \subseteq S$ von paarweise nicht kollidierenden Aufgaben mit $i \in S'$.

Beweis:

Sei $S^* \subseteq S$ **optimale Auswahl**.

Annahme: $i \notin S^*$.

2.2.1 Optimale Auswahl von Aufgaben

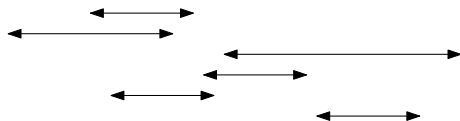
Lemma 2.5

Es sei S eine Menge von Aufgaben und es sei $i \in S$ eine Aufgabe mit dem **frühesten Fertigstellungszeitpunkt** f_i . Dann gibt es eine optimale Auswahl $S' \subseteq S$ von paarweise nicht kollidierenden Aufgaben mit $i \in S'$.

Beweis:

Sei $S^* \subseteq S$ **optimale Auswahl**.

Annahme: $i \notin S^*$.



2.2.1 Optimale Auswahl von Aufgaben

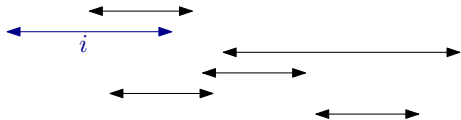
Lemma 2.5

Es sei S eine Menge von Aufgaben und es sei $i \in S$ eine Aufgabe mit dem **frühesten Fertigstellungszeitpunkt** f_i . Dann gibt es eine optimale Auswahl $S' \subseteq S$ von paarweise nicht kollidierenden Aufgaben mit $i \in S'$.

Beweis:

Sei $S^* \subseteq S$ **optimale Auswahl**.

Annahme: $i \notin S^*$.



2.2.1 Optimale Auswahl von Aufgaben

Lemma 2.5

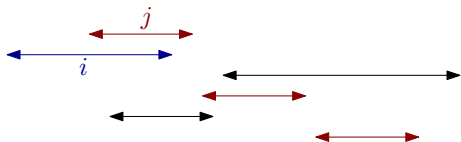
Es sei S eine Menge von Aufgaben und es sei $i \in S$ eine Aufgabe mit dem **frühesten Fertigstellungszeitpunkt** f_i . Dann gibt es eine optimale Auswahl $S' \subseteq S$ von paarweise nicht kollidierenden Aufgaben mit $i \in S'$.

Beweis:

Sei $S^* \subseteq S$ **optimale Auswahl**.

Annahme: $i \notin S^*$.

$$j = \min_{k \in S^*} f_k$$



2.2.1 Optimale Auswahl von Aufgaben

Lemma 2.5

Es sei S eine Menge von Aufgaben und es sei $i \in S$ eine Aufgabe mit dem **frühesten Fertigstellungszeitpunkt** f_i . Dann gibt es eine optimale Auswahl $S' \subseteq S$ von paarweise nicht kollidierenden Aufgaben mit $i \in S'$.

Beweis:

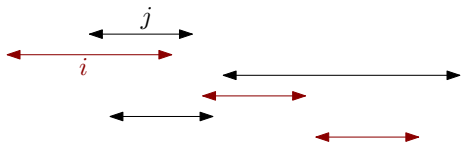
Sei $S^* \subseteq S$ **optimale Auswahl**.

Annahme: $i \notin S^*$.

$$j = \min_{k \in S^*} f_k$$

$\Rightarrow S' := (S^* \setminus \{j\}) \cup \{i\}$ kollidiert nicht

und $|S'| = |S^*|$



2.2.1 Optimale Auswahl von Aufgaben

Lemma 2.5

Es sei S eine Menge von Aufgaben und es sei $i \in S$ eine Aufgabe mit dem **frühesten Fertigstellungszeitpunkt** f_i . Dann gibt es eine optimale Auswahl $S' \subseteq S$ von paarweise nicht kollidierenden Aufgaben mit $i \in S'$.

Beweis:

Sei $S^* \subseteq S$ **optimale Auswahl**.

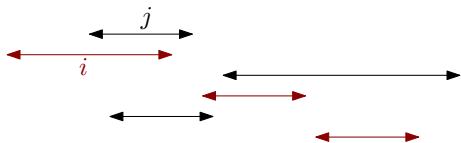
Annahme: $i \notin S^*$.

$$j = \min_{k \in S^*} f_k$$

$\Rightarrow S' := (S^* \setminus \{j\}) \cup \{i\}$ kollidiert nicht

und $|S'| = |S^*|$

$\Rightarrow S'$ ist optimale Auswahl mit $i \in S'$.



2.2.1 Optimale Auswahl von Aufgaben

Lemma 2.5

Es sei S eine Menge von Aufgaben und es sei $i \in S$ eine Aufgabe mit dem **frühesten Fertigstellungszeitpunkt** f_i . Dann gibt es eine optimale Auswahl $S' \subseteq S$ von paarweise nicht kollidierenden Aufgaben mit $i \in S'$.

Theorem 2.6

Der Algorithmus GREEDYENDE wählt für jede Instanz eine größtmögliche Menge von paarweise nicht kollidierenden Aufgaben aus.

2.2.1 Optimale Auswahl von Aufgaben

Lemma 2.5

Es sei S eine Menge von Aufgaben und es sei $i \in S$ eine Aufgabe mit dem **frühesten Fertigstellungszeitpunkt** f_i . Dann gibt es eine optimale Auswahl $S' \subseteq S$ von paarweise nicht kollidierenden Aufgaben mit $i \in S'$.

Theorem 2.6

Der Algorithmus GREEDYENDE wählt für jede Instanz eine größtmögliche Menge von paarweise nicht kollidierenden Aufgaben aus.

Beweis:

Invariante in Zeile 2 von GREEDYENDE:

S^* kann mit Auswahl von Aufgaben aus S zu optimaler Lösung erweitert werden. □

2.2.1 Optimale Auswahl von Aufgaben

```
GREEDYENDE(int[] s, int[] f)
    // Sei  $f[0] \leq f[1] \leq \dots \leq f[n-1]$ .
1    $S^* = \{0\}$ ;
2    $k = 0$ ;
3   for (int  $i = 1$ ;  $i < n$ ;  $i++$ ) {
4       if ( $s[i] \geq f[k]$ ) {
5            $S^* = S^* \cup \{i\}$ ;
6            $k = i$ ;
7       }
8   }
9   return  $S^*$ ;
```

2.2.1 Optimale Auswahl von Aufgaben

```
GREEDYENDE(int[] s, int[] f)
    // Sei  $f[0] \leq f[1] \leq \dots \leq f[n-1]$ .
1    $S^* = \{0\}$ ;
2    $k = 0$ ;
3   for (int  $i = 1$ ;  $i < n$ ;  $i++$ ) {
4       if ( $s[i] \geq f[k]$ ) {
5            $S^* = S^* \cup \{i\}$ ;
6            $k = i$ ;
7       }
8   }
9   return  $S^*$ ;
```

Theorem 2.7

Die Laufzeit des Algorithmus GREEDYENDE beträgt $O(n \log n)$. Sind die Aufgaben bereits aufsteigend nach ihrem Fertigstellungszeitpunkt sortiert, so beträgt die Laufzeit $O(n)$.

2.2.2 Rucksackproblem mit teilbaren Objekten

Rucksackproblem (Knapsack Problem (KP))

Eingabe: **Nutzen** $p_1, \dots, p_n \in \mathbb{N}$
 Gewichte $w_1, \dots, w_n \in \mathbb{N}$
 Kapazität $t \in \mathbb{N}$

2.2.2 Rucksackproblem mit teilbaren Objekten

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 $w_1 x_1 + \dots + w_n x_n \leq t$

2.2.2 Rucksackproblem mit teilbaren Objekten

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 $w_1 x_1 + \dots + w_n x_n \leq t$

Terminologie:

- $x = (x_1, \dots, x_n) \in \{0, 1\}^n$ nennen wir **Lösung**.
- Gilt $w_1 x_1 + \dots + w_n x_n \leq t$, so heißt x **gültige Lösung**.

2.2.2 Rucksackproblem mit teilbaren Objekten

Fraktionales Rucksackproblem

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 $w_1 x_1 + \dots + w_n x_n \leq t$

Terminologie:

- $x = (x_1, \dots, x_n) \in \{0, 1\}^n$ nennen wir **Lösung**.
- Gilt $w_1 x_1 + \dots + w_n x_n \leq t$, so heißt x **gültige Lösung**.

2.2.2 Rucksackproblem mit teilbaren Objekten

GREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = \frac{t}{w_i}$; $t = 0$; }
- 7 $i++$;
- 8 }
- 9 **return** (x_1, \dots, x_n);

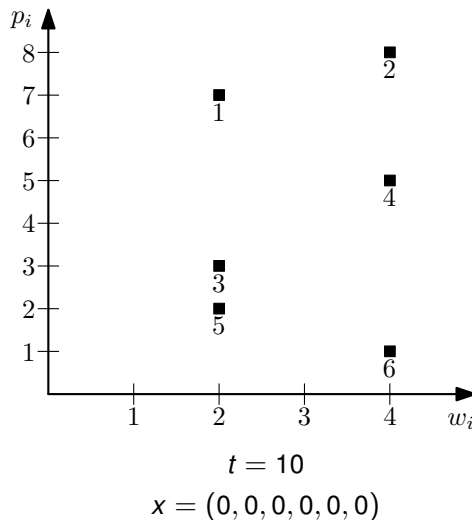
2.2.2 Rucksackproblem mit teilbaren Objekten

GREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = \frac{t}{w_i}$; $t = 0$; }
- 7 $i++$;
- 8 };
- 9 **return** (x_1, \dots, x_n);



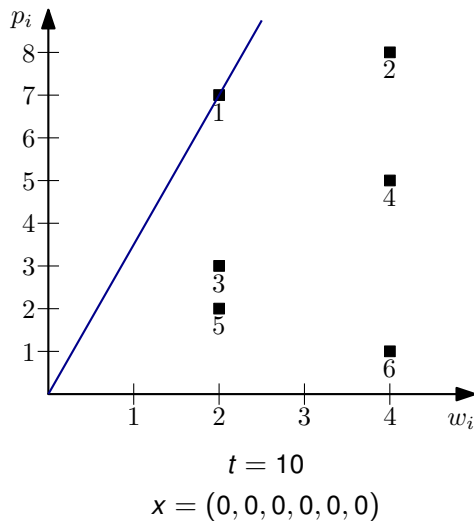
2.2.2 Rucksackproblem mit teilbaren Objekten

GREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = \frac{t}{w_i}$; $t = 0$; }
- 7 $i++$;
- 8 };
- 9 **return** (x_1, \dots, x_n);



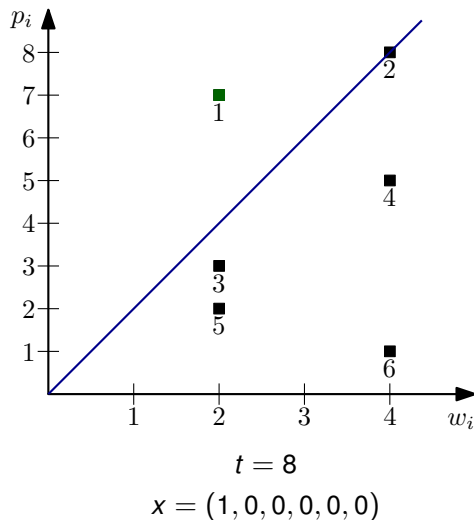
2.2.2 Rucksackproblem mit teilbaren Objekten

GREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = \frac{t}{w_i}$; $t = 0$; }
- 7 $i++$;
- 8 };
- 9 **return** (x_1, \dots, x_n);



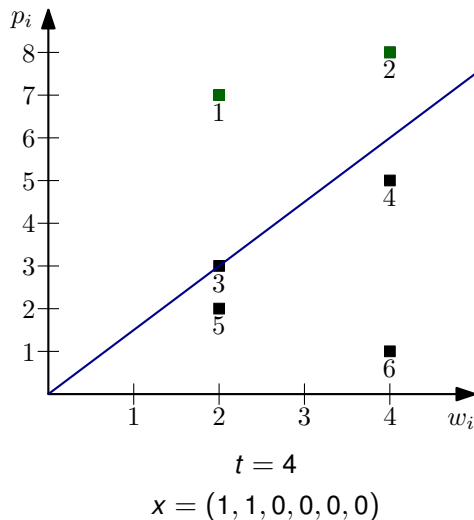
2.2.2 Rucksackproblem mit teilbaren Objekten

GREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = \frac{t}{w_i}$; $t = 0$; }
- 7 $i++$;
- 8 };
- 9 **return** (x_1, \dots, x_n);



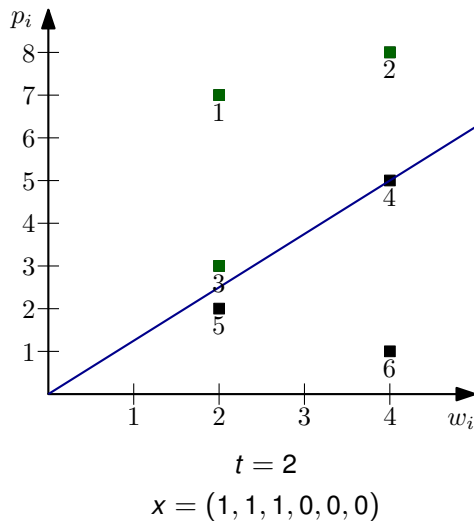
2.2.2 Rucksackproblem mit teilbaren Objekten

GREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = \frac{t}{w_i}$; $t = 0$; }
- 7 $i++$;
- 8 };
- 9 **return** (x_1, \dots, x_n);



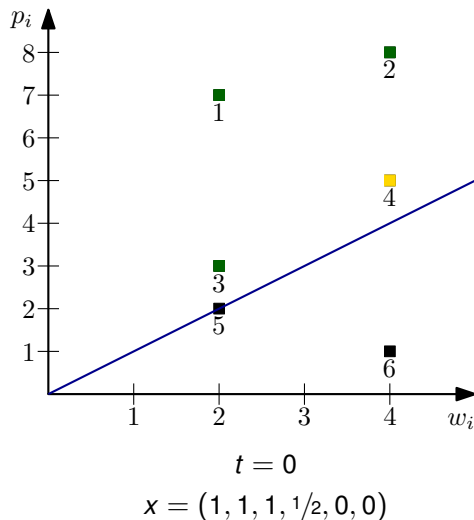
2.2.2 Rucksackproblem mit teilbaren Objekten

GREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = \frac{t}{w_i}$; $t = 0$; }
- 7 $i++$;
- 8 };
- 9 **return** (x_1, \dots, x_n);



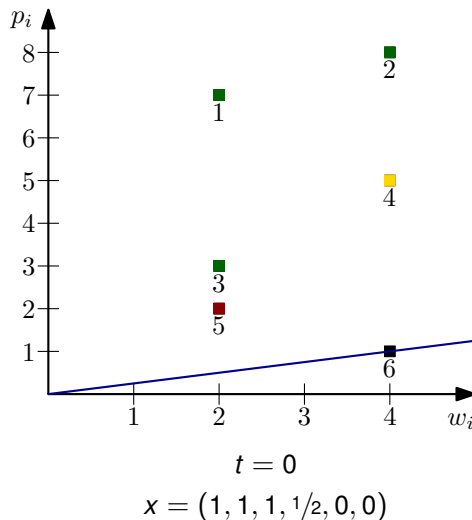
2.2.2 Rucksackproblem mit teilbaren Objekten

GREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = \frac{t}{w_i}$; $t = 0$; }
- 7 $i++$;
- 8 };
- 9 **return** (x_1, \dots, x_n);



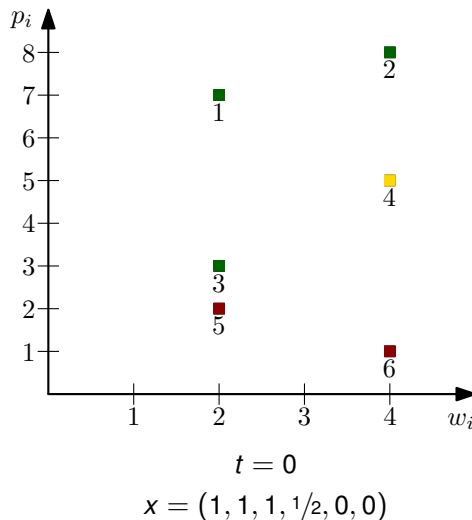
2.2.2 Rucksackproblem mit teilbaren Objekten

GREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = \frac{t}{w_i}$; $t = 0$; }
- 7 $i++$;
- 8 };
- 9 **return** (x_1, \dots, x_n);



2.2.2 Rucksackproblem mit teilbaren Objekten

Theorem 2.8

Der Algorithmus GREEDYKP berechnet für jede Instanz des Rucksackproblems mit teilbaren Objekten in Zeit $O(n \log n)$ eine optimale Lösung.

2.2.2 Rucksackproblem mit teilbaren Objekten

Theorem 2.8

Der Algorithmus GREEDYKP berechnet für jede Instanz des Rucksackproblems mit teilbaren Objekten in Zeit $O(n \log n)$ eine optimale Lösung.

Beweis: Laufzeit wird durch Sortieren dominiert.

2.2.2 Rucksackproblem mit teilbaren Objekten

Theorem 2.8

Der Algorithmus GREEDYKP berechnet für jede Instanz des Rucksackproblems mit teilbaren Objekten in Zeit $O(n \log n)$ eine optimale Lösung.

Beweis: Laufzeit wird durch Sortieren dominiert.

Korrektheit: Sei $\sum_{i=1}^n w_i > t$, sonst packt GREEDYKP optimalerweise alle Objekte ein.

2.2.2 Rucksackproblem mit teilbaren Objekten

Theorem 2.8

Der Algorithmus GREEDYKP berechnet für jede Instanz des Rucksackproblems mit teilbaren Objekten in Zeit $O(n \log n)$ eine optimale Lösung.

Beweis: Laufzeit wird durch Sortieren dominiert.

Korrektheit: Sei $\sum_{i=1}^n w_i > t$, sonst packt GREEDYKP optimalerweise alle Objekte ein.

- $x^* = (x_1^*, \dots, x_n^*) \in [0, 1]^n$ sei optimale Lösung.
- $x = (x_1, \dots, x_n) \in [0, 1]^n$ GREEDYKP-Lösung.

2.2.2 Rucksackproblem mit teilbaren Objekten

Theorem 2.8

Der Algorithmus GREEDYKP berechnet für jede Instanz des Rucksackproblems mit teilbaren Objekten in Zeit $O(n \log n)$ eine optimale Lösung.

Beweis: Laufzeit wird durch Sortieren dominiert.

Korrektheit: Sei $\sum_{i=1}^n w_i > t$, sonst packt GREEDYKP optimalerweise alle Objekte ein.

- $x^* = (x_1^*, \dots, x_n^*) \in [0, 1]^n$ sei optimale Lösung.
- $x = (x_1, \dots, x_n) \in [0, 1]^n$ GREEDYKP-Lösung.
- Es gibt $i \in \{1, \dots, n\}$ mit $x_1 = \dots = x_{i-1} = 1$, $x_i < 1$ und $x_{i+1} = \dots = x_n = 0$.

2.2.2 Rucksackproblem mit teilbaren Objekten

Theorem 2.8

Der Algorithmus GREEDYKP berechnet für jede Instanz des Rucksackproblems mit teilbaren Objekten in Zeit $O(n \log n)$ eine optimale Lösung.

Beweis: Laufzeit wird durch Sortieren dominiert.

Korrektheit: Sei $\sum_{i=1}^n w_i > t$, sonst packt GREEDYKP optimalerweise alle Objekte ein.

- $x^* = (x_1^*, \dots, x_n^*) \in [0, 1]^n$ sei optimale Lösung.
- $x = (x_1, \dots, x_n) \in [0, 1]^n$ GREEDYKP-Lösung.
- Es gibt $i \in \{1, \dots, n\}$ mit $x_1 = \dots = x_{i-1} = 1$, $x_i < 1$ und $x_{i+1} = \dots = x_n = 0$.
- Es gilt $\sum_{i=1}^n x_i w_i = t$ und $\sum_{i=1}^n x_i^* w_i = t$.

2.2.2 Rucksackproblem mit teilbaren Objekten

Idee: Transformiere x^* Schritt für Schritt in x , ohne den Nutzen zu verringern.

- $i = \min\{k \mid x_k^* < 1\}$

i	1	2	3	4	5	6
p_i	8	3	9	6	2	1
w_i	4	2	6	4	2	4

$$t = 5$$

2.2.2 Rucksackproblem mit teilbaren Objekten

Idee: Transformiere x^* Schritt für Schritt in x , ohne den Nutzen zu verringern.

- $i = \min\{k \mid x_k^* < 1\}$
- $x_{i+1}^* = \dots = x_n^* = 0 \Rightarrow x^* = x$

i	1	2	3	4	5	6
p_i	8	3	9	6	2	1
w_i	4	2	6	4	2	4

$t = 5$

2.2.2 Rucksackproblem mit teilbaren Objekten

Idee: Transformiere x^* Schritt für Schritt in x , ohne den Nutzen zu verringern.

- $i = \min\{k \mid x_k^* < 1\}$
- $x_{i+1}^* = \dots = x_n^* = 0 \Rightarrow x^* = x$
- Sonst $j = \max\{k \mid x_k^* > 0\}$. Es gilt $j > i$.

i	1	2	3	4	5	6
p_i	8	3	9	6	2	1
w_i	4	2	6	4	2	4

$t = 5$

2.2.2 Rucksackproblem mit teilbaren Objekten

Idee: Transformiere x^* Schritt für Schritt in x , ohne den Nutzen zu verringern.

- $i = \min\{k \mid x_k^* < 1\}$
- $x_{i+1}^* = \dots = x_n^* = 0 \Rightarrow x^* = x$
- Sonst $j = \max\{k \mid x_k^* > 0\}$. Es gilt $j > i$.
- Verschiebe Gewicht ε von j nach i .

Sei $\varepsilon = \min\{w_i - x_i^* w_i, x_j^* w_j\}$.

Setze $x_i^* := x_i^* + \frac{\varepsilon}{w_i}$ und $x_j^* := x_j^* - \frac{\varepsilon}{w_j}$.

i	1	2	3	4	5	6
p_i	8	3	9	6	2	1
w_i	4	2	6	4	2	4

$t = 5$

2.2.2 Rucksackproblem mit teilbaren Objekten

Idee: Transformiere x^* Schritt für Schritt in x , ohne den Nutzen zu verringern.

- $i = \min\{k \mid x_k^* < 1\}$
- $x_{i+1}^* = \dots = x_n^* = 0 \Rightarrow x^* = x$
- Sonst $j = \max\{k \mid x_k^* > 0\}$. Es gilt $j > i$.
- Verschiebe Gewicht ε von j nach i .

Sei $\varepsilon = \min\{w_i - x_i^* w_i, x_j^* w_j\}$.

Setze $x_i^* := x_i^* + \frac{\varepsilon}{w_i}$ und $x_j^* := x_j^* - \frac{\varepsilon}{w_j}$.

i	1	2	3	4	5	6
p_i	8	3	9	6	2	1
w_i	4	2	6	4	2	4

$$t = 5$$

$$x = (1, 1/2, 0, 0, 0, 0)$$

$$x^* = (1, 0, 1/12, 1/8, 0, 0)$$

2.2.2 Rucksackproblem mit teilbaren Objekten

Idee: Transformiere x^* Schritt für Schritt in x , ohne den Nutzen zu verringern.

- $i = \min\{k \mid x_k^* < 1\}$
- $x_{i+1}^* = \dots = x_n^* = 0 \Rightarrow x^* = x$
- Sonst $j = \max\{k \mid x_k^* > 0\}$. Es gilt $j > i$.
- Verschiebe Gewicht ε von j nach i .


Sei $\varepsilon = \min\{w_i - x_i^* w_i, x_j^* w_j\}$.

Setze $x_i^* := x_i^* + \frac{\varepsilon}{w_i}$ und $x_j^* := x_j^* - \frac{\varepsilon}{w_j}$.

i	1	2	3	4	5	6
p_i	8	3	9	6	2	1
w_i	4	2	6	4	2	4

$t = 5$

$$x = (1, 1/2, 0, 0, 0, 0)$$

$$x^* = (1, 0, \overset{i}{1/12}, \overset{j}{1/8}, 0, 0)$$


2.2.2 Rucksackproblem mit teilbaren Objekten

Idee: Transformiere x^* Schritt für Schritt in x , ohne den Nutzen zu verringern.

- $i = \min\{k \mid x_k^* < 1\}$
- $x_{i+1}^* = \dots = x_n^* = 0 \Rightarrow x^* = x$
- Sonst $j = \max\{k \mid x_k^* > 0\}$. Es gilt $j > i$.
- Verschiebe Gewicht ε von j nach i .


Sei $\varepsilon = \min\{w_i - x_i^* w_i, x_j^* w_j\}$.

Setze $x_i^* := x_i^* + \frac{\varepsilon}{w_i}$ und $x_j^* := x_j^* - \frac{\varepsilon}{w_j}$.

i	1	2	3	4	5	6
p_i	8	3	9	6	2	1
w_i	4	2	6	4	2	4

$t = 5$

$$x = (1, 1/2, 0, 0, 0, 0)$$

$$x^* = (1, 0, 1/12, 1/8, 0, 0)$$


$$(1, 1/4, 1/12, 0, 0, 0)$$

2.2.2 Rucksackproblem mit teilbaren Objekten

Idee: Transformiere x^* Schritt für Schritt in x , ohne den Nutzen zu verringern.

- $i = \min\{k \mid x_k^* < 1\}$
- $x_{i+1}^* = \dots = x_n^* = 0 \Rightarrow x^* = x$
- Sonst $j = \max\{k \mid x_k^* > 0\}$. Es gilt $j > i$.
- Verschiebe Gewicht ε von j nach i .


Sei $\varepsilon = \min\{w_i - x_i^* w_i, x_j^* w_j\}$.


Setze $x_i^* := x_i^* + \frac{\varepsilon}{w_i}$ und $x_j^* := x_j^* - \frac{\varepsilon}{w_j}$.

i	1	2	3	4	5	6
p_i	8	3	9	6	2	1
w_i	4	2	6	4	2	4

$t = 5$

$$x = (1, 1/2, 0, 0, 0, 0)$$

$$x^* = (1, 0, 1/12, 1/8, 0, 0)$$


$$(1, 1/4, 1/12, 0, 0, 0)$$


2.2.2 Rucksackproblem mit teilbaren Objekten

Idee: Transformiere x^* Schritt für Schritt in x , ohne den Nutzen zu verringern.

- $i = \min\{k \mid x_k^* < 1\}$
- $x_{i+1}^* = \dots = x_n^* = 0 \Rightarrow x^* = x$
- Sonst $j = \max\{k \mid x_k^* > 0\}$. Es gilt $j > i$.
- Verschiebe Gewicht ε von j nach i .


Sei $\varepsilon = \min\{w_i - x_i^* w_i, x_j^* w_j\}$.


Setze $x_i^* := x_i^* + \frac{\varepsilon}{w_i}$ und $x_j^* := x_j^* - \frac{\varepsilon}{w_j}$.

i	1	2	3	4	5	6
p_i	8	3	9	6	2	1
w_i	4	2	6	4	2	4

$t = 5$

$$x = (1, 1/2, 0, 0, 0, 0)$$

$$x^* = (1, 0, \overset{i}{1/12}, \overset{j}{1/8}, 0, 0)$$


$$(1, \overset{i}{1/4}, \overset{j}{1/12}, 0, 0, 0)$$


$$(1, 1/2, 0, 0, 0, 0)$$

2.2.2 Rucksackproblem mit teilbaren Objekten


Idee: Transformiere x^* Schritt für Schritt in x , ohne den Nutzen zu verringern.


- $i = \min\{k \mid x_k^* < 1\}$
- $x_{i+1}^* = \dots = x_n^* = 0 \Rightarrow x^* = x$
- Sonst $j = \max\{k \mid x_k^* > 0\}$. Es gilt $j > i$.
- Verschiebe Gewicht ε von j nach i .
Sei $\varepsilon = \min\{w_i - x_i^* w_i, x_j^* w_j\}$.
Setze $x_i^* := x_i^* + \frac{\varepsilon}{w_i}$ und $x_j^* := x_j^* - \frac{\varepsilon}{w_j}$.
- $\Delta \text{ Gewicht} = \frac{\varepsilon}{w_i} \cdot w_i - \frac{\varepsilon}{w_j} \cdot w_j = 0$

i	1	2	3	4	5	6
p_i	8	3	9	6	2	1
w_i	4	2	6	4	2	4

$t = 5$

$$x = (1, 1/2, 0, 0, 0, 0)$$

$$x^* = (1, 0, 1/12, 1/8, 0, 0)$$


$$(1, 1/4, 1/12, 0, 0, 0)$$


$$(1, 1/2, 0, 0, 0, 0)$$

2.2.2 Rucksackproblem mit teilbaren Objekten

Idee: Transformiere x^* Schritt für Schritt in x , ohne den Nutzen zu verringern.

- $i = \min\{k \mid x_k^* < 1\}$
- $x_{i+1}^* = \dots = x_n^* = 0 \Rightarrow x^* = x$
- Sonst $j = \max\{k \mid x_k^* > 0\}$. Es gilt $j > i$.

- Verschiebe Gewicht ε von j nach i .

Sei $\varepsilon = \min\{w_i - x_i^* w_i, x_j^* w_j\}$.


Setze $x_i^* := x_i^* + \frac{\varepsilon}{w_i}$ und $x_j^* := x_j^* - \frac{\varepsilon}{w_j}$.


- $\Delta \text{Gewicht} = \frac{\varepsilon}{w_i} \cdot w_i - \frac{\varepsilon}{w_j} \cdot w_j = 0$
 $\Delta \text{Nutzen} = \frac{\varepsilon}{w_i} \cdot p_i - \frac{\varepsilon}{w_j} \cdot p_j \geq 0$ da $\frac{p_i}{w_i} \geq \frac{p_j}{w_j}$

i	1	2	3	4	5	6
p_i	8	3	9	6	2	1
w_i	4	2	6	4	2	4

$t = 5$

$$x = (1, 1/2, 0, 0, 0, 0)$$

$$x^* = (1, 0, 1/12, 1/8, 0, 0)$$


$$(1, 1/4, 1/12, 0, 0, 0)$$


$$(1, 1/2, 0, 0, 0, 0)$$

2.2.2 Rucksackproblem mit teilbaren Objekten


Idee: Transformiere x^* Schritt für Schritt in x , ohne den Nutzen zu verringern.


- $i = \min\{k \mid x_k^* < 1\}$
- $x_{i+1}^* = \dots = x_n^* = 0 \Rightarrow x^* = x$
- Sonst $j = \max\{k \mid x_k^* > 0\}$. Es gilt $j > i$.
- Verschiebe Gewicht ε von j nach i .
Sei $\varepsilon = \min\{w_i - x_i^* w_i, x_j^* w_j\}$.
Setze $x_i^* := x_i^* + \frac{\varepsilon}{w_i}$ und $x_j^* := x_j^* - \frac{\varepsilon}{w_j}$.
- $\Delta \text{Gewicht} = \frac{\varepsilon}{w_i} \cdot w_i - \frac{\varepsilon}{w_j} \cdot w_j = 0$
 $\Delta \text{Nutzen} = \frac{\varepsilon}{w_i} \cdot p_i - \frac{\varepsilon}{w_j} \cdot p_j \geq 0$ da $\frac{p_i}{w_i} \geq \frac{p_j}{w_j}$
- Hinterher gilt $x_i^* = 1$ oder $x_j^* = 0$.
 \Rightarrow Endlich viele Verschiebungen bis $x^* = x$. \square

i	1	2	3	4	5	6
p_i	8	3	9	6	2	1
w_i	4	2	6	4	2	4

$t = 5$

$$x = (1, 1/2, 0, 0, 0, 0)$$

$$x^* = (1, 0, 1/12, 1/8, 0, 0)$$


$$(1, 1/4, 1/12, 0, 0, 0)$$


$$(1, 1/2, 0, 0, 0, 0)$$

2.2.2 Rucksackproblem mit teilbaren Objekten

GREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = \frac{t}{w_i}$; $t = 0$; }
- 7 $i++$;
- 8 }
- 9 **return** (x_1, \dots, x_n);

2.2.2 Rucksackproblem mit teilbaren Objekten

INTGREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = 0$; $t = 0$; }
- 7 $i++$;
- 8 }
- 9 **return** (x_1, \dots, x_n);

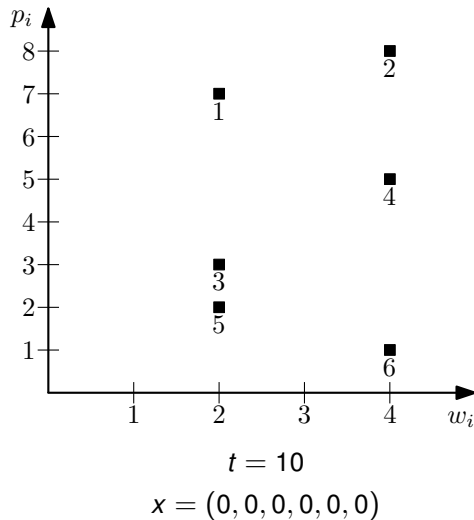
2.2.2 Rucksackproblem mit teilbaren Objekten

INTGREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = 0$; $t = 0$; }
- 7 $i++$;
- 8 };
- 9 **return** (x_1, \dots, x_n);



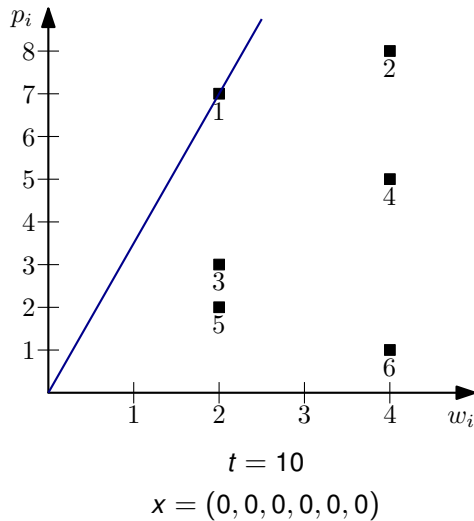
2.2.2 Rucksackproblem mit teilbaren Objekten

INTGREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = 0$; $t = 0$; }
- 7 $i++$;
- 8 };
- 9 **return** (x_1, \dots, x_n);



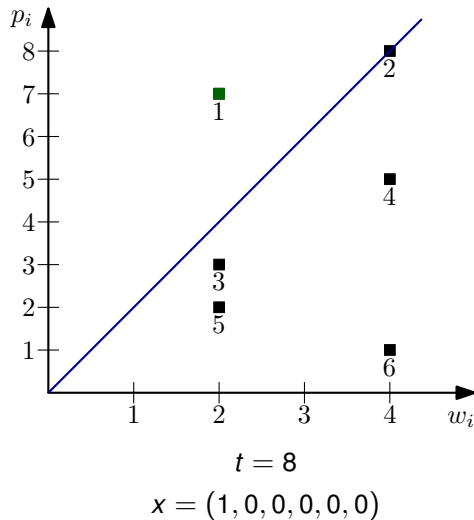
2.2.2 Rucksackproblem mit teilbaren Objekten

INTGREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = 0$; $t = 0$; }
- 7 $i++$;
- 8 };
- 9 **return** (x_1, \dots, x_n);



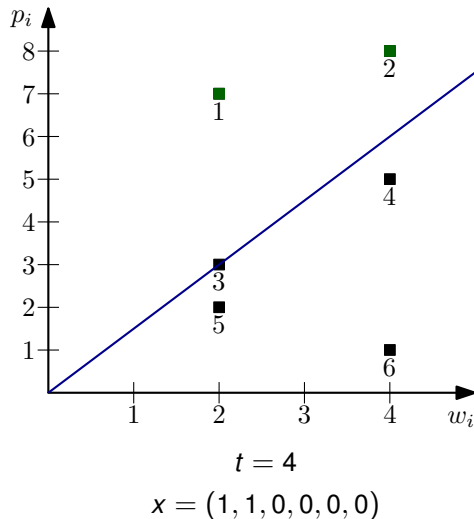
2.2.2 Rucksackproblem mit teilbaren Objekten

INTGREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = 0$; $t = 0$; }
- 7 $i++$;
- 8 };
- 9 **return** (x_1, \dots, x_n);



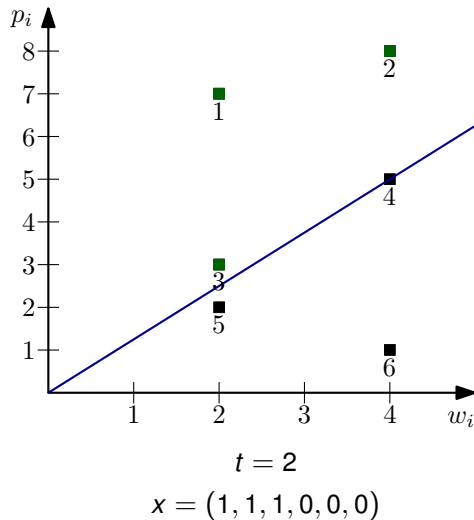
2.2.2 Rucksackproblem mit teilbaren Objekten

INTGREEDYKP

- Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- int** $i = 1$;
- while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- else** { $x_i = 0$; $t = 0$; }
- $i++$;
- }
- return** (x_1, \dots, x_n);



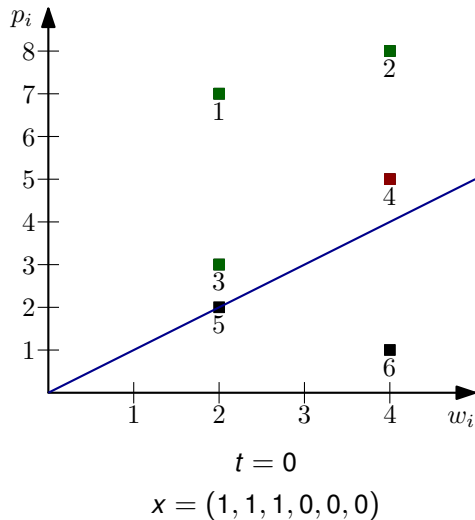
2.2.2 Rucksackproblem mit teilbaren Objekten

INTGREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = 0$; $t = 0$; }
- 7 $i++$;
- 8 }
- 9 **return** (x_1, \dots, x_n);



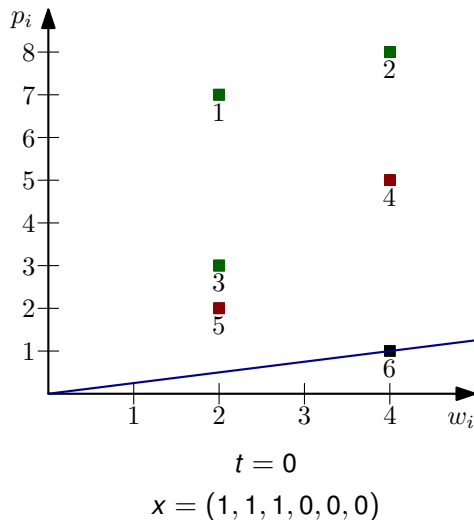
2.2.2 Rucksackproblem mit teilbaren Objekten

INTGREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = 0$; $t = 0$; }
- 7 $i++$;
- 8 };
- 9 **return** (x_1, \dots, x_n);



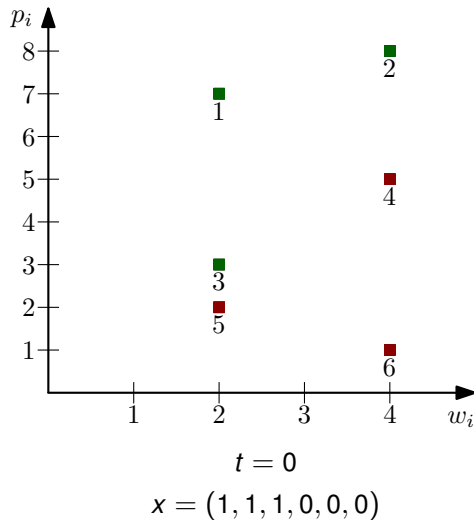
2.2.2 Rucksackproblem mit teilbaren Objekten

INTGREEDYKP

- 1 Sortiere die Objekte gemäß ihrer Effizienz. Danach gelte

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

- 2 **for** (**int** $i = 1$; $i \leq n$; $i++$) { $x_i = 0$; }
- 3 **int** $i = 1$;
- 4 **while** ($(t > 0) \ \&\& \ (i \leq n)$) {
- 5 **if** ($t \geq w_i$) { $x_i = 1$; $t = t - w_i$; }
- 6 **else** { $x_i = 0$; $t = 0$; }
- 7 $i++$;
- 8 };
- 9 **return** (x_1, \dots, x_n);



2.2.2 Rucksackproblem mit teilbaren Objekten

Beobachtung

Die Lösung, die INTGREEDYKP für das (ganzzahlige) Rucksackproblem berechnet, kann beliebig schlecht sein.

2.2.2 Rucksackproblem mit teilbaren Objekten

Beobachtung

Die Lösung, die INTGREEDYKP für das (ganzzahlige) Rucksackproblem berechnet, kann beliebig schlecht sein.

Beispiel: Sei $t = M > 2$ beliebig.

i	1	2
p_i	2	M
w_i	1	M

2.2.2 Rucksackproblem mit teilbaren Objekten

Beobachtung

Die Lösung, die INTGREEDYKP für das (ganzzahlige) Rucksackproblem berechnet, kann beliebig schlecht sein.

Beispiel: Sei $t = M > 2$ beliebig.

i	1	2
p_i	2	M
w_i	1	M

Optimale Lösung besteht nur aus Objekt 2 und hat **Nutzen M** .

2.2.2 Rucksackproblem mit teilbaren Objekten

Beobachtung

Die Lösung, die INTGREEDYKP für das (ganzzahlige) Rucksackproblem berechnet, kann beliebig schlecht sein.

Beispiel: Sei $t = M > 2$ beliebig.

i	1	2
p_i	2	M
w_i	1	M

Optimale Lösung besteht nur aus Objekt 2 und hat **Nutzen M** .

INTGREEDYKP packt nur Objekt 1 in den Rucksack und erreicht **Nutzen 2**.

2.2.2 Rucksackproblem mit teilbaren Objekten

APPROXKP

// Annahme: $w_1, \dots, w_n \leq t$

- 1 Berechne mit INTGREEDYKP eine Lösung $x^* = (x_1^*, \dots, x_n^*)$.
- 2 $j = \arg \max_{i \in \{1, \dots, n\}} p_i$; // Index eines Objektes mit maximalem Nutzen
- 3 **if** $(\sum_{i=1}^n p_i x_i^* \geq p_j)$ **return** x^* ;
- 4 **else return** $x' = (x'_1, \dots, x'_n)$ mit $x'_i = \begin{cases} 0 & \text{falls } i \neq j, \\ 1 & \text{falls } i = j. \end{cases}$

2.2.2 Rucksackproblem mit teilbaren Objekten

APPROXKP

// Annahme: $w_1, \dots, w_n \leq t$

- 1 Berechne mit INTGREEDYKP eine Lösung $x^* = (x_1^*, \dots, x_n^*)$.
- 2 $j = \arg \max_{i \in \{1, \dots, n\}} p_i$; // Index eines Objektes mit maximalem Nutzen
- 3 **if** $(\sum_{i=1}^n p_i x_i^* \geq p_j)$ **return** x^* ;
- 4 **else return** $x' = (x'_1, \dots, x'_n)$ mit $x'_i = \begin{cases} 0 & \text{falls } i \neq j, \\ 1 & \text{falls } i = j. \end{cases}$

Theorem 2.9

Der Algorithmus APPROXKP berechnet auf jeder Eingabe für das Rucksackproblem mit n Objekten in Zeit $O(n \log n)$ eine gültige ganzzahlige Lösung, deren Nutzen mindestens halb so groß ist wie der Nutzen einer optimalen ganzzahligen Lösung.

2.2.2 Rucksackproblem mit teilbaren Objekten

Beweis: Laufzeit wird durch Sortieren dominiert.

2.2.2 Rucksackproblem mit teilbaren Objekten

Beweis: Laufzeit wird durch Sortieren dominiert.

- $y \in [0, 1]^n$ GREEDYKP-Lösung
- $x^* \in \{0, 1\}^n$ INTGREEDYKP-Lösung

2.2.2 Rucksackproblem mit teilbaren Objekten

Beweis: Laufzeit wird durch Sortieren dominiert.

- $y \in [0, 1]^n$ GREEDYKP-Lösung
- $x^* \in \{0, 1\}^n$ INTGREEDYKP-Lösung

$$x^* = (\underbrace{1, \dots, 1}_{k-1}, 0, \dots, 0)$$

$$y = (\underbrace{1, \dots, 1}_{k-1}, f, \dots, 0) \quad \text{für ein } f < 1$$

2.2.2 Rucksackproblem mit teilbaren Objekten

Beweis: Laufzeit wird durch Sortieren dominiert.

- $y \in [0, 1]^n$ GREEDYKP-Lösung
- $x^* \in \{0, 1\}^n$ INTGREEDYKP-Lösung

$$x^* = (\underbrace{1, \dots, 1}_{k-1}, 0, \dots, 0)$$

$$y = (\underbrace{1, \dots, 1}_{k-1}, f, \dots, 0) \quad \text{für ein } f < 1$$

- $\text{OPT} \leq \sum_{i=1}^{k-1} p_i + fp_k \leq \sum_{i=1}^k p_i$

2.2.2 Rucksackproblem mit teilbaren Objekten

Beweis: Laufzeit wird durch Sortieren dominiert.

- $y \in [0, 1]^n$ GREEDYKP-Lösung
- $x^* \in \{0, 1\}^n$ INTGREEDYKP-Lösung

$$x^* = (\underbrace{1, \dots, 1}_{k-1}, 0, \dots, 0)$$

$$y = (\underbrace{1, \dots, 1}_{k-1}, f, \dots, 0) \quad \text{für ein } f < 1$$

- $\text{OPT} \leq \sum_{i=1}^{k-1} p_i + fp_k \leq \sum_{i=1}^k p_i$
- Nutzen der APPROXKP-Lösung:

$$\max \left\{ \sum_{i=1}^{k-1} p_i, p_j \right\}$$

2.2.2 Rucksackproblem mit teilbaren Objekten

Beweis: Laufzeit wird durch Sortieren dominiert.

- $y \in [0, 1]^n$ GREEDYKP-Lösung
- $x^* \in \{0, 1\}^n$ INTGREEDYKP-Lösung

$$x^* = (\underbrace{1, \dots, 1}_{k-1}, 0, \dots, 0)$$

$$y = (\underbrace{1, \dots, 1}_{k-1}, f, \dots, 0) \quad \text{für ein } f < 1$$

- $\text{OPT} \leq \sum_{i=1}^{k-1} p_i + fp_k \leq \sum_{i=1}^k p_i$
- Nutzen der APPROXKP-Lösung:

$$\max \left\{ \sum_{i=1}^{k-1} p_i, p_j \right\} \geq \frac{1}{2} \left(\sum_{i=1}^{k-1} p_i + p_j \right)$$

2.2.2 Rucksackproblem mit teilbaren Objekten

Beweis: Laufzeit wird durch Sortieren dominiert.

- $y \in [0, 1]^n$ GREEDYKP-Lösung
- $x^* \in \{0, 1\}^n$ INTGREEDYKP-Lösung

$$x^* = (\underbrace{1, \dots, 1}_{k-1}, 0, \dots, 0)$$

$$y = (\underbrace{1, \dots, 1}_{k-1}, f, \dots, 0) \quad \text{für ein } f < 1$$

- $\text{OPT} \leq \sum_{i=1}^{k-1} p_i + fp_k \leq \sum_{i=1}^k p_i$
- Nutzen der APPROXKP-Lösung:

$$\max \left\{ \sum_{i=1}^{k-1} p_i, p_j \right\} \geq \frac{1}{2} \left(\sum_{i=1}^{k-1} p_i + p_j \right) \geq \frac{1}{2} \left(\sum_{i=1}^{k-1} p_i + p_k \right)$$

2.2.2 Rucksackproblem mit teilbaren Objekten

Beweis: Laufzeit wird durch Sortieren dominiert.

- $y \in [0, 1]^n$ GREEDYKP-Lösung
- $x^* \in \{0, 1\}^n$ INTGREEDYKP-Lösung

$$x^* = (\underbrace{1, \dots, 1}_{k-1}, 0, \dots, 0)$$

$$y = (\underbrace{1, \dots, 1}_{k-1}, f, \dots, 0) \quad \text{für ein } f < 1$$

- $\text{OPT} \leq \sum_{i=1}^{k-1} p_i + fp_k \leq \sum_{i=1}^k p_i$
- Nutzen der APPROXKP-Lösung:

$$\max \left\{ \sum_{i=1}^{k-1} p_i, p_j \right\} \geq \frac{1}{2} \left(\sum_{i=1}^{k-1} p_i + p_j \right) \geq \frac{1}{2} \left(\sum_{i=1}^{k-1} p_i + p_k \right) = \frac{1}{2} \left(\sum_{i=1}^k p_i \right)$$

2.2.2 Rucksackproblem mit teilbaren Objekten

Beweis: Laufzeit wird durch Sortieren dominiert.

- $y \in [0, 1]^n$ GREEDYKP-Lösung
- $x^* \in \{0, 1\}^n$ INTGREEDYKP-Lösung

$$x^* = (\underbrace{1, \dots, 1}_{k-1}, 0, \dots, 0)$$

$$y = (\underbrace{1, \dots, 1}_{k-1}, f, \dots, 0) \quad \text{für ein } f < 1$$

- $\text{OPT} \leq \sum_{i=1}^{k-1} p_i + fp_k \leq \sum_{i=1}^k p_i$
- Nutzen der APPROXKP-Lösung:

$$\max \left\{ \sum_{i=1}^{k-1} p_i, p_j \right\} \geq \frac{1}{2} \left(\sum_{i=1}^{k-1} p_i + p_j \right) \geq \frac{1}{2} \left(\sum_{i=1}^{k-1} p_i + p_k \right) = \frac{1}{2} \left(\sum_{i=1}^k p_i \right) \geq \frac{\text{OPT}}{2} \quad \square$$