

Aufgabe 7:

a)

Algorithmus:

17	18	5	44
7/9	5,5	12,5	9/8

1. Sortiere Objekte nach w_i (Gewichte) aufsteigend ?
2. Initialisiere Matrix mit $n+1$ Zeilen (n = Anzahl der Objekte) und $W+1$ Spalten (W = Gewichtsbeschränkung)
3. Beläge erste Zeile und Spalte mit 0.
4. Beliegung der Matrix: zeilenweise Iteration

4.1 Wenn $w_i \leq$ Spaltenindex:

Suche nach maximalem Nutzen (x_i eingebracht oder nicht) c_i und beläge die ~~ge~~ aktuelle Zelle mit ~~dem~~ diesem Maximum:

4.1.1 Nutzen, wenn x_i nicht eingebracht wird:
Wert der Zelle darüber4.1.2 Nutzen, wenn x_i eingebracht wird:
 c_i (~~aktive~~ Nutzen von x_i) + Wert der Zelle ($i-1$, (Spaltenindex - w_i))4.2 Wenn $w_i >$ Spaltenindex:
Beläge Zelle mit dem Wert der Zelle darüber

Pseudo code:

```
1  dyn (X, w, c, W) :  
2  // Sortiere Objekte nach w; aufsteig. w = w1, ..., wn ∈ N entsprechende Gewichte  
3  m ← m [n+1] [W+1] // initialisiere Matrix:  
4  for i=0 to n do: // Beliebige erste Zeile  
5    m [0] [i] ← 0  
6  end // Beliebige erste Spalte  
7  for j=0 to W do: // for j=0 to n do:  
8    m [j] [0] ← 0  
9  end  
10 for i=1 to n do:  
11   for j=0 to W do:  
12     if wi ≤ j :  
13       m [i] [j] ← max (m [i-1] [j], ci + m [i-1] [j-wi])  
14     end if  
15   else:  
16     m [i] [j] ← m [i-1] [j]  
17   end else  
18 end  
19 end  
20 return m
```

Diese Fälle kommt man ein Sackenimplizieren

2/2

Beispiel:

Objekte K:	1	2	3	4	5
Gewicht w:	1	3	4	5	2
Nutzen c:	1	4	5	7	4
					$W=9$

1. Sortierung:

K	1	2	3	4	5
w	1	2	3	4	5
c	1	4	4	5	7

2. Befüllung der Matrix nach Algorithmus:

w	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1
2	0	1	4	5	5	5	5	5	5	5
3	0	1	4	5	5	8	9	9	9	9
4	0	1	4	5	5	8	9	10	10	13
5	0	1	4	5	5	8	9	11	12	13

13 ← max value

b)

Nach erstellen der Belegen der in a) beschriebenen Matrix folgender Algorithmus zur Bestimmung der eingepackten Elemente

5. ~~Hertere~~ noch Erstelle leere Liste

6. Hertere rückwärts durch Matrix bei dem maximalen Wert beginnend

6.1 Wenn der Wert der Zelle über der aktuellen Zelle dem Wert der aktuellen Zelle entspricht:
Das Objekt wurde nicht eingepackt

Nächste zu betrachtende Zelle: die Zelle über der aktuellen Zelle

6.2 Sind die in 6.1 beschriebenen Werte verschieden:
Das Objekt wurde eingepackt und wird ~~der~~ zur Liste hinzugefügt.

Nächste zu betrachtende Zelle:
Zeile darüber, Spalte (aktueller Spaltenindex - aktuelles Gewicht)

Pseudocode:

```
1  } s. a)  
19  
20 List items ← new List()  
21 i ← n  
22 j ← w  
23 while i > 0 do:  
24     if m[i][j] = m[i-1][j]:  
25         i ← i-1  
26     end if  
27     else:  
28         add x; to items  
29         j ← j - w;  
30         i ← i - 1  
31     end else  
32 end  
33 return items
```

c)

Algorithmus.

1. Initialisiere Matrix mit $W+1$ Spalten und beliege jede Zelle mit 0
2. Sortiere Objekte nach w_i aufsteigend

3. Belegung des Arrays: (Iteriere durch x_i und Array)

- 3.1 Wenn $w_i >$ Spaltenindex: Zelle bleibt unverändert

- 3.2 Wenn $w_i =$ Spaltenindex:

- 3.2.1 Wenn $c_i >$ Wert der Zelle:

update Wert der Zelle nach c_i

- 3.2.2 Wenn $c_i \leq$ Wert der Zelle:

Zelle bleibt unverändert

- 3.3 Wenn $w_i <$ Spaltenindex:

Finde x_j 's mit $\sum w_j \leq W-w_i$ & $\max \sum c_j$; ~~mit $w_j < w_i$~~

~~Wenn~~

- 3.3.1 Wenn $c_i + \sum c_j >$ aktueller Zellenwert:

Update Zelle mit $c_i + \sum c_j$

- 3.3.2 Wenn $c_i + \sum c_j \leq$ aktueller Zellenwert:

Zelle bleibt unverändert

Psudocode:

```

1  dynred (X, w, c, W):
  // initialisiere Array
2  m  $\leftarrow$  m [W+1]
3  Sortiere  $x_i$  nach  $w_i$  aufsteigend
4  for i=1 to n
5    for i=0 to W do:
6      m[i]  $\leftarrow$  0
7    end
8    for i=1 to n do:
9      for j=1 to W do:
10        if  $w_i = j$ :
11          if  $c_i > m[j]$ :
12            m[j]  $\leftarrow$   $c_i$ 
13          end if
14        if  $w_i < j$ :
15          find max  $\sum_{k \leq i} c_k$  über  $c_k$  (Teilsummen aller  $x_k$  mit  $w_k \leq w_i$  mit  $w_k < w_i$ )
16          if  $c_i + \sum_{k \leq i} c_k > m[j]$ 
17            m[j]  $\leftarrow$   $c_i + \sum_{k \leq i} c_k$ 
18          end if
19        end if
20      end
21    end
22  return m

```

$X := x_1, \dots, x_n \in \mathbb{N}$ Objekte
 $w = w_1, \dots, w_n \in \mathbb{N}$ entsprechende Gewichte
 $c = c_1, \dots, c_n \in \mathbb{N}$ entsprechende Nutzen
 $W =$ Gewichtslimit

Man soll Zeilen j und $j-1$ abspeichern, damit man überprüfen kann, ob es zum wechsel des Objekts $j-1$ wieder anpassen. Ausserdem sollte diese Information verstecken

Beispiel (aus a) (berücksichtigt):

x_i	1	2	3	4	5
w:	1	2	3	4	5
a_i	1	9	4	5	7

(Sehr spezielles Beispiel)

Iterationsen

$i=1:$

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

$i=2:$

0	1	2	3	4	5	6	7	8	9
0	1	4	5	5	5	5	5	5	5

$i=3:$

0	1	2	3	4	5	6	7	8	9
0	1	4	5	5	8	9	9	9	9

$i=4:$

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

$i=5:$

0	1	2	3	4	5	6	7	8	9
0	1	4	5	5	8	9	11	12	13

max value

d)

Algorithmus:

1. Initialisiere Array mit $w+1$ Spalten und belege jede Zelle mit 0
 2. Sortiere Objekte nach w ; aufsteigend
 3. Initialisiere Liste items und kopiere alle x_i hinein
 4. Belegung des Arrays (Iteriere durch x_i und Array):
 - 4.1 Wenn $w_i >$ Spaltenindex:
Zelle bleibt unverändert
 x_i wird aus items gelöscht
 - 4.2 Wenn $w_i =$ Spaltenindex:
 - 4.2.1 Wenn $c_i >$ Wert der Zelle:
update Wert der Zelle nach c_i
Ist x_i noch nicht in items enthalten, wird es hinzugefügt
 - 4.2.2 Wenn $c_i \leq$ Wert der Zelle:
Zelle bleibt unverändert
 x_i wird aus items gelöscht (wenn nicht schon zuvor gelöscht)
 - 4.3 Wenn $w_i <$ Spaltenindex:
Finde max (s. c))
- ~~Wenn max~~ \geq
- 4.3.1 Wenn max \neq aktueller Zellenwert:
update Zellenwert nach max
Ist x_i noch nicht in items enthalten, wird es hinzugefügt
 - 4.3.2 Wenn max \leq aktueller Zellenwert:
Zelle bleibt unverändert
Ist x_i in items enthalten, wird es gelöscht

Pseudo code

```

1   } s. c
2
3
4
5
6
7 List items ← new list ∅()
8 copy X to items
9 for i=1 to n do:
10   for j = 0 to W do:
11     if  $w_i > j$ :
12       delete  $x_i$  from items  $\star^1$ 
13     end if
14     if  $w_i = j$ :
15       if  $c_i \geq m[j]$ :
16          $m[j] \leftarrow c_i$ 
17         add  $x_i$  to items  $\star^2$ 
18       end if
19       if  $c_i < m[j]$ :
20         delete  $x_i$  from items  $\star^1$ 
21       end if
22     end if
23     if  $w_i < j$ :
24       find max ( Teilsummen über  $c_k$  aller  $x_k$  mit  $\sum w_k < W - w_i$   

 $\wedge w_k \leq w_i$ )  $\star^1$ 
25     if  $c_i + \sum c_k > m[j]$ :
26        $m[j] \leftarrow c_i + \sum c_k$  Der plausibl. Fehler  
als bei 7c)
27       add  $x_i$  to items  $\star^2$ 
28     end if
29   else:
30     delete  $x_i$  from items  $\star^1$ 
31
32
33
34   end end end
  
```

\star^1 (if x_i is in items)

\star^2 (if x_i is not in items)

Aufgabe 8.

Du nutzt nicht die Berechnung von einer

Lösung mit einer begrenzten Anzahl von Iterationen.

Du errechnest direkt die letzte Teilzellzeile.

a)

Variablen $c(k)$ für $k = 0, \dots, W$

$c(k)$ gibt den maximalen Nutzen aller Objekte an,

~~Teilmenge $\{1, \dots, k\}$~~ unter der Bedingung, dass das Gesamtgewicht der Objekte ~~gleich~~ den Wert k nicht überschreitet.

Die Werte werden durch die Rekurrenz

$$c(k) := \max \{c_i + c(k - w_i) \}$$

berechnet.

* für $k = 0, \dots, W$

(-1,5)

b)

k	0	1	2	3	4	5	6	7	8	9	10
$c(k)$	0	7	2	4	5	7	9	10	11	12	14
i_{\max}	0	4	4	2	2	7	3	3	$1 \sqrt{3}$	$2 \sqrt{3}$	$1 \sqrt{3}$

i_{\max} gibt den Gegenstand ~~i~~, ~~der~~ ~~an~~ ~~dem~~ ~~Wert~~ ~~an~~ ~~dem~~

~~maximalen Nutzen~~, an dem der Wert $c_i + c(k - w_i)$ maximal

$$c(10) = c_{10} + c(10 - w_1) = c_1 + c(9) = c_1 + c_1 + c(9 - w_1)$$

$$= 4 + 2c_1 + 0 = 2 \cdot 7 = 14$$

$$\Rightarrow x_1 = 2, x_2 = 0, x_3 = 0, x_4 = 0$$

$$x^* = (2, 0, 0, 0) \checkmark$$

$$c(10) = c_3 + c(10 - w_3) = c_3 + c(7) = c_3 + c_2 + c(7 - w_2)$$

$$= c_3 + c_2 + c(7) = c_3 + c_2 + c_4 + c(0)$$

$$= 9 + 4 + 7 = 20$$

$$\Rightarrow x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 1$$

$$x = (0, 1, 1, 1) \checkmark$$

(5,5)

(10)