Aufgabe 1. Wir betrachten eine Funktion $f\colon \mathbb{N} \to \mathbb{Z}$, die wie folgt definiert ist:

$$f(n) := \begin{cases} 1 & \text{wenn } n \in \{1,2\} \\ |n \cdot f(n-2)| - (f(n-1))^2 & \text{wenn } n > 2 \text{ und } n \text{ gerade ist und} \\ |n \cdot f(n-1)| - (f(n-2))^2 & \text{wenn } n > 2 \text{ und } n \text{ ungerade ist.} \end{cases}$$

(a) Machen Sie sich mit der Funktion vertraut, indem Sie die Werte für $f(1), f(2), \ldots, f(8)$ angeben.

(b) Geben Sie einen naiven rekursiven Algorithmus (ohne dynamische Programmierung) an, der $f(n)$ berechnet.

(c) Adaptieren Sie Ihren naiven Algorithmus, sodass dieser dynamische Programmierung verwendet. Der Algorithmus soll immer noch rekursiv vorgehen. Geben Sie die Laufzeit des neuen Algorithmus in O-Notation an. Was ist der Vorteil gegenüber Ihrem Algorithmus aus Unteraufgabe (b)?

(d) Modifizieren Sie Ihren Algorithmus weiter, sodass dieser $f(n)$ iterativ und mittels dynamischer Programmierung arbeitet. Geben Sie die Laufzeit des neuen Algorithmus in O-Notation an.

(a)

$$\begin{aligned} f(1) &= 1 \\ f(2) &= 2 \\ f(3) &= |3 \cdot 1| - 1^2 = 2 \\ f(4) &= |4 \cdot 1| - 2^2 = 0 \\ f(5) &= |5 \cdot 0| - 2^2 = -4 \\ f(6) &= |6 \cdot 0| - (-4)^2 = -16 \\ f(7) &= |7 \cdot (-16)| - (-4)^2 = 96 \\ f(8) &= |8 \cdot (-16)| - 96^2 = -9088 \end{aligned}$$

(b)
```
def f(n):
    if n > 2:
        if n % 2 == 0:
            return abs(n*f(n-2)) - f(n-1)**2
        else:
            return abs(n*f(n-1)) - f(n-2)**2
    elif n > 0:
        return 1
    else:
        raise ValueError()
```

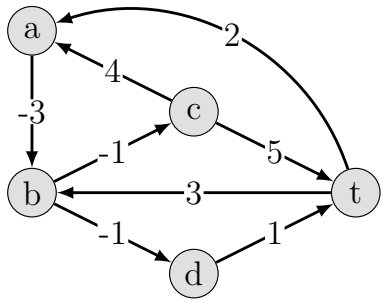
(c)
```
f_values = None

def f(n):
    global f_values
    if n < 1:
        raise ValueError()
    f_values = [None] * n
    return f_main(n)

def f_main(n):
    global f_values
    if f_values[n-1] != None:
        return f_values[n-1]
    res = 1
    if n > 2:
        if n % 2 == 0:
            res = abs(n*f_main(n-2)) - f_main(n-1)**2
        else:
            res = abs(n*f_main(n-1)) - f_main(n-2)**2
    f_values[n-1] = res
    return res
```

Laufzeit: $O(n)$
Vorteil: Die Werte müssen nicht immer neu berechnet werden, sobald ein Wert für n einmal berechnet wurde kann er einfach wiederverwendet werden.

(d)
```
def f(n):
    if n < 1:
        raise ValueError()
    f_values = [None] * n
    for i in range(1, n+1):
        f_values[i-1] = 1
        if i > 2:
            if i % 2 == 0:
                f_values[i-1] = abs(i*f_values[(i-1)-2]) -
                   f_values[(i-1)-1]**2
            else:
                f_values[i-1] = abs(i*f_values[(i-1)-1]) -
                   f_values[(i-1)-2]**2
    return f_values[n-1]
```

Laufzeit: $O(n)$

Aufgabe 2. Gegeben ist folgender gerichteter Graph mit Kantengewichten.



(a) Wenden Sie den Algorithmus namens Simple-Shortest-Path-DP zur Berechnung von kürzesten Wegen aus der Vorlesung auf den Graphen oben an. Hierbei soll der Startknoten a sein und der Zielknoten t. Es genügt, den Tabelleninhalt nach jeder Iteration der Hauptschleife anzugeben.

Hinweis: Bevor Sie diese Teilaufgabe lösen, lesen Sie zuerst die zweite Teilaufgabe.

(b) Erklären Sie, wie Sie die Tabelle und den Algorithmus modifizieren können, sodass Sie einen negativen Kreis im Eingabegraphen finden können, sofern einer existiert. Nutzen Sie Ihre Einsicht, um mithilfe Ihrer Tabelle einen negativen Kreis im obigen Graphen zu finden.

Hinweis: Speichern Sie neben Zahlenwerten auch entsprechende Nachfolger in der Tabelle.

(a)

	0	**1**	**2**	**3**	**4**	**5**
t	0	0	0	0	-1	-1
a	∞	∞	∞	-3	-3	-3
b	∞	∞	0	0	0	-1
c	∞	5	5	5	1	1
d	∞	1	1	1	0	0

(b) Man könnte den Algorithmus nach Termination noch einmal (also n+1 mal) ausführen und überprüfen, ob sich ein Eintrag geändert hat. Die Tabelle müsste natürlich auch um eine Spalte erweitert werden.
In dem Graphen existiert ein negativer Kreis, da $OPT(n, b) = 0 \neq OPT(n + 1, b) = -1$

Aufgabe 3. Gegeben ist folgende Instanz des Rucksackproblems:

#	Wert	Gewicht
1	2	1
2	5	2
3	8	3
4	10	4
5	12	5
6	14	6

Kapazität $G = 11$

(a) Wenden Sie den aus der Vorlesung bekannten Greedy-Algorithmus für das Rucksackproblem auf die obige Instanz an. Welche Gegenstände werden ausgewählt? Welchen Gesamtwert haben diese?

(b) Lösen Sie die obige Instanz jetzt durch dynamische Programmierung. Geben Sie dazu, wie aus der Vorlesung bekannt, die vollständige Belegung der 2-dimensionalen Lösungstabelle an.

Geben Sie die Menge der für die optimale Lösung ausgewählten Gegenstände an und markieren Sie in der Tabelle durch Einkreisen all jene Felder, die der Algorithmus `Find-Solution` aus der Vorlesung bei der Berechnung der Lösungsmenge A ausliest und verwendet. Welchen Gesamtwert hat die optimale Lösung?

(a)

#	**Wert**	**Gewicht**	$\frac{w_i}{g_i}$
3	8	3	$2.\dot{6}$
2	5	2	2.5
4	10	4	2.5
6	14	6	$2.\dot{3}$
5	12	5	2.2
1	2	1	2

Greedy: $w = 8 + 5 + 10 + 2 = 25 \quad g = 3 + 2 + 4 + 1 = 10$

(b)

	0	1	2	3	4	5	6	7	8	9	10	11
$\emptyset$	0	(0)	0	0	0	0	0	0	0	0	0	0
$\{1\}$	0	(2)	2	(2)	2	2	2	2	2	2	2	2
$\{1,2\}$	0	2	5	(7)	7	7	(7)	7	7	7	7	7
$\{1,2,3\}$	0	2	5	8	10	13	(15)	15	15	15	15	15
$\{1,2,3,4\}$	0	2	5	8	10	13	(15)	18	20	23	25	(25)
$\{1,2,3,4,5\}$	0	2	5	8	10	13	15	18	20	23	25	(27)
$\{1,2,3,4,5,6\}$	0	2	5	8	10	13	15	18	20	23	25	(27)

Gewählte Gegenstände: $\{5, 3, 2, 1\}$
Gesamtwert: 27