be-OI 2019 **Finale - SENIOR** Samstag 30. März 2019	**Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp** PRÉNOM : ..... NOM : ..... ÉCOLE :	O **Réservé**

Belgische Informatik-Olympiade (Dauer : maximal 2 Stunden)

Dieses Dokument ist der Fragebogen für das Finale der belgischen Informatik-Olympiade 2019. Es enthält 5 Fragen, die innerhalb von **maximal 2 Stunden** gelöst werden müssen.

Allgemeine Hinweise (bitte sorgfältig lesen bevor du die Fragen beantwortest)

1. Überprüfe, ob du die richtigen Fragen erhalten hast. (s. Kopfzeile oben):
   - Für Schüler bis zum zweiten Jahr der Sekundarschule: Kategorie **Kadetten**.
   - Für Schüler im dritten oder vierten Jahr der Sekundarschule: Kategorie **Junior**.
   - Für Schüler der Sekundarstufe 5 und höher: Kategorie **Senior**.
2. Gebe deinen Namen, Vornamen und deine Schule **nur auf der erste Seite** an.
3. Gebe **deine Antworten** auf den in diesem Dokument **am Ende des Formulars** dafür vorgesehenen Seiten an.
4. Wenn du dich bei einer Antwort vertan hast, dann beantworte sie unbedingt **auf dem selbem Blatt** (ggf. auf der Rückseite).
5. Schreibe **gut lesbar** mit einem **blauen oder schwarzen Stift oder Kugelschreiber**.
6. Du kannst nur das Schreibmaterial dabei haben; Taschenrechner, Mobiltelefone, ... sind **verboten**.
7. Du kannst jederzeit weitere Kladdeblätter an einer Aufsichtsperson anfragen.
8. Wenn du fertig bist, gibst du die erste Seite (mit deinem Namen) und die Seiten (mit den Antworten) ab. Du kannst die anderen Seiten behalten.
9. Alle Codeauszüge aus der Anweisung sind in **Pseudo-Code**. Auf den folgenden Seiten findest du eine **Beschreibung** des Pseudo-Code, den wir verwenden.
10. Wenn du mit einem Code antworten musst, dann benutze den **Pseudo-Code** oder eine **Sprache aktuelle aktuelle Programmiersprache** (Java, C, C++, Pascal, Python,....). Syntaxfehler werden bei der Auswertung nicht berücksichtigt.

Viel Erfolg!

Die belgische Olympiade der Informatik ist möglich dank der Unterstützung durch unsere Sponsoren und Mitglieder:

docbyte Value your content

Pseudocode Checkliste

Die Daten werden in Variablen gespeichert. Wir ändern den Wert einer Variablen mit $\leftarrow$. In einer Variablen können wir ganze Zahlen, reelle Zahlen oder Tabellen speichern (siehe weiter unten), sowie boolesche (logische) Werte: wahr/falsch (**`true`**) oder falsch/fehlerhaft (**`false`**). Es ist möglich, arithmetische Operationen auf Variablen durchzuführen. Zusätzlich zu den vier traditionelle Operatoren ($+$, $-$, $\times$ und $/$), kann man auch den Operator % verwenden. Wenn a und b ganze Zahlen sind, dann stellt a/b und $a\%b$ den Quotienten bzw. den Rest der ganzen Division dar. Zum Beispiel, wenn $a = 14$ und $b = 3$, dann $a/b = 4$ und $a\%b = 2$.

Hier ist ein erstes Beispiel für einen Code, in dem die Variable age den Wert 17 erhält.

```
geburtsjahr ← 2002
age ← 2019 − geburtjahr
```

Um Code nur auszuführen, wenn eine bestimmte Bedingung erfüllt ist, verwendet man den Befehl **`if`** und möglicherweise den Befehl **`else`**, um einen anderen Code auszuführen, wenn die Bedingung falsch ist. Das nächste Beispiel prüft, ob eine Person volljährig ist und speichert den Eintrittspreis für diese Person in der Variable $preis$. Beachte die Kommentare im Code.

```
if (alter ≥ 18)
{
    preis ← 8 // Das ist ein Kommentar.
}
else
{
    preis ← 6 // billiger!
}
```

Manchmal, wenn eine Bedingung falsch ist, muss eine andere überprüft werden. Dazu können wir **`else if`** verwenden, was so ist, als würde man ein anderes **`if`** innerhalb des **`else`** des ersten **`if`** ausführen. Im folgenden Beispiel gibt es 3 Alterskategorien, die 3 unterschiedlichen Preisen für das Kinoticket entsprechen.

```
if (alter ≥ 18)
{
    preis ← 8 // Preis fuer eine erwachsene Person.
}
else if (alter ≥ 6)
{
    preis ← 6 // Preis fuer ein Kind ab 6 Jahren.
}
sonst
{
    preis ← 0 // Kostenlos fuer ein Kind unter 6 Jahren.
}
```

Um mehrere Elemente mit einer einzigen Variablen zu manipulieren, verwenden wir eine Tabelle. Die einzelnen Elemente einer Tabelle werden durch einen Index gekennzeichnet (in eckigen Klammern nach dem Namen der Tabelle). Das erste Element einer Tabelle tab ist der Index 0 und wird mit $tab[0]$ bezeichnet. Der zweite ist der Index 1 und der letzte ist der Index $N - 1$, wenn die Tabelle N Elemente enthält. Wenn beispielsweise die Tabelle tab die 3 Zahlen 5, 9 und 12 (in dieser Reihenfolge) enthält, dann $tab[0] = 5$, $tab[1] = 9$, $tab[2] = 12$. Die Tabelle hat die Länge 3, aber der höchste Index ist 2.

Um Code zu wiederholen, z.B. um durch die Elemente einer Tabelle zu durchlaufen, kann man ein Schleifencodefor verwenden. Die Schreibweise **for** ($i \leftarrow a$ **to** b **step** k) stellt eine Schleife, die so lange wiederholt wird, wie $i \leq b$, in der i mit dem Wert a beginnt und wird am Ende jedes Schrittes um den Wert k erhöht. Das folgende Beispiel berechnet die Summe der Elemente in der Tabelle. tab Angenommen, die Tabelle ist N lang. Die Summe befindet sich am Ende der Ausführung des Algorithmus in der Variable sum.

```
summe ← 0
for (i ← 0 to N − 1 step 1)
{
    summe ← summe + tab[i]
}
```

Sie können eine Schleife auch mit dem Befehl **while** schreiben, der den Code wiederholt, solange seine Bedingung wahr ist. Im folgenden Beispiel wird eine positive ganze Zahl N durch 2 geteilt, dann durch 3, dann um 4 . . . , bis sie nur noch aus einer Ziffer besteht (d.h. bis $N < 10$).

```
d ← 2
while (N ≥ 10)
{
    N ← N/d
    d ← d + 1
}
```

Häufig befinden sich die Algorithmen in einer Struktur und werden durch Erklärungen ergänzt. Nach `Eingabe` wird jedes Argument (Variabel) definiert, die als Eingaben für den Algorithmus angegeben werden. Nach `Ausgabe` wird der Zustand bestimmter Variablen am Ende der Algorithmusausführung und möglicherweise der zurückgegebenen Wert definiert. Ein Wert kann mit dem Befehl **return** zurückgegeben werden. Wenn dieser Befehl ausgeführt wird, stoppt der Algorithmus und der angegebene Wert wird zurückgegeben.

Hier ist ein Beispiel für die Berechnung der Summe der Elemente einer Tabelle.

```
Eingabe: tab, ein Tabelle mit N Zahlen.
         N, die Anzahl der Elemente der Tabelle.
Ausgabe: sum, die Summe aller in der Tabelle enthaltenen Zahlen.

summe ← 0
for (i ← 0 to N − 1 step 1)
{
    summe ← summe + tab[i]
}
return summe
```

Hinweis: In diesem letzten Beispiel wird die Variable i nur als Zähler für die Schleife **for** verwendet. Es gibt also keine Erklärung darüber in `Eingabe` oder `Ausgabe`, und ihr Wert wird nicht zurückgegeben.

Frage 1 – Pixel-Einführung

Ein Bildschirm besteht aus Pixel. Ein FullHD-Bildschirm besteht z.B. aus 1080 Linien mit jeweils 1920 Pixel. Manche Pixel können defekt sein. Wenn es zu viele werden oder diese schlecht platziert sind, muss der Bildschirm ersetzt werden.

Ein Testgerät gibt eine Tabelle $pix[\,][\,]$ mit $r \times c$ Zahlen (positif oder null), bei der r die Zahl der Linien und c die Zahl der Spalten des getesteten Bildschirm ist. Die Linien sind von oben nach unten nummeriert von 0 bis $r-1$ und die Spalten sind von links nachs rechts nummeriert, von 0 bis $c-1$.

Der Wert $pix[i][j]$ representiert den Zustand des Pixels, das sich in der Linie i und der Spalte j befindet. Je höher der Wert ist, desto besser funktioniert das Pixel. Um festzustellen ob ein Pixel defekt ist, vergleicht man $pix[i][j]$ mit dem Wert Q, der vom Bildschirmtyp und den Qualitätsansprüchen abhängt. Das Pixel funktioniert korrekt, wenn $pix[i][j] \geq Q$ (anders gesagt, das Pixel ist defekt, wenn $pix[i][j] < Q$).

Hier ein Beispiel, das das Resultat eines Bildschirmtests eines Bildschirms mit 6 Linien und 12 Spalten anzeigt.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
0	141	101	40	207	129	157	247	244	235	229	108	159
1	175	223	222	254	196	162	152	127	235	243	161	183
2	232	123	226	72	245	109	99	87	254	115	142	0
3	172	123	130	178	155	100	198	226	157	110	202	183
4	112	198	212	241	233	191	218	68	76	72	248	162
5	127	137	110	219	196	194	111	213	141	111	249	147

pix[ ][ ]

Wenn der Qualitätswert Q auf 100 festgelegt ist, dann werden die 8 grauen Pixel als defekt deklariert.
Das Pixel der Linie 0 und der Spalte 2 ist defekt, da $pix[0][2] = 40$ ist kleiner als $Q = 100$.
Mit diesem Q-Wert enthalten 3 Linien und 7 Spalten defekte Pixel. Es geben 3 aufeinander folgende, defekte Pixel in der Linie 4, aber die Linie mit den meisten defekten Pixeln ist die Linie 2, die 4 defekte Pixel enthält.

Vervollständige die folgenden Algorithmen, in dem du `...` ersetzt, um das gefragte Resultat zu errechnen. In jedem Algorithmus kannst du die Tabelle $pix[\,][\,]$ nutzen, welche die Resultate des T Bildschirmtesters enthält, die Variabeln r und c, die die Zahl der Linien und Spalten des Bildschirms enthält und die Variable Q, die die Qualitätsgrenze enthält.

Pixel-Algorithmus : Wirf den Bildschirm weg, wenn die defekten Pixel das Limit überschreiten.

```
Input  : pix[][], r, c und Q.
         dmax, die maximale Anzahl defekter Pixel, die akzeptiert werden kann.
Output : "XX" wenn die defekten Pixel mehr sind als dmax,
         "OK" wenn die defekten Pixel weniger oder gleich sind an dmax.

d ← 0
for (i ← 0 to r-1 step 1)
{
    for (j ← 0 to c-1 step 1)
    {
        if ( ... )                    //(a)
        {
            d ← ...                   //(b)
        }
    }
}


if ( ... )                            //(c)
{
    return "XX"
}
else
{
    return "OK"
}
```

Q1(a) [2 pts]	Welches ist die Anweisung (a) im Pixel-Algorithmus?
Solution: $pix[i][j] < Q$	

Q1(b) [1 pt]	Welches ist die Anweisung (b) im Pixel-Algorithmus?
Solution: $d + 1$	

Q1(c) [2 pts]	Welches ist die Anweisung (c) im Pixel-Algorithmus?
Solution: $d > dmax$ (andere Lösung: $dmax < d$)	

Linien-Algorithmus : Zähle die Anzahl Linien mit mindestens einem defektem Pixel.

```
Input  : pix[][], r, c und Q.
Output : dr, die Anzahl der Linien mit mindestens einem defektem Pixel.

dr ← 0
for (i ← 0 to r-1 step 1)
{
    j ← 0
    while (j < c) and (...)      //(d)
    {
        j ← ...                  //(e)
    }

    if ( ... )                   //(f)
    {
        ...                      //(g)
    }
}
return dr
```

Bemerke, wie ein Computer Anweisungen vom Typ (Bedingung 1) **and** (Bedingung 2) auswertet :
wenn (Bedingung 1) **false** ist, wird (Bedingung 2) nicht ausgewertet. Das ist unnötig, da man schon weiss, dass die Anweisung schon **false** ist.

Q1(d) [2 pts]	**Welches ist die Anweisung (d) im Linien-Algorithmus?**
Solution: $pix[i][j] \geq Q$	

Q1(e) [2 pts]	**Welches ist die Anweisung (e) im Linien-Algorithmus?**
Solution: $j + 1$	

Q1(f) [2 pts]	**Welches ist die Anweisung (f) im Linien-Algorithmus?**
Solution: $j < c$ (andere Lösung: $j \neq c$)	

Q1(g) [2 pts]	**Welches ist die Anweisung (g) im Linien-Algorithmus?**
Solution: $dr \leftarrow dr + 1$ (andere akzeptierte Antwort: dr++)	

Man zählt die defekten Pixel in einem Rechteck, von dem man die Position des oberen linken Pixels (Linie $r1$, Spalte $c1$) und die Position des unteren rechten Pixel (Linie $r2$, Spalte $c2$) angibt.
Der folgende Algorithmus durchläuft die ganze Tabelle, um das Rechteck zu analysieren.

Rechteck-Algorithmus : Zählt die Anzahl defekter Pixel des Rechtecks.

```
Input  : pix[][], r, c und Q.
         r1 und c1 die Linie und Spalte des oberen, linken Pixels.
         r2 und c2 die Linie und Spalte des unteren, rechten Pixels.
         0 ≤ r1 ≤ r2 < r und 0 ≤ c1 ≤ c2 < c
Output : d, die Anzahl defekter Pixel des Rechtecks.
d ← 0
for (i ← r1 to ... step 1)          //(h)
{
    for (j ← ... to ... step 1)     //(i) et (j)
    {
        if ( ... )                  //(k)
        {
            ...                     //(l)
        }
    }
}
return d
```

Q1(h) [1 pt]	**Welches ist die Anweisung (h) im Rechteck-Algorithmus?**
Solution: $r2$	

Q1(i) [1 pt]	**Welches ist die Anweisung (i) im Rechteck-Algorithmus?**
Solution: $c1$	

Q1(j) [1 pt]	**Welches ist die Anweisung (j) im Rechteck-Algorithmus?**
Solution: $c2$	

Q1(k) [1 pt]	**Welches ist die Anweisung (k) im Rechteck-Algorithmus?**
Solution: $pix[i][j] < Q$	

Q1(l) [1 pt]	**Welches ist die Anweisung (l) im Rechteck-Algorithmus?**
Solution: $d \leftarrow d + 1$	

Q1(m) [2 pts]	**Wie oft wird die Anweisung (k) ausgerechnet, wenn $r1 = 500$, $c1 = 480$, $r2 = 599$, $c2 = 511$?**
Solution: 3200 (genauer: $100 \times 32 = 3200$)	

Frage 2 – Pixel-Suite

Diese Frage nutzt die gleichen Variabeln wie die vorherige ("Pixel-Einführung").
Um die Angabe zu verstehen solltest du zuerst "Pixel-Einführung" lesen.
Du kannst dennoch alles beantworten, selbst wenn du nicht alles in "Pixel-Einführung" beantwortet hattest.

Man muss manchmal hintereinander, defekte Pixel in verschiedenen Rechtecken zählen, die unterschiedliche Größen und Positionen auf dem Bildschirm haben.
In dem Fall ist der **Rechteck-Algorithmus** der "Pixel-Einführung" nicht effektiv; es geht auch schneller.
Dafür muss man eine neue Tabelle `pixtot[ ][ ]` nutzen, die sehr schnell (ohne **`for`** noch **`while`** Schleife) die Anzahl defekter Pixel des Rechtecks anzeigt.

`pixtot[ ][ ]` hat die selbe Anzahl Linien und Spalten wie `pix[ ][ ]`.
`pixtot[i][j]` ist die Anzahl defekter Pixel des Rechtecks mit Koordinaten (Linie 0, Spalte 0) bis Linie *i* und Spalte *j*.

Hier nun Beispiele mit dem kleinen Bildschirm aus "Pixel-Einführung".
(Die Farben sind eingefügt worden, um die verschiedenen Werte besser zu sehen.)

	0	1	2	3	4	5	6	7	8	9	10	11
0	141	101	40	207	129	157	247	244	235	229	108	159
1	175	223	222	254	196	162	152	127	235	243	161	183
2	232	123	226	72	245	109	99	87	254	115	142	0
3	172	123	130	178	155	100	198	226	157	110	202	183
4	112	198	212	241	233	191	218	68	76	72	248	162
5	127	137	110	219	196	194	111	213	141	111	249	147

pix[ ][ ]

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

pixtot[3][6] = 3

Es gibt 3 defekte Pixel im Rechteck von der Ecke oben links bis Linie 3 und Spalte 6.

	0	1	2	3	4	5	6	7	8	9	10	11
0	141	101	40	207	129	157	247	244	235	229	108	159
1	175	223	222	254	196	162	152	127	235	243	161	183
2	232	123	226	72	245	109	99	87	254	115	142	0
3	172	123	130	178	155	100	198	226	157	110	202	183
4	112	198	212	241	233	191	218	68	76	72	248	162
5	127	137	110	219	196	194	111	213	141	111	249	147

pix[ ][ ]

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

pixtot[2][7] = 4

Es gibt 4 defekte Pixel im Rechteck von der Ecke oben links bis Linie 2 und Spalte 7.

Hier die Tabelle $pixtot[\,][\,]$ eines *anderen Bildschirms* mit 8 Linien und 16 Spalten.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2
2	0	0	0	1	1	1	1	1	1	3	4	5	5	5	5	6
3	0	0	0	1	1	1	1	1	1	3	4	5	5	5	5	6
4	0	0	0	1	1	2	3	3	3	5	6	7	7	7	7	8
5	0	0	0	1	1	2	3	3	3	5	6	7	7	7	7	9
6	0	0	0	1	1	2	3	3	3	6	7	8	8	8	8	10
7	0	0	0	1	1	2	3	3	3	6	7	8	8	8	8	10

pixtot[ ][ ]

Benutze diese Tabelle um die Fragen zu beantworten.

Q2(a) [1 pt]	**Wie viele defekte Pixel gibt es auf diesem Bildschirm?**
Solution: 10	

Q2(b) [1 pt]	**Wie viele defekte Pixel gibt es auf Linie 2?**
Solution: 4	

Q2(c) [1 pt]	**Welche Linien haben keine defekten Pixel (gib die Zahlen getrennt durch Kommas an) ?**
Solution: 3, 7	

Q2(d) [1 pt]	**Wie viele Spalten haben keine defekten Pixel?**
Solution: 9	

In den folgenden Fragen benutzen wir folgende **Schreibweise** :

- "Pixel $(r1, c1)$" für das Pixel der Linie $r1$ und Spalte $c1$,
- "Rechteck $(r1, c1) - (r2, c2)$" für das Rechteck mit linker, oberen Ecke auf Pixel $(r1, c1)$ und rechten, unteren Ecke auf Pixel auf $(r2, c2)$.

Q2(e) [2 pts]	**Wie viele defekte Pixel gibt es im Rechteck** $(0, 0) - (6, 6)$ **?**
Solution: 3	

Q2(f) [2 pts]	**Wie viele defekte Pixel gibt es im Rechteck** $(3, 3) - (6, 6)$ **?**
Solution: 2	

Q2(g) [2 pts]	**Wie viele defekte Pixel gibt es im Rechteck** $(2, 0) - (2, 12)$ **?**
Solution: 3	

Q2(h) [2 pts]	**Ist Pixel** $(4, 9)$ **defekt?**
Solution: $NEIN$	

Q2(i) [3 pts]	**Wie viele defekte Pixel gibt es im Rechteck** $(2, 4) - (6, 10)$ **?**
Solution: 4	

Q2(j) [5 pts]	**Generell, wenn** $1 \leq r1 < r2 < r$ **und** $1 \leq c1 < c2 < c$**, gib die Anzahl defekter Pixel des Dreiecks** $(r1, c1) - (r2, c2)$ **in Funktion des Elements** $pixtot[\,][\,]$ **an?**
Solution: $pixtot[r2][c2] - pixtot[r1 - 1][c2] - pixtot[r2][c1 - 1] + pixtot[r1 - 1][c1 - 1]$	

Du musst den Algorithmus vervollständigen "fülle pixtot", der erlaubt $pixtot[\,][\,]$ anhand von $pix[\,][\,]$ zu berechnen.
Dieser Algorithmus erhält die Tabelle $pix[\,][\,]$ des Testgeräts, wie in "Pixel-Einführung" festgelegt.
Zu Beginn ist die Tabelle $pixtot[\,][\,]$ gefüllt mit 0: für alle i und j, $pixtot[i][j] = 0$.
Der Algorithmus muss $pixtot[\,][\,]$ befüllen, damit für alle i und j $pixtot[i][j]$ die Anzahl der defekten Pixel des Rechtecks $(0,0) - (i,j)$ enthält.

pixtot-Füllen-Algorithmus : Nutze die Tabelle $pix[\,][\,]$ um die Tabelle $pixtot[\,][\,]$ zu füllen.

```
Input  : pix[][], r, c, Q und pixtot[][] initialisiert mit 0.
Output : pixtot[][] gefüllt wie erklärt.

//Pixel oben links des Bildschirms.
if (pix[0][0] < Q )
{
    pixtot[0][0]=1
}

//Rest der ersten Linie.
for (j ← 1 to c-1 step 1)
{
    pixtot[0][j] ← pixtot[0][j-1]
    if (pix[0][j] < Q )
    {
        pixtot[0][j] ← ...              //k
    }
}

//Andere Linien.
for (i ← 1 to r-1 step 1)
{
    d = 0 //Zähler der defekten Pixel der Linie.
    for (j ← 0 to c-1 step 1)
    {
        if (pix[i][j] < Q )
        {
            ...                         //l
        }
        pixtot[i][j] ← ...              //m
    }
}
return pixtot
```

Q2(k) [2 pts]	**Welches ist die Anweisung (k) im Füllen-Algorithmus pixtot ?**
Solution: $pixtot[0][j] + 1$ (andere Antwort: $pixtot[0][j-1] + 1$)	

Q2(l) [2 pts]	**Welches ist die Anweisung (l) im Füllen-Algorithmus pixtot ?**
Solution: $d \leftarrow d + 1$	

Q2(m) [4 pts]	**Welches ist die Anweisung (m) im Füllen-Algorithmus pixtot ?**
Solution: $pixtot[i-1][j] + d$	


Solutions

Question 3 – Arthur, Merlin, das Seemonster und die anderen...

König Arthur bräuchte schnell sein Excalibur-Schwert, um einen Drachen zu besiegen, der in der Nachbarschaft herumläuft. Merlin, der Magier, erklärt, dass es mehrere Schritte gibt, um dorthin zu gelangen. Leider versagt Merlins Gedächtnis (er ist 103 Jahre alt!) und er ist nicht mehr in der Lage, die Vorgehensweise sehr strukturiert zu erklären. An Folgendes erinnert er sich noch:

(a) Die Goblins werden einen Diamanten für drei Goldmünzen geben.

(b) Die Hexe wird einen Liebestrank für ein Stück Kupfer zubereiten.

(c) Das Seemonster will einen Liebestrank und einen Strauß Rosen, um eine Meerjungfrau zu verführen. Im Gegenzug wird er Arthur Excalibur geben.

(d) Die Goblinbank wird eine Silbermünze und zwei Kupfermünzen für zwei Goldmünzen geben.

(e) Die Dame des Sees wird Excalibur für einen Diamanten geben.

(f) Du kannst einen Strauß Rosen für eine Silbermünze auf dem Markt kaufen.

Merlin gibt Arthur **vier Goldmünzen** und sagt ihm, dass er Excalibur mit diesen Aktionen bekommen sollte.
Er präzisiert, dass Arthur nicht verpflichtet ist, alle diese Aktionen durchzuführen, dass er jedoch jede einzelne bei Bedarf mehrmals ausgeführen kann.

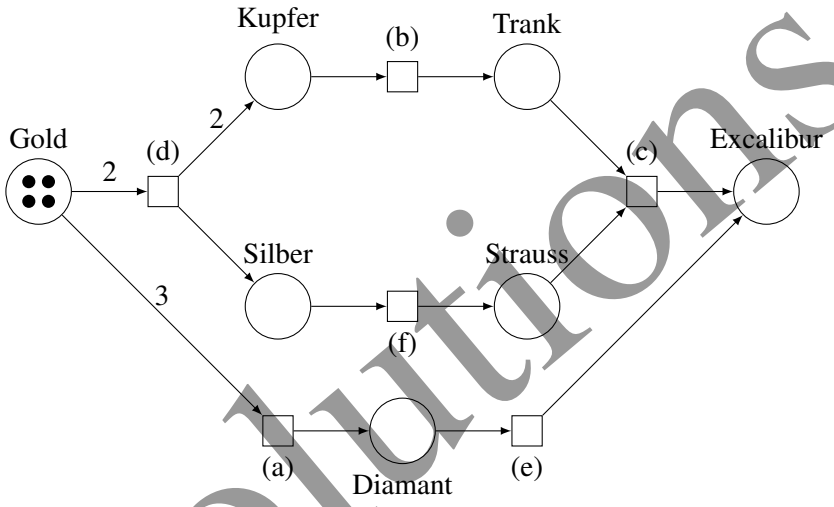
Arthur beginnt, über die Methode nachzudenken, die er anwenden könnte, beginnend mit den vier Goldmünzen, die Merlin ihm gab. Es sieht mehrere Abfolgen von Aktionen vor. Zum Beispiel: (a), (b), (c), was bedeutet, dass Arthur zunächst drei Goldmünzen gegen einen Diamanten eintauscht (so dass er jetzt eine Goldmünze und einen Diamanten hat), versucht dann, von der Hexe einen Liebestrank zu kaufen, was unmöglich ist, weil er zur Zeit keine Kupfermünze hat... Arthur lässt sich nicht entmutigen...

Gib für die folgenden Handlungsabläufe durch Ankreuzen der entsprechenden Kästchen auf dem Antwortbogen an, ob sie "Unmöglich" oder "Machbar" sind. Für die Machbaren, die es dir ermöglichen, Excalibur zu erhalten, kreuze **auch** auf das Kästchen "Excalibur" an.

	Unmöglich	Machbar	Excalibur	Aktionsabfolgen
Q3(a) [1 pt]	☐	☒	☐	(d), (f), (b)
Q3(b) [1 pt]	☒	☐	☐	(d), (b), (c), (f)
Q3(c) [1 pt]	☒	☐	☐	(a), (d), (b), (f), (c)
Q3(d) [1 pt]	☐	☒	☒	(a), (e)
Q3(e) [1 pt]	☒	☐	☐	(d), (a), (e)
Q3(f) [1 pt]	☐	☒	☒	(d), (d), (b), (f), (b), (b), (f), (c)

Q3(g) [2 pts]	**Gib alle Abfolgen von Aktionen, die es ermöglichen, Excalibur in einer minimalen Anzahl von Schritten zu erhalten, während du Arthur zwei Goldstücke überlässt.**
Solution: (d), (b), (f), (c) und (d), (f), (b), (c)	

Arthur verflucht Merlin und sein schlechtes Gedächtnis.
Er beschließt, ein Bild in den Sand zu zeichnen, um sich weiter zu helfen. Er unterscheidet die *Ressourcen* (Goldmünzen, Diamanten, Tränke, . . .), die er mit Kreisen darstellt, und die *Aktionen*, die er mit Rechtecken darstellt.
Er setzt Steine in die Kreise, um die Anzahl der ihm zur Verfügung stehenden Ressourcen zu zählen. Zum Beispiel gibt es zu Beginn seiner Suche vier Steine im "Goldmünzenkreis".
Es zieht einen Pfeil von einer Ressource R zu einer Aktion A, wenn A R benötigt, um stattfinden zu können, und gibt auf dem Pfeil die Menge der benötigten Ressourcen an. Zum Beispiel gibt es einen Pfeil namens 2, der vom "Goldmünzenkreis" bis (d) geht.
Ebenso zeigt ein Pfeil von einer Aktion A zu einer Ressource R an, dass die Aktion A diese Ressource R produziert (und in welcher Menge).
Um zu vereinfachen, zeigt Arthur nichts auf dem Pfeil an, wenn die produzierten oder verbrauchten Menge der Ressourcen 1 ist.
Das ist die Zeichnung, die er bekommt:



Wenn Arthur zum Beispiel jetzt die Aktion (d) ausführt, sind nur noch zwei Steine in der Ressource "Gold" vorhanden, aber es wird zwei in der Ressourcen "Kupfer" und eine Ressource "Silber" geben.
Ebenso, um die Aktion (c) auszuführen, brauchst du mindestens einen Stein in der Ressource "Trank" **und** *mindestens* 1 Stein in der Ressource "Blumenstrauß".
Die Durchführung dieser Aktion (c) entfernt einen Stein von diesen beiden Ressourcen und fügt einen in "Excalibur" hinzu, dem Ziel von Arthurs Suche.
Um eine Aktion auszuführen, ist es daher notwendig, dass die Bedingungen, die von *all* den eingehenden Pfeilen der Aktion vorgegeben sind, erfüllt sind, und es löst die Effekte aus, die durch *alle* ausgehenden Pfeilen einer Aktion angegeben werden.

Angenommen, Arthur beginnt mit vier Steinen in "Gold" und keine Steine in den anderen Ressourcen, wie viele Steine wird es in jeder der Ressourcen geben, nachdem Arthur die folgenden Handlungsabläufe abgeschlossen hat?

Q3(h) [6 pts]	**Nach (d), (b), (f) ?**
Solution: Gold: 2 Kupfer: 1 Silber: 0 Trank: 1 Bouquet: 1 Diamant: 0 Excalibur: 0	

Q3(i) [6 pts]	**Nach (d), (b), (b), (d), (b), (f) ?**
Solution: Gold: 0 Kupfer: 1 Silber: 1 Trank: 3 Bouquet: 1 Diamant: 0 Excalibur: 0	

Arthur beginnt dann eine neue Suche, um den Heiligen Gral zu finden. Das sind Merlins neue Anweisungen:

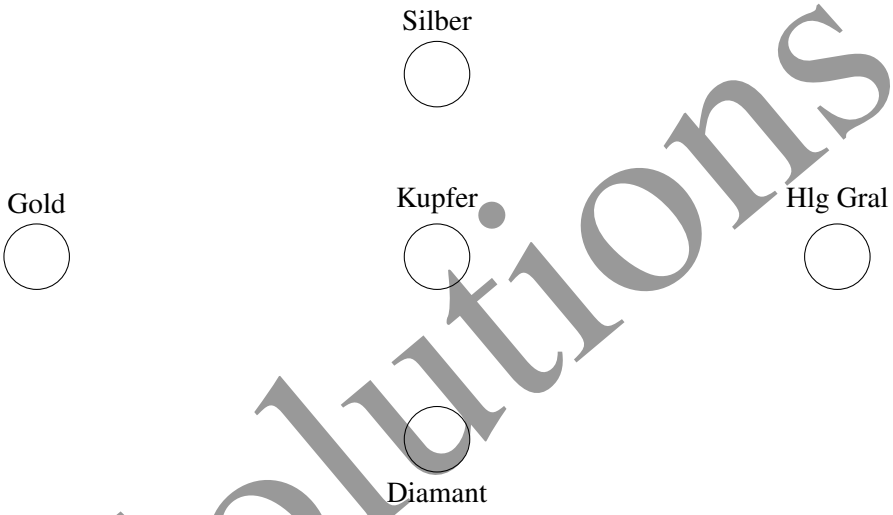
(a) Die Goblins geben dir zwei Silber- und eine Kupfermünze für eine Goldmünze.

(b) Der Juwelier wird dir einen Diamanten für zwei Goldmünzen verkaufen.

(c) Mein alter Freund Gandalf ist derart kurzsichtig und grosszügig, dass er dir eine Goldmünze für jede Silbermünze gibt, die du ihm gibst.

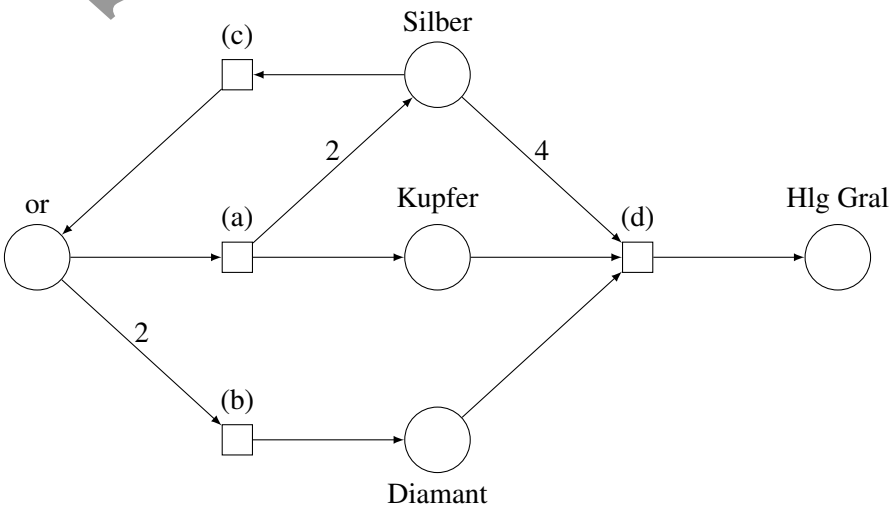
(d) Die Amazonen haben einen neuen Online-Shop eröffnet, wo du den Heiligen Gral für vier Silbermünzen, einen Kupfermünze und einen Diamanten kaufen kannst!

Wie bei der vorherigen Suche möchte Arthur eine Zeichnung haben, die diese Möglichkeiten zusammenfasst. Er beginnt damit, Kreise (Ressourcen) zu zeichnen, braucht aber deine Hilfe, um die Aktionen und die Pfeile hinzuzufügen.



Q3(j) [8 pts] **Füge die Aktionen (a), (b), (c) und (d) und die Pfeile (mit Mengen) gemäß Merlins Beschreibung hinzu. (machen deine Tests in einer Kladde, bevor du sie sauber in den Antwortbogen kopierst).**

Solution:



Jetzt, wo Arthur die Zeichnung hat, stellt er sich selbst ein paar Fragen. Kannst du ihm helfen?

Q3(k) [2 pts]	**Ist die Sequenz (a), (c), (c), (a), (c), (c), (a), (c), (c) möglich, wenn man mit nur einem Goldstück beginnt? Wenn ja, wie viele Goldmünzen wird Arthur danach haben?**
Solution: JA, 4 Goldmünzen.	

Q3(l) [2 pts]	**Was ist die minimale Anzahl von Goldstücken, die Arthur anfangs benötigt, um den Heiligen Gral zu erhalten?**
Solution: 1	

Q3(m) [3 pts]	**Ausgehend von dieser Anzahl von Goldmünzen, was ist die Mindestanzahl von Aktionen, die Arthur ausführen muss, um den Heiligen Gral zu erhalten ?**
Solution: 13, z.B. mit der Sequenz: (a), (c), (c), (a), (c), (c), (a), (c), (c), (a), (a), (b), (d)	

Question 4 – Nombres au tableau

Alice amüsiert sich mit ganzen, positiven Zahlen, die in einer schwarzen Tabelle stehen. Sie sucht 2 Zahlen aus und löscht sie, dann schreibt sie entweder ihre Summe, oder die absolute Differenz[1], oder ihr Produkt auf.
Es gibt jetzt ein Zahl weniger in der Tabelle und Alice wiederholt dies bis nur noch eine Zahl vorhanden ist.
Alice will *die kleinst mögliche für diese letzte Zahl* erhalten, sie nennt diese minimale Zahl *die Antwort*.

Beispiel mit 1, 4, 6: Alice könnte mit addieren beginnen 1 und 6, was 4 und 7 in der Tabelle übrig lässt, welche sie danach durch ihre absolute Differenz ersetzt $|4-7|=3$. Aber sie könnte es besser machen: beginnend mit der Addition 1 und 4, bleibt 5 und 6 in der Tabelle, anschliessend ihre absolute Differenz $|5-6|=1$. Es ist nicht möglich, es besser zu machen, also ist *die Antwort*, die Alice sucht 1.

Beispiel 2, 4, 8: Alice ersetzt 2 und 4 durch ihr Produkt 8 in der Tabelle. Dann kann man die Differenz nehmen $|8-8|$ um 0 zu erhalten, was das kleinst mögliche Resultat ergibt und *die Antwort* ist.

Q4(a) [2 pts]	**Welches ist die Antwort zu Liste 2, 3, 4, 29 ?**
Solution: 5	

Q4(b) [2 pts]	**Welches ist die Antwort zu Liste 4, 6, 7 ?**
Solution: 3	

Q4(c) [2 pts]	**Welches ist die Antwort zu Liste 1, 1, 100 ?**
Solution: 0	

Q4(d) [2 pts]	**Welches ist die Antwort zu Liste 1, 2, 4, 8 ?**
Solution: 0	

Q4(e) [2 pts]	**Welches ist die Antwort zu Liste 1, 3, 5, 16 ?**
Solution: 0	

In der folgenden Frage, sucht man eine Liste für die *die Antwort* vorgegeben ist. Wenn du eine findest, schreibe die Elemente getrennt durch Komma auf. Schreibe "NEIN" wenn es keine gibt. Eine Liste kann mehrere Mal die selbe Ziffer enthalten.
Achtung, denke daran, dass *die Antwort* der *kleinste Wert* ist, den man am Ende erreichen kann.

Q4(f) [2 pts]	**Existiert eine ganze Liste 4 zwischen 1 und 4 für welche die Antwort 0 ist?**
Solution: 1,1,1,1 (alles mit 2 mal dem gleichen Wert funktioniert)	

Q4(g) [2 pts]	**Existiert eine ganze Liste 3 zwischen 1 und 4 für welche die Antwort 1 ist?**
Solution: 1,2,4 oder 2,3,4	

[1] Die absolute Differenz von zwei Zahlen x und y ist der absolute Wert ihrer Differenz, die $|x-y|$ geschrieben wird und erhalten wird indem man die kleine von der großen Zahl abziehst. Beispiele: die absolute Differenz von 3 und 5 ergibt $|3-5|=|5-3|=2$, die absolute Differenz von 7 und 7 ergibt $|7-7|=0$.

Q4(h) [2 pts]	**Existiert eine ganze Liste 3 zwischen 1 und 4 für welche die Antwort 2 ist?**
Solution: NEIN	

Q4(i) [3 pts]	**Existiert eine ganze Liste 4 zwischen 1 und 10 für welche die Antwort 1 ist?**
Solution: 1,4,6,8 oder 1,4,7,9 oder 2,6,9,10 oder 3,6,7,8 oder 6,7,8,10	

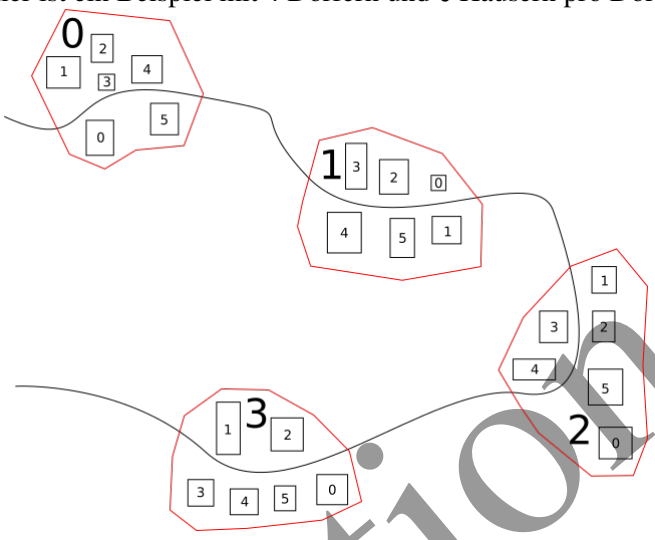
Q4(j) [3 pts]	**Existiert eine ganze Liste 4 zwischen 1 und 100 für welche die Antwort 80 ist?**
Solution: zum beispiel 1,2,4,92 oder 1,2,5,95 oder 1,3,5,100 oder 1,85,96,100	

Q4(k) [3 pts]	**Existiert eine ganze Liste 5 zwischen 1 und 200 für welche die Antwort 60 ist?**
Solution: zum beispiel 1,2,4,11,192	

Frage 5 – Zwei Nummerierungen

Eine gebirgige Region besteht aus n Dörfern entlang der gleichen Straße. Die Dörfer sind von 0 bis $n-1$ durchnummeriert und enthalten alle die gleiche Anzahl von Häusern. In jedem Dorf werden die Häuser durch eine **lokale Nummer** von 0 bis $t-1$ gekennzeichnet.

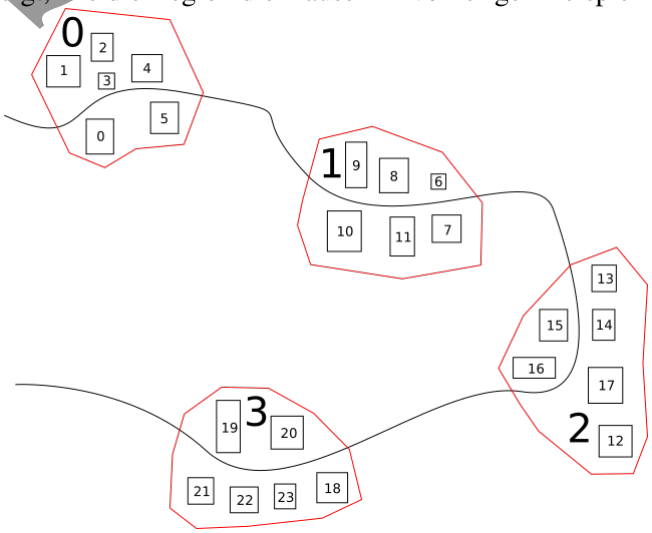
Hier ist ein Beispiel mit 4 Dörfern und 6 Häusern pro Dorf.



Die Dörfer sind rot mit ihren Zahlen in großen Buchstaben geschrieben.
Jedes Haus wird durch ein kleines Rechteck mit seiner lokalen Nummer dargestellt.

Jedes Haus hat auch eine **regionale Nummer**, die von der Region vergeben wird, die alle Dörfer verwaltet. Regionale Nummern werden in aufsteigender Reihenfolge ausgehend von 0 vergeben, indem die Dörfer in der Reihenfolge ihrer Nummern und der lokalen Reihenfolge der Häuser in jedem Dorf durchlaufen werden. Also fangen wir mit den Häusern des Dorfes 0 an, weiter geht's mit denen des Dorfes 1, dann mit denen des Dorfes 2,...

Die folgende Zeichnung soll dir helfen, das zu verstehen.
Es wird gezeigt, wie die Region die Häuser im vorherigen Beispiel nummeriert.



Die **regionale Nummer** jedes Hauses ist in dem kleinen Rechteck vermerkt, das es repräsentiert.

Q5(a) [2 pts]	**Wenn es** 13 **Dörfer mit** 12 **Häusern gibt, was ist die regionale Nummer des Hauses mit der lokalen Nummer** 3 **im Dorf** 6 **?**
Solution: 75	

Q5(b) [2 pts]	**Wenn die Dörfer** 17 **Häuser haben, was ist die regionale Nummer des Hauses mit der lokalen Nummer** 14 **im Dorf** 5 **?**
Solution: 99	

Q5(c) [2 pts]	**Wenn die Dörfer** 19 **Häuser haben, was ist die Nummer des Dorfes und die lokale Nummer des Hauses mit der regionalen Nummer** 107 **?**
Solution: Nummer des Dorfes 5 , lokale Nummer 12	

Q5(d) [2 pts]	**Wenn das Haus mit der lokalen Nummer** 13 **im Dorf** 3 **die regionale Nummer** 64 **hat, wie hoch ist die Anzahl der Häuser in jedem Dorf?**
Solution: 17	

Q5(e) [2 pts]	**Wenn es n Dörfer mit t Häusern gibt, gib einen Ausdruck der regionalen Nummer des Hauses mit der lokalen Nummer a im Dorf b an.**
Solution: $b \cdot t + a$	

Herr Dubois muss alle Häuser in der Region in einer ganz bestimmten Reihenfolge besuchen. Er muss mit den Häusern mit der lokalen Nummer 0 beginnen, in der Reihenfolge der Dörfer, mit den Häusern mit der lokalen Nummer 1 fortfahren, in Reihenfolge der Dörfer, und so weiter, bis er mit dem Haus mit der größten lokalen Nummer im letzten Dorf endet.
Ein Beamter in der Region muss eine Liste der Regionalnummern in der Reihenfolge der Besuche erstellen. Zum Beispiel, mit 4 Dörfern mit jeweils 6 Häusern (wie auf den Zeichnungen), ist die Liste 0, 6, 12, 18, 1, 7, 13, 19, 2, 8, 14, 20, 3, 9, 15, 21, 4, 10, 16, 22, 5, 11, 17, 23.
Der Algorithmus **Besuche** erstellt diese Art von Liste unter Berücksichtigung der Anzahl der Dörfer und Häuser pro Dorf. Er füllt eine Tabelle $V[\,]$ aus (die so viele Elemente enthält, wie es Häuser in der Region gibt), so dass für jeden Index i, $V[i]$ die regionale Nummer des Hauses ist, in das Herr Dubois während des $i-ten$-Besuchs gehen soll (wie üblich sind die Besuche von 0 durchnummeriert).

Algorithmus Besuche :

```
Input: n, die Anzahl der Dörfer.
       t, die Anzahl der Häuser pro Dorf.
       V[ ], die Tabelle wurde mit 0 initialisiert.
Output: V[ ],  gefüllt mit regionalen Nummern in der Reihenfolge der Besuche.

k ← 0
for (i ← 0 to t-1 step 1)
{
    for (j ← 0 to n-1 step 1)
    {
        V[...] ← ...        //(f) (g)
        k ← k+1
    }
```

```
}
return V
```

Q5(f) [2 pts]	Was ist der Ausdruck (f) im Besuche-Algorithmus?
Solution: k	

Q5(g) [3 pts]	Was ist der Ausdruck (g) im Besuche-Algorithmus?
Solution: $j \times t + i$	

Bevor wir fortfahren, erinnern wir uns daran, dass, wenn a und b ganze Zahlen sind, dann sind a/b und $a\%b$ jeweils der Quotient und der Rest der ganzen Division von a durch b.
Wenn man zum Beispiel 31 durch 7 teilt, dann erhält man 4 und einen Rest von 3. Also $31/7 = 4$ und $31\%7 = 3$.

Q5(h) [2 pts]	**Wenn es n Dörfer mit t Häusern gibt, gib einen Ausdruck der Dorfnummer des Hauses an, das die regionale Nummer r hat.**
Solution: r/t	

Q5(i) [2 pts]	**Wenn es n Dörfer mit t Häusern gibt, gib einen Ausdruck der lokalen Nummer des Hauses an, das die regionale Nummer r hat.**
Solution: $r\%t$	

Der Algorithmus **Besuche 2** liefert das gleiche Ergebnis wie der Algorithmus **Besuche** mit einer einzigen Schleife.
Kannst du es vervollständigen?
Hilfe: Der zu findende Ausdruck hat die Form $(...) \times (...) + (...)/(...)$
Nicht alle Klammern sind notwendig, aber wir müssen auch % verwenden.

Algorithmus Besuche 2 :

```
Input: n, die Anzahl der Dörfer.
       t, die Anzahl der Häuser pro Dorf.
       V[ ], das Array wurde mit 0 initialisiert.
Output: V[ ],  gefüllt mit regionalen Nummern in der Reihenfolge der Besuche.

for (i ← 0 to (n × t − 1) step 1)
{
       V[i] ← ...
}
return V
```

Q5(j) [6 pts]	Was ist der fehlende Ausdruck im Besuche 2 Algorithmus?
Solution: $(i\%n) \times t + i/n$	

Hier ist ein dritter Algorithmus, der das gleiche Ergebnis wie **Besuche** und **Besuche 2** liefert.
Kannst du es vervollständigen?
Sei vorsichtig, verbringe nicht zu viel Zeit damit, bevor du den Rest des Fragebogens abschließt!

Algorithmus Besuche 3 :

```
Input: n, die Anzahl der Dörfer.
       t, die Anzahl der Häuser pro Dorf.
       V[ ], das Array wurde mit 0 initialisiert.
Output: V[ ], gefüllt mit regionalen Nummern in der Reihenfolge der Besuche.

for (i ← 0 to (n × t − 1) step 1)
{
        V[...] ← i
}
return V
```

Q5(k) [5 pts]	Was ist der fehlende Ausdruck im Besuche 3 Algorithmus?
Solution: $(i\%t) \times n + i/t$	