be-OI 2024 **Finale - KADETT** Samstag 23. März 2024	**Füllt diesen Rahmen bitte in GROSSBUCHSTABEN und LESERLICH aus** VORNAME : ..... NAME : ..... SCHULE :	O **Reserviert**

Belgische Informatik-Olympiade 2024 (Dauer: maximal 2 Stunden)

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 **Kadett**.
   - Für Schüler im dritten oder vierten Jahr der Sekundarschule: Kategorie **Junior**.
   - Für Schüler der Sekundarstufe 5 und höher: Kategorie **Senior**.
2. Gib deinen Namen, Vornamen und deine Schule **nur auf dieser Seite** an.
3. **Die Antworten** sind auf den dafür vorgesehenen Antwortbogen einzutragen. Schreibe **gut lesbar** mit einem **blauen oder schwarzen Stift oder Kugelschreiber**.
4. Verwende einen Bleistift und einen Radiergummi, wenn du am Entwurf in den Frageblättern arbeitest.
5. Du darfst nur Schreibmaterial dabei haben; Taschenrechner, Mobiltelefone, ... sind **verboten**.
6. Du kannst jederzeit weitere Kladdeblätter bei einer Aufsichtsperson fragen.
7. Wenn du fertig bist, gibst du die erste Seite (mit deinem Namen) und die letzten Seiten (mit den Antworten) ab, du kannst die anderen Seiten behalten.
8. Alle Codeauszüge aus der Anweisung sind in **Pseudo-Code**. Auf den folgenden Seiten findest du eine **Beschreibung** des Pseudo-Code, den wir verwenden.
9. Wenn du mit einem Code antworten musst, dann benutze den **Pseudo-Code** oder eine **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 unserer Mitglieder:

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 Arrays (Tabellen) speichern (siehe weiter unten), sowie boolesche (logische) Werte: wahr/richtig (**`true`**) oder falsch/fehlerhaft (**`false`**). Es ist möglich, arithmetische Operationen auf Variablen durchzuführen. Zusätzlich zu den vier traditionellen Operatoren ($+$, $-$, $\times$ und $/$), kann man auch den Operator % verwenden. Wenn a und b ganze Zahlen sind, dann stellt a/b den Quotienten und $a\%b$ 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 $alter$ den Wert 17 erhält.

```
geburtsjahr ← 2007
alter ← 2024 − geburtsjahr
```

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 $preis$. Beobachte 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.
}
else
{
    preis ← 0 // Kostenlos fuer ein Kind unter 6 Jahren.
}
```

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

Um Code zu wiederholen, z.B. um durch die Elemente eines Arrays zu laufen, kann man eine **for**-Schleife verwenden. Die Schreibweise **for** ($i \leftarrow a$ **to** b **step** k) stellt eine Schleife dar, 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 dem Array $tab[\,]$. Angenommen, das Array ist n lang. Die Summe befindet sich am Ende der Ausführung des Algorithmus in der Variable $summe$.

```
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 durch 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 Werte 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 eines Arrays.

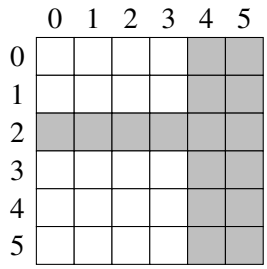
```
Eingabe: tab[ ], ein Array mit n Zahlen.
         n, die Anzahl der Elemente des Arrays.
Ausgabe: sum, die Summe aller im Array enthaltenen Zahlen.

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 – Tests

Sie müssen die Kästchen einer Tabelle schwärzen, indem Sie einen logischen Test auswerten, bei dem es um die Zeilen- und Spaltennummern der Kästchen geht. Die Variable `i` enthält eine Zeilennummer und die Variable `j` enthält eine Spaltennummer. Wenn der Test für ein Wertepaar `i` und `j` wahr ist, wird das Kästchen am Schnittpunkt der Zeile mit der Nummer `i` und der Spalte mit der Nummer `j` geschwärzt.

Hier ist ein Beispiel für eine Bedingung: `(i==2) or (j>3)`
Die Bedingung ist wahr, wenn die Zeilennummer 2 ist oder wenn die Spaltennummer größer als 3 ist.
Die Zeilennummern werden links und die Spaltennummern oberhalb der Tabelle notiert.
Daher müssen Sie die Kästchen wie unten beschrieben schwärzen.



Logische Tests verwenden die Vergleichsoperatoren aus den folgenden Beispielen.

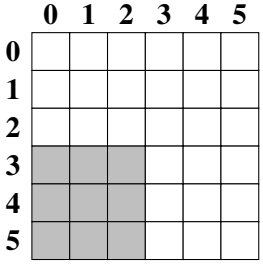
`i==1`	Wahr, wenn `i` gleich an 1 ist.
`j>3`	Wahr, wenn `j` größer als 3 ist.
`j>=4`	Wahr, wenn `j` größer oder gleich an 4 ist.
`i+j<5`	Wahr, wenn `i+j` kleiner als 5 ist.
`i<=j+2`	Wahr, wenn `i` kleiner oder gleich an `j+2` ist.
`(i==3) or (j==2)`	Wahr, wenn mindestens einer der Bedingungen wahr ist.
`(i<2) and (j>3)`	Wahr, wenn beide Bedingungen wahr sind.

Einige Fragen verwenden auch die Begriffe aus den unten stehenden Beispielen.

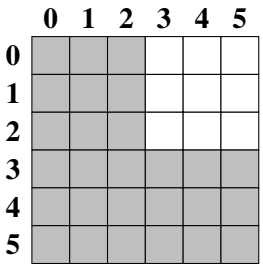
`max(14,18)`	Maximum-Funktion. Gibt den größten Wert zurück, also 18.
`min(i,j)`	Minimum-Funktion. Gibt den kleinsten Wert zurück.

Die Lösungen werden unten angezeigt.

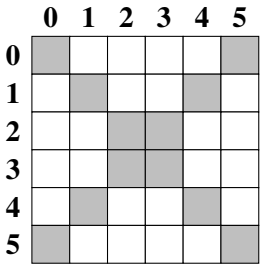
Q1(a) /2 **Schwärze die Kästchen, die die Bedingung `(i>=3) and (j<3)` erfüllen.**



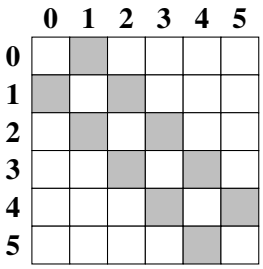
Q1(b) /2	**Schwärze die Kästchen, die die Bedingung `(i>=3) or (j<3)` erfüllen.**



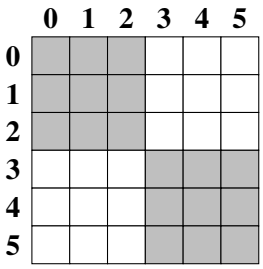
Q1(c) /4	**Schwärze die Kästchen, die die Bedingung `(i==j) or (i+j==5)` erfüllen.**



Q1(d) /4	**Schwärze die Kästchen, die die Bedingung `(i==j+1) or (i==j-1)` erfüllen.**



Q1(e) /4	**Schwärze die Kästchen, die die Bedingung `min(i,j)>=3 or max(i,j)<=2` erfüllen.**

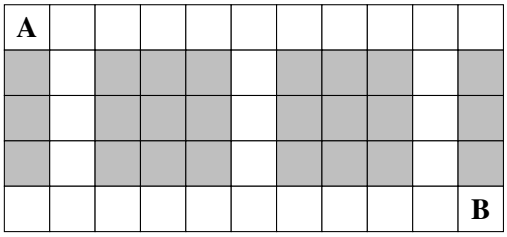


Frage 2 – Wege

Wie viele Wege?

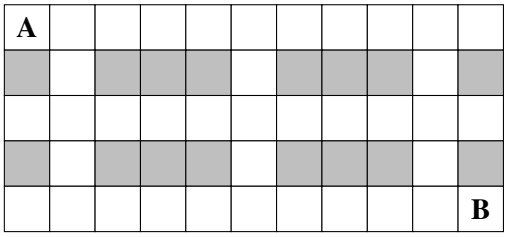
Die folgenden Abbildungen zeigen Räume mit quadratischen Fliesen (weiße Kästchen) und Hindernissen (graue Kästchen). Ein Roboter startet in **A** oben links und muss sich bis **B** unten rechts bewegen. Bei jedem Schritt kann er nur ein Feld nach unten oder nach rechts vorrücken. Er kann nicht nach links und nicht nach oben vorrücken. Der Roboter muss auf den weißen Feldern bleiben, er darf niemals über ein graues Feld fahren. Wie viele verschiedene Wege kann der Roboter in jedem Fall von **A** nach **B** nehmen?

Q2(a) /1 **Wie viele verschiedene Wege kann der Roboter von A nach B nehmen?**



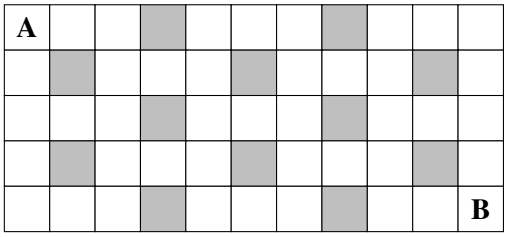
Lösung: 3

Q2(b) /1 **Wie viele verschiedene Wege kann der Roboter von A nach B nehmen?**



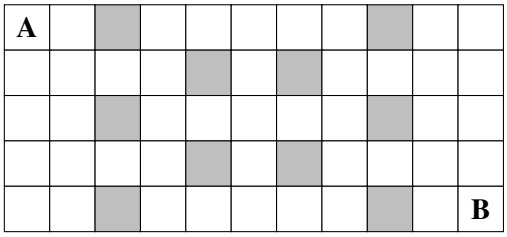
Lösung: 6

Q2(c) /2 **Wie viele verschiedene Wege kann der Roboter von A nach B nehmen?**



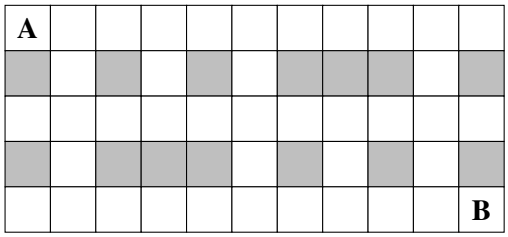
Lösung: 1

Q2(d) /2 **Wie viele verschiedene Wege kann der Roboter von A nach B nehmen?**



Lösung: 4

Q2(e) /3	**Wie viele verschiedene Wege kann der Roboter von A nach B nehmen?**

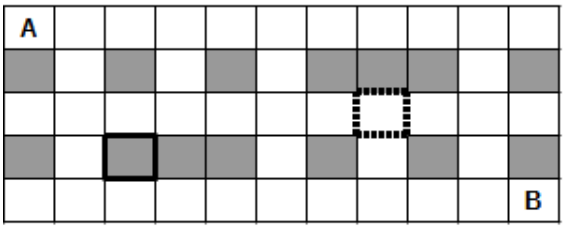


Lösung: 11

Ein Programm zum Zählen der Wege

Es gibt verschiedene Möglichkeiten, die Wege von **A** nach **B** zu zählen. Ein DP-Algorithmus (Dynamische Programmierung) ermöglicht dies in allen Fällen und wird in schwierigen Fällen unverzichtbar.

Das unvollständige Listing eines DP-Algorithmus wird weiter unten angegeben. Der Raum wird darin durch eine Tabelle `t[][]` repräsentiert, die **0** für jedes weiße Feld und **1** für jedes graue Feld enthält.



Raum

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

Dazugehörige Tabelle `t[][]`

Im obigen Beispiel hat die Tabelle `t[][]` 5 Zeilen und 11 Spalten. Die Zeilennummern werden auf der linken Seite der Tabelle notiert, die Spaltennummern oberhalb der Tabelle. Jedes Kästchen ist durch seine Zeilen- und Spaltennummer gekennzeichnet. Das dick umrandete Kästchen ist grau, das entsprechende Element `t[3][2]` ist also gleich **1**. Das gestrichelt umrandete Feld ist weiß, das entsprechende Element `t[2][7]` ist also gleich **0**. Die Felder **A** und **B** sind natürlich weiß und werden durch **0** in `t[][]` repräsentiert.

Der DP-Algorithmus berechnet für jedes Feld, wie viele verschiedene Wege **A** mit diesem Feld verbinden. Die Anzahl der Wege zum Feld **A** ist **1**, da man sich dort am Anfang befindet und es unmöglich ist, später dorthin zurückzukehren. Die Anzahl der Wege ist **0** für graue Felder (da man sie nie erreichen kann). Die Anzahl der Wege zu einem weißen Feld kann aus der Anzahl der Wege zu seinen linken und oberen Nachbarn berechnet werden (dies ist die wesentliche Eigenschaft, die es zu einem DP-Programm macht). Achten Sie auf die Felder in der ersten Spalte (die keinen linken Nachbarn haben) und in der ersten Zeile (die keinen oberen Nachbarn haben)!

Die Anzahl der Wege wird in einem Array `dp[][]` gespeichert, das die gleiche Anzahl an Zeilen und Spalten hat wie das Array `t[][]`. Zu Beginn des Algorithmus ist das Array `dp[][]` mit **0** gefüllt. Das Programm beginnt in Feld **A** und schreitet in Leserichtung fort (Zeile für Zeile, in jeder Zeile nach rechts). Im Beispielraum wird nach Ausführung des Algorithmus `dp[3][2]` gleich **0** sein, da kein Pfad ein graues Feld erreichen kann, und `dp[2][7]` wird die Anzahl der verschiedenen Wege enthalten, die von **A** zu dem gestrichelt umrandeten Feld führen.

Das Programm startet mit der Anzahl der Zeilen (es sind nicht immer 5!) in der Variablen `ni`, der Anzahl der Spalten in der Variablen `nj`, dem korrekt ausgefüllten Array `t[][]`, um den zu analysierenden Raum darzustellen, und dem mit **0**

gefüllten Array `dp[][]`.
Das Programm soll die Anzahl der verschiedenen Wege zurückgeben, die von **A** nach **B** führen.

Q2(f) /6	**Vervollständigen Sie die ____ im DP-Programm.** **Note zwischen 0 und 6. Für jeden Fehler oder jede fehlende Antwort verlieren Sie 1 Punkt.**
Lösung : Die Lösungen sind unten grau hinterlegt.	

```
dp[0][0] ← 1
for (i ← 0 to ni-1 step 1) {
   for (j ← 0 to nj-1 step 1) {

      if (t[i][j] = 1) { dp[i][j] ← 0 }
      else {
         if (i ≠ 0) { dp[i][j] ← dp[i-1][j] }

         if (j ≠ 0) { dp[i][j] ← dp[i][j] + dp[i][j-1] }
      }
   }
}
return dp[ni-1][nj-1]
```

Unerreichbare Felder

Einige weiße Felder sind nicht erreichbar. Das bedeutet, dass der Roboter sie von **A** aus wegen der Hindernisse (graue Kästchen) nicht erreichen kann. Wir gehen davon aus, dass das DP-Programm ausgeführt wurde, also haben wir die Anzahl der Zeilen in der Variablen `ni`, die Anzahl der Spalten in der Variablen `nj`, die Tabelle `t[][]`, die korrekt gefüllt ist, um den Raum darzustellen, und die Tabelle `dp[][]`, die durch das DP-Programm ausgefüllt wird.
Vervollständigen Sie das folgende Programm NoPath, das die Anzahl der unerreichbaren weißen Felder zurückgeben soll.

Q2(g) /5	**Ergänzen Sie die ____ im Programm NoPath, das die unzugänglichen weißen Felder zählt.** **Note zwischen 0 und 5. Für jeden Fehler oder jede fehlende Antwort verlieren Sie 1 Punkt.**
Lösung : Die Lösungen sind unten grau hinterlegt.	

```
NoPath ← 0
for (i ← 0 to ni-1 step 1) {
   for (j ← 0 to nj-1 step 1) {

      if (dp[i][j]=0 and t[i][j]=0) { NoPath ← NoPath + 1 }
   }
}
return NoPath
```

Zwischenfelder

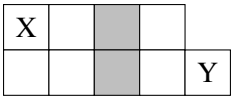
Wir wollen die Anzahl der Wege zählen, die über ein Zwischenfeld **X** führen. Der Roboter startet immer in **A** oben links und muss sich bis zu **B** unten rechts bewegen, aber er muss zwingend durch **X** gehen.

Raum 1: Wir wissen, dass es 12 verschiedene Wege zwischen **A** und **X** und 7 verschiedene Wege zwischen **X** und **B** gibt.

Q2(h) /1	**Wie viele Wege führen im Raum 1 von A über X nach B?**
Lösung : $12 \cdot 7 = 84$	

In den folgenden Fragen gibt es zwei Zwischenfelder **X** und **Y**.
Jedes Mal haben wir einen unvollständigen Plan des Raumes, auf dem die Felder **X** und **Y** eingezeichnet sind, und wir kennen
N(A,X) die Anzahl der verschiedenen Wege zwischen **A** und **X**, **N(X,B)** die Anzahl der verschiedenen Wege zwischen **X** und **B**,
N(A,Y) die Anzahl der unterschiedlichen Wege zwischen **A** und **Y**, **N(Y,B)** die Anzahl der unterschiedlichen Wege zwischen **Y** und **B**.

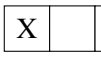
Raum 2: Wir wissen, dass N(A,X)=5, N(X,B)=12, N(A,Y)=8, N(Y,B)=10 und wir haben den Teilplan



.

Q2(i) /2	**Wie viele Wege führen in Raum 2 von A nach B über X oder Y?**
Lösung : $(5 \cdot 12) + (8 \cdot 10) = 140$	

Raum 3: Wir wissen, dass N(A,X)=6, N(X,B)=16, N(A,Y)=10, N(Y,B)=8 und wir haben die Teilplan



.

Q2(j) /1	**Wie viele Wege führen in Raum 3 von A nach B durch die beiden Felder X und Y?**
Lösung : $6 \cdot 8 = 48$	

Q2(k) /2	**Wie viele Wege führen in Raum 3 von A nach B über X oder Y (oder über beide)?**
Lösung : $(6 \cdot 16) + (10 \cdot 8) - (6 \cdot 8) = 128$	

Frage 3 – Loops

Betrachten wir das folgende Programm **LoopA**. Es besteht aus drei verschachtelten **`for`**-Schleifen und einem Test mit drei Bedingungen, die wahr sein müssen, um eine Anzeigeanweisung auszuführen.

```
for (i ← 1 to 100 step 1) {
   for (j ← 1 to 100 step 1) {
      for (k ← 1 to 100 step 1) {
         if (i<j and j<k and i+j+k=100) {
            print(i,j,k)
         }
      }
   }
}
```

LoopA zeigt alle Tripel (Gruppe mit 3 Zahlen) von ganzen Zahlen größer als null an, deren Summe 100 beträgt.
In jedem Tripel werden die drei Zahlen in aufsteigender Reihenfolge angezeigt.
So zeigt **LoopA** das Tripel `(20,30,50)` an, aber nicht das Tripel `(50,20,30)`.

Q3(a) /1	**Welches ist das erste angezeigte Tripel?**
Lösung : $(1, 2, 97)$	

Q3(b) /1	**Welches ist das zehnte angezeigte Tripel?**
Lösung : $(1, 11, 88)$	

Q3(c) /1	**Welches ist das letzte angezeigte Tripel?**
Lösung : $(32, 33, 35)$	

Q3(d) /1	**Wie oft wird der Test in LoopA ausgewertet?**
Lösung : $100 \cdot 100 \cdot 100 = 1000000$ Mal	

Das Programm **LoopB** zeigt genau dasselbe an wie **LoopA**.
Er verwendet jedoch einen einfacheren Test: Die Bedingungen `i<j` und `j<k` werden nicht mehr benötigt.

Q3(e) /3	**Ergänzen Sie die ____ in LoopB so, dass es das Gleiche wie LoopA anzeigt.** **Note zwischen 0 und 3. Für einen Fehler oder eine fehlende Antwort verlieren Sie 1 Punkt.**
Lösung : Die Lösungen sind unten grau hinterlegt.	

```
for (i ← 1 to 100 step 1) {
   for (j ← i+1 to 100 step 1) {
      for (k ← j+1 to 100 step 1) {
         if (i+j+k=100) {
            print(i,j,k)
         }
      }
   }
}
```

Q3(f) /1	**Wird der Test in LoopB öfters, seltener oder gleich oft ausgewertet wie der Test in LoopA?** **☐ Öfters ☒ Seltener ☐ Gleich oft** **Der Test wurde 161700 Mal in LoopB ausgewertet.**

Q3(g) /1	**Wird der Code in LoopB öfters, seltener oder gleich oft ausgeführt wie der in LoopA?** **☐ Öfters ☐ Seltener ☒ Gleich oft** **Beide Programme zeigen genau das Gleiche an!**

Das Programm **LoopC** zeigt genau das Gleiche wie **LoopA** und **LoopB** an.
Der Test hat sich wieder geändert und es gibt nur noch zwei **`for`**-Schleifen.

Q3(h) /3	**Ergänzen Sie die ____ in LoopC so, dass es das Gleiche wie LoopA und LoopB anzeigt.** **Note zwischen 0 und 3. Für einen Fehler oder eine fehlende Antwort verlieren Sie 1 Punkt.**
Lösung : Die Lösungen sind unten grau hinterlegt.	

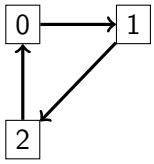
```
for (i ← 1 to 100 step 1) {
   for (j ← i+1 to 100 step 1) {
      k = 100-i-j
      if (j<k) {
         print(i,j,k)
      }
   }
}
```

Q3(i) /1	**Wird der Test in LoopC öfters, seltener oder gleich oft ausgewertet wie der Test in LoopB?** **☐ Öfters ☒ Seltener ☐ Gleich oft** **Der Test wurde 4950 Mal in LoopC ausgewertet.**

Frage 4 – Teleportationen

Professor Spock hat eine Reihe von Teleportationsnetzen entwickelt, mit dem Dinge und Menschen gebeamt werden können. Ein solches Netzwerk besteht aus n Teleportationskabinen, die von 0 bis $n-1$ durchnummeriert sind. In jeder Kabine kann man sich zu genau einer **anderen** Kabine teleportieren. (es handelt sich noch um Prototypen, jede Kabine ermöglicht derzeit nur den Zugang zu einem festgelegten Ziel). Um Sicherheitskontrollen durchzuführen, möchte Spock überprüfen, ob die wiederholte Verwendung von Transportern sicher ist. Können Sie ihm dabei helfen?

Das erste Netzwerk, das der Professor entwickelt hat, besteht aus nur drei Kabinen.



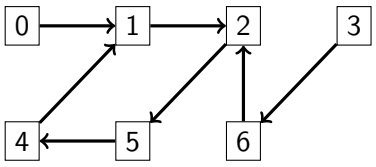
Der Professor testet sein Netzwerk, indem er einen Gegenstand, der sich zu Beginn in der Kabine mit der Nummer 0 befindet, mehrmals hintereinander teleportiert. Der Gegenstand kommt nach einer Teleportation in Kabine Nummer 1 an, nach zwei Teleportationen in Kabine Nummer 2, nach drei Teleportationen wieder in Kabine Nummer 0 und so weiter.

Q4(a) /1	**Ausgehend von der Kabine 0, in welcher Kabine kommt das Objekt nach 10 Teleportationen an?**
Lösung : 1	

Q4(b) /2	**Ausgehend von der Kabine 0, in welcher Kabine kommt das Objekt nach 50 Teleportationen an?**
Lösung : 2	

Q4(c) /2	**Ausgehend von der Kabine 0, in welcher Kabine kommt das Objekt nach 1000 Teleportationen an?**
Lösung : 1	

Da die Tests erfolgreich waren, wurde das **zweite Netz** mit sieben Kabinen eingerichtet.

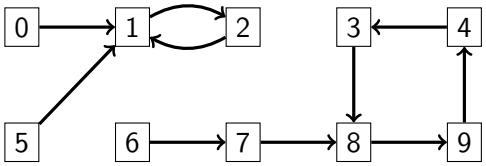


Q4(d) /2	**In welche Kabine gelangt man nach 50 Teleportationen, wenn man in der Kabine 4 beginnt?**
Lösung : 2	

Q4(e) /2	**In welche Kabine gelangt man nach 1000 Teleportationen, wenn man in der Kabine 3 beginnt?**
Lösung : 4	

Sie stellen fest, dass man bei Anwendung einer großen Anzahl von Teleportationen am Ende immer wieder in den selben Kabinen zurückkehrt. So werden im vorherigen Netz, ausgehend von der Kabine mit der Nummer 0, nacheinander die Kabinen 0, 1, 2, 5, 4, 1, 2, 5, 4, 1, 2, 5, 4,$\cdots$ besucht. Eine Kabine, zu der man nach einer bestimmten Anzahl von Teleportationen wieder zurückkehrt, wird als *zu einem Zyklus gehörend* bezeichnet. Ein *Zyklus* ist eine Liste von Kabinen, in der jede Kabine zur nächsten führt und die letzte Kabine wieder zur ersten. Die Länge eines Zyklus ist die Anzahl der Kabinen in ihm. Das **zweite Netz** hat einen Zyklus der Länge 4, der aus den Kabinen 1, 2, 5, 4 besteht.

Unten ist das **dritte Netz** mit 10 Kabinen und 2 Zyklen, einer mit der Länge 2, der andere mit der Länge 4, abgebildet.



Der Professor hat bewiesen, dass man in all seinen *Teleportationsnetzwerken*, in denen man sich von jeder Kabine zu genau einer anderen Kabine teleportieren kann, nach einer bestimmten Anzahl von Teleportationen immer in einen Zyklus gelangt, egal von welcher Kabine aus man startet.

Q4(f) /3	**Nach wie vielen Teleportationen ist man in einem Netzwerk mit `n` Kabinen von einer beliebigen Kabine aus sicher in einem Zyklus angekommen?**
Lösung : `n-2`	

Es ist hilfreich, computergestützt den Zyklus zu bestimmen, in den man von einer bestimmten Kabine aus gelangt.
Ein Teleportationsnetz kann durch eine Tabelle `T[]` dargestellt werden, die für jede Kabine angibt, zu welcher Kabine teleportiert wird.
So ist die Tabelle für das dritte Netzwerk aus 10 Umkleidekabinen `T=[1, 2, 1, 8, 3, 1, 7, 8, 9, 4]`.
Der Zyklus der Länge 2 wird durch die Beziehungen `T[1]=2` und `T[2]=1` übersetzt.
Die Zykluslänge 4 wird durch die Beziehungen `T[3]=8`, `T[8]=9`, `T[9]=4` und `T[4]=3` verdeutlicht.
Über die Kabinen 0, 1, 2 und 5 gelangt man zum Zyklus der Länge 2.
Mit den anderen Kabinen gelangt man zum Zyklus der Länge 4 (nach 2 Teleportationen, wenn man von der Kabine 6 aus startet).

Sie sollen das folgende **Programm A** ausfüllen. Als Eingaben wird ein Array `T[]` verwendet, das das zu untersuchende Netz darstellt, die Anzahl `n` der Kabinen und die Nummer `c` der Startkabine.
Die Ausgabe ist ein Array aus logischen (booleschen) Werten `cyc[]` mit der Länge `n`, so dass `cyc[i]` gleich an **`true`** ist, wenn die Kabinen mit der Nummer `i` zu dem Zyklus gehört, den man von der Kabine `c` aus erreicht, ansonsten **`false`**.
Das Programm verwendet intern ein Array `check[]` mit `n` logischen Werten.
Sie können den logischen Operator **`not`** verwenden (**`not true`** ist gleich **`false`** und **`not false`** ist gleich **`true`**).

Q4(g) /5	**Füllen Sie die ____ in Programm A aus.** **Note zwischen 0 und 5. Für jeden Fehler oder jede fehlende Antwort verlieren Sie 1 Punkt.**
Lösung : Die Lösungen sind unten grau hinterlegt.	

```
Input : n, T[], c       Output : cyc[]
for (i ← 0 to n-1 step 1) {
   check[i] ← false
   cyc[i] ← false
}
pos ← c
while (not check[pos]) {
    check[pos] ← true
    pos ← T[pos]
}
while (not cyc[pos]) {
    cyc[pos] ← true
```

```
    pos ← T[pos]
}
return cyc[]
```

Es ist auch wichtig zu wissen, welche Kabinen die Zyklen des Netzwerks bilden. Das **Programm B** füllt das Boolesche Array `cy[]` so aus, dass `cy[i]` gleich **`true`** ist, wenn die Kabine mit der Nummer `i` zu einem Zyklus gehört, und sonst gleich **`false`** ist. Das Programm verwendet intern ein Array `pos[]` mit `n` Ganzzahlen.

Q4(h) /4	**Füllen Sie die ____ in Programm B aus.** **Note zwischen 0 und 4. Sie verlieren 2 Punkte für jeden Fehler oder jede fehlende Antwort.**
Lösung : Die Lösungen sind unten grau hinterlegt.	

```
Input : n, T[]    Output : cy[]
for (i ← 0 to n-1 step 1) {
    pos[i] ← i
    cy[i] ← false
}
for(j ← 0 to n-1 step 1) {
    for (i ← 0 to n-1 step 1) {
        pos[i] ← T[pos[i]]
    }
}
for (i ← 0 to n-1 step 1) {
    cy[pos[i]] ← true
}
return cy[]
```

Auch die Längen der Zyklen sind wichtig. Das **Programm C** muss das Array `lency[]` so füllen, dass `lency[i]` gleich 0 ist, wenn die Kabine mit der Nummer `i` nicht zu einem Zyklus gehört, und ansonsten der Länge dieses Zyklus entspricht. Das **Programm C** verwendet die Tabelle `cy[]`, die zuvor von **Programm B** berechnet wurde.

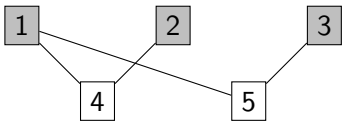
Q4(i) /5	**Füllen Sie die ____ in Programm C aus.** **Note zwischen 0 und 5. Für jeden Fehler oder jede fehlende Antwort verlieren Sie 1 Punkt.**
Lösung : Die Lösungen sind unten grau hinterlegt.	

```
Input : n, T[], cy[]    Output : lency[]
for (i ← 0 to n-1 step 1) {
    if(cy[i]) {
        pos ← T[i]
        len ← 1
        while(pos ≠ i) {
            pos ← T[pos]
            len ← len+1
        }
        lency[i] ← len
    }
   else { lency[i] ← 0 }
}
return lency[]
```

Frage 5 – Quidditch

Professor Dumbledore hat ein Problem: Er muss ein Quidditch-Spiel (den Lieblingssport der Zauberer) organisieren und die Schüler in zwei Gruppen einteilen, aus denen er Mannschaften zusammenstellen kann. Das Problem ist, dass sich viele Schüler nicht gut verstehen. Ron will zum Beispiel nicht mehr mit Draco reden, seit dieser schleimige Nacktschnecken in sein Bett gezaubert hat.

Professor Dumbledore hat beschlossen, die Feindschaften der Schüler in einem Graphen, wie dem unten abgebildeten, festzuhalten. Er besteht aus "Knoten" mit den Namen der Schüler (hier zur besseren Lesbarkeit durch Zahlen ersetzt) und aus Verbindungen (genannt "Kanten") zwischen diesen Knoten, die anzeigen, wenn zwei Schüler sich nicht verstehen.



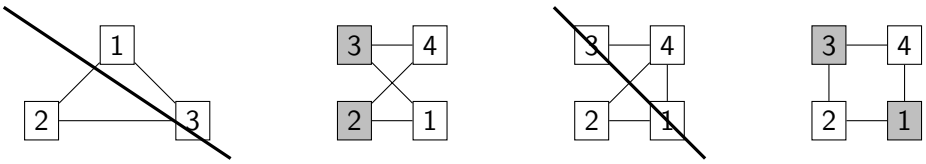
Erstes Beispiel: Ein Graph mit 5 Knoten und 4 Kanten.

So sieht man an diesem Beispiel, dass 1 und 4 sich nicht verstehen und daher nicht in derselben Gruppe sein können. Dagegen versteht sich 1 gut mit 2 und 3.

Dumbledore teilt die Schüler daher in zwei Gruppen ein, um zwei Teams zu bilden. Diese beiden Gruppen müssen nicht unbedingt gleich groß sein, aber **es können nicht zwei Schüler, die sich nicht verstehen, in derselben Gruppe sein**.

Im obigen Beispiel ist es leicht zu erkennen, dass wir zwei Gruppen haben: $\{1, 2, 3\}$ und $\{4, 5\}$. Um dies besser sichtbar zu machen, färbt Dumbledore die Schüler der ersten Gruppe grau und lässt die anderen weiß.

Hier sind weitere Graphen. Für jeden von ihnen wird gefragt, ob man ihre Knoten wie oben erläutert in zwei Gruppen aufteilen kann. Wenn das möglich ist, dann malen Sie die Knoten einer der beiden Gruppen aus. Wenn das nicht möglich ist, dann streichen Sie den Graphen durch.
Sie können unten im Entwurf arbeiten. Vergessen Sie nicht, Ihre Antwort **auf den Antwortbogen** zu kopieren.



Q5(a) /2	**Färben Sie die Knoten einer Gruppe im Graphen (a) ein oder streichen Sie den Graphen durch, wenn das nicht möglich ist.**
Lösung : Siehe oben.	

Q5(b) /2	**Färben Sie die Knoten einer Gruppe im Graphen (b) ein oder streichen Sie den Graphen durch, wenn das nicht möglich ist.**
Lösung : Siehe oben.	

Q5(c) /2	**Färben Sie die Knoten einer Gruppe im Graphen (c) ein oder streichen Sie den Graphen durch, wenn das nicht möglich ist.**
Lösung : Siehe oben.	

Q5(d) /2	**Färbe die Knoten einer Gruppe im Graphen ein (d) oder streiche den Graphen durch, wenn das nicht möglich ist.**
Lösung : Siehe oben.	

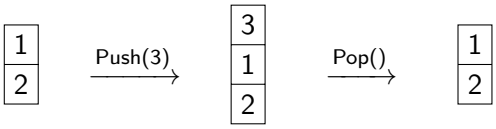
Professor Dumbledore möchte nun einen Algorithmus finden, der diese Einteilung in Gruppen vornimmt, mit anderen Worten, seine Graphen automatisch einfärbt.

Er schlägt in seinem Lieblingsbuch nach: dem “De Algorithmica”, das der Magier Cormenius (und seine Kollegen) im 13. Jahrhundert geschrieben hat. Leider wurde auf der Seite, die Dumbledore interessiert, unauslöschliche Zaubertinte verschüttet und Teile des Algorithmus fehlen.

Dumbledore versteht, dass er eine Datenstruktur namens “Stapel” braucht. Ein Stapel ermöglicht es, Knoten festzuhalten, wenn man durch einen Graphen läuft, wie bei einem Stapel Teller, wobei ein Knoten unten im Stapel liegt, ein anderer oben, *usw.*, bis man die Spitze des Stapels erreicht hat. Die folgenden Operationen können auf einem Stapel durchgeführt werden (siehe Beispiel weiter unten):

1. $push(x)$ fügt einen Knoten x ganz oben auf dem Stapel hinzu;
2. $pop()$ gibt den Namen der obersten Stelle des Stapels zurück und entfernt ihn vom Stapel;
3. $empty()$ gibt **`true`** zurück, wenn der Stapel leer ist, und gibt ansonsten **`false`** zurück.

Hier ist ein Beispiel, in dem wir von einem Stapel mit zwei Elementen ausgehen, 3 hinzufügen und dann wieder entfernen (die letzte Operation $pop()$ gibt also 3 zurück).



Q5(e) /1	**Angenommen, wir beginnen mit einem leeren Stapel. Geben Sie eine Sequenz von $push()$ an, mit der Sie den Stapel erhalten:** 
Lösung : Push(87), Push(11), Push(42)	

Q5(f) /1	**Angenommen, wir haben den Stapel**  **und der die folgende Abfolge von Operationen anwendet:** Pop(), Pop(), Push(24), Push(37), Push(71), Pop(), Pop()**. Was ist der resultierende Stapel?**

Lösung: 

Q5(g) /1	**Welchen Wert gibt das letzte $pop()$ der Sequenz aus der vorherigen Frage zurück?**
Lösung : Der letzte Aufruf von $pop()$ gibt 37 zurück.	

Professor Dumbledore kann nun einen ersten Algorithmus für *Graphenverlauf* untersuchen.
Der zu durchlaufende Graph besteht aus `n` Knoten und wird durch die Tabelle der logischen (booleschen) Werte `Edge[][]` beschrieben. `Edge[x][y]` und `Edge[y][x]` haben den Wert **`true`**, wenn sich zwischen den Knoten `x` und `y` eine Kante befindet, und haben ansonsten den Wert **`false`**.
Die Graphen aus dem ersten Beispiel `Edge[1][4]` und `Edge[4][1]`, `Edge[1][5]` und `Edge[5][1]`, `Edge[2][4]` und `Edge[4][2]`, `Edge[3][5]` und `Edge[5][3]` haben den Wert **`true`**. Alle anderen Werte im Array `Edge[][]` für diesen Graphen sind gleich **`false`**.

Der Algorithmus erhält die Anzahl der Knoten `n`, das Array `Edge[][]` und einen Anfangsknoten `s`, an dem der Durchlauf beginnt.
Er verwendet eine Tabelle mit `color[]`, mit der die Knoten im Laufe der Zeit eingefärbt werden.
Zu Beginn des Programms setzen wir `color[i]` für alle Knoten `i` auf -1.
Um einen Knoten mit `i` einzufärben, setzen wir `color[i]` auf 0 und dieser Wert ändert sich danach nicht mehr.
Der Algorithmus verwendet auch einen Stapel, der anfangs leer ist, um Knoten festzuhalten, die auf eine Bearbeitung warten.
Hinweis: Anders als in der Informatik üblich, beginnen die Indizes in diesem Programm bei 1.

```
Input : n, Edge[][], s

for (i ← 1 to n step 1){
   color[i] ← -1
}

Push(s)
color[s] ← 0

while(not IsEmpty()) {
   x ← Pop()
   for (y ← 1 to n step 1){
      if (Edge[x][y] and color[y] = -1){
         Push(y)
         color[y] ← 0
      }
   }
}
```

Q5(h) /3	**In welcher Reihenfolge färbt der Algorithmus die Knoten des Graphen im ersten Beispiel (mit 5 Knoten und 4 Kanten) ein, wenn der Anfangsknoten `s=4` ist?**
Lösung : 4, 1, 2, 5, 3	

Professor Dumbledore versucht schließlich, den Algorithmus zu finden, mit dem man die Knoten des Graphen in zwei Gruppen aufteilen kann.
Er versteht, dass er zwei verschiedene Farben verwenden muss, die 0 und 1 sein werden, die den beiden Gruppen entsprechen.
Er versteht, dass, wenn ein Knoten die Farbe 0 hat, alle Knoten, die durch eine Kante mit ihm verbunden sind, die Farbe 1 haben müssen und umgekehrt.
Zur Hilfe in dieser Aufgabe, nutzt Professor Dumbledore eine Funktion `invert` wie `invert(0)=1` und `invert(1)=0`.
Können Sie mit diesen Ideen den unten stehenden Algorithmus vervollständigen?
Er soll ein Array `color[]` mit den Farben jedes Knotens zurückgeben, wenn eine Trennung möglich ist, und **`false`** zurückgeben, wenn eine Trennung nicht möglich ist.

Q5(i) /6	**Ergänzen Sie die ____ im Algorithmus.** **Note zwischen 0 und 6. Sie verlieren 2 Punkte für jeden Fehler oder jede fehlende Antwort.**
Lösung : Die Lösungen sind unten grau hinterlegt.	

```
Input : n, Edge[][], s          Output: color[] or false
for (i ← 1 to n step 1){
   color[i] ← -1
}
Push(s)
color[s] ← 0

while(not IsEmpty()) {
   x ← Pop()
   for (y ← 1 to n step 1){
       if (Edge[x][y]){
          if (color[y] = color[x] ) {
             return false
          } else if (color[y] = -1) {
             Push(y)
             color[y] = invert(color[x])
          }
       }
   }
}
return color[]
```