

be-OI 2020

**Finale -
SENIOREN**
Samstag 7. März 2020

**Füllt diesen Rahmen bitte in GROSSBUCHSTABEN und
LESERLICH aus**

VORNAME :
NAME :
SCHULE :

O**Reserviert**

Belgische Informatik-Olympiade 2020 (Dauer : maximal 2 Stunden)

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

- 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 **Junioren**.
 - Für Schüler der Sekundarstufe 5 und höher: Kategorie **Senioren**.
- Gebe deinen Namen, Vornamen und deine Schule **nur auf der erste Seite** an.
- Gebe **deine Antworten** auf den in diesem Dokument **am Ende des Formulars** dafür vorgesehenen Seiten an.
- Wenn du dich bei einer Antwort vertan hast, dann beantworte sie unbedingt **auf dem selbem Blatt** (ggf. auf der Rückseite).
- Schreibe **gut lesbar** mit einem **blauen oder schwarzen Stift oder Kugelschreiber**.
- Du kannst nur Schreibmaterial dabei haben; Taschenrechner, Mobiltelefone, ... sind **verboten**.
- Du kannst jederzeit weitere Kladdeblätter bei einer Aufsichtsperson fragen.
- 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.
- Alle Codeauszüge aus der Anweisung sind in **Pseudo-Code**. Auf den folgenden Seiten findest du eine **Beschreibung** des Pseudo-Code, den wir verwenden.
- Wenn du mit einem Code antworten musst, dann benutze den **Pseudo-Code** oder eine **Sprache 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 Mitglieder:



©2020 Belgische Informatik-Olympiade (beOI) ASBL

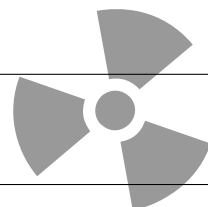
Dieses Werk wird unter den Bedingungen der Creative Commons Attribution 2.0 Belgium License zur Verfügung gestellt.

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/richtig (**true**) oder falsch/fehlerhaft (**false**). Es ist möglich, arithmetische Operationen auf Variablen durchzuführen. Zusätzlich zu den vier traditionellen Operatoren (+, −, × 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 *alter* den Wert 17 erhält.

```
geburtsjahr ← 2003
alter ← 2020 − 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*. 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* 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 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 laufen, kann man ein Schleifencodefor 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 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 ( $i \leftarrow 0$  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 ( $N \geq 10$ )
{
     $N \leftarrow N/d$ 
     $d \leftarrow d + 1$ 
}
```

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 einer Tabelle.

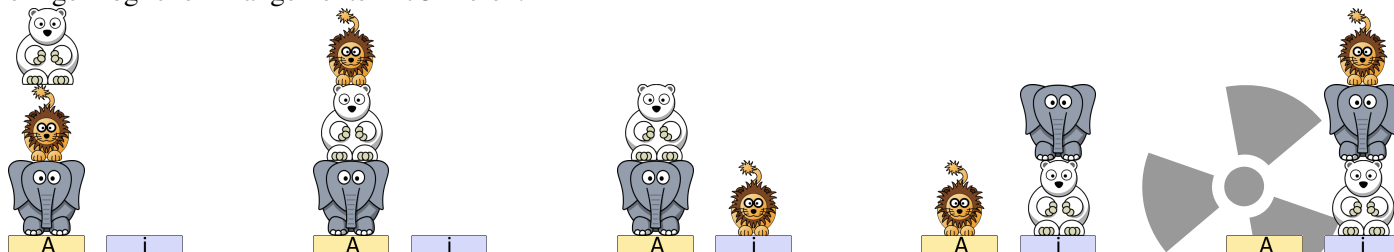
```
Eingabe: tab, ein Tabelle mit N Zahlen.
         N, die Anzahl der Elemente der Tabelle.
Ausgabe: sum, die Summe aller in der Tabelle enthaltenen Zahlen.

summe ← 0
for ( $i \leftarrow 0$  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 – Zirkus

Der Dompteur Zarbi hat mit seinen Tierrobotern Zirkusnummern entwickelt. Er verwendet zwei Podeste **A** und **i** (die wir **A** und **i** notieren), auf denen die Tiere ohne Höhenbegrenzung übereinander stehen können. Die Tiere können beliebig verteilt werden, ein Stand kann leer sein und alle Tiere können übereinander gestapelt werden. Hier sind einige mögliche Arrangements mit 3 Tieren:



Q1(a) [2 Pkte]	Auf wie viele verschiedene Arten können 2 Tiere auf den Sockeln platziert werden?
Lösung: 6	
Q1(b) [4 Pkte]	Auf wie viele verschiedene Arten können 3 Tiere auf den Sockeln platziert werden?
Lösung: 24	
Q1(c) [4 Pkte]	Auf wie viele verschiedene Arten können n Tiere auf die Podeste gestellt werden?
Lösung: $(n + 1)!$	

Während einer Show gibt der Dompteur Zarbi Anweisungen, die die Tiere dazu bringen, die Plätze zu wechseln.

Anweisungen zum “Klettern”.

MA: Bei dieser Anweisung klettert das unterste Tier auf dem Podest **A** auf die Spitze seines Stapels.

Mi: Bei dieser Anweisung klettert das unterste Tier auf dem Podest **i** auf die Spitze seines Stapels.

(Es passiert nichts, wenn sich weniger als 2 Tiere auf dem betreffenden Podest befinden.)

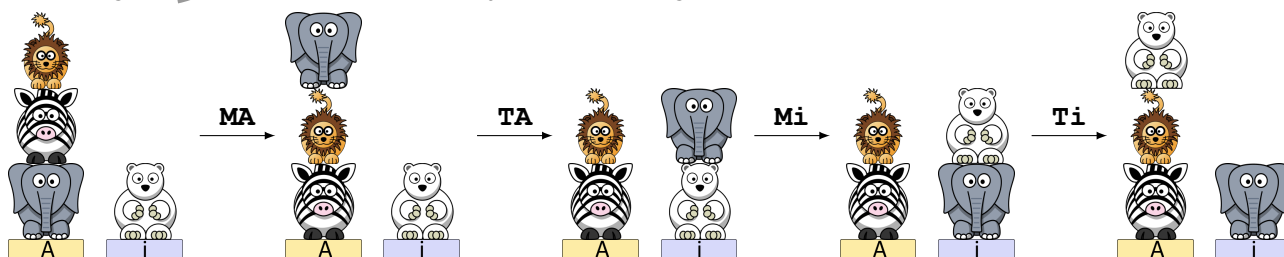
Anweisungen zum “Springen”.

TA: Bei dieser Anweisung springt das höchste Tier vom Podest **A** auf die Spitze des Stapels auf dem Podest **i**.

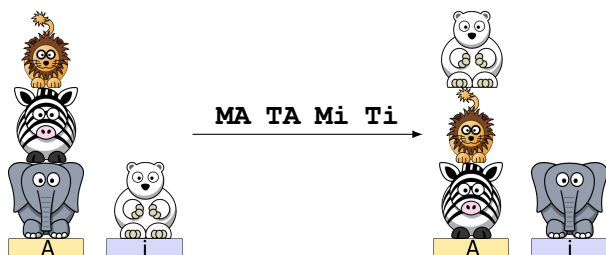
Ti: Bei dieser Anweisung springt das höchste Tier vom Podest **i** auf die Spitze des Stapels auf dem Podest **A**.

(Es passiert nichts, wenn sich keine Tiere auf dem betreffenden Podest befinden.)

Beispiel: 4 Tiere werden wie auf dem Bild links platziert und der Dompteur gibt die Anweisungen **MA TA Mi Ti**. Die Bilder zeigen, wie sich die Situation nach jeder Anweisung entwickelt.

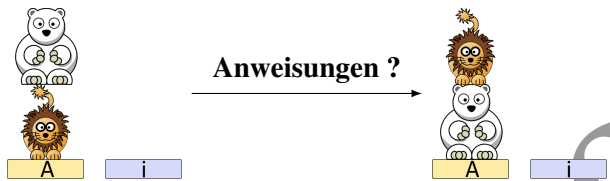

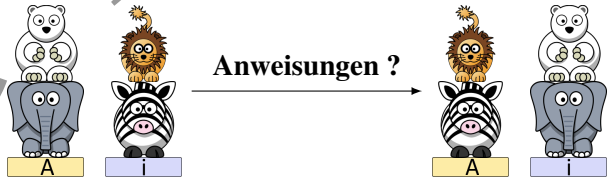


Man kann dies zusammenfassen, indem man einen einzigen Pfeil mit allen Anweisungen darauf zeichnet:



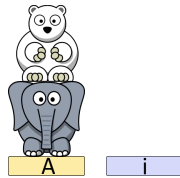
Die Länge einer Anweisungsliste ist die Anzahl der Anweisungen in der Liste. Zum Beispiel hat die Anweisungsliste **MA TA Mi Ti** die Länge 4.

In den folgenden Fragen müsst ihr eine Liste von Anweisungen aus {**MA**, **Mi**, **TA**, **Ti**} finden, um von der linken zur rechten Situation zu kommen. **Eure Liste sollte so kurz wie möglich sein.** Wenn eure Liste nicht die kürzeste Länge hat, erhaltet ihr nur die Hälfte der Punkte für die Frage.

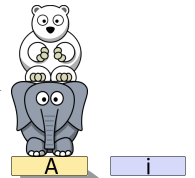
Q1(d) [2 Pkte]	
Lösung: MA	
Q1(e) [2 Pkte]	
Lösung: TA MA Ti MA	
Q1(f) [2 Pkte]	
Lösung: zum Beispiel MA TA TA Mi Ti Mi Ti	

Q1(g) [4 Pkte]

Zwei Tiere sollten so schnell wie möglich alle Positionen durchlaufen. Ausgehend von der links dargestellten Position müssen alle Tiere alle anderen Positionen einmal durchlaufen, bevor sie zur Ausgangsposition zurückkehren.



Anweisungen, die alle Positionen durchlaufen?



Lösung: **MA TA TA Mi Ti Ti ou TA TA Mi Ti Ti MA**

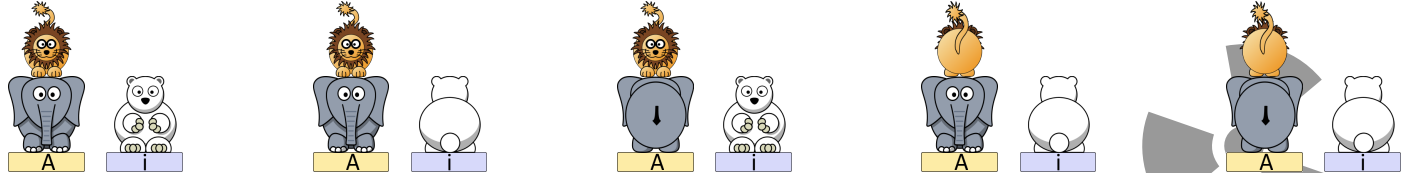
Fügen wir neue Anweisungen für das “Drehen” hinzu:

RA: Bei dieser Anweisung dreht sich das höchste Tier auf dem Podest um 180° (es macht eine halbe Drehung).

Ri: Bei dieser Anweisung dreht sich das höchste Tier auf dem Podest um 180° (es macht eine halbe Drehung).

(Es passiert nichts, wenn sich keine Tiere auf dem betreffenden Podest befinden.)

Jedes Tier kann nun von vorne oder von hinten gesehen werden und die folgenden Positionen werden als unterschiedlich betrachtet:

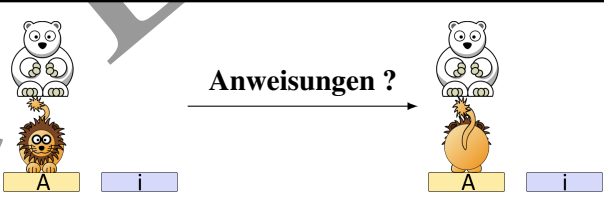


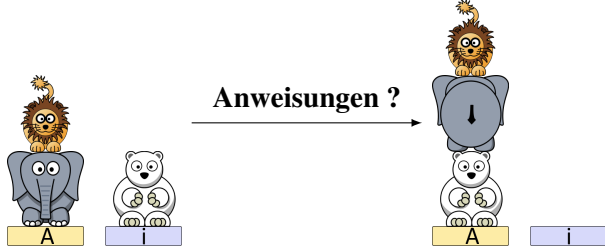
Q1(h) [2 Pkte]	Wenn man bedenkt, dass jedes Tier von vorne oder von hinten gesehen werden kann, wie viele verschiedene Möglichkeiten gibt es, 2 Tiere auf die Podeste zu platzieren?
Lösung: $6 \cdot 2^2 = 24$	
Q1(i) [4 Pkte]	Wenn man bedenkt, dass jedes Tier von vorne oder von hinten gesehen werden kann, wie viele verschiedene Möglichkeiten gibt es, 3 Tiere auf die Podeste zu platzieren?
Lösung: $24 \cdot 2^3 = 192$	
Q1(j) [4 Pkte]	Wenn man bedenkt, dass jedes Tier von vorne oder von hinten gesehen werden kann, wie viele verschiedene Möglichkeiten gibt es, n Tiere auf die Podeste zu stellen?
Lösung: $(n + 1)! \cdot 2^n$	

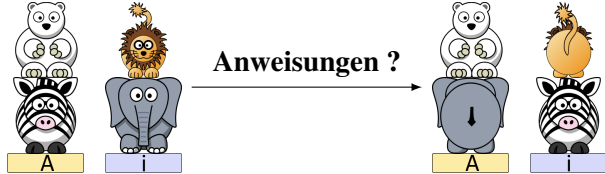
Wenn ein Tier “klettert” oder “springt”, ändert es weder von “vorne noch von hinten gesehen” seine Orientierung.

In den folgenden Fragen müsst ihr eine Liste von Anweisungen aus **{MA, Mi, TA, Ti, RA, Ri}** finden, um von der linken zur rechten Situation zu kommen.

Eure Liste sollte so kurz wie möglich sein. Wenn eure Liste nicht die kürzeste Länge hat, erhaltet ihr nur die Hälfte der Punkte für die Frage.

Q1(k) [2 Pkte]	
Lösung: TA RA Ti ou MA RA MA	

Q1(l) [2 Pkte]	
Lösung: Ti MA RA MA	

Q1(m) [2 Pkte]	
Lösung: Zum Beispiel: Ri Mi Ri Ti MA TA MA Mi	

Frage 2 – Türe öffnen

Eine Tür wird durch einen vierstelligen Geheimcode gesichert.

Die Tür öffnet sich sofort, sobald 4 nacheinander gedrückte Ziffern mit dem Geheimcode übereinstimmen.

Wenn ihr zum Beispiel 6,4,5,5,0,8,6,7 nacheinander drückt, dann öffnet sich die Tür, wenn der Geheimcode 6455, 4550, 5508, 5086 oder 0867 lautet.

Wir können einen Roboter programmieren. Der Befehl `press(a)` veranlasst den Roboter, die Taste zu drücken (der Kode `a` ist natürlich eine der Ziffern von 0 bis 9).

Programm 1 :

```

for (a ← 0 to 9 step 1) {
  for (b ← 0 to 9 step 1) {
    for (c ← 0 to 9 step 1) {
      for (d ← 0 to 9 step 1) {
        press(a)
        press(b)
        press(c)
        press(d)
      }
    }
  }
}

```

In den folgenden Fragen bitten wir euch, nur die während der Durchführung von Programm 1 gedrückten Tasten zu berücksichtigen.

Q2(a) [1 Pkt]	Wie viele Tasten werden bei der Ausführung von Programm 1 insgesamt gedrückt?
Lösung: $4 \cdot 10^4 = 40000$	
Q2(b) [1 Pkt]	Wie oft wird die Taste <input type="text" value="5"/> gedrückt?
Lösung: 4000	
Q2(c) [2 Pkte]	Wie viele 4-stellige Kombinationen werden von Programm 1 getestet? Wenn eine Kombination mehr als einmal getestet wird, müssen alle Versuche gezählt werden.
Lösung: $40000 - 3 = 39997$	
Q2(d) [1 Pkt]	Wie viele verschiedene Kombinationen von 4 Ziffern werden von Programm 1 getestet? Wenn eine Kombination mehr als einmal getestet wird, wird nur ein Versuch gezählt.
Lösung: 10000	
Q2(e) [1 Pkt]	Wird sich die Tür beim Ausführen von Programm 1 unabhängig vom Geheimcode sicher öffnen?
Lösung: JA	

Nummerieren wir die durch das Programm generierten Tastendrucke ab **0** (Macht hier keinen Fehler, wir fangen bei **Null** an zu zählen). Die ersten 20 Tasten (nummeriert von 0 bis 19) sind 0,0,0,0, 0,0,0,1, 0,0,0,2, 0,0,0,3, 0,0,0,4. Beachtet,

dass die Tasten, deren Nummer ein Vielfaches von 4 ist, immer der Anweisung `press(a)` im Programm entsprechen, und nicht der Anweisung `press(b)`, noch `press(c)`, noch `press(d)`.

Q2(f) [1 Pkt]	Welche Taste drückt der Roboter, wenn er die Taste mit der Nummer 1420 drückt?
Lösung: <input type="text" value="0"/>	
Q2(g) [1 Pkt]	Welche Taste drückt der Roboter, wenn er die Taste mit der Nummer 1422 drückt?
Lösung: <input type="text" value="5"/>	
Q2(h) [1 Pkt]	Gebt die 8 Tasten ein, die der Roboter nacheinander drückt, beginnend mit der Taste mit der Nummer 1548.
Lösung: 0387 0388	

Eine **“gute Folge”** ist eine Folge von 8 Tasten, die der Roboter während der Ausführung von Programm 1 nacheinander drückt **beginnend mit einer Taste deren Nummer ein Vielfaches von 4 ist**, d.h. die mit einer Taste beginnt, die während einer `press(a)` Anweisung im Programm gedrückt wird. Beispiele für “gute Folgen” sind: 0000 0001, 2307 2308, 6499 6500 (zur besseren Lesbarkeit wurde in der Mitte ein Leerzeichen eingefügt).

Beachtet, dass eine gute Folge immer aus 2 aufeinanderfolgenden 4-stelligen Zahlen besteht.

Wir sagen, dass eine gute Folge **eine Kombination testet** wenn diese Kombination in der guten Folge vorkommt.

Zum Beispiel testet die gute Folge **2307 2308** die Kombinationen 2307, 3072, 0723, 7230 und 2308.

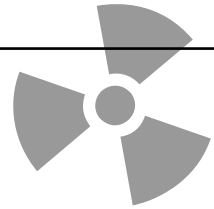
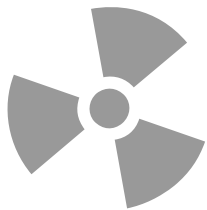
Oder noch ein Beispiel: die gute Folge **7171 7172** testet die Kombinationen 7171 et 1717, nochmal 7171 und 1717, und schließlich 7172.

Beachtet, dass die beiden Tests für 7171 nicht gleichzeitig stattfinden, sondern dass es sich um zwei verschiedene Tests der gleichen Kombination handelt (das Gleiche gilt für 1717).

Q2(i) [5 Pkte]	Findet die 5 guten Folgen, die die Kombination 7043 testen. Hilfe: man kann die Folge auf verschiedene Arten “schneiden” (17043, 71043, ...).
Lösung: 0437 0438, 3704 3705, 4370 4371, 7042 7043 et 7043 7044	
Q2(j) [1 Pkt]	Wie oft drückt der Roboter nacheinander die Tasten 7,0,4,3? Mit anderen Worten: Wie oft wird die Kombination 7043 getestet?
Lösung: 4 (die guten Folgen 7042 7043 und 7043 7044 führen den gleichen Test durch)	
Q2(k) [3 Pkte]	Findet 3 gute Folgen, die die Kombination 8383 testen.
Lösung: 3838 3839, 8382 8383 und 8383 8384	
Q2(l) [1 Pkt]	Wie oft drückt der Roboter nacheinander die Tasten 8,3,8,3? Mit anderen Worten: Wie oft wird die Kombination 8383 getestet?
Lösung: 4 (Achtung! Zählt nicht mehrmals den gleichen Test.)	
Q2(m) [4 Pkte]	Findet alle guten Folgen, die die Kombination 9900 testen.
Lösung: 0990 0991, 8999 9000, 9899 9900 und 9900 9901.	

Q2(n) [1 Pkt]	Wie oft drückt der Roboter nacheinander die Tasten 9,9,0,0? Mit anderen Worten: Wie oft wird die Kombination 9900 getestet?
Lösung: 3	

Q2(o) [1 Pkt]	Wie oft drückt der Roboter nacheinander die Tasten 9,5,6? Hilfe: Wenn X eine Taste ist, dann sind zum Beispiel 956X und 56X9 Geheimcode.
Lösung: $4 \cdot 10 = 40$	

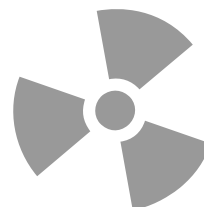


Betrachten wir nun ein nicht-deterministisches Programm, d.h. einige Tastendrucke werden zufällig generiert. In diesem Programm wählt die Funktion `random()` zufällig eine Zahl von 0 bis 9 aus.

Seid vorsichtig bei der zweiten Schleife: Es handelt sich nicht um eine Schleife von 0 bis 9, sondern um eine Schleife von 0 bis `a`.

Programm 2 :

```
for (a ← 0 to 9 step 1) {
  for (d ← 0 to a step 1) {
    press(a)
    press(random())
    press(random())
    press(d)
  }
}
```



In den folgenden Fragen bitten wir euch, nur die während der Durchführung von Programm 2 gedrückten Tasten zu berücksichtigen.

Q2(p) [3 Pkte]	Wie viele 4-stellige Kombinationen werden von Programm 2 getestet? Einige Kombinationen können mehr als einmal getestet werden, zählt dann alle Versuche.
Lösung: $(1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10) \cdot 4 - 3 = 217$	

Q2(q) [2 Pkte]	Angenommen, der Geheimcode lautet 7431. Ist es möglich, dass sich die Tür öffnet, wenn Programm 2 ausgeführt wird?
Lösung: JA	

Q2(r) [2 Pkte]	Angenommen, der Geheimcode lautet 4444. Ist es möglich, dass sich die Tür öffnet, wenn Programm 2 ausgeführt wird?
Lösung: JA	

Q2(s) [2 Pkte]	Angenommen, der Geheimcode lautet 2345. Ist es möglich, dass sich die Tür öffnet, wenn Programm 2 ausgeführt wird?
Lösung: JA	

*Erklärung: z.B. durch Schreiben der Tasten ab dem Zeitpunkt, wenn `a` in der ersten Schleife 3 und `d` in der zweiten Schleife 2 ist, wird das Programm 3**2 3**3 erzeugen, wobei die Sterne (*) Zufallszahlen sind und es möglich ist, dass die letzten 2 Sterne (*) 4 und 5 sind.*

Q2(t) [2 Pkte]	Angenommen, der Geheimcode lautet 4297. Ist es möglich, dass sich die Tür öffnet, wenn Programm 2 ausgeführt wird?
Lösung: JA	

*Erklärung: z.B. durch Schreiben der Tasten ab dem Zeitpunkt, wenn `a` in der ersten Schleife 9 und `d` in der zweiten Schleife 2 ist, wird das Programm 9**2 9**3 erzeugen, wobei die Sterne (*) Zufallszahlen sind und es möglich ist, dass der zweite Stern (*) 4 und der dritte Stern 7 ist.*

Es ist möglich, ein optimales Programm zu schreiben, das eine Folge von Tasten erzeugt, in der alle 4-stelligen Kombinationen jeweils genau einmal getestet werden. In den folgenden Fragen bitten wir euch, nur die während der Durchführung von diesem optimalen Programm gedrückten Tasten zu berücksichtigen.

Q2(u) [4 Pkte]	Wie viele Tasten werden bei der Durchführung eines optimalen Programms insgesamt gedrückt?
Lösung: $10^4 + 3 = 10003$.	

Erklärung: Es gibt 10000 Kombinationen zu testen, jeder Tastendruck muss eine neue Kombination starten, vergisst die letzten 3 Ziffern der letzten Kombination nicht.

Es gibt mehrere Möglichkeiten, ein optimales Programm zu schreiben, aber in jedem Fall sind die ersten 3 Ziffern der erzeugten Sequenz die gleichen wie die letzten 3.

Wenn zum Beispiel eine optimale Sequenz mit 123 beginnt, dann wird sie auch mit 123 enden.

Q2(v) [2 Pkte]	Wenn ein Programm Optimal123 ausgeführt wird, das einen mit 123 beginnenden Sequenz erzeugt, wie oft drückt der Roboter dann die Tasten 9,5,6?
Lösung: 10	

Q2(w) [2 Pkte]	Wenn ein Programm Optimal123 ausgeführt wird, das einen mit 123 beginnenden Sequenz erzeugt, wie oft drückt der Roboter dann die Tasten 1,2,3?
Lösung: 11	

Frage 3 – Versteckspiel

Alice spielt mit Bob Verstecken. Sie spielen in einem Haus mit $n \geq 2$ Räumen, die von 0 bis $n-1$ nummeriert sind. Alice wird sich verstecken, und Bob wird versuchen, Alice zu finden. Bob ist sehr vorhersehbar, daher weiß Alice im Voraus, welche Räume Bob durchsuchen wird und in welcher Reihenfolge. Bob wird $m \geq 1$ durchsuchen und möglicherweise einige Räume mehrmals durchsuchen. Wenn Bob einen Raum durchsucht, während Alice dort ist, gewinnt Bob. Alice will sicherstellen, dass Bob nicht gewinnt, und dazu kann sie den Raum wählen, in dem sie sich zunächst versteckt, und sie kann sich zwischen Bobs Durchsuchungen auch von Raum zu Raum bewegen. Aber jeder Zug ist riskant (sie könnte entdeckt werden). Deshalb möchte Alice die Anzahl der Züge, die sie machen muss, minimieren.

Stellen wir uns zum Beispiel vor, dass das Haus $n = 3$ Räume mit den Nummern 0, 1 und 2 hat, und dass Bob die folgende Sequenz von $m = 4$ Durchsuchungen durchführt:

$$\text{bob}[] = \{1, 2, 0, 2\}.$$

Eine mögliche Lösung für Alice besteht darin, sich zunächst im Raum 2 zu verstecken, dann zum Raum 0 zu gehen, bevor Bob den Raum 2 sucht, und schließlich zum Raum 1 zu gehen, bevor Bob den Raum 0 durchsucht. Alice würde sich 2 mal bewegen und ihre Position in der Zeit würde durch die Sequenz beschrieben werden:

$$\text{alice}[] = \{2, 0, 1, 1\}.$$

Aber Alice kann es besser machen: Nachdem sie den Raum 2 verlassen hat, könnte sie direkt in den Raum 1 ziehen und dort bis zum Ende des Spiels bleiben. Sie würde einmal umziehen und ihre Position in der Zeit wäre:

$$\text{alice}[] = \{2, 1, 1, 1\}.$$

Andererseits gibt es keine Möglichkeit, weniger als eine Bewegung zu machen. Alice kann nicht das ganze Spiel über im selben Raum bleiben, weil Bob alle Räume mindestens einmal durchsucht. Für $n = 3$ und $\text{bob}[] = \{1, 2, 0, 2\}$ beträgt Alices Mindestanzahl an Zügen also 1.

Q3(a) [2 Pkte]	Wie viele Züge muss Alice mindestens machen, wenn $n = 2$ und Bob durch die Räume $\text{bob}[] = \{0, 1, 0, 1\}$ geht?
Lösung: 3 ($\text{alice}[] = \{1, 0, 1, 0\}$)	
Q3(b) [2 Pkte]	Wie viele Züge muss Alice mindestens machen, wenn $n = 3$ und Bob durch die Räume $\text{bob}[] = \{0, 1, 0, 1\}$ geht?
Lösung: 0 ($\text{alice}[] = \{2, 2, 2, 2\}$)	
Q3(c) [2 Pkte]	Wie viele Züge muss Alice mindestens machen, wenn $n = 3$ und Bob durch die Räume $\text{bob}[] = \{2, 1, 2, 0\}$ geht?
Lösung: 1 (Zum Beispiel, $\text{alice}[] = \{0, 0, 1, 1\}$)	
Q3(d) [2 Pkte]	Wie viele Züge muss Alice mindestens machen, wenn $n = 3$ und Bob durch die Räume $\text{bob}[] = \{0, 1, 2, 0, 2, 0\}$ geht?
Lösung: 1 ($\text{alice}[] = \{2, 2, 1, 1, 1, 1\}$)	
Q3(e) [2 Pkte]	Wie viele Züge muss Alice mindestens machen, wenn $n = 4$ und Bob durch die Räume $\text{bob}[] = \{0, 1, 2, 3, 0\}$ geht?
Lösung: 1 (Zum Beispiel, $\text{alice}[] = \{3, 3, 3, 2, 2\}$)	

Q3(f) [2 Pkte]	Wie viele Züge muss Alice mindestens machen, wenn $n = 4$ und Bob durch die Räume $\text{bob}[] = \{0, 1, 2, 3, 0, 1\}$ geht?
Lösung: 1 ($\text{alice}[] = \{3, 3, 3, 2, 2, 2\}$)	
Q3(g) [3 Pkte]	Wie viele Züge muss Alice mindestens machen, wenn $n = 4$ und Bob durch die Räume $\text{bob}[] = \{0, 1, 2, 3, 0, 1, 2\}$ geht?
Lösung: 2 (Zum Beispiel, $\text{alice}[] = \{3, 3, 3, 2, 2, 3, 3\}$)	
Q3(h) [4 Pkte]	Was ist die kleinste Anzahl Durchsuchungen in einem Haus mit n Zimmern, die Bob durchführen muss, um Alice zu zwingen, mindestens k-mal umzuziehen? Diese kleinste Anzahl nennen wir m.
Lösung: $k(n - 1) + 1$	

Mit Kode wird alles noch lustiger

Alice beschließt, ein Programm zu schreiben, das für sie bestimmt, in welchem Raum sie sich während jeder von Bobs m Durchsuchungen verstecken soll.

Die Programmparameter sind:

- n : die Anzahl der Räume.
- m : die Anzahl der Durchsuchungen, die Bob machen wird.
- $\text{bob}[]$ (eine Tabelle mit der Länge m) : die Sequenz der Durchsuchungen, die Bob durchführen wird.

Das Programm muss $\text{alice}[]$ (eine Tabelle mit der Länge m) berechnen: eine Liste von m Räumen, in denen Alice sein muss, um nicht von Bob entdeckt zu werden, während sie sich so wenig wie möglich bewegt.

Es kann mehrere Lösungen geben, die die Mindestanzahl an Zügen ergeben; wenn dies der Fall ist, dann ist jede davon akzeptabel und Alices Programm gibt nur eine davon.

Das Programm befindet sich auf der nächsten Seite.

Hier sind einige Erklärungen, die euch das Verständnis erleichtern sollen.

- cnt : wird verwendet, um die Anzahl der verschiedenen Räume zu zählen, die Bob nach Alices letztem Zug zu besuchen beabsichtigt.
- $\text{t}[]$ (eine Tabelle der Länge n): wenn $\text{t}[\text{v}] = \text{false}$ bedeutet das, dass Bob plant, den Raum v nach Alices letztem Zug zu besuchen.
- Die erste Schleife **for** sucht die Verstecke von Alice, ausser ihr letztes Versteck.
- Das Ende des Programms ab $\text{a} \leftarrow -1$ sucht nach dem letzten Versteck von Alice.

Die Programm ist unvollständig und ihr werdet gebeten, die fehlenden Ausdrücke zu finden.

Q3(i) [2 Pkte]	Was ist der Ausdruck (i) im Algorithmus ?
Lösung: n	
Q3(j) [2 Pkte]	Was ist der Ausdruck (j) im Algorithmus ?
Lösung: 1	

Q3(k) [3 Pkte]	Was ist der Ausdruck (k) im Algorithmus ?
Lösung: <code>bob[i]</code>	
Q3(l) [2 Pkte]	Was ist der Ausdruck (l) im Algorithmus ?
Lösung: <code>t[v]</code>	
Q3(m) [2 Pkte]	Was ist der Ausdruck (m) im Algorithmus ?
Lösung: <code>j < m</code>	

Eingabe : n, m, bob[]

Ausgabe : alice[]

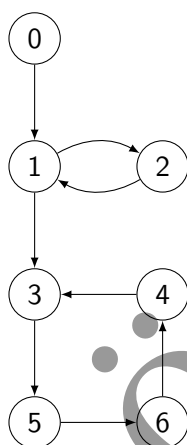
```
alice[] ← {-1, ..., -1} (eine Tabelle der Länge m, initialisiert mit -1)
t[] ← {true, ..., true} (eine Tabelle der Länge n, initialisiert mit true)
cnt ← 0
j ← 0
for (i ← 0 to m-1 step 1)
{
    if (t[bob[i]])
    {
        cnt ← cnt + 1
        if (cnt = ...)           //(i)
        {
            cnt ← ...           //(j)
            while (j < i)
            {
                alice[j] ← ... //(k)
                j ← j + 1
            }
            for (v ← 0 to n-1 step 1)
            {
                t[v] ← true
            }
        }
        t[bob[i]] ← false
    }
}
a ← -1
for (v ← 0 to n-1 step 1)
{
    if (...)                     //(l)
    {
        a ← v
    }
}
while (...)                     //(m)
{
    alice[j] ← a
    j ← j + 1
}
return alice[]
```

Frage 4 – Al Capone

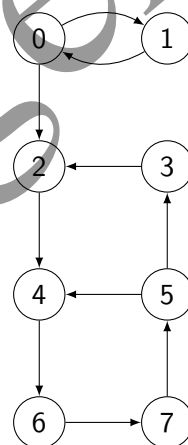
Der berühmte Gangster Al Capone hat beschlossen, sein Netzwerk von illegalen Bars in Chicago neu zu organisieren. Er will mehrere Bars in *Viertel* gruppieren, und jedes Viertel wird von einem Bandenführer geleitet. Um seinen Komplizen die Flucht im Falle von Polizeirazzien zu ermöglichen, möchte er, dass folgende Bedingung erfüllt wird: *zwei Bars befinden sich im gleichen Viertel, wenn und nur dann wenn es in jeder Richtung mindestens einen Weg gibt, der es den Komplizen ermöglicht, mit dem Auto von der einen zur anderen Bar zu gelangen (eventuell vorbei an anderen Bars)*. Ein Viertel kann eine beliebige Anzahl von Bars haben, solange die oben genannte Bedingung für jede Bar-Paar erfüllt ist. Zum Beispiel, die Bars 0 und 1 und 2 sind im selben « Viertel », wenn es möglich ist mit dem Auto von 0 nach 1, von 1 nach 0, von 1 nach 2, von 2 nach 1, von 0 nach 2 und von 2 nach 0 zu fahren.

Hier sind zwei Karten, die die Bars von Al Capone in Chicago repräsentieren. Jede Bar wird durch einen Kreis (mit der Nummer der Bar) dargestellt, und die Pfeile zeigen die (Einweg-)Routen an, die man nehmen kann, um mit dem Auto *direkt* (d.h. ohne an andere Bars vorbeizufahren) von einer Bar zur anderen zu gelangen.

Karte 1:



Karte 2:



Mit der **Karte 1** kann man zum Beispiel (direkt) mit dem Auto von 3 bis 5 fahren. Man kann auch mit dem Auto von 5 bis 3 fahren, aber man muss an 6 und 4 vorbeifahren (weil die Route von 5 nach 3 eine Einbahn ist).

Gebt für jede der folgenden Aussagen über die **Karte 1** an, ob sie wahr oder falsch sind.

	Wahr	Falsch	Aussagen zur Karte 1 .
Q4(a) [1 Pkt]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Man kann mit dem Wagen von 1 nach 0 fahren.
Q4(b) [1 Pkt]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Man kann mit dem Wagen von 6 nach 1 fahren.
Q4(c) [1 Pkt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Man kann mit dem Wagen von 6 nach 5 fahren.
Q4(d) [1 Pkt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1 und 2 können im selben Viertel sein.
Q4(e) [1 Pkt]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	6 und 2 können im selben Viertel sein.
Q4(f) [1 Pkt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	3, 4 und 6 können im selben Viertel sein.
Q4(g) [1 Pkt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	3, 4, 5 und 6 können im selben Viertel sein.
Q4(h) [1 Pkt]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1, 2, 3, 4, 5 und 6 können im selben Viertel sein.

Gebe für jede der folgenden Aussagen über die **Karte 2** an, ob sie wahr oder falsch sind.

	Wahr	Falsch	Aussagen zur Karte 2 .
Q4(i) [1 Pkt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Man kann mit dem Wagen von 1 nach 0 fahren.
Q4(j) [1 Pkt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Man kann mit dem Wagen von 2 nach 7 fahren.
Q4(k) [1 Pkt]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Man kann mit dem Wagen von 7 nach 1 fahren.
Q4(l) [1 Pkt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2 und 5 können im selben Viertel sein.
Q4(m) [1 Pkt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2, 3 und 4 können im selben Viertel sein.
Q4(n) [1 Pkt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	4, 5, 6 und 7 können im selben Viertel sein.
Q4(o) [1 Pkt]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	4, 5, 6 und 7 können im selben Viertel sein und es gibt kein größeres Viertel, das diese vier Bars enthält.
Q4(p) [1 Pkt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2, 3, 4, 5, 6 und 7 können im selben Viertel sein.
Q4(q) [1 Pkt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2, 3, 4, 5, 6 und 7 können im selben Viertel sein und es gibt kein größeres Viertel, das diese sechs Bars enthält.

Leider hat Capone keine so genauen Karten von seinen Bars, deshalb schickt er seinen Komplizen Joe zu Fuß (das ist diskreter) auf die Straßen von Chicago. Capone erklärt Joe, wie er durch die Stadt gehen soll: « Ich gebe dir die Adresse einer ersten Bar. Du folgst allen Straßen, die von dieser Bar ausgehen, unter Beachtung der Einbahnstraßen: zuerst die im Westen (links auf der Karte), dann die im Süden (unten), dann die im Osten (rechts), dann die im Norden (oben), falls es sie gibt. Wenn du am Ende einer Straße eine Bar findest, **in der du während deiner Erkundung noch nie zuvor gewesen bist**, wiederholst du die gleiche Prozedur von dieser neuen Bar aus, oder du gehst zu der vorherigen Bar zurück. Wenn du alle Straßen von einer Bar aus erkundet hast, kehrst du auch zurück. »

Da Joe nicht sehr schlau ist, gibt Capone ihm den Algorithmus `Explore` (siehe Programm unten), dem er genau folgen soll. Der Algorithmus ist eine Funktion, die mit der ersten von Capone vorgegebenen Bar als Parameter aufgerufen wird. Diese Funktion füllt eine Tabelle `order`, deren Index die Nummer einer Bar (von 0 bis $N-1$) ist und die die Reihenfolge (von 1 bis N) angibt, in der die Bars besucht werden.

Wenn beispielsweise `order[i]=1`, bedeutet dies, dass die Bar mit der Nummer i die erste von Capone angegebene Bar ist, und so weiter.

Auf der **Karte 1** und unter der Annahme, dass die Bar 0 die erste ist, wird der Algorithmus `Explore` zuerst die Bar 0 besuchen, dann die 1 (die einzige von 0 aus zugängliche Bar), dann die 3 (wir besuchen zuerst die Bar im Süden, bevor wir die im Osten besuchen), dann 5, 6, 4 und schließlich 2 (nach der Rückkehr).

Q4(r) [3 Pkte]	Gebe den Inhalt der Tabelle <code>order</code> am Ende der Ausführung der Funktion <code>Explore</code> auf Karte 1 an, wobei angenommen wird, dass die erste Bar die Bar 0 ist.
Lösung: [1, 2, 7, 3, 6, 4, 5]	

Q4(s) [3 Pkte]	Gebe die Reihenfolge an, in der die Bars von der Karte 2 von dem Algorithmus <code>Explore</code> besucht werden, ausgehend von der Bar 0.
Lösung: 0, 2, 4, 6, 7, 5, 3, 1.	

Eingabe: N , die Anzahl der Bars

I , die Nummer der ersten Bar (zwischen 0 und $N-1$)

$\text{order}[] \leftarrow \{-1, \dots, -1\}$ (eine Tabelle der Länge N , initialisiert mit -1)

$\text{cpt} \leftarrow 1$ (ein Zähler initialisiert mit 1)

Explore(bar X)

```

{
  if (order[X] = -1)
  {
    order[X] ← cpt
    cpt ← cpt+1

    for (d = ouest, sud, est, nord)
    {
      if (es gibt eine Route mit Richtung d, ausgehend von X)
      {
        Y ← die Bar am Ende der Route mit Richtung d, ausgehend von X
        Explore(Y)
      }
    }
  }
}

```

Explore(I) (füllt die Tabelle $\text{order}[]$ fuer eine Erkundung ausgehend von der Bar I)

Q4(t) [3 Pkte]

Gebe den Inhalt der Tabelle $\text{order}[]$ am Ende der Ausführung der Funktion Explore auf Karte 2 an, wobei angenommen wird, dass die erste Bar die Bar 0 ist.

Lösung: [1, 8, 2, 7, 3, 6, 4, 5]

Nun, da Joe diesen ersten Algorithmus verstanden zu haben scheint, erweitert Capone die `Explore` Funktion, um die Karte in Viertel zu zerlegen. Zunächst bittet er Joe, ein Notizbuch mitzunehmen, damit er eine Liste der von ihm besuchten Bars führen kann: Jedes Mal, wenn er an einer neuen Bar ankommt, fügt er sie am Ende seiner Liste hinzu. Wenn Joe zum Beispiel auf **Karte 1** an der Bar Nummer 5 (zum Zeitpunkt des Aufrufs von `Explore(5)`) ankommt, enthält seine Liste: 0, 1, 3.

Q4(u) [3 Pkte]	Was steht auf Joes Liste, wenn er bei der Bar Nummer 3 auf Karte 2 ankommt, d.h. wenn er <code>Explore(3)</code> anruft?
Lösung: 0, 2, 4, 6, 7, 5.	

Dann fügt Capone der Funktion `Explore` eine zweite Tabelle hinzu. Diese erlaubt es Joe, sich für jede Bar die *kleinste Indexnummer* (in der Tabelle `order` angegeben) zu merken, zu der von dieser Bar aus gefahren werden kann. Capone erkannte dann, dass er durch die Suche nach den Bars X wie `order[X]=low[X]` die Viertel auf der Grundlage der von Joe geführten Liste identifizieren kann. All dies wird in dem neuen Algorithmus im Detail erklärt (siehe Programm unten).

Wenn `Explore(I)` ausgeführt wird, füllt Joe nicht nur die Tabellen `order[]` und `low[]` aus, sondern grenzt auch die Viertel ab, in denen jede Bar (zur Erinnerung) von jeder anderen Bar im Viertel mit dem Auto erreichbar ist.

Q4(v) [3 Pkte]	Was ist der Inhalt der Tabelle <code>low[]</code> am Ende der Ausführung des Algorithmus auf Karte 1 (ausgehend von der Bar 0)?
Lösung: [1, 2, 2, 3, 3, 3, 3]	

Q4(w) [3 Pkte]	Auf Karte 2 (ausgehend von der Bar 0), was ist der Wert von <code>low[5]</code> wenn <code>Explore(3)</code> aufgerufen wird?
Lösung: 3	

Q4(x) [3 Pkte]	Auf Karte 2 (ausgehend von der Bar 0), was ist der Inhalt der Liste von Joe, wenn <code>Explore(1)</code> aufgerufen wird?
Lösung: 0	

Q4(y) [3 Pkte]	Auf Karte 1 (ausgehend von der Bar 0), was sind die vom Capone-Algorithmus berechneten Viertel? Gebe jedes Viertel als eine Reihe von Barnummern an, zum Beispiel {0, 1, 4, 5}, {2, 3, 6}.
Lösung: {0}, {1, 2}, {3, 4, 5, 6}	

Q4(z) [3 Pkte]	Auf Karte 2 (ausgehend von der Bar 0), was sind die vom Capone-Algorithmus berechneten Viertel? Gebe jedes Viertel als eine Reihe von Barnummern an.
Lösung: {0, 1}, {2, 3, 4, 5, 6, 7}	

Eingabe: I, die erste Bar
N, die Anzahl der Bars

order[] \leftarrow {-1, ..., -1} (eine Tabelle der Länge N, initialisiert mit -1)
low[] \leftarrow {-1, ..., -1} (eine Tabelle der Länge N, initialisiert mit -1)
cpt \leftarrow 1 (ein Zähler, initialisiert mit 1)

Explore(bar X)

```
{
  order[X]  $\leftarrow$  cpt
  low[X]  $\leftarrow$  cpt
  Füge X am Ende der Liste hinzu
  cpt  $\leftarrow$  cpt+1
  for (d = ouest, sud, est, nord)
  {
    if (es gibt eine Tote mit Richtung d, ausgehend von X)
    {
      Y  $\leftarrow$  die Bar am Ende der Route mit Richtung d, ausgehend von X
      if (order[Y] = -1)
      {
        Explore(Y)
        low[X]  $\leftarrow$  min(low[X], low[Y])
      }
      else if (Y ist in der Liste)
      {
        low[X]  $\leftarrow$  min(low[X], order[Y])
      }
    }
  }

  if (low[X] = order[X])
  {
    Erstellt eine neues Viertel, die alle Bars enthaelt,
    die sich in der Liste von X (eingeschlossen) befinden,
    und loeschen Sie aus der Liste.
  }
}
```

Explore(I)

Frage 5 – Verwaltungskriege

In einer weit, weit entfernten Welt kämpfen mehrere Länder um die Vorherrschaft auf ihrem Planeten. Ihre Kriege sind anders als unsere und werden ohne Blutvergießen geführt. Sie stellen die Verwaltungsdirektoren der Länder in wilden verbalen Auseinandersetzungen gegeneinander (in Wirklichkeit, wie wir sehen werden, ist es die Macht der Verwaltungsdienste, die zählt). An dieser Stelle ist es wahrscheinlich hilfreich, zu erklären, was eine **Einheit** ist. Ein bestimmtes Land kann in Provinzen unterteilt werden, jede dieser Provinzen kann in Unterprovinzen unterteilt werden, jede ihrer Unterprovinzen kann in Unterunterprovinzen unterteilt werden usw. Jedes Land, jede Provinz, jede Unterprovinz, jede Unterunterprovinz, jede Unterunterunterprovinz, ... ist eine **Einheit**.

Nehmen wir das Beispiel des Landes Heldor:



Fig 1: Karte von Heldor

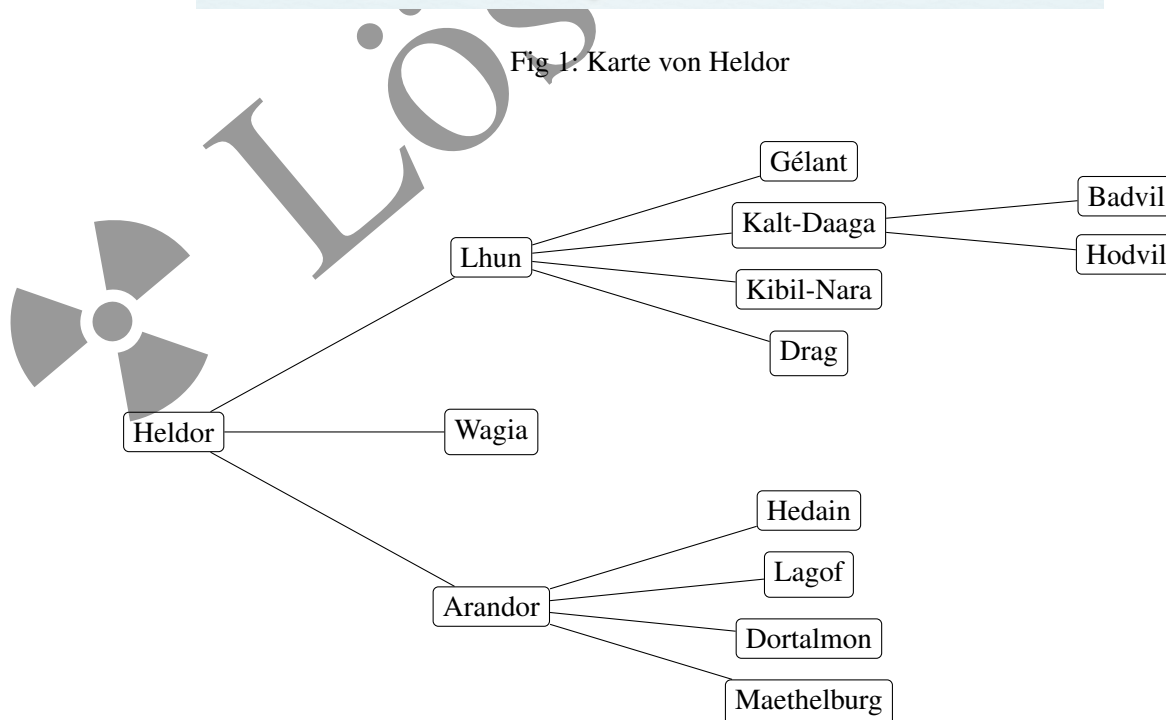


Fig 2: Administrative Karte der **Einheiten** von Heldor.

Jede **Einheit** hat ihre eigene Verwaltung mit eigener Macht. Offensichtlich bedeutet die administrative Komplexität, dass eine Einheit umso weniger effizient ist, je mehr Einheiten sie unter ihr verwaltet (Bürokratie ist in allen Welten ein Problem).

Zum Beispiel verwaltet Lhun 7 Einheiten: sie selbst und 6 Einheiten unter ihr. Man sagt, dass Lhun 7 **Untereinheiten** hat (weil Lhuns zentrale Verwaltung als eine eigene Untereinheit gezählt werden muss!).

Andererseits gilt: Je kleiner eine **Einheit** im Verhältnis zum Land ist, desto weniger Geld hat sie und desto weniger mächtig ist ihre Verwaltung.

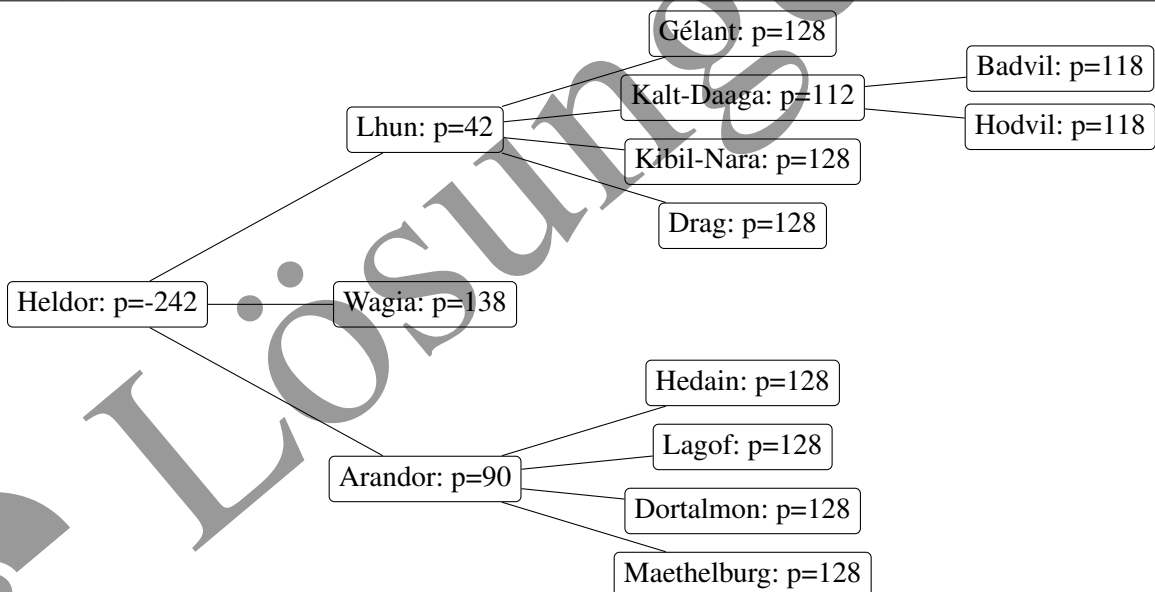
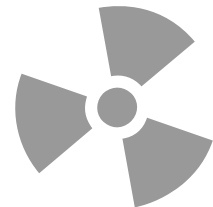
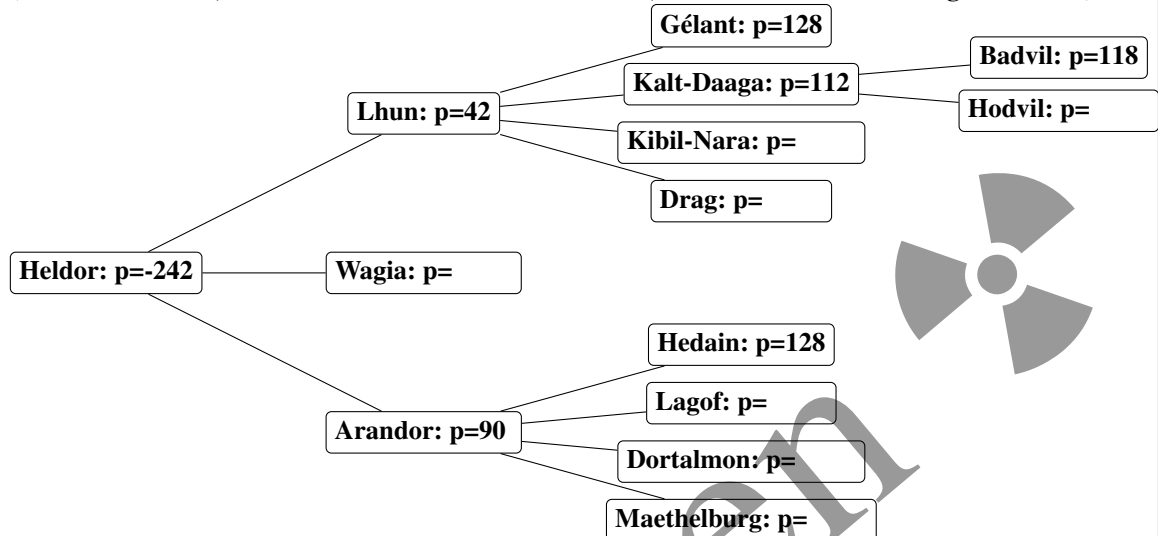
Insbesondere, wenn **X** eine **Einheit** auf der Ebene n ist (0 für ein Land, 1 für eine Provinz, 2 für eine Unterprovinz, 3 für eine Unter-Unterprovinz, ...), und wenn s die Anzahl der Untereinheiten von **X** ist, dann ist p , die **Verwaltungsmacht** von **X**, hat dann den Wert $p = 150 - 10n - 2s^2$.

- Beispiel für die Unter-Unterprovinz Badvil: $n = 3$ und $s = 1$, dann $p = 150 - 10 \cdot 3 - 2 \cdot 1^2 = 150 - 30 - 2 = 118$.
- Beispiel für die Provinz Lhun: $n = 1$ und $s = 7$, dann $p = 150 - 10 \cdot 1 - 2 \cdot 7^2 = 150 - 10 - 98 = 42$.
- Beispiel für das Land Heldor: $n = 0$ und $s = 14$, dann $p = 150 - 10 \cdot 0 - 2 \cdot 14^2 = 150 - 392 = -242$.

Q5(a) [7 Pkte]

Vervollständigt die administrative Karte von Heldor, indem ihr die Macht der Einheiten vermerkt.

(Falls erforderlich, antwortet zuerst hier in einer Kladde, bevor ihr den Antwortbogen ausfüllt.)



Lösung:

Ihr seht, dass Heldor mit Verwaltungsarbeit überfordert ist und dass seine Macht negativ ist, d.h. Heldor verbraucht Ressourcen, um seine Verwaltung über Wasser zu halten!

In ähnlicher Weise könnte eine Unter-Unter-Unter-... Provinz eine negative oder Null-Leistung haben, selbst wenn sie außer sich selbst keine andere Untereinheit hat. Aber wie viele "Unter" benötigt man?

Q5(b) [2 Pkte]

Wie viele "Unter" muss man vor dem Wort "Provinz" voranstellen, um sicherzustellen, dass seine Macht gleich Null oder negativ (≤ 0) ist?

Lösung: 14 (weil $150 - 10n - 2 \leq 0 \iff n \geq 14,8$ und es gibt 14 "Unter" im Niveau 15)

Was wirklich zählt, ist die **Gesamtmacht**. Die **Gesamtmacht** einer Einheit **X** erhält man durch Addition der Mächte aller Untereinheiten von **X** (also ohne **X** selbst zu vergessen). Zum Beispiel ist Badvils **Gesamtmacht** gleich ihrer Macht (118), da sie keine andere Untereinheit als sich selbst hat.

Auf der anderen Seite ist Lhuns **Gesamtmacht** die Summe von 7 Mächten (Hilfe: Der Wert beträgt 774).

Q5(c) [1 Pkt]	Wie hoch ist die Gesamtmacht von Hodvil?
Lösung: 118	
Q5(d) [1 Pkt]	Wie hoch ist die Gesamtmacht von Wagia?
Lösung: 138	
Q5(e) [2 Pkte]	Wie hoch ist die Gesamtmacht von d'Arandor?
Lösung: $90 + 4 \cdot 128 = 602$	
Q5(f) [2 Pkte]	Wie hoch ist die Gesamtmacht von Heldorf?
Lösung: $-242 + 774 + 138 + 602 = 1272$	

Wenn zwei Länder gegeneinander antreten, gewinnt das Land mit der größeren **Gesamtmacht**. Im Falle eines Unentschiedens verstricken sich die Verwaltungen in endlose Streitigkeiten und die Länder verschwinden in einem Verfahrensfehler. Die Funktion `GESAMTMACHT(X, n)` ermöglicht die **Gesamtmacht** einer Einheit x auf der Ebene n .

Q5(g) [2 Pkte]	Gebt den Ausdruck, dass das Land A den Kampf gegen das Land B gewinnt. erinnert euch, dass alle Länder ein Niveau $n=0$ haben und dass es im Fall einer Gleichheit keinen Gewinner gibt.
Lösung: <code>GESAMTMACHT(A, 0) > GESAMTMACHT(B, 0)</code>	

Jede Einheit der Ebene n kennt die Liste ihrer Untereinheiten der Ebene $n + 1$, die ihre **Kinder** genannt werden. Dieser seltsame Name stammt aus der Graphentheorie, die zum Zeichnen administrativer Karten verwendet wird. Auf einer Verwaltungskarte verbindet eine separate Linie jede Einheit mit jedem ihrer Kinder (siehe Abb. 2). Die Kinder eines Landes sind seine Provinzen, die Kinder einer Provinz sind seine Unterprovinzen,...

Um die administrative Macht einer Einheit x zu berechnen, ist es notwendig, ihre Anzahl von Untereinheiten (einschließlich der Einheit) zu kennen. Hier ist der unvollständige Pseudo-Code (eine Zeile fehlt) der Funktion `BERECHNE_S(X)`, die dies tut.

```

Funktion BERECHNE_S(X) {
  anz_einheiten ← 1
  fuer jedes K Kind von X {
    ... //(i)
  }
  return anz_einheiten
}

```

Beachtet, dass die Zeile "fuer jedes Kind K von X" beginnt eine Schleife, die für jedes Kind von x einmal ausgeführt wird, und dieses Kind wird durch die Variabel K in der Schleife bestimmt. Zum Beispiel, wenn $x = \text{Heldorf}$, wird die Schleife 3-mal ausgeführt, einmal mit $E = \text{Lhun}$, einmal mit $E = \text{Wagia}$, einmal mit $E = \text{Arandor}$. Wenn x kein Kind hat, wird die Schleife nicht ausgeführt (sie wird 0-mal ausgeführt, da es 0 Kinder gibt).

Q5(h) [2 Pkte]		Was ist die Zeile (i) im Pseudo-Code der Funktion BERECHNE_S () ?
	<input type="checkbox"/>	<code>anz_einheiten \leftarrow BERECHNE_S(E) + 1</code>
	<input type="checkbox"/>	<code>anz_einheiten \leftarrow anz_einheiten + 1</code>
	<input checked="" type="checkbox"/>	<code>anz_einheiten \leftarrow anz_einheiten + BERECHNE_S(E)</code>
	<input type="checkbox"/>	<code>anz_einheiten \leftarrow BERECHNE_S(E)</code>
	<input type="checkbox"/>	<code>anz_einheiten \leftarrow anz_einheiten + BERECHNE_S(E) + 1</code>

Vervollständige den Pseudo-Code der Funktion `GESAMTMACHT(X, n)`, die (wie bereits berichtet) die Berechnung der **Gesamtmacht** einer Einheit X der Ebene n ermöglicht.

```

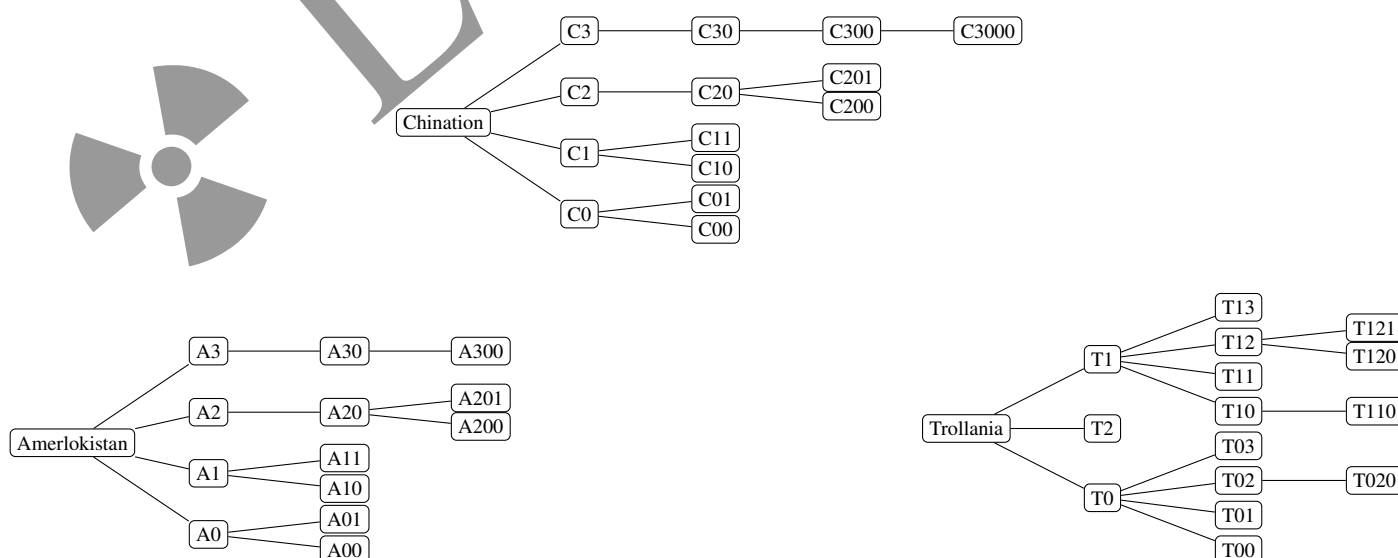
Funktion GESAMTMACHT(X, n) {
    macht_von_X ← ...                // (i)
    macht_der_kinder ← 0
    fuer jedes K Kind von X {
        macht_der_kinder ← macht_der_kinder + ...    // (j)
    }
    return ...                        // (k)
}

```

Hinweise: Ihr könnt und solltet `BERECHNE_S(X)` verwenden. Bedenkt, dass diese Funktion korrekt implementiert ist, auch wenn ihr die vorige Frage nicht beantwortet habt. Um das Quadrat einer Zahl (r) zu berechnen, multipliziert ihr sie mit sich selbst ($r*r$) oder ihr setzt sie in die Potenz 2 (r^2).

Q5(i) [2 Pkte]	Was ist der Ausdruck (i) im Pseudocode der Funktion <code>GESAMTMACHT(X, n)</code>?
Lösung: $150 - 10*n - 2*BERECHNE_S(X)*BERECHNE_S(X)$	
Q5(j) [2 Pkte]	Was ist der Ausdruck (j) im Pseudocode der Funktion <code>GESAMTMACHT(X, n)</code>?
Lösung: <code>GESAMTMACHT(E, n+1)</code>	
Q5(k) [2 Pkte]	Was ist der Ausdruck (k) im Pseudocode der Funktion <code>GESAMTMACHT(X, n)</code>?
Lösung: <code>macht_von_X + macht_der_kinder</code>	

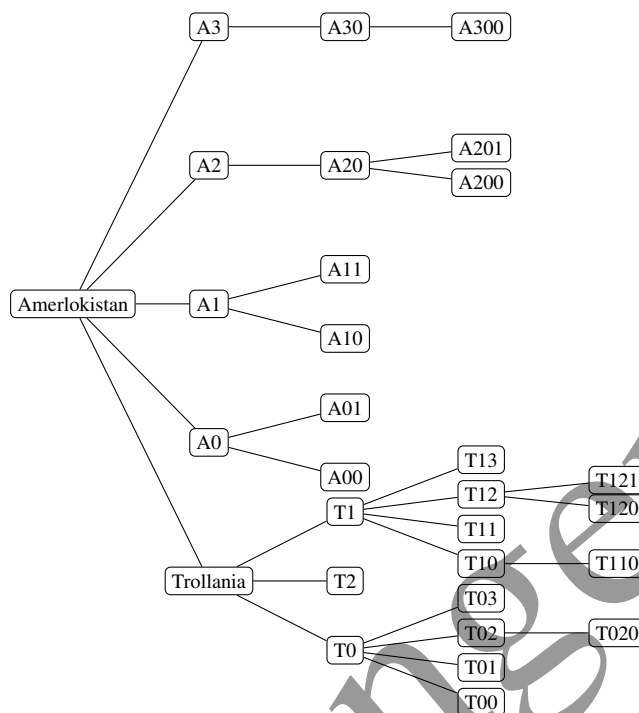
Wenn zwei Länder in den Krieg ziehen, wird das unterlegene Land vom überlegenen Land absorbiert und wird zu einer seiner Provinzen. Dies verändert natürlich die administrative Macht des Siegerlandes! Nehmen wir ein Beispiel. Hier sind die administrativen Karten von drei Ländern (statt der Namen wurden Postleitzahlen verwendet, sie sind kürzer).



Unsere Spione haben für Sie die **Gesamtmacht** dieser drei Länder berechnet:

Chination: 1352	Amerlokistan: 1332	Trollania: 1324
------------------------	---------------------------	------------------------

Im Falle eines Krieges zwischen Amerlokistan und Trollanien würde Amerlokistan Trollanien absorbieren. So würde die Karte nach diesem Sieg aussehen:



Amerlokistans neue **Gesamtmacht** nach dem Krieg würde **1088** entsprechen.

Es gibt eine Formel zur Berechnung der **Gesamtmacht** eines Kriegssiegers.

Q5(l) [5 Pkte]		Sei x_1 ein Land mit Gesamtmacht t_1 mit s_1 Untereinheiten (einschließlich x_1). Sei x_2 ein Land mit Gesamtmacht t_2 mit s_2 Untereinheiten (einschließlich x_2). Wenn $t_1 > t_2$, welcher Ausdruck berechnet dem Gewinner der Auseinandersetzung zwischen x_1 und x_2 die neue Gesamtmacht?
	<input type="checkbox"/>	$t_1 + t_2$
	<input type="checkbox"/>	$t_1 + t_2 - 10 \cdot s_2$
	<input type="checkbox"/>	$t_1 + t_2 - 2 \cdot s_2 \cdot s_2$
	<input type="checkbox"/>	$t_1 + t_2 + 2 \cdot s_1 \cdot s_1 - 2 \cdot (s_1 + s_2) \cdot (s_1 + s_2)$
	<input type="checkbox"/>	$t_1 + t_2 + 2 \cdot s_2 \cdot s_2 - 2 \cdot (s_1 + s_2) \cdot (s_1 + s_2)$
	<input checked="" type="checkbox"/>	$t_1 + t_2 - 10 \cdot s_2 + 2 \cdot s_1 \cdot s_1 - 2 \cdot (s_1 + s_2) \cdot (s_1 + s_2)$
	<input type="checkbox"/>	$t_1 + t_2 - 10 \cdot s_2 + 2 \cdot s_2 \cdot s_2 - 2 \cdot (s_1 + s_2) \cdot (s_1 + s_2)$

In dieser Welt gibt es immer **nur einen Krieg auf einmal**, immer zwischen den **beiden größten Verwaltungsmächten**. Erinnern wir uns daran, dass der Gewinner jedes Mal den Verlierer absorbiert, und dass im Falle eines Gleichstandes die Verwaltungen in einen endlosen Streit verstrickt werden und die beiden Länder verschwinden. Dann ziehen die grössten beiden neuen Einheiten gegeneinander in den Krieg usw., usw., bis nur noch eine Nation übrig ist (oder gar keine mehr...).

Q5(m) [2 Pkte]		Wie wird der erste Krieg zwischen Chination, Amerlokistan und Trollanien ausgetragen?
	<input checked="" type="checkbox"/>	Chination würde Amerlokistan schlagen.
	<input type="checkbox"/>	Amerlokistan würde Chination schlagen.
	<input type="checkbox"/>	Chination würde Trollania schlagen.
	<input type="checkbox"/>	Trollania würde Chination schlagen.
	<input type="checkbox"/>	Trollania würde Amerlokistan schlagen.
	<input type="checkbox"/>	Amerlokistan würde Trollania schlagen.

Q5(n) [3 Pkte]	Was wird die neue Gesamtmacht des Siegers dieses ersten Krieges sein?
Lösung: Chination gewinnt und seine neue Gesamtmacht ist 1312	

Erläuterungen zur Lösung.

Für Chination: $t1=1352$, $s1=15$.

Für Amrelokistan: $t2=1332$, $s2=14$.

Chination gewinnt und seine neue Gesamtmacht ist $1352 + 1332 - 10 \cdot 14 + 2 \cdot 15 \cdot 15 - 2 \cdot 29 \cdot 29 = 1312$

Q5(o) [2 Pkte]		Wer wird der endgültige Sieger sein, nach dem zweiten Krieg zwischen dem Drittland und dem Sieger des ersten Krieges?
	<input type="checkbox"/>	Chination
	<input type="checkbox"/>	Amerlokistan
	<input checked="" type="checkbox"/>	Trollania

Q5(p) [3 Pkte]	Was ist die neue Gesamtmacht des Endsiegers?
Lösung: Trollanien gewinnt und seine neue Gesamtmacht ist -1192	

Erläuterungen zur Lösung.

Für Trollania: $t1=1324$, $s1=16$.

Für Chination: $t2=1312$, $s2=29$.

Trollania gewinnt und seine neue Gesamtmacht ist $1324 + 1312 - 10 \cdot 29 + 2 \cdot 16 \cdot 16 - 2 \cdot 45 \cdot 45 = -1192$

Alles mit der Hand zu machen, ist langweilig, nicht wahr? Es ist an der Zeit, ein Verfahren einzuführen.

Alle Länder werden in einer Liste geführt: `liste_laender`. Diese Liste unterstützt drei Operationen:

- `liste_laender.groesse()`, die die Anzahl der Länder in der Liste zurückgibt,
- `liste_laender.groesste_verwaltung()`, die das Land mit der grössten Gesamtmacht zurückgibt **und** es aus der Liste entfernt (wenn mehrere Länder die gleiche Gesamtleistung haben, dann wird eines von ihnen “zufällig” ausgewählt).
- `liste_laender.hinzufuegen(X)`, die das Land `X` zu der Liste hinzufügt.

Darüber hinaus enthalten die Variablen “Land” eine Beschreibung der Karte des Landes. Wenn `A` und `B` 2 Variablen “Land” sind, dann verändert das Kommando `A.provinz_hinzufuegen(B)` die Variabel `A`, um eine neue Provinz hinzuzufügen, die eine perfekte Kopie des Landes `B` ist.

Ihr werdet gebeten, die Funktion `KRIEGE` auszufüllen, die aufeinanderfolgende Kriege zwischen den Ländern in der Liste simuliert, bis nur noch ein (oder kein) Land mehr in der Liste vorhanden ist.

```

funktion KRIEGE() {
  while (...) // (e1)
  {
    A ← ... // (e2)
    B ← ... // (e2)
    p_tot_A ← GESAMTMACHT(A, 0)
    p_tot_B ← GESAMTMACHT(B, 0)
    if (p_tot_A > p_tot_B)
    {
      A.Provinz_hinzufuegen(B)
      ... // (e3)
    }
  }
}

```

Q5(q) [2 Pkte]	Wie lautet der Ausdruck (e1) in der Funktion <code>KRIEGE()</code> ?
Lösung: <code>liste_laender.groesse() > 1</code>	
Q5(r) [2 Pkte]	Wie lautet der Ausdruck (e2), der sich zweimal in der Funktion <code>KRIEGE()</code> befindet?
Lösung: <code>liste_laender.groesste_verwaltung()</code>	
Q5(s) [2 Pkte]	Wie lautet der Ausdruck (e3) in der Funktion <code>KRIEGE()</code> ?
Lösung: <code>liste_laender.liste_laender(A)</code>	