be-OI 2020	Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp	O
Finale - JUNIOR Saturday 7th March 2020	PRÉNOM : . . . . NOM : . . . . ÉCOLE :	**Réservé**

Finals of the Belgian Informatics Olympiad 2020 (time allowed : **2h maximum**)

General information (read carefully before attempting the questions)

1. Verify that you have received the correct set of questions (as mentioned above in the header):

   - for the students in the second year of the Belgian secundary school system or below (US grade 8, UK year 9): category **cadets**.
   - for the students in the third or fourth year of the Belgian secundary school system or equivalent (US grade 9-10, UK year 10-11): category **junior**.
   - for the students in the fifth year of the Belgian secundary school system or equivalent (US grade 11, UK year 12) and above: category **senior**.

2. Write your first name (PRÉNOM), first name (NOM) and school (ÉCOLE) **on the first page only**.

3. Write **your answers** on the provided answer sheets, which you will find **at the end of the booklet**.

4. If you have to write outside the answer boxes due to a mistake, then continue writing **on the same sheet of paper**.

5. Write **clearly and legibly** with a blue or black **pen or ballpoint**.

6. You may only have writing materials with you. Calculator, GSM, . . . are **forbidden**.

7. You can always request extra scratch paper from the invigilator.

8. When you have finished, **hand in this first page (with your name on it), and the pages with your answers**. You can keep the other pages.

9. All the snippets of code in the exercises are written in **pseudo-code**. On the next pages you will find a description of the pseudo-code that we use.

10. If you have to respond with code, you can do so in **pseudo-code** or in any **current programming language** (such as Java, C, C ++, Pascal, Python, ...). We do not deduct points for syntax errors.

Good Luck !

The Belgian IT Olympiad is possible thanks to the support from our members:

Pseudo-code checklist

Data is stored in variables. We change the value of a variable using ←. In a variable, we can store whole numbers, real numbers, or arrays (see below), as well as Boolean (logical) values: true/correct (**true**) or false/wrong (**false**). It is possible to perform arithmetic operations on variables. In addition to the four conventional operators ($+$, $-$, $\times$ and $/$), you can also use the operator %. If a and b are whole numbers, then a/b and $a\%b$ denote respectively the quotient and the remainder of the division. For example, if $a = 14$ and $b = 3$, then $a/b = 4$ and $a\%b = 2$.

Here is a first code example, in which the variable age receives 17.

```
birthYear ← 2003
age ← 2020 − birthYear
```

To run code only if a certain condition is true, we use the instruction **if** and possibly the instruction **else** to execute another code if the condition is false. The next example checks if a person is of legal age and stores the price of their movie ticket in the variable $price$. Note the comments in the code.

```
if (age ≥ 18)
{
    price ← 8 // This is a comment.
}
else
{
    price ← 6 // cheaper !
}
```

Sometimes when one condition is false, we have to check another. For this we can use **else if**, which comes down to executing another **if** inside the **else** of the first **if**. In the following example, there are 3 age categories that correspond to 3 different prices for the movie ticket.

```
if (age ≥ 18)
{
    price ← 8 // Price for a person of legal age (adult).
}
else if (age ≥ 6)
{
    price ← 6 // Price for children aged 6 or older.
}
else
{
    price ← 0 // Free for children under 6.
}
```

To handle several elements with a single variable, we use an array. The individual elements of an array are identified by an index (which is written in square brackets after the name of the array). The first element of an array tab has index 0 and is denoted $tab[0]$. The second element has index 1 and the last has index $N − 1$ if the array contains N elements. For example, if the array tab contains the 3 numbers 5, 9 and 12 (in this order), then $tab[0]= 5$, $tab[1]= 9$, $tab[2]= 12$. The array is size 3, but the highest index is 2.

To repeat code, for example to browse the elements of an array, we can use a **for** loop. The notation **for (**$i \leftarrow a$ **to** b **step** k**)** represents a loop which will be repeated as long as $i \leq b$, in which i begins with the value a, and is increased by k at the end of each step. The following example calculates the sum of the elements of the array tab assuming its size is N. The sum is found in the variable sum at the end of the execution of the algorithm.

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

You can also write a loop using the instruction **while**, which repeats code as long as its condition is true. In the next example, we're going to divide a positive integer N by 2, then by 3, then by 4 . . . until it is a single digit number (i.e. until $N < 10$).

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

Often the algorithms will be in a frame and preceded by descriptions. After **Input**, we define each of the arguments (variables) given as input to the algorithm. After **Output**, we define the state of certain variables at the end of the algorithm execution and possibly the returned value. A value can be returned with the instruction **return**. When this instruction is executed, the algorithm stops and the given value is returned.

Here is an example using the calculation of the sum of the elements of an array.

```
Input  : tab, an array of N numbers.
         N, the number of elements in the array.
Output : sum, the sum of all the numbers in the array.


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

Note: in this last example, the variable i is only used as a counter for the **for** loop. There is therefore no description for it either in **Input** or in **Output**, and its value is not returned.

Question 1 – Circus

Professor Zarbi has developed circus tricks with his animal robots. He uses two stands ☐ A ☐ and ☐ i ☐ (which will be noted ⬜A⬜ and ⬜i⬜) on which the animals can be placed, stacked one above the other without any height limit. Animals can be arranged in any way, one stand may be empty and all animals may be stacked on the other.

Here are some possible arrangements with 3 animals:
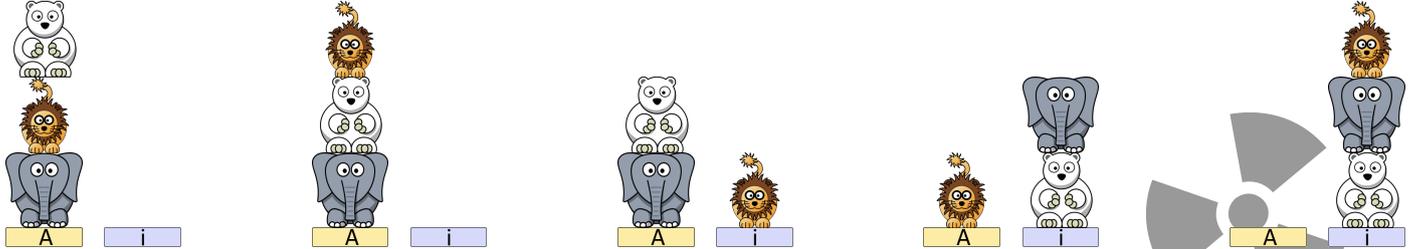
Q1(a) [2 pts]	How many different ways exist to arrange 2 animals on the stands ?
Solution: 6	

Q1(b) [4 pts]	How many different ways exist to arrange 3 animals on the stands ?
Solution: 24	

During the show, Professor Zarbi gives instructions that make the animals change places.

Instructions for "climbing".
MA: On hearing it, the lowest animal on the stand ⬜A⬜ climbs to the top of its stack.
Mi: On hearing it, the lowest animal on the stand ⬜i⬜ climbs to the top of its stack.
(Nothing happens if there are less than 2 animals on the corresponding stand.)

Instructions for "jumping".
**TA**: On hearing it, the highest animal on the stand ⬜A⬜ jumps and lands at the top of the stack on the stand ⬜i⬜.
Ti: On hearing it, the highest animal on the stand ⬜i⬜ jumps and lands at the top of the stack on the stand ⬜A⬜.
(Nothing happens if there are no animals on the corresponding stand.)

Example: 4 animals are arranged as shown on the left image and the professor successively gives the instructions **MA TA Mi Ti**. The images show how the animal arrangement evolve after each instruction.
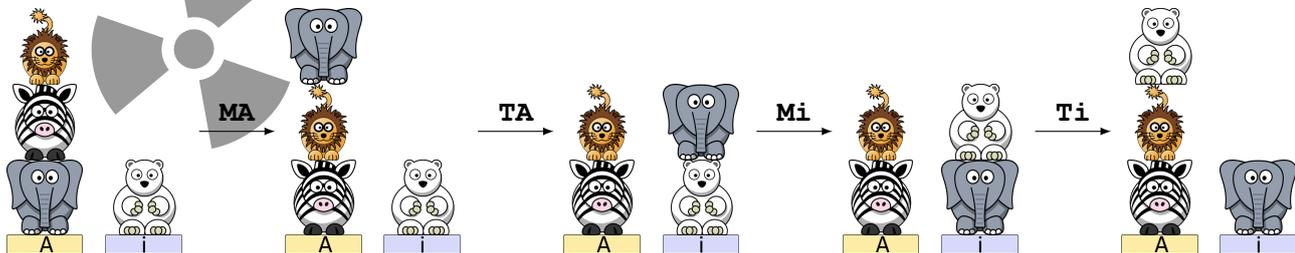
We can summarize it by drawing a single arrow carrying all of the instructions:
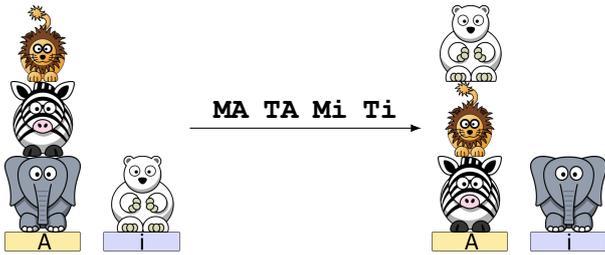
The length of an instruction list is the number of instructions in the list. For example, the instruction list **MA TA Mi Ti** is of length 4.

In the questions that follow, you should find a list of instructions among {**MA**, **Mi**, **TA**, **Ti**} to transform the left arrangement into the right arrangement. **Your list should be as short as possible.** If your list is not of minimum length, you score only half the points for the question.

Q1(c) [2 pts]	
Solution: **MA**	

Q1(d) [2 pts]	
Solution: **TA MA Ti MA**	

|	
Q1(e) [2 pts]	
Solution: for example **MA TA TA Mi Ti Mi Ti**	

Q1(f) [4 pts] | **Two animals must pass through all the positions as quickly as possible. Starting from the position shown on the left, you must go through all the other positions only once, before returning to the starting position.**

Instructions passing through all positions?

Solution: **MA TA TA Mi Ti Ti** ou **TA TA Mi Ti Ti MA**

Solutions
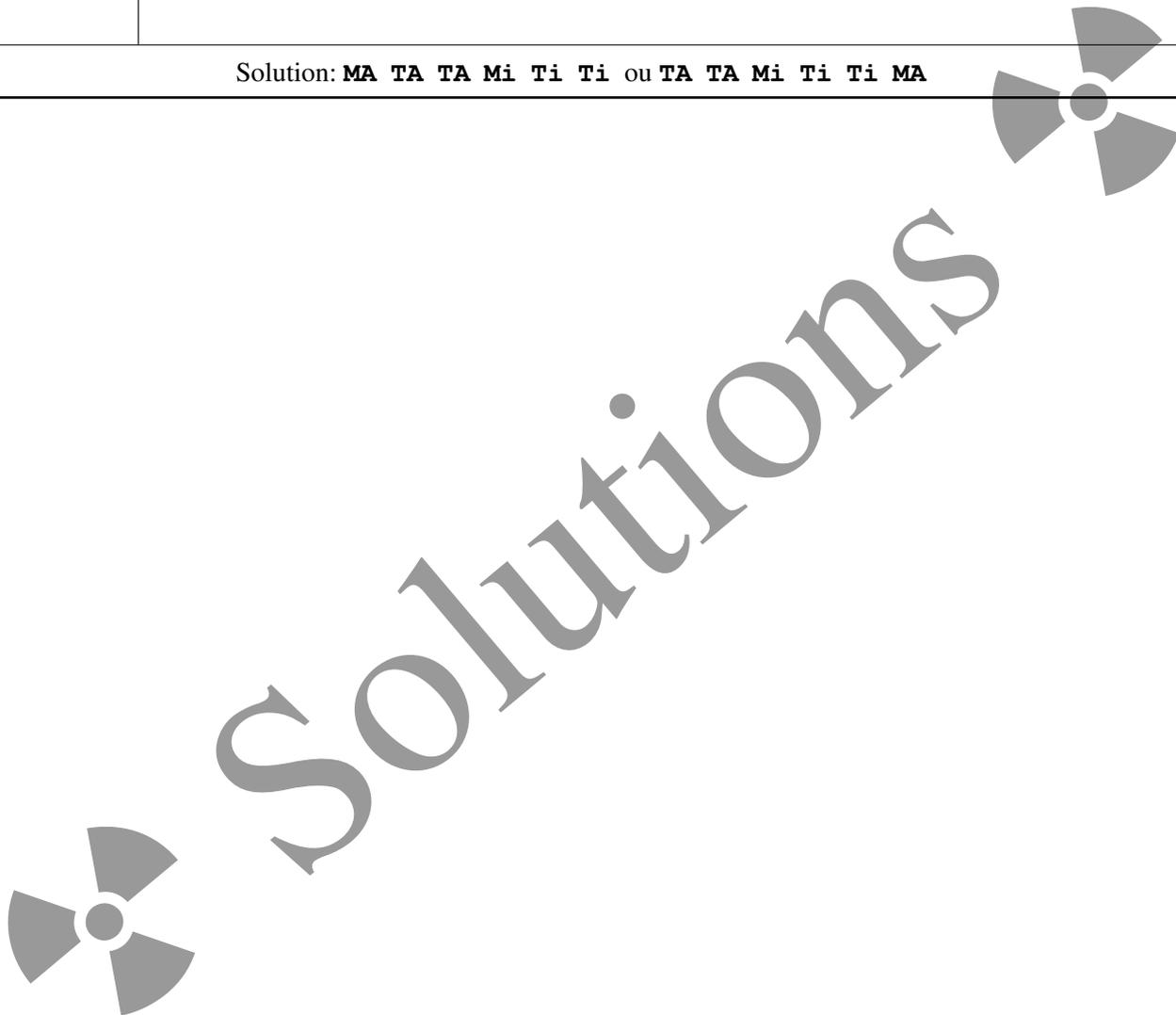
Let us add new instructions for "turning".

RA: On hearing it, the highest animal on the stand $\boxed{\texttt{A}}$ turns around by 180° (makes a half-turn).

Ri: On hearing it, the highest animal on the stand $\boxed{\texttt{i}}$ turns around by 180° (makes a half-turn).

(Nothing happens if there are no animals on the corresponding stand.)

Each animal can now be seen from the front or from the back and the following positions are considered to be different:
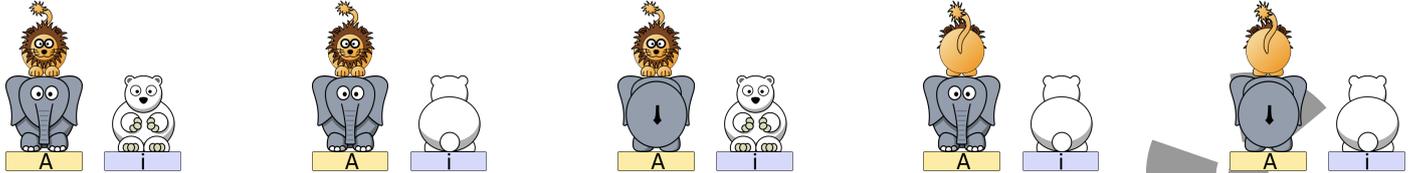
Q1(g) [2 pts]	Taking into account that each animal can be seen from the front or from the back, how many different ways exist to arrange 2 animals on the stands ?
Solution: $6 \cdot 2^2 = 24$	

Q1(h) [4 pts]	Taking into account that each animal can be seen from the front or from the back, how many different ways exist to arrange 3 animals on the stands ?
Solution: $24 \cdot 2^3 = 192$	

When an animal "climbs" or "jumps" it does not change its orientation "seen from the front or from the back".

In the questions that follow, you should find a list of instructions among {**MA**, **Mi**, **TA**, **Ti**, **RA**, **Ri**} to transform the left arrangement into the right arrangement.

Your list should be as short as possible. If your list is not of minimum length, you score only half the points for the question.

Q1(i) [2 pts]	
Solution: **TA RA Ti** ou **MA RA MA**	

Q1(j) [2 pts]	
Solution: **Ti MA RA MA**	

Q1(k) [2 pts]		
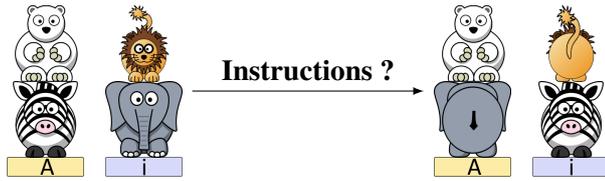
Solution: for example: **Ri Mi Ri Ti MA TA MA Mi**

Question 2 – Open the door

A door is secured by a secret 4-digit code.

The door opens immediately as soon as 4 keys pressed consecutively correspond to the secret code.

For example, if you successively press 6,4,5,5,0,8,6,7 the door will open if the secret code is 6455, 4550, 5508, 5086 or 0867.

We can program a robot using the command `press(a)` which causes the robot to press the $\boxed{a}$ key (a being of course a number from 0 to 9).

Program 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 the following questions, you only have to take into account the keys pressed in Program 1.

Q2(a) [1 pt]	**During the execution of Program 1, how many keys will be pressed in total ?**
Solution: $4 \cdot 10^4 = 40000$	

Q2(b) [1 pt]	**How many times will the $\boxed{5}$ key be pressed ?**
Solution: 4000	

Q2(c) [2 pts]	**How many 4-digit combinations will be tested by Program 1 ? If a combination is tested more than once, count all the attempts.**
Solution: $40000 - 3 = 39997$	

Q2(d) [1 pt]	**How many different 4-digit combinations will be tested by Program 1 ? If a combination is tested more than once, count only one test.**
Solution: 10000	

Q2(e) [1 pt]	**Can you be certain that the door opens during the execution of Program 1 regardless of the secret code ?**
Solution: YES	

Let's number the keystrokes executed by the program with integers starting from **0** (make no mistake, we start counting from **zero**). The first 20 keystrokes (numbered from 0 to 19) are 0,0,0,0, 0,0,0,1, 0,0,0,2, 0,0,0,3, 0,0,0,4. Note that any keystroke numbered with a multiple of 4 always corresponds to an instruction `press(a)` of the program, and never to

press(b), neither press(c), nor press(d).

Q2(f) [1 pt]	**What is the key pressed by the robot during execution of the keystroke with number 680 ?**
Solution: $\boxed{0}$	

Q2(g) [1 pt]	**What is the key pressed by the robot during execution of the keystroke with number 682 ?**
Solution: $\boxed{7}$	

Q2(h) [1 pt]	**Give the 8 digits pressed successively by the robot, starting with the digit corresponding to the keystroke with number 732.**
Solution: 0183 0184	

A "**good sequence**" is a sequence of 8 digits pressed successively by the robot during the execution of Program 1 and **starting with a keystroke with an number that is a multiple of 4**, in other words starting with a key pressed during an instruction press(a) of the program.
Here are examples of "good sequences": 0000 0001, 2307 2308, 6499 6500 (a space has been inserted in the middle to facilitate reading).
Note that a good sequence always consists of 2 successive 4-digit numbers.
We say that a good sequence **tests a combination** if this combination appears in the good sequence.
For example, the good sequence **2307 2308** tests the combinations 2307, 3072, 0723, 7230 and 2308.
Another example: the good sequence **7171 7172** tests the combinations 7171 and 1717, again 7171 and 1717, and finally 7172.
Note that the two tests of 7171 do not take place at the same time, they are two different tests of the same combination (same for 1717).

Q2(i) [5 pts]	**Find the 5 good sequences that test the combination 3108. Hint: it is possibe to "split" the sequences in different ways (|3108, 3|108, 31|08, …).**
Solution: 08**31 08**32, 108**3 108**4, 3107 **3108**, 3108 3109 and 8**310 8**311	

Q2(j) [1 pt]	**How many times does the robot successively press the keys 3,1,0,8 ? In other words, how many different times has the combination 3108 been tested ?**
Solution: 4 (the good sequences 3107 3108 and 3108 3109 perform the same test)	

Q2(k) [3 pts]	**Find the 3 good sequences that test the combination 6464.**
Solution: 4**646 464**7, 6463 6464 and **6464 64**65	

Q2(l) [1 pt]	**How many times does the robot successively press the keys 6,4,6,4 ? In other words, how many different times has the combination 6464 been tested ?**
Solution: 4 (be careful not to count the same test more than once)	

Q2(m) [4 pts]	**Find 4 good sequences that test the combination 9990.**
Solution: 89**99 90**00, 989**9 9900**, 9989 **9990** and 9990 9991.	

| Q2(n) [1 pt]	How many times does the robot successively press the keys 9,9,9,0 ? In other words, how many different times has the combination 9990 been tested ?
Solution: 3	

Q2(o) [1 pt]	How many times does the robot successively press the keys 8,4,5 ? Hint: If X is a digit, for instance 845X, 45X8 and 5X84 are secret codes.
Solution: $4 \cdot 10 = 40$	

Solutions

Let us now consider a non-deterministic program, that is to say that some keys are pressed at random.
In this program, the `random()` function randomly chooses a number from 0 to 9.
Pay attention to the boundaries of the second loop: it is not a loop from 0 to 9 but from 0 to **a**.

Program 2 :

```
for (a ← 0 to 9 step 1) {
  for (d ← 0 to a step 1) {
        press(a)
        press(random())
        press(random())
        press(d)
  }
}
```
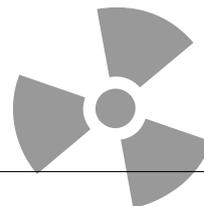
In the following questions, you only have to take into account only the keys pressed in Program 2.

Q2(p) [3 pts]	**How many 4-digit combinations will be tested by Program 2 ? It is possible that certain combinations are tested several times, you should count all the attempts.**
Solution: $(1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10) \cdot 4 - 3 = 217$	

Q2(q) [2 pts]	**Suppose the secret code is 7431. Is it possible that the door opens when we execute program 2 ?**
Solution: YES	

Q2(r) [2 pts]	**Suppose the secret code is 4444. Is it possible that the door opens when we execute program 2 ?**
Solution: YES	

Q2(s) [2 pts]	**Suppose the secret code is 2345. Is it possible that the door opens when we execute program 2 ?**
Solution: YES	

*Explanations: for example by writing the keys pressed from the moment when **a** is equal to 3 in the first loop and **d** is equal to 2 in the second, the program will generate 3**2 3**3 where the * are random numbers and the last 2 * may be 4 and 5.*

It is possible to write an optimal program that generates a keystroke sequence in which all combinations of 4 digits are each tested exactly once. In the questions that follow, we ask to take into account only the keys pressed by such an optimal program.

Q2(t) [4 pts]	**How many keys will be pressed in total when running an optimal program ?**
Solution: $10^4 + 3 =$ 10,003.	

Explanation: There are 10,000 combinations to test, each keystroke must start a new combination, we must not forget the last 3 digits of the last combination.

There are several ways to write an optimal program, but in each case the first 3 digits of the generated sequence are the same as the last three.
For example, if an optimal sequence begins with 123 then it will also end with 123.

Q2(u) [2 pts]	**If you run an Optimal123 program that generates a sequence starting with 123, how many times will the robot successively press the keys 8,4,5 ?**
Solution: 10	

Q2(v) [2 pts]	**If you execute an Optimal123 program which generates a sequence starting with 123, how many times will the robot successively press the 1,2,3 keys ?**
Solution: 11	
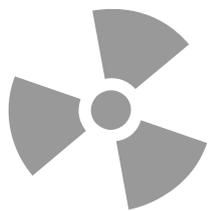
Solutions
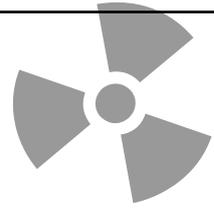
Question 3 – Hide-and-seek

Alice plays hide-and-seek with Bob. They play in a house with $n \geq 2$ rooms, numbered from 0 to $n-1$. Alice will hide, and Bob's goal is to find Alice. Bob is very predictable, so Alice knows in advance which rooms Bob will search and in what order. Bob is going to do $m \geq 1$ searches, and he can possibly search certain rooms several times. If Bob makes search in a room while Alice is there, Bob wins. Alice's goal is to make sure that Bob does not win, and to do this, she can choose the room where she hides at the start, and she can also move from one room to another between two searches of Bob. But each her movement is risky (she could be caught), so Alice needs to minimize the number of moves she has to make.

For example, imagine that the house has $n = 3$ rooms numbered as 0, 1, 2, and that Bob will make the following sequence of $m = 4$ searches:

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

One of possible solutions for Alice is to hide in the room 2 first, then to move to the room 0 before Bob searches the room 2, and finally to move to the room 1 before Bob searches the room 0. Alice will move 2 times and her position over time will be described by the sequence:

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

But Alice can do better: after leaving the room 2, she could move directly into the room 1 and stay there till the end of the game. She can move only once and her position over time would be:

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

However, it is impossible to make less than one movement. Alice cannot stay in the same room during the game, because Bob searches all the rooms at least once. Thus, for $n = 3$ and $\text{bob[]} = \{1, 2, 0, 2\}$, the minimal number of Alice's movements is 1.

Q3(a) [2 pts]	**What is Alice's minimum number of movements if $n = 2$ and Bob searches the rooms as bob[] = $\{0, 1, 0, 1\}$?**
Solution: 3 (alice[] = $\{1, 0, 1, 0\}$)	

Q3(b) [2 pts]	**What is Alice's minimum number of movements if $n = 3$ and Bob searches the rooms as bob[] = $\{0, 1, 0, 1\}$?**
Solution: 0 (alice[] = $\{2, 2, 2, 2\}$)	

Q3(c) [2 pts]	**What is Alice's minimum number of movements if $n = 3$ and Bob searches the rooms as bob[] = $\{2, 1, 2, 0\}$?**
Solution: 1 (for example, alice[] = $\{0, 0, 1, 1\}$)	

Q3(d) [2 pts]	**What is Alice's minimum number of movements if $n = 3$ and Bob searches the rooms as bob[] = $\{0, 1, 2, 0, 2, 0\}$?**
Solution: 1 (alice[] = $\{2, 2, 1, 1, 1, 1\}$)	

Q3(e) [2 pts]	**What is Alice's minimum number of movements if $n = 4$ and Bob searches the rooms as bob[] = $\{0, 1, 2, 3, 0\}$?**
Solution: 1 (par exemple, alice[] = $\{3, 3, 3, 2, 2\}$)	

Q3(f) [2 pts]	What is Alice's minimum number of movements if $n = 4$ and Bob searches the rooms as **bob[]** $= \{0, 1, 2, 3, 0, 1\}$?
Solution: 1 (alice[] $= \{3, 3, 3, 2, 2, 2\}$)	

Q3(g) [3 pts]	What is Alice's minimum number of movements if $n = 4$ and Bob searches the rooms as **bob[]** $= \{0, 1, 2, 3, 0, 1, 2\}$?
Solution: 2 (par exemple, alice[] $= \{3, 3, 3, 2, 2, 3, 3\}$)	

Q3(h) [4 pts]	In a n room house, what is the least number of searches m that Bob has to do to force Alice to move at least k times?
Solution: $k(n-1)+1$	

Everything becomes funnier with the code

Alice decides to write a program which determines in which room she should hide during each of Bob's m searches.

The program entries are:

- n : the number of rooms.
- m : the number of searches that Bob is going to make.
- bob[] (an array of length m) : the sequence of searches that Bob is going to make.

The program must calculate alice[] (an array of length m): a list of m rooms where Alice must hide to avoid being catched by Bob, while moving the least times possible.
There can be several solutions which give the minimum number of movements; if so, any of them is acceptable and Alice's program gives just one.

The program listing is on the next page.
Here are some explanations to help you understand it.

- cnt : used to count the number of different rooms that Bob is going to visit after Alice's last movement.
- t[] (an array of length n): if t[v]=**false** this means that Bob is going to search in the room v after Alice's last movement.
- The first **for** loop searches for Alice's hiding places, except the last one.
- The end of the program from a $\leftarrow$ -1 searches for Alice's last hiding place.

The listing is incomplete and we propose you to find the missing expressions.

Q3(i) [2 pts]	What is the expression (i) in the algorithm ?
Solution: 1	

Q3(j) [4 pts]	What is the expression (j) in the algorithm ?
Solution: bob[i]	

```
Input  : n, m, bob[]
Output : alice[]

alice[] ← {-1, ..., -1} (an array of length m, initialized with -1)
t[] ← {true, ..., true} (an array of length n, initialized with true)
cnt ← 0
j ← 0
for (i ← 0 to m-1 step 1)
{
    if (t[bob[i]])
    {
        cnt ← cnt + 1
        if (cnt = n)
        {
            cnt ← ...           // (i)
            while (j < i)
            {
                alice[j] ← ...  // (j)
                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 (t[v])
    {
        a ← v
    }
}
while (j < m)
{
    alice[j] ← a
    j ← j + 1
}
return alice[]
```

Solutions

Question 4 – Al Capone

The famous gangster Al Capone has decided to reorganize his network of underground bars in Chicago. He wants to group several bars in *districts*, and each district will be under the leadership of a gang leader. In order to allow his accomplices to flee in the event of police raids, he wants the following condition to be met: *two bars are in the same district if and only if there is at least one route in each direction allowing to flee from one to the other by car (possibly passing by other bars)*. A neighborhood can have any number of bars as long as the above condition is met for any pair of bars. For instance, the bars 0 and 1 et 2 will be in the same « districtr » if it is possible to drive by car from 0 to 1, from 1 to 0, from 1 to 2, from 2 to 1, from 0 to 2 and from 2 to 0.

Here are two maps of Al Capone bars in Chicago. Each bar is represented by a circle (with the bar number) and the arrows indicate the roads (one way) that can be followed to go *directly (i.e. without going past other bars)* from one bar to another by car.
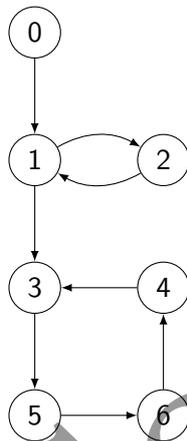
Map 1:   **Map 2:**
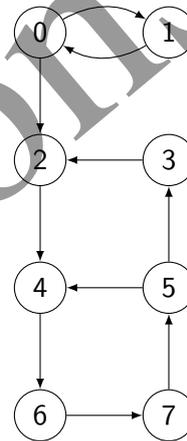
For instance, with **Map 1**, you can go (directly) by car from 3 to 5. You can also go by car from 5 to 3 but you have to pass in front of 6 and 4 (because the direct road from 5 to 3 is a forbidden direction.).

Indicate, for each of the following statements about **Map 1**, if they are true or false.

	Vrai	Faux	Statements about **Map 1**.
Q4(a) [1 pt]	☐	☒	You can go by car from 1 to 0.
Q4(b) [1 pt]	☐	☒	You can go by car from 6 to 1.
Q4(c) [1 pt]	☒	☐	You can go by car from 6 to 5.
Q4(d) [1 pt]	☒	☐	1 and 2 can be in the same district.
Q4(e) [1 pt]	☐	☒	6 and 2 can be in the same district.
Q4(f) [1 pt]	☒	☐	3, 4 and 6 can be in the same district.
Q4(g) [1 pt]	☒	☐	3, 4, 5 and 6 can be in the same district.
Q4(h) [1 pt]	☐	☒	1, 2, 3, 4, 5 and 6 can be in the same district.

Indicate, for each of the following statements about **Map 2**, if they are true or false.

	Vrai	Faux	Statements about **Map 2**.
Q4(i) [1 pt]	☒	☐	You can go by car from 1 to 0.
Q4(j) [1 pt]	☒	☐	You can go by car from 2 to 7.
Q4(k) [1 pt]	☐	☒	You can go by car from 7 to 1.
Q4(l) [1 pt]	☒	☐	2 and 5 can be in the same district.
Q4(m) [1 pt]	☒	☐	2, 3 and 4 can be in the same district..
Q4(n) [1 pt]	☒	☐	4, 5, 6 and 7 can be in the same district.
Q4(o) [1 pt]	☐	☒	4, 5, 6 and 7 can be in the same district and there exists no larger dictrict containing these 4 bars.
Q4(p) [1 pt]	☒	☐	2, 3, 4, 5, 6 and 7 can be in the same district..
Q4(q) [1 pt]	☒	☐	2, 3, 4, 5, 6 and 7 can be in the same district. and there exists no larger dictrict containing these 6 bars.

Unfortunately, Capone does not have such precise maps of his bars, and therefore sends his accomplice Joe on foot (it's more discreet) to visit the streets of Chicago. Capone explains to Joe how he should go about traveling the city: « I give you the address of a first bar. You will follow all the roads that start from this bar, respecting the one-way traffic: first the one to the west (on the left on the map), then the one to the south (on the bottom), then the one to the east (on the right of the map), then the one to the north (top) if they exist. If at the end of a road you find **a bar you've never been through** during your exploration, you repeat the same procedure from this new bar, otherwise you retrace your steps to the previous bar. When you have explored all the roads that start from a bar, you will also retrace your steps. »

As Joe is not very smart, Capone gives him the algorithm `Explore` (see listing below) to follow by the book. This is in the form of a function, which is called with the first bar given by Capone as a parameter. This function fills an array `order` of which the index is the number of a bar (from 0 to N-1), and which gives the order (from 1 to N) in which the bars are visited.
For example, if `order[i]=1`, it means that bar number `i` is the first one given by Capone, etc

For example, on **Map 1**, and assuming that the bar 0 is the first, the algorithm `Explore` will first visit the bar 0, then the 1 (only bar accessible from 0), then the 3 (we visit first the bar in the south before visiting the one in the east), then 5, 6, 4 and finally 2 (after having retraced its steps).

Q4(r) [3 pts]	Give the contents of the array **order[]** after executing the function **Explore** on Map 1, assuming that the first bar is bar 0.
	Solution: `[1,2,7,3,6,4,5]`

Q4(s) [3 pts]	Give the order in which the bars are visited by the algorithm **Explore** on Map 2 starting from bar 0.
	Solution: 0, 2, 4, 6, 7, 5, 3, 1.

Q4(t) [3 pts]	Give the contents of the array **order[]** after executing the function **Explore** on Map 2, assuming that the first bar is bar 0.
	Solution: `[1,8,2,7,3,6,4,5]`

```
Input: N, the number of bars
       I, the number of the first bar (between 0 and N-1)

order[] ← {-1, ..., -1} (an array with length N, initialised with -1 everywhere)
cpt ← 1 (a counter initialised to 1)

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

    for (d = west, south, east, north)
    {
      if (there is a road heading d starting from X)
      {
        Y ← the bar at the end of the road heading d starting from X
        Explore(Y)
      }
    }
  }
}

Explore(I) (fills the array order[] for an exploration starting from bar number I)
```

Now that Joe seems to have understood this first algorithm, Capone enriches the `Explore` function to cut the map into districts. First, he asks Joe to take a notebook with him, which will allow him to keep a list of the bars he has visited: every time he arrives at a new bar, he adds it to the bottom of his list. For example, on **Map 1**, when Joe arrives at bar number 5 (when entering the function call `Explore(5)`), his list contains: 0, 1, 3.

Q4(u) [3 pts]	What is the content of Joe's list when he arrives at bar 3 on Map 2, in other words, at the time of calling **Explore(3)** ?
	Solution: 0, 2, 4, 6, 7, 5.

Capone then adds a second array `low[]` to the `Explore` function. This allows Joe to remember, for each bar, what is the *lowest order number* (given in the array `order`) which can be accessed from this bar. Capone then realized that by looking for the X bars such as `order[X]=low[X]`, we can identify the districts on the basis of the list maintained by Joe. All of this is explained in detail in the new algorithm (see listing below).

En exécutant `Explore(I)`, Joe va non seulement remplir les tableaux `order[]` et `low[]`, mais il va aussi délimiter les quartiers dans lesquels chaque bar est (pour rappel) accessible en voiture depuis n'importe quel autre bar du quartier.

By running `Explore(I)`, Joe will not only fill in the arrays `order[]` and `low[]`, but it will also determine the districts in which each bar is (as a reminder) accessible by car from any other bar in the district.

Q4(v) [3 pts]	What are the contents of the array **low[]** after executing the algorithm on Map 1 (with initial bar 0) ?
	Solution: [1,2,2,3,3,3,3]

```
Input: I, the first bar
        N, the number of bars

order[] ← {-1, ..., -1} (an array of length N, initialised with -1s)
low[] ← {-1, ..., -1} (an array of length N, initialised with -1s)
cpt ← 1 (a counter initialised to 1)

Explore2(bar X)
{
  order[X] ← cpt
  low[X] ← cpt
  Add X at the bottom of the list
  cpt ← cpt+1
  for (d = west, south, east, north)
  {
    if (there is a road heading d starting from X)
    {
      Y ← the bar at the end of the road heading d starting from X
      if (order[Y] = -1)
      {
        Explore(Y)
        low[X] ← min(low[X],low[Y])
      }
      else if (Y is in the list)
      {
        low[X] ← min(low[X],order[Y])
      }
    }
  }

  if (low[X] = order[X])
  {
    Create a new district containing all the bars that appear
    on the list starting from X (included), and delete them from the list.
  }
}

Explore(I)
```

Q4(w) [3 pts]	On Map 2 (initial bar 0), what is the value of `low[5]` at the moment `Explore(3)` is called ?
Solution: 3

Q4(x) [3 pts]	On Map 2 (initial bar 0), what is the content of Joe's list when `Explore(1)` is called ?
Solution: 0

Q4(y) [3 pts]	**On Map 1 (initial bar** 0)**, which are the districts obtained by Capone's algorithm ? Indicate each district as a set of bar numbers,** **for instance** $\{0, 1, 4, 5\}$**,** $\{2, 3, 6\}$**.**
	Solution: $\{0\}, \{1, 2\}, \{3, 4, 5, 6\}$

Q4(z) [3 pts]	**On Map 2 (initial bar** 0)**, which are the districts obtained by Capone's algorithm ? Indicate each district as a set of bar numbers.**
	Solution: $\{0, 1\}, \{2, 3, 4, 5, 6, 7\}$
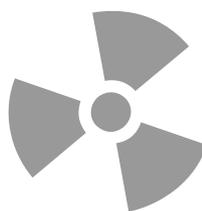
Solutions
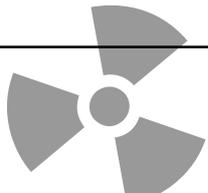
Question 5 – Administrative wars

In a very-very distant world, several countries compete to dominate their own planet. Their wars are different from ours and are done without shedding blood. The countries send their administrative directors to wild verbal debates (in reality, as we will see, it is the power of the administrative services that counts). At this point, it is useful to explain what an **entity** is. A given country can be divided into provinces, each of these provinces can be divided into sub-provinces, each of these sub-provinces can be divided into sub-sub-provinces, and so on. Each country, province, sub-province, sub-sub-province, sub-sub-sub-province, . . . is an **entity**.

Let us look at the example of the country of Heldor :
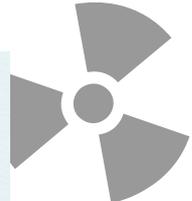

Fig 1: Geographic map of Heldor
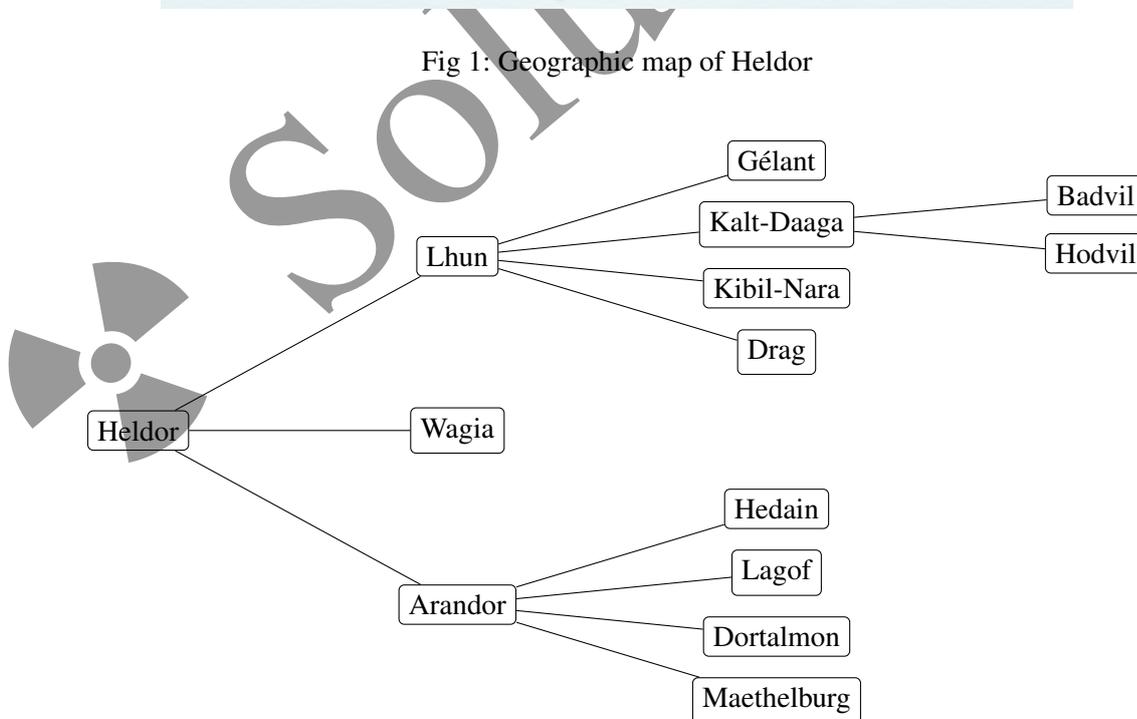
Fig 2: Heldor's **entities** administrative map

Each **entity** has its own administration which has its own power. Obviously, administrative complexity means that the more an entity manages entities below them, the less efficient it is (bureaucracy is a problem in all worlds).
For example, Lhun manages 7 entities: Lhun itself and 6 entities below it. One can say that Lhun has 7 **sub-entities** (because we have to count the central administration of Lhun as a particular sub-entity !).

On the contrary, the smaller an **entity** is compared to the country, the less it has money and the less its administration is powerful.

More precisely, if **X** is a n-th level **entity** (0 for a country, 1 for a province, 2 for a sub-province, 3 for a sub-sub-province, ...) and if s is the number of sub-entities of **X** then p, the **administrative power** of **X**, is $\boxed{p = 150 - 10n - 2s^2}$.

- Example for Badvil sub-sub-province: $n = 3$ and $s = 1$, thus $p = 150 - 10 \cdot 3 - 2 \cdot 1^2 = 150 - 30 - 2 = 118$.
- Example for Lhun province: $n = 1$ and $s = 7$, thus $p = 150 - 10 \cdot 1 - 2 \cdot 7^2 = 150 - 10 - 98 = 42$.
- Example for Heldor country: $n = 0$ and $s = 14$, thus $p = 150 - 10 \cdot 0 - 2 \cdot 14^2 = 150 - 392 = -242$.

Q5(a) [7 pts]	**Complete the administrative map of Heldor by noting the powers of the entities.**
	(If necessary, reply here, on the draft, before copying your result onto the answer sheet.)
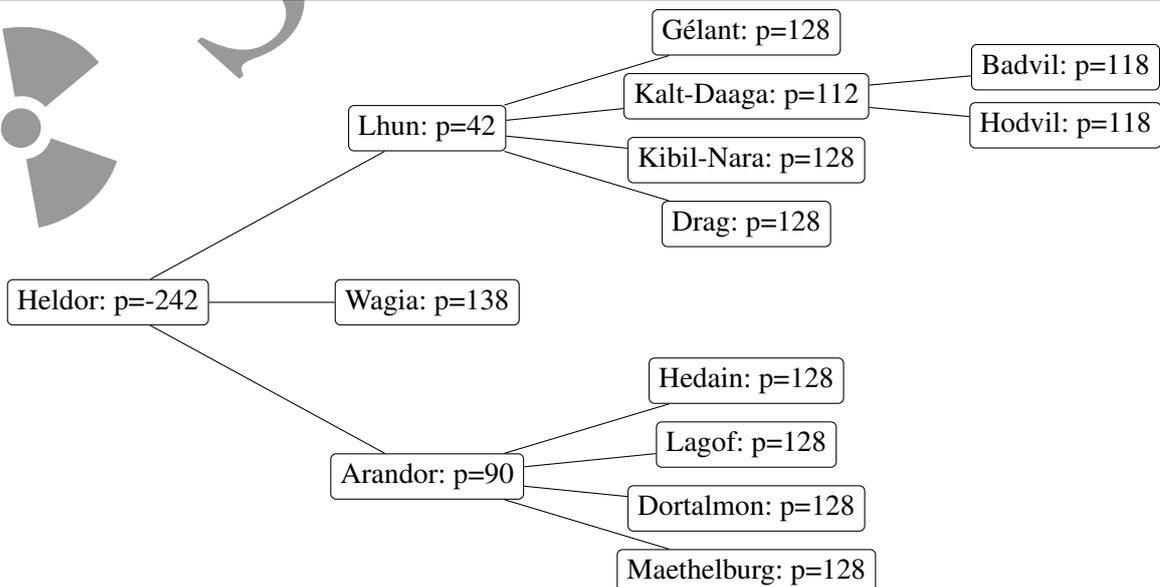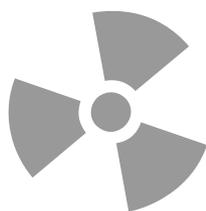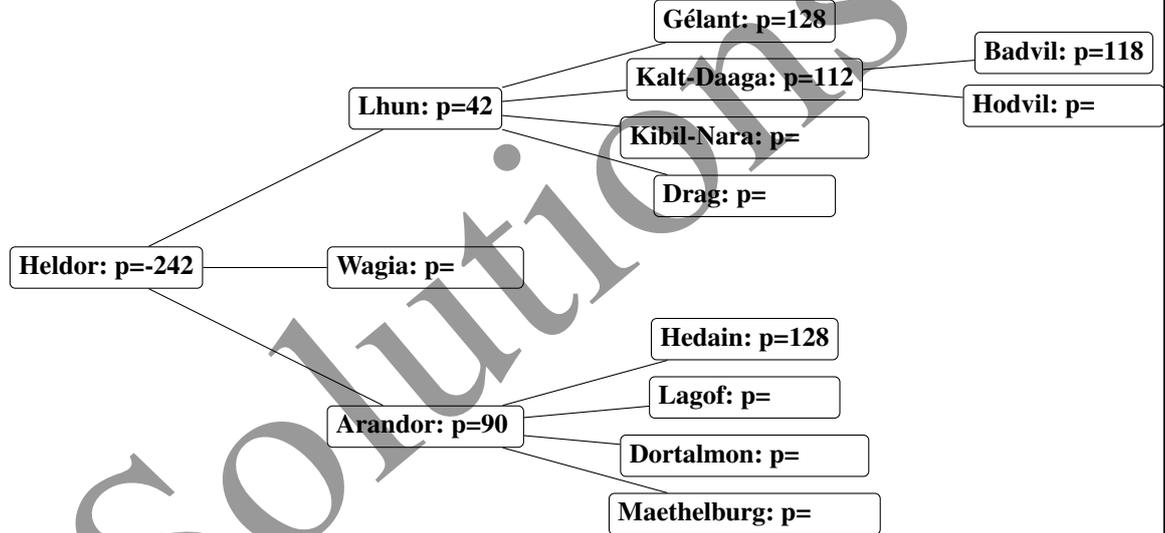
Solution:

You conclude that Heldor is submerged by the administrative work and that Heldor's power is negative, which means that Heldor consumes resources to keep his administration afloat !

Likewise, a sub-sub-sub . . . province could have a negative or zero power, even if it has no other sub-entity except itself. But how many "sub's" is needed?

Q5(b) [2 pts]	How many "sub's" should we put before the word "province" to be sure that its power is zero or negative (≤ 0) ?
Solution: 14 (because $150 - 10n - 2 \leq 0 \iff n \geq 14.8$ and there is 14 "sub's" at the level 15)	

What really matters is the **total power**. The **total power** of an entity **X** is obtained by adding the powers of all the sub-entities of **X** (without forgetting **X** itself). For example, the Badvil's **total power** is equal to its power (118), since it has no other sub-entity than itself.

On the other hand, the **total power** of Lhun is a sum of 7 powers (help: its value is 774).

Q5(c) [1 pt]	What is the total power of Hodvil ?
Solution: 118	

Q5(d) [1 pt]	What is the total power of Wagia ?
Solution: 138	

Q5(e) [2 pts]	What is the total power of Arandor ?
Solution: $90 + 4 \cdot 128 = 602$	

Q5(f) [2 pts]	What is the total power of Heldor ?
Solution: $-242 + 774 + 138 + 602 = 1272$	

When two countries stand again each other, the one with the largest **total power** wins. In the event of equal power, the administrations become entangled and the countries disappear in a bureaucracy abyss. Let be a function TOTAL_POWER (X, n) which calculates the **total power** of an entity X of level n.

Q5(g) [2 pts]	Find an expression indicating if the country **A** wins a withstand against the country **B**. Remember that countries have a level **n=0** and that in case of equality there is no winner.
Solution: TOTAL_POWER(A, 0)>TOTAL_POWER(B, 0)	

Each entity at level *n* knows the list of its sub-entities at level $n + 1$ which are called its **children**. This strange name comes from the theory of graphs, used to draw administrative maps. On the administrative map, a separate line links each entity to each of its children (see Fig. 2). The children of a country are its provinces, the children of a province are its sub-provinces, . . .

To calculate the administrative power of an entity X, it is necessary to know its number of sub-entities (including X itself). Here is the incomplete pseudo-code (one line is missing) of the function CALCULATE_S(X) which does this.

```
function CALCULATE_S(X) {
    nb_entities ← 1
    for every E child of X  {
```

```
        ... //(i)
    }
    return nb_entities
}
```

Note that the line "**for** every E child of X" starts a loop which will be executed once for each child of X, and that this child will be designated by the variable E in the loop. For example, if X=Heldor, the loop will be executed 3 times, once with E=Lhun, once with E=Wagia, once with E=Arandor. If X has no children, the loop is not executed (it is executed 0 times since there is 0 children).

Q5(h) [2 pts]		What is the line **(i)** in the pseudo-code of the function **CALCULATE_S()** ?
	☐	nb_entities ← CALCULATE_S(E)+1
	☐	nb_entities ← nb_entities+1
	☒	nb_entities ← nb_entities+CALCULATE_S(E)
	☐	nb_entities ← CALCULATE_S(E)
	☐	nb_entities ← nb_entities+CALCULATE_S(E)+1

Solutions

Complete the pseudo-code of the function TOTAL_POWER(X, n) which allows (as already mentioned) to calculate the **total power** of an entity X of level n.

```
function TOTAL_POWER(X, n) {
    power_of_X ← ...              //(i)
    power_of_children ← 0
    for every E child of X {
        power_of_children ← power_of_children + ...     //(j)
    }
    return ...                        //(k)
}
```

Hints: you can and should use CALCULATE_S(X). Consider that this function is correctly implemented, even if you did not answer the previous question. To calculate the square of a number (r), multiply it by itself (r*r) or put it to the power of 2 (r^2).

Q5(i) [2 pts]	What is the expression **(i)** in the pseudo-code of the function **TOTAL_POWER(X, n)** ?
Solution: 150-10*n-2*CALCULATE_S(X)*CALCULATE_S(X)	

Q5(j) [2 pts]	What is the expression **(j)** in the pseudo-code of the function **TOTAL_POWER(X, n)** ?
Solution: TOTAL_POWER(E, n+1)	

| Q5(k) [2 pts]	What is the expression **(k)** in the pseudo-code of the function **TOTAL_POWER(X, n)** ?
Solution: power_of_X + power_of_children	

When two countries go to war, the losing country is annexed by the winning country and becomes one of its provinces. This obviously changes the administrative power of the winning country ! Let's take an example. Here are the administrative maps of three countries (we used postal codes instead of names, it is shorter).
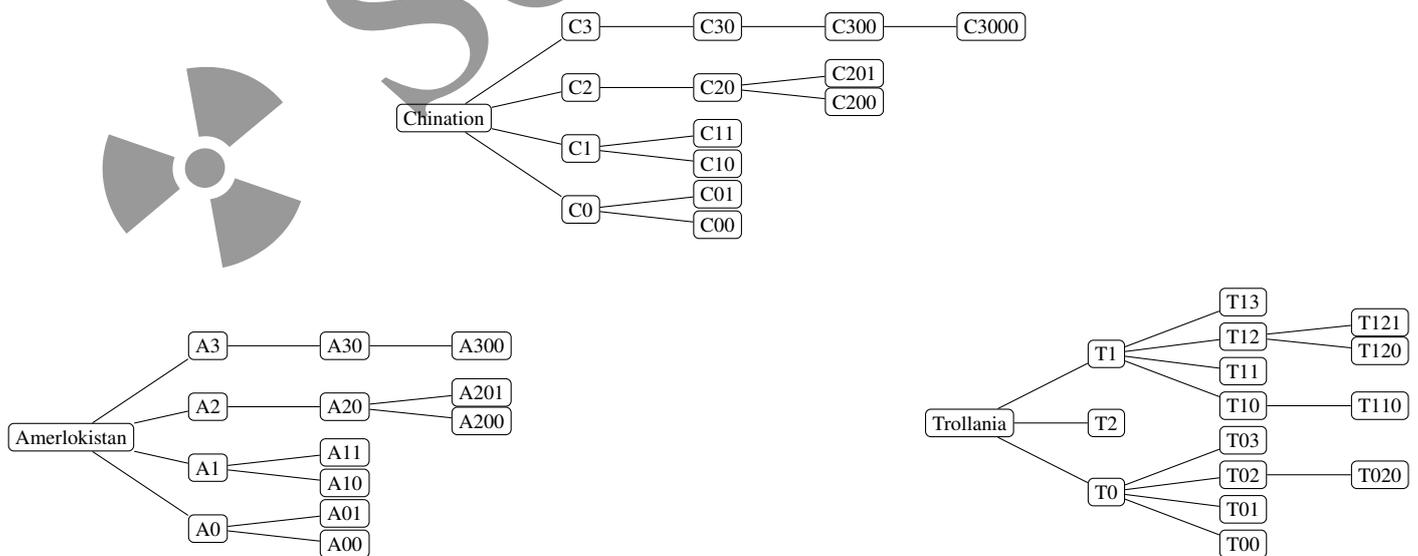
Our spies have calculated the **total powers** of these three countries for you :

Chination: **1352**	Amerlokistan: **1332**	Trollania: **1324**

In the case of war between Amerlokistan and Trollania, Amerlokistan would annex Trollania.
Here is what would happen to the administrative map of Amerlokistan after its victory :
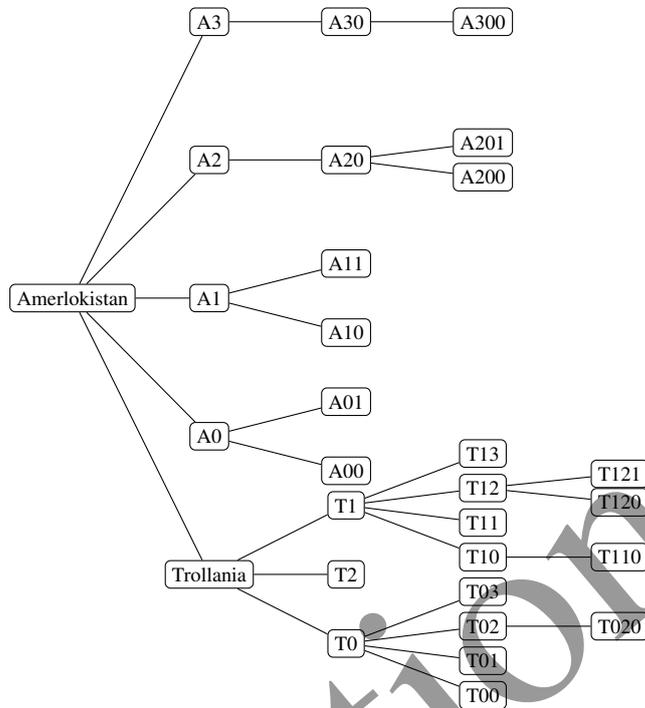
The new **total power** of Amerlokistan after the war would be equal to **1088**.

There is a formula for calculating the **total power** of a war winner.

Q5(l) [5 pts]		Let **X1** be a country of **total power t1** having **s1** sub-entities (**X1** included). Let **X2** be a country of **total power t2** having **s2** sub-entities (**X2** included). If **t1**>**t2** which expression gives the new total power of the winner of the war between **X1** and **X2** ?
	☐	`t1 + t2`
	☐	`t1 + t2 - 10*s2`
	☐	`t1 + t2 - 2*s2*s2`
	☐	`t1 + t2 + 2*s1*s1 - 2*(s1+s2)*(s1+s2)`
	☐	`t1 + t2 + 2*s2*s2 - 2*(s1+s2)*(s1+s2)`
	☒	`t1 + t2 - 10*s2 + 2*s1*s1 - 2*(s1+s2)*(s1+s2)`
	☐	`t1 + t2 - 10*s2 + 2*s2*s2 - 2*(s1+s2)*(s1+s2)`

Doing all these calculation manually is awful, isn't it ? It is time to implement a procedure.

All countries are now saved in a list: `list_country`. This list supports three operations :

- `list_country.size()` that returns the number of the countries in the list,
- `list_country.greatest_admin()` that returns the country of the list which has the greatest total power **and** removes it from the list (if several countries are equal with the greatest total power, one of them is chosen "randomly").
- `list_country.annex(X)` that adds the country `X` into the list.

In addition, the variables "contry" contain a description of the administrative map of the country. If `A` and `B` are two variables "contry", then the instruction `A.annex_province(B)` modify the variable `A` adding a new province, an identical copy of the country `B`.

We propose you to complete the `WARS` function which simulates successive wars between the countries in the list, until there is only one (or no) country left in the list.

```
function WARS() {
    while (...)                        //(e1)
    {
        A ← ...                //(e2)
        B ← ...                //(e2)
        p_tot_A ← TOTAL_POWER(A, 0)
        p_tot_B ← TOTAL_POWER(B, 0)
        if (p_tot_A > p_tot_B)
        {
            A.annex_province(B)
            ...                //(e3)
        }
    }
}
```

Q5(m) [2 pts]	What is the expression **(e1)** in the function `WARS()` ?
Solution: `list_country.size()>1`	

Q5(n) [2 pts]	What is the expression **(e2)** which appears twice in the function `WARS()` ?
Solution: `list_country.greatest_admin()`	

Q5(o) [2 pts]	What is the expression **(e3)** in the function `WARS()` ?
Solution: `list_country.annex(A)`	