be-OI 2024 **Final - SENIOR** Saturday, March 23, 2024	**Fill in this box in CAPITAL LETTERS please** FIRST NAME : .... LAST NAME : .... SCHOOL :	0 **Reserved**

Finals of the Belgian Informatics Olympiad 2024 (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 **cadet**.
   - 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, last name and school **on this page only**.
3. Write **your answers** on the provided answer sheets. Write **clearly and legibly** with a blue or black **pen or bic**.
4. Use a pencil and eraser when working in draft form on the question sheets.
5. You may only have writing materials with you. Calculator, GSM, smartphone ... are **forbidden**.
6. You can always request extra scratch paper from the invigilator.
7. 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.
8. 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.
9. 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:

©2024 Olympiade Belge d'Informatique (beOI) ASBL
This work is made available under the terms of the Creative Commons Attribution 2.0 Belgium License

Pseudo-code checklist

Data is stored in variables. We change the value of a variable using $\leftarrow$. 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 ← 2007
age ← 2024 − 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*. Look at 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 – Tests

You are requested to fill the cells of an array by computing the truth value of a test that ranges on the row and column numbers of these cells. We assume that variable `i` contains a row number and variable `j` contains a column number. If the test evaluates to true for a pair of values `i` and `j`, you must fill the cell which is in row `i` and column `j`.

Here is an example condition: `(i==2)` **`or`** `(j>3)`.
This condition is true if the row number is equal to 2 or if the column number is larger than 3.
The row numbers are displayed on the left and the column numbers are displayed on the top of the array.
The cells must thus be filled as follows.

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

The tests rely onthe comparison operators from the examples hereunder.

`i==1`	True if `i` is equal to 1.
`j>3`	True if `j` is greater than 3.
`j>=4`	True if `j` is greater than or equal to 4.
`i+j<5`	True if `i+j` is less than 5.
`i<=j+2`	True if `i` is less than or equal to `j+2`.
`(i==3)` **`or`** `(j==2)`	True if at least one of the two conditions is true.
`(i<2)` **`and`** `(j>3)`	True if both conditions are true.

Some questions will also make use of the notations found in the following examples:

`max``(14,18)`	Maximum function. Returns the greatest value, hence 18.
`min``(i,j)`	Minimum function. Returns the lowest value.
`11/4`	Operator `/`, quotient of integer division. Returns 2, since 11 = **2** · 4 + 3
`11%4`	Operator `%`, remainder of the integer division. Returns 3, since 11 = 2 · 4 + **3**

Solutions are displayed hereunder.

Q1(a) /2 **Fill in the cells that verify the condition `(i==j) or (i+j==5)`**

	0	**1**	**2**	**3**	**4**	**5**
0						
1						
2						
3						
4						
5						

Q1(b) /2 **Fill in the cells that verify the condition** `min(i,j)<=1 or max(i,j)>=4`

	0	1	2	3	4	5
0	■	■	■	■	■	■
1	■	■	■	■	■	■
2	■	■			■	■
3	■	■			■	■
4	■	■	■	■	■	■
5	■	■	■	■	■	■

Q1(c) /4 **Fill in the cells that verify the condition** `max(i-j,j-i)>=3`

	0	1	2	3	4	5
0				■	■	■
1					■	■
2						■
3	■					
4	■	■				
5	■	■	■			

Q1(d) /4 **Fill in the cells that verify the condition** `min(i,j)%2==0`

	0	1	2	3	4	5
0	■	■	■	■	■	■
1	■					
2	■		■	■	■	■
3	■		■			
4	■		■		■	■
5	■		■		■	

Q1(e) /4 **Fill in the cells that verify the condition** `(i/2+j/2)%2==0`

	0	1	2	3	4	5
0	■	■			■	■
1	■	■			■	■
2			■	■		
3			■	■		
4	■	■			■	■
5	■	■			■	■

Question 2 – Paths

How many paths ?

The pictures hereunder show rooms with square floor tiles (white cells) and obstacles (grey cells). A robot moves from **A** in the upper left corner and must reach **B** in the lower right corner. At each step, it can only move by one cell down or right. It can never move up nor left. The robot must remain on white cells, it can never move through a grey cell. In each case, how many different paths can the robot take to move from **A** to **B**?

Q2(a) /1 **How many different paths can the robot take to move from A to B?**

Solution: 4

Q2(b) /1 **How many different paths can the robot take to move from A to B?**

Solution: 11

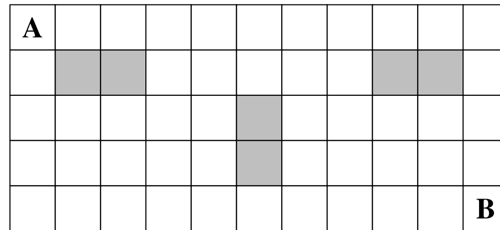
Q2(c) /2 **How many different paths can the robot take to move from A to B?**

Solution: 15

Q2(d) /2 **How many different paths can the robot take to move from A to B?**

Solution: 43

Q2(e) /3 **How many different paths can the robot take to move from A to B?**

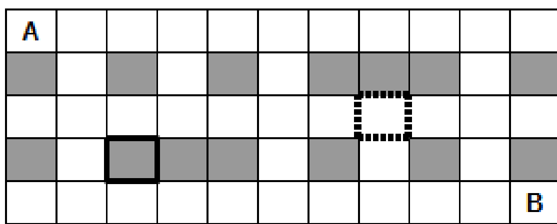


Solution: 131

A program to count the paths

There are different ways to count the paths from **A** to **B**. A DP algorithm (Dynamic Programming) can do it in all cases and becomes necessary in the most difficult cases.

The incomplete listing of a DP algorithm is given hereunder. The room is encoded as an array `t[][]` with a **0** for each white cell, and a **1** for each grey cell.



Room

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

Correponding array `t[][]`

In the example above, the array `t[][]` has 5 rows and 11 columns. The row numbers are displayed on the left of the array and the column numbers are displayed on the top of the array. Each cell is represented by its row number and its column number. The cell with a thick border is grey, so the corresponding element `t[3][2]` is equal to **1**. The cell with dotted border is white, so the corresponding element `t[2][7]` is equal to **0**. Cells **A** and **B** are of course white and thus represented by **0**'s in `t[][]`.

The DP algorithm computes, for each cell, how many different paths exist between **A** and this cell. The number of paths to reach **A** is equal to **1**, since we are on this cell at the beginning and we cannot come back to this cell. The number of paths is equal to **0** for the grey cells (since we can never reach them). The number of paths to reach a white cell can be computed from the number of paths to reach its top and left neighbours (this is the central property to obtain a DP program).

The numbers of paths are stored in an array `dp[][]` with the same number of rows and columns as the array `t[][]`. At the beginning of the program, the array `dp[][]` is filled with **0**'s. The program begins with cell **A** and progresses in the natural reading direction (row by row, from left to right in each row). With the example room, after the execution of the algorithm, `dp[3][2]` will be equal to **0** because no path can reach a grey cell and `dp[2][7]` will be equal to the number of different paths from the cell **A** to the cell with dotted border.

The program receives: the number of rows in variable `ni`, the number of columns in variable `nj`, the array `t[][]` filled to represent the room and the array `dp[][]` filled with **0**'s.
The program must return the number of different paths from **A** to **B**.

Q2(f) /6	**Fill in the ____ in the DP program.** **Marks between 0 et 6. You will lose 1 point for each wrong or missing answer.**
Solution : Solutions are displayed on a grey background.	

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

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

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

Unreachable cells

Some white cells are unreachable. This means that the robot cannot reach them from **A** because of obstacles (grey cells). We assume that the DP program has been executed and that we have: the number of rows in variable `ni`, the number of columns in variable `nj`, the array `t[][]` filled to represent the room and the array `dp[][]` filled by the DP program. You are requested to fill in the blanks in the NoPath program hereunder. It must return the number of unreachable white cells.

Q2(g) /5	**Fill in the ____ in the NoPath program that counts the unreachable white cells.** **Marks between 0 et 5. You will lose 1 point for each wrong or missing answer.**
Solution : Solutions are displayed on a grey background.	

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

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

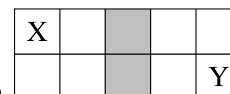
Intermediary cells

We now want to count the number of paths going through an intermediary cell **X**. The robot still starts from **A** in the upper left corner and must reach **B** in the lower right corner, but it must pass through **X**.

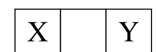
Room 1: We know that there are 21 different paths between **A** and **X** and 9 different paths between **X** and **B**.

Q2(h) /1	**In room 1, how many paths are there from A to B, passing through X?**
	Solution : $21 \cdot 9 = 189$

In the next questions, there are two intermediary cells **X** and **Y**.
In all cases, we have an incomplete map of the room that contains **X** and **Y** and we know:
N(A,X) the number of different paths between **A** and **X**, **N(X,B)** the number of different paths between **X** and **B**,
N(A,Y) the number of different paths between **A** and **Y**, **N(Y,B)** the number of different paths between **Y** and **B**.

Room 2: We know that N(A,X)=5, N(X,B)=12, N(A,Y)=8, N(Y,B)=10 and we have the partial map 

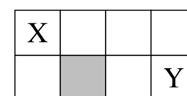
Q2(i) /2	**In room 2, how many paths are there from A to B, passing through either X or Y?**
	Solution : $(5 \cdot 12) + (8 \cdot 10) = 140$

Room 3: We know that N(A,X)=6, N(X,B)=16, N(A,Y)=10, N(Y,B)=8 and we have the partial map 

.

Q2(j) /1	**In room 3, how many paths are there from A to B, passing through both X and Y?**
	Solution : $6 \cdot 8 = 48$

Q2(k) /2	**In room 3, how many paths are there from A to B, passing through X or Y (or both) ?**
	Solution : $(6 \cdot 16) + (10 \cdot 8) - (6 \cdot 8) = 128$

Room 4: We know that N(A,X)=6, N(X,B)=11, N(A,Y)=18, N(Y,B)=3 and we have the partial map 

.

Q2(l) /1	**In room 4, how many paths are there from A to B, passing through both X and Y?**
	Solution : $6 \cdot 2 \cdot 3 = 36$

In room 4, we also know that there are 18 paths from **A** to **B** that pass through neither **X** nor **Y**.

Q2(m) /2	**In room 4, how many paths from A to B are there in total?**
	Solution : $(6 \cdot 11) + (18 \cdot 3) - (6 \cdot 2 \cdot 3) + 18 = 102$

General case: We know N(A,X), N(X,B), N(A,Y), N(Y,B) and **N(X,Y)** which is the number of different paths from **X** to **Y** (and there is no path from **Y** to **X**).

Q2(n) /1	**Which expression gives the number of paths from A to B, passing through X and Y?**
	Solution : N(A,X)·N(X,Y)·N(Y,B)

Q2(o) /2	**Which expression gives the number of paths from A to B, passing through X or Y (or both)?**
	Solution : N(A,X)·N(X,B) + N(A,Y)·N(Y,B) − N(A,X)·N(X,Y)·N(Y,B)

Question 3 – Loops

Let us consider the program **LoopA** hereunder. It contains three nested **for** loops and one test with 3 conditions that must all be true to execute a print instruction.

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

LoopA prints all triplets of integer numbers larger than zero and whose sum is equal to 100.
In each triplet, the three numbers are printed in increasing order.
For instance, **LoopA** prints triplet `(20,30,50)` but not triplet `(50,20,30)`.

Q3(a) /1	What is the first triplet to be printed?
Solution : $(1, 2, 97)$	

Q3(b) /1	What is the tenth triplet to be printed?
Solution : $(1, 11, 88)$	

Q3(c) /1	What is the last triplet to be printed?
Solution : $(32, 33, 35)$	

Q3(d) /1	How many times is the test evaluated in LoopA ?
Solution : $100 \cdot 100 \cdot 100 = 10,000,00$ times	

Program **LoopB** prints exactly the same thing as **LoopA**.
But it relies on a simpler test: conditions `i<j` and `j<k` are not useful anymore.

Q3(e) /3	Complete the ____ in LoopB so that it prints the same thing as LoopA. Marks between 0 and 3. You will lose 1 mark for each mistake or missing answer.
Solution : Correct answers are displayed on a grey background hereunder.	

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

Q3(f) /1	**Is the test in LoopB evaluated more often than, less often than or the same number of times as the test in LoopA? ☐ More often ☒ Less often ☐ Same number of times** **The test is evaluated 161,700 times in LoopB.**

Q3(g) /1	**Is the `print` instruction in LoopB executed more often than, less often than or the same number of times as in LoopA? ☐ More often ☐ Less often ☒ Same number of times** **The two programs print exactly the same thing!**

Program **LoopC** prints exactly the same thing as **LoopA** and **LoopB**.
The test has been modified again and we have kept only 2 **`for`** loops.

Q3(h) /3	**Complete the ____ in LoopC so that it prints the same thing asLoopA and LoopB.** **Marks between 0 and 3. You will lose 1 mark for each mistake or missing answer.**
Solution : Correct answers are displayed on a grey background hereunder.	

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

Q3(i) /1	**Is the test in LoopC evaluated more often than, less often than or the same number of times as the test in LoopB? ☐ More often ☒ Less often ☐ Same number of times** **The test is evaluated 4,950 times in LoopC.**

Let us try to be more precise.

Q3(j) /2	**How many times is the test evaluated in LoopC ?**
Solution : 4,950	

Here is finally program **LoopD**. It has only 2 loops and no test.

Q3(k) /5	**Complete the ____ in LoopD so that it still prints the same thing.** **The first `for` loop must be executed as few times as possible.** **Marks between 0 and 5. You will lose 1 mark for each mistake or missing answer.**
Solution : Correct answers are displayed on a grey background hereunder.	

```
for (i ← 1 to 32 step 1) {
   for (j ← i+1 to (99-i)/2 step 1) {
      k = 100-i-j
      print(i,j,k)
   }
}
```

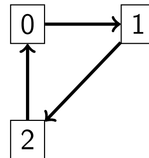
Be careful, the next question is hard. Do not try to solve it before solving all the rest !

Q3(l) /2	**How many different triplets are printed by these programs?**
Solution : 784	

Question 4 – Teleportations

Professor Spock has invented a series of *teleportation networks*. Such a network is made of n teleportation pods, numbered from 0 to $n-1$. Each pod allows one to teleport to exactly one **other** pod (since they are still prototypes, each pod allows teleportation to only one fixed destination for now). In order to ensure safety of the system, Spock wants to check that the repeated use of those teleportation devices is harmless. Can you assist him?

The **first network** developed by the professor has only 3 pods.



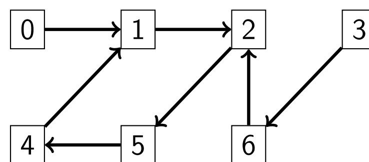
The professor tests his network by teleporting several times in a row an object which is initially in pod number 0. The object reaches pod number 1 after one teleportation, reaches pod number 2 after two teleportations, comes back to pod number 0 after three teleportations, and so on.

Q4(a) /1	Starting in pod 0, which pod is reached by the object after 10 teleportations?
Solution : 1	

Q4(b) /2	Starting in pod 0, which pod is reached by the object after 50 teleportations?
Solution : 2	

Q4(c) /2	Starting in pod 0, which pod is reached by the object after 1000 teleportations?
Solution : 1	

Since everything went fine during the first test, we can now test the **second network**, with 7 pods.

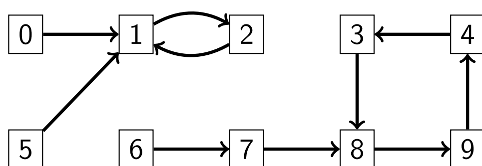


Q4(d) /2	Starting in pod 4, which pod is reached after 50 teleportations?
Solution : 2	

Q4(e) /2	Starting in pod 3, which pod is reached after 1000 teleportations?
Solution : 4	

One can now see that, by repeatedly applying a large number of teleportations, one always ends up in the same pods. Thus, starting from 0 in the previous network, one visits successively pods 0, 1, 2, 5, 4, 1, 2, 5, 4, 1, 2, 5, 4,$\cdots$ We say that a pod that allows one to come back to itself after a certain number of teleportations is a pod that *belongs to a cycle*. A *cycle* is a list of pods, where each pod leads to the next one and where the last one leads back to the first one. The length of a cycle is the number of pods that it contains. The **second network** above thus contains a cycle of lenght 4 made of pods 1, 2, 5, 4.

The **third network** hereunder contains 10 pods and 2 cycles, one of length 2, the other of length 4.



The professor has proven that, in all *teleportation networks*, where each pod allows one to teleport to exactly one other pod, we will always reach a cycle after a certain number of teleportations, no matter what the starting pod is.

Q4(f) /3	**In a network with `n` pods, starting from any pod, after how many teleportations are we certain to reach a cycle?**
Solution : `n-2`	

It is useful to compute automatically the cycle that one reaches from a given pod.
A teleportation network can be represented by an array `T[]` which gives, for each pod, the pod to which the teleportation occurs.
Thus, the array corresponding to the **third network** made of `n=10` pods is `T=[1, 2, 1, 8, 3, 1, 7, 8, 9, 4]`.
The length 2 cycle is witnessed by `T[1]=2` and `T[2]=1`.
The length 4 cycle is witnessed by `T[3]=8`, `T[8]=9`, `T[9]=4` and `T[4]=3`.
Pods number 0, 1, 2 et 5 allow one to reach the cycle of length 2.
The other pods allow one to reach the cycle of length 4 (after 2 teleportations when starting from pod 6).

You have to complete **Program A** hereunder. It takes as input an array `T[]` that represents the network to study, the number `n` of pods and the number `c` of the starting pod.
It generates as output an array of Boolean values `cyc[]` of length `n`, such that `cyc[i]` is equal to **`true`** if pod number `i` belongs to the cycle that is reachable from pod `c`, **`false`** otherwise.
Internally, the program uses an array `check[]` of `n` Boolean values.
You can use the logical operator **`not`** (**`not true`** is equal to **`false`** and **`not false`** is equal to **`true`**).

Q4(g) /5	**Fill in the ____ in program A.** **Marks between 0 and 5. You will lose 1 mark for each mistake or missing answer.**
Solution : Correct answers are displayed on a grey background hereunder.	

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

It is also important to know which pods are in the cycles of the network. **Program B** must fill in the Boolean array `cy[]` in such a way that `cy[i]` will be equal to **`true`** if pod number `i` belongs to a cycle, and to **`false`** otherwise. Internally, the program uses an array `pos[]` of `n` integer values.

Q4(h) /4	**Fill in the ____ in program B.** **Marks between 0 and 4. You will lose 2 marks for each mistake or missing answer.**
Solution : Correct answers are displayed on a grey background hereunder.	

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

The lengths of the cycles matter as well. **Program C** must fill in the array `lency[]` so that `lency[i]` will be equal to 0 if pod number `i` does not belong to a cycle and to the length of the cycle it belongs to otherwise. **Program C** relies on the array `cy[]` previously computed by **Program B**.

Q4(i) /5	**Fill in the ____ in program C.** **Marks between 0 and 5. You will lose 1 mark for each mistake or missing answer.**
Solution : Correct answers are displayed on a grey background hereunder.	

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

We know that all pods lead to a cycle by making a sufficient amount of teleportations.
Program D fills in the arrays `beforecy[]` and `firstcy[]`:

- `beforecy[i]` must contain the number of teleportations to be made from pod `i` to reach a cycle,
- `firstcy[i]` must contain the number of the first pod that belongs to a cycle that one meets starting from pod `i`.

Program D relies on the array `cy[]` previously computed by **Program B**.

Q4(j) /5	**Fill in the ____ in program D.** **Marks between 0 and 5. You will lose 1 mark for each mistake or missing answer.**
Solution : Correct answers are displayed on a grey background hereunder.	

```
Input  : n, T[], cy[]    Output : beforecy[], firstcy[]
for (i ← 0 to n-1 step 1) {
   len ← 0
   pos ← i
   while(not cy[pos]) {
      len ← len + 1
      pos ← T[pos]
   }
   beforecy[i] ← len
   firstcy[i] ← pos
}
return beforecy[], firstcy[]
```

Since you have arrays `lency[]`,`beforecy[]` and `firstcy[]` at your disposal, you can now complete **Program E**, which returns the number of the destination pod if one teleports `k` times from pod number `c`.
You will score 4 marks if your program is correct and the loop always executes at most `n` times.
You will score 2 marks if your program is correct but the loop might execute more than `n` times.

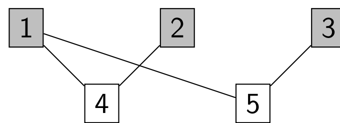
Q4(k) /4	**Fill in the ____ in program E (0, 2 or 4 marks).**
Solution : Correct answers are displayed on a grey background hereunder.	

```
Input  : n, T[], c, k, lency[], beforecy[], firstcy[]
Output : a cabin number
if(k < beforecy[c]) {
    pos ← c
}
else {
    pos ← firstcy[c]
    k ← (k - beforecy[c]) % lency[pos]
}
for(i ← 0 to k-1 step 1) {
    pos ← T[pos]
}
return pos
```

Question 5 – Quidditch

Professor Dumbledore is in trouble: he needs to organise a Quidditch game (the favourite sport of the wizards) and must distribute the pupils in two groups from which he will be able to draw teams. The main issue is that many pupils do not get along together very well. For example, Ron doesn't want to speak with Draco anymore since he conjured slugs in his bed.

Professor Dumbledore has decided to record the enmities of the pupils in a graph like the one displayed below. It contains "nodes" labeled with the names of the pupils (here replaced by numbers to keep it readable) and links (called "edges") between those nodes whenever two pupils do not like each other.



First example: a graph with 5 nodes and 4 edges.

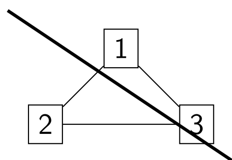
We can see on this example that 1 and 4 do not like each other and thus cannot be in the same group. However, 1 is fine with 2 and with 3.

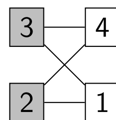
Thus, Dumbledore wants to split the pupils in two groups in order to form two teams. These two groups do not need to be of the same size, but **we cannot have two pupils that do not like each other in the same group**.

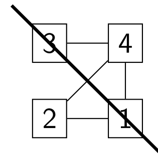
On the example above, it is easy to find two groups: $\{1, 2, 3\}$ and $\{4, 5\}$. To make it more readable, Dumbledore colors the nodes from the first group in grey and leaves the others in white.

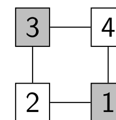
Here are other graphs. For each of them, we want to know whether it is possible to split their nodes into two groups as explained before. When it is possible, we ask you to color the nodes. If it is impossible, cross out the graph.

You can use the drawings hereunder as scratch, but **do not forget to copy your answers back to the answer sheet.**









(a) (b) (c) (d)

Q5(a) /2	Color the nodes from one group in graph (a), or cross out the graph if it is not possible.
Solution : See above.	

Q5(b) /2	Color the nodes from one group in graph (b), or cross out the graph if it is not possible.
Solution : See above.	

Q5(c) /2	Color the nodes from one group in graph (c), or cross out the graph if it is not possible.
Solution : See above.	

Q5(d) /2	Color the nodes from one group in graph (d), or cross out the graph if it is not possible.
Solution : See above.	

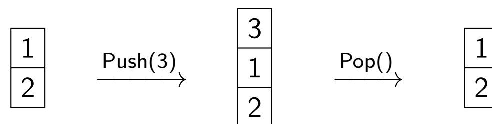
Now, professor Dumbledore wants to find an algorithm to make this distribution in two groups. That is, he wants to color the graphs automatically.

To achieve this, he has a good look at his favourite magical book: “De Algorithmica”, written by mage Cormenius (and his colleagues) in the thirteenth century. Unfortunately, some un-erasable magic ink has been spilled on the page that is relevant to Dumbledore’s goal and some parts of the algorithm are missing.

Dumbledore understands that he will need a data structure called a “stack”. A stack allows one to remember nodes when traversing a graph, similarly to a stack of plates, with a node at the bottom, another one on top of it and so forth until the top of the stack is reached. The following operations can be applied to the stack (see examples hereunder):

1. Push(x) adds a node x on the top of the stack;
2. Pop() returns the node at the top of the stack and removes it;
3. IsEmpty() returns **`true`** if the stack is empty and returns **`false`** otherwise.

Here is an example where we start with a stack of two elements, then 3 is added, then removed (the last Pop() thus returns 3).



Q5(e) /1	**Let us assume we start with an empty stack. Give a sequence of** Push() **that creates the stack:** 42 / 11 / 87 (top to bottom)
Solution : Push(87), Push(11), Push(42)	

Q5(f) /1	**Let us assume that we start with the stack** 3 / 1 / 2 (top to bottom) **and that we apply successively the operations:** Pop(), Pop(), Push(24), Push(37), Push(71), Pop(), Pop(). **What is the resulting stack?**

Solution:

Q5(g) /1	**What is the value returned by the last** Pop() **of the sequence in the previous question?**
Solution : The last Pop() returns 37.	

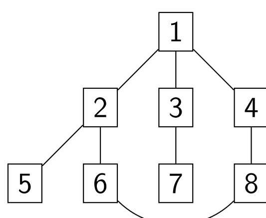
Professor Dumbledore can now consider a first *graph traversal* algorithm.
The graph to traverse contains `n` nodes and is described by the array of Boolean values `Edge[][]`.
`Edge[x][y]` and `Edge[y][x]` are both equal to **`true`** if there is an edge between nodes `x` and `y`, and equal to **`false`** otherwise.
With the graph of the first example, `Edge[1][4]` and `Edge[4][1]`, `Edge[1][5]` and `Edge[5][1]`, `Edge[2][4]` and `Edge[4][2]`, `Edge[3][5]` and `Edge[5][3]` are equal to **`true`**. All the other values in array `Edge[][]` are equal to **`false`** for this graph.

The algorithm receives the number of nodes `n`, the array `Edge[][]` and an initial node `s` where the traversal starts.
It relies on an array `color[]` *to color* the nodes during the traversal.
At the beginning of the program, `color[i]` is initialised to -1 for all nodes `i`.
To color a node `i`, we set `color[i]` to 0 and this value will not change anymore.
The algorithm uses a stack which is initially empty to store nodes that have to be visited later.
Note that, contrary to our habits, the indices of the nodes start with 1 in this program.

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

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

Push(s)
color[s] ← 0

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

Q5(h) /3	**In which order does the algorithm color the nodes of the graph of the first example (with 5 nodes and 4 edges) if the initial node is `s=4`?**
Solution : 4, 1, 2, 5, 3	

Q5(i) /5	**In which order does the algorithm color the nodes of the graph hereunder if the initial node is 1?**
Solution : 1, 2, 3, 4, 8, 6, 7, 5	



Finally, professor Dumbledore tries to recover the algorithm that separates the nodes of the graph in two groups. He understands that he must use two different colors, which will be 0 and 1, corresponding to the two groups. He also understands that if a node has color 0, all nodes connected to it by an edge must have color 1, and vice-versa.

With these ideas in mind, can you complete the algorithm hereunder ? It must return an array `color[]` containing the colors of all nodes if the split in two groups is possible and return **`false`** if it is impossible.

Q5(j) /6	**Fill in the ____ in the algorithm.** **Marks between 0 and 6. You will lose 2 marks for each mistake or missing answer.**
Solution : Correct answers are displayed on a grey background hereunder.	

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

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