be-OI 2015 **Demi-Finale** mercredi 19 novembre 2014	**Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp** PRÉNOM : ..... NOM : ..... ÉCOLE :	100 **Réservé**

Olympiade belge d'Informatique (durée : 3h maximum)

Ce document est le questionnaire de la demi-finale de l'Olympiade belge d'Informatique 2015. Il comporte neuf questions qui doivent être résolues en **3h au maximum**. Chaque question est accompagnée de son score maximal possible.

Notes générales (à lire attentivement avant de répondre aux questions)

1. N'indiquez votre nom, prénom et école **que sur la première page**. Indiquez vos réponses sur les pages prévues à cet effet, à la fin du formulaire. Si, suite à une rature, vous êtes amené à écrire hors d'un cadre, répondez obligatoirement sur la même feuille, sans quoi votre réponse ne pourra être corrigée.

2. Quand vous avez terminé, remettez au surveillant la première page (avec votre nom) et les pages avec les réponses. Vous pouvez conserver les autres.

3. Vous ne pouvez avoir que de quoi écrire avec vous; les calculatrices, GSM, ... sont **interdits**.

4. Vos réponses doivent être écrites **au stylo ou au bic** bleu ou noir. Pas de réponses laissées au crayon. Si vous désirez des feuilles de brouillon, demandez-en auprès d'un surveillant.

5. Tous les extraits de code de l'énoncé sont en **pseudo-code**. Vous trouverez, sur la page suivante, une **description** du pseudo-code que nous utilisons.

   Vous **devez** répondre aux questions ouvertes en **pseudo-code** ou dans l'un des **langages de programmation autorisés** (Java, C, C++, Pascal, Python, PHP, JavaScript, (Visual) Basic). Les erreurs de syntaxe ne sont pas prises en compte pour l'évaluation. Sauf mention contraire, vous ne pouvez utiliser aucune fonction prédéfinie à l'exception de `max(a,b)`, `min(a,b)` et `pow(a,b)` qui calcule a^b.

6. Pour toutes les questions à choix multiples (cases à cocher), une mauvaise réponse vous fait *perdre* le même nombre de points qu'elle peut vous en rapporter. Ne pas répondre ne fait ni perdre ni gagner de points. Si vous obtenez un score négatif à une question, votre score pour cette question sera ramené à zéro. Pour répondre à une question à choix multiples, cochez la case (☒) correspondant à votre réponse. Pour annuler une réponse, noircissez entièrement la case (■).

7. Vous **ne pouvez** à aucun moment **communiquer** avec qui que ce soit, excepté avec les surveillants. Toute question logistique peut leur être posée. A des fins d'égalité entre les participants des différents centres, vous ne **pouvez pas** poser de question **de compréhension** aux surveillants. Toute erreur dans l'énoncé doit être considéré comme faisant partie de l'épreuve.

8. Vous **ne pouvez pas quitter votre place** pendant l'épreuve. Si vous devez vous rendre aux toilettes, faites-en la demande à un surveillant. Ce dernier pourra décider d'accepter ou de refuser votre requête selon qu'il soit possible, ou pas, de vous accompagner tout en assurant la surveillance de l'épreuve. Selon le lieu où vous présentez l'épreuve, il peut vous être interdit de manger ou boire durant celle-ci.

Aide-mémoire pseudo-code

Les données sont stockées dans des variables. On change la valeur d'une variable à l'aide de $\leftarrow$. Dans une variable, nous pouvons stocker des nombres entiers, des nombres réels, ou des tableaux (voir plus loin), ainsi que des valeurs booléennes (logiques): vrai/juste (**`true`**) ou faux/erroné (**`false`**). Il est possible d'effectuer des opérations arithmétiques sur des variables. En plus des quatre opérateurs classiques ($+$, $-$, $\times$ et $/$), vous pouvez également utiliser $\%$: si a et b sont des nombres entiers, alors a/b et $a\%b$ désignent respectivement le quotient et le reste de la division entière. Par exemple, si $a = 14$ et $b = 3$, alors: $a/b = 4$ et $a\%b = 2$. Dans le code suivante, la variable age reçoit 20.

```
annee_naissance ← 1994
age ← 2014 − annee_naissance
```

Pour exécuter du code uniquement si une certaine condition est vraie, on utilise l'instruction **`if`** et éventuellement l'instruction **`else`** pour exécuter un autre code si la condition est fausse. L'exemple suivant vérifie si une personne est majeure et stocke le prix de son ticket de cinéma dans la variable $prix$. Notez les commentaires dans le code.

```
if (age ≥ 18)
{
    prix ← 8 // Ceci est un commentaire.
}
else
{
    prix ← 6 // Prix réduit
}
```

Pour manipuler plusieurs éléments avec une seule variable, on utilise un tableau. Les éléments individuels d'un tableau sont indiqués par un index (que l'on écrit entre crochets après le nom du tableau). Le premier élément d'un tableau tab est d'indice 0 et est noté $tab[0]$. Le second est celui d'indice 1 et le dernier est celui d'indice $N - 1$ si le tableau contient N éléments. Par exemple, si le tableau tab contient les 3 nombres 5, 9 et 12 (dans cet ordre), alors $tab\,[0] = 5$,$tab\,[1] = 9$, $tab\,[2] = 12$. La longueur est 3, mais l'index le plus élevé est 2.

Pour répéter du code, par exemple pour parcourir les éléments d'un tableau, on peut utiliser une boucle **`for`**. La notation **`for`** $(i \leftarrow a$ **`to`** b **`step`** $k)$ représente une boucle qui sera répétée tant que $i \leq b$, dans laquelle i commence à la valeur a, et est augmentée de k à la fin de chaque étape. L'exemple suivant calcule la somme des éléments du tableau tab en supposant que sa taille vaut N. La somme se trouve dans la variable sum à la fin de l'exécution de l'algorithme.

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

On peut également écrire une boucle à l'aide de l'instruction **`while`** qui repète du code tant que sa condition est vraie. Dans l'exemple suivant, on va diviser un nombre entier positif N par 2, puis par 3, ensuite par 4 ... jusqu'à ce qu'il ne soit plus composé que d'un seul chiffre (c'est-à-dire qu'il est < 10).

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

Question 1 – Les Bons et les Truands (16 pts)

Sur l'île de Sergio Leone il y a deux camps: les Bons, qui disent toujours la vérité, et les Truands, qui mentent toujours (les Brutes ont été éliminées). Il est impossible de les reconnaître autrement que par la véracité de leurs dires. Parmi les affirmations suivantes, marquez celles dont vous êtes *certain* qu'elles sont vraies. Quand vous ne pouvez pas déterminer si une affirmation est vraie ou fausse avec les informations dont vous disposez, indiquez « *pas certain* ».

Arthur dit : *Je suis un Bon.*

		Affirmation
Q1(a) [1 pt]	Certain / Pas certain	Arthur est un Bon
Q1(b) [1 pt]	Certain / Pas certain	Arthur est un Truand

Candice dit : *Je suis du même camp que Brice.*

		Affirmation
Q1(c) [1 pt]	Certain / Pas certain	Brice est un Bon
Q1(d) [1 pt]	Certain / Pas certain	Candice est une Bonne

Danaë dit : *De deux choses l'une : ou bien il y a un trésor sur l'île, ou bien je suis une Truande.*

		Affirmation
Q1(e) [1 pt]	Certain / Pas certain	Danaë est une Truande
Q1(f) [1 pt]	Certain / Pas certain	Il y a un trésor sur l'île

Ethan dit : *Francis est un truand.* Francis répond : *c'est faux.*

		Affirmation
Q1(g) [1 pt]	Certain / Pas certain	Francis est un Truand
Q1(h) [1 pt]	Certain / Pas certain	Ethan est un Truand
Q1(i) [1 pt]	Certain / Pas certain	Un des deux est un Bon, l'autre un Truand

Gaëlle dit : *Hubert et Irène sont du même camp.*

		Affirmation
Q1(j) [1 pt]	Certain / Pas certain	Au moins un des trois est un Bon
Q1(k) [1 pt]	Certain / Pas certain	Au moins deux des trois sont des Bons
Q1(l) [1 pt]	Certain / Pas certain	Un nombre impair des trois sont des Bons
Q1(m) [1 pt]	Certain / Pas certain	Au moins un des trois est un Truand

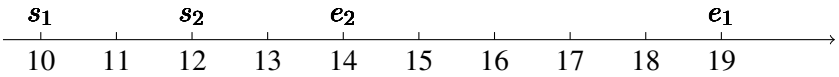
Louis, qui n'est pas Villageois, dit : *tous les Villageois sont des Truands*. Un Villageois dit : *Louis est un Bon*. Un autre Villageois dit : *Louis est un Truand*.

		Affirmation
Q1(n) [1 pt]	Certain / Pas certain	Louis est un Bon
Q1(o) [1 pt]	Certain / Pas certain	Louis est un Truand
Q1(p) [1 pt]	Certain / Pas certain	Certains Villageois sont des Truands

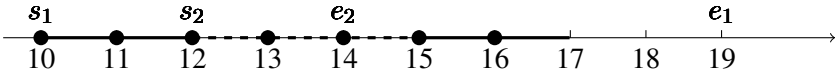
Question 2 – Le technicien (12 pts)

Un technicien doit effectuer deux réparations chez des clients, que nous appelons client 1 et client 2. Le technicien travaille par plages d'une heure, qui commencent à l'heure juste (par exemple, la plage qui commence à 10h dure de 10h00 à 10h59, et une nouvelle plage commence à 11h). La réparation nécessitera d_1 plages ($d_1 > 0$) chez le client 1 et d_2 ($d_2 > 0$) plages chez le client 2. Lorsqu'il commence la réparation chez un client, le technicien l'effectue d'une seule traite (il ne peut donc pas effectuer les deux réparations en même temps). Le premier client donne la fenêtre de temps pendant laquelle il souhaiterait que la réparation se fasse: $[s_1, e_1]$, où s_1 et e_1 sont des heures justes (entières). Cela signifie que le technicien peut commencer une plage d'une heure chez le client 1 au plus tôt à l'heure s_1 et peut commencer une plage d'une heure au plus tard à l'heure e_1. Par exemple, si $[s_1, e_1] = [10, 12]$, le technicien pourra travailler au plus durant 3 plages d'une heure: celle qui commence à 10h, celle qui commence à 11h, et celle qui commence à 12h. Idem pour le deuxième client, une fenêtre de temps est spécifiée $[s_2, e_2]$. On suppose que les fenêtres sont assez grandes pour les durées des réparations: $e_1 - s_1 + 1 \geq d_1$ et $e_2 - s_2 + 1 \geq d_2$. On suppose également que le temps nécessaire pour se déplacer d'un client à l'autre est négligeable.

On souhaiterait savoir dans quel(s) cas les deux réparations sont possibles, étant donné les contraintes des clients. Par exemple, si $[s_1, e_1] = [10, 19]$, $[s_2, e_2] = [12, 14]$, $d_1 = 4$ et $d_2 = 3$:

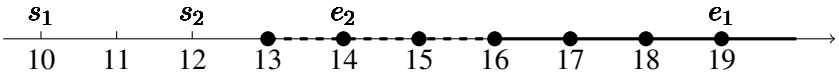


Si le technicien travaille chez le client 1 (trait continu) à 10h et 11h, puis chez le client 2 (pointillés) à 12h, 13h, et 14h, puis à nouveau chez le client 1 à 15h et 16h, on a:



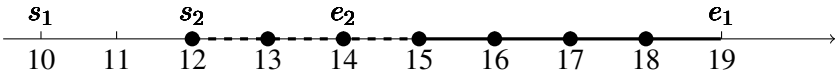
Cette solution ne convient pas, car le technicien ne travaille pas d'une seule traite. chez le client 1.

De même, la situation suivante, ne convient pas:



car le technicien ne respecte pas les contraintes du client 2.

Par contre la situation suivante:



respecte toutes les contraintes. La situation décrite ci-dessus admet donc une solution.

Pour les questions qui suivent, on vous demande de cocher une et une seule réponse (il existe toujours une réponse correcte).

Q2(a) [6 pts]	**Sous quelle condition existe-t-il une solution pour effectuer les réparations dans les fenêtres de temps pour chaque client?**
(1)	$(s_1 + d_1 > e_2 - d_2)$ ou $(s_2 + d_2 > e_1 - d_1)$
(2)	$s_1 \neq s_2$ et $e_1 \neq e_2$
(3)	$(s_1 + d_1 - 1 \leq e_2 - d_2)$ ou $(s_2 + d_2 - 1 \leq e_1 - d_1)$
(4)	$(e_2 - d_1 - d_2) \geq s_1$

Q2(b) [6 pts]	**S'il y a une solution, quel est le moment le plus tôt auquel le technicien pourra commencer la réparation chez le client 1 ?**
(1)	$e_2 + 1$
(2)	$s_2 + d_2$
(3)	$e_2 - d_2 - d_1 + 1$
(4)	$\max(s_2 + d_2, s_1)$
(5)	s_1
(6)	si $(s_1 + d_1 - 1 \leq e_2 - d_2)$ alors s_1 sinon $\max(s_2 + d_2, s_1)$

Question 3 – Black & white (22 pts)

Face à vous, une série de boîtes opaques et fermées. Chacune contient soit une boule noire, soit une boule blanche. Le contenu initial exact des boîtes vous est inconnu: vous ignorez, au début, quelle boîte contient quelle couleur.

On vous demande de trouver les meilleurs stratégies de résolution pour une série de problèmes. Les meilleurs stratégies attendues ici sont celles qui permettent d'effectuer le moins d'opérations possible dans le pire des cas, c'est-à-dire quand la configuration de départ est la moins favorable.

Exemple:

Configuration : Il y a 9 boîtes et une seule boule noire au total.
Problème : Déterminer la position de la boule noire, tout en ouvrant le moins de boîtes possible dans le pire des cas.
Question : Combien de boîtes faut il ouvrir, avec la stratégie trouvée, dans le pire des cas ?
Réponse : 8 **Explication :** Il est nécessaire d'ouvrir chacune des boîtes une à une, par exemple de gauche à droite. Au pire des cas, elle se trouve dans l'une des deux dernières boîtes. Si elle ne se trouve pas dans les huit premières, il n'est pas nécessaire d'ouvrir la dernière pour savoir qu'elle s'y trouve.

Problème 1 :

Configuration : Il y a 9 boîtes et 7 boules blanches au total.
Problème : Déterminer la position d'une des boules noires, tout en ouvrant le moins de boîtes possible dans le pire des cas.

Q3(a) [3 pts]	**Combien de boîtes faut il ouvrir, avec la stratégie trouvée, dans le pire des cas ?**

Problème 2 :

Configuration : Les boîtes alternent boules blanches et noires (par exemple $BNBNB$ ou NBN). Il y a n boîtes.
Problème : Déterminer le nombre total de boules noires, tout en ouvrant le moins de boîtes possible dans le pire des cas.

Q3(b) [2 pts]	**Si $n = 3$, combien de boîtes faut il ouvrir, avec la stratégie trouvée, dans le pire des cas ?**

Q3(c) [2 pts]	**Si $n = 8$, combien de boîtes faut il ouvrir, avec la stratégie trouvée, dans le pire des cas ?**

Q3(d) [2 pts]	**Si $n = 85$, combien de boîtes faut il ouvrir, avec la stratégie trouvée, dans le pire des cas ?**

../..

Problème 3 :

Configuration : De gauche à droite, les boîtes contiennent uniquement des boules blanches puis uniquement des boules noires (par exemple $BNNNN$, $BBBNN$ ou NN). Il y a n boîtes.
Problème : Déterminer le nombre total de boules noires, tout en ouvrant le moins de boîtes possible dans le pire des cas.

Q3(e) [2 pts]	**Si $n = 3$, combien de boîtes faut il ouvrir, avec la stratégie trouvée, dans le pire des cas ?**

Q3(f) [2 pts]	**Si $n = 15$, combien de boîtes faut il ouvrir, avec la stratégie trouvée, dans le pire des cas ?**

Q3(g) [3 pts]	**Si $n = 82$, combien de boîtes faut il ouvrir, avec la stratégie trouvée, dans le pire des cas ?**

Problème 4 :

Configuration: Il y a n boîtes.
Problème: Regrouper toutes les boules blanches à gauche et toutes les boules noires à droite (comme la configuration du problème 3), en effectuant uniquement des échanges de contenu, entre deux boîtes adjacentes, tout en effectuant le moins d'échanges possible dans le pire des cas.

Q3(h) [2 pts]	**Si $n = 4$, combien d'échanges faut il effectuer, avec la stratégie trouvée, dans le pire des cas ?**

Q3(i) [2 pts]	**Si $n = 20$, combien d'échanges faut il effectuer, avec la stratégie trouvée, dans le pire des cas ?**

Q3(j) [2 pts]	**Si $n = 21$, combien d'échanges faut il effectuer, avec la stratégie trouvée, dans le pire des cas ?**

Question 4 – Halte à la fraude (18 pts)

Le tout nouveau gouvernement de Palombie a décidé, afin de redresser l'économie nationale, de faire la chasse aux fraudeurs. Dans leur collimateur: les entreprises qui accordent trop de congés à leurs employés, faisant ainsi drastiquement chuter le PIB qui n'est pourtant pas très élevé...

Pour ce faire, chaque entreprise doit fournir aux agences gouvernementales le nombre n d'employés, et un tableau `congés` à n lignes et 366 colonnes dont chaque case contient un Booléen (**`true`** ou **`false`**). Le tableau `congés` indique quand chaque employé (identifié par son numéro unique i compris entre 0 et $n-1$) est en congé: la case `tableau[`i`][`j`]` contient **`true`** si l'employé i est en congé le j^{e} jour de l'année, **`false`** sinon.

Le gouvernement a décide de faire appliquer les trois règles suivantes:

1. un employé a droit à maximum 40 jours (!) de congé par an;
2. à tout moment, la moitié des employés doivent être présents dans la firme;
3. une période de congé ne peut excéder 15 jours.

Un fonctionnaire a donc été chargé d'écrire le programme nécessaire pour traiter le tableau et détecter les fraudes. Malheureusement, ce fonctionnaire est parti en congé, pour une période de trois mois, avant d'avoir terminé son travail (eh oui, le programme de contrôle du ministère n'étant pas encore terminé, les fraudes continuent...) Le gouvernement palombien compte sur votre aide pour compléter les informations manquantes, et vous promet de vous offrir trois semaines de... congés payés sur l'île paradisiaque de Santa Banana, si vous y parvenez !

```
Input : un tableau congés[n][366], un numéro d'employé e
Output: true si l'employé e n'accumule pas plus de 40 jours de
        congés sur l'année, false sinon.

sum ← [...]          // (a)
i ← 0
while (i < 366)
{
  if (congés[e][i] = true)
  {
     [...]                        //  (b)
  }
  i ← i+1
}
if (sum > 40)
{
  return [...]                    // (c)
}
else
{
  return [...]                    // (d)
}
```

../..

Q4(a) [2 pts]	**Quelle est l'expression** `(a)` **?**

Q4(b) [2 pts]	**Quelle est l'expression** `(b)` **?**

Q4(c) [1 pt]	**Quelle est l'expression** `(c)` **?**

Q4(d) [1 pt]	**Quelle est l'expression** `(d)` **?**

```
Input : un tableau congés[n][366]
Output: true s'il y a toujours au moins la moitié des employés qui
        travaillent chaque jour de l'année, false sinon

for (i ← 0 to 365 step 1)
{
  [...]                                  // (e)
  for (j ← [...] to [...] step 1)  // (f), (g)
  {
    if ([...] = true)                  // (h)
    {
      sum ← sum + 1
    }
  }
  if (sum > n/2)
  {
    return false
  }
}
return true
```

Q4(e) [2 pts]	**Quelle est l'instruction** `(e)` **?**

Q4(f) [1 pt]	**Quelle est l'expression** `(f)` **?**

Q4(g) [1 pt]	**Quelle est l'expression** `(g)` **?**

Q4(h) [2 pts]	**Quelle est l'expression** `(h)` **?**

../..

```
Input : un tableau congés[n][366], un numéro d'employé e
Output: true si l'employé e n'a aucune période de
        congé de plus de 15 jours, false sinon

i ← 0
while(i < 366)
{
  if (congés[e][i] = true)
  {
    [...]                       // (i)
    while ([...])               // (j)
    {
      i ← i+1
    }
    if (i-deb > 15)
    {
      return false
    }
  }
  else
  {
    i ← i+1
  }
}
return true
```

Q4(i) [3 pts]	**Quelle est l'instruction** `(i)` **?**

Q4(j) [3 pts]	**Quelle est la condition** `(j)` **?**

Question 5 – Fonctions récursives (14 pts)

Vous vous souvenez peut-être des nombres de Fibonacci appris lors de vos leçons de mathématiques. Le 0-ième nombre de Fibonacci est 0, et le 1-er est 1. Pour tout $n > 1$, le n-ième nombre de Fibonacci est la somme du $(n-1)$-ième et $(n-2)$-ième nombres de Fibonacci. Les huit premiers nombres de Fibonacci sont donc $0, 1, 1, 2, 3, 5, 8, 13$. Les nombres de Fibonacci peuvent être définis mathématiquement comme suit.

$$\mathrm{Fib}(0) = 0$$
$$\mathrm{Fib}(1) = 1$$
$$\mathrm{Fib}(n) = \mathrm{Fib}(n-1) + \mathrm{Fib}(n-2), \text{ pour } n > 1$$

C'est ce que l'on nomme une *fonction récursive*: la fonction Fib est définie en fonction d'elle-même. Dans un langage de programmation, il est facilement possible de transcrire une telle définition comme une fonction qui s'appelle elle-même:

```
Input : n, un nombre naturel, pour lequel nous voulons calculer le nombre de Fibonacci
Output : le n-ième nombre de Fibonacci

Fib(n)
{
  if (n = 0)
  {
    return 0
  }
  else if (n = 1)
  {
    return 1
  }
  else
  {
    return Fib(n − 1) + Fib(n − 2)
  }
}
```

Q5(a) [1 pt]	**Quel est le résultat de l'appel de fonction** $\mathrm{Fib}(9)$**?**

Q5(b) [2 pts]	**Combien de fois la fonction** Fib **s'appelle-t-elle elle-même après l'appel à** $\mathrm{Fib}(2)$ **?**

Q5(c) [3 pts]	**Combien de fois la fonction** Fib **s'appelle-t-elle elle-même après l'appel à** $\mathrm{Fib}(5)$ **?**

../..

Des informaticiens malins ont trouvé une façon différente (et, espérons-le, meilleure) de calculer les nombres de Fibonacci. Nous pouvons l'exprimer dans un langage de programmation sous la forme de la fonction récursive $\mathrm{BetterFib}$ suivante:

```
Input : n, un nombre naturel, pour lequel nous voulons calculer le nombre de Fibonacci
        a, un nombre naturel qui est initialement 0
        b, un nombre naturel qui est initialement 1
        i, un nombre naturel qui est initialement 0
Output : le n-ième nombre de Fibonacci

BetterFib(n, a, b, i) {
  if (i = n)
  {
    return a
  }
  else
  {
    return BetterFib(n, b, a+b, i+1)
  }
}
```

Pour calculer le n-ième nombre de Fibonacci, on appelle $\mathrm{BetterFib}(n, 0, 1, 0)$. On comprend sans doute mieux le code ci-dessus quand on se rend compte que, lors de chaque appel à $\mathrm{BetterFib}$, a contient toujours le i-ième nombre de Fibonacci, et b contient toujours le $(i+1)$-ième nombre de Fibonacci.

Q5(d) [4 pts]	**Combien de fois la fonction** $\mathrm{BetterFib}$ **s'appelle-t-elle elle-même après l'appel à** $\mathrm{BetterFib}(2, 0, 1, 0)$ **?**

Q5(e) [4 pts]	**Combien de fois la fonction** $\mathrm{BetterFib}$ **s'appelle-t-elle elle-même après l'appel à** $\mathrm{BetterFib}(5, 0, 1, 0)$ **?**

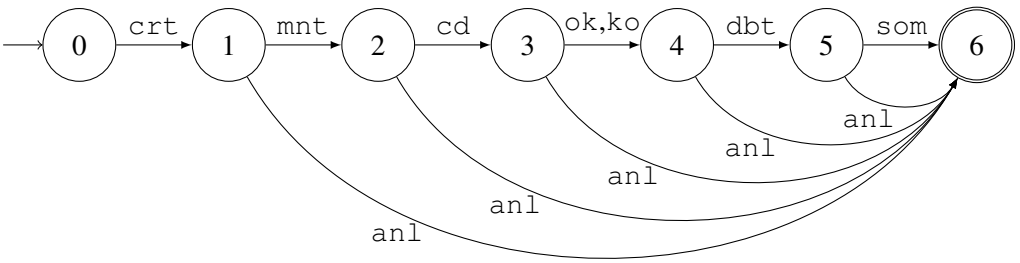
Question 6 – Le distributeur de billets de la banque Picsou (26 pts)

Dans un souci de modernisation, l'oncle Picsou a décidé de mettre en place un système de distributeurs de billets de banque pour ses clients. Afin d'économiser sur les coûts de production, il a décidé de sous-traiter le développement des distributeurs à ses neveux Riri, Fifi et Loulou.

Le distributeur qu'ils ont imaginé peut effectuer les actions suivantes:

Nom de l'action	Description
`crt`	L'utilisateur entre sa carte dans le distributeur pour démarrer la transaction
`mnt`	L'utilisateur entre le montant demandé
`cd`	L'utilisateur entre son code
`ok`	Le code est reconnu comme correct par le distributeur
`ko`	Le code est reconnu comme erroné par le distributeur
`dbt`	L'argent est débité du compte
`som`	Le distributeur délivre la somme et la carte est restituée
`anl`	L'utilisateur appuie sur le touche « annulation » et la carte est restituée

Afin de décrire précisément *dans quel ordre ces actions peuvent s'enchaîner*, Riri, Fifi et Loulou proposent le diagramme suivant:



Sur ce diagramme, chaque cercle représente un *état* possible du distributeur (on les a numérotés de 0 à 6), et chaque flèche représente une *action possible*. À tout moment, le distributeur est dans un état unique. Initialement, le distributeur est dans l'état 0, comme l'indique la petite flèche à gauche. Chaque action a pour effet de changer l'état courant, comme indiqué par les flèches. Par exemple, si le distributeur est dans l'état 1, les actions `mnt` et `anl` sont possibles: exécuter l'action `mnt` change l'état courant en 2, et l'action `anl` change l'état courant en 6. Quand une flèche est étiquetée par plusieurs actions, séparées par des virgules, cela signifie qu'on peut choisir une (et une seule) de ces actions pour changer d'état. Par exemple, pour aller de 3 à 4, il faut faire soit l'action `ok`, soit l'action `ko`. Enfin, l'état 6 est un état final (indiqué par le bord double): la transaction est terminée quand le distributeur arrive dans cet état.

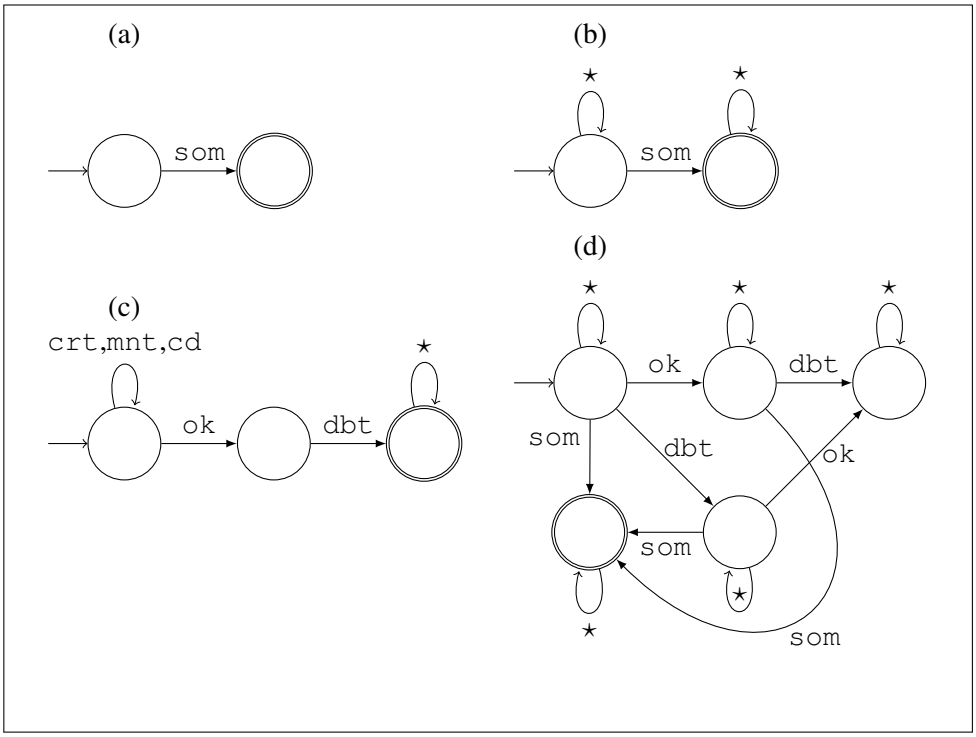
Toutes les exécutions possibles du distributeur sont représentées par ce schéma. Par exemple, l'utilisateur pourrait insérer sa carte (action `crt`, l'état courant devient 1), puis entrer un montant (action `mnt`, l'état courant devient 2), puis se raviser et annuler la transaction (action `anl`, l'état courant devient l'état final 6), ce qui termine la transaction. Comme on peut le voir, le distributeur de Riri, Fifi et Loulou souffre de quelques défauts… Parmi les scénarios suivants, lesquels correspondent à des exécutions possibles (allant de l'état initial à l'état final) du diagramme ci-dessus ?

		Est-il possible de… ?
Q6(a) [2 pts]	Possible / Pas possible	recevoir de l'argent (action `som`) sans avoir entré aucun code (action `cd`).
Q6(b) [2 pts]	Possible / Pas possible	avoir son compte débité (action `dbt`) sans recevoir l'argent (action `som`).
Q6(c) [2 pts]	Possible / Pas possible	recevoir de l'argent (action `som`) sans avoir entré de code correct (action `ok`).
Q6(d) [2 pts]	Possible / Pas possible	recevoir de l'argent (action `som`) sans avoir inséré sa carte (action `crt`).

L'oncle Picsou n'est pas très heureux de ces problèmes potentiels ! Cela lui fait perdre du temps, et le temps, c'est de l'argent ! L'oncle Picsou décide donc que toutes les exécutions du distributeur doivent respecter la propriété suivante:

> **Si** l'argent est distribué (action `som`) **alors** l'utilisateur doit avoir entré un code correct (action `ok`) **et** l'argent doit avoir été débité (action `dbt`).

Afin de rendre les choses parfaitement claires, l'oncle Picsou souhaite utiliser le même type de diagramme que celui utilisé par Riri, Fifi et Loulou pour spécifier *certaines exécutions problématiques*. Il propose quatre diagrammes, donnés ci-dessous. Le symbole $\star$ a été utilisé par souci de clarté, pour représenter « toutes les autres actions ». Par exemple sur le diagramme (b) ci-dessous, le symbole $\star$ sur la boucle de l'état initial signifie: « toutes les actions, sauf `som` », étant donné que l'action `som` apparaît sur la flèche entre l'état initial et l'état final. En d'autres termes, on aurait pu remplacer le $\star$ sur la boucle de l'état initial par « `crt`,`mnt`,`cd`,`ok`,`ko`,`som`,`dbt`,`anl` ». Le symbole $\star$ sur la boucle de l'état final, par contre, peut être compris comme « n'importe quelle action » étant donné qu'il n'y a pas d'autre flèche sortant de l'état final.



On vous demande de caractériser les exécutions de ces quatre diagrammes (peu importe si ces exécutions sont permises par le distributeur ou pas). Comme dans le diagramme donné par Riri, Fifi et Loulou, une exécution commence nécessairement dans l'état initial et se termine dans l'état final (celles qui ne se terminent pas dans l'état final doivent être ignorées).

		Est-il vrai que… ?
Q6(e) [2 pts]	Oui / Non	toutes les exécutions du diagramme (a) violent la propriété.
Q6(f) [2 pts]	Oui / Non	il existe des exécutions du diagramme (a) qui violent la propriété.
Q6(g) [2 pts]	Oui / Non	toutes les exécutions du diagramme (b) violent la propriété.
Q6(h) [2 pts]	Oui / Non	il existe des exécutions du diagramme (b) qui violent la propriété.
Q6(i) [2 pts]	Oui / Non	toutes les exécutions du diagramme (c) violent la propriété.
Q6(j) [2 pts]	Oui / Non	il existe des exécutions du diagramme (c) qui violent la propriété.
Q6(k) [3 pts]	Oui / Non	toutes les exécutions du diagramme (d) violent la propriété.
Q6(l) [3 pts]	Oui / Non	il existe des exécutions du diagramme (d) qui violent la propriété.

Question 7 – Jeux d'entiers (20 pts)

À l'époque déjà lointaine où les ordinateurs n'existaient pas, les gens devaient faire preuve d'inventivité pour meubler l'ennui d'une vie terne, dépourvue de jeux vidéos. Dans un moment particulièrement désespéré, un groupe d'amis a inventé ce jeu : une personne choisit deux entiers strictement positifs, et les autres doivent trouver une manière de passer de l'un à l'autre suivant certaines règles choisies à l'avance.

Un exemple de règles serait que l'on peut ajouter 10 au nombre ou enlever 1. Donc, pour aller de 7 à 25, on pourrait passer par :

$$7 \xrightarrow{+10} 17 \xrightarrow{+10} 27 \xrightarrow{-1} 26 \xrightarrow{-1} 25$$

Pour rendre les choses plus intéressantes, ils ont décidé d'ajouter une contrainte : le chemin trouvé doit contenir le moins d'étapes possible. L'exemple ci-dessus en possède 4, représentées par les flèches.

Votre rôle est de déterminer, pour certaines règles et deux entiers, le nombre **minimal** d'étapes nécessaires pour passer du premier au deuxième nombre. S'il n'est pas possible de passer de l'un à l'autre, écrivez -1.

- **Règles 1 :** on peut ajouter 5 au nombre, y soustraire 5, le multiplier par 5 ou le diviser par 5, à condition qu'à chaque étape le résultat soit un entier strictement positif (donc on ne peut diviser par 5 que si le nombre est un multiple de 5). Par exemple, un chemin acceptable serait :

$$10 \xrightarrow{-5} 5 \xrightarrow{\div 5} 1 \xrightarrow{+5} 6$$

Q7(a) [2 pts]	**Combien d'étapes sont nécessaires pour passer de 2 à 3 ? Écrire -1 si c'est impossible.**

Q7(b) [2 pts]	**Combien d'étapes sont nécessaires pour passer de 15 à 7 ? Écrire -1 si c'est impossible.**

Q7(c) [2 pts]	**Combien d'étapes sont nécessaires pour passer de 30 à 60 ? Écrire -1 si c'est impossible.**

- **Règles 2 :** on peut ajouter 3 au nombre, ou échanger deux chiffres du nombre, tant que cela ne place pas de zéro au début du nombre. Par exemple, un chemin acceptable serait :

$$24 \xrightarrow{+3} 27 \xrightarrow{\leftrightarrow} 72 \xrightarrow{+3} 75 \xrightarrow{+3} 78$$

Q7(d) [2 pts]	**Combien d'étapes sont nécessaires pour passer de 13 à 55 ? Écrire -1 si c'est impossible.**

Q7(e) [2 pts]	**Combien d'étapes sont nécessaires pour passer de 12 à 37 ? Écrire -1 si c'est impossible.**

Q7(f) [2 pts]	**Combien d'étapes sont nécessaires pour passer de 98 à 14 ? Écrire -1 si c'est impossible.**

Q7(g) [2 pts]	**Combien d'étapes sont nécessaires pour passer de 99 à 36 ? Écrire -1 si c'est impossible.**

../..

- **Règles 3 :** on peut multiplier le nombre par 2, ou y ajouter 1 ; on peut aussi enlever tous les chiffres du nombre qui sont plus grands ou égaux à 5, simultanément, à condition que le nombre résultant ait au moins un chiffre et que son premier chiffre ne soit pas zéro. Par exemple, un chemin acceptable serait :

$$24 \xrightarrow{\times 2} 48 \xrightarrow{\cancel{8}} 4 \xrightarrow{\times 2} 8 \xrightarrow{+1} 9$$

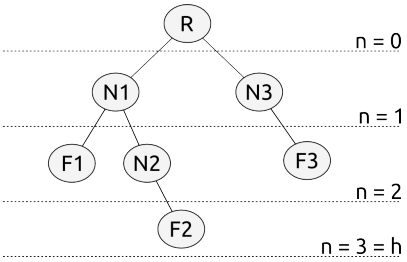
Q7(h) [2 pts]	**Combien d'étapes sont nécessaires pour passer de 5 à 4 ? Écrire -1 si c'est impossible.**

Q7(i) [2 pts]	**Combien d'étapes sont nécessaires pour passer de 20 à 3 ? Écrire -1 si c'est impossible.**

Q7(j) [2 pts]	**Combien d'étapes sont nécessaires pour passer de 113 à 1 ? Écrire -1 si c'est impossible.**

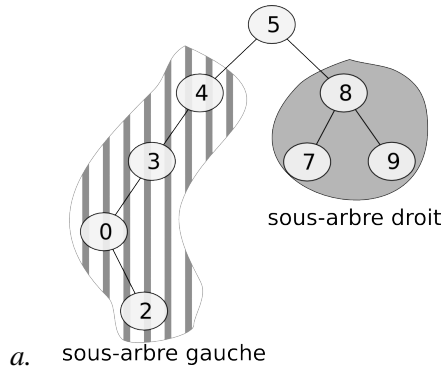
Question 8 – Arbres binaires de recherche équilibrés (25 pts)

Afin d'accéder à des données de manière performante, il est possible de les représenter sous la forme d'un arbre binaire de recherche. Un arbre est typiquement représenté comme l'exemple ci-dessous. Chaque élément de l'arbre est appelé un **nœud**. Dans un arbre binaire, chaque nœud peut être le **père** de deux **enfants** au plus. Sur l'image, N1 est le **père** de deux enfants: F1 et N2. Les nœuds au bas de l'image, qui n'ont pas d'enfant, sont appelés les **feuilles** de l'arbre. Un arbre a également un seul élément sans père: c'est l'élément tout en haut de l'image, que nous appelons la **racine** de l'arbre. Un nœud peut contenir une **valeur**. Dans le cadre de cette question, cette valeur sera toujours une entier unique.

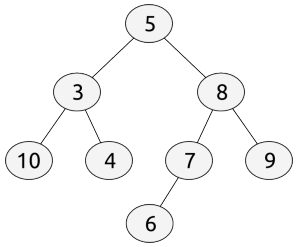


La **hauteur** d'un arbre est le nombre maximum de nœuds qu'il faut parcourir dans l'arbre pour atteindre une feuille depuis la racine, sans la compter. Dans l'exemple précédent, la hauteur de l'arbre est de **3** car il faut parcourir 3 nœuds pour atteindre la feuille la plus éloignée de la racine F2. Le **niveau** d'un nœud est sa profondeur par rapport à la racine, donc la feuille F2 est de niveau 3, par exemple.

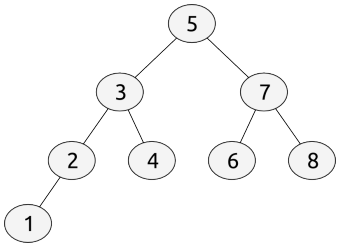
Un arbre binaire peut être **équilibré**, c'est-à-dire que pour tout nœud de l'arbre, les hauteurs de ses **sous-arbres** (celui de son fils droit et celui de son fils gauche) ne peuvent différer de plus d'une unité. Par exemple, dans les exemples suivants, l'arbre de gauche n'est pas équilibré car les deux sous-arbres du nœud **5** ont des hauteurs qui diffèrent de plus de 1: la hauteur du sous-arbre gauche vaut 3 et la hauteur du sous-arbre droit vaut 1 ($3 - 1 > 1$).



a.



b.



c.

Q8(a) [2 pts]	**Quel est le niveau du nœud 4 dans l'arbre *a.* ci-dessus ?**

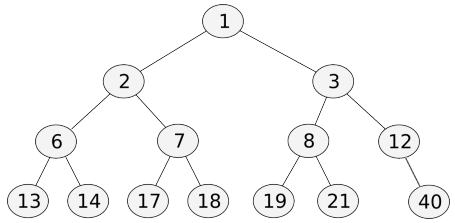
Q8(b) [2 pts]	**Quelle est la hauteur de l'arbre *b.* ci-dessus ?**

Q8(c) [2 pts]	**Quelle est la hauteur du sous-arbre du nœud 3 dans l'arbre *c.* ci-dessus ?**

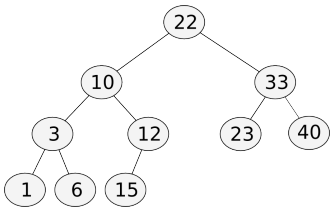
Une forme particulière d'arbre de recherche est l'arbre **AVL** (du nom de ses inventeurs, Georgy **A**delson-**V**elsky et Evguenii **L**andis). Les arbres AVL sont des arbres binaires équilibrés dont tous les nœuds n satisfont la propriété suivante: tous les nœuds du sous-arbre gauche de n ont une valeur *inférieure* à n, et tous les nœuds du sous-arbre droit de n ont une valeur *supérieure* à n. Parmi les arbres ci-dessus, l'arbre *b.* est équilibré, mais pas AVL. Par contre l'arbre *c.* est un AVL.

Indiquez, pour chacune des 6 propositions ci-dessous, s'il s'agit d'un AVL ou pas.

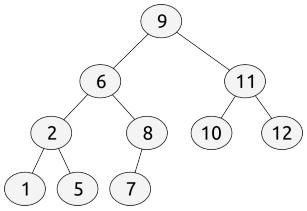
1.



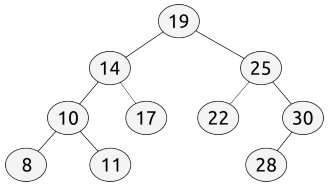
2.



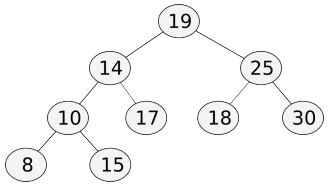
3.



4.

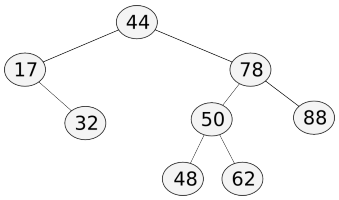


5.



		Numéro de l'arbre
Q8(d) [1 pt]	AVL / Non-AVL	Arbre *1.*
Q8(e) [1 pt]	AVL / Non-AVL	Arbre *2.*
Q8(f) [1 pt]	AVL / Non-AVL	Arbre *3.*
Q8(g) [1 pt]	AVL / Non-AVL	Arbre *4.*
Q8(h) [1 pt]	AVL / Non-AVL	Arbre *5.*

Considérons maintenant l'arbre AVL suivant :



On voudrait modifier l'arbre ci-dessous en ajoutant ou supprimant des valeurs de l'arbre tout en conservant les propriétés d'un arbre AVL et en effectuant un nombre minimal de déplacements de nœuds de l'arbre de départ. On considère qu'un nœud est déplacé si la valeur de son père a changé. Pour chaque sous-question, on vous demande de dessiner l'arbre AVL obtenu après avoir effectué l'opération demandée. **Pour chaque sous-question, repartez de l'arbre initial**.

Q8(i) [2 pts]	**Dessinez l'arbre AVL obtenu après l'ajout de la valeur 14 à l'arbre initial.**

Q8(j) [2 pts]	**Dessinez l'arbre AVL obtenu après l'ajout de la valeur 30 à l'arbre initial.**

Q8(k) [2 pts]	**Dessinez l'arbre AVL obtenu après l'ajout de la valeur 40 à l'arbre initial.**

Q8(l) [4 pts]	**Dessinez l'arbre AVL obtenu après l'ajout de la valeur 54 à l'arbre initial.**

Q8(m) [4 pts]	**Dessinez l'arbre AVL obtenu après la suppression de la valeur 32 de l'arbre initial.**

Question 9 – Double 1 (19 pts)

Votre tâche est d'écrire une fonction qui double tous les "1" dans un tableau de n nombres. Par exemple, si le tableau contient $[1, 1, 5, 1, 4]$ avant l'appel de la fonction, il devra contenir $[1, 1, 1, 1, 5, 1, 1, 4]$ après l'appel à celle-ci. Pour simplifier les choses, le tableau qui vous est fourni a une taille $2n$, ce qui permet de modifier le tableau sans devoir en créer un nouveau.

Voici la définition des entrées et de la sortie de l'algorithme.

```
Input : n, un nombre entier.
        tab, un tableau de nombres entiers de taille 2n.
Output: tab est modifié pour que tous les 1 parmi les n premiers nombres du
        tableau initial soient doublés.
```

Nous vous proposons deux algorithmes permettant de résoudre ce même problème, vous devez les compléter.

Algorithme 1

```
count ← 0
for (i ← 0 to [...] step 1)        // (a)
{
  if (tab[[...]] = 1)                        // (b)
  {
    for (j ←[...] to i+1 step -1)        // (c)
    {
      tab[[...]] ← tab[[...]] // (d), (e)
    }
    count ← count + 1
  }
}
```

Q9(a) [2 pts]	**Quelle doit être l'expression** `(a)` **?**

Q9(b) [3 pts]	**Quelle doit être l'expression** `(b)` **?**

Q9(c) [2 pts]	**Quelle doit être l'expression** `(c)` **?**

Q9(d) [2 pts]	**Quelle doit être l'expression** `(d)` **?**

Q9(e) [2 pts]	**Quelle doit être l'expression** `(e)` **?**

../..

Algorithme 2

```
count ← 0
for (i ← 0 to n − 1 step 1)
{
  if (tab[i] = 1)
  {
    count ← count + 1
  }
}
for (j ←[...] to [...] step [...])  // (f), (g), (h)
{
  [...]                          // (i)
  if (tab[j] = 1)
  {
    tab[j+[...]] ← 1  // (j)
    count ← count − 1
  }
}
```

Q9(f) [2 pts]	**Quelle doit être l'expression** `(f)` **?**

Q9(g) [1 pt]	**Quelle doit être l'expression** `(g)` **?**

Q9(h) [1 pt]	**Quelle doit être l'expression** `(h)` **?**

Q9(i) [5 pts]	**Quelle doit être l'expression** `(i)` **?**

Q9(j) [4 pts]	**Quelle doit être l'expression** `(j)` **?**

En sachant que l'algorithme 1 prend environ 8 minutes pour s'exécuter sur un ordinateur moderne lorsque n vaut 1 000 000.

Q9(k) [5 pts]	**Combien de temps ce même ordinateur prendra-t-il pour exécuter l'algorithme 2 ?**
(1)	Environ 10 millisecondes
(2)	Environ 4 minutes
(3)	Environ 8 minutes
(4)	Environ 15 minutes
(5)	Plusieurs jours

be-OI 2015 mercredi 19 novembre 2014	**Réservé** 100

Donnez vos réponses ici !

	Certain	Pas certain	Affirmation	/2
Q1(a) /1	☐	☐	Arthur est un Bon	
Q1(b) /1	☐	☐	Arthur est un Truand	

	Certain	Pas certain	Affirmation	/2
Q1(c) /1	☐	☐	Brice est un Bon	
Q1(d) /1	☐	☐	Candice est une Bonne	

	Certain	Pas certain	Affirmation	/2
Q1(e) /1	☐	☐	Danaë est une Truande	
Q1(f) /1	☐	☐	Il y a un trésor sur l'île	

	Certain	Pas certain	Affirmation	/3
Q1(g) /1	☐	☐	Francis est un Truand	
Q1(h) /1	☐	☐	Ethan est un Truand	
Q1(i) /1	☐	☐	Un des deux est un Bon, l'autre un Truand	

	Certain	Pas certain	Affirmation	/4
Q1(j) /1	☐	☐	Au moins un des trois est un Bon	
Q1(k) /1	☐	☐	Au moins deux des trois sont des Bons	
Q1(l) /1	☐	☐	Un nombre impair des trois sont des Bons	
Q1(m) /1	☐	☐	Au moins un des trois est un Truand	

	Certain	Pas certain	Affirmation	/3
Q1(n) /1	☐	☐	Louis est un Bon	
Q1(o) /1	☐	☐	Louis est un Truand	
Q1(p) /1	☐	☐	Certains Villageois sont des Truands	

Q2(a)		**Ne cochez qu'UNE seule réponse !**	/6
(1)	☐	$(s_1 + d_1 > e_2 - d_2)$ ou $(s_2 + d_2 > e_1 - d_1)$	
(2)	☐	$s_1 \neq s_2$ et $e_1 \neq e_2$	
(3)	☐	$(s_1 + d_1 - 1 \leq e_2 - d_2)$ ou $(s_2 + d_2 - 1 \leq e_1 - d_1)$	
(4)	☐	$(e_2 - d_1 - d_2) \geq s_1$	

Q2(b)		**Ne cochez qu'UNE seule réponse !**	/6
(1)	☐	$e_2 + 1$	
(2)	☐	$s_2 + d_2$	
(3)	☐	$e_2 - d_2 - d_1 + 1$	
(4)	☐	$\max(s_2 + d_2, s_1)$	
(5)	☐	s_1	
(6)	☐	si $(s_1 + d_1 - 1 \leq e_2 - d_2)$ alors s_1 sinon $\max(s_2 + d_2, s_1)$	

Q3(a) **Un nombre**	/3
Q3(b) **Un nombre**	/2
Q3(c) **Un nombre**	/2
Q3(d) **Un nombre**	/2
Q3(e) **Un nombre**	/2
Q3(f) **Un nombre**	/2
Q3(g) **Un nombre**	/3
Q3(h) **Un nombre**	/2
Q3(i) **Un nombre**	/2
Q3(j) **Un nombre**	/2
Q4(a) **Une expression**	/2
Q4(b) **Une expression**	/2

Q4(c) **Une expression**	/1
Q4(d) **Une expression**	/1
Q4(e) **L'instruction** `(e)`	/2
Q4(f) **Une expression**	/1
Q4(g) **Une expression**	/1
Q4(h) **Une expression**	/2
Q4(i) **Une instruction**	/3
Q4(j) **Une condition**	/3
Q5(a) **Un nombre naturel**	/1
Q5(b) **Un nombre d'appels**	/2
Q5(c) **Un nombre d'appels**	/3
Q5(d) **Un nombre d'appels**	/4
Q5(e) **Un nombre d'appels**	/4

	Possible	Pas possible	Est-il possible de… ?	/8
Q6(a) /2	☐	☐	recevoir de l'argent (action `som`) sans avoir entré aucun code (action `cd`).	
Q6(b) /2	☐	☐	avoir son compte débité (action `dbt`) sans recevoir l'argent (action `som`).	
Q6(c) /2	☐	☐	recevoir de l'argent (action `som`) sans avoir entré de code correct (action `ok`).	
Q6(d) /2	☐	☐	recevoir de l'argent (action `som`) sans avoir inséré sa carte (action `crt`).	

	Oui	Non	Est-il vrai que… ?	/18
Q6(e) /2	☐	☐	toutes les exécutions du diagramme (a) violent la propriété.	
Q6(f) /2	☐	☐	il existe des exécutions du diagramme (a) qui violent la propriété.	
Q6(g) /2	☐	☐	toutes les exécutions du diagramme (b) violent la propriété.	
Q6(h) /2	☐	☐	il existe des exécutions du diagramme (b) qui violent la propriété.	
Q6(i) /2	☐	☐	toutes les exécutions du diagramme (c) violent la propriété.	
Q6(j) /2	☐	☐	il existe des exécutions du diagramme (c) qui violent la propriété.	
Q6(k) /3	☐	☐	toutes les exécutions du diagramme (d) violent la propriété.	
Q6(l) /3	☐	☐	il existe des exécutions du diagramme (d) qui violent la propriété.	

Q7(a) **Le nombre minimal d'étapes, ou** -1 /2

. . . . . .

Q7(b) **Le nombre minimal d'étapes, ou** -1 /2

. . . . . .

Q7(c) **Le nombre minimal d'étapes, ou** -1 /2

. . . . . .

Q7(d) **Le nombre minimal d'étapes, ou** -1 /2

. . . . . .

Q7(e) **Le nombre minimal d'étapes, ou** -1 /2

. . . . . .

Q7(f) **Le nombre minimal d'étapes, ou** -1 /2

. . . . . .

Q7(g) **Le nombre minimal d'étapes, ou** -1 /2

. . . . . .

Q7(h) **Le nombre minimal d'étapes, ou** -1	/2
Q7(i) **Le nombre minimal d'étapes, ou** -1	/2
Q7(j) **Le nombre minimal d'étapes, ou** -1	/2
Q8(a) **Le niveau du nœud 4 de l'arbre *a.* ?**	/2
Q8(b) **La hauteur de l'arbre *b.* ?**	/2
Q8(c) **La hauteur du sous arbre du nœud 3 de l'arbre *c.* ?**	/2

	AVL	Non-AVL	Numéro de l'arbre	
Q8(d) /1	☐	☐	Arbre *1.*	/5
Q8(e) /1	☐	☐	Arbre *2.*	
Q8(f) /1	☐	☐	Arbre *3.*	
Q8(g) /1	☐	☐	Arbre *4.*	
Q8(h) /1	☐	☐	Arbre *5.*	

Q8(i) **L'arbre après ajout de la valeur 14**	/2

Q8(j) **L'arbre après ajout de la valeur 30**	/2
Q8(k) **L'arbre après ajout de la valeur 40**	/2
Q8(l) **L'arbre après ajout de la valeur 54**	/4

Q8(m) **L'arbre après suppression de la valeur 32**	/4
Q9(a) **Une expression**	/2
Q9(b) **Une expression**	/3
Q9(c) **Une expression**	/2
Q9(d) **Une expression**	/2
Q9(e) **Une expression**	/2
Q9(f) **Une expression**	/2
Q9(g) **Une expression**	/1
Q9(h) **Une expression**	/1
Q9(i) **Une instruction**	/5
Q9(j) **Une expression**	/4

Q9(k)		**Ne cochez qu'UNE seule réponse !**	/5
(1)	☐	Environ 10 millisecondes	
(2)	☐	Environ 4 minutes	
(3)	☐	Environ 8 minutes	
(4)	☐	Environ 15 minutes	
(5)	☐	Plusieurs jours	