be-OI 2013 **Demi-finale** 23 janvier 2013	**Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp** PRÉNOM : ..... NOM : ..... ÉCOLE :	**Réservé**

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

Ce document est le questionnaire de la demi-finale de l'Olympiade belge d'Informatique 2013. Il comporte neuf questions qui doivent être résolues en **3h au maximum**. Chaque question est accompagnée d'un temps de résolution indicatif, ainsi que 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**. Sur toutes les autres pages, vous ne pouvez écrire que dans les **cadres prévus** pour votre réponse. 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. Vous ne pouvez avoir que de quoi écrire avec vous ; les calculatrices, GSM, ... sont **interdits**.

3. 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.

4. 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 et PHP). 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.

5. 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 (■).

6. 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.

7. 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 19.

```
annee_naissance ← 1993
age ← 2012 − annee_naissance
```

Pour exécuter du code uniquement si une certaine condition est vérifiée, on utilise l'instruction **`if`** et éventuellement l'instruction **`else`** pour exécuter un autre code si cette même condition est fausse. L'exemple suivant vérifie si une personne est majeure et stocke le prix de son ticket de cinéma selon le cas dans une variable entière, $prix$.

```
if (age ≥ 18)
{
    prix ← 8
}
else
{
    prix ← 6
}
```

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 – Pensons logique (5 min – 4 pts)

Considérons le fragment de programme suivant dans lequel la fonction `Afficher(`x`)` affiche **`true`** si x est *vrai* (true) et **`false`** sinon.

```
Input : deux valeurs entières a et b

if (a = b and a ≠ 0)
{
   Afficher(b = 0)
}
else
{
    if (b ≠ 0)
    {
       Afficher(a = 0)
    }
    else
    {
       Afficher(a ≠ b)
    }
}
```

Indiquez, pour chacune des affirmations suivantes, si elle est **correcte** ou **incorrecte**.

	Correcte	Incorrecte	
Q1(a)	☒	☐	Si a et b sont nulles ($= 0$), alors **`false`** est affiché.
Q1(b)	☒	☐	Si **`true`** est affiché, il n'est pas possible que a et b soient égales.
Q1(c)	☐	☒	Si ni a ni b sont nulles, alors **`true`** est affiché.
Q1(d)	☒	☐	Si a et b sont égales, alors **`false`** est affiché, peu importe la valeur de a et b.

Question 2 – Recherche un nombre dans une table (5 min – 4 pts)

John a écrit le morceau de programme suivant pour tester si un nombre n est présent ou pas dans un tableau donné tab de 20 éléments (pour rappel : les positions d'un tableau sont numérotées à partir de 0 !)

```
Input  : un tableau tab à 20 éléments et une valeur entière n
Output : affiche un message indiquant si tab contient n ou non

index ← 19
while (index > 0 and tab[index] ≠ n)
{
   index ← index − 1
}
if ( [...] )
{
   Afficher("Le nombre est dans le tableau")
}
else
{
   Afficher("Le nombre n'est pas dans le tableau")
}
```

John n'est pas tout à fait certain de la condition qu'il doit utiliser dans l'instruction **`if`** (sixième ligne). Il considère quatre options (ci-dessous).

Marquez les options qui constituent une condition valide pour l'instruction **`if`**. Par "valide", nous entendons que le programme doit donner le bon résultat quelles que soient les valeurs contenues dans le tableau `tab`.

	valide	non valide	
Q2(a)	☐	☒	$index \leq 0$
Q2(b)	☒	☐	$tab[index] = n$
Q2(c)	☐	☒	$tab[index] = n$ **and** $index \leq 0$
Q2(d)	☐	☒	$tab[index] = n$ **or** $index \leq 0$

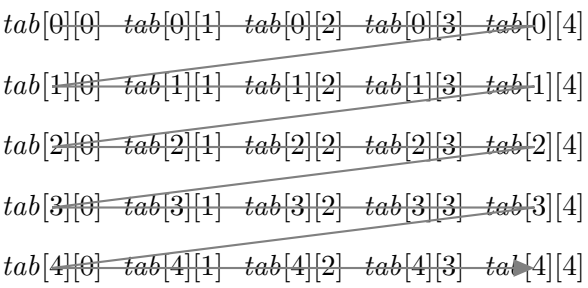
Question 3 – Parcours de tableaux à 2 dimensions (20 min – 8 pts)

Dans cette question, nous allons nous intéresser au parcours des éléments d'un tableau à deux dimensions. Dans tous les extraits de code suivants, le tableau tab possède N lignes, N colonnes et l'élément à la ligne d'indice i et colonne d'indice j est noté $tab[i][j]$. La fonction *FaireQuelqueChose* est une fonction quelconque dont il n'est pas nécessaire de connaître le fonctionnement.

Considérons le code suivant :

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

Pour $N = 5$, ce code fait en sorte que le tableau soit parcouru comme représenté ci-contre :



Bien sûr, il existe d'autres façons de parcourir les éléments du tableau. Considérons les deux exemples suivants :

Q3(a)

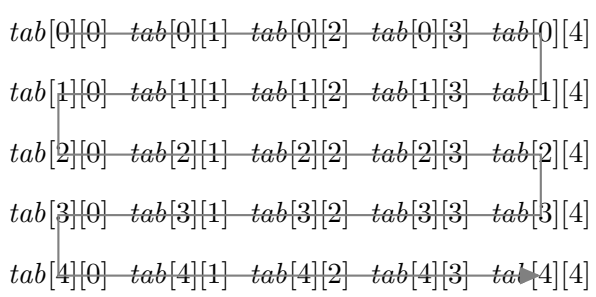
```
for (i ← 0 to N − 1 step 1)
{
  if (i % 2 = 0)                    // c'est-à-dire, si i est pair
  {
    for (j ← 0 to N − 1 step 1)
    {
      FaireQuelqueChose(tab[i][j])
    }
  }
  else
  {
    for (j ← N − 1 to 0 step −1)
    {
      FaireQuelqueChose(tab[i][j])
    }
  }
}
```

Q3(b)

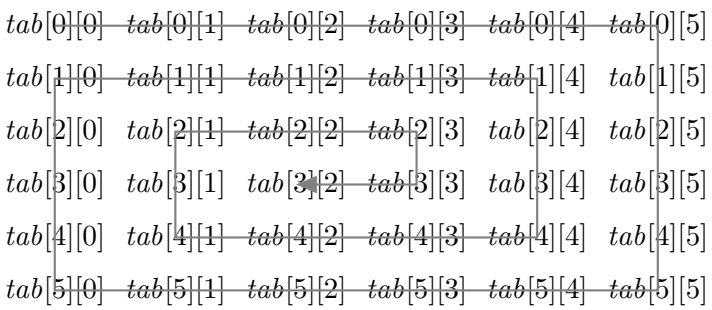
```
direction ← 0
x ← 0
y ← −1
hauteur ← N + 1
largeur ← N
while (largeur ≠ 0 and hauteur ≠ 0)
{
   pasX ← direction % 2                    // 0 si direction est pair, 1 sinon
   pasY ← (direction + 1) % 2              // 1 si direction est pair, 0 sinon
   if ((direction % 4) ≥ 2)
   {
      pasX ← −pasX
      pasY ← −pasY
   }
   nombre ← 0
   if (pasY = 0)
   {
      nombre ← largeur
      largeur ← largeur − 1
   }
   else
   {
      nombre ← hauteur
      hauteur ← hauteur − 1
   }
   for (i ← 1 to nombre − 1 step 1)
   {
      x ← x + pasX
      y ← y + pasY
      FaireQuelqueChose(tab[x][y])
   }
   direction ← (direction + 1) % 4
}
```

Dessinez sur les tableaux ci-dessous **l'ordre dans lequel les éléments sont parcourus** par les algorithmes précédents. N'oubliez pas de **dessiner une flèche** pour indiquer la direction du parcours (comme sur l'exemple).

Q3(a) avec $N = 5$



Q3(b) avec $N = 6$



Question 4 – La bibiothèque (10 min – 8 pts)

Vous souhaitez ranger les livres de votre chambre par ordre de taille (hauteur), du plus petit au plus grand. Bien que tous de tailles différentes, certains livres ont des tailles très similaires. Vous choisissez donc de trier ces livres directement sur l'étagère en n'échangeant à chaque étape que deux livres adjacents.

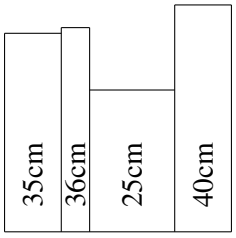


FIGURE 1 – Exemple

Par exemple, si vous deviez trier les livres ci-contre (FIGURE 1), vous pourriez inverser le deuxième et le troisième livre, et ensuite le premier et le deuxième livre afin d'avoir vos livres ordonnés (deux étapes).

En suivant cette même règle, en combien d'étapes au **minimum** pouvez-vous trier l'étagère ci-contre (FIGURE 2) ?

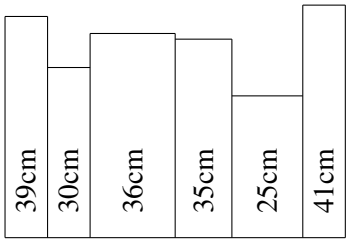


FIGURE 2 – Q4(a)

Q4(a) [3 pts] **un nombre**

8

Un ami vous propose de plutôt commencer par mesurer tous vos livres, puis de sélectionner le plus petit et de l'échanger avec le livre le plus à gauche, et de recommencer ce processus : sélectionner le deuxième plus petit livre pour l'échanger avec le deuxième le plus à gauche, etc. Pour vous prouver que cela fonctionne, il a commencé à écrire un algorithme que voici :

```
Input  : tab, un tableau de taille n d'entiers tous différents.
Output : Le tableau tab est trié par ordre croissant.

for (i ← 0 to n − 2 step 1)
{
  k ← i
  for (j ← i + 1 to n − 1 step 1)
  {
    if ( [...] )                    // (b)
    {
      k ← j
    }
  }
  if (k ≠ i)
  {
    swap(tab, i, k);               // échange les livres aux positions i et k
  }
}
```

Complétez cet algorithme afin qu'il trie correctement le tableau `tab`.

Q4(b) [5 pts] **une expression**

$tab[j] < tab[k]$ *ou* $tab[j] \leq tab[k]$ (k est l'indice de la plus petite valeur entre l'indice i (inclus) et la fin.)

Question 5 – L'enregistreur (10 min – 6 (+3) pts)

Pour ne rien rater de vos émissions favorites lors de votre séjour à l'Olympiade Internationale en Australie, vous souhaitez programmer votre propre enregistreur permettant d'enregistrer sur disque un ensemble d'émissions TV. La principale limitation de votre système est qu'il ne peut enregistrer qu'une émission à la fois.

Un enregistrement X est défini par une heure de début (notée $X.debut$) et une heure de fin ($X.fin$). Un enregistement X est en conflit avec un enregistrement Y si une partie ou l'entièreté de X a lieu pendant la durée de Y.

On vous demande d'écrire une expression qui est vraie si et seulement si deux enregistrements X et Y sont en conflit. Pour cela, vous ne pouvez utiliser que les morceaux d'expression suivants, ainsi que des "et" (**and**), des "ou" (**or**) et des parenthèses.

A) $X.debut < Y.debut$	B) $X.debut > Y.debut$	C) $X.debut < Y.fin$	D) $X.debut > Y.fin$
E) $X.fin < Y.debut$	F) $X.fin > Y.debut$	G) $X.fin < Y.fin$	H) $X.fin > Y.fin$

Par exemple, si vous pensez vérifier que X et Y sont en conflit revient à vérifier que $X.fin > Y.fin$ et ($X.debut < Y.fin$ ou $X.fin < Y.debut$), répondez "H **and** (C **or** E)".

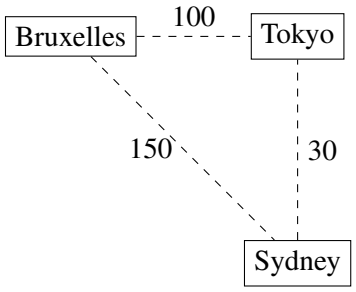
Bonus : vous obtiendrez un bonus si vous écrivez la solution la plus courte possible (c'est-à-dire, utilisant le moins d'expressions possible).

Q5[6+3 pts] **une expression**

C and F

Question 6 – Restons connectés (15 min – 14 pts)

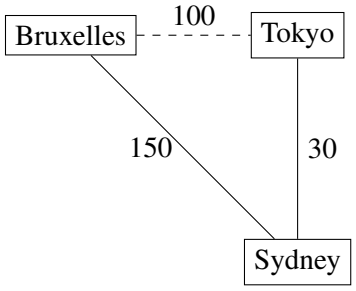
Un opérateur de télécommunication souhaite déployer un nouveau réseau à haut débit. Il a identifié une série de villes qui doivent être reliées au réseau, ainsi que les liaisons (bidirectionnelles) qu'il envisage de réaliser (avec leurs coûts). Il a représenté tout cela sur une carte comme celle-ci :



où l'on voit que le coût de relier Bruxelles à Sydney serait de 150, alors que relier Tokyo à Sydney coûte 30.

L'opérateur souhaite sélectionner certaines de ces liaisons potentielles, de façon à ce que :
- chaque ville soit connectée, directement ou indirectement à toutes les autres ;
- le coût total des liaisons sélectionnées soit aussi bas que possible.

Sur cet exemple, l'opérateur propose de sélectionner Bruxelles – Sydney et Sydney – Tokyo, pour un coût total de $150 + 30 = 180$. Remarquez que Bruxelles est bien reliée à Tokyo, indirectement, *via* Sydney :

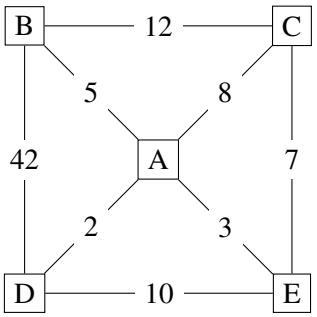


L'opérateur peut-il faire mieux ? Si oui, quel est le coût minimum qu'il peut atteindre ?

Q6(a) [2 pts] **« non » ou bien un nombre**

130

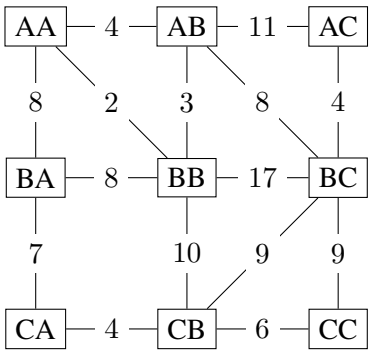
L'opérateur considère maintenant une nouvelle carte, donnée ci-dessous. Quelle est maintenant l'ensemble des liaisons qu'il faut construire pour atteindre le coût minimal tout en garantissant toujours que chaque ville est connectée, directement ou indirectement à toutes les autres ?



Q6(b) [4 pts] **une liste de connexions, de la forme : « A–B, B–C,… »**

A–B, A–E, A–D, C–E

Quel est le coût optimal que l'opérateur peut atteindre sur cette nouvelle carte ?



Q6(c) [5 pts] **un nombre**

42 (en utilisant : AA–BB, AA–BA, BA–CA, BB–AB, AB–BC, BC–AC, CA–CB, CB–CC)

De façon générale, s'il y a n villes à connecter, quel sera le *nombre de liaisons* à construire pour assurer que toutes les villes sont connectées, directement ou indirectement, à un coût minimum ?

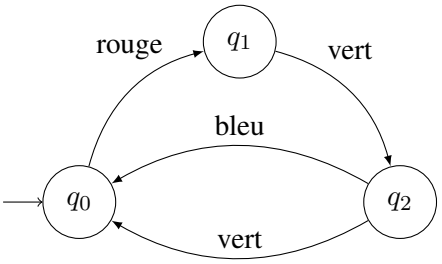
Q6(d) [3 pts] **une expression**

$n - 1$

Question 7 – La vaisselle australienne (15 min – 16 pts)

Dans un restaurant reculé du fin fond de l'Australie, Joe Bloggs, un commis de cuisine, à qui on a confié la tâche ingrate de faire la vaisselle, fait tout pour se venger. Les assiettes de ce restaurant sont rouges, vertes ou bleues, et Joe a décidé qu'il n'accepterait de laver et ranger les assiettes qu'à la condition que celles-ci lui arrivent dans un ordre bien déterminé, qu'il fixe chaque matin. Toutes les assiettes qui ne respectent pas cet ordre seront systématiquement cassées...

Afin d'expliquer de manière concise et précise l'ordre qu'il attend, Joe utilise des diagrammes constitués de cercles, appelés *états*, et de flèches appelées *transitions*. Chaque transition porte le nom d'une couleur d'assiette. Voici un exemple de diagramme :



Voici comment on interprète ces diagrammes. Les états permettent de savoir ce que Joe est prêt à accepter comme couleurs d'assiettes, à savoir les couleurs qui étiquettent les flèches *sortant* de l'état. Au début de sa journée, Joe est dans l'état *initial* (q_0 sur l'exemple), représenté par une courte flèche horizontale. Chaque fois que Joe reçoit une assiette, il regarde s'il existe une transition étiquettée par la couleur de l'assiette et partant de son état courant. Si oui, il lave et range l'assiette, et change d'état : l'état courant devient celui qui se trouve à l'extrémité de la transition. Sinon, Joe casse l'assiette et attend la suivante (l'état ne change pas).

Par exemple, avec le diagramme ci-dessus, Joe lavera la séquence d'assiettes suivante : rouge, vert, vert, rouge, vert, bleu. Mais il cassera la deuxième et la troisième assiette de la séquence : rouge, bleu, rouge, vert.

En utilisant le diagramme ci-dessus, combien d'assiettes Joe cassera-t-il dans la séquence : rouge, bleu, vert, vert, bleu, bleu, rouge ?

Q7(a) [2 pts] **un nombre**

3

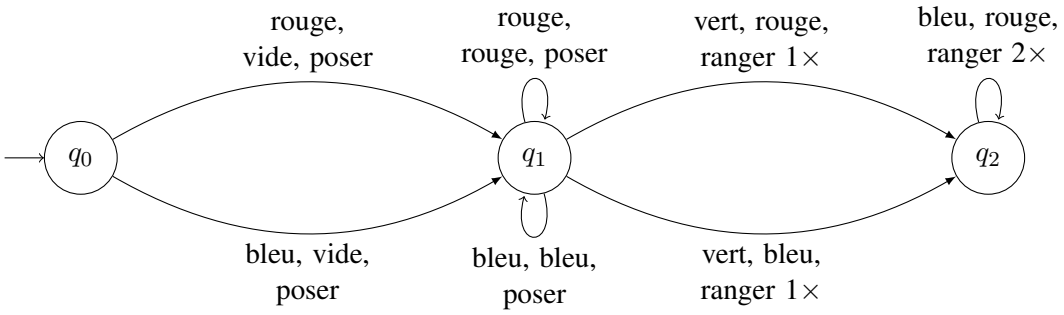
En utilisant le diagramme ci-dessus, donnez une séquence de 8 assiettes que Joe ne cassera pas, et qui ne contient pas d'assiette bleue.

Q7(b) [4 pts] **une séquence de huit couleurs (utilisez R pour rouge et V pour vert)**

R V V R V V R V

Au bout d'une semaine, les serveurs du restaurant ont bien compris comment lire les diagrammes et transmettent les assiettes de façon à ce que Joe n'en casse plus aucune. Mais Joe est malin et décide, pour rendre les choses plus compliquées, qu'il ne *rangera plus systématiquement les assiettes lavées*. Il s'autorisera parfois à poser l'assiette lavée au sommet d'une pile à côté de son évier, et n'acceptera de laver certaines assiettes qu'à la condition que l'assiette au sommet de la pile soit aussi de la bonne couleur... sacré Joe !

Les diagrammes ressemblent maintenant à ceci :



On voit maintenant que chaque transition est étiquetée par *trois informations* : c_1, c_2 et a. La *première* est une couleur (rouge, vert, bleu) et indique la couleur de l'assiette que Joe doit recevoir pour exécuter cette transition (comme dans le cas simple du début de la question). La *seconde* information est soit une couleur (rouge, vert, bleu), soit « vide » et indique ce que Joe attend de la pile d'assiette pour exécuter la transition : s'il s'agit d'une couleur, Joe veut voir cette couleur au sommet de la pile ; s'il s'agit de « vide », Joe veut que la pile soit vide. La *troisième* est une *action*, et indique ce que Joe fera avec les assiettes. Il y a trois actions possibles (pour une transition étiquetée par c_1, c_2, a) :

Action a	**Effet**
poser	Joe lave l'assiette de couleur c_1 qu'il a reçue et la pose au sommet de la pile.
ranger 1×	Joe lave l'assiette de couleur c_1 qu'il a reçue, et la range (cela ne modifie donc pas la pile).
ranger 2×	Joe lave l'assiette de couleur c_1 qu'il a reçue, la range, et range également l'assiette de couleur c_2 qu'il prend au sommet de la pile (le sommet de la pile est donc modifié).

Comme d'habitude, si Joe se trouve dans un certain état, et ne trouve pas, à partir de cet état, de transition qui satisfasse ses exigences, il casse l'assiette qu'il reçoit. Sur l'exemple ci-dessus, si on transmet à Joe des assiettes : bleu, vert, bleu, Joe lavera la première assiette et la posera sur la pile, lavera et rangera la verte, mais cassera la seconde bleu, car il exige une assiette rouge au sommet de la pile.

En utilisant ce second diagramme, combien d'assiettes Joe cassera-t-il dans la séquence suivante : rouge, bleu, vert, bleu, bleu ?

Q7(c) [2 pts] **un nombre**

2 (la seconde et la dernière)

Même question avec la séquence suivante : rouge, rouge, rouge, vert, bleu, bleu, bleu ?

Q7(d) [2 pts] **un nombre**

0

Même question avec la séquence suivante : rouge, rouge, vert, bleu, bleu, bleu ?

Q7(e) [2 pts] **un nombre**

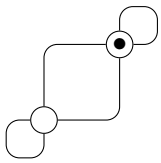
1 (la dernière car la pile est vide)

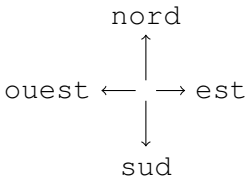
Supposons maintenant que Joe reçoive d'abord n assiettes rouges, puis *une* assiette verte, puis m assiettes bleues. Toujours en utilisant le diagramme ci-dessus, indiquez, pour chacune des conditions ci-dessous, lesquelles permettent de garantir que Joe ne casse aucune assiette (quelques soient les valeurs de n et m satisfaisant la condition) :

	Garantit que Joe ne casse rien	Ne garantit *pas* que Joe ne casse rien	Condition
Q7(f) [1pt]	☐	☒	$n < m$
Q7(g) [1pt]	☒	☐	$n > m$
Q7(h) [1pt]	☒	☐	$n = m$ et $n > 0$
Q7(i) [1pt]	☐	☒	$n + m$ est pair

Question 8 – Sous le pôle sud (25 min – 9+4 pts)

À la fin de l'année 2012, des chercheurs australiens découvrent des signes de vie extraterrestre sur terre, comme prévu par les Mayas. Pour être précis, ces chercheurs ont retrouvé plusieurs cartes décrivant des chambres et des tunnels souterrains très anciens, sous le pôle sud. La première carte a été découverte le soir de Noël, dans le *Queen Mary Land*. La carte ressemble au schéma ci-dessous, et montre deux chambres de forme circulaire, reliées par quatre tunnels :





Par ailleurs, le hasard est venu en aide aux scientifiques : ils sont parvenus, à l'aide d'un sonar, à localiser une des chambres souterraines, et même à percer un tunnel qui y mène. Maintenant, ils souhaiteraient envoyer un petit robot dans cette chambre.

Étant donné que les tunnels sont vraiment creusés très profonds, le robot doit recevoir toutes ses instructions avant d'y être envoyé. Une fois dans la chambre, il ne peut plus être contacté, si ce n'est qu'à la fin de sa mission il aura la possibilité d'envoyer, aux scientifiques restés à la surface, une courte vidéo.

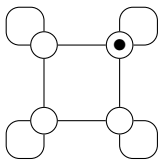
Le robot ne comprend que quatre instructions simples : `nord`, `est`, `sud` et `ouest`. Quand il reçoit, par exemple, l'instruction `nord`, il emprunte la sortie la plus au nord de la chambre dans laquelle il se trouve, suit le tunnel, et continue à rouler jusqu'à atteindre la chambre suivante. Ensuite, il exécute l'instruction suivante (les trois autres instructions fonctionnent de façon similaire, mais avec les autres directions).

Les scientifiques sont particulièrement intéressés par la chambre qui est marquée d'un rond noir sur la carte. Malheureusement, ils ne savent pas dans quelle chambre le robot sera déposé. Quelles instructions doit-on donner au robot pour s'assurer qu'il atteint la bonne chambre ?

Les chercheurs australiens sont malins, et réalisent vite qu'il existe une solution simple à leur problème. S'ils utilisent par exemple (sur la carte ci-dessus) la série d'instructions `ouest nord`, le robot finira toujours dans la bonne chambre, indépendemment de l'endroit où il a commencé. Il existe d'ailleurs d'autres solutions : les plus courtes sont les instructions (uniques) `nord` ou `est`.

Après Noël, les scientifiques ont trouvé d'autres cartes extraterrestres, et, pour trois d'entre elles, ils ont réussi à trouver une des chambres souterraines. Les nouvelles cartes sont un peu plus compliquées que la première. Pouvez-vous aider à programmer le robot correctement ?

Mac.Robertson Land

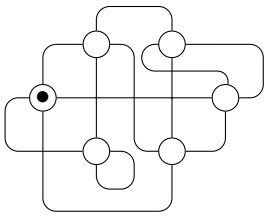


Donnez les instructions qu'il faut donner au robot pour la carte ci-contre. Après la dernière instruction, le robot doit être arrivé dans la chambre qui est marquée d'un point noir, peu importe la chambre dans laquelle il a commencé.

Q8(a) [3 pts] — **une série d'instructions**

`nord est` ***ou*** `est nord`

Prince Charles Mountains

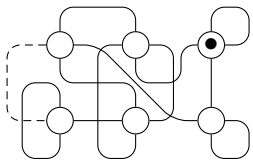


Même exercice pour cette carte. Un bonus sera accordé si la séquence est aussi courte que possible (là où des lignes se croisent, les tunnels sont en réalité superposés).

Q8(b) [3+2 pts] — **une série d'instructions**

`sud est ouest`

Wilhelm II Land



Même exercice pour cette carte. Un bonus sera également accordé si la séquence est aussi courte que possible.

Attention. Nous n'avons aucune confiance dans le tunnel le plus à gauche (en pointillé). *En aucun cas*, le robot ne pourra l'emprunter !

Q8(c) [3+2 pts] **une série d'instructions**

```
est sud sud nord ou est nord est nord
```

Question 9 – Mines anti-personnel (15 min – 12 pts)

Dans le cadre de sa mission de recherche et destruction de mines anti-personnel, la défense vous propose la mission suivante. Le service de déminage de la défense possède un équipement sophistiqué de détection des mines, le MINE SWEEPER T-1000. Cet équipement est capable de produire des cartes renseignant l'emplacement présumé des mines. Le T-1000 découpe l'espace à analyser en zones carrées et indique pour chaque zone s'il contient ou pas une mine. Sachant que les mines sont sensibles aux vibrations, il est dangereux de se déplacer dans les zones situées à proximité d'au moins une mine. La défense vous confie la tâche de déterminer automatiquement les zones dangereuses en calculant le nombre de mines à proximité de chaque zone.

Le T-1000 vous fournit sous format électronique la carte renseignant l'emplacement des mines. La carte prend la forme d'un tableau `mines` à 2 dimensions comprenant M lignes et N colonnes. Chaque cellule correspond à une zone. La cellule de ligne i et de colonne j du tableau contient la valeur 1 si une mine a été détectée dans la zone (pour rappel, les lignes et colonnes sont indexées à partir de 0). La cellule contient la valeur 0 sinon. Voici un exemple de tableau `mines`, avec $N = 2$ lignes et $M = 3$ colonnes :

0	0	**1**
0	**1**	0

Le programme ci-dessous se charge de calculer le nombre de mines à proximité de chaque zone. La sortie du programme est un tableau `mines_cnt` de même taille que le tableau `mines` donné en entrée. Chaque cellule `mines_cnt[`i`][`j`]` de ce nouveau tableau doit contenir le nombre total de mines qui se trouvent dans `mines[`i`][`j`]` et dans les 8 cellules adjacentes (ou moins si la cellule se situe le long des limites du terrain). Par exemple, le tableau ci-dessous est le contenu de `mines_cnt` que le programme doit calculer, si on lui passe le tableau `mines` donné ci-dessus :

1	2	2
1	2	2

On vous demande de compléter les parties manquantes du code pour que le programme calcule correctement le contenu de `mines_cnt` :

```
Input  : mines, un tableau de M lignes et N colonnes, dont les cases ne
         contiennent que des 0 (pas de mine) ou des 1 (une mine), avec M et N ≥ 2
Output : mines_cnt, un tableau de M lignes et N colonnes, dont toutes
         les cases sont initialisées à 0.

for (i ← 0 to M − 1 step 1)
{
  for (di ← −1 to 1 step 1)
  {
    for (dj ← 0 to 1 step 1)
    {
      if ( [...] )  // (a)
      {
        mines_cnt[i][0] ← mines_cnt[i][0] + mines[i+di][dj] ;
        mines_cnt[i][N − 1] ← mines_cnt[i][N − 1] + mines[...][...] ; // (b)
      }
    }
  }
}

for (j ← 1 to N − 2 step 1)
{
  for (di ← 0 to 1 step 1)
  {
    for (dj ← −1 to 1 step 1)
    {
      mines_cnt[0][j] ← mines_cnt[0][j] + mines[di][j+dj] ;
      mines_cnt[M − 1][j] ← mines_cnt[M − 1][j] + mines[...][...] ; // (c)
    }
  }
}

for (i ← 1 to M − 2 step 1)
{
  for (j ← 1 to N − 2 step 1)
  {
    for (di ← −1 to 1 step 1)
    {
      for (dj ← −1 to 1 step 1)
      {
        mines_cnt[i+di][j+dj] ← mines_cnt[i+di][j+dj] + mines[i+di][j+dj] ;
      }
    }
  }
}
```

Q9(a) [4 pts] **une expression**

```
( (i+di) >= 0 ) and ( (i+di) < M )
```

Q9(b) [4 pts] **les coordonnées de la case**

```
i+di, N-1-dj
```

Q9(c) [4 pts] **les coordonnées de la case**

```
M-1-di, j+dj
```