

be-OI 2018

Finale - JUNIOR
samedi 17 mars 2018

Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp

PRÉNOM :
NOM :
ÉCOLE :

O

Réservé

Olympiade belge d'Informatique (durée : 2h maximum)

Ce document est le questionnaire de la finale de l'Olympiade belge d'Informatique 2018. Il comporte 8 questions qui doivent être résolues en **2h au maximum**.

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

- Vérifiez que vous avez bien reçu la bonne série de questions (mentionnée ci-dessus dans l'en-tête):
 - Pour les élèves jusqu'en deuxième année du secondaire: catégorie **cadets**.
 - Pour les élèves en troisième ou quatrième année du secondaire: catégorie **junior**.
 - Pour les élèves de cinquième année du secondaire et plus: catégorie **senior**.
- 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** (au verso si nécessaire).
- Écrivez de façon **bien lisible** à l'aide d'un **stylo ou stylo à bille** bleu ou noir.
- Vous ne pouvez avoir que de quoi écrire avec vous; les calculatrices, GSM, ... sont **interdits**.
- Vous pouvez toujours demander des feuilles de brouillon supplémentaires à un surveillant.
- Quand vous avez terminé, **remettez la première page (avec votre nom) et les pages avec les réponses**. Vous pouvez conserver les autres.
- Tous les extraits de code de l'énoncé sont en **pseudo-code**. Vous trouverez, sur les pages suivantes, une **description** du pseudo-code que nous utilisons.
- Si vous devez répondre en code, vous devez utiliser le **pseudo-code** ou un **langage de programmation courant** (Java, C, C++, Pascal, Python, ...). Les erreurs de syntaxe ne sont pas prises en compte pour l'évaluation.

Bonne chance !

L'Olympiade Belge d'Informatique est possible grâce au soutien de nos sponsors et de nos membres:



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 (+, −, × et /), vous pouvez également utiliser l'opérateur %. 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 $a\%b = 2$.

Voici un premier exemple de code, dans lequel la variable *age* reçoit 17.

```
anneeNaissance ← 2001
age ← 2018 − anneeNaissance
```

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 // moins cher !
}
```

Parfois, quand une condition est fausse, on doit en vérifier une autre. Pour cela on peut utiliser **else if**, qui revient à exécuter un autre **if** à l'intérieur du **else** du premier **if**. Dans l'exemple suivant, il y a 3 catégories d'âge qui correspondent à 3 prix différents pour le ticket de cinéma.

```
if (age ≥ 18)
{
    prix ← 8 // Prix pour une personne majeure.
}
else if (age ≥ 6)
{
    prix ← 6 // Prix pour un enfant de 6 ans ou plus.
}
else
{
    prix ← 0 // Gratuit pour un enfant de moins de 6 ans.
}
```

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. Le tableau est de taille 3, mais l'indice 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 augmenté 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 répète 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 jusqu'à ce que $N < 10$).

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

Souvent les algorithmes seront dans un cadre et précédés d'explications. Après **Input**, on définit chacun des arguments (variables) donnés en entrée à l'algorithme. Après **Output**, on définit l'état de certaines variables à la fin de l'exécution de l'algorithme et éventuellement la valeur retournée. Une valeur peut être retournée avec l'instruction **return**. Lorsque cette instruction est exécutée, l'algorithme s'arrête et la valeur donnée est retournée.

Voici un exemple en reprenant le calcul de la somme des éléments d'un tableau.

```
Input : tab, un tableau de  $N$  nombres.
          $N$ , le nombre d'éléments du tableau.
Output : sum, la somme de tous les nombres contenus dans le tableau.

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

Remarque: dans ce dernier exemple, la variable i est seulement utilisée comme compteur pour la boucle **for**. Il n'y a donc aucune explication à son sujet ni dans **Input** ni dans **Output**, et sa valeur n'est pas retournée.

Question 1 – Propagation

Présentation

Le tableau ci-dessous représente l'état à un moment donné d'une rangée d'organismes unicellulaires. Chaque cellule est représentée par une case noire si elle est vivante ou par une case blanche si elle est morte.

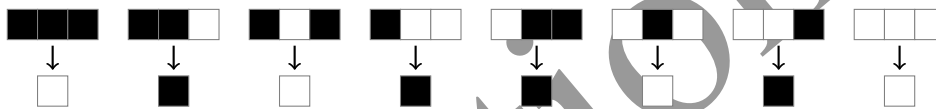


L'état d'une cellule (vivante ou morte) évolue d'une génération à l'autre par des règles très simples qui dépendent seulement de son état et de l'état de ses deux voisins (de gauche et de droite) à la génération précédente. Pour les extrémités, il faut considérer que le tableau est complété à gauche et à droite par des cellules qui restent toujours mortes.

Par exemple, une règle pourrait être:

- Une cellule qui a exactement une seule cellule voisine vivante devient (ou reste) vivante.
- Sinon la cellule meurt ou reste morte.

Cette règle peut être représentée comme suit:



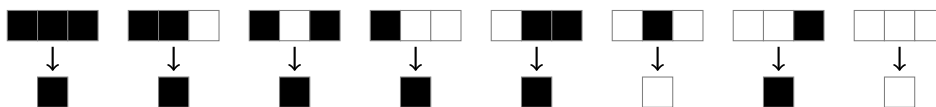
En appliquant cette règle, la 2^e génération de notre exemple devient:



Q1(a) [2 pts]	<p>Quelle sera la 3^e génération en appliquant cette règle ? Cherchez la solution ici avant de la recopier au propre sur la page des réponses.</p> <div style="border: 1px solid black; height: 20px; width: 100%;"></div>
<p>Solution: </p>	

Une autre règle

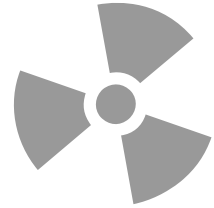
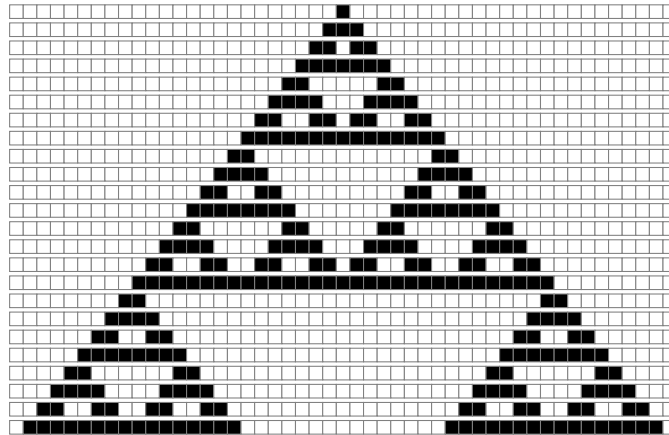
Si la nouvelle règle de génération est la suivante:



Et si la situation de départ (1^{re} génération) est:



Voici la représentation graphique d'une évolution suivant encore une autre règle :



Q1(e) [2 pts]	<p>Quelle règle donne ce résultat ? Cherchez la solution ici avant de la recopier au propre sur la page des réponses.</p>
Solution:	

Question 2 – Football

Lors d'un championnat de football chaque équipe rencontre chaque autre 2 fois : une fois à domicile et l'autre fois en déplacement chez l'adversaire. Il y a n équipes, numérotées de 0 à $n - 1$ et les résultats des matchs sont enregistrés dans 2 tableaux $n \times n$.

Lorsque l'équipe i reçoit l'équipe j , le nombre de buts marqués par l'équipe i est enregistré dans $A[i][j]$ et le nombre de buts marqués par l'équipe j est enregistré dans $B[i][j]$. Cela n'a de sens que si $i \neq j$ (car une équipe ne peut s'affronter elle-même), mais les tableaux ont été initialisés afin que pour chaque numéro i on ait $A[i][i] = B[i][i] = 0$.

Voici par exemple les résultats d'un championnat pour $n = 4$ équipes.

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

A[[]]

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

B[[]]

$A[2][3] = 4$ et $B[2][3] = 0$, donc l'équipe 2 a gagné 4 à 0 quand elle a reçu l'équipe 3.
 $A[0][1] = B[0][1] = 2$, donc l'équipe 1 a fait match-nul 2 à 2 en déplacement chez l'équipe 0.

Le championnat est terminé et tous les résultats de tous les matchs ont été encodés. Complétez les algorithmes suivants en remplaçant les \dots pour calculer les résultats demandés. Dans chaque algorithme, vous pouvez utiliser la variable n qui contient le nombre d'équipes, et les tableaux $A[[]]$ et $B[[]]$ qui contiennent les résultats.

Algorithme 1X2 : Analyser les résultats d'un match et retourner "1" si l'équipe qui joue à domicile a gagné, "2" si c'est l'équipe en déplacement qui a gagné et "X" s'il y a eu match-nul.

Input : i le numéro de l'équipe qui joue à domicile, $0 \leq i < n$.
 j le numéro de l'équipe qui joue en déplacement, $0 \leq j < n$.
Output : "1", "2" ou "X" en fonction du gagnant.

```

if ( ... ) // (a)
{
    return "1"
}
else if ( ... ) // (b)
{
    return "X"
}
else
{
    return "2"
}
    
```

Q2(a) [2 pts]	Quelle est l'expression (a) dans l'algorithme 1X2 ?
Solution: $A[i][j] > B[i][j]$ (autre solution: $B[i][j] < A[i][j]$)	

Q2(b) [2 pts]	Quelle est l'expression (b) dans l'algorithme 1X2 ?
Solution: $A[i][j] = B[i][j]$	

Algorithme Buts : Calculer le nombre total de buts marqués moins le nombre total de buts encaissés par une équipe lors du championnat.

Input : k , le numéro d'une équipe, $0 \leq k < n$.
Output : g , le nombre total de buts marqués moins le nombre total de buts encaissés par l'équipe k durant le championnat.

```

g ← 0
for (i ← 0 to n-1 step 1)
{
    g ← ...           //(c)
}
return g

```

Q2(c) [4 pts]

Quelle est l'expression (c) dans l'algorithme Buts ?

Solution: $g + A[k][i] - B[k][i] + B[i][k] - A[i][k]$

Algorithme Points : Calculer le nombre total de points obtenus par une équipe lors du championnat. Comme d'habitude on accorde 3 points pour chaque victoire, 1 point pour chaque match-nul et 0 point pour chaque défaite.

Input : k , le numéro d'une équipe, $0 \leq k < n$.
Output : p , le nombre de points gagnés par l'équipe k durant le championnat.

```

p ← 0
for (i ← 0 to n-1 step 1)
{
    if ( A[k][i] ... )           //(d)
    {
        ...                       //(e)
    }
    else if ( A[k][i] ... )       //(f)
    {
        p ← p + 1
    }

    if ( ... )                   //(g)
    {
        ...                       //(h)
    }
    else if ( ... )               //(i)
    {
        p ← p + 1
    }
}
return ...                       //(j)

```

Q2(d) [1 pt]

Comment compléter l'expression (d) dans l'algorithme Points ?

Solution: $> B[k][i]$

Q2(e) [1 pt]	Quelle est l'instruction (e) dans l'algorithme Points ?
Solution: $p \leftarrow p + 3$	
Q2(f) [1 pt]	Comment compléter l'expression (f) dans l'algorithme Points ?
Solution: $= B[k][i]$	
Q2(g) [1 pt]	Quelle est l'expression (g) dans l'algorithme Points ?
Solution: $B[i][k] > A[i][k]$ (autre solution: $A[i][k] < B[i][k]$)	
Q2(h) [1 pt]	Quelle est l'instruction (h) dans l'algorithme Points ?
Solution: $p \leftarrow p + 3$	
Q2(i) [1 pt]	Quelle est l'expression (i) dans l'algorithme Points ?
Solution: $B[i][k] = A[i][k]$	
Q2(j) [2 pts]	Quelle est l'expression (j) dans l'algorithme Points ?
Solution: $p - 2$ (en effet, 2 matchs nuls sont comptés par erreur car $A[k][k] = B[k][k]$)	










Question 3 – Jeu de dés

Un programme doit vérifier si certaines combinaisons sont obtenues en lançant 5 dés. Le programme tire au hasard 5 nombres compris entre 1 et 6 et les stocke en ordre croissant dans les cases d'indices 1 à 5 du tableau $d[]$.

Par exemple, si les dés générés sont 5, 1, 6, 5 et 1 alors $d[1] = 1$, $d[2] = 1$, $d[3] = 5$, $d[4] = 5$ et $d[5] = 6$.

Donc on aura toujours $1 \leq d[1] \leq d[2] \leq d[3] \leq d[4] \leq d[5] \leq 6$.

Le tableau suivant contient les combinaisons à tester:

Combinaison	Description	Exemple
<i>Yams</i>	5 dés identiques	
<i>Carré</i>	Au moins 4 dés identiques	
<i>Full</i>	3 dés identiques et les 2 autres dés identiques	
<i>Brelan</i>	Au moins 3 dés identiques	
<i>Paire</i>	Au moins 2 dés identiques	
<i>Double paire</i>	2 Paires sans chevauchement	
<i>Grande suite</i>	5 valeurs successives	
<i>Petite suite</i>	Au moins 4 valeurs successives	
<i>Rien</i>	Rien de ce qui précède	

Attention, un même résultat peut donner plusieurs combinaisons simultanément. Un *Yams* est aussi un *Carré*, un *Full*, un *Brelan*, une *Double paire* et une *Paire*. De même toute *Grande suite* est aussi une *Petite suite*, et l'exemple de *Petite suite* donné dans le tableau est aussi une *Paire*.

Les instructions de la plupart des langages informatiques sont des mots anglais. En particulier on utilise **true** (vrai), **false** (faux), **or** (ou), **and** (et).

Si un test contient plusieurs conditions reliées par des **or**, le test est **true** si au moins une des conditions est **true**.

Si un test contient plusieurs conditions reliées par des **and**, le test est **true** si toutes les conditions sont **true**.

Par exemple, si $taille = 180$ et $age = 16$:

$$\begin{array}{l}
 ((taille = 155) \text{ or } (age = 12)) \text{ est } \mathbf{false} \quad \left| \quad ((taille = 155) \text{ and } (age = 12)) \text{ est } \mathbf{false} \\
 ((taille = 155) \text{ or } (age = 16)) \text{ est } \mathbf{true} \quad \left| \quad ((taille = 155) \text{ and } (age = 16)) \text{ est } \mathbf{false} \\
 ((taille = 180) \text{ or } (age = 16)) \text{ est } \mathbf{true} \quad \left| \quad ((taille = 180) \text{ and } (age = 16)) \text{ est } \mathbf{true}
 \end{array}$$

S'il y a des parenthèses, il faut comme d'habitude commencer par évaluer le contenu des parenthèses les plus intérieures.

Pour marquer les points aux questions ci-dessous, vous devez cocher TOUTES les combinaisons pour lesquelles le test est toujours **true**, et ne cocher AUCUNE combinaison pour laquelle le test est parfois **false**.

Par exemple, si un test est **true** pour chaque *Brelan* et chaque *Full*, alors il faut cocher les cases *Brelan* et *Full*. Si le test est **false** pour au moins un *Carré*, alors il ne faut pas cocher la case *Carré*.

Q3(a) [1 pt]	$(d[1] = d[2]) \text{ or } (d[2] = d[3]) \text{ or } (d[3] = d[4]) \text{ or } (d[4] = d[5])$ est toujours true pour ... <input checked="" type="checkbox"/> <i>Yams</i> <input checked="" type="checkbox"/> <i>Carré</i> <input checked="" type="checkbox"/> <i>Full</i> <input checked="" type="checkbox"/> <i>Brelan</i> <input checked="" type="checkbox"/> <i>Paire</i> <input checked="" type="checkbox"/> <i>Double paire</i> <input type="checkbox"/> <i>Grande suite</i> <input type="checkbox"/> <i>Petite suite</i> <input type="checkbox"/> <i>Rien</i>
Solution: les bonnes cases sont cochées.	

Q3(b) [1 pt]	$(d[1] = d[3]) \text{ or } (d[2] = d[4]) \text{ or } (d[3] = d[5])$ est toujours true pour ... <input checked="" type="checkbox"/> <i>Yams</i> <input checked="" type="checkbox"/> <i>Carré</i> <input checked="" type="checkbox"/> <i>Full</i> <input checked="" type="checkbox"/> <i>Brelan</i> <input type="checkbox"/> <i>Paire</i> <input type="checkbox"/> <i>Double paire</i> <input type="checkbox"/> <i>Grande suite</i> <input type="checkbox"/> <i>Petite suite</i> <input type="checkbox"/> <i>Rien</i>
Solution: les bonnes cases sont cochées.	

Q3(c) [2 pts]	$((d[1] = d[2]) \text{ and } (d[3] = d[5])) \text{ or } ((d[1] = d[3]) \text{ and } (d[4] = d[5]))$ est toujours true pour ... <input checked="" type="checkbox"/> <i>Yams</i> <input type="checkbox"/> <i>Carré</i> <input checked="" type="checkbox"/> <i>Full</i> <input type="checkbox"/> <i>Brelan</i> <input type="checkbox"/> <i>Paire</i> <input type="checkbox"/> <i>Double paire</i> <input type="checkbox"/> <i>Grande suite</i> <input type="checkbox"/> <i>Petite suite</i> <input type="checkbox"/> <i>Rien</i>
Solution: les bonnes cases sont cochées.	

Q3(d) [2 pts]	$(d[1] \neq d[2]) \text{ and } (d[2] \neq d[3]) \text{ and } (d[3] \neq d[4]) \text{ and } (d[4] \neq d[5]) \text{ and } (d[5] - d[1] = 4)$ est toujours true pour ... <input type="checkbox"/> <i>Yams</i> <input type="checkbox"/> <i>Carré</i> <input type="checkbox"/> <i>Full</i> <input type="checkbox"/> <i>Brelan</i> <input type="checkbox"/> <i>Paire</i> <input type="checkbox"/> <i>Double paire</i> <input checked="" type="checkbox"/> <i>Grande suite</i> <input type="checkbox"/> <i>Petite suite</i> <input type="checkbox"/> <i>Rien</i>
Solution: les bonnes cases sont cochées.	

Q3(e) [2 pts]	$(d[1] = 1) \text{ and } (d[2] = 2) \text{ and } (d[2] \neq d[3]) \text{ and } (d[3] \neq d[4]) \text{ and } (d[4] = 5) \text{ and } (d[5] = 6)$ est toujours true pour ... <input type="checkbox"/> <i>Yams</i> <input type="checkbox"/> <i>Carré</i> <input type="checkbox"/> <i>Full</i> <input type="checkbox"/> <i>Brelan</i> <input type="checkbox"/> <i>Paire</i> <input type="checkbox"/> <i>Double paire</i> <input type="checkbox"/> <i>Grande suite</i> <input type="checkbox"/> <i>Petite suite</i> <input checked="" type="checkbox"/> <i>Rien</i>
Solution: les bonnes cases sont cochées.	

Q3(f) [2 pts]

$\left((d[1] = d[2]) \text{ or } ((d[2] = d[3]) \text{ and } (d[4] = d[5])) \right) \text{ and } ((d[3] = d[4]) \text{ or } (d[4] = d[5]))$
est toujours **true** pour ...

- Yams* *Carré* *Full* *Brelan* *Paire*
 Double paire *Grande suite* *Petite suite* *Rien*

Solution: les bonnes cases sont cochées.



Solutions



Question 4 – La fourmi de Langton

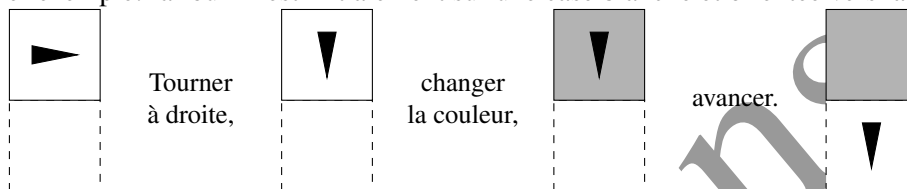
Les cases d'une grille peuvent être blanches ou grises. Une fourmi est placée sur une case et se déplace d'une case à la fois vers la gauche, la droite, le haut ou le bas selon ces règles:

- Si la fourmi est sur une case blanche, elle fait un quart de tour vers la droite, change la couleur de la case (qui devient grise) et avance d'une case.
- Si la fourmi est sur une case grise, elle fait un quart de tour vers la gauche, change la couleur de la case (qui devient blanche) et avance d'une case.

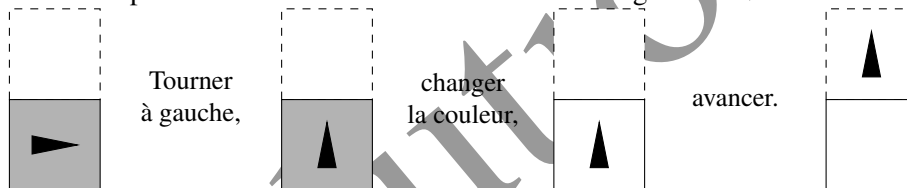
Dans les exemples suivants, la fourmi est représentée par le triangle noir, l'angle le plus aigu est la tête et donne l'orientation.

La case dessinée avec des tirets est grise ou blanche, sa couleur n'a pas d'importance.

Premier exemple: la fourmi est initialement sur une case blanche et orientée vers la droite.



Deuxième exemple: la fourmi est initialement sur une case grise et orientée vers la droite.



On vous demande de compléter l'algorithme page suivante qui simule le comportement de la fourmi pendant k étapes sur une grille $n \times m$ stockée dans le tableau $grid[][]$ contenant uniquement des 0 (couleur blanche) et des 1 (couleur grise).

- Le premier indice du tableau est le numéro de colonne, variant de 0 à gauche jusqu'à $n - 1$ à droite.
- Le second indice du tableau est le numéro de ligne, variant de 0 en bas jusqu'à $m - 1$ en haut.

Par exemple si $grid[3][7] = 1$, cela veut dire que la case à l'intersection de la colonne 3 (la quatrième en partant de la gauche) et de la ligne 7 (la huitième en partant du bas) est grise.

La position de la fourmi est enregistrée dans les variables x pour la colonne et y pour la ligne.

La direction de la fourmi est codée dans les variables dx et dy qui peuvent chacune valoir 0, 1 ou -1 .

La variable dx indique de combien le numéro de colonne de la fourmi change si elle avance.

De même dy indique de combien le numéro de ligne de la fourmi change si elle avance.

- Si la fourmi est orientée vers le haut alors $dx = 0$ et $dy = 1$,
- si la fourmi est orientée vers la droite alors $dx = 1$ et $dy = 0$,
- si la fourmi est orientée vers le bas alors $dx = 0$ et $dy = -1$,
- si la fourmi est orientée vers la gauche alors $dx = -1$ et $dy = 0$.

La grille est cylindrique:

- La fourmi peut avancer au-delà du bord droit ou du bord gauche. Si elle le fait, elle réapparaît sur la même ligne par le bord d'en face.
- Par contre la fourmi ne peut franchir les bords du bas et du haut. Si elle essaie, elle reste bloquée sur sa case et fait demi-tour (180°).

Complétez l'algorithme en remplaçant chaque . . . par une expression ou une instruction.

Input : k , un entier positif, le nombre d'étapes à simuler.
 $grid$, un tableau à 2 dimensions contenant uniquement des 0 et des 1.
 n , le nombre de colonnes du tableau $grid$.
 m , le nombre de lignes du tableau $grid$.
 x , la colonne initiale de la fourmi, $0 \leq x \leq n-1$.
 y , la ligne initiale de la fourmi, $0 \leq y \leq m-1$.
 dx et dy , 2 valeurs (0, 1 ou -1), la direction initiale de la fourmi.

Output : le tableau $grid$ a été modifié après k déplacements de la fourmi.

```

for (i ← 1 to ... step 1)           //(a)
{
  temp ← dx
  if (grid[x][y] = ... )           //(b)
  {
    dx ← dy
    dy ← -temp
  }
  else
  {
    dx ← ...                       //(c)
    dy ← ...                       //(d)
  }

  grid[x][y] ← 1- ...              //(e)

  x ← ...                          //(f)
  if (x < 0 )
  {
    x ← ...                        //(g)
  }
  else if (x = ... )               //(h)
  {
    ...                            //(i)
  }

  ...                              //(j)
  if ((y < ... ) or (y = ... ))    //(k) (l)
  {
    y ← ...                        //(m)
    dy ← ...                       //(n)
  }
}
return ← grid

```

Q4(a) [1 pt]	Quelle est l'expression (a) ?
Solution: k	
Q4(b) [1 pt]	Quelle est l'expression (b) ?
Solution: 0	
Q4(c) [1 pt]	Quelle est l'expression (c) ?
Solution: -dy	
Q4(d) [1 pt]	Quelle est l'expression (d) ?
Solution: temp	
Q4(e) [1 pt]	Quelle est l'expression (e) ?
Solution: grid[x][y]	
Q4(f) [1 pt]	Quelle est l'expression (f) ?
Solution: $x + dx$	
Q4(g) [1 pt]	Quelle est l'instruction (g) ?
Solution: $x \leftarrow n - 1$	
Q4(h) [1 pt]	Quelle est l'expression (h) ?
Solution: n	
Q4(i) [1 pt]	Quelle est l'instruction (i) ?
Solution: $x \leftarrow 0$	
Q4(j) [1 pt]	Quelle est l'instruction (j) ?
Solution: $y \leftarrow y + dy$	
Q4(k) [1 pt]	Quelle est l'expression (k) ?
Solution: 0	
Q4(l) [1 pt]	Quelle est l'expression (l) ?
Solution: m	
Q4(m) [1 pt]	Quelle est l'expression (m) ?
Solution: $y - dy$	

Q4(n) [1 pt]	Quelle est l'expression (n) ?
--------------	-------------------------------

Solution: -dy

 Solutions 

Question 5 – Combien de chemins ?

Les dessins ci-dessous montrent des salles avec des dalles carrées (cases blanches) et des obstacles (cases grises). Un robot (cercle noir) est sur la case en haut à gauche et doit se déplacer jusqu'à la cible (cercle blanc) qui est sur la case en bas à droite. Le robot est endommagé et à chaque étape il peut seulement avancer d'une case vers le bas ou vers la droite. Il ne peut jamais aller vers la gauche, ni vers le haut. Le robot doit rester sur les cases blanches, il ne peut pas traverser les obstacles gris. Dans chaque cas, combien de chemins différents le robot peut-il emprunter pour rejoindre la cible ?

Explications: Il y a plusieurs façons de compter les chemins. Un algorithme DP (programmation dynamique) permet de le faire dans tous les cas et devient indispensable dans les cas difficiles. Partir de la cible et progresser vers le haut et la gauche en notant dans chaque case combien de chemins différents mènent de cette case à la cible, ce nombre est la somme des nombres dans sa voisine de droite et dans sa voisine du dessous. La réponse est le nombre marqué dans la case de départ du robot.

Q5(a) [2 pts] Combien de chemins différents le robot peut-il emprunter ?

6	6	6	3	3	1	1		
		3		2		1		
		3	2	2	1	1		
		1		1		1		
		1	1	1	1	1	1	1

Solution: 6

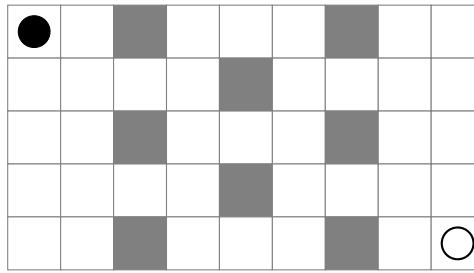
Q5(b) [2 pts] Combien de chemins différents le robot peut-il emprunter ?

8	6	5	5					
2	1		5	4	4			
1	1	1	1		4	3	3	1
			1	1	1		2	1
					1	1	1	1

Solution: 8

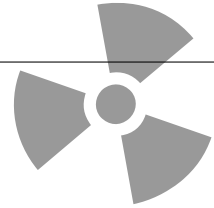
Q5(c) [2 pts]

Combien de chemins différents le robot peut-il emprunter ?



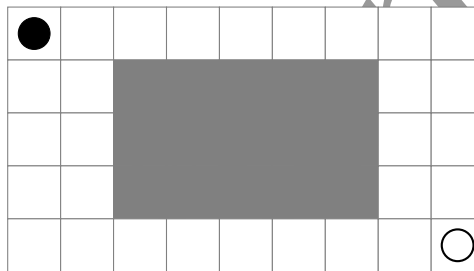
4	2								
2	2	2	2						
			2	2	2				
					2	2	2	1	
								1	1

Solution: 4



Q5(d) [2 pts]

Combien de chemins différents le robot peut-il emprunter ?

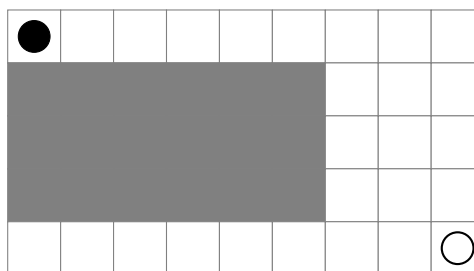


10	6	5	5	5	5	5	5	1	
4	1							4	1
3	1							3	1
2	1							2	1
1	1	1	1	1	1	1	1	1	1

Solution: 10

Q5(e) [2 pts]

Combien de chemins différents le robot peut-il emprunter ?



15	15	15	15	15	15	15	5	1	
							10	4	1
							6	3	1
							3	2	1
							1	1	1

Solution: 15

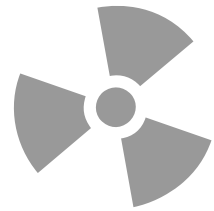
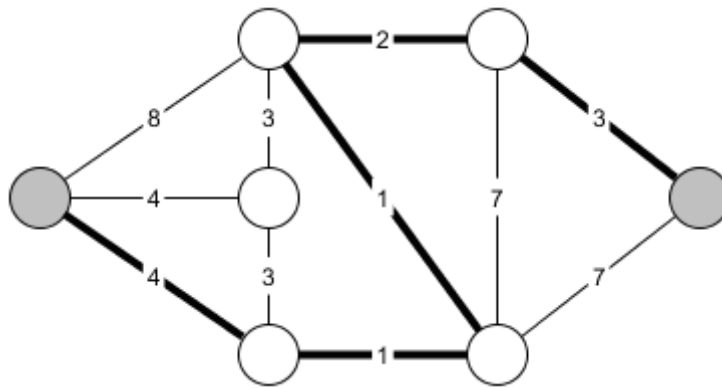
Question 6 – Plus court chemin

Un graphe est un ensemble de sommets (les cercles) et d'arêtes (les lignes) entre les sommets. Dans cette tâche, on examine des graphes où chaque arête a une longueur (notée sur l'arête). La longueur d'une arête n'est pas liée à la longueur de la ligne avec laquelle elle est dessinée.

Un chemin entre 2 sommets est une séquence d'arêtes qu'on peut suivre pour passer d'un sommet à l'autre.

La longueur d'un chemin est la somme des longueurs des arêtes qui constituent ce chemin.

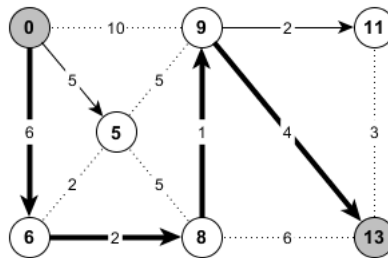
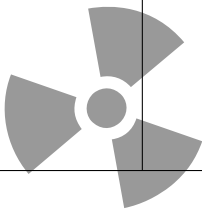
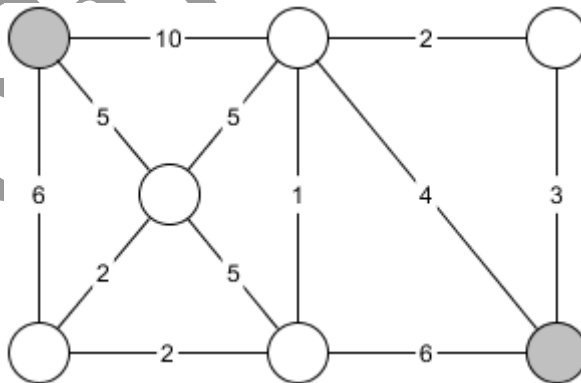
La plus courte distance entre deux sommets est la longueur du plus court chemin entre eux.



Dans l'exemple ci-dessus, on a tracé en traits épais le plus court chemin entre les 2 sommets gris. La plus courte distance entre les sommets gris est égale à la longueur de ce chemin, c'est à dire $4 + 1 + 1 + 2 + 3 = 11$.

Q6(a) [4 pts]

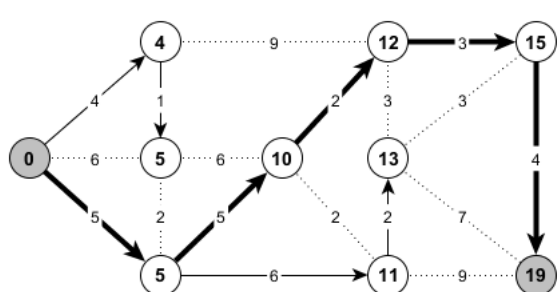
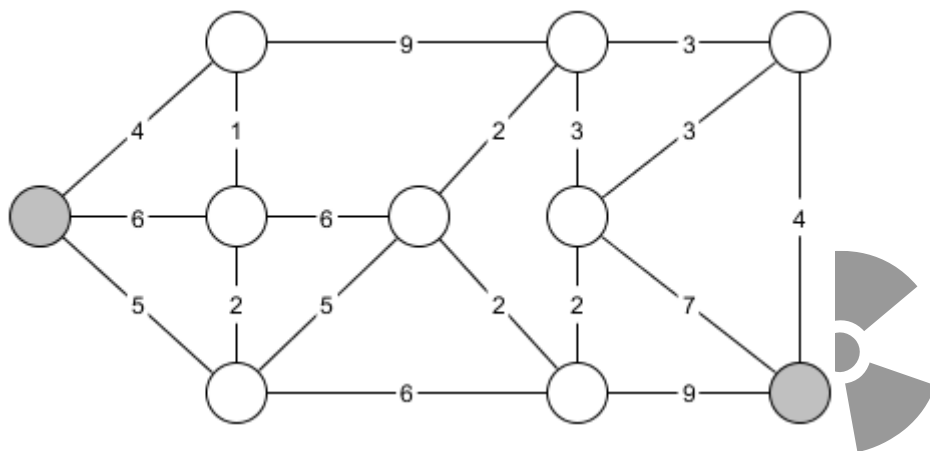
Quelle est la plus courte distance entre les 2 sommets gris de ce graphe à 7 sommets ?



Solution: 13

Q6(b) [4 pts]

Quelle est la plus courte distance entre les 2 sommets gris de ce graphe à 10 sommets ?



Solution: 19

Explications: On peut résoudre ces problèmes avec l'algorithme de Dijkstra. Partir d'un des sommets gris (par exemple celui de gauche) et noter au fur et à mesure dans chaque sommet la plus courte distance au sommet de départ. Traiter d'abord les sommets les plus proches et mettre à jour les distances chaque fois qu'une plus courte est trouvée. Certaines arêtes (en pointillés) ne sont jamais utilisées dans les plus courts chemins, les autres arêtes (avec une flèche) forment un arbre (il n'y a pas de chemin en boucle) qui recouvre le graphe. Le chemin le plus court entre les 2 sommets gris est en traits gras. La réponse est la distance inscrite dans le sommet gris de droite.

Question 7 – Photocopie de codes-barres

Vous êtes employé dans un centre d'étiquetage d'articles de supermarché et votre boulot est de photocopier des codes-barres. Ils se composent d'un certain nombre de cases blanches et noires mises les unes à la suite des autres. En voici un exemple:



Suite à de malencontreux incidents, votre photocopieuse voit un peu flou. Elle peut commettre les erreurs suivantes: si dans l'original, une case est entourée de deux cases de la couleur opposée, elle peut adopter cette couleur dans la copie. En ce qui concerne les cases aux extrémités du code-barres, elles peuvent changer de couleur si la seule case adjacente est de la couleur opposée dans l'original. Dans tous les autres cas, la couleur originale est conservée.

Dans l'exemple précédent, la 2^e case, qui est blanche, peut devenir noire car la 1^{re} et la 3^e sont toutes deux noires:



mais la 4^e ne peut pas devenir noire car la 5^e est blanche.

La 6^e case, qui est noire, peut devenir blanche:



Attention, plusieurs erreurs peuvent survenir en même temps, de telle sorte que l'exemple du début pourrait devenir:



Prédiction

Dans les questions suivantes, on vous donne un code-barres original et on vous demande combien de copies différentes pourraient en résulter.


Par exemple, étant donné l'original ci-dessous:





les seules copies qu'il pourrait donner sont:




donc la réponse est 2.

Q7(a) [1 pt]	Étant donné l'original suivant, combien de copies différentes sont possibles ? 
Solution: 2	

Q7(b) [1 pt]	Étant donné l'original suivant, combien de copies différentes sont possibles ? 
Solution: 1	

Q7(c) [2 pts]	Étant donné l'original suivant, combien de copies différentes sont possibles ? 
Solution: 4	

Q7(d) [2 pts]	Étant donné l'original suivant, combien de copies différentes sont possibles ? 
Solution: 32	

Explication: Chaque case isolée (entourée de couleurs différentes) multiplie le nombre de possibilités par 2.

Rétrospective

Dans les questions suivantes, on vous donne une copie de codes-barres et on vous demande combien de code-barres originaux auraient pu la produire.

Par exemple, étant donné la copie ci-dessous:



les seuls originaux qui auraient pu la produire sont:



donc la réponse est 2.

Q7(e) [1 pt]	<p>Étant donné la copie suivante, combien d'originaux différents auraient pu la produire ?</p> <div style="text-align: center;"> </div>
Solution: 3	

Q7(f) [1 pt]	<p>Étant donné la copie suivante, combien d'originaux différents auraient pu la produire ?</p> <div style="text-align: center;"> </div>
Solution: 2	

Q7(g) [2 pts]	<p>Étant donné la copie suivante, combien d'originaux différents auraient pu la produire ?</p> <div style="text-align: center;"> </div>
Solution: 3	

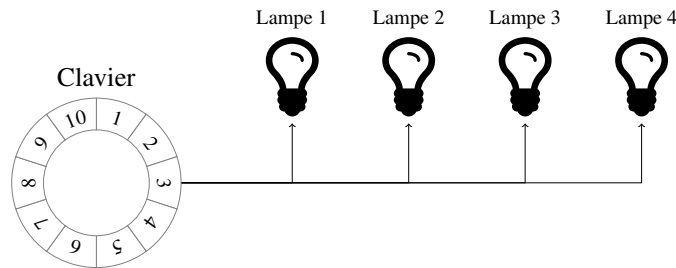
Q7(h) [2 pts]	<p>Étant donné la copie suivante, combien d'originaux différents auraient pu la produire ?</p> <div style="text-align: center;"> </div>
Solution: 3	

Explication: La première intuition à avoir est d'essayer de résoudre le problème avec un DP. Par exemple, on peut définir $dp[a][b][i]$ comme le nombre de possibilités pour les i premières cases du code-barres original (préfixe $[1, i]$), qui se terminent avec la couleur a , et en supposant que la couleur suivante est b . Il suffit alors d'écrire la récurrence soigneusement et on peut remplir un tableau assez rapidement. C'est assez rapide pour résoudre la plupart des sous-questions.

En observant les récurrences ou les valeurs, on peut alors facilement découvrir que les résultats suivent la suite de Fibonacci, en avançant d'une étape à chaque fois que deux cases consécutives sont de la même couleur. Ça rend les calculs suffisamment rapides pour résoudre toutes les sous-questions en peu de temps.

Question 8 – Code à lampes

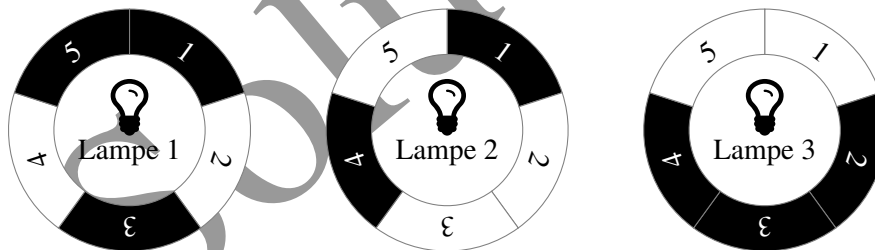
Vous travaillez aux services secrets et planchez sur un nouveau système de communication à longue distance qui doit permettre de transmettre les nombres de 1 à m . L'émetteur, schématisé ci-dessous, est constitué d'une ligne de n lampes numérotées de 1 à n et d'un clavier circulaire à m touches numérotées dans l'ordre de 1 à m .



Exemple avec $n = 4$ lampes et $m = 10$ nombres.

Pour transmettre un nombre il suffit d'appuyer sur la touche correspondante du clavier. Cela allume certaines lampes (peut-être aucune ou toutes) et éteint les autres. La liste des effets de chaque touche sur chaque lampe s'appelle une *configuration*. Deux touches ne peuvent pas allumer (et éteindre) exactement les mêmes lampes afin qu'il soit possible de les distinguer en observant les lampes. Une configuration est dite *correcte* si cette condition est respectée.

Exemple : l'illustration ci-dessous montre une *configuration* pour $m = 5$ nombres et $n = 3$ lampes. On a dessiné le clavier autour de chaque lampe en noircissant certaines touches. Appuyer sur une touche allume les lampes pour lesquelles la touche est blanche et éteint les lampes pour lesquelles la touche est noircie.



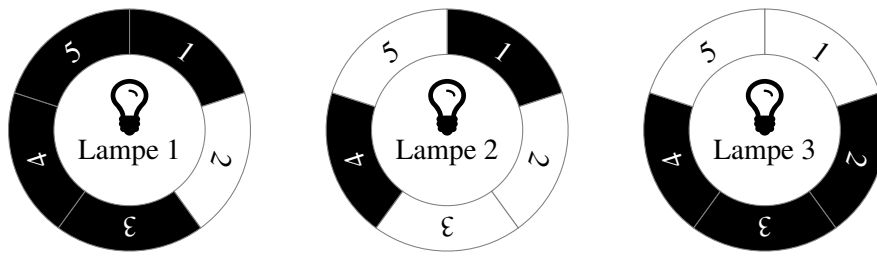
Configuration correcte avec $n = 3$ lampes, $m = 5$ nombres et 5 groupes blancs.

Dans cet exemple, la touche 1 éteint les lampes 1 et 2 et allume la lampe 3. La touche 2 allume les lampes 1 et 2 et éteint la lampe 3. Les 5 touches ont des effets différents sur les lampes, donc cette configuration est *correcte*.

Des **touches blanches adjacentes** constituent un *groupe blanc*. Dans l'exemple ci-dessus les touches 2 et 3 de la lampe 2 forment un *groupe blanc*. Les touches 1 et 5 de la lampe 3 forment également un *groupe blanc*. On considère également qu'une **touche blanche isolée** forme un *groupe blanc* d'un seul élément. La lampe 1 a donc deux *groupes blancs* (touche 2 seule, et touche 4 seule), la lampe 2 a aussi deux *groupes blancs*, et la lampe 3 n'a qu'un seul *groupe blanc*.

La construction du circuit électronique actionnant les lampes est plus simple quand il y a moins de *groupes blancs*. Une *configuration optimale* est une combinaison *correcte* qui **minimise le nombre total de groupes blancs**.

La configuration de l'exemple précédent, avec au total 5 groupes blancs, n'est pas *optimale*, comme le prouve la *configuration correcte* suivante qui a seulement 4 *groupes blancs*.



Une configuration correcte pour $n = 3$ et $m = 5$ avec seulement 4 groupes blancs.

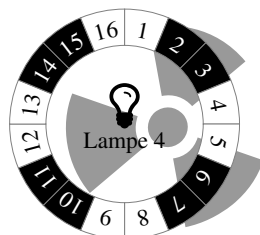
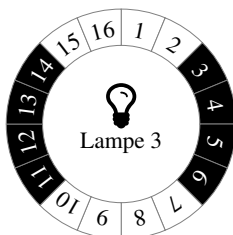
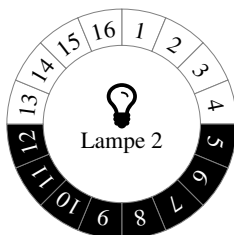
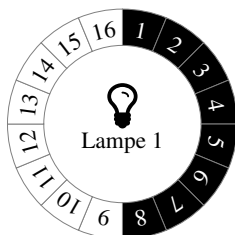
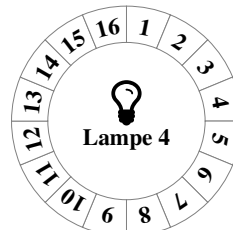
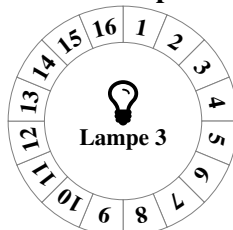
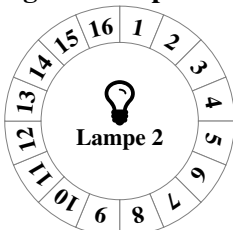
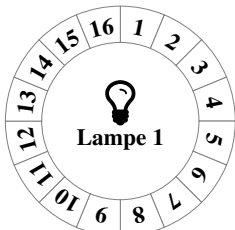
Pour chacune des questions suivantes, les valeurs de n (nombre de lampes) et m (nombre de touches) sont données, et on vous demande de dessiner une configuration *optimale*. Une configuration correcte mais non-optimale donne la moitié des points. Cherchez la solution sur cette feuille avant de la recopier au propre sur la page des réponses.

<p>Q8(a) [2 pts]</p>	<p>Donnez une configuration optimale pour $n = 2$ lampes et $m = 4$ touches.</p>
<p>Solution:</p>	

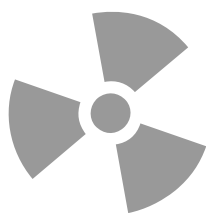
<p>Q8(b) [2 pts]</p>	<p>Donnez une configuration optimale pour $n = 3$ lampes et $m = 8$ touches.</p>
<p>Solution:</p>	

Q8(c) [4 pts]

Donnez une configuration optimale pour $n = 4$ lampes et $m = 16$ touches.



>>> D'autres solutions existent ! <<<



Solutions