

be-OI 2020

Finale - SENIOR
samedi 7 mars 2020

Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp

PRÉNOM :
NOM :
ÉCOLE :

O

Réservé

Finale de l'Olympiade belge d'Informatique 2020 (durée : 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 membres:



©2020 Olympiade Belge d'Informatique (beOI) ASBL

Cette oeuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution 2.0 Belgique.

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 ← 2003  
age ← 2020 − 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 ( $i \leftarrow 0$  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 ( $N \geq 10$ )
{
    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.

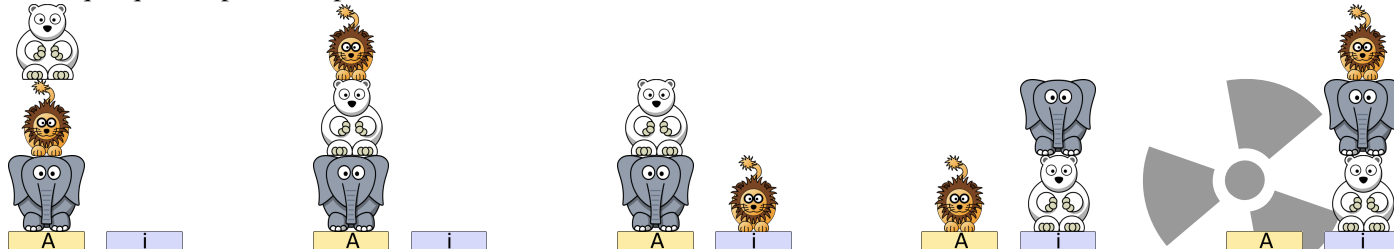
sum ← 0
for ( $i \leftarrow 0$  to  $N - 1$  step 1)
{
    sum ← sum + tab[i]
}
return sum
```

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 – Circus

Le professeur Zarbi a mis au point des numéros de cirque avec ses robots animaux. Il utilise deux socles **A** et **i** (qu'on notera **A** et **i**) sur lesquels les animaux peuvent se placer, empilés les uns au-dessus des autres sans limite de hauteur. Les animaux peuvent être répartis de n'importe quelle manière, il se peut qu'un socle soit vide et que tous les animaux soient empilés sur l'autre.

Voici quelques dispositions possibles avec 3 animaux:



Q1(a) [2 pts] De combien de manières différentes peut-on placer 2 animaux sur les socles ?

Solution: 6

Q1(b) [4 pts] De combien de manières différentes peut-on placer 3 animaux sur les socles ?

Solution: 24

Q1(c) [4 pts] De combien de manières différentes peut-on placer n animaux sur les socles ?

Solution: $(n + 1)!$

Pendant un spectacle, le professeur Zarbi donne des instructions qui font en sorte que les animaux changent de place.

Instructions pour “grimper”.

MA: En l'entendant, l'animal le plus bas sur le socle **A** grimpe au sommet de sa pile.

Mi: En l'entendant, l'animal le plus bas sur le socle **i** grimpe au sommet de sa pile.

(Rien ne se passe s'il y a moins de 2 animaux sur le socle concerné.)

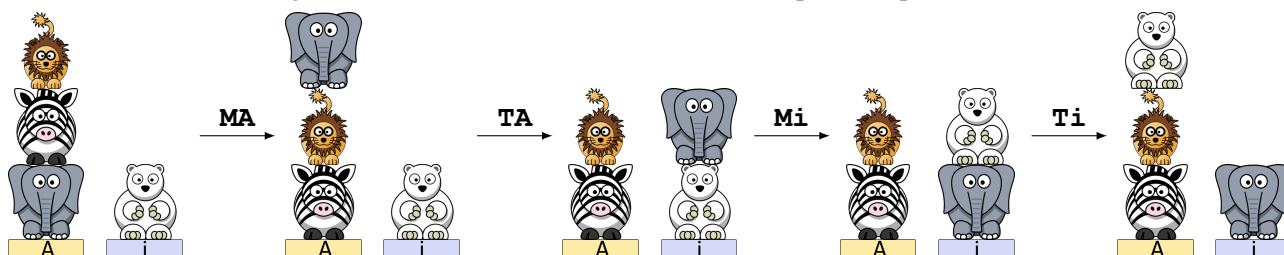
Instructions pour “sauter”.

TA: En l'entendant, l'animal le plus haut sur le socle **A** saute et atterrit en haut de la pile du socle **i**.

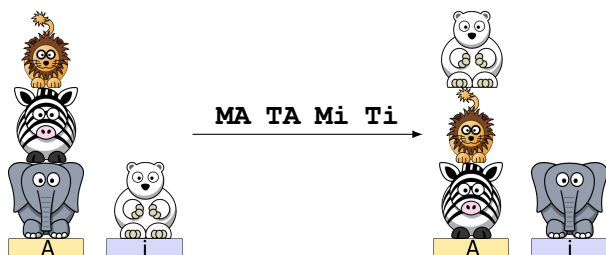
Ti: En l'entendant, l'animal le plus haut sur le socle **i** saute et atterrit en haut de la pile du socle **A**.

(Rien ne se passe s'il n'y a aucun animal sur le socle concerné.)

Exemple: 4 animaux sont placés comme sur l'image de gauche et le professeur donne successivement les instructions **MA TA Mi Ti**. Les images montrent comment évolue la situation après chaque instruction.

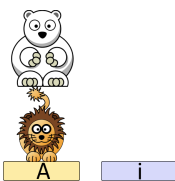
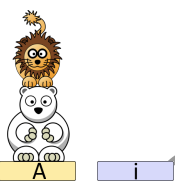
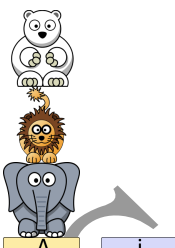
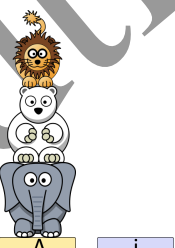
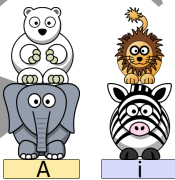
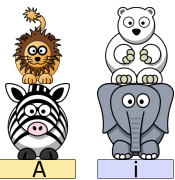


On peut résumer en traçant une seule flèche portant l'ensemble des instructions:



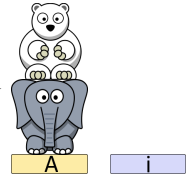
La longueur d'une liste d'instructions est le nombre d'instructions dans la liste. Par exemple la liste d'instructions **MA TA Mi Ti** est de longueur 4.

Dans les questions qui suivent, vous devez trouver une liste d'instructions parmi {**MA**, **Mi**, **TA**, **Ti**} pour passer de la situation de gauche à la situation de droite. **Votre liste doit être la plus courte possible.** Si votre liste n'est pas de longueur minimale, vous marquez seulement la moitié des points à la question.

Q1(d) [2 pts]	 Instructions ? 
Solution: MA	
Q1(e) [2 pts]	 Instructions ? 
Solution: TA MA Ti MA	
Q1(f) [2 pts]	 Instructions ? 
Solution: par exemple MA TA TA Mi Ti Mi Ti	

Q1(g) [4 pts]

Deux animaux doivent passer par toutes les positions le plus rapidement possible. Partant de la position présentée à gauche, il faut passer une seule fois par toutes les autres positions avant de revenir à la position de départ.

Instructions faisant passer par toutes les positions ? Solution: **MA TA TA Mi Ti Ti** ou **TA TA Mi Ti Ti MA**

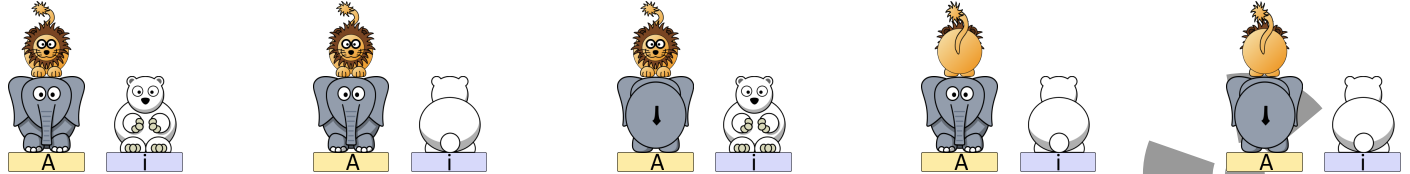
Ajoutons de nouvelles instructions pour “tourner”.

RA: En l’entendant, l’animal le plus haut sur le socle **A** tourne sur lui-même de 180° (il fait un demi-tour).

Ri: En l’entendant, l’animal le plus haut sur le socle **i** tourne sur lui-même de 180° (il fait un demi-tour).

(Rien ne se passe s’il n’y a aucun animal sur le socle concerné.)

Chaque animal peut maintenant être vu de face ou de dos et on considère que les positions suivantes sont différentes:



Q1(h) [2 pts]

En tenant compte que chaque animal peut-être vu de face ou de dos, de combien de manières différentes peut-on placer 2 animaux sur les socles ?

$$\text{Solution: } 6 \cdot 2^2 = 24$$

Q1(i) [4 pts]

En tenant compte que chaque animal peut-être vu de face ou de dos, de combien de manières différentes peut-on placer 3 animaux sur les socles ?

$$\text{Solution: } 24 \cdot 2^3 = 192$$

Q1(j) [4 pts]

En tenant compte que chaque animal peut-être vu de face ou de dos, de combien de manières différentes peut-on placer n animaux sur les socles ?

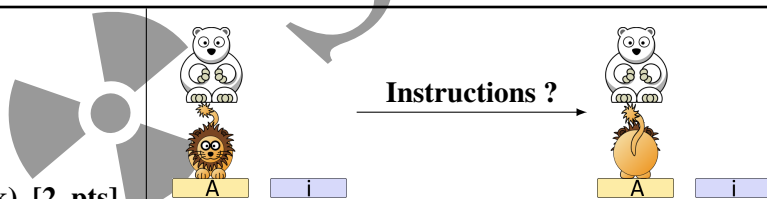
$$\text{Solution: } (n + 1)! \cdot 2^n$$

Quand un animal “grimpe” ou “saute” cela ne change pas son orientation “vu de face ou de dos”.

Dans les questions qui suivent, vous devez trouver une liste d’instructions parmi **{MA, Mi, TA, Ti, RA, Ri}** pour passer de la situation de gauche à la situation de droite.

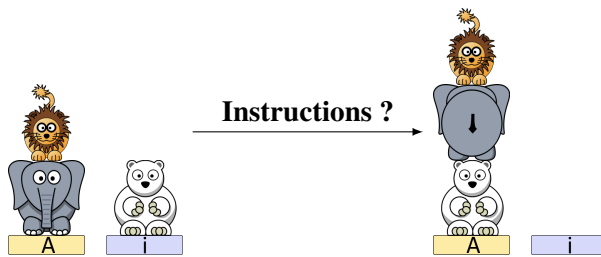
Votre liste doit être la plus courte possible. Si votre liste n’est pas de longueur minimale, vous marquez seulement la moitié des points à la question.

Q1(k) [2 pts]



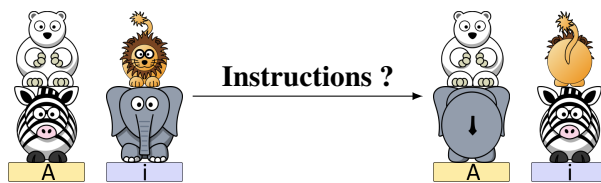
Solution: TA RA Ti ou MA RA MA

Q1(l) [2 pts]



Solution: **Ti MA RA MA**

Q1(m) [2 pts]



Solution: par exemple: **Ri Mi Ri Ti MA TA MA Mi**

Question 2 – Ouvrir la porte

Une porte est sécurisée par un code secret à 4 chiffres.

La porte s'ouvre immédiatement dès que 4 chiffres pressés consécutivement correspondent au code secret.

Par exemple, si on appuie successivement sur 6,4,5,5,0,8,6,7 la porte s'ouvrira si le code secret est 6455, 4550, 5508, 5086 ou 0867.

Nous pouvons programmer un robot en utilisant la commande `press(a)` qui provoque l'appui de la touche `a` par le robot (a étant bien sûr un chiffre de 0 à 9).

Programme 1 :

```
for (a ← 0 to 9 step 1) {
  for (b ← 0 to 9 step 1) {
    for (c ← 0 to 9 step 1) {
      for (d ← 0 to 9 step 1) {
        press(a)
        press(b)
        press(c)
        press(d)
      }
    }
  }
}
```

Dans les questions qui suivent, on demande de tenir compte uniquement des appuis effectués lors de l'exécution du Programme 1.

Q2(a) [1 pt]	Durant l'exécution du Programme 1, combien d'appuis seront effectués en tout ?
Solution: $4 \cdot 10^4 = 40000$	
Q2(b) [1 pt]	Combien de fois la touche <code>5</code> sera-t-elle enfoncée ?
Solution: 4000	
Q2(c) [2 pts]	Combien de combinaisons de 4 chiffres seront testées par le Programme 1 ? Si une combinaison est testée plusieurs fois, il faut compter tous les essais.
Solution: $40000 - 3 = 39997$	
Q2(d) [1 pt]	Combien de combinaisons différentes de 4 chiffres seront testées par le Programme 1 ? Si une combinaison est testée plusieurs fois, il faut compter un seul essai.
Solution: 10000	
Q2(e) [1 pt]	La porte s'ouvrira-t-elle à coup sûr lors de l'exécution du Programme 1 quel que soit le code secret ?
Solution: OUI	

Numérotions les appuis générés par le programme à partir de **0** (ne vous trompez pas, on commence à compter à partir de **zéro**). Les 20 premiers appuis (numérotés de 0 à 19) sont 0,0,0,0, 0,0,0,1, 0,0,0,2, 0,0,0,3, 0,0,0,4. Remarquez que les

appuis dont le numéro est multiple de 4 correspondent toujours à une instruction `press(a)` du programme, et jamais à `press(b)`, ni `press(c)`, ni `press(d)`.

Q2(f) [1 pt]	Quel est le chiffre enfoncé par le robot lors de l'appui numéro 1420 ?
Solution: <input type="text" value="0"/>	
Q2(g) [1 pt]	Quel est le chiffre enfoncé par le robot lors de l'appui numéro 1422 ?
Solution: <input type="text" value="5"/>	
Q2(h) [1 pt]	Donnez les 8 chiffres enfoncés successivement par le robot en commençant par l'appui numéro 1548.
Solution: 0387 0388	

Une “bonne suite” est une suite de 8 chiffres enfoncés successivement par le robot durant l'exécution du Programme 1 et **commençant par un appui de numéro multiple de 4**, autrement dit commençant par un chiffre enfoncé lors d'une instruction `press(a)` du programme.

Voici des exemples de “bonnes suites”: 0000 0001, 2307 2308, 6499 6500 (un espace a été inséré au milieu pour faciliter la lecture).

Remarquez qu'une bonne suite est toujours constituée de 2 nombres successifs de 4 chiffres.

On dit qu'une bonne suite **teste une combinaison** si cette combinaison apparaît dans la bonne suite.

Par exemple, la bonne suite **2307 2308** teste les combinaisons 2307, 3072, 0723, 7230 et 2308.

Autre exemple, la bonne suite **7171 7172** teste les combinaisons 7171 et 1717, à nouveau 7171 et 1717, et enfin 7172.

Remarquez que les deux tests de 7171 n'ont pas lieu en même temps, il s'agit de deux tests différents de la même combinaison (même chose pour 1717).

Q2(i) [5 pts]	Trouvez les 5 bonnes suites qui testent la combinaison 7043. Aide: on peut “couper” la suite de différentes manières (17043, 71043, ...).
Solution: 0437 0438, 3704 3705, 4370 4371, 7042 7043 et 7043 7044	
Q2(j) [1 pt]	Combien de fois le robot appuie-t-il successivement sur les touches 7,0,4,3 ? Autrement dit, combien de fois différentes la combinaison 7043 est-elle testée ?
Solution: 4 (les bonnes suites 7042 7043 et 7043 7044 font le même test)	
Q2(k) [3 pts]	Trouvez les 3 bonnes suites qui testent la combinaison 8383.
Solution: 3838 3839, 8382 8383 et 8383 8384	
Q2(l) [1 pt]	Combien de fois le robot appuie-t-il successivement sur les touches 8,3,8,3 ? Autrement dit, combien de fois différentes la combinaison 8383 est-elle testée ?
Solution: 4 (attention de ne pas compter plusieurs fois le même test)	
Q2(m) [4 pts]	Trouvez toutes les bonnes suites qui testent la combinaison 9900.
Solution: 0990 0991, 8999 9000, 9899 9900 et 9900 9901.	

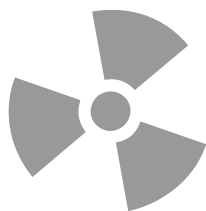
Q2(n) [1 pt]

Combien de fois le robot appuie-t-il successivement sur les touches 9,9,0,0 ? Autrement dit, combien de fois différentes la combinaison 9900 est-elle testée ?

Solution: 3

Q2(o) [1 pt]

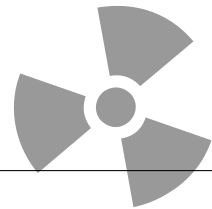
Combien de fois le robot appuie-t-il successivement sur les touches 9,5,6 ? Aide: Si X est un chiffre, par exemple 956X et 56X9 sont des codes secrets.

Solution: $4 \cdot 10 = 40$ 

Considérons maintenant un programme non-déterministe, c'est-à-dire que certains appuis sont générés au hasard. Dans ce programme, la fonction `random()` choisit au hasard un chiffre de 0 à 9. Soyez attentif aux limites de la seconde boucle: ce n'est pas une boucle de 0 à 9 mais de 0 à **a**.

Programme 2 :

```
for (a ← 0 to 9 step 1) {
  for (d ← 0 to a step 1) {
    press(a)
    press(random())
    press(random())
    press(d)
  }
}
```



Dans les questions qui suivent, on demande de tenir compte uniquement des appuis effectués lors de l'exécution du Programme 2.

Q2(p) [3 pts]	Combien de combinaisons de 4 chiffres seront testées par le Programme 2 ? Il se peut que certaines combinaisons soient testées plusieurs fois, il faut compter tous les essais.
Solution: $(1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10) \cdot 4 - 3 = 217$	

Q2(q) [2 pts]	Supposons que le code secret soit 7431. Est-il possible que la porte s'ouvre si on exécute le programme 2 ?
Solution: OUI	

Q2(r) [2 pts]	Supposons que le code secret soit 4444. Est-il possible que la porte s'ouvre si on exécute le programme 2 ?
Solution: OUI	

Q2(s) [2 pts]	Supposons que le code secret soit 2345. Est-il possible que la porte s'ouvre si on exécute le programme 2 ?
Solution: OUI	

*Explications: par exemple en écrivant les appuis à partir du moment où **a** vaut 3 dans la première boucle et **d** vaut 2 dans la seconde, le programme va générer 3**2 3**3 où les * sont des nombres au hasard et il se peut que les 2 derniers * soient 4 et 5.*

Q2(t) [2 pts]	Supposons que le code secret soit 4297. Est-il possible que la porte s'ouvre si on exécute le programme 2 ?
Solution: OUI	

*Explications: par exemple en écrivant les appuis à partir du moment où **a** vaut 9 dans la première boucle et **d** vaut 2 dans la seconde, le programme va générer 9**2 9**3 où les * sont des nombres au hasard et il se peut que la seconde * soit 4 et la troisième étoile soit 7.*

Il est possible d'écrire un programme optimal qui génère une séquence d'appuis dans laquelle toutes les combinaisons de 4 chiffres sont testées exactement une fois chacune. Dans les questions qui suivent, on demande de tenir compte uniquement des appuis effectués lors de l'exécution d'un tel programme optimal.

Q2(u) [4 pts]**Combien d'appuis seront effectués en tout lors de l'exécution d'un programme optimal ?**Solution: $10^4 + 3 = 10003$.

Explications: Il y a 10000 combinaisons à tester, chaque appui doit commencer une nouvelle combinaison, il ne faut pas oublier les 3 derniers chiffres de la dernière combinaison.

Il y a plusieurs façons d'écrire un programme optimal, mais dans chaque cas les 3 premiers chiffres de la séquence générée sont les mêmes que les trois derniers.

Par exemple, si une séquence optimale commence par 123 alors elle se terminera aussi par 123.

Q2(v) [2 pts]**Si on exécute un programme Optimal123 qui génère une séquence commençant par 123, combien de fois le robot enfoncera-t-il successivement les touches 9,5,6 ?**

Solution: 10

Q2(w) [2 pts]**Si on exécute un programme Optimal123 qui génère une séquence commençant par 123, combien de fois le robot enfoncera-t-il successivement les touches 1,2,3 ?**

Solution: 11

Question 3 – Cache-cache

Alice joue à cache-cache avec Bob. Ils jouent dans une maison à $n \geq 2$ pièces, numérotées de 0 à $n - 1$. Alice va se cacher, et Bob va essayer de trouver Alice. Bob est très prévisible, donc Alice sait à l'avance quelles pièces Bob va fouiller et dans quel ordre. Bob va faire $m \geq 1$ fouilles, et peut éventuellement fouiller certaines pièces plusieurs fois. Si Bob fouille une pièce pendant qu'Alice s'y trouve, Bob gagne. Alice veut s'assurer que Bob ne gagne pas, et pour ce faire, elle peut choisir la pièce où elle se cache au début, et elle peut aussi se déplacer d'une pièce à l'autre entre deux fouilles de Bob. Mais chaque déplacement est risqué (elle pourrait se faire détecter), donc Alice veut minimiser le nombre de déplacements qu'elle doit effectuer.

Par exemple, imaginons que la maison a $n = 3$ pièces numérotées 0, 1 et 2, et que Bob va effectuer la séquence suivante de $m = 4$ fouilles:

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

Une solution possible pour Alice est de se cacher d'abord dans la pièce 2, puis de se déplacer vers la pièce 0 avant que Bob ne fouille la pièce 2, et enfin de se déplacer vers la pièce 1 avant que Bob ne fouille la pièce 0. Alice se déplacerait 2 fois et sa position au cours du temps serait décrite par la séquence:

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

Mais Alice peut mieux faire: après avoir quitté la pièce 2, elle pourrait se déplacer directement dans la pièce 1 et y rester jusqu'à la fin du jeu. Elle se déplacerait une seule fois et sa position au cours du temps serait :

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

En revanche, il n'y a pas moyen de faire moins qu'un déplacement. Alice ne peut pas rester dans la même pièce pendant toute la durée du jeu puisque Bob fouille toutes les pièces au moins une fois. Donc, pour $n = 3$ et $\text{bob}[] = \{1, 2, 0, 2\}$, le nombre minimal de déplacements d'Alice est 1.

Q3(a) [2 pts]	Quel est le nombre minimal de déplacements d'Alice si $n = 2$ et si Bob fouille les pièces $\text{bob}[] = \{0, 1, 0, 1\}$?
Solution: 3 ($\text{alice}[] = \{1, 0, 1, 0\}$)	
Q3(b) [2 pts]	Quel est le nombre minimal de déplacements d'Alice si $n = 3$ et si Bob fouille les pièces $\text{bob}[] = \{0, 1, 0, 1\}$?
Solution: 0 ($\text{alice}[] = \{2, 2, 2, 2\}$)	
Q3(c) [2 pts]	Quel est le nombre minimal de déplacements d'Alice si $n = 3$ et si Bob fouille les pièces $\text{bob}[] = \{2, 1, 2, 0\}$?
Solution: 1 (par exemple, $\text{alice}[] = \{0, 0, 1, 1\}$)	
Q3(d) [2 pts]	Quel est le nombre minimal de déplacements d'Alice si $n = 3$ et si Bob fouille les pièces $\text{bob}[] = \{0, 1, 2, 0, 2, 0\}$?
Solution: 1 ($\text{alice}[] = \{2, 2, 1, 1, 1, 1\}$)	
Q3(e) [2 pts]	Quel est le nombre minimal de déplacements d'Alice si $n = 4$ et si Bob fouille les pièces $\text{bob}[] = \{0, 1, 2, 3, 0\}$?
Solution: 1 (par exemple, $\text{alice}[] = \{3, 3, 3, 2, 2\}$)	

Q3(f) [2 pts]	Quel est le nombre minimal de déplacements d'Alice si $n = 4$ et si Bob fouille les pièces $\text{bob}[] = \{0, 1, 2, 3, 0, 1\}$?
Solution: 1 ($\text{alice}[] = \{3, 3, 3, 2, 2, 2\}$)	
Q3(g) [3 pts]	Quel est le nombre minimal de déplacements d'Alice si $n = 4$ et si Bob fouille les pièces $\text{bob}[] = \{0, 1, 2, 3, 0, 1, 2\}$?
Solution: 2 (par exemple, $\text{alice}[] = \{3, 3, 3, 2, 2, 3, 3\}$)	
Q3(h) [4 pts]	Dans une maison à n pièces, quel est le plus petit nombre de fouilles m que Bob doit effectuer pour forcer Alice à se déplacer au moins k fois?
Solution: $k(n - 1) + 1$	

Tout devient plus drôle avec du code

Alice décide d'écrire un programme qui détermine à sa place dans quelle pièce elle doit se cacher pendant chacune des m fouilles de Bob.

Les entrées du programme sont:

- n : le nombre de pièces.
- m : le nombre de fouilles que Bob va effectuer.
- $\text{bob}[]$ (un tableau de longueur m) : la séquence de fouilles que Bob va effectuer.

Le programme doit calculer $\text{alice}[]$ (un tableau de longueur m) : une liste de m pièces où Alice doit se trouver pour éviter de se faire découvrir par Bob, tout en se déplaçant le moins de fois possible.

Il peut y avoir plusieurs solutions qui donnent le nombre minimum de déplacements; si c'est le cas, n'importe laquelle est acceptable et le programme d'Alice en donne une seule.

Le listing du programme est sur la page suivante.

Voici quelques explications pour vous aider à le comprendre.

- cnt : sert à compter le nombre de pièces différentes que Bob à l'intention de visiter après le dernier déplacement d'Alice.
- $\text{t}[]$ (un tableau de longueur n) : si $\text{t}[v] = \text{false}$ cela veut dire que Bob compte visiter la pièce v après le dernier déplacement d'Alice.
- La première boucle **for** recherche les cachettes d'Alice, sauf la dernière.
- La fin du programme à partir de $a \leftarrow -1$ recherche la dernière cachette d'Alice.

Le listing est incomplet et on vous demande de trouver les expressions manquantes.

Q3(i) [2 pts]	Quelle est l'expression (i) dans l'algorithme ?
Solution: n	
Q3(j) [2 pts]	Quelle est l'expression (j) dans l'algorithme ?
Solution: 1	

Q3(k) [3 pts]	Quelle est l'expression (k) dans l'algorithme ?
----------------------	--

Solution: <code>bob[i]</code>	
-------------------------------	--

Q3(l) [2 pts]	Quelle est l'expression (l) dans l'algorithme ?
----------------------	--

Solution: <code>t[v]</code>	
-----------------------------	--

Q3(m) [2 pts]	Quelle est l'expression (m) dans l'algorithme ?
----------------------	--

Solution: <code>j < m</code>	
---------------------------------	--

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

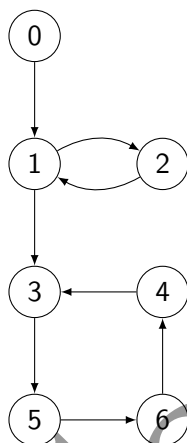
```
alice[] ← {-1, ..., -1} (un tableau de longueur m, initialisé avec des -1)
t[] ← {true, ..., true} (un tableau de longueur n, initialisé avec des true)
cnt ← 0
j ← 0
for (i ← 0 to m-1 step 1)
{
    if (t[bob[i]])
    {
        cnt ← cnt + 1
        if (cnt = ...) // (i)
        {
            cnt ← ... // (j)
            while (j < i)
            {
                alice[j] ← ... // (k)
                j ← j + 1
            }
            for (v ← 0 to n-1 step 1)
            {
                t[v] ← true
            }
        }
        t[bob[i]] ← false
    }
}
a ← -1
for (v ← 0 to n-1 step 1)
{
    if (...) // (l)
    {
        a ← v
    }
}
while (...) // (m)
{
    alice[j] ← a
    j ← j + 1
}
return alice[]
```

Question 4 – Al Capone

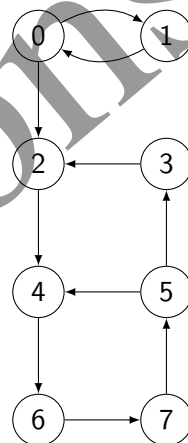
Le célèbre gangster Al Capone a décidé de réorganiser son réseau de bars clandestins à Chicago. Il veut regrouper plusieurs bars en *quartiers*, et chaque quartier sera sous la direction d'un chef de gang. Afin de permettre à ses complices de fuir en cas de descentes de police, il veut que la condition suivante soit respectée: *deux bars sont dans le même quartier si et seulement s'il existe au moins un trajet dans chaque sens permettant de fuir de l'un à l'autre en voiture (éventuellement en passant devant d'autres bars)*. Un quartier peut comporter n'importe quel nombre de bars tant que la condition ci-dessus est respectée pour toute paire de bars. Par exemple, les bars 0 et 1 et 2 seront dans le même « quartier » s'il est possible d'aller en voiture de 0 à 1, de 1 à 0, de 1 à 2, de 2 à 1, de 0 à 2 et de 2 à 0.

Voici deux cartes représentant des bars d'Al Capone à Chicago. Chaque bar est représenté par un cercle (portant le numéro du bar) et les flèches indiquent les routes (à sens unique) qu'on peut suivre pour aller *directement* (c'est-à-dire sans passer devant d'autres bars) d'un bar à l'autre en voiture.

Carte 1:



Carte 2:



Par exemple, avec la **Carte 1**, on peut aller (directement) en voiture de 3 à 5. On peut aussi aller en voiture de 5 à 3 mais il faut passer devant 6 et 4 (car la route directe de 5 à 3 est un sens interdit).

Indiquez, pour chacune des affirmations suivantes au sujet de la **Carte 1**, si elles sont vraies ou fausses.

	Vrai	Faux	Affirmations concernant la Carte 1 .
Q4(a) [1 pt]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	On peut aller en voiture de 1 à 0.
Q4(b) [1 pt]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	On peut aller en voiture de 6 à 1.
Q4(c) [1 pt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	On peut aller en voiture de 6 à 5.
Q4(d) [1 pt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1 et 2 peuvent être dans le même quartier.
Q4(e) [1 pt]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	6 et 2 peuvent être dans le même quartier.
Q4(f) [1 pt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	3, 4 et 6 peuvent être dans le même quartier.
Q4(g) [1 pt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	3, 4, 5 et 6 peuvent être dans le même quartier.
Q4(h) [1 pt]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1, 2, 3, 4, 5 et 6 peuvent être dans le même quartier.

Indiquez, pour chacune des affirmations suivantes au sujet de la **Carte 2**, si elles sont vraies ou fausses.

	Vrai	Faux	Affirmations concernant la Carte 2 .
Q4(i) [1 pt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	On peut aller en voiture de 1 à 0.
Q4(j) [1 pt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	On peut aller en voiture de 2 à 7.
Q4(k) [1 pt]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	On peut aller en voiture de 7 à 1.
Q4(l) [1 pt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2 et 5 peuvent être dans le même quartier.
Q4(m) [1 pt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2, 3 et 4 peuvent être dans le même quartier.
Q4(n) [1 pt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	4, 5, 6 et 7 peuvent être dans le même quartier.
Q4(o) [1 pt]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	4, 5, 6 et 7 peuvent être dans le même quartier et il n'existe pas de quartier plus grand contenant ces 4 bars.
Q4(p) [1 pt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2, 3, 4, 5, 6 et 7 peuvent être dans le même quartier.
Q4(q) [1 pt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2, 3, 4, 5, 6 et 7 peuvent être dans le même quartier et il n'existe pas de quartier plus grand contenant ces 6 bars.

Malheureusement, Capone ne dispose pas de cartes aussi précises de ses bars, et envoie donc son complice Joe à pied (c'est plus discret) visiter les rues de Chicago. Capone explique à Joe comment il doit s'y prendre pour parcourir la ville: « Je te donne l'adresse d'un premier bar. Tu suivras toutes les routes qui partent de ce bar, en respectant les sens uniques: d'abord celle à l'ouest (à gauche sur la carte), puis celle au sud (en bas), puis celle à l'est (à droite), puis celle au nord (en haut) si elles existent. Si au bout d'une route tu trouves **un bar par où tu n'es encore jamais passé** durant ton exploration, tu recommences la même procédure à partir de ce nouveau bar, sinon tu reviens sur tes pas au bar précédent. Quand tu auras exploré toutes les routes partant d'un bar, tu reviens aussi sur tes pas. »

Comme Joe n'est pas très malin, Capone lui donne l'algorithme `Explore` (voir listing plus loin) à suivre à la lettre. Celui-ci se présente sous forme d'une fonction, qui est appelée avec en paramètre le premier bar donné par Capone. Cette fonction remplit un tableau `order` dont l'indice est le numéro d'un bar (de 0 à $N-1$), et qui donne l'ordre (de 1 à N) dans lequel les bars sont visités.

Par exemple, si `order[i]=1`, cela signifie que le bar numéro i est le premier donné par Capone, etc

Par exemple, sur la **Carte 1**, et en supposant que le bar 0 est le premier, l'algorithme `Explore` visitera d'abord le bar 0, puis le 1 (seul bar accessible à partir de 0), puis le 3 (on visite d'abord le bar au sud avant de visiter celui à l'est), puis 5, 6, 4 et enfin 2 (après être revenus sur ses pas).

Q4(r) [3 pts]	Donnez le contenu du tableau <code>order</code> à la fin de l'exécution de la fonction <code>Explore</code> sur la Carte 1, en supposant que le premier bar est le bar 0.
Solution: [1, 2, 7, 3, 6, 4, 5]	

Q4(s) [3 pts]	Donnez l'ordre dans lequel les bars sont visités par l'algorithme <code>Explore</code> appelé sur la Carte 2 à partir du bar 0.
Solution: 0, 2, 4, 6, 7, 5, 3, 1.	

Q4(t) [3 pts]	Donnez le contenu du tableau <code>order[]</code> à la fin de l'exécution de la fonction <code>Explore</code> sur la Carte 2, en supposant que le premier bar est le bar 0.
Solution: [1, 8, 2, 7, 3, 6, 4, 5]	

Input: N, le nombre de bars

I, le numero du premier bar (entre 0 et N-1)

order[] \leftarrow {-1, ..., -1} (un tableau de longueur N, initialise avec des -1)

cpt \leftarrow 1 (un compteur initialise à 1)

Explore(bar X)

```
{
  if (order[X] = -1)
  {
    order[X]  $\leftarrow$  cpt
    cpt  $\leftarrow$  cpt+1

    for (d = ouest, sud, est, nord)
    {
      if (il y a une route de direction d partant de X)
      {
        Y  $\leftarrow$  le bar au bout de la route de direction d partant de X
        Explore(Y)
      }
    }
  }
}
```

Explore(I) (remplit le tableau order[] pour une exploration partant du bar numero I)

Maintenant que Joe semble avoir compris ce premier algorithme, Capone enrichit la fonction `Explore` pour permettre de découper la carte en quartiers. Tout d'abord, il demande à Joe d'emporter avec lui un calepin, qui lui permettra de tenir une liste des bars qu'il a visité: à chaque fois qu'il arrive dans un nouveau bar, il ajoute celui-ci au bas de sa liste. Par exemple, sur la **Carte 1**, quand Joe arrive au bar numéro 5 (au moment d'entrer dans l'appel `Explore(5)`, sa liste contient: 0, 1, 3.

Q4(u) [3 pts]	Quel est le contenu de la liste de Joe quand il arrive au bar numéro 3 sur la Carte 2, c'est-à-dire au moment de l'appel à <code>Explore(3)</code> ?
Solution: 0, 2, 4, 6, 7, 5.	

Ensuite, Capone ajoute un second tableau `low[]` à la fonction `Explore`. Celui-ci permet à Joe de retenir, pour chaque bar, quel est le *plus petit numéro d'ordre* (donné dans le tableau `order`) auquel on peut accéder depuis ce bar. Capone s'est alors rendu compte qu'en cherchant les bars X tels que `order[X] = low[X]`, on peut identifier les quartiers sur base de la liste maintenue par Joe. Tout cela est expliqué en détail dans le nouvel algorithme (voir listing plus loin):

En exécutant `Explore(1)`, Joe va non seulement remplir les tableaux `order[]` et `low[]`, mais il va aussi délimiter les quartiers dans lesquels chaque bar est (pour rappel) accessible en voiture depuis n'importe quel autre bar du quartier.

Q4(v) [3 pts]	Quel est le contenu du tableau <code>low[]</code> à la fin de l'exécution de l'algorithme sur la Carte 1 (bar initial 0) ?
Solution: [1, 2, 2, 3, 3, 3, 3]	

Q4(w) [3 pts]	Sur la Carte 2 (bar initial 0), quelle est la valeur de <code>low[5]</code> au moment de l'appel <code>Explore(3)</code> ?
Solution: 3	

Q4(x) [3 pts]	Sur la Carte 2 (bar initial 0), quel est le contenu de la liste de Joe au moment de l'appel <code>Explore(1)</code> ?
Solution: 0	

Q4(y) [3 pts]	Sur la Carte 1 (bar initial 0), quels sont les quartiers calculés par l'algorithme de Capone ? Indiquez chaque quartier comme un ensemble de numéros de bars, par exemple {0, 1, 4, 5}, {2, 3, 6}.
Solution: {0}, {1, 2}, {3, 4, 5, 6}	

Q4(z) [3 pts]	Sur la Carte 2 (bar initial 0), quels sont les quartiers calculés par l'algorithme de Capone ? Indiquez chaque quartier comme un ensemble de numéros de bars.
Solution: {0, 1}, {2, 3, 4, 5, 6, 7}	

Input: I, le premier bar
N, le nombre de bars

order[] \leftarrow {-1, ..., -1} (un tableau de longueur N, initialise avec des -1)
low[] \leftarrow {-1, ..., -1} (un tableau de longueur N, initialise avec des -1)
cpt \leftarrow 1 (un compteur initialise à 1)

Explore(bar X)

```
{  
  order[X]  $\leftarrow$  cpt  
  low[X]  $\leftarrow$  cpt  
  Ajouter X au bas de la liste  
  cpt  $\leftarrow$  cpt+1  
  for (d = ouest, sud, est, nord)  
  {  
    if (il y a une route de direction d partant de X)  
    {  
      Y  $\leftarrow$  le bar au bout de la route de direction d partant de X  
      if (order[Y] = -1)  
      {  
        Explore(Y)  
        low[X]  $\leftarrow$  min(low[X], low[Y])  
      }  
      else if (Y est dans la liste)  
      {  
        low[X]  $\leftarrow$  min(low[X], order[Y])  
      }  
    }  
  }  
  if (low[X] = order[X])  
  {  
    Creer un nouveau quartier contenant tous les bars qui apparaissent  
    sur la liste a partir de X (compris), et les effacer de la liste.  
  }  
}
```

Explore(I)

Question 5 – Guerres administratives

Dans un monde lointain, très lointain, s'affrontent plusieurs pays pour dominer leur planète. Leurs guerres sont différentes des nôtres et se font sans verser de sang. Elles opposent les directeurs administratifs des pays dans des joutes verbales endiablées (en réalité, comme nous le verrons, c'est la puissance des services administratifs qui compte). À ce moment, il est probablement utile d'expliquer ce qu'est une **entité**. Un pays donné peut être divisé en provinces, chacune de ces provinces peut être divisée en sous-provinces, chacune de ses sous-provinces peut être divisée en sous-sous-provinces, et ainsi de suite. Chaque pays, province, sous-province, sous-sous-province, sous-sous-sous-Province, ... est une **entité**.

Prenons l'exemple du pays de Heldor :



Fig 1: Carte géographique de Heldor

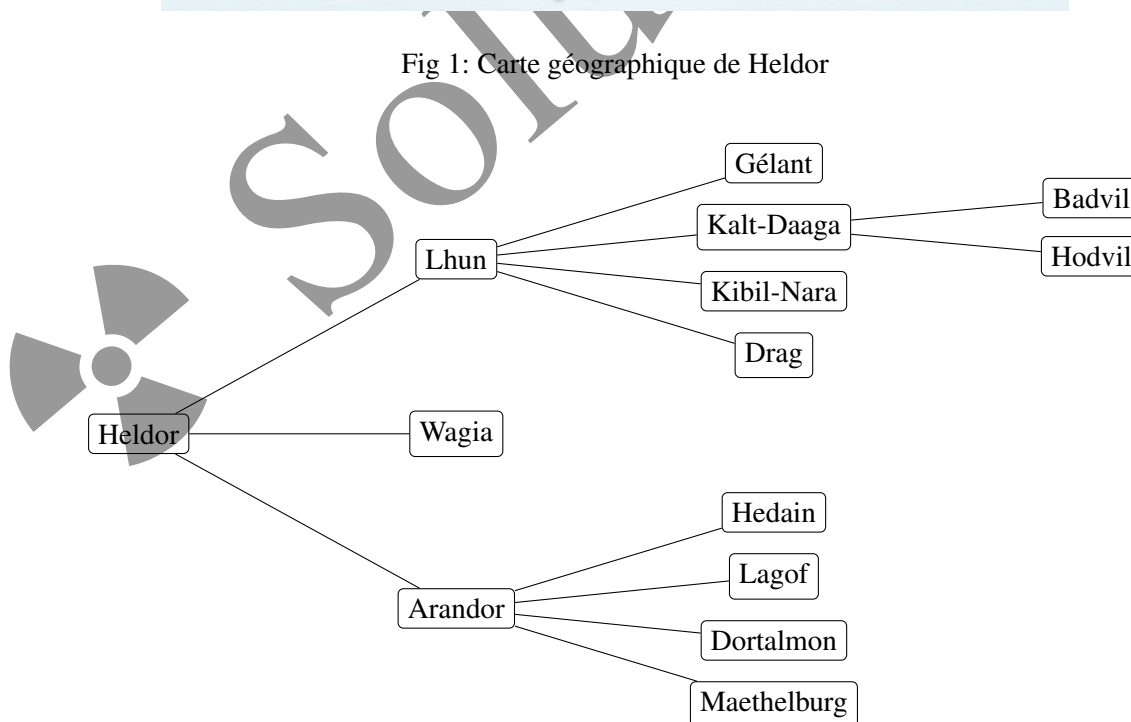


Fig 2: Carte administrative des **entités** de Heldor

Chaque **entité** dispose de sa propre administration qui a sa propre puissance. Évidemment, la complexité administrative fait que plus une entité gère d'entités en dessous d'elles, moins elle est efficace (la bureaucratie est un problème dans tous les mondes).

Par exemple, Lhun gère 7 entités: elle-même et 6 entités en dessous d'elle. On dit que Lhun a 7 **sous-entités** (car il faut compter l'administration centrale de Lhun comme une sous-entité particulière !).

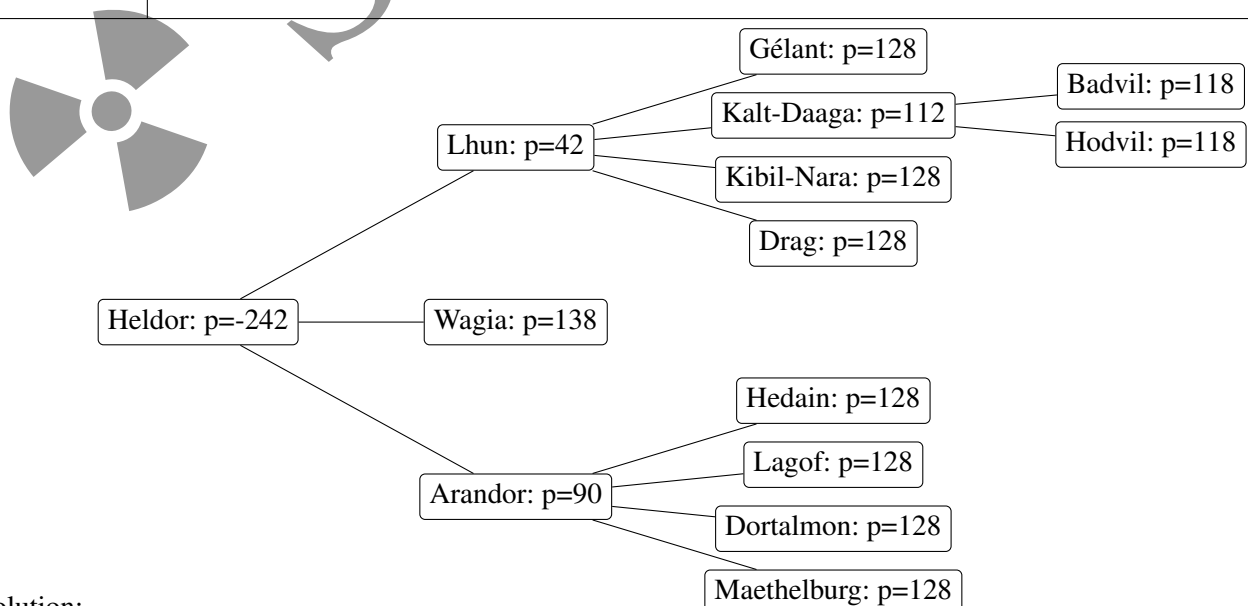
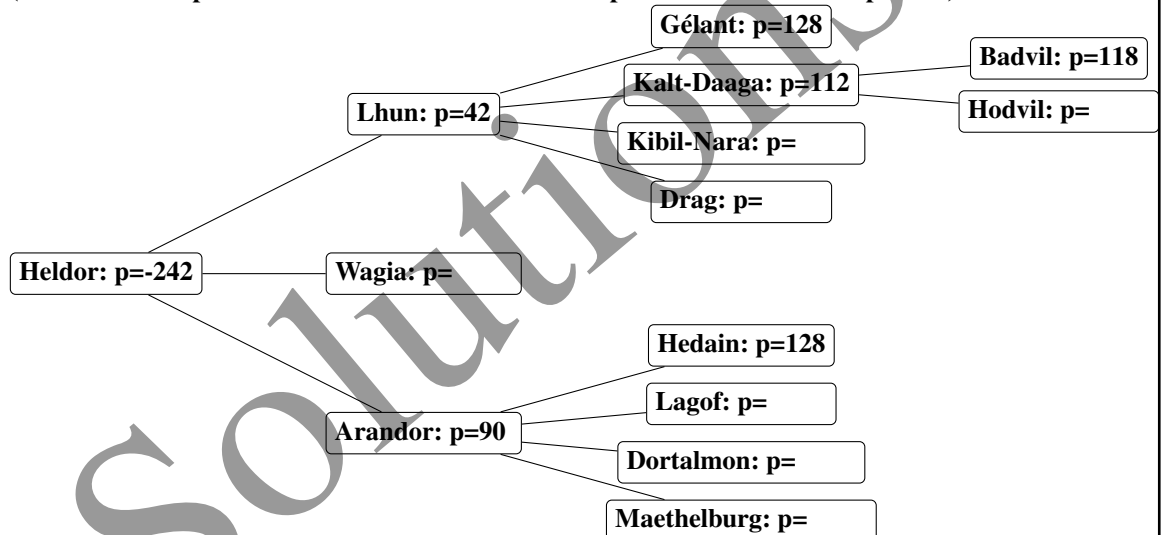
Par ailleurs, plus une **entité** est petite par rapport au pays, moins elle a d'argent et moins son administration est puissante.

Plus précisément, si **X** est une **entité** de niveau n (0 pour un pays, 1 pour une province, 2 pour une sous-province, 3 pour une sous-sous-province, ...) et si s est le nombre de sous-entités de **X** alors p , la **puissance administrative** de **X**, s'élève à $p = 150 - 10n - 2s^2$.

- Exemple de la sous-sous-province de Badvil: $n = 3$ et $s = 1$, donc $p = 150 - 10 \cdot 3 - 2 \cdot 1^2 = 150 - 30 - 2 = 118$.
- Exemple de la province de Lhun: $n = 1$ et $s = 7$, donc $p = 150 - 10 \cdot 1 - 2 \cdot 7^2 = 150 - 10 - 98 = 42$.
- Exemple du pays Heldor: $n = 0$ et $s = 14$, donc $p = 150 - 10 \cdot 0 - 2 \cdot 14^2 = 150 - 392 = -242$.

Q5(a) [7 pts]

Complétez la carte administrative de Heldor en notant les puissances des entités.
 (Si nécessaire répondez ici au brouillon avant de recopier sur la feuille de réponses.)



Solution:

Vous constatez que Heldorf est submergé par le travail administratif et que sa puissance est négative, c'est-à-dire que Heldorf consomme des ressources pour maintenir son administration à flot !

De même, une sous-sous-sous... province pourrait avoir une puissance négative ou nulle, même si elle n'a pas d'autre sous-entité qu'elle-même. Mais combien de "sous" faut-il ?

Q5(b) [2 pts]	Combien de "sous" faut-il mettre devant le mot "province" pour être certain que sa puissance soit nulle ou négative (≤ 0) ?
Solution: 14 (car $150 - 10n - 2 \leq 0 \iff n \geq 14,8$ et il y a 14 "sous" au niveau 15)	

Ce qui compte vraiment, c'est la **puissance totale**. La **puissance totale** d'une entité **X** est obtenue en additionnant les puissances de toutes les sous-entités de **X** (donc sans oublier **X** elle-même). Par exemple, la **puissance totale** de Badvil est égale à sa puissance (118), puisqu'elle n'a pas d'autre sous-entité qu'elle-même.

Par contre, la **puissance totale** de Lhun est une somme de 7 puissances (aide: elle vaut 774).

Q5(c) [1 pt]	Quelle est la puissance totale de Hodvil ?
Solution: 118	

Q5(d) [1 pt]	Quelle est la puissance totale de Wagia ?
Solution: 138	

Q5(e) [2 pts]	Quelle est la puissance totale d'Arandor ?
Solution: $90 + 4 \cdot 128 = 602$	

Q5(f) [2 pts]	Quelle est la puissance totale de Heldorf ?
Solution: $-242 + 774 + 138 + 602 = 1272$	

Quand deux pays s'affrontent, celui qui a la plus grande **puissance totale** l'emporte. En cas d'égalité, les administrations s'empêchent et les pays disparaissent dans un vice de procédure. Soit une fonction `PUISSANCE_TOTALE(X, n)` qui permet de calculer la **puissance totale** d'une entité **X** de niveau **n**.

Q5(g) [2 pts]	Donnez l'expression indiquant si le pays A gagne une bataille contre le pays B. Rappelez-vous que les pays ont un niveau $n=0$ et qu'en cas d'égalité il n'y a pas de vainqueur.
Solution: <code>PUISSANCE_TOTALE(A, 0) > PUISSANCE_TOTALE(B, 0)</code>	

Chaque entité de niveau **n** connaît la liste de ses sous-entités de niveau **n + 1** qu'on appelle ses **enfants**. Ce nom étrange provient de la théorie des graphes, utilisée pour tracer les cartes administratives. Sur une carte administrative, un trait distinct relie chaque entité à chacun de ses enfants (voir Fig 2). Les enfants d'un pays sont ses provinces, les enfants d'une province sont ses sous-provinces,...

Pour calculer la puissance administrative d'une entité **X**, il faut connaître son nombre de sous-entités (elle comprise). Voici le pseudo-code incomplet (il manque une ligne) de la fonction `CALCULER_S(X)` qui fait cela.

```
fonction CALCULER_S(X) {
```

```

nb_entites ← 1
pour chaque E enfant de X  {
    ... //(i)
}
return nb_entites

```

Notez que la ligne “pour chaque E enfant de X” commence une boucle qui sera exécutée une fois pour chaque enfant de X, et que cet enfant sera désigné par la variable E dans la boucle. Par exemple, si X=Heldor, la boucle sera exécutée 3 fois, une fois avec E=Lhun, une fois avec E=Wagia, une fois avec E=Arandor. Si X n’a pas d’enfant, la boucle n’est pas exécutée (elle est exécutée 0 fois puisqu’il y a 0 enfant).

Q5(h) [2 pts]		Quelle est la ligne (i) dans le pseudo-code de la fonction CALCULER_S () ?
	<input type="checkbox"/>	nb_entites ← CALCULER_S(E) + 1
	<input type="checkbox"/>	nb_entites ← nb_entites + 1
	<input checked="" type="checkbox"/>	nb_entites ← nb_entites + CALCULER_S(E)
	<input type="checkbox"/>	nb_entites ← CALCULER_S(E)
	<input type="checkbox"/>	nb_entites ← nb_entites + CALCULER_S(E) + 1

Complétez le pseudo-code de la fonction `PUISSANCE_TOTALE(X, n)` qui permet (comme cela a déjà été signalé) de calculer la **puissance totale** d'une entité x de niveau n .

```

fonction PUISSANCE_TOTALE(X, n) {
  puissance_de_X ← ...           //(i)
  puissance_des_enfants ← 0
  pour chaque E enfant de X {
    puissance_des_enfants ← puissance_des_enfants + ...   //(j)
  }
  return ...                     //(k)
}

```

Indices: vous pouvez et devriez utiliser `CALCULER_S(X)`. Considérez que cette fonction est correctement implémentée, même si vous n'avez pas répondu à la question précédente. Pour calculer le carré d'un nombre (r), multipliez-le par lui-même ($r \times r$) ou mettez-le à la puissance 2 (r^2).

Q5(i) [2 pts] Quelle est l'expression (i) dans le pseudo-code de la fonction `PUISSANCE_TOTALE(X, n)` ?

Solution: $150 - 10 \times n - 2 \times \text{CALCULER_S}(X) \times \text{CALCULER_S}(X)$

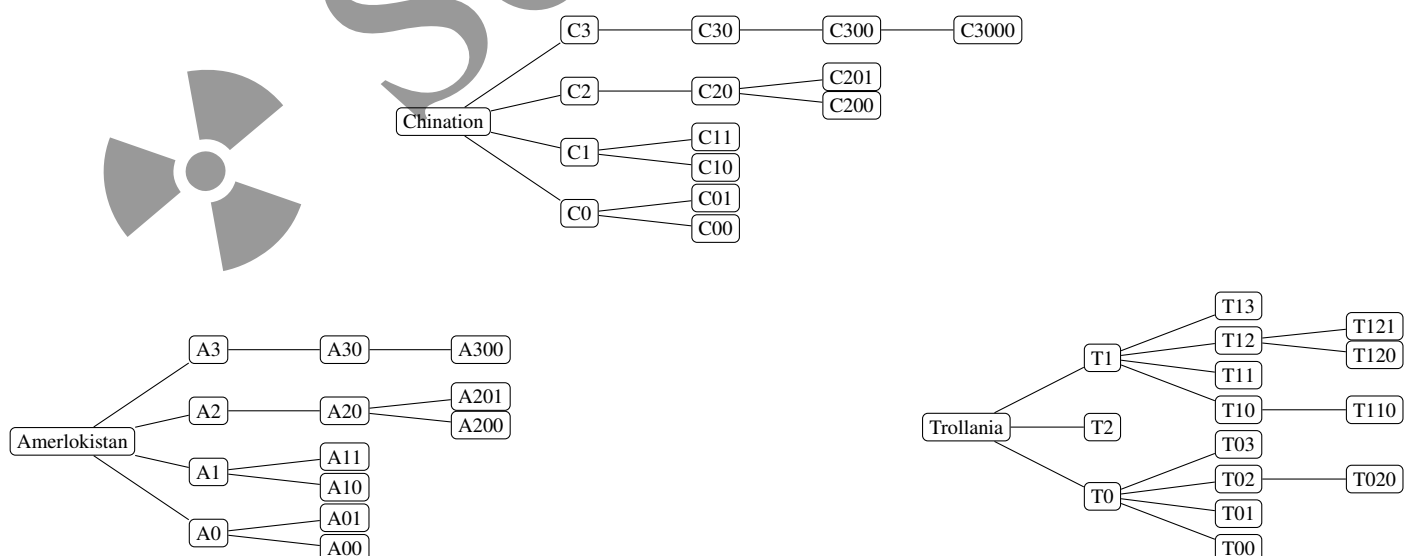
Q5(j) [2 pts] Quelle est l'expression (j) dans le pseudo-code de la fonction `PUISSANCE_TOTALE(X, n)` ?

Solution: `PUISSANCE_TOTALE(E, n+1)`

Q5(k) [2 pts] Quelle est l'expression (k) dans le pseudo-code de la fonction `PUISSANCE_TOTALE(X, n)` ?

Solution: `puissance_de_X + puissance_des_enfants`

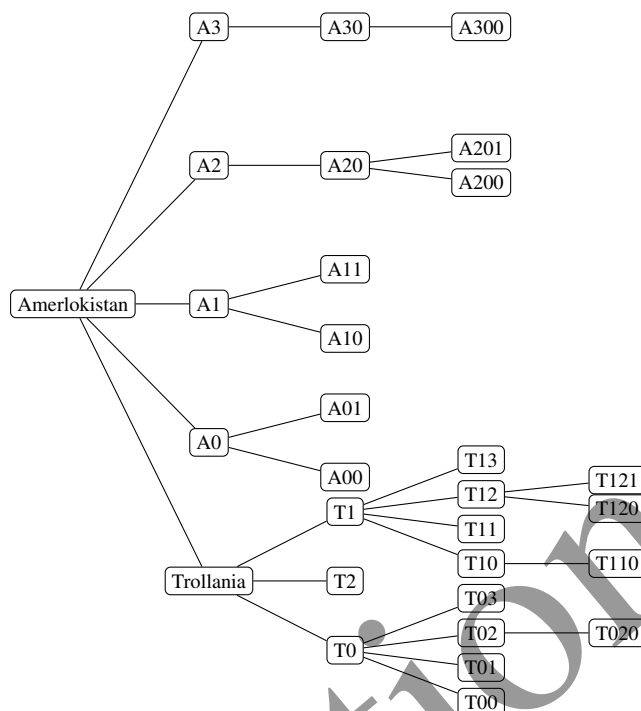
Quand deux pays se font la guerre, le pays perdant est absorbé par le pays gagnant et devient l'une de ses provinces. Cela change évidemment la puissance administrative du pays gagnant ! Prenons un exemple. Voici les cartes administratives de trois pays (on a utilisé des codes postaux au lieu des noms, c'est plus court).



Nos espions ont calculé les **puissances totales** de ces trois pays pour vous :

Chinaton: 1352	Amerlokistan: 1332	Trollania: 1324
-----------------------	---------------------------	------------------------

En cas de guerre entre Amerlokistan et Trollania, c'est Amerlokistan qui absorberait Trollania.
 Voici ce que deviendrait la carte administrative de Amerlokistan après cette victoire :



La nouvelle **puissance totale** de Amerlokistan après la guerre serait égale à **1088**.

Il existe une formule pour calculer la **puissance totale** du gagnant d'une guerre.

Q5(l) [5 pts]		Soit x_1 un pays de puissance totale t_1 ayant s_1 sous-entités (x_1 compris). Soit x_2 un pays de puissance totale t_2 ayant s_2 sous-entités (x_2 compris). Si $t_1 > t_2$ quelle expression donne la nouvelle puissance totale du gagnant de l'affrontement entre x_1 et x_2 ?
	<input type="checkbox"/>	$t_1 + t_2$
	<input type="checkbox"/>	$t_1 + t_2 - 10*s_2$
	<input type="checkbox"/>	$t_1 + t_2 - 2*s_2*s_2$
	<input type="checkbox"/>	$t_1 + t_2 + 2*s_1*s_1 - 2*(s_1+s_2)*(s_1+s_2)$
	<input type="checkbox"/>	$t_1 + t_2 + 2*s_2*s_2 - 2*(s_1+s_2)*(s_1+s_2)$
	<input checked="" type="checkbox"/>	$t_1 + t_2 - 10*s_2 + 2*s_1*s_1 - 2*(s_1+s_2)*(s_1+s_2)$
	<input type="checkbox"/>	$t_1 + t_2 - 10*s_2 + 2*s_2*s_2 - 2*(s_1+s_2)*(s_1+s_2)$

Dans ce monde, il n'y a **qu'une guerre à la fois**, toujours **entre les deux plus grandes puissances** administratives. Rappelons qu'à chaque fois, le gagnant absorbe le perdant et qu'en cas d'égalité, les administrations s'empêchent et les deux pays disparaissent. Ensuite, les deux nouvelles plus grandes puissances se font la guerre, etc, etc, jusqu'à qu'il ne reste plus qu'une seule nation (ou pas du tout...).

Q5(m) [2 pts]		Comment se déroulera la première guerre entre Chination, Amerlokistan et Trollania ?
	<input checked="" type="checkbox"/>	Chination battraient Amerlokistan.
	<input type="checkbox"/>	Amerlokistan battraient Chination.
	<input type="checkbox"/>	Chination battraient Trollania.
	<input type="checkbox"/>	Trollania battraient Chination.
	<input type="checkbox"/>	Trollania battraient Amerlokistan.
	<input type="checkbox"/>	Amerlokistan battraient Trollania.

Q5(n) [3 pts]	Quelle sera la nouvelle puissance totale du vainqueur de cette première guerre ?
Solution: Chination gagne et sa nouvelle puissance totale vaut 1312	

Explications pour la solution.

Pour Chination: $t1=1352$, $s1=15$.

Pour Amerlokistan: $t2=1332$, $s2=14$.

Chination gagne et sa nouvelle puissance totale vaut $1352 + 1332 - 10 \cdot 14 + 2 \cdot 15 \cdot 15 - 2 \cdot 29 \cdot 29 = 1312$

Q5(o) [2 pts]		Qui sera le gagnant final, après la deuxième guerre opposant le troisième pays au gagnant de la première guerre ?
	<input type="checkbox"/>	Chination
	<input type="checkbox"/>	Amerlokistan
	<input checked="" type="checkbox"/>	Trollania

Q5(p) [3 pts]	Quelle est la nouvelle puissance totale du vainqueur final ?
Solution: Trollania gagne et sa nouvelle puissance totale vaut -1192	

Explications pour la solution.

Pour Trollania: $t1=1324$, $s1=16$.

Pour Chination: $t2=1312$, $s2=29$.

Trollania gagne et sa nouvelle puissance totale vaut $1324 + 1312 - 10 \cdot 29 + 2 \cdot 16 \cdot 16 - 2 \cdot 45 \cdot 45 = -1192$

Faire tout cela à la main est rébarbatif, n'est-ce pas ? Il est temps d'implémenter une procédure.

Tous les pays sont maintenus dans une liste: `liste_pays`. Cette liste supporte trois opérations :

- `liste_pays.taille()` qui retourne le nombre de pays dans la liste,
- `liste_pays.plus_grande_admin()` qui retourne le pays de la liste qui a la plus grande puissance totale **et** le supprime de la liste (si plusieurs pays sont à égalité avec la plus grande puissance totale, l'un d'entre-eux est choisi "au hasard").
- `liste_pays.ajouter(X)` qui ajoute le pays `X` dans la liste.

De plus, les variables "pays" contiennent une description de la carte administrative du pays. Si `A` et `B` sont deux variables "pays", alors l'instruction `A.ajouter_province(B)` modifie la variable `A` pour y ajouter une nouvelle province, copie conforme du pays `B`.

On vous demande de compléter la fonction `GUERRES` qui simule les guerres successives entre les pays de la liste, jusqu'à ce qu'il ne reste plus qu'un (ou aucun) pays dans la liste.

```

fonction GUERRES() {
  while (...) // (e1)
  {
    A ← ... // (e2)
    B ← ... // (e2)
    p_tot_A ← PUISSANCE_TOTALE(A, 0)
    p_tot_B ← PUISSANCE_TOTALE(B, 0)
    if (p_tot_A > p_tot_B)
    {
      A.ajouter_province(B)
      ... // (e3)
    }
  }
}

```

Q5(q) [2 pts]	Quelle est l'expression (e1) dans la fonction GUERRES () ?
----------------------	---

Solution: `liste_pays.taille() > 1`

Q5(r) [2 pts]	Quelle est l'expression (e2) qui se trouve deux fois dans la fonction GUERRES () ?
----------------------	---

Solution: `liste_pays.plus_grande_admin()`

Q5(s) [2 pts]	Quelle est l'expression (e3) dans la fonction GUERRES () ?
----------------------	---

Solution: `liste_pays.ajouter(A)`