Deuxième épreuve d'admissibilité
Éléments de correction

Les réponses proposées dans ce document ne sont que des exemples de réponse et en aucun cas l'unique réponse attendue. Les programmes proposés sont élémentaires. Des solutions plus élégantes pouvaient être proposées. Des commentaires dans le code étaient les bienvenus.

Préambule : le sujet comporte 5 parties. L'ensemble du sujet porte sur une même thématique qui est la recherche d'une information satisfaisant certaines contraintes. Cette thématique est illustrée sur l'exemple du Mastermind, en parties II., III. et IV. Les notations sont communes à toutes les parties et sont introduites au fur et à mesure du sujet. Les parties peuvent être traitées de façon indépendante, sauf pour la partie III. où on pourra réutiliser ou admettre les notations et résultats de la partie II., et pour la partie IV. où on pourra réutiliser les notions, contenus, algorithmes et fonctions Python déterminés dans les parties II. et III.

Dans ce sujet, certains contenus sont à destination des élèves de lycée. Ces contenus sont en italique dans le sujet.

Pour certaines questions, une longueur de réponse attendue est indiquée. Nous vous invitons à respecter ces indications.

Partie I. Le trésor et la fausse pièce

Vous avez proposé à vos classes de première en spécialité NSI, le projet ci-dessous.

Vous êtes un ou une pirate qui vient de récupérer sa part du trésor constitué d'un beau tas de pièces d'or. Hélas, vous avez vent d'une supercherie : l'une des pièces n'est pas en or massif, mais en un alliage plus léger que l'or et est simplement plaquée or, pour qu'on ne puisse pas la distinguer visuellement des autres pièces. Le seul signe distinctif est qu'elle est plus légère que les autres. Vous n'avez pas à votre disposition un atelier de chimiste ni un laboratoire scientifique vous permettant d'étudier très précisément ces pièces, mais simplement une balance à plateaux sans aucun poids, comme celle donnée ci-dessous :

Pouvez-vous, avec cette simple balance, retrouver la fausse pièce ?

I.1. L'une de vos classes a été beaucoup plus rapide que les autres et les élèves ont très rapidement proposé une réponse utilisant une approche par dichotomie. Pour alimenter leur curiosité et leur créativité, vous voulez les amener à une solution, dite *par trichotomie*, utilisant un découpage en 3 ensembles (contrairement à 2 ensembles utilisés avec l'approche dichotomique). Afin de les aider, vous préparez un énoncé donné ci-dessous (partie en italique) et que vous comptez leur distribuer. **Rédiger un corrigé de cet énoncé, tel que vous le proposeriez à vos élèves.**

Exemples pour un petit nombre de pièces dans le trésor.

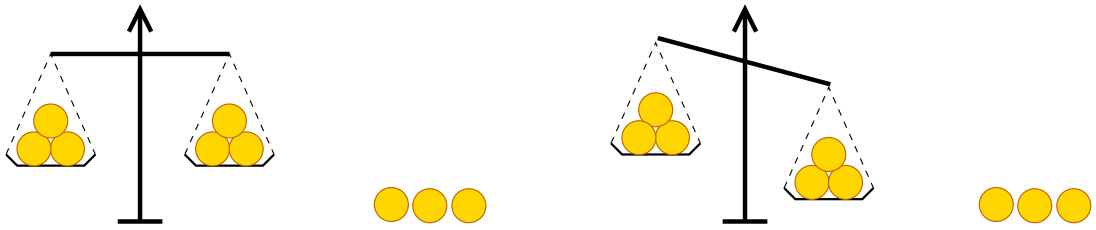
1. *Expliquer comment procéder avec 2 pièces pour retrouver la fausse pièce.*
2. *Expliquer comment procéder en une seule pesée quand il y a 3 pièces.*
3. *Expliquer comment conclure en deux pesées quand il y a 4 pièces.*

Une solution possible :

- *Avec 2 pièces : on en met une dans chaque plateau, on détermine laquelle est la plus légère.*
- *Avec 3 pièces : on en met une dans un plateau, une dans l'autre plateau, une n'est pas utilisée pour la pesée. Si la balance indique que l'un des plateaux contient une pièce plus légère, il s'agit de la fausse pièce. Si les deux pièces ont des poids égaux, la fausse pièce est celle qui n'a pas été utilisée pour la pesée.*
- *Avec 4 pièces : on met 2 pièces dans chaque plateau, on identifie le "tas" de deux pièces le plus léger et on cherche, dans ce tas, la pièce la plus légère des deux, comme précédemment, grâce à une seconde pesée. Si on veut utiliser l'algorithme qui viendra ensuite avec la division par 3 et le reste qui vaut soit 1 soit 2, il faut plutôt procéder ainsi : on fait 3 tas, deux avec une seule pièce et un avec deux pièces. On compare les deux tas d'une seule pièce en les pesant :*

soit ils sont identiques et dans ce cas la pièce fausse est dans le tas restant de deux pièces, soit l'un des deux est plus léger et c'est la fausse pièce. Si la fausse pièce est dans le tas restant, on la cherche en une seule pesée en reprenant le cas de 2 pièces déjà vu.

Examinons maintenant comment faire avec 9 pièces. On procède en divisant les 9 pièces en 3 tas, et selon la réponse de la balance on identifie le tas contenant la pièce plus légère, comme indiqué sur la figure 1.



(a) Premier cas (b) Second cas

FIGURE 1 – Exemple avec 9 pièces.

Dans le cas illustré en figure 1a, le tas contenant la pièce la plus légère est le tas qui n'est pas sur un plateau. On identifie, en une pesée supplémentaire, la pièce la plus légère parmi les 3 pièces de ce tas.

Dans le cas illustré en figure 1b, le tas contenant la pièce la plus légère est le tas qui est sur le plateau de gauche. On identifie, en une pesée supplémentaire, la pièce la plus légère parmi les 3 pièces de ce tas.

On peut donc conclure en deux pesées. Examinons maintenant le cas général d'un trésor de n pièces.

Algorithme dans le cas général d'un trésor de n pièces (n est une donnée d'entrée de l'algorithme).

4. *Dans le cas où n est une puissance de 3 ($n = 3^k$, k entier naturel) : écrire un algorithme itératif qui permet d'identifier, parmi n pièces, la pièce fausse qui est plus légère que les autres.*

Une solution possible :

```
Entrées : n le nombre de pièces, pièces le tableau contenant n pièces
# par convention, comme en Python, les indices du tableau "pièces" vont de 0 à n-1
k = log (n)/log (3)
tas = pièces[0:n]
pour i = k en décroissant jusqu'à 1 faire
  # taille du tas courant : 3^i, on le découpe en 3 sous-tas de tailles égales
  tas1 = tas[0:3^(i-1)]
  tas2 = tas[3^(i-1):2*3^(i-1)]
  tas3 = tas[2*3^(i-1):3^i]
  # peser et comparer tas1 et tas2
  si (tas1 est plus léger que tas2) alors
    tas = tas1 # l'algorithme continuera avec comme nouveau tas de pièces tas1
  sinon si (tas1 est plus lourd que tas2) alors
    tas = tas2 # l'algorithme continuera avec comme nouveau tas de pièces tas2
  sinon # c'est tas3 qui contient la pièce plus légère
    tas = tas3 # l'algorithme continuera avec comme nouveau tas de pièces tas3
renvoyer tas # tas ne contient qu'une seule pièce
```

ou bien, si on veut travailler uniquement avec les indices du tableau contenant les pièces :

```
Entrées : n le nombre de pièces, pièces le tableau contenant n pièces
k = log (n)/log (3)
début = 0; fin = n
pour i = k en décroissant jusqu'à 1 faire
  # tas1 : pièces[début1:fin1]
  début1 = début ; fin1 = début1 + 3^(i-1)
  # tas2 : pièces[début2:fin2]
  début2 = fin1 ; fin2 = début2 + 3^(i-1)
  # tas3 : pièces[début3:fin3]
  début3 = fin2 ; fin3 = fin
  si (pièces[début1:fin1] est plus léger que pièces[début2:fin2]) alors
    début = début1; fin = fin1 # le nouveau tas est tas1
  sinon si (pièces[début1:fin1] est plus lourd que pièces[début2:fin2]) alors
    début = début2; fin = fin2 # le nouveau tas est tas2
  sinon
    début = début3; fin = fin3 # le nouveau tas est tas3
renvoyer pièces[début]
```

5. *Expliquer comment votre algorithme fonctionne avec 27 pièces.*
   Aide : on devrait obtenir le résultat en 3 pesées.

Une solution possible : En images pour commencer.
Première pesée :

Dans le cas de gauche, on recommence avec le tas de 9 pièces posé à côté de la balance. On conclut en deux pesées supplémentaires.
Dans le cas de droite, l'un des tas sur la balance est plus léger que l'autre (celui de gauche dans notre dessin). On recommence avec ce tas de 9 pièces le plus léger. On conclut en deux pesées supplémentaires.

Revenons à l'algorithme : *comment fonctionne-t-il avec 27 pièces au départ ?*
`k=3; début=0 ; fin=27`
Il y aura donc `k=3` *itérations.*

- *Première itération : puisque* `début0=1` *et* `fin1=0 + 3^2`*, le premier tas est* `tas1 = pièces[0:9]`*; on détermine ensuite que* `début2 = 9` *et* `fin2 = 9+3^2=18` *et donc* `tas2 = pièces[9:18]`*; enfin,* `début3=18` *et* `fin3=fin` *et* `tas3=pièces[18:27]`*. Chaque tas contient 9 pièces. Supposons que, après la pesée, on a déterminé que c'est le 2e tas qui contient la fausse pièce, dans ce cas* `début=9` *et* `fin=18` *à la fin de la première itération : le nouveau tas contient 9 pièces.*
- *Deuxième itération :* `début1=début` *et* `fin1=début+3^1=début1+3` *: le premier tas contient maintenant 3 pièces. De la même façon, les deuxième et troisième tas contiennent 3 pièces chacun. Avec notre exemple,* `début1=9`*,* `fin1=12`*,* `début2=12`*,* `fin2=15`*,* `début3=15` *et* `fin3=18`*. Après la 2e pesée, à la fin de la deuxième itération, le nouveau tas contient donc 3 pièces.*
- *Troisième itération : on découpe en 3 sous-tas qui contiennent 1 pièce chacun, en effet* `début1=début`*,* `fin1=début1 + 3^0 =début1`*, ensuite* `début2=fin1` *et* `fin2=début2+1`*, enfin* `début3=fin2` *et* `fin3=fin`*.*

On se retrouve dans la situation de la question 2, à laquelle on a répondu avec une seule pesée.

Les 3 itérations sont détaillées par souci de clarté, on aurait aussi pu dire qu'avec un tas à 9 pièces, on sait conclure en 2 pesées comme expliqué après la Figure 1, mais cela n'aurait pas permis de vérifier les calculs d'indices jusqu'à la fin.

6. *Dans le cas d'un nombre quelconque de pièces, adapter l'algorithme précédent pour identifier la fausse pièce.*

Une solution possible :

```
Entrées : n le nombre de pièces, pièces le tableau contenant n pièces
k = ceil (log (n)/log (3)) # ceil = partie entière supérieure
début = 0; fin = n
tant que (fin-début) >= 3 faire
  # division euclidienne de (fin-début) par 3 : fin-début = 3*p+q avec 0<=q<=2
  p = (fin - début)/3; q = reste ((fin-début)/3)
  si q == 0 alors
    # tas1 : pièces[début1:fin1] contient p pièces
    début1 = début ; fin1 = début1 + p
    # tas2 : pièces[début2:fin2] contient p pièces
    début2 = fin1 ; fin2 = début2 + p
    # tas3 : pièces[début3:fin3] contient p pièces
    début3 = fin2 ; fin3 = fin
  sinon si q == 1 alors
    # tas1 : pièces[début1:fin1] contient p pièces
    début1 = début ; fin1 = début1 + p
    # tas2 : pièces[début2:fin2] contient p pièces
    début2 = fin1 ; fin2 = début2 + p
    # tas3 : pièces[début3:fin3] contient p+1 pièces
    début3 = fin2 ; fin3 = fin
  sinon # q == 2
    # tas1 : pièces[début1:fin1] contient p+1 pièces
    début1 = début ; fin1 = début1 + p + 1
    # tas2 : pièces[début2:fin2] contient p+1 pièces
    début2 = fin1 ; fin2 = début2 + p + 1
    # tas3 : pièces[début3:fin3] contient p pièces
    début3 = fin2 ; fin3 = fin

  # tas1 et tas2 contiennent le même nombre de pièces et peuvent donc être comparés
  si (pièces[début1:fin1] est plus léger que pièces[début2:fin2]) alors
    début = début1; fin = fin1 # le nouveau tas est tas1
  sinon si (pièces[début1:fin1] est plus lourd que pièces[début2:fin2]) alors
    début = début2; fin = fin2 # le nouveau tas est tas2
  sinon
    début = début3; fin = fin3 # le nouveau tas est tas3

# ici pièces[début:fin] contient 1 ou 2 pièces
si (fin-début == 1) alors
  renvoyer  pièce[début]
sinon # fin-début == 2
  # mettre une pièce dans chaque plateau de la balance
  si pièces[début] est plus légère que pièces[début+1] alors
    renvoyer pièce[début]
  sinon
    renvoyer pièce[début+1]
```

7. *Afin de comparer avec la recherche dichotomique, remplir le tableau suivant, en indiquant, pour chaque méthode et pour différentes valeurs de n, le nombre maximal de pesées.*

n	*Dichotomie*	*Trichotomie*
2		
3		
6		
8		
9		
16		
27		

Quelle est, d'après vous, la méthode la plus efficace en nombre de pesées ?

Une solution possible :

n	*Dichotomie*	*Trichotomie*
2	*1*	*1*
3	*1*	*1*
6	*2*	*2*
8	*3*	*2*
9	*3*	*2*
16	*4*	*3*
27	*4*	*3*

Pour 2 et 3 pièces, la réponse a été donnée dans les premières questions.
Pour 6 pièces : avec une méthode par dichotomie, on compare 2 tas de 3 pièces avec la première pesée, puis on identifie la fausse pièce dans un tas de 3 pièces en une pesée comme précédemment ; avec une méthode par trichotomie, on compare 2 tas de 2 pièces avec la première pesée, puis on identifie la fausse pièce dans un tas de 2 pièces avec une seconde pesée.
Pour 8 pièces : avec une méthode par dichotomie, on compare 2 tas de 4 pièces avec la première pesée, puis en 2 pesées supplémentaires on identifie la fausse pièce dans un tas de 4 pièces ; avec une méthode par trichotomie, on compare 2 tas de 3 pièces avec la première pesée, puis on identifie la fausse pièce dans un tas de 2 ou 3 pièces en une pesée comme précédemment.
Pour 9 pièces : avec une méthode par dichotomie, on compare 2 tas de 4 pièces avec la première pesée, puis en 2 pesées supplémentaires on identifie la fausse pièce dans un tas de 4 pièces ; avec une méthode par trichotomie, on a vu dans l'exemple avant la question 4 que 2 pesées suffisent.
Pour 16 pièces : avec une méthode par dichotomie, on compare 2 tas de 8 pièces avec la première pesée, puis en 3 pesées supplémentaires on identifie la fausse pièce dans un tas de 8 pièces ; avec une méthode par trichotomie, on compare 2 tas de 5 pièces avec la première pesée, puis on identifie la fausse pièce dans un tas de 5 ou 6 pièces en 2 pesées comme précédemment.
Pour 27 pièces : avec une méthode par dichotomie, on compare 2 tas de 14 pièces avec la première pesée, puis on compare 2 tas de 7 pièces avec une 2e pesée, ensuite on compare 2 tas de 3 pièces avec une troisième pesée et enfin on détermine la fausse pièce dans un tas de 3 pièces en une quatrième et dernière pesée ; avec une méthode par trichotomie, on a vu en question 5 que 3 pesées suffisent.

I.2. La question 6 est difficile pour les élèves. En particulier, le terme *adapter* n'est pas suffisamment précis. **Proposer, en les justifiant, une ou plusieurs aides que vous pourriez proposer aux élèves afin de leur permettre de répondre à cette question.**
(Répondre en 6 lignes maximum.)

Une solution possible : Il faut amener les élèves à distinguer selon le reste de la division euclidienne par 3. On peut suggérer plus de cas particuliers à examiner, au lieu de se limiter à 2, 3, 4, comme 5, 6, 7 et 8, ou également demander, après l'exemple avec 27 pièces, de résoudre les cas avec 25 et 26 pièces. On peut alors demander d'expliciter la division euclidienne par 3, et la valeur du reste, pour ces nombres de pièces. On peut aussi leur proposer explicitement de traiter séparément les 3 cas de valeur du reste par la division euclidienne par 3.

I.3. **En transposant les notions de variants et d'invariants, vues pour l'algorithme de recherche dichotomique dans un tableau trié en classe de 1ère NSI, donner à vos élèves un schéma de la preuve de correction de l'algorithme par trichotomie.**
(Répondre en 10 lignes maximum.)

Une solution possible :
- un invariant, illustrant le schéma de l'algorithme, est que le tas considéré est toujours celui qui contient la pièce plus légère, c'est-à-dire que l'indice de la pièce la plus légère est toujours compris entre `début` et `fin`. Si c'est vérifié en entrée de l'algorithme, on peut vérifier que les 3 tas `tas1`, `tas2` et `tas3` forment une partition du tas `pièces[début:fin]` en début de boucle et on n'a donc perdu aucune pièce. Le tas retenu définissant les nouvelles valeurs de `début` et `fin` est bien le plus léger (contenant la pièce plus légère) : l'invariant est donc vrai en fin de boucle.
- le variant est la taille du tas $t =$ `fin - début` qui devient à l'étape suivante $t \leqslant \lceil t/3 \rceil$, il s'agit d'un entier naturel strictement décroissant, ce qui permet de démontrer la terminaison de l'algorithme.

Partie II. Mastermind : le joueur humain devine

Vous proposez à vos élèves de Terminale, en spécialité NSI, un projet qui réalise le jeu du Mastermind. Les règles du Mastermind sont fournies en annexe 1. Dans ce projet, les élèves travailleront sur deux situations :
- le joueur (humain) tente de découvrir une configuration de pions cachée par la machine (le joueur est alors appelé le décodeur et la machine le codificateur) ;
- le joueur (humain) cache une configuration de pions à la machine qui doit la découvrir (le joueur est alors appelé le codificateur et la machine le décodeur).

Dans cette partie II., on considère que le joueur est le décodeur et que la machine est le codificateur.

Dans la suite, on notera N le nombre de couleurs et P le nombre de pions cachés. On appellera *configuration* un ensemble de P pions dont les positions et les couleurs sont déterminées. C est le nombre maximal de configurations que le décodeur peut proposer.

Vous préparez un énoncé à distribuer aux élèves qui a pour but de les guider dans la réalisation de ce projet.

II.1. Parmi les structures de données de type liste, p-uplet et dictionnaire, vous choisissez la structure de données de liste pour représenter une configuration. **Donner les arguments qui seront présentés aux élèves pour justifier ce choix.**

Une solution possible : Les critères à considérer sont i) le fait que la position de chaque élément importe, ii) que ce soit modifiable (ce qui élimine le p-uplet sauf si l'usage du p-uplet avec des recopies est envisagée) et iii) rapide à parcourir mais sans clé (ce qui élimine le dictionnaire).

II.2. **Donner la correction de l'énoncé, donné ci-dessous (partie en italique), qui sera proposé aux élèves.** Pour la question 2.d de l'énoncé, seules les lignes à compléter peuvent être données sur votre copie avec les numéros de lignes associés.

1. *La configuration choisie par le codificateur est générée aléatoirement.*
   Écrire une fonction Python qui renvoie une configuration aléatoire selon une loi uniforme sur l'ensemble des configurations. L'en-tête de cette fonction sera le suivant :
   `def genere_configuration(n_couleurs,n_positions)`
   Vous pourrez utiliser la fonction `randint` *de la bibliothèque* `random` *donnée en annexe 2 pour choisir aléatoirement une couleur, qui est représentée par un entier entre* 1 *et* N.
   Il est possible d'écrire ce programme de façon itérative et de façon récursive. Vous écrirez les deux versions.

Une solution possible :
Version itérative : on place les couleurs successivement sur chaque position.

```
def genere_configuration(n_couleurs=6, n_positions=4):
    """Renvoie une configuration aléatoire générée
       uniformément sur l'ensemble de toutes les
       configurations possibles
    paramètres :
        n_couleurs : nombre de couleurs
        n_positions : taille de la configuration
    Valeur de retour : liste d'entiers dans {1, n_couleurs}
                                        configuration
    Effet de bord : aucun
    Méthode : iteration
    >>> seed(43)
    >>> genere_configuration
```

```
    [1, 3, 6, 2]
    """
    L=[]
    if ( n_positions == 0) :
        return []
    else :
        for i in range(n_positions):
            L.append(randint(1,n_couleurs))
    return L
```

Version récursive : une configuration de taille n est composée d'une configuration de taille n-1 et d'une couleur en position n.

```
def genere_configuration_recursif(n_couleurs=6, n_positions=4):
    if ( n_positions == 0) :
        return []
    else :
        return (genere_configuration(n_couleurs,n_positions-1) +
                [randint(1,n_couleurs)] )
```

2. *Programmation, en Python, du jeu Mastermind quand le décodeur est le joueur humain*

   (a) *Écrire une fonction Python itérative calculant et retournant le nombre de pions bien placés entre deux configurations, c'est-à-dire les pions de même couleur et de même position entre les deux configurations (nombre de fiches noires). On l'appellera* `n_bien_places` *et son en-tête est :*

   `def n_bien_places(configuration_1,configuration_2)`

Une solution possible : *L'objectif est de parcourir simultanément les deux configurations avec un même indice.*

```
def n_bien_places(configuration_1,configuration_2):
    """ calcule le nombre de couleurs communes aux
        deux configurations et en même position
    paramètres :
        configuration_1 , configuration_2 : configurations
                de même taille
     Valeur de retour : entier ,
    Effet de bord : aucun
    >>> n_bien_places([1,2,3,4],[4,3,2,1])
    0
    >>> n_bien_places([1,2,3,4],[1,3,2,4])
    2
    >>> n_bien_places([1,2,3,4],[1,2,3,4])
    4
    """
    s=0
    n_positions = len(configuration_1)
    for i in range(n_positions):
```

```
        if configuration_1[i] == configuration_2[i] :
            s = s + 1
    return s
```

(b) *Donner un algorithme qui calcule le nombre de pions communs entre deux configurations (l'ordre des pions dans la configuration n'a pas d'importance). Si nécessaire, vous pourrez utiliser une structure de données auxiliaire en expliquant votre choix. La fonction Python qui implante cet algorithme sera appelé* `n_pions_communs` *et son en-tête est :*

```
def n_pions_communs(configuration_1,configuration_2)
```

L'implantation de cette fonction n'est pas demandée.

Une solution possible : *Il faut une méthode de parcours des 2 configurations simultanément. Une solution consiste à utiliser une structure de dictionnaire : dans une première phase on associe à chaque configuration une liste de couleurs et de nombre de pions de cette couleur puis on compare ces deux dictionnaires.*
La solution proposée ci-dessous parcourt la première configuration et supprime le cas échéant la couleur dans la deuxième configuration (attention il faut travailler sur une copie) en incrémentant un compteur.

```
import copy

def n_pions_communs(configuration_1,configuration_2):
    """ calcule le nombre de couleurs communes aux
         deux configurations incluant les répétitions
    parametres :
        configuration_1 , configuration_2 :
          configurations
    valeur de retour : entier
    Effet de bord : aucun
    """
    configuration_2_copy = copy.deepcopy(configuration_2)
    n_communs = 0
    for c in configuration_1 :
        if c in configuration_2_copy :
            n_communs = n_communs +1
            configuration_2_copy.remove(c)
    return n_communs
```

(c) *En déduire une fonction Python renvoyant le nombre de fiches blanches et de fiches noires qui sont déterminées lorsque deux configurations sont comparées. Pour rappel, le nombre de fiches noires correspond au nombre de pions étant bien placés (et donc de la même couleur) et le nombre de fiches blanches correspond au nombre de pions de la bonne couleur qui ne sont pas bien placés. On appellera la fonction* `n_noirs_blancs` *et son en-tête est :*

```
def n_noirs_blancs(configuration_1,configuration_2)
```

Une solution possible : *Le nombre de fiches blanches est égal au nombre de pions communs moins le nombre de pions bien placés.*

```
def n_noirs_blancs(configuration_1,configuration_2) :
    a = n_bien_places(configuration_1,configuration_2)
    b = n_pions_communs(configuration_1,configuration_2)
    return (a,b-a)
```

(d) *Compléter le squelette de la fonction Python interactive, donnée ci-dessous, permettant d'effectuer une partie de Mastermind dans laquelle le codificateur est la machine et le décodeur est le joueur humain. Cette fonction comprend la génération d'une configuration à faire deviner, la saisie des propositions successives de configurations par l'utilisateur et les réponses correspondantes de la machine via les fiches blanches et noires. Les "..." données dans le squelette sont à compléter et peuvent représenter des variables, des expressions, des instructions ou des suites d'instructions.*

```
1  def joueur_devine(n_couleurs=6,n_positions=4,n_essais=10):
2
3    configuration_cachee =
4          genere_configuration(n_couleurs,n_positions)
5    print("Taille de la configuration cachée: ",n_positions)
6
7    i = ...       # nombre d'essais déja réalisés
8    trouve = ... # variable booléenne qui indique si la
9                  # configuration a été trouvée
10
11   while ... :
12       ...
13       m = input("Essai "+str(i)+" : ")
14           # le joueur entre une configuration
15       ... # conversion de la configuration entrée
16           # dans le bon type
17     print("     "+str(...)+" bien placés - "
18                    +str(...)+" mal placés")
19     trouve = ...
20
21   if trouve :
22     return "Bravo vous avez trouvé en "
23                 + str(...) + " essais"
24   else :
25      return "Perdu vous avez épuisé vos " + str(...)
26          + " tentatives, bonne réponse = "
27          + str(configuration_cachee)
```

Une solution possible : *Un exemple de solution*

```
1  def joueur_devine(n_couleurs=6, n_positions=4,
2                                  n_essais=10):
3      configuration_cachee =
4        genere_configuration(n_couleurs=6, n_positions=4)
```

```
5     print("Taille de la configuration cachée : ",
6               n_positions)
7     i = 0
8     trouve = False
9
10
11    while (not(trouve) and (i < n_essais)) :
12        i = i + 1
13        m = input("Essai "+str(i)+" : ")
14        configuration_proposee =
15                          [int(c) for c in m.split()]
16        reponse = n_noirs_blancs(configuration_cachee,
17                                 configuration_proposee)
18        print("                ",reponse)
19        trouve = (reponse[0] == n_positions)
20
21    if trouve :
22        return ("Bravo vous avez trouvé en " + str(i) +
23                    " essais")
24    else :
25        return ("Perdu vous avez épuisé vos " +
26              str(i) + " tentatives, bonne réponse = " +
27              str(configuration_cachee) )
```

II.3. L'année précédente, vous aviez aussi proposé, en TP, à une classe de terminale, de programmer le jeu du Mastermind avec 5 couleurs, 5 positions et un nombre maximal de 10 essais pour le décodeur, mais vous ne leur aviez pas fourni l'énoncé précédent que vous avez préparé cette année. Un groupe d'élèves vous a rendu la fonction donnée en annexe 4. **Analyser cette production, en se référant aux numéros de ligne si nécessaire.**

(Répondre en 20 lignes maximum.)

Une solution possible : Ce code fonctionne probablement mais est compliqué à lire et à comprendre. Les points à reprendre sont :

- **la structure du code :** répétition du traitement de chaque jeton de couleur l'un après l'autre ; présence de compteurs `bon` et `mauvais` qui ne sont pas utilisés dans le code. Ils sont seulement affichés pour le joueur alors qu'ils pourraient permettre de détecter la victoire par exemple ;
- **les notations et commentaires :** problème de lisibilité, avec des jetons qui s'appellent `a`, `b` ... `e` pour les jetons de couleur du joueur et en regard `aa`, `bb` ... `ee` pour les jetons de la configuration à deviner ; absence totale de commentaires des lignes 4 à 26 ;
- **les expressions conditionnelles :** beaucoup de répétitions et énumérations complètes de tests entre les lignes 7 à 27 et ligne 39 ; on attendait des "elif" plutôt que `if a==aa` suivi de `if aa!=a` ;
- **les duplications :** les lignes 7 à 10 sont copiées pour former les lignes 11-14 puis 15-18, 19-22 et enfin 23-26 ;

- **la portée des variables et les arguments de fonction :** la fonction ne prend aucun argument en entrée et ne renvoie rien. Il y a 3 variables globales `tour`, `bon` et `mauvais`, ce qui ne se justifie pas particulièrement pour les deux dernières qui ne sont utilisées que de façon interne à cette fonction.

II.4. **Proposer une remédiation pour ce groupe d'élèves sur les points qui vous paraissent importants.**
(Répondre en 10 lignes maximum.)

Une solution possible : Voici une proposition de remédiation pour chacun des points évoqués en réponse à la question II.3, mais toutes les réponses n'étaient pas attendues :
- **structure du code :** demander une solution qui s'écrive en un nombre limité de lignes, avec la limite qui est précisée ;
- **notations et commentaires :** échanger les codes et demander à un groupe d'élèves, qui ne sont donc pas les auteurs du code, d'effectuer une modification ;
- **expressions conditionnelles :** travailler sur la vérification que tous les cas ont été traités, et une seule fois, en déroulant sur des exemples à la main ; travailler sur l'évaluation des conditions booléennes et sur l'élimination de la redondance ;
- **duplications :** demander une version du jeu, au choix, pour enfant (4 jetons et 8 tours) ou pour expert·e (8 jetons et 15 tours) en indiquant que la version effectuée peut facilement être transformée pour l'autre version ;
- **portée des variables et arguments de fonction :** l'objectif est de faire écrire des fonctions avec des paramètres. Pour cela on pourrait par exemple demander à réfléchir à un serveur multi-joueur, pour voir la limite de l'utilisation de variables globales dans un projet plus gros.

Partie III. Mastermind : la machine devine

Dans cette partie, il s'agit d'étudier le cas où le joueur (humain) joue le rôle du codificateur et la machine celui du décodeur.
Vous réfléchissez aux différentes questions et activités que vous aimeriez aborder avec vos élèves. Répondez aux questions suivantes.

III.1. Étude de l'ensemble de toutes les configurations

III.1.a. **Donner la formule du nombre de configurations possibles en fonction des paramètres N et P du jeu.**

Une solution possible : Il y a N couleurs possibles pour chacun des P pions, ce qui donne N^P configurations possibles.

III.1.b. **Avec les valeurs habituelles pour le jeu, qui sont $N = 6$ et $P = 4$ pour les plus jeunes joueurs et joueuses, et dans la version classique $N = 8$ et $P = 5$, combien y a-t-il de configurations possibles ?** Vous pourrez vous aider de la table de calculs donnée en annexe 3.

Une solution possible : La table de l'annexe 3 nous permet de dire que la version pour les plus jeunes comporte $N^P = 6^4 = 1296$ configurations et que la version classique comporte $N^P = 8^5 = 32768$ configurations.

III.1.c. **Est-il envisageable et raisonnable de créer et de mémoriser toutes les configurations dans ces deux cas de figures ?**
Quelle comparaison pourriez-vous proposer à vos élèves pour qu'ils puissent arriver à la même conclusion que vous ? Vous pourrez supposer qu'un entier est codé sur 64 bits.

Une solution possible : Quand $N = 6$ et $P = 4$, il y a 1296 configurations possibles, ce qui est très peu à mémoriser. Si on suppose qu'il faut 4 octets pour mémoriser la couleur et 4 octets pour gérer la position du pion dans la liste, il faut donc 32 octets par configuration et un peu plus de 40 ko (kilo-octets) pour mémoriser toutes les configurations possibles, ce qui est infime aujourd'hui. Quand $N = 8$ et $P = 5$, il y a 32768 configurations possibles. Toutes les stocker en mémoire requiert un peu plus de 1 Mo, ce qui est également très peu. À titre de comparaison, une clé USB contient 16 Go, c'est-à-dire 16000 fois 1 Mo.

III.2. Vous avez trouvé sur le Web un projet, traitant du Mastermind, déjà réalisé. Pour savoir si vous pouvez vous en inspirer, vous l'analysez. **Que fait la fonction Python suivante ? Prouver qu'elle n'omet aucune configuration.**

```
def mystere(n_couleurs = 6,n_positions = 4) :
    if ( n_positions == 0) :
        return [[]]
    else :
        l = []
        for element in mystere(n_couleurs,n_positions-1):
            for couleur in range(1,n_couleurs+1) :
                l.append(element + [couleur])
        return l
```

Une solution possible : Cette fonction énumère toutes les configurations possibles, dans le cas le plus simple par défaut. On peut lui donner un nom plus explicite : `enumerer_configurations(n_couleurs,n_positions)`.
Attention car les couleurs sont indicées de 1 à `n_couleurs`.

```
    if ( n_positions == 0) :
        return [[]]
    else :
        l = []
        for element in
             enumerer_configurations(n_couleurs,n_positions-1):
            for couleur in range(1,n_couleurs+1) :
                l.append(element + [couleur])
        return l
```

La preuve de cette fonction récursive se décompose en une preuve de correction partielle et une preuve de terminaison. On veut montrer que la fonction `enumerer_configurations(n_couleurs, n_positions)` renvoie une liste de toutes les configurations possibles (chaque configuration apparaît une et une seule fois dans la liste).

On suppose que les paramètres d'appels vérifient les préconditions : `n_couleurs` est un entier supérieur ou égal à 0 ; `n_positions` est un entier supérieur ou égal à 1.

La fonction `enumerer_configurations` est récursive il faut donc prouver la terminaison et la correction partielle.

Terminaison Le paramètre d'appel `n_positions` est un entier positif, strictement décroissant lors de l'appel interne dans la fonction. Le cas `n_positions == 0` garantit l'arrêt de la séquence des appels récursifs dans l'arbre des appels.

Correction partielle On fait un raisonnement par induction. Lorsque le paramètre d'appel `n_positions` est égal à 0 la fonction retourne la liste vide, ce qui est correct. Supposons que les appels internes à la fonction soient corrects.

Soit une configuration quelconque $c_n = [i_1, \cdots, i_n]$ donnée sur n positions. Comme toute configuration ayant n positions se décompose en une configuration sur les $n-1$ premières positions ainsi qu'une couleur sur la dernière position, $c_n = [i_1, \cdots, i_{n-1}]+[i_n]$. La correction de l'appel interne entraine que $[i_1, \cdots, i_{n-1}]$ apparaît une et une seule fois dans la liste renvoyée par l'appel `enumerer_configurations(n_couleurs,n_positions-1)`. La couleur i_n n'est utilisée qu'une et une seule fois dans la boucle interne `for couleur in range(1,n_couleurs+1)`. Par conséquent la configuration c_n n'apparaîtra qu'une et une seule fois dans la liste renvoyée par l'appel principal. La correction partielle est donc acquise.

La terminaison et la correction partielle étant démontrées, le programme est correct.

Un dessin de l'arbre des appels permet d'illustrer le fonctionnement de cette fonction.

Comme cela a été fait dans ce projet, vous décidez vous aussi d'utiliser une liste pour stocker toutes les configurations possibles.

III.3. **Donner un plan de cours, de niveau NSI Terminale, sur les structures de données linéaires, comme les listes, piles et files, en explicitant les exemples utilisés comme illustrations.** Un extrait du programme concernant ce sujet est donné dans la table 1.
(Répondre en 10 lignes maximum.)

Contenu	**Capacités attendues**	**Commentaires**
Listes, piles, files : structures linéaires	Distinguer des structures par le jeu des méthodes qui les caractérisent. Choisir une structure de données adaptée à la situation à modéliser.	On distingue les modes FIFO (first in first out) et LIFO (last in first out) des piles et des files. Les listes n'existent pas de façon native en Python.

TABLE 1 – Extrait du programme de NSI Terminale - Structures de données.

Une solution possible : Un plan de cours possible est d'indiquer, pour chaque structure de données, i) une spécification de la structure avec les opérations associées, ii) au moins un ou deux exemples à donner aux élèves et iii) la description d'une activité à réaliser avec les élèves : **Piles :** i) les éléments sont ajoutés ou retirés à un endroit de la pile, souvent

appelé "haut" de la pile ; ii) exemple avec une pile d'assiettes ou avec Annuler / Rétablir dans un éditeur de texte ; iii) activité pour "renverser" un mot : on empile les lettres du mot une à une et on les ressort ensuite de la pile.
Files : i) les éléments sont ajoutés en fin de file ou retirés en début de file ; ii) exemple avec une file d'attente dans un magasin ou avec la file d'attente d'impression ; iii) activité de programmation d'un parcours en largeur d'un arbre ou d'un graphe avec une file.
Listes : i) les éléments peuvent être ajoutés ou retirés à n'importe quel endroit dans une liste. Les listes permettent d'implanter les structures de données comme les piles et les files ; ii) exemple avec une chaîne humaine ; iii) activité qui permet d'insérer une carte dans une liste de cartes triées ou de retirer une carte.

III.4. Vous réfléchissez aux différents éléments à donner et aux programmes à demander aux élèves lorsque la machine tente de deviner la configuration choisie par le codificateur.

III.4.a. Vous supposez que la machine a proposé une configuration, nommée `tentative`, et a reçu du codificateur, en retour, le nombre de fiches noires, noté `n_bien_places`, et le nombre de fiches blanches, noté `n_mal_placés`.

Vous comptez demander à vos élèves un algorithme qui commence avec l'ensemble des configurations qui étaient encore candidates avant la proposition de la configuration `tentative` (ensemble qui sera appelé `configurations_possibles`) et qui le modifie pour qu'il ne contienne plus que l'ensemble des configurations qui pourront être candidates au tour suivant, une fois les fiches noires et blanches reçues du codificateur suite à la proposition de `tentative`. **Écrire cet algorithme pour pouvoir donner un corrigé à vos élèves.** La fonction Python correspondante s'appellera `reduction_configurations_possibles`. Il ne vous est pas demandé d'écrire cette fonction, mais seulement de **donner son en-tête avec ses paramètres d'entrée.**

Une solution possible : La fonction prend en argument une liste de configurations et le résultat du joueur. Elle retourne une nouvelle liste de configurations admissibles pour la réponse du joueur.

```
def reduction_configurations_possibles(
            configurations_possibles,
            tentative,
            n_bien_places,
            n_mal_places):
```

Il suffit de remarquer que la fonction de comparaison de deux configurations `n_noirs_blancs` peut être employée ici : si la comparaison entre la configuration proposée `tentative` et une `configuration` de la liste des configurations candidates diffère du résultat donné par le joueur `(n_bien_places,n_mal_places)` alors cette configuration ne peut pas être la configuration à deviner.
On donne ici la portion de programme python qui répond à la question.

```
configurations_possibles2 = []
  for configuration in  configurations_possibles:
    if (n_noirs_blancs(configuration,tentative) ==
                  (n_bien_places,n_mal_places)):
      configurations_possibles2.append(configuration)
```

```
configurations_possibles = configurations_possibles2
```

ou plus concis en utilisant les listes par compréhension :

```
configurations_possibles =
    [configuration for configuration in
          configurations_possibles if
             (n_noirs_blancs(configuration,tentative) ==
                   (n_bien_places,n_mal_places)) ]
```

III.4.b. **Quel(s) argument(s) donneriez-vous à vos élèves pour qu'ils choisissent la liste, parmi les structures linéaires de type liste, pile et file, afin de stocker l'ensemble des configurations possibles ?**

Une solution possible : Il faut les faire réfléchir aux opérations qui vont être effectuées sur cette structure de données. Ces opérations sont la "création de toutes les configurations possibles" puis "suppression de celles qui ne coïncident pas avec les nombres de fiches noires et blanches", suppression qui peut être n'importe où dans la structure de données. N'importe laquelle des structures proposées (liste, file ou pile) permet d'ajouter facilement une configuration, puisque peu importe sa position. En revanche, la structure de données qui permet la suppression d'éléments n'importe où (et pas uniquement en début ou en fin) est la liste.

III.4.c. **Quel argument donneriez-vous à vos élèves pour qu'ils comprennent que n'importe quelle configuration peut être choisie dans l'ensemble `configurations_possibles` pour le tour suivant ?**

Une solution possible : Insister sur le fait que toutes les configurations dans l'ensemble `configurations_possibles` satisfont les contraintes (sur le nombre de fiches noires et de fiches blanches) et peuvent donc être candidates pour le tour suivant. Comme on n'a aucune information complémentaire, n'importe laquelle peut être choisie. En revanche, si on savait comment le choix d'une configuration permet de réduire la taille de l'ensemble au tour suivant, on aurait un critère de choix, On peut faire une analogie avec la recherche d'une personne dans une population : par exemple le sexe permet de diviser par 2 la population restante à l'étape suivante, la couleur de cheveux coupe en 2/3 (de brun·e·s) et 1/3 (d'autres couleurs). Avec une telle connaissance, on a intérêt à choisir le critère qui assure la plus petite population pour le tour suivant. Pour notre problème, on n'a pas cette indication pour guider notre choix.

Puisque n'importe quelle configuration de l'ensemble `configurations_possibles` peut être proposée, une solution simple est de proposer la première configuration de la liste. Une autre solution est de choisir au hasard une configuration dans cette liste. Voici les extraits python correspondants.

```
tentative = configurations_possibles[0]
```

ou

```
tentative = random.choice(configurations_possibles)
```

III.4.d. **Écrire le corrigé de la fonction Python interactive, demandée à vos élèves, permettant d'effectuer une partie où c'est la machine qui devine la configuration choisie par le joueur.** Ce programme comprend la génération de toutes les configurations possibles par la machine, les propositions successives de la machine et les réponses correspondantes du joueur.
Voici son en-tête :

```
def machine_devine(n_couleurs=6,n_positions=4,n_essais=10)
```

Une solution possible : Une solution élémentaire

```
def machine_devine(n_couleurs=6, n_positions=4, n_essais=10):
  ## génération de toutes les configurations
  configurations_possibles =
    enumerer_configurations(n_couleurs,n_positions)

  ## interface utilisateur
  print("Couleurs de 1 à ",n_couleurs)
  print("Taille de la configuration cachée : ",n_positions)

  ## initialisation
  i = 0 # nombre d'itérations
  trouve = False

  while (not(trouve) and (i < n_essais)) :
    i = i + 1
    tentative = configurations_possibles[0]
    configurations_possibles.remove(tentative)

    reponse  = input("Essai "+str(i)+"    "+str(tentative)+" : " )
    reponse_joueur = [int(c) for c in reponse.split()]

    trouve = (reponse_joueur[0] == n_positions)

    if not(trouve) :
      configurations_possibles =
         [configuration for configuration in
            configurations_possibles if
              (n_noirs_blancs(configuration,tentative) ==
                reponse_joueur) ]
   if trouve :
      return ("Bravo la machine a trouvé en " + str(i)
```

```
                    + " essais " + str(tentative))
    else :
        return ("Perdu la machine a épuisé ses "
                    + str(i) + " tentatives")
```

Partie IV. Évaluation et analyse d'un projet

Vous donnez le projet de Mastermind à vos élèves. Vous les guidez avec les différentes étapes que vous avez établies dans les parties II. et III. Vous avez réservé 20h pour travailler sur ce projet avec vos élèves. À la fin du projet, les élèves devront rendre un programme qui permet d'effectuer une partie, avec le joueur (humain) qui peut être codificateur ou décodeur.

IV.1. **Proposer un barème qui vous permettra d'évaluer le travail rendu par les élèves. Expliquer les différents éléments que vous comptez évaluer ainsi que le nombre de points attribués pour chacun de ces éléments.**
(Répondre en 15 lignes maximum.)

Une solution possible :
Pour évaluer le travail rendu par les élèves, une approche possible est de se baser sur un certain nombre de compétences attendues en NSI Terminale. Ci-dessous, voici une liste de compétences qui pourraient être évaluées dans le cadre de ce projet avec un nombre de points associés, la note finale du projet étant sur 20.

- Concevoir des solutions algorithmiques : 5 points
- Traduire un algorithme dans un langage de programmation : 5 points
- Développer des processus de mise au point de validation des programmes : 2,5 points
- Qualité de la réalisation logicielle / Présence de commentaires dans les programmes : 2,5 points
- Savoir présenter le travail réalisé et sa solution : 2,5 points
- Faire preuve d'autonomie : 1,5 points
- Faire preuve de créativité : 1 point

Une élève vous a rendu un projet dont un extrait figure en annexe 5. Elle est allée beaucoup plus loin que ce qui était attendu sur ce projet. Notamment, elle a fait jouer la machine contre elle-même et elle a estimé expérimentalement le nombre moyen d'essais effectués par la machine pour trouver la configuration cachée. Pour cela, la machine a généré des configurations aléatoires avec un nombre de couleurs N et une taille P fixés et l'élève a compté le nombre d'essais effectués par la machine pour trouver chaque configuration. Elle a limité à 10 le nombre maximal d'essais.

IV.2. **Que pensez-vous de son approche qui consiste à baser son estimation du nombre moyen d'essais effectués sur plusieurs configurations aléatoires ?** Justifier votre réponse.

Une solution possible :

- Le choix de données **aléatoire** est justifié lorsque la taille de l'espace des données est trop grand pour pouvoir faire un test exhaustif sur toutes les données. Dans le cadre du projet $N^P \leqslant 10^6$, donc une approche exhaustive est faisable.
  Lorsque la taille de l'espace des données est trop importante, un échantillonnage va permettre d'explorer cet espace des données. Pour que cette exploration soit "analysable" il faut garantir que les configurations testées soient indépendantes les unes des autres (au sens statistique) et de loi uniforme sur l'espace des données.
- La **moyenne** n'est pas forcément significative du comportement de l'algorithme. Une petite analyse de la répartition du nombre de tentatives (un histogramme) permettrait de savoir si un indice de tendance centrale comme la moyenne a un intérêt.

IV.3. **Que pensez-vous de sa remarque : "... plus le nombre de boules et le nombre de couleurs sont élevés, plus le nombre de simulations est bas..." ?** Justifier votre réponse.

Une solution possible : La **taille de l'échantillon** influe sur la qualité de l'estimateur de la valeur moyenne. Il faudrait compléter par un calcul d'écart-type (voire d'intervalle de confiance). Mais l'élève diminue la taille de son échantillon quand la taille augmente, or la variabilité augmente en fonction de la taille, donc les 2 effets se combinent pour diminuer la précision de l'estimation.

IV.4. **Dans la version pour jeunes joueurs du jeu,** $N = 6$ **et** $P = 4$**, pensez-vous qu'il est raisonnable de fixer la limite du nombre d'essais à** 10 **?** Justifier votre réponse à partir des valeurs obtenues par l'élève sur le nombre moyen d'essais.

Une solution possible : Dans ce cas, on se situe à plus du double de la valeur moyenne, cela paraît raisonnable. Néanmoins on ne sait pas si certaines configurations sont vraiment plus difficiles que d'autres.

IV.5. **Quel indicateur l'élève aurait-elle pu mesurer expérimentalement pour affiner la limite sur le nombre d'essais raisonnables à fixer ?**

Une solution possible : Deux indicateurs pourraient présenter un intérêt :
- les pires cas observés, avec une analyse des pire cas,
- la variabilité autour de la valeur moyenne par le calcul de l'écart-type.

Tout ceci pourrait être capturé dans un échantillon complet et post-traité ensuite via une bibliothèque de statistique (comme par exemple la bibliothèque Python statistics.py).

IV.6. **Qu'est-ce que l'élève aurait pu mesurer pour analyser, expérimentalement, le coût de son algorithme, en plus du nombre d'essais effectués ?**

Une solution possible : Dans l'esprit de ce problème, l'idée est de réduire à chaque itération la taille de l'ensemble des configurations possibles. Comme pour la recherche

dichotomique (où la taille de cet ensemble est divisée par 2) ou trichotomique (où la taille de cet ensemble est divisée par 3), le facteur de réduction (reliant les tailles de l'espace des configurations restant à explorer entre un essai et le suivant) indique la capacité discriminante de la réponse sous forme de pions bien placés et mal placés. L'élève aurait pu essayer de déterminer expérimentalement ce facteur de réduction, qui est variable selon le nombre de boules et de couleurs, mais également variable selon le coup joué.

IV.7. Vous aidez l'élève à réfléchir à la complexité du problème de recherche d'une configuration cachée. Pour cela, vous lui décrivez un algorithme qui trouve la configuration cachée en $N + \frac{P(P-1)}{2}$ essais pour N couleurs et P positions avec $N > P$. **Décrire cet algorithme. Quel est le nombre d'essais proposés par cet algorithme dans le cas du jeu pour jeunes joueurs et joueuses :** $N = 6$ **et** $P = 4$**? Et dans le cas classique** $N = 8$ **et** $P = 5$**?**

Une solution possible : Dans une première phase on teste chacune des couleurs (configuration où tous les pions sont de la même couleur). On détermine ainsi le nombre de pions de chaque couleur dans la configuration cachée. Comme on teste les N couleurs on a N essais.

Pour chaque pion dont on connait la couleur, on teste sa position en utilisant une couleur qui n'est pas dans la configuration cachée. Pour le premier pion on fait $P - 1$ tests, pour le 2e $P - 2$ tests, etc., soit $\frac{P(P-1)}{2}$ tests.

Pour $N = 6$ et $P = 4$ on obtient $6 + 6 = 12$ essais, pour $N = 8$ et $P = 5$ on obtient 18 essais

IV.8. **Que devrait en conclure l'élève sur le problème de recherche d'une configuration cachée ?** Justifier votre réponse.

Une solution possible : On dispose donc d'un algorithme qui est de coût – en nombre d'essais – polynomial et en nombre de couleurs et en nombre de positions. Donc l'algorithme est "raisonnable" en temps. Cependant, selon la méthode utilisée, par exemple dans la 1ère partie de ce projet où il faut générer toutes les configurations, l'espace mémoire utilisé est de taille exponentielle en le nombre de positions. (Autrement dit, l'algorithme implanté dans cette épreuve est de coût exponentiel en temps puisqu'il faut générer au préalable toutes les configurations.)

IV.9. **En vous inspirant de l'idée de la partie I., montrer que le problème de recherche pour** N **couleurs et** P **positions ne peut pas être résolu, pour le pire cas, en moins de** $\log_b N^P$ **où** b **est une base que l'on calculera.**

Indication : calculer le nombre de réponses différentes possibles, en termes de fiches noires et blanches, pour un essai.

Une solution possible : L'idée est de chercher à majorer le facteur de réduction de la taille de l'espace restant à explorer (le facteur qui vaut 2 pour la dichotomie et 3 pour la trichotomie).

Considérons le nombre de de réponses différentes qui peuvent être obtenues pour une configuration proposée : on a P positions et la réponse à une configuration candidate correspond à un nombre de pions noirs p_n et un nombre de pions blancs p_b avec évidemment

$p_n + p_b \leqslant P$. Le nombre de réponses possibles correspond donc au nombre de solutions de l'inéquation ci-dessus.

Ce nombre correspond à $\binom{P+2}{2}$ avec une configuration non observable (*i.e.* configuration qui ne peut pas avoir lieu) lorsque $p_n = P - 1$ et $p_b = 1$ ($P - 1$ noirs et 1 blanc) (toutes les autres configurations étant observables).

On note $d = \binom{P+2}{2} - 1$.

À chaque étape, l'algorithme peut diviser l'ensemble des configurations possibles en au plus d sous-ensembles de configurations. Donc en C étapes il obtiendra au plus d^C sous-ensembles de configurations.

Comme il doit trouver une configuration parmi N^P, il est nécessaire que le nombre d'étapes effectuées C vérifie $d^C \geqslant N^P$.

□

On peut également dire qu'une configuration est codée par la séquence des réponses fournies par l'algorithme. Donc en C étapes on ne pourrait coder au plus que d^C configurations différentes.

Partie V. Enjeu sociétal

Vous êtes sensible à la protection des données personnelles, sujet qui peut être abordé, entre autres, dans les thématiques Web et Réseaux sociaux du programme SNT. Vous avez l'intention de consacrer 1h30, en classe de Seconde, lors de l'enseignement de SNT, sur les questions d'anonymat et de ré-identification et plus généralement sur les risques liés aux données individuelles que l'on laisse sur le Web et en particulier sur les réseaux sociaux. Des extraits du programme de SNT sur ces thématiques sont donnés en annexe 6. Vous avez aussi trouvé un article, donné en annexe 6, qui traite du problème de la ré-identification et que vous allez exploiter pour votre cours.

V.1. **En quoi le problème de l'anonymat et de la ré-identification est-il similaire au principe du Mastermind, projet que vous avez travaillé avec vos élèves de Terminale ? Expliquer comment vous pourriez utiliser le Mastermind pour illustrer les principes d'anonymat et de ré-identification à vos élèves de Seconde.**

Une solution possible : On peut considérer la configuration cachée comme une configuration anonyme, et les propositions et questions posées par la personne qui devine, comme des questions et des croisements (entre les différentes configurations proposées et les informations recueillies sur chacune de ces configurations). L'objectif est donc de retrouver / ré-identifier la configuration cachée.

On pourrait faire jouer les élèves de Seconde (par binôme par exemple) et remplacer progressivement les couleurs par des attributs physiques pour commencer (couleur des cheveux, longueur, lunettes, couleur du pull ou T-shirt), voire des informations plus personnelles (initiales, ami·e·s, langues étudiées).

On pourrait aussi demander quelles informations personnelles leur paraissent sensibles et lesquelles les élèves seraient prêt·e·s à divulguer, pour déterminer si on peut les ré-identifier à partir de ces données. On peut aussi insister sur le petit nombre de données à

fournir et le faible effort à déployer pour la ré-identification.

V.2. **Donner 3 illustrations que vous pourriez présenter aux élèves afin qu'ils prennent conscience de toutes les informations personnelles qu'ils peuvent être amenés à donner et qui peuvent permettre leur ré-identification.**

Une solution possible : Voici 3 exemples possibles :

1. On peut citer un exemple sur des données médicales qui peuvent être collectées par les hôpitaux : même si des données telles que le nom, l'adresse et le numéro de sécurité sociale sont enlevées de la base, il est possible d'identifier le détenteur d'un dossier médical avec seulement des informations comme la ville, la date de naissance et le genre.
2. On peut utiliser l'histoire de Marc L. qui a laissé des traces sur les réseaux sociaux : toute la vie d'un inconnu choisi au hasard sur un réseau social et re-construite à partir des traces qu'il a laissées : Facebook, Flickr pour les photos, Youtube, et presse locale.
3. Enfin, on peut mentionner les données de géolocalisation (par exemple comme les données GPS des téléphones portables) avec lesquelles on peut retrouver l'adresse de son foyer et son identité.

V.3. **Citer 2 contextes où le partage de données anonymisées est d'intérêt public.**

Une solution possible : On peut citer les contextes suivantes :

- données sur la consommation des ménages (consommations domestiques d'énergie et d'eau notamment) : cela permet de savoir quelles mesures préconiser pour économiser des ressources ;
- données judiciaires et sur certaines décisions de justice : cela permet d'étudier l'impartialité des décisions de justice selon la couleur de peau, le quartier d'origine, etc.

V.4. **Citer 2 situations où la ré-identification est problématique.**

Une solution possible :

- des données médicales pourraient être utilisées par des compagnies d'assurance pour fixer le montant des cotisations ;
- la ré-identification du foyer et des heures d'occupation de ce foyer permettrait de savoir quand un logement est libre pour le cambrioler.

V.5. **Citer 3 mini-activités sur machine qui pourraient aider les élèves à comprendre quelles informations personnelles sont stockées et comment ils peuvent essayer de réguler les traces qu'ils laissent dans leurs usages quotidiens.**

Une solution possible :

- Effectuer une navigation / requête et visiter la première page proposée, regarder les propositions regardant les cookies. Apprendre à les effacer. Utiliser un outil qui indique combien de trackers tracent notre consultation de cette page.
- Éplucher les préférences du navigateur : sauvegarde de mots de passe, géolocalisation.
- Commencer à installer une nouvelle application sur le téléphone de l'enseignant (sans aller jusqu'au bout) et étudier les demandes d'accès : géolocalisation, contacts. Discuter de l'intérêt de ces demandes, avec les élèves, pour l'usage prévu.

V.6. **Donner le plan de cours d'1h30 que vous projetez de faire sur ce sujet. Vous décrirez les différentes notions que vous comptez aborder et les illustrations associées le cas échéant, ainsi que les activités que vous comptez réaliser avec les élèves et l'ordre dans lequel vous comptez aborder ces notions et ces activités.**
(Répondre en 20 lignes maximum.)

Une solution possible : Ce cours pourrait aborder les points suivants :

1. sensibiliser les élèves au fait que les données personnelles sont laissées à beaucoup d'endroits ;
2. sensibiliser les élèves au fait que l'anonymat présente aussi des dangers ;
3. discuter sur l'anonymat et le sentiment d'impunité sur les réseaux sociaux, sentiment qui n'est pas justifié ni moralement ni techniquement ;
4. avoir une discussion équilibrée sur l'intérêt de disposer de données personnelles anonymisées et le danger en abordant la question de la ré-identification ;
5. introduire une solution technique comme la confidentialité différentielle. , faire le jeu proposé sur la page Wikipédia `https://fr.wikipedia.org/wiki/Confidentialit%C3%A9_diff%C3%A9rentielle` pour l'illustrer.

Les deux premiers points pourraient être traités ensemble par groupes de 4 ou 5 suivis par une discussion collective rapide sur ces sujets (10 minutes). Le troisième point serait traité collectivement (5 minutes). Le quatrième point serait aussi abordé collectivement avec, entre autres, la lecture et une discussion sur l'article publié dans The Conversation et donné dans l'annexe 6 du sujet (30 minutes). Enfin la technique de confidentialité différentielle, proposée dans le cinquième point, est d'abord présentée collectivement avec un accent particulier sur l'introduction de l'aléa sur les données (10 minutes). Un jeu peut ensuite être lancé au sein de groupes d'une dizaine d'élèves pour illustrer la notion de confidentialité différentielle. La question posée aux élèves pourrait être "qui a déjà menti sur son âge : soit en se rajeunissant pour bénéficier d'un tarif réduit, soit en se vieillissant pour accéder à des services avancés". Les réponses seraient ensuite collectées, et en connaissant le nombre de réponses honnêtes, on peut le comparer avec celui qui peut être estimé avec la technique de confidentialité différentielle (30 minutes). Récapituler et conclure, collectivement mais en guidant fermement la discussion (5 minutes).

V.7. **L'évaluation des acquis sur ce cours se fera par QCM. Proposer deux questions, avec 4 choix chacune, qui testeront, pour l'une, les connaissances acquises et pour l'autre, le savoir-faire (comme le raisonnement et la mise en œuvre par exemple). Justifier le**

choix des questions. Pour chaque question, justifier le choix des 4 réponses proposées. Vous indiquerez aussi si les réponses sont justes ou fausses.

Une solution possible :

Dans le QCM suggéré ci-dessous, plusieurs réponses justes peuvent être proposées sur certaines questions.

Question 1 : Quelles sont les données personnelles parmi ? :

i) nom,
ii) photo,
iii) réseaux d'amis
iv) pays

Réponse : nom et photo sont des données personnelles car ils permettent d'identifier la personne associée à ces données. Réseaux d'amis et pays ne sont pas, en revanche, des données personnelles.

Question 2 : Est-il possible de ne pas communiquer certaines données dans les cas suivants ? :

i) son nom sur Facebook
ii) son numéro de carte bancaire lors d'un achat sur un site marchand
iii) l'accès à ses contacts lorsqu'on installe une application de gestion des tâches sur son téléphone
iv) la réception de cookies lorsqu'on visite un site Web

Réponses : On doit laisser son nom chez Facebook, cela fait partie des Conditions Générales d'Utilisation. On doit donner ses identifiants de paiement (carte bancaire ou PayPal par exemple) pour payer sur un site marchand. On peut décocher le fait de donner accès à ses contacts lorsqu'on installe certaines applications sur son téléphone. On peut refuser les cookies, ou les nettoyer en allant dans la rubrique "Préférences" de son navigateur.

Annexe 1 : Règles du Mastermind

Les règles du Mastermind présentées ci-dessous sont inspirées de Wikipedia (`https://fr.wikipedia.org/wiki/Mastermind` consulté le 24 octobre 2019). Le Mastermind, ou Master Mind, est un jeu de société pour deux joueurs dont le but est de trouver un code. C'est un jeu de réflexion, et de déduction, inventé par Mordecai Meirowitz dans les années 1970.

Le jeu d'origine se joue avec un joueur appelé codificateur et un joueur appelé décodeur.

Déroulement du jeu : le codificateur dispose de pions de N couleurs différentes. Il commence par placer P pions dont il a choisi les couleurs sans qu'ils soient vus de l'autre joueur à l'arrière d'un cache qui les masquera à la vue de celui-ci jusqu'à la fin de la partie. Il doit prendre soin de ne pas révéler la configuration qu'il a choisie, c'est-à-dire la couleur et la position des pions. Rien ne l'empêche de choisir plusieurs pions d'une même couleur dans une même configuration.

Le décodeur doit trouver quels sont les P pions choisis par le codificateur, c'est-à-dire leurs couleur et position. Le décodeur propose une configuration : il place P pions dans les trous de la première rangée la plus proche de lui, cf. Figure 2a.

Une fois les pions placés, le codificateur indique :

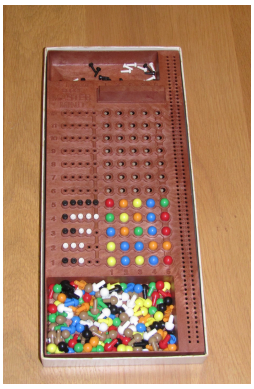
— le nombre de pions de la bonne couleur bien placés en utilisant le même nombre de fiches noires à côté de la rangée proposée par le décodeur ;

— puis le nombre de pions de la bonne couleur, mais mal placés, en insérant le même nombre de fiches blanches à côté des fiches noires.

S'il n'y a aucune correspondance, le codificateur ne met aucune fiche.

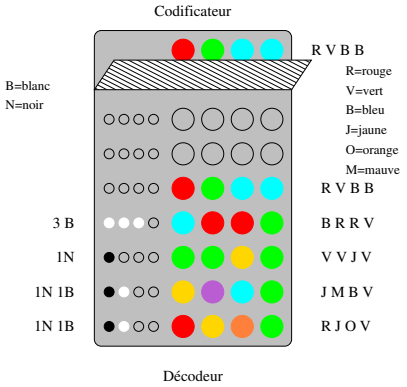
Une fois que le codificateur a indiqué le nombre de fiches noires et le nombre de fiches blanches associés à la configuration proposée par le décodeur, ce dernier propose une nouvelle configuration. Le processus se répète jusqu'à ce que le décodeur ait trouvé la bonne configuration ou jusqu'à ce qu'il ait proposé C configurations qui ne correspondent pas à la configuration choisie par le codificateur.

La figure 2 donne des exemples de déroulement de parties de Mastermind. Le Mastermind classique fonctionne avec 8 couleurs et 5 positions et la configuration doit être trouvée en au plus 12 coups.



(a) Vue du jeu côté décodeur

(b) Vue du jeu côté codificateur



(c) Schéma du jeu

FIGURE 2 – Exemples de Mastermind.

Annexe 2 : Bibliothèque random

Bibliothèque random de Python

Voici quelques explications pour utiliser la fonction `randint` de la bibliothèque `random` de Python.

Tirer aléatoirement un nombre entier

Pour tirer aléatoirement un nombre entier selon une loi uniforme, on peut utiliser la fonction `randint`. Cette fonction prend deux paramètres en entrée : une valeur minimale et une valeur maximale.
Voici un exemple pour tirer des entiers entre 1 et 5. Le premier paramètre doit être inférieur au second.

```
import random
print random.randint(1, 5)
```

Le résultat sera soit 1, soit 2, soit 3, soit 4 ou 5.

Annexe 3 : Table de calculs

La table suivante donne la valeur de x^y pour différentes valeurs de x et de y.

	$y=2$	$y=3$	$y=4$	$y=5$	$y=6$	$y=7$	$y=8$	$y=9$
$x=2$	4	8	16	32	64	128	256	512
$x=3$	9	27	81	243	729	2187	6561	19683
$x=4$	16	64	256	1024	4096	16384	65536	262144
$x=5$	25	125	625	3125	15625	78125	390625	1953125
$x=6$	36	216	1296	7776	46656	279936	1679616	10077696
$x=7$	49	343	2401	16807	117649	823543	5764801	40353607
$x=8$	64	512	4096	32768	262144	2097152	16777216	134217728

Annexe 4 : Production d'élèves

Production d'élèves – partie II., question II.3

```
def verification():
    # definit le nombre de couleurs justes
    # et de couleurs qui ne sont pas dans la combinaison a trouver
    bon=0
    mauvais=0
    tour=tour+1
    if a==aa:
        bon=bon+1
    if aa!=a and aa!=b and aa!=c and aa!=d and aa!=e:
        mauvais=mauvais+1
    if b==bb:
        bon=bon+1
     if bb!=b and bb!=a and bb!=c and bb!=d and bb!=e:
         mauvais=mauvais+1
    if c==cc:
        bon=bon+1
    if cc!=c and cc!=b and cc!=a and cc!=d and cc!=e:
        mauvais=mauvais+1
    if d==dd:
        bon=bon+1
    if dd!=d and dd!=b and dd!=c and dd!=a and dd!=e:
        mauvais=mauvais+1
    if e==ee:
        bon=bon+1
    if ee!=e and ee!=b and ee!=c and ee!=d and ee!=a:
        mauvais=mauvais+1
    if a==aa and b==bb and c==cc and d==dd and e==ee:
        # Si la combinaison est la méme que celle é trouver,
        # ouvre une fenétre de victoire
        fen_gagne=Toplevel()
        fen_gagne.title("Mastermind")
        text=Label(fen_gagne,text="Vous avez gagné !
                                   Félicitations !")
        text.pack()
        global fen_gagne
    if tour==10 :
        # Si au 10eme tour, la combinaison n'a pas encore ete
        #                                   trouvée, défaite
        if aa!=a or bb!=b or cc!=c or dd!=d or ee!=e :
                fen_perdu=Toplevel()
                fen_perdu.title("Mastermind")
                text=Label(fen_perdu,text=
                 "C est la 10e tentative sans succés,
                                   vous avez donc perdu la partie")
                text.pack()
                global fen_perdu
    bon=str(bon)
    mauvais=str(mauvais)
    global tour,bon,mauvais
```

Annexe 5 : Analyse d'un projet d'élève sur le Mastermind

Analyse d'un projet d'élève sur le Mastermind – partie IV.

Extraits d'un projet d'élève sur le Mastermind (accessible à `https://issuu.com/wile/docs/mastermind`).

Note : dans cette production, la « version classique » correspond à la version pour les plus jeunes joueurs du Mastermind, avec 4 positions et 6 couleurs et une boule correspond à un pion.

4. Analyse des résultats

4.1. Le nombre de coups pour gagner

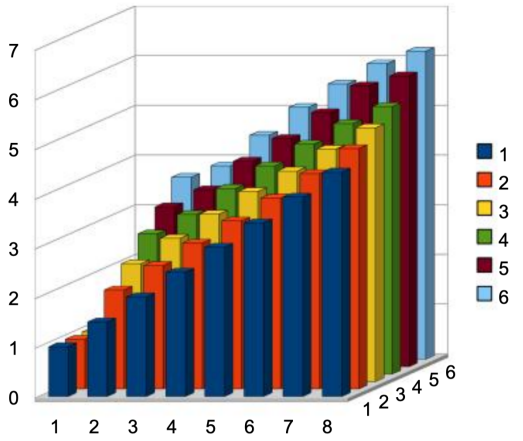
Grâce aux nombreuses simulations, j'ai pu découvrir le nombre de coups qu'il faut en moyenne pour découvrir la combinaison cachée en fonction du nombre de couleurs et du nombre de boules. J'ai effectué de 100 à 10'000 simulations par cas, cela dépend de la complexité de l'opération. Ainsi, plus le nombre de boules et le nombre de couleurs sont élevés, plus le nombre de simulations est bas, car sinon, cela prendrait trop de temps. Les valeurs sont donc moins précises pour ces cas.

Moyenne du nombre de coups en fonction du nombre de boules et de couleurs

	1 boule	2 boules	3 boules	4 boules	5 boules	6 boules
1 couleur	1.000	1.000	1.000	1.000	1.000	1.000
2 couleurs	1.503	1.996	2.373	2.822	3.216	3.663
3 couleurs	2.006	2.485	2.884	3.214	3.562	3.893
4 couleurs	2.499	2.938	3.371	3.740	4.135	4.507
5 couleurs	3.008	3.387	3.825	4.192	4.596	5.070
6 couleurs	3.495	3.844	4.238	4.631	5.117	5.540
7 couleurs	4.014	4.338	4.680	5.043	5.650	5.950
8 couleurs	4.513	4.841	5.109	5.388	5.850	6.200

Rouge : Mastermind classique

Graphique correspondant au tableau ci-dessus



axe Z (verticalement) : nombre de coups
axe Y (en profondeur) : nombre de boules
axe X (horizontalement de gauche à droite) : nombre de couleurs

Annexe 6

Documents pour la partie V.

Cette annexe comprend un extrait du programme de SNT en Seconde concernant les thématiques du Web et des réseaux sociaux, ainsi qu'un article tiré du site The Conversation.

Extrait du programme de SNT en Seconde concernant la thématique du Web.

Impacts sur les pratiques humaines

Dans l'histoire de la communication, le Web est une révolution : il a ouvert à tous la possibilité et le droit de publier ; il permet une coopération d'une nature nouvelle entre individus et entre organisations : commerce en ligne, création et distribution de logiciels libres multi-auteurs, création d'encyclopédies mises à jour en permanence, etc. ; il devient universel pour communiquer avec les objets connectés.

Le Web permet aussi de diffuser toutes sortes d'informations dont ni la qualité, ni la pertinence, ni la véracité ne sont garanties et dont la vérification des sources n'est pas toujours facile. Il conserve des informations, parfois personnelles, accessibles partout sur de longues durées sans qu'il soit facile de les effacer, ce qui pose la question du droit à l'oubli. Il permet une exploitation de ses données, dont les conséquences sociétales sont encore difficiles à estimer : recommandation à des fins commerciales, bulles informationnelles, etc. En particulier, des moteurs de recherche permettent à certains sites d'acquérir de la visibilité sur la première page des résultats de recherche en achetant de la publicité qui apparaîtra parmi les liens promotionnels.

Extrait du programme de SNT en Seconde concernant la thématique des réseaux sociaux.

Les données et l'information

Les différents réseaux sociaux permettent l'échange d'informations de natures différentes : textes, photos, vidéos. Certains limitent strictement la taille des informations, d'autres autorisent la publication, mais de façon limitée dans le temps. Certains permettent l'adjonction d'applications tierces (plug-ins) qui peuvent ajouter des fonctionnalités supplémentaires.

Toutes les applications de réseautage social utilisent d'importantes bases de données qui gèrent leurs utilisateurs, l'ensemble des données qu'ils partagent, mais aussi celles qu'ils consentent à fournir (sans toujours le savoir), y compris sur leur vie personnelle.

Article tiré du site The Conversation

Seules 3 pages de cet article sont données (la dernière page n'a pas été reproduite, elle contenait des suggestions de lecture et ne concernait pas le sujet que vous devez traiter).

Cet article comporte une erreur de frappe : il faut lire **font** à la place de **sont** dans la phrase *En réponse, les compagnies et organismes qui les collectent affirment souvent qu'elles le sont de manière « anonyme ».*

THE CONVERSATION
L'expertise universitaire, l'exigence journalistique

Dans le métro. Photo by Martin Adams on Unsplash

Données anonymes… bien trop faciles à identifier

17 septembre 2019, 21:01 CEST

Auteur

Luc Rocher
Doctorant, ingénierie mathématique, Université catholique de Louvain

Téléphones, ordinateurs, cartes de crédit, dossiers médicaux, montres connectées, ou encore assistants virtuels : chaque instant de nos vies – en ligne et hors ligne – produit des données personnelles, collectées et partagées à grande échelle. Nos comportements, nos modes de vie, s'y lisent facilement. Mais faut-il s'en inquiéter ? Après tout, ces données qui nous révèlent sont souvent anonymisées par les organismes qui les collectent. C'est du moins ce que l'on peut lire sur leurs sites. Leur travail est-il efficace ? Et les données anonymes le sont-elles vraiment ? Dans notre dernier article publié dans la revue *Nature Communications*, nous développons une méthode mathématique qui montre que c'est loin d'être acquis. Elle a pu nous amener à réidentifier des individus parmi des bases de données anonymes et fortement échantillonnées, remettant en question les outils utilisés actuellement pour partager les données personnelles à travers le monde.

Matière première

D'abord, quelques ordres de grandeur. Ces dix dernières années, nos données personnelles ont été collectées à une vitesse inégalée : 90 % de celles circulant sur Internet ont été créées il y a moins de deux ans ! Objets connectés, informations médicales ou financières, réseaux sociaux, ces données sont

la matière première de l'économie numérique comme de la recherche scientifique moderne. Mais, très vite, on a vu apparaître certaines dérives. Notamment les atteintes à la vie privée qui se sont multipliées. Témoin, parmi de nombreuses affaires, le scandale Cambridge Analytica… Depuis, 80 % des Européen·ne·s estiment avoir perdu le contrôle sur leurs données.

Traitement d'échantillons viraux. Les données personnelles de santé sont parmi les plus sensibles. James Gathany/CDC

En réponse, les compagnies et organismes qui les collectent affirment souvent qu'elles le sont de manière « anonyme ». Par exemple, la société Transport for London (TfL), en charge du métro londonien, a entrepris de surveiller les déplacements des passagers sur le réseau via les signaux wifi « anonymes » de leurs téléphones portables. En Belgique, plus de 15 hôpitaux revendent les données confidentielles de leurs patients à une multinationale, Quintiles IMS, sous couvert d'anonymat. Enfin, en France, Orange et SFR ont revendu des données de géolocalisation en temps réel ou en différé, données là encore « anonymisées ».

Point intéressant, une donnée anonyme n'est plus considérée comme donnée personnelle. Elle échappe donc aux régimes de protection comme le RGPD en Europe. Partager des données personnelles anonymisées ne nécessite donc plus le consentement des participant·e·s… Puisqu'ils et elles sont anonymes !

Ré-identification

Or, des chercheur·e·s et journalistes ont depuis longtemps montré que certaines données anonymes peuvent être ré-identifiées. Dans les années 1990, Latanya Sweeney avait pu ré-identifier les données médicales de William Weld (alors gouverneur du Massachusetts), sur base de son code postal, sa date de naissance et son genre. Deux journalistes allemands ont récemment ré-identifié l'historique de navigation d'un juge et d'un député, retrouvant leurs préférences sexuelles et leurs traitements médicaux dans des données anonymes obtenues en se faisant passer pour des acheteurs potentiels. Et, aux États-Unis, les dossiers fiscaux du président américain Trump ont pu lui être ré-attribués par le *New York Times* en utilisant des données anonymes publiées par le fisc américain, l'IRS.

Compagnies et gouvernements minimisent souvent ces ré-identifications. Leur ligne de défense : parmi des petites bases de données, toujours incomplètes, personne ne saura jamais si une ré-identification est correcte ou non et si des chercheur·e·s ou journalistes ont vraiment réidentifié la bonne personne.

Cela implique que l'organisme collecteur fasse un travail dit d'*échantillonage* sur la base de données. Ainsi, l'autorité de protection des données australienne [OAIC], suggère dans son guide de dés-identification que l'échantillonnage augmente « l'incertitude qu'une personne particulière fasse réellement partie d'une base de données anonyme ». Prenons un exemple pour expliquer cela. Admettons que votre employeur retrouve des données vous correspondant dans un échantillon de 10 000 patients, soit 1 % d'une large base de données médicales. Ces données – comprenant par exemple votre lieu et date de naissance, genre, statut marital, etc. – pourraient bien appartenir à une autre personne qui partage ces caractéristiques. Car cette base de données de 10 000 personnes ne représente que 0,015 % de la population française. Et ces données réidentifiées pourraient

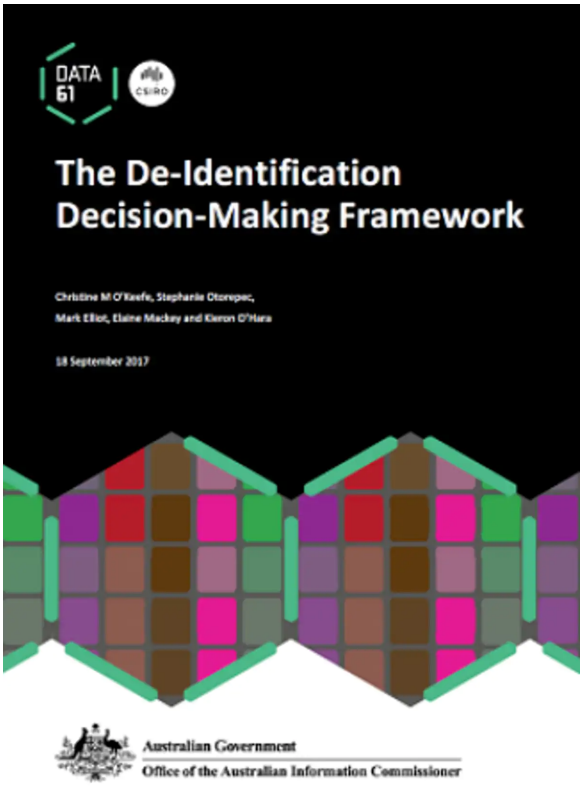
correspondre à n'importe quelle autre personne parmi les 99,985 % autres Français·e·s.

Échantillonner (partager par exemple 1 % d'une base de données) est ainsi une technique largement utilisée. Réduire la taille des données partagées permet de justifier que ces données sont anonymes, car personne ne pourra jamais prouver qu'une ré-identification est correcte.

Un algorithme qui remet en question l'anonymat

Le problème ? Nos travaux démontrent au contraire qu'un algorithme peut apprendre à estimer, avec grande précision, si des données réidentifiées appartiennent bien à la bonne personne ou non.

Il y a bien entendu, si c'est en France, de nombreux hommes trentenaires, habitant à Paris. Si je retrouve un seul homme de 30 ans parmi les données anonymes de 1 000 personnes, collectées et revendues par un cabinet d'assurance parisien, il y a peu de chance qu'elles correspondent à mon voisin Émeric. Les données correspondant à ces trois attributs (homme, 30 ans, habitant à Paris) seront sans doute celles d'un autre Français.



Un guide pour protéger les données en Australie. Australian Government, CC BY

Mais au fur et à mesure que ces données s'enrichissent, qu'on apprend davantage de caractéristiques, il devient illusoire qu'une seconde personne ait les mêmes caractéristiques. Il y a ainsi sans doute un seul homme à Paris, né le 5 janvier 1989, roulant en vélo électrique et habitant avec ses deux enfants (deux filles) et un berger allemand : mon voisin Émeric.

Après avoir « appris » quelles caractéristiques rendent les individus uniques, notre algorithme génère des populations synthétiques pour estimer si un individu peut se démarquer parmi des milliards de personnes. Le modèle développé permettrait par exemple aux journalistes du New York Times de savoir à coup sûr si les dossiers identifiés appartenaient vraiment à Donald Trump.

Nos résultats montrent que 99,98 % des Américains seraient correctement ré-identifiés dans n'importe quelle base de données en utilisant 15 attributs démographiques. Les chiffres sont similaires à travers le monde (16 attributs en ajoutant la nationalité). Une quinzaine de caractéristiques qui suffisent à identifier un individu, ce n'est hélas pas beaucoup. Le « data broker » Acxiom, un courtier de données qui achète et qui revend nos données personnelles dans 60 pays, possède par exemple jusqu'à 5,000 attributs par personne.

Nos travaux remettent ainsi en question les pratiques actuelles utilisées pour dés-identifier des données personnelles. Cela interroge sur les limites de l'anonymisation : utiliser ainsi ces données protège-t-il toujours notre vie privée ? Alors que les standards d'anonymisation sont en passe d'être redéfinis par les pouvoirs publics, au niveau national et au sein de l'Union européenne, il est crucial pour ces standards d'être rigoureux, de promouvoir de meilleures méthodes de partage des données, et de prendre en compte tout risque futur. C'est à la fois important pour nos vies privées, pour la croissance de l'économie numérique et pour le dynamisme de la recherche scientifique.