Le *mot* le plus court

Année 2022-2023

Juliette Boutin-Cadane, Apolline Gerland (Terminale générale)

Établissement : Lycée Paul Guérin, Niort (79)
Encadrées par : Fabien Aoustin, Thomas Forget
Chercheur : Abdallah El Hamidi, chercheur université de La Rochelle

Dans cet article, nous avons tenté de déterminer le « mot » le plus court contenant l'ensemble des toutes les permutations d'un certain nombre de lettres. L'article se résume principalement en un algorithme général permettant de simuler de tels « mots ».

Présentation du problème

Notre problématique consiste à choisir un nombre *n* de lettres, et déterminer la taille du « mot » le plus court contenant l'ensemble de toutes les permutations possibles de ces lettres. (1)

Si l'on choisit $n = 1$ lettre, qui serait par exemple *A,* alors le « mot » le plus court est composé de cette seule lettre.

Si l'on choisit $n = 2$ lettres, qui seraient par exemple *A* et *B,* alors les permutations possibles sont *AB* et *BA*. Et un « mot » le plus court qui les contient est composé de 3 lettres : Il s'agit de *ABA* ou de *BAB*, qui contiennent chacun les deux permutations.

Si l'on choisit $n = 3$ lettres, qui seraient par exemple *A*, *B* et *C,* alors il y a $3! = 6$ permutations possibles : *ABC*, *ACB*, *BAC*, *BCA*, *CAB* et *CBA*. (2)

Et on essaye de construire pas à pas un « mot » le plus court possible les contenant toutes :

On commence par *ABC* , que l'on complète en *ABCA*, ce qui fait déjà apparaître deux permutations (*ABC* et *BCA*), puis au « mot » *ABCAB*, qui contient déjà 3 des 6 permutations cherchées.

On remarque alors que l'on ne peut pas poursuivre ainsi, car la permutation suivante qui apparaîtrait serait *ABC*, qui a déjà été utilisée.

On décide alors de prendre l'unique permutation non encore choisie et qui commence par *B* (la dernière lettre de *ABCAB*). Nous arrivons alors au « mot » *ABCABAC*, qui se fait ensuite compléter naturellement pour donner le « mot » de 9 lettres suivant : *ABCABACBA*.

Nous avons donc trouvé un « mot » de 9 lettres comportant l'ensemble des 6 permutations.

On remarque de plus qu'un « mot » comportant les 6 permutations comporte au moins 8 lettres (les 3 lettres de la permutation initiale utilisée, puis les 5 lettres qui seraient nécessaires pour avoir les 5 autres permutations. (3) Et qu'il n'est pas possible de trouver un « mot » de 8 lettres car, comme l'exemple précédent l'illustre, nous sommes obligées de faire à au moins une reprise un saut d'une lettre dans le « mot ».
Nous en concluons que le « mot » le plus court qui comporte les 6 permutations possibles de trois lettres est composé de 9 lettres.

Nous nous sommes alors intéressées aux « mots » comportant l'ensemble des permutations possibles de $n = 4$ lettres, et nous avons cherché le « mot » le plus court qui les contient toutes.

Nous remarquons tout d'abord qu'il y a $4! = 24$ permutations possibles de 4 lettres :

ABCD	*BACD*	*CABD*	*DABC*
ABDC	*BADC*	*CADB*	*DACB*
ACBD	*BCAD*	*CBAD*	*DBAC*
ACDB	*BCDA*	*CBDA*	*DBCA*
ADBC	*BDAC*	*CDAB*	*DCAB*
ADCB	*BDCA*	*CDBA*	*DCBA*

Commençons par décrire ce que donne la méthode pas à pas :

On débute assez naturellement pas *ABCDABC* (qui contient 4 permutations), mais à partir de là, on ne peut pas mettre la lettre *D*, qui conduirait à la permutation *ABCD* déjà utilisée.

Trois stratégies sont possibles, et nous ne pouvons pas être certaines par avance que l'une est meilleure que l'autre.

(1) On part des deux dernières lettres, et on utilise l'autre permutation débutant par *BC*, ce qui conduirait au « mot » *ABCDABCAD*, qui serait à poursuivre.

(2) On part de la dernière lettre en reprenant une permutation qui n'a pas encore été utilisée, ce qui donnerait par exemple *ABCDABCABD* ou *ABCDABCDBA*, qui seraient à poursuivre.

(3) On ignore totalement le « mot » composé initialement et on repart d'une nouvelle permutation.

Cette dernière stratégie conduit, par exemple, à créer un « mot » en 6 parties, dont chacune débute par une des permutations commençant par la lettre *A* :

*ABC*DABC *ABD*CABD *ACB*DACB *ACD*BACD *ADB*CADB *ADC*BADC

Ce « mot » de 36 lettres n'est très certainement pas le plus court possible, mais il contient les 24 permutations possibles des quatre lettres.

On en déduit ainsi qu'un premier majorant au nombre de lettres du « mot » le plus court est 36.

Pour revenir sur ce « mot » le plus court, il est plus compliqué d'évaluer le meilleur choix entre les stratégies (1) ou (2), et de leurs différentes implications.

Mais après beaucoup de tâtonnements pour tenter de trouver plus court, nous avons trouvé des mots de 35 lettres puis finalement nous sommes arrivées au « mot » de 33 lettres suivant :

ABCDABCADBCABDCABACDBACBDACBADCBA

Sans avoir la garantie qu'il s'agissait du « mot » le plus court possible, le temps passé à se casser la tête dessus n'étant malheureusement pas un argument suffisant…

Étant donnée la difficulté d'obtenir le « mot » le plus court utilisant 4 lettres, nous avons alors programmé l'algorithme suivant, qui propose des mots, pour un certain nombre de lettres données. La stratégie de l'algorithme reprenant celle présentée ci-dessus.

Plutôt que les lettres *A*, *B*, *C*, …, cet algorithme utilise des chiffres 1, 2, 3, ...

Il affiche tout d'abord toutes les permutations cherchées, selon le nombre de lettres demandé, puis propose un « mot » qui les contient toutes, et finit en donnant le nombre de lettres de ce « mot ».

Notre programme a des commandes aléatoires (lors des moments où il faudrait faire des choix), qui permettent d'afficher des mots différents à chaque lancement de l'algorithme. (4)

```
import random                              (# : notes d'édition)
def solutions(x):
    """Toutes les combinaisons possibles des chiffres de 1 à x"""
    l=[]                  # l sera la liste contenant les permutations
    t=0
    k=1
    t={}
    for i in range(0,x): # cette boucle sert à calculer la factorielle de x
        k=k*(x-i)
    while len(l) < k :
        p=[]
        b=[]
        q=0
        h={}
        for y in range (1,x+1):
            p.append(y)
            random.shuffle(p)
        for r in p:
            b.append(r)     # cette boucle sert juste à dire : b=p
        b.reverse()
            s=0
        for d in range(1,x+1):
            s+=(p[d-1])*(10**(d-1))    # cette boucle sert à transformer une
                                       suite de chiffres en nombre
        if s not in l:
            l.append(s)
            t[str(s)]=b
    return(t)

def clé(x):
    """Nombres comportant toutes les solutions(x)"""
    dict=solutions(x)  # cette fonction retourne une liste avec 3éléments :
                       la liste des permutations apparaissant dans (J), le
                       mot solution (y), la longueur du mot solution
    l=[]
    J=[]
    R=[]
    z=[]
    a=[]
    y=[]
    t=0
    dict1 = dict.copy()
    d=1
```

```
for i,r in dict.items():
        l.append(i)
        if i==l[0]:           #ce cas n'apparait que lorsque la liste (l) ne
                              contient qu'un élément
            R = r[:]
            J.append(i)
            y =r[:]
            dict1.pop(i)
        else:
            while len(dict1)>0:
                for g,t in dict.items():
                    z=[]
                    a=[]
                    for f in range(len(i)-d):  #pour définir z(dernier
        chiffre) et a(premier chiffre)
                        z.append(str(R[f+d]))
                        a.append(str(t[f]))
                    if str(g) not in dict1.keys():    #si la solution est
                                                      déjà utilisée
                        if g==J[-1] and g!=J[0]:    #si la solution est
                                    la dernière juste mise, prendre un
                                    chiffre de moins pour z et a
                            d+=1
                    else:
                            pass
                    elif z == a:  #si les derniers chiffres z = les
                                premiers chiffres de a, alors rajouter le
                                nombre correspondant à a dans les clés
                        J.append(g)
                        R=[]
                        R = t[:]
                     for d1 in range(-(d),0): #cette boucle prolonge le
                                    mot solution y en lui ajoutant le
                                    suffixe de t de longueur d
                        y.append(R[x+d1])
                    dict1.pop(g)
                    d=1
    return(J,y,len(y))

def verification(x):
    d=solutions(x)
    j,y,t=clé(x)
    v=0
    p=False
    for i in d.keys():
        if i in j:
            v+=1
    if v == len(d):
        p=True
    return(p)

def moyenne(d,x):
    """moyenne de chiffre dans la clé"""
    v=0
    for p in range(0,d):
        i,o,k=clé(x)
        v+=k
    return(v/d)

print(clé(3))
```

Lorsqu'on le teste pour 3 lettres distinctes, on ne trouve que des « mots » de 9 lettres.

Lorsqu'on le teste pour 4 lettres distinctes, on trouve des « mots » faisant entre 33 et 35 lettres, ce qui nous a conforté dans l'idée que le « mot » le plus court ferait bien 33 lettres.

Lorsqu'on le teste pour 5 lettres distinctes, on trouve des « mots » ayant entre 153 (rarement) et 166 lettres. Ce qui amène à conjecturer qu'un « mot » le plus court serait composé de 153 lettres.

Mais aussi qu'il faudrait trouver une stratégie plus claire pour construire les « mots », en prenant aussi en considération d'autres possibilités stratégiques dues au nombre plus important de lettres. (5)

Nous avons aussi finalement constaté un lien entre ces différentes valeurs :

Pour $n = 1$, on trouve $1 = 1!$.

Pour n =2, on trouve $3 = 2! + 1!$.

Pour $n = 3$, on trouve $9 = 3! + 2! + 1!$.

Pour $n = 4$, on conjecture que l'on trouve $33 = 4! + 3! + 2! + 1!$.

Pour $n = 5$, l'algorithme nous amène à conjecturer que l'on trouve $153 = 5! + 4! + 3! + 2! + 1!$.

Pour $n = 6$, cette logique devrait amener à trouver un « mot » de 873 lettres, mais l'algorithme ne semble donner que des « mots » comprenant entre 914 et 938 lettres, ce qui montre probablement ses limites, dues exclusivement à la nécessité d'une meilleure stratégie de création de « mot ». (6)

Des recherches sur Internet nous ont conduit à trouver que ces « mots les plus courts » que l'on cherche sont appelés *super-permutations*, et que la suite définie par $u_n = \sum_{k=1}^{n} k!$ est la suite référencée A007489 sur le site de l'OEIS. (7)

Et malheureusement, cette valeur u_n devient un majorant strict à partir de $n = 6$ lettres. (8)

La question de la valeur exacte du nombre de lettres reste encore aujourd'hui un problème non résolu pour une valeur quelconque de n.

(Voir par exemple l'article « *Le secret d'Arsène Lupin : Les super-permutations* » de Jean-Paul Delahaye, publié dans le magazine Pour la science de Juillet 2020.)

Notes d'édition

(1) Quelques précisions : Dans le langage courant comme dans les mathématiques, un mot est une suite de lettres. La seule distinction est que, du point de vue mathématique, toute suite de lettres est un mot (sans qu'elle n'ait de sens sémantique).
Dans cet article, on considère les permutations des lettres de l'alphabet. Si cet alphabet possède n lettres, une permutation est donc un mot de longueur n contenant exactement une fois chaque lettre de l'alphabet.

(2) Sur un alphabet de 3 lettres, il y a 3 choix possibles pour la première lettre de la permutation, puis 2 choix pour la seconde et un seul choix pour la dernière. Cela donne donc $3 \times 2 \times 1 = 6$ possibilités au total. Ce nombre n'est autre que la factorielle de 3.

(3) Il reste à démontrer formellement qu'un mot de longueur 8 ne contient jamais toutes les permutations de 3 lettres.

(4) L'algorithme présenté utilise le langage Python. Il peut être facilement testé en ligne. Voir par exemple : https://www.programiz.com/python-programming/online-compiler/

(5) Vu le caractère aléatoire de l'algorithme, il faut comprendre que 166 n'est pas la borne maximale des mots qui peuvent être produits. Il s'agit juste du maximum constaté par les autrices lors de leurs tests. À titre d'information, un autre test a produit un mot de longueur 170.
De plus, pour mesurer le caractère rare de la solution minimale, on peut compter le nombre de fois que la solution 153 apparaît dans un grand échantillon de tests.

(6) Comme dans la note précédente, insistons sur le fait que 914 n'est pas la longueur minimale des mots produits par l'algorithme. Il serait intéressant de vérifier si cette façon de générer des mots permet d'obtenir un mot solution de longueur 872 (meilleure borne connue actuellement).

(7) Pour décrypter "OEIS" voir le lien : https://oeis.org/A007489 pour la suite citée.

(8) Pour un alphabet de taille n, la longueur d'un mot de le plus court est plus petite que u_n. On pourra se reporter par exemple à l'article de Jean-Paul Delahaye, accessible en ligne sur : https://www.pourlascience.fr/sd/mathematiques/le-secret-d-arsene-lupin-les-superpermutations-19655.php.