R1.06 TP 4

Suites aliquotes

3. Recherche de nombres sociables

13 novembre 2025 – B. Colombel

Sommaire

1 Décomposition en facteurs premiers **1**

2 Un peu de mathématiques **2**

3 Recherche systématique de nombres sociables **4**

Rappels : un nombre sociable est un entier positif répondant à la conjecture de Catalan, c'est-à-dire, dont la suite aliquote se termine par un 1 ou forme un cycle à partir d'un certain rang.

Actuellement, il n'est pas prouvé que tous les nombres entiers positifs sont des nombres sociaux. Par exemple, personne ne sait si la suite aliquote de 276 se termine ou forme un cycle et personne n'a prouvé non plus que ce n'était pas possible !

Pour étudier de plus près les nombres sociaux, nous allons encore améliorer l'efficacité de nos méthodes, toujours en utilisant les nombres premiers.

1 Décomposition en facteurs premiers

Théorème 1
Soit n un entier naturel supérieur où égal à 2. Alors :

1. n se décompose en un produits de facteurs premiers.
2. Cette décomposition est unique à l'ordre des facteurs près.

Par exemple,

$$6\,468 = 2^2 \times 3 \times 7^2 \times 11$$

Nous pourrons utiliser la fonction `factor(n)` :

[1]:
```
factor(6468)
```

[1]:
```
2^2 * 3 * 7^2 * 11
```

L'affichage du résultat est trompeur. La fonction 'factor(n)' retourne en réalité une liste de *tuples*[1] de longeur 2 :

[2]:
```
F = factor(n)
F[0], F[1], F[2], F[3]
```

[2]:
```
((2, 2), (3, 1), (7, 2), (11, 1))
```

Dans chaque couple, le premier élément est le facteur premier et le deuxième l'exposant correspondant. On accède à ces informations de la manière suivante :

1. un tuple est une sorte de liste mais qui est, contrairement à elles, immuable, c'est-à-dire,qu'un tuple n'est pas modifiable.

[3]:
```
print(F[1][0]) # affichage du 2e facteur premier
print(F[1][1]) # affichage de la puissance correspondante
```

[3]:
```
3
1
```

Remarque. La décomposition de grands nombres n'est pas immédiate. Essayons, par exemple, avec un nombre supérieur à 10 millions :

[4]:
```
%time factor(10^43-1)
```

[4]:
```
CPU times: user 15.3 ms, sys: 0 ns, total: 15.3 ms
Wall time: 18.5 ms
3^2 * 173 * 1527791 * 1963506722254397 * 2140992015395526641
```

2 Un peu de mathématiques

Lien entre diviseurs et facteurs premiers La décomposition en facteurs premiers d'un entier n permet de déterminer les diviseurs de n. Nous obtiendrons les diviseurs de n en considérant toutes multiplications de ces facteurs premiers entre eux.

Prenons comme exemple $n = 24 = 2^3 \times 3$.

[5]:
```
factor(24)
```

[5]:
```
2^3 * 3
```

Nous obtenons tous les diviseurs de 24 en considérant toutes les multiplications de tous ces facteurs entre eux (en utilisant de 0 à 3 fois le chiffre 2 et de 0 à 1 fois le chiffre 3).

avec 4 facteurs	$2 \times 2 \times 2 \times 3 = 24$
avec 3 facteurs	$2 \times 2 \times 3 = 12$ $2 \times 2 \times 2 = 8$
avec 2 facteurs	$2 \times 3 = 6$ $2 \times 2 = 4$
avec 1 seul facteur	3 2
sans aucun facteur	1

Nous retrouvons bien l'ensemble des diviseurs de 24 : $\{1; 2; 3; 4; 6; 8; 12; 24\}$.

Revenons aux suites aliquotes Déterminer le terme suivant d'un nombre dans la suite aliquote revient à calculer la somme de ses diviseurs propres. Il existe un moyen rapide de calculer la somme de ces diviseurs.

— au facteur premier 2, qui apparait 3 fois dans la décomposition de 24, on fait correspondre le nombre :

$$\left(2^0 + 2^1 + 2^2 + 2^3\right)$$

— au facteur premier 3 qui apparait 1 seule fois dans la décomposition de 24, on associe le nombre :

$$\left(3^0 + 3^1\right)$$

Puis, nous multiplions les nombres ainsi obtenus :

$$s = \left(2^0 + 2^1 + 2^2 + 2^3\right) \times \left(3^0 + 3^1\right)$$

Le nombre ainsi construit vaut exactement la somme de tous les diviseurs de 24. En effet :

$$
\begin{aligned}
s &= (2^0 \times 3^0) + (2^0 \times 3^1) + (2^1 \times 3^0) + (2^1 \times 3^1) \\
&\quad + (2^2 \times 3^0) + (2^2 \times 3^1) + (2^3 \times 3^0) + (2^3 \times 3^1) \\
s &= 1 + 3 + 2 + 6 + 4 + 12 + 8 + 24
\end{aligned}
$$

Pour obtenir la somme des diviseurs stricts de 24, il suffit de retrancher 24 à s.

On peut encore faire mieux. les sommes du type

$$1 + p + p^2 + p^3 + \cdots + p^e$$

peuvent se calculer rapidement. En effet, il s'agit de la somme des $e + 1$ premiers termes de la suite géométrique de premier terme 1 et de raison p. On a donc :

$$1 + p + p^2 + p^3 + \cdots + p^e = \frac{p^{e+1} - 1}{p - 1}$$

Finalement, si un nombre n a la décomposition en k facteurs premiers :

$$n = \prod_{i=1}^{k} p_i^{e_i}$$

où les p_i sont les facteurs premiers, les e_i leurs exposants et $\prod$ le symbole du produit, alors la somme de ses diviseurs est égale à :

$$\boxed{\prod_{i=1}^{k} \frac{p_i^{e_i+1} - 1}{p_i - 1}} \tag{1}$$

Il suffit de retirer n pour avoir la somme de ses diviseurs stricts.

Exercice 1. En utilisant le résultat précédent, écrire une fonction en sage `suivant_aliquote2(n)` qui prend en entrée un entier positif n et qui donne en sortie la somme de ses diviseurs stricts, c'est-à-dire, le terme suivant dans la suite aliquote.

Pour réaliser les progrès réaliser, on peut, à titre d'exemple comparer le temps d'exécution pour déterminer la liste des nombres parfaits et des nombres amicaux inférieurs à 1000. Avec les fonctions du TP n° 2, on obtenait :

[7]:
```
# Résultats obtenus avec les fonctions du TP3
n = 1000

start_1 = time()
parfaits(n)
t1 = time() - start_1

start_2 = time()
amicaux(n)
t2 = time() - start_2

print(t1, t2)
```

[7]:
```
0.09298110008239746 0.11736941337585449
```

Exercice 2. Déterminer les temps obtenus pour déterminer les nombres parfaits et amicaux en utilisant votre nouvelle fonction `suivant_aliquote2(n)` puis commenter.

3 Recherche systématique de nombres sociables

Pour faire une recherche efficace et automatique, nous aurions besoin d'une méthode qui calcule une suite aliquote et qui s'arrête lorsque le nombre 1 est atteint ou un cycle est atteint. Il va falloir s'occuper des problèmes suivants :

— la conjecture a été testée que jusqu'à une certaine limite. On ne sait donc pas ce qui se passe pour les très grands nombres.

— Pour des petits nombres, les calculs n'ont pas pu être menés à leur terme. C'est le cas de 276 : personne n'a pu aller assez loin pour trouver 1 ou un cycle ; personne n'a pu prouver non plus que ça n'arriverait pas.

Pour tester si un cycle est atteint, nous pouvons rechercher si la valeur calculée est déjà présente dans le tableau des éléments de la suite.

Pour pallier au 2e problème, nous pouvons limiter le nombre de termes calculés.

Exercice 3. Écrire une fonction en Sage `analyse_suite(n, k)` qui calcule jusqu'à k termes de la suite aliquote issue de n et s'arrête prématurément si 1 ou un cycle est atteint. La variable de sortie `L` est la liste des termes calculés.

Exemples d'utilisation.

[11]:
```
analyse_suite(143, 10)
```

[11]:
```
[143, 25, 6, 6]
```

[12]:
```
analyse_suite(42, 10)
```

[12]:
```
[42, 54, 66, 78, 90, 144, 259, 45, 33, 15, 9]
```

[13]:
```
analyse_suite(42, 100)
```

[13]:
```
[42, 54, 66, 78, 90, 144, 259, 45, 33, 15, 9, 4, 3, 1]
```

Exercice 4. En utilisant ce qui précède, calculer :

1. 4 nombres parfaits inférieurs à 10 000 ;
2. 5 couples de nombres parfaits inférieurs à 10 000 ;
3. 1 cycle de longueur 5 en testant les suites qui démarrent entre 10 000 et 20 000.
4. Que se passe-t-il pour $n = 14\,316$?
5. (a) Déterminer le cycle le plus long pour des nombres de départ entre 1 et 100 (attention, si vous partez de nombres plus grand que 100, vous risquez de tomber sur des cycles trop longs pour être calculés dans un temps raisonnable).
   (b) Déterminer le second cycle le plus long *qui ne soit pas inclus dans le premier*.