Devoir maison d'informatique n°2

Nombres de Mersenne

Il est conseillé d'imprimer ses scripts Python et de bien les documenter.

1 Test de primalité

1. Écrire une fonction booléenne `estPremier(n)` qui prend en argument un entier naturel non nul n qui renvoie `True` si n est premier et `False` sinon. On pourra utiliser librement le résultat suivant : un entier $n \geqslant 2$ qui n'est divisible par aucun entier $d \geqslant 2$ tel que $d^2 \leqslant n$, est premier.
2. Dans le pire des cas, combien de divisions effectue la fonction `estPremier` pour tester la primalité d'un entier n ?
3. Construire la liste des nombres premiers inférieurs à 1000. Combien d'éléments possède cette liste ?

2 Nombres de Mersenne

Si $n \in \mathbb{N}^*$, on appelle **nombre de Mersenne** d'indice n, l'entier $M_n = 2^n - 1$. On s'intéresse aux nombres de Mersenne premiers.

4. Écrire un script qui détermine les huit premiers nombres de Mersenne qui sont premiers (afficher vos instructions). On utilisera obligatoirement une boucle du type «tant que».
5. Essayer de déterminer le neuvième, commenter.
6. On démontrera en mathématiques que si M_n est premier, alors n est premier. La réciproque est-elle vraie ?

Les nombres de Mersenne permettent d'obtenir des nombres premiers «gigantesques». Le 12 avril 2009 a été découvert le 47-ième nombre premier de Mersenne [1], il s'agit de $2^{42\ 643\ 801} - 1$.

7. Donner l'écriture binaire de ce nombre de Mersenne. En déduire une estimation de la quantité de mémoire nécessaire pour stocker ce nombre dans un ordinateur. Comparer avec la quantité de mémoire vive (RAM) de votre ordinateur.

1. Ce n'est d'ailleurs pas le plus grand nombre de Mersenne trouvé, voir le site `mersenne.org`.

8. (Facultatif) On peut montrer que le nombre de chiffres de l'écriture décimale d'un entier $n > 0$ est $\lfloor \log_{10} n \rfloor + 1$ où $\lfloor\ \rfloor$ désigne la partie entière. On rappelle que $\lfloor x \rfloor$ est l'**unique** entier vérifiant

$$x - 1 < \lfloor x \rfloor \leqslant x.$$

Donner le nombre de chiffres de l'écriture décimale de ce nombre de Mersenne. On essaiera de trouver un moyen intelligent qui évite la force brute et donc le calcul de ce nombre de Mersenne gigantesque.

3 Test de Lucas-Lehmer

On peut considérablement améliorer le test de primalité d'un nombre de Mersenne grâce au Test de Lucas-Lehmer.

Théorème 1 (Lucas-Lehmer) *Soit $p \geqslant 3$ un nombre impair, $M = 2^p - 1$ et u la suite définie par :*

$$u_0 = 4 \quad et \quad \forall n \in \mathbb{N},\ u_{n+1} = u_n^2 - 2 \mod M.$$

Alors le nombre de Mersenne $M = 2^p - 1$ est premier si et seulement si $u_{p-2} \equiv 0 \mod M$.

Par exemple avec $p = 5$, on a $M = 2^5 - 1 = 31$. Ainsi $u_1 = 4^2 - 2 \mod 31 = 14$, $u_2 = 14^2 - 2 \mod 31 = 8$, $u_3 = 8^2 - 2 \mod 31 = 62 \mod 31 = 0 \mod 31$. Donc $2^5 - 1$ est premier.

9. Écrire une fonction `suite_test(n, p)` qui prend en argument deux entiers naturels n et p et renvoie l'entier u_n.
10. En déduire une fonction booléenne `lucas(p)` qui prend en argument un entier $p \geqslant 2$ et renvoie `True` si $2^p - 1$ est premier et `False` sinon.
11. Vérifier que M_{61} est le neuvième nombre de Mersenne premier. Combien de calculs de modulo sont nécessaires pour ce test avec la fonction `lucas` ? Et si l'on exécute `est_premier(2**61-1)` ?
12. Déterminer les nombres de Mersenne premiers d'indice inférieur à 1000.

4 Nombres parfaits

Un entier $n \geqslant 1$ est dit **parfait** si la somme de ses diviseurs **stricts** (c'est-à-dire les diviseurs strictement inférieurs à n) est égale à n.

Par exemple, les diviseur stricts de 10 sont $1, 2$ et 5 (on ne prend pas le diviseur 10). On a $1+2+5 = 8 \neq 10$ donc 10 n'est pas parfait.

13. Écrire une fonction `estParfait(n)` qui prend en argument un entier naturel non nul n qui renvoie `true` si n est parfait et `false` sinon.
14. Écrire une fonction `listeParfait(N)` qui prend en argument un entier naturel non nul N et qui renvoie la liste des nombres parfaits inférieurs ou égaux à N.
15. En déduire les nombres parfaits inférieurs à 1000, afficher le temps de calcul à l'aide de la fonction `time` du module `time`.

Remarque : on ne sait toujours pas s'il existe des nombres parfaits impairs.

Le cas pair est plus simple, on a le résultat suivant :

Théorème 2 *Un nombre **pair** est parfait si et seulement si il est de la forme*

$$2^{n-1}(2^n - 1)$$

avec n un entier tel que le nombre de Mersenne $2^n - 1$ soit premier.

16. A l'aide ce dernier théorème, écrire une autre fonction `listeParfait_pair(N)` qui prend en argument un entier naturel non nul N et qui renvoie les nombres parfaits **pairs** inférieurs ou égaux à N.
17. Afficher les nombres parfaits pairs inférieurs à un milliard ainsi que le temps de calcul. Tester votre fonction avec $N = 10^{36}$ puis avec $N = 10^{37}$. Si l'ordinateur n'y arrive pas avec 10^{37}, votre fonction est améliorable...