TP5 : Génération des nombres premiers

Dans ce TP, nous allons nous intéresser au problème de la génération de tous les nombres premiers inférieurs à une valeur d'entrée N.

Exercice 1 (Pour commencer)**.**
On rappelle qu'un nombre n est premier si et seulement si $n \geq 2$, et ses seuls diviseurs sont 1 et n.

1. Écrire une fonction `estPremier(n)` qui renvoie `True` si n est premier, et `False` sinon. Expliquer pourquoi on peut avoir une complexité dans le pire cas en $O(\sqrt{n})$. Que dire de sa complexité en moyenne ?
2. En utilisant la fonction `estPremier`, écrire une fonction `listePremiers(N)` qui renvoie la liste de tous les nombres premiers inférieurs à N. Donner une estimation naïve de sa complexité. Que pensez-vous de cette estimation ?

Exercice 2 (Le crible d'Ératosthène)**.**
Le crible d'Ératosthène est une méthode plus efficace que celle vue à l'exercice précédent pour générer les nombres premiers. Voici sa description sur Wikipédia.

> *L'algorithme procède par élimination : il s'agit de supprimer d'une table des entiers de 2 à N tous les multiples d'un entier (autres que lui-même).*
>
> *En supprimant tous ces multiples, à la fin il ne restera que les entiers qui ne sont multiples d'aucun entier à part 1 et eux-mêmes, et qui sont donc les nombres premiers.*
>
> *On commence par rayer les multiples de 2, puis les multiples de 3 restants, puis les multiples de 5 restants, et ainsi de suite en rayant à chaque fois tous les multiples du plus petit entier restant.*
>
> *On peut s'arrêter lorsque le carré du plus petit entier restant est supérieur au plus grand entier restant, car dans ce cas, tous les non-premiers ont déjà été rayés précédemment.*
>
> *À la fin du processus, tous les entiers qui n'ont pas été rayés sont les nombres premiers inférieurs à N.*

1. On se propose de coder une première version du crible. Pour cela, on initialise un tableau `candidat` de taille N à `True`, et on passe ses deux premières cases à `False`. Au cours du crible, à chaque fois qu'un entier n est rayé, on passe `candidat[n]` à `False`.
   Écrire une fonction `Eratosthene(N)` qui renvoie la liste des nombres premiers inférieurs à N.
2. Tester cette fonction sur des valeurs de N croissantes en mesurant son temps d'exécution. On pourra utiliser pour cela la fonction `process_time()` du module `time`. Jusqu'à quelle valeur de n renvoie-t-elle un résultat en moins de 10 secondes ?

```
from time import process_time()

start = process_time()
# Calcul complexe
print("Le calcul complexe a pris", process_time()-start, "secondes")
```

3. On remarque que dans le crible, au moment de supprimer tous les multiples d'un nombre premier n, tous les multiples inférieurs à n^2 ont déjà été rayés, puisqu'ils contiennent un facteur premier plus petit que n.

   Écrire une fonction `Eratosthene2(N)` qui utilise cette optimisation : pour chaque nombre premier n, on retire les multiples de n à partir de n^2 au lieu de $2n$. Comparer le temps d'exécution de `Eratosthene2(N)` et de `Eratosthene(N)` sur des grandes valeurs de N.

Exercice 3 (Implémentation linéaire du crible)**.**
La complexité pour rayer tous les multiples d'un nombre premier n dans le crible est $\Theta(N/n)$. Cette opération se fait sur tous les nombres premiers inférieurs à $\sqrt{N}$; on note $\mathbb{P}_{\sqrt{N}}$ cet ensemble. La complexité du crible d'Eratosthène implémenté dans l'exercice précédent est donc

$$\Theta\left(\sum_{n\in\mathbb{P}_{\sqrt{N}}} \frac{N}{n}\right).$$

Il est possible de donner la valeur asymptotique de cette somme, qui vaut $\Theta(N \ln\ln N)$. Cela peut s'interpréter de la sorte : en moyenne, chaque nombre du tableau est considéré $\Theta(\ln\ln N)$ fois en tant que multiple d'un nombre premier — en d'autres termes, les nombres $n < N$ ont en moyenne $\ln\ln N$ facteurs premiers distincts.

On propose une solution pour éviter cette redondance et atteindre une complexité $O(N)$, qui pousse l'idée d'amélioration de la question 1.3 jusqu'au bout. Au moment de considérer les multiples du plus petit nombre premier non rayé n, on ne génère que les multiples avec un nombre qui n'est pas déjà rayé. Cela nécessite d'avoir constamment accès à la liste des nombres non rayés pendant l'exécution du crible.

Nous allons pour cela utiliser une liste doublement chaînée un peu particulière. Nous allons utiliser un tableau `t` dont les cases sont des objets de la classe `Case` définie ci-dessous. Elle contient deux valeurs `pred` et `succ`, initialisées à `None`, et qui indiquent la position de la case non vide qui suit ou qui précède dans `t`, respectivement.

```
class Case :
    prec=None   // position de la case non-vide précédente
    suiv=None   // position de la case non-vide suivante
```

1. On veut construire une liste doublement chaînée qui initialement contient toutes les valeurs entre 0 et $n-1$. Pour cela, on utilise un tableau dont chaque élément est de type `Case`. Le premier élément a pour prédécesseur `None`, et le dernier élément a pour successeur `None`. Autrement, pour chaque indice i dans le tableau, le prédécesseur est l'indice $i-1$, et le successeur l'indice $i+1$. Écrire une fonction `ListeDoublementChainee(n)` qui initialise un tel tableau.
2. Écrire une fonction `RetirerElement(x,t)` qui retire l'élément `x` de la liste doublement chaînée `t`, de complexité $O(1)$. On utilisera pour cela les opérations décrites en cours. Attention au cas où `x` est le premier ou le dernier élément de `t` !
3. Étant donné un élément `x` présent dans `t`, et une valeur `xmax`, écrire une fonction `SousListe(x,xmax,t)` qui renvoie la liste des éléments compris entre `x` et `xmax` dans `t`.
4. Écrire une fonction `Eratosthene3(N)` qui implémente le crible d'Ératosthène de complexité linéaire décrit ci-dessus. Pour chaque nombre premier n dans la boucle principale, on calcule la liste des multiples de n avec un nombre pas encore rayé

compris entre n en N/n, puis on retire chacun de ces multiples de la liste doublement chaînée des nombres pas encore rayés. Attention, il faut bien calculer tous les multiples avant de les supprimer de la liste, autrement on en oubliera (par exemple, si $n = 3$, on ne doit pas retirer 9 avant d'avoir calculé le multiple $3 \times 9 = 27$).

5. Tester la fonction `Eratosthene3(N)`. Parvenez-vous à trouver une valeur de N pour laquelle son exécution est plus rapide que celle de `Eratosthène(N)` ? Essayez d'en estimer la valeur.