

|  |  |  |
| --- | --- | --- |
| **Organisation** | : | Université Virtuelle du Burkina |
| **Filière** | : | Fouilles de Données et Intelligence Artificielle |
| **Niveau** | : | Master |
| **Promotion** | : | 2022-2023 |

**MINI-PROJET**

Théories des graphes et complexité algorithmique

**Chargé du Cours :** Dr Azise Oumar DIALLO

**GROUPE 6**

|  |  |  |
| --- | --- | --- |
| **Nom** | **Prénom** | **E-mail** |
| **SAVADOGO** | Idrissa | savadogoidrissa17@gmail.com |
| **SIRIMA** | Constant |  |
| **NANA** | Jerémie |  |
| **KIEMA** | Wend-denda Arsène | kiemaarsene@gmail.com |

**Problème : Générateur de Nombres Premiers (Prime Generator)**

# Description du Problème

Peter souhaite générer des nombres premiers pour son cryptosystème. Votre tâche consiste à générer tous les nombres premiers compris entre deux nombres donnés.

* Entrée :

Le programme prend en entrée le nombre t de cas de test dans une seule ligne (t <= 10).

Chaque ligne des t lignes suivantes contient deux nombres m et n (1 <= m <= n <= 1000000000, n-m <= 100000) séparés par un espace. Ces deux nombres délimitent l'intervalle dans lequel chercher les nombres premiers.

* Sortie :

Pour chaque cas de test, le programme doit imprimer tous les nombres premiers p tels que m <= p <= n, un nombre par ligne, les cas de test étant séparés par une ligne vide.

Exemple :

|  |
| --- |
| Entrée :  2  1 10  3 5  Sortie :  2  3  5  7  3  5 |

* Les contraintes de temps et de mémoire :

Le temps d’exécution du programme ne doit pas excéder 6 secondes et l’utilisation de la mémoire est limité à 1536MB.

**Le langage Python a été utilisé pour résoudre le problème.**

# Cas Pratique dans la Vie Réelle

Imaginons qu'une entreprise de sécurité souhaite mettre en place un système de chiffrement personnalisé pour ses communications sensibles. Pour cela, elle doit générer des nombres premiers dans un intervalle spécifique pour créer les clés de chiffrement. Le générateur de nombres premiers serait alors utilisé pour déterminer tous les nombres premiers compris entre les bornes de l'intervalle nécessaire pour sécuriser les communications.

# Algorithme pour Résoudre le Problème

1. Définir une fonction qui prend deux entiers m et n en entrée, représentant les bornes de l'intervalle.
2. Pour chaque nombre x dans l'intervalle [m, n], vérifier s'il est premier.
3. Pour vérifier si x est premier, diviser x par tous les entiers de 2 à la racine carrée de x.
4. S'il n'y a aucun diviseur, alors x est premier et doit être imprimé.
5. Répéter les étapes 2 à 4 pour chaque nombre dans l'intervalle.
6. Répéter le processus pour chaque cas de test.

# Complexité du Problème

Pour analyser la complexité du code, nous devons examiner chaque fonction séparément :

1. crible\_eratosthenes :
   * La boucle principale de cette fonction s'exécute jusqu'à ce que p \* p dépasse la valeur limite, ce qui prendra approximativement sqrt(limite) itérations.
   * À chaque itération, la fonction marque les multiples de p comme non premiers dans une liste de taille limite+1. Cela nécessite environ limite/p opérations.
   * Donc, la complexité totale de cette fonction est approximativement O(sqrt(limite) \* (limite/2 + limite/3 + limite/5 + ...)), où les valeurs 2, 3, 5, ... sont les nombres premiers trouvés lors de l'itération. Cela peut être simplifié à O(n log log n), où n est la valeur de la limite.
2. afficher\_numbre\_premiers\_entre :
   * La fonction commence par appeler crible\_eratosthenes avec la valeur de limite = int(n \*\* 0.5) + 1, donc cette partie de la fonction a une complexité de O(sqrt(n) \* log log sqrt(n)), qui est O(sqrt(n) \* log log n).
   * Ensuite, la fonction parcourt la liste des nombres premiers jusqu'à la valeur limite et marque les nombres non premiers dans la plage (m, n). Cette partie a une complexité d'environ O(n/m).
   * Donc, la complexité totale de cette fonction est approximativement O(sqrt(n) \* log log n + n/m), où n est la valeur de n et m est la valeur de m.
3. Boucle principale :
   * La boucle principale du programme lit le nombre de cas test t et, pour chaque cas de test, appelle la fonction afficher\_nombre\_premiers\_entre(m, n). La complexité totale de cette boucle principale est O(t \* (sqrt(n) \* log log n + n/m)), où n et m sont les valeurs respectives de n et m pour chaque cas de test.

En résumé, la complexité totale du code est O(t \* (sqrt(n) \* log log n + n/m)), où t est le nombre de test cases, n est la valeur de n pour chaque test case, et m est la valeur de m pour chaque test case.

La complexité dépendra fortement des valeurs des test cases. Si les test cases ont des valeurs élevées pour n et m, la complexité sera plus élevée. Cependant, étant donné les contraintes mentionnées dans le problème (t <= 10, n-m <= 100000), le code devrait fonctionner efficacement dans la plupart des cas.

Références :

<https://www.spoj.com/>

<https://zanotti.univ-tln.fr/ALGO/I51/Crible.html>

<https://www.maths-et-tiques.fr/index.php/histoire-des-maths/nombres/les-nombres-premiers>

[https://chat.openai.com](https://chat.openai.com/)