Analyse de Complexité Algorithmique

Votre Nom

20 février 2025

1 Conceptualisation

Source : `exo1/`
Compilation : `gcc main.c myarray.c -I ./`

Après avoir appris les bases sur la complexité algorithmique, il est temps de pratiquer.

Step 1 : Analyse de la complexité

Vous tombez à pic, nous venons tout juste de finaliser les fonctions principales d'une librairie permettant de gérer des tableaux d'entiers. Puisque vous venez d'arriver, commencez par la documentation.

Calcul de la complexité Calculez la complexité en notation $\mathcal{O}$ de la fonction suivante :

`removeElement(...)`

Rendu : un document exprimant la complexité $\mathcal{O}$ ainsi que le calcul détaillé.

Step 2 : Implémentation d'une fonction optimisée

Vous avez déjà terminé ? Parfait ! Il reste encore beaucoup de choses à réaliser. Tenez, il y a une fonction qui n'a pas encore été implémentée.
Écrivez la fonction suivante :

`removeElementS(...)`

Les tests ont déjà été écrits, vous avez juste à la rendre conforme. Le seul prérequis est le suivant :
— Cette fonction doit avoir une complexité inférieure à $\mathcal{O}$ (n^2).

Note : D'après Mr_Pay, il est possible de la réaliser en $\mathcal{O}(n)$.

Rendu le programme ainsi qu'un justificatif écrit de la complexité théorique de votre implémentation.

2 Comparaison et optimisation

Source : `exo2/`
Compilation : `gcc main.c myarray.c -I ./ -lm`

Hello newbie, bienvenue dans la cour des grands. La direction a décidé qu'il fallait que l'on améliore notre calculateur. Tu vas devoir refactoriser la librairie d'opération.
Voici l'état actuel, la librairie fonctionne et on a une base de test solide. On manipule de très très grands nombres, Ouet des nombres qui ne rentrent même plus dans des `long long`, nous sommes donc obligés de les stocker différemment. Deux moyens de stockage sont possibles :

— En chaîne de caractères (`char *`) avec chaque caractère représentant un chiffre.
— En tableau d'entiers`int *` pour optimiser certains calculs.

On a de un gros problème sur la fonction de multiplication, le problème tu t'en doute, nous sommes beaucoup trop lents. Tu vas donc commencer par là.

J'ai déjà créé deux algorithmes, mais je n'ai pas eu le temps de les essayer.

Écriture et comparaison de deux algorithmes de multiplication

Algorithme 1

J'ai inventé un algorithme de fou ! Si je prends deux entiers et que je les divise en deux, puis que je réadditionne les parties multipliées entre elles de manière récursive, cela pourrait accélérer le calcul.

On pose : pour deux grands entiers a et b :

$$a \times b = (10^n) \times a_0 b_0 + a_1 b_1 + ((a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1) \times 10^{n/2}$$

Tel que :

$$a = a_0 \times 10^{n/2} + a_1, \quad b = b_0 \times 10^{n/2} + b_1$$

Note : Ici, on considère que les entiers fournis seront de même taille.

Step 0 : Condition de fin de récursion

Comme cet algorithme fonctionne avec une récursion, il faut poser une condition d'arrêt. **Condition à définir : TODO ???**

Step 1 : Création des sous-entiers

Diviser chaque entier en deux sous-entiers de taille $n/2$ (avec n la taille du grand entier).

Step 2 : Calcul des trois entiers intermédiaires

Créer trois grands entiers : `fst`, `snd`, `thrd`, représentant les calculs suivants :

$$\text{Fst} = a_0 b_0 \quad \text{thrd} = a_1 b_1$$

$$\text{snd} = (a_0 + a_1) \times (b_0 + b_1) - \text{Fst} - \text{thrd}$$

Step 3 : Reformation du nombre final

Effectuer :
— Un décalage de bits (extitbit shift) sur extttfst et extttsnd.
— Additionner extttfst, extttsnd et exttttthrd.
Attention : Vérifier la fin de la récursion.

Algorithme 2

Je me suis rappelé de mes vieux cours de prépa, si on utilise la transformée de Fourier, on pourrait réduire drastiquement la complexité de l'algorithme.

Bon ici il va falloir utiliser les librairie complex et math. // ilustration lib a utilisé

Comment ça marche,

On va appliquer une transformé de fourrier sur nos nombres

$$\mathrm{Ft}(a) = a', \quad \mathrm{Ft}(b) = b'$$

On réalise une multiplication entre nos 2 nombre de sorte à ce que chaque digit de a' soit multiplier avec son symétrique de b'.

$$c' = \sum_{i=0}^{n-1} a'_i \times b'_i \times 10^i, \quad \text{pour } 0 \leq i < n$$

Puis, on applique l'inverse de la transformée de Fourier sur le résultat c' :

$$\mathrm{IFT}(c') = c$$

On convertit chaque nombre complexe en nombre réel :
Fonction : `f() =(int)(creal('digit') + 0.5) ;`

$$\text{résultat} = \sum_{i=0}^{n-1} f(c_i) \times 10^i$$

Reste à faire : Implémenter la transformée de Fourier ou trouver une bibliothèque existante qui le fait