EL KANBI Asmae
HANI Pamella

Compte-rendu du TP2 : La suite de Fibonacci

L'objectif principal de cette séance de TP a été d'étudier les différents types d'algorithmes permettant de calculer les termes de la suite de Fibonacci, et d'analyser leurs complexités afin de comparer les différentes approches/méthodes utilisées.

Approche itérative

Valeurs

On remarque que les valeurs pour n>47 sont fausses. En effet, nous avions comme type de la valeur de retour un int : ce type a pour valeur maximale 2147483647 (10 chiffres). Et à n=47, F(47) = 2971215073 (10 chiffres). On voit bien que F(47) dépasse la valeur maximale que peut contenir un entier int en C : c'est pour cela que la valeur obtenue est incorrecte. Nous avons donc pensé à modifier le type du résultat renvoyé en unsigned long long, c'est-à-dire le type prenant la plus grande valeur possible dans une variable (20 chiffres), en prenant des valeurs précises.

Ainsi, nous pourrons aller au maximum jusqu'à n = 93. F(93) sera correcte, F(94) sera incorrecte. Si la précision des valeurs importe peu, on peut également choisir le type double, qui prend 37 chiffres. Avec ce type, on pourra atteindre n = 173.

> On choisira alors de renvoyer le résultat de type double pour le reste des fonctions implémentées (cf. fibo.c)

Temps d'exécution

Le programme ne met jamais plus d'une seconde à s'exécuter, même pour des valeurs astronomiques de n.

Complexité

Cet algorithme a une complexité linéaire O(n), on le voit notamment avec le temps d'exécution.

Approche récursive

Temps d'exécution et complexité

À partir de N = 43, la fonction prend plus d'une seconde à s'exécuter. Pour N = 50, cela prend 44 secondes. Cette fonction a une complexité beaucoup trop grande, elle est clairement exponentielle.

Nous décidons alors de trouver un nouvel algorithme récursif qui aurait une plus petite complexité afin que le programme s'exécute rapidement pour des valeurs correctes de N (N=50 par exemple).

Pour N=60, N=75 et N=100, le programme s'exécute en moins d'une seconde.

Sa complexité est de : O(log(n)).

Méthode du nombre d'or

Temps d'exécution

Le temps d'exécution est inférieure à une seconde, pour toute valeur de n.

Complexité

Cette fonction appelle la fonction puissance qui est récursive, et qui a une complexité de O(log(n)). On peut donc en déduire que la fonction fibo_golden a donc une complexité de O(log(n)).

Approche matricielle

Valeurs

En comparant les résultats de cet algorithme avec ceux de l'approche itérative, la deuxième version récursive ainsi que celle du nombre d'or, lorsqu'on exécute les fonctions pour n = 60 et n = 75, les résultats sont essentiellement les mêmes.

Cependant, pour n = 100, en comparant l'ensemble des algorithmes, les résultats diffèrent sur les 6 derniers chiffres. Pourtant, en calculant la marge d'erreur entre la vraie valeur de F(100) et les valeurs trouvées les fonctions fibo_iter, fibo_rec2, fibo_golden et fibo_matrice, on trouve qu'elle est infiniment petite (environ 10^⁻14%). On peut conclure que celle-ci est négligeable. La principale raison de cette erreur est dû à l'imprécision du type double.

Temps d'exécution

Le temps d'exécution est toujours inférieure à une seconde, pour n'importe quelle valeur de n.

Complexité

Cette fonction utilise l'algorithme de l'exponentiation rapide (O(log(n)) qui est beaucoup plus efficace que la fonction puissance itérative qui a une complexité linéaire. Elle a donc une complexité de O(log(n)).

Conclusion générale

On a vu, lors de ce TP, qu'il existe plusieurs manières d'implémenter la suite de Fibonacci. Cependant, certaines sont inefficaces, comme la première version récursive de la fonction, et d'autres sont beaucoup plus rapides et adaptées aux grandes valeurs comme la version itérative, la deuxième version récursive ou encore la version matricielle.