Escapade algorithmique avec Fibonacci

Lucas RODRIGUEZ

21 mars 2019 - 16h30

Plan général

- Problématique générale
- Sommaire (5 axes)
- **1ère idée (Approche naïve)** : formule de récurrence (un terme de la suite résulte de la somme des 2 précédents)
- Implémentation avec liste (Complexité temporelle linéaire, Complexité spatiale linéaire)
- Implémentation sans liste (Complexité temporelle linéaire, Complexité spatiale constante)
- **2ème idée** : formule explicite (suite de fibo $\iff$ suite récurrente d'ordre 2, polynôme caractéristique dont les racines sont le nombre d'or et son conjugué). On obtient une formule explicite de $(F_n)_{n\in\mathbb{N})}$ [Formule de Binet]
- Implémentation (Permet de calculer F_n sans avoir les termes précédents MAIS nécessite une exponentiation coûteuse en mémoire)
- **3ème idée** : Récursivité (cas de base + propagation)

Programme récursif : un processus, qui dépend de paramètres et qui fait appel à ce même processus sur d'autres arguments plus « simples ». Autrement dit, c'est un algorithme qui s'invoque lui-même

- Meilleure flexibilité MAIS pas performant (calcule plusieurs fois des termes)
- Amélioration possible au niveau de la mise en cache (pour limiter les calculs inutiles). Optimisation du temps possible au détriment de l'espace et inversement
- Comparaison des 2 méthodes (iter et rec) : linéaire & exponentielle $\Rightarrow$ Récursivité pas adapté au calcul des fibos (comportement chaotique)
- **4ème idée** : On repart de la formule de récurrence et on pose le vecteur X_n et la matrice carrée A. Par analogie avec la formule de récurrence, on obtient la suite de matrice $(X_n)_{n\in\mathbb{N}}$ définie par récurrence de raison A d'où cette formule explicite : $\forall n \in \mathbb{N}, X_n = A^n * X_0$. Pour trouver le nombre de Fibonacci souhaité, on a :

$$A^n = \begin{pmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{pmatrix}$$

  Pour exponentier rapidement et de manière performante la matrice A, on décide d'utiliser la méthode d'expo rapide qui permet de réduire considérablement le nombre de multiplications effectuées.
- Au lieu d'utiliser une exponentiation naïve, on décide d'implémenter une version récursive avec ces 2 éléments remarquables. La complexité de cet algorithme est logarithmique.
- On l'applique maintenant au produit matriciel ! ! et on en conclut sur la complexité logarithmique de `fibo_exporap`.
- **Comparaison des capacités** : On a à ce stade 3 algorithmes qui fonctionne. L'un d'eux, la version récursive peut être éliminé avant les tests unitaires : en effet, il ne satisfait pas les contraintes de complexité imposées. (on rencontre un Overflow vers $n = 10^3$).
- **Graphique**
- **Conclusion** : La version avec l'expo rapide est la plus performante (plus rapide et moins coûteuse en espace mémoire)
- **Application** Les nbres de Fibo interviennent dans l'analyse de l'algorithme d'Euclide. Ce dernier permet de calculer le PGCD de 2 entiers notés ici a et b. Voici 2 versions : l'une itérative, l'autre récursive

  Ils effectuent le même nombre d'opérations élémentaires ! ! Le th de Lamé nous indique que le nbre de div.euc. réalisées par l'algo est $\leq$ à 5 fois le nbre de chiffres du plus petit nbre entre a et b

— Le pgcd de 2 fibos consécutifs est égal à 1 et nécessite exactement n div.euc.
— **Conclusion** : Les nbres de Fibo sont des indicateurs pour la complexité (nbres d'opérations à effectuer) de l'algo d'Euclide.