Algorithmique et Programmation 1

TD7 : Complexité

1 Opérations sur les matrices

1. Écrivez une fonction `add(A,B)`, qui calcule la somme de deux matrices (listes de listes) A et B qu'on suppose de même taille. Quelle est la complexité de cette fonction, en fonction de la taille de a et b ?
2. Écrivez une fonction `prod_vec(A,X)`, qui calcule le produit d'une matrice A avec un vecteur X, en supposant leurs tailles compatibles. Quelle est la complexité de votre fonction ?
3. Écrivez une fonction `puissance(a,n)`, qui élève une matrice a, supposée carrée, à la puissance n. On pourra pour cela utiliser la fonction `prod` définie en cours, itérée $n-1$ fois. Quelle est la complexité de votre fonction ?
4. Même question en utilisant une exponentiation rapide, comme vu dans le TD précédent.

$$A^n = \begin{cases} \left(A^{n/2}\right)^2 & \text{si } n \text{ est pair} \\ A * \left(A^{\lfloor n/2 \rfloor}\right)^2 & \text{si } n \text{ est impair} \end{cases}$$

2 Recherche dans des listes

1. On considère la fonction `find(x,lst)` qui renvoie la position de la première occurrence de `x` dans la liste `lst`, ou -1 si `x` n'apparaît pas dans la liste `lst`, telle qu'implémentée dans le TD4. On suppose que la fonction s'arrête dès qu'elle a trouvé une occurence de x.

   Quelle est la complexité en moyenne de cette fonction ? Dans le pire cas ?
2. On suppose maintenant que la liste `lst` est triée. Réécrire la fonction `find(x,lst)` en utilisant une recherche dichotomique.

   Le principe de la recherche dichotomique consiste à couper `lst` en deux et comparer l'élément recherché `x` avec l'élément central `y`. S'ils sont égaux, on renvoie la position correspondante. Si $x < y$, on continue la recherche dichotomique sur la liste de gauche, et si $x > y$, on continue la recherche dichotomique sur la liste de droite.
3. Quelle est la complexité dans le pire cas de la recherche dichotomique ?

3 Retour sur Fibonacci

Les nombres de Fibonacci F_n sont définis par la formule de récurrence d'ordre 2 suivante :

$$(F_{-1} = 0) \quad F_0 = 1, \quad F_1 = 1, \quad \forall n \geq 2, \quad F_n = F_{n-1} + F_{n-2}$$

Comme vu en cours, l'implémentation récursive naïve d'une fonction calculant le n-ième nombre de Fibonacci a une complexité exponentielle. Il est possible de faire beaucoup mieux, comment nous allons le voir dans cet exercice.

1. Écrivez une fonction récursive `fibonacciCouple(n)` qui renvoie le couple (F_n, F_{n+1}). On ne veut qu'un seul appel récursif au sein du corps de la fonction `fibonacciCouple(n)`.
2. Écrivez une fonction wraper `fibonacci2(n)` qui utilise la fonction récursive `fibonacciCouple` pour calculer la valeur de F_n. Quelle est la complexité de votre fonction ?
3. On considère la matrice
   $$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.$$
   Quel est le résultat de l'opération suivante ?
   $$A \times \begin{pmatrix} F_{n-1} \\ F_n \end{pmatrix}$$
4. Expliquez comment obtenir le vecteur (F_{n-1}, F_n) à partir de la matrice A et du vecteur $(0, 1)$.
5. En utilisant les opérations sur les matrices définies à l'exercice 1, écrivez une fonction `fibonacci3(n)` qui calcule F_n, en utilisant la matrice A. Quelle est la complexité de votre fonction ?