Devoir à la maison

Ce devoir est à rendre le mercredi 7 décembre au début du cours. Une partie très importante de la note dépendra de la clarté du pseudocode et des explications. **Une justification succincte du fonctionnement de vos algorithmes et de leurs complexités est obligatoire.** Les étudiants dont le nom commence par A à C inclus ferons le premier exercice, ceux dont le nom commence par D à H feront le deuxième, ceux dont le nom commence par I à M feront le troisième et les autres feront le quatrième.

Polynômes

Un polynôme s'écrit $P = a_0 + a_1X + a_2X^2 \cdots + a_nX^n$. On veut implémenter un type abstrait pour les polynômes. Il s'agit d'implémenter des fonctions qui vont permettre, de sommer, de multiplier, de dériver, d'intégrer des polynômes pour pouvoir ensuite oublier l'implémentation concrète des polynômes et utiliser uniquement ces fonctions. Donner systématiquement la complexité de vos algorithmes.

(a) Proposer une structure de données pour un polynôme, sous forme de tableau de ses monômes (un monôme est un terme de la forme a_iX^i).

(b) Donner une fonction qui affiche un polynôme.

(c) Donner une fonction qui renvoie la somme de deux polynômes.

(d) Donner une fonction qui renvoie le produit de deux polynômes.

(e) Proposer maintenant un type pour un polynôme, sous forme de liste de ses monômes. Attention les monômes de coefficient 0 ne doivent pas être représentés.

(f) Pour cette nouvelle représentation, donner une fonction qui affiche un polynôme, une qui fait la somme de deux polynômes et une dernière qui fait le produit.

(g) Donner une fonction qui réalise la dérivation d'un polynôme.

Exponentiation de matrices

Soit A et B deux matrices carrées de dimension n, le produit de ces deux matrices $C = A * B$ est défini par $C_{i,j} = \sum_{0 \leq \alpha \leq n-1} A_{i,\alpha} B_{\alpha,j}$.

(a) Proposer un algorithme qui retourne le produit de deux matrices. Rappeler sa complexité.

(b) On veut calculer la puissance k-ème d'une matrice A. Proposer un algorithme simple qui calcule A^k en utilisant votre procédure de produit. Quelle est sa complexité ?

(c) Améliorer cet algorithme en vous inspirant de l'exponentiation rapide des nombres. La complexité de cet algorithme est elle meilleure que celle de l'algorithme précédent ?

La suite de Fibonnacci est définie par $F_n = F_{n-1} + F_{n-2}$ et $F_0 = F_1 = 1$. On se donne le programme récursif

```
int Fibo(int n){
if (n==0 || n==1) return 1;
return (Fibo(n-1) + Fibo(n-2));
}
```

(d) Quelle est la complexité de Fibo (en nombre d'additions) ?

(e) Remarquez que

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$$

En déduire un algorithme qui utilise l'exponentiation rapide de matrices pour calculer F_n. Donner sa complexité.

Solitaire

On va modéliser le casse tête du solitaire (anglais). Pour la définition aller voir

`https://fr.wikipedia.org/wiki/Solitaire_(casse-tête)`

Donner systématiquement la complexité de vos algorithmes.

1. Donner une structure de donnée SOLITAIRE qui permettra de stocker l'état du plateau du solitaire à n'importe quel moment de sa résolution. Donner une structure de donnée POS pour représenter la position d'un pion.
2. Donner un algorithme qui prend en entrée un SOLITAIRE s et deux POS p1 et p2 et qui renvoie vrai si le mouvement du pion de p1 à p2 est possible, sinon faux.
3. Étant donné un SOLITAIRE s donner un algorithme qui vérifie si il existe un déplacement de pion possible pour s.
4. Proposer un algorithme qui joue des déplacements aléatoires corrects tant qu'il y en a encore.
5. Comment représenter l'ensemble des parties de solitaire possibles avec une structure de données ? Comment grâce à cette structure peut-on trouver le nombre minimal de pions qui restent sur le plateau ?

Listing

On manipule des listes dont chaque élément contient un caractère, elles représentent donc des mots. Donner la complexité de chacun de vos algorithmes, elle doit être la plus basse possible.

1. Donner un algorithme qui prend en entrée une liste et un caractère et qui renvoie le nombre de fois que ce caractère apparaît.
2. Étant donné en entrée les listes l_1 et l_2, donner un algorithme qui décide si la liste l_1 est contenue dans l_2. Par exemple *oba* est contenu dans *baobab* mais pas dans *oababa*.
3. On suppose qu'on a un opérateur de comparaisons $<$ sur les caractères, qu'on utilise comme la comparaison sur les entiers. Donner un algorithme qui prend en entrée l_1 et l_2 et qui renvoie vrai si l_1 est plus petit que l_2 lexicographiquement, faux sinon. On dit que l_1 est plus petit lexicographiquement que l_2, si la première lettre qui diffère entre l_1 et l_2 est pus petite dans l_1. L'absence de lettre est considéré comme plus petit que n'importe quelle lettre.
4. Étant donné un tableau de listes de caractères, donner un algorithme stable qui trie ce tableau dans l'ordre lexicographique. La taille du tableau est n et la taille maximum d'une liste est l.
5. Donner un algorithme de partitionnement qui prend en entrée un tableau de listes de caractères et qui renvoie un tableau contenant les listes triées selon leur première lettre.
6. Utiliser l'algorithme de partitionnement (éventuellement modifié) et une méthode diviser pour régner pour proposer un algorithme de tri efficace.