DS INFO N°4

Exercice 1 (Questions de cours).

1. Donner la définition de la complexité amortie d'un algorithme. Quelle est la complexité amortie d'une succession d'invocations de l'opération `push` à partir d'un tableau vide dans le cas d'une implémentation de pile à l'aide d'un tableau redimensionnable ? (pas de démo demandée)
2. On considère un algorithme dont la complexité est définie en fonction d'une taille n de donnée, par la relation de récurrence suivante : $T(n) = 4T\left(\left\lceil \frac{n}{3} \right\rceil\right) + n$. Donner un ordre de grandeur asymptotique de son comportement quand la taille de la donnée d'entrée devient très grande. Justifier par un calcul.
3. Donner 3 implémentations concrètes différentes vues en cours pour la structure de données abstraite de **file** : une avec des maillons chaînés, une avec un tableau de taille fixe, une avec des piles. Faites un dessin et donner quelques éléments clés pour expliquer chaque implémentation.

Exercice 2.

On propose la fonction OCaml `rev` ci-dessous implémentée avec l'opérateur de concaténation `@`. On note n la taille de la liste donnée en entrée.

```
# let rec rev l =
  match l with
  | [] -> []
  | h::t -> (rev t) @ [h];;
val rev : 'a list -> 'a list = <fun>
```

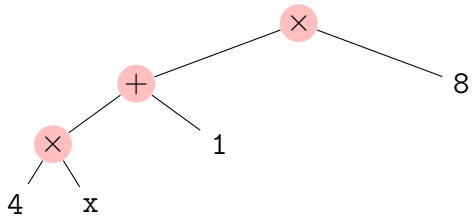
1. Donner un ordre de grandeur asymptotique de sa complexité, en justifiant proprement par des calculs.
2. Réécrire cette fonction `rev` pour qu'elle soit linéaire en la taille n de la liste donnée en entrée.

Exercice 3 (Représentation d'une expression algébrique par un arbre binaire).

Une expression algébrique peut être représentée par un arbre dont les nœuds contiennent des lexèmes. Un lexème est :

- soit une valeur numérique : on se limitera aux entiers, par exemple 3, -7, 11 ;
- soit l'indéterminée x, aussi appelée variable (on se limite aux expressions algébriques à une seule indéterminée)
- soit une opération binaire, et on se limitera aux 3 opérations binaires $+$, $-$, $\times$. On ne traite pas le cas des opérations unaires, comme par exemple l'opérateur $-$ dans l'expression (-2) car on considère que le symbole $-$ fait partie de la valeur numérique entière dans ce cas.

Par exemple, l'expression $(4x + 1) \times 8$ peut être représentée par l'arbre :



1. Dessiner l'arbre **binaire** (deux fils maximum par nœuds) correspondant à l'expression algébrique

$$(3 + 2(x + 1)) \times ((x + 6)(8x - 3)),$$

en respectant les parenthèses pour la mise sous forme d'arbre binaire.

2. Expliquez quelles propriétés de l'addition et de la multiplication permettent toujours de se ramener à un arbre **binaire**. Y-a-t-il unicité ?

3. Quelle(s) propriété(s) doit vérifier l'arbre si l'expression algébrique est bien formée ?

4. Donner le résultat affiché par les parcours pré-fixe, infixe et post-fixe pour

- l'arbre donné en exemple dans l'énoncé
- l'arbre dessiné à la question 1.

Quel parcours correspond à notre manière habituelle d'écrire les expressions algébriques ?

5. On définit en OCaml le type `expralg` comme ceci :

```
type 'a expralg = Val of 'a | Var | Add of 'a expralg * 'a expralg |
  Sub of 'a expralg * 'a expralg | Mult of 'a expralg * 'a expralg
```

a. D'après cette définition, combien y a t-il de manières de construire une expression algébrique ?

b. Écrire deux lignes de code OCaml permettant de créer les expressions algébriques $(4x + 1) \times 8$ et $(3 + 2(x + 1)) \times ((x + 6)(8x - 3))$

c. Écrire une fonction `evaluate: expralg -> int -> int` qui évalue l'expression en une valeur choisie de l'indéterminée x.

Exercice 4 (Notation polonaise inversée (à faire après l'exercice 3)).

La notation polonaise inverse (aussi appelée RPN pour *reversed polish notation*) correspond à l'écriture post-fixe d'une expression algébrique. Elle a été étudiée et mise en valeur par le mathématicien polonais Lukasiewicz. Elle est utilisée dans certains langages de programmation ainsi que pour certaines calculatrices, notamment celles de la marque Hewlett-Packard. Nous allons voir comment, à l'aide d'une pile, évaluer une expression algébrique donnée en notation polonaise inversée.

On part donc d'une expression algébrique écrite sous sa forme post-fixée. L'idée est d'utiliser une pile pour stocker les opérandes, et d'implémenter les opérations de façon à ce que l'évaluation de l'expression n'utilise que les valeurs au sommet de la pile Par exemple, pour évaluer $(x+4)-2$, noté $x\,4+2-$ en RPN, pour la valeur $x=10$, on effectue les actions suivantes au fur à mesure que l'on dépile la RPN :

- x : on empile 10
- 4 : on empile 4
- $+$: on dépile deux valeurs, que l'on additionne, et on remet le résultat dans la pile.
- 2 : on empile 2
- $-$: on dépile deux valeurs (v_2, puis v_1), que l'on soustrait $v_1 - v_2$, et on remet le résultat dans la pile.

1. Détaillez toutes les étapes de cet algorithme d'évaluation en $x=1$ sur l'expression RPN de l'expression $(4x+1)\times 8$. On écrira les valeurs contenues dans la pile à chaque étape de l'algorithme jusqu'à la fin. On réfléchira à une manière efficace de présenter les étapes de l'algorithme.

2. Quel est le cas d'arrêt de cet algorithme ?

3. Écrire une fonction `rpn : expralg -> string list` qui renvoie une liste de chaînes de caractères correspondant à l'écriture RPN d'une expression algébrique. Par exemple, si l'identificateur `expr3` désigne l'objet de type `expralg` représentant l'expression $(x+4)-2$, l'appel `rpn expr3` renvoie la liste `[''x''; ''4''; ''+''; ''2''; ''-'']`. On pourra utiliser la fonction de conversion `string_of_int` qui convertit un entier en chaîne de caractères. On représentera la multiplication par le caractère `*`.

4. Écrire une fonction `evaluate_rpn: string list -> int -> int` qui évalue une expression algébrique donnée sous sa forme RPN en un entier x choisi. On pourra utiliser la fonction de conversion `int_of_string` qui convertit une chaîne de caractères en entier, quand cela est possible.

5. Comparer en terme de complexité spatiale la fonction d'évaluation `evaluate` sur les expressions sous forme d'arbres implémentée à l'exercice 2 et la fonction `evaluate_rpn` qui évalue l'expression à partir de son écriture RPN. Quelques phrases percutantes suffisent.

Exercice 5 (Algorithme d'Euclide et théorème de Lamé).

1. (Question de cours, étudiée au dernier DS) Écrire rapidement une fonction OCaml **récursive** `euclide` implémentant l'algorithme d'Euclide qui calcule le PGCD de deux entiers a et b. On se limitera au cas des entiers naturels, c'est-à-dire que $a \geq 0$ et $b \geq 0$.

2. De quelle(s) quantité(s) dépend a priori la complexité de l'algorithme ?

3. On considère la suite de Fibonacci définie par :

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_{n+2} &= F_{n+1} + F_n \end{aligned}$$

On dit que la suite de Fibonacci est définie par une relation de récurrence d'ordre 2 car on a besoin des deux termes précédents pour calculer le terme courant.

a. Calculer les termes de la suite de Fibonacci jusqu'à F_5.

b. Montrer que la suite de Fibonacci est positive et strictement croissante pour $n \geq 2$.

c. Montrer que la reste de la division euclidienne de F_{n+2} par F_{n+1} est égal à F_n.

4. On admet dans la suite l'ordre de grandeur asymptotique suivant : $F_n \underset{n\to+\infty}{\sim} \frac{1}{\sqrt{5}}\varphi^n$ où $\varphi = \frac{1+\sqrt{5}}{2}$ est le nombre d'or.

a. Expliquer pourquoi on peut utiliser n'importe quelle fonction logarithme $\log_\alpha$, avec un réel α tel que $\alpha > 1$, pour effectuer une analyse asymptotique **en ordre de grandeur** faisant intervenir un logarithme.

b. Montrer que, pour tout $n \geq 1$, si l'algorithme d'Euclide effectue n appels récursifs pour calculer `pgcd(a,b)`, alors cela nous donne une bonne inférieure sur les entrées : $a \geq F_{n+2}$ et $b \geq F_{n+1}$. *Indication : procéder par récurrence simple en écrivant très clairement la propriété démontrée.*

c. On considère que la complexité temporelle de l'algorithme T est correctement représentée par le nombre d'appels récursifs n effectués pour les deux entrées choisies. Déduire des questions précédentes une domination asymptotique de $T(a, b)$ quand a et b deviennent grands.

d. Montrer que, $\forall n \in \mathbb{N}$, l'appel de l'algorithme d'Euclide récursif avec $a \leftarrow F_{n+1}$ et $b \leftarrow F_{n)}$ va engendrer exactement n appels récursifs.

e. En quoi ce dernier résultat permet-il d'affiner l'analyse asymptotique effectuée ?

Ce résultat est connu sous le nom de **théorème de Lamé**.

Exercice 6 (DM (maths/info) Démonstration de l'ordre asymptotique sur la suite de Fibonacci).

1. Justifier l'égalité matricielle suivante :

$$\begin{pmatrix} F_{n+1} \\ F_{n+2} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$$

2. On définit la suite de vecteurs $(U_n)_{n\in\mathbb{N}}$ de terme général :

$$U_n = \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$$

Écrire une relation de récurrence définissant la suite $(U_n)_{n\in\mathbb{N}}$, en n'oubliant pas l'initialisation de la récurrence. Quelle est l'ordre de cette récurrence ?

3. On note dans toute la suite :

$$\varphi^+ = \varphi = \frac{1+\sqrt{5}}{2} \quad \text{et} \quad \varphi^- = \frac{1-\sqrt{5}}{2}$$

Montrer que φ^+ et φ^- sont les racines du polynômes $X^2 - X - 1$. En déduire **sans calcul** les valeurs de $\varphi^+ \times \varphi^-$ et $\varphi^+ + \varphi^-$.

4. On note $A \in \mathbb{R}^{2\times 2}$ la matrice à coefficients réels à deux lignes et deux colonnes définie par :

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

Montrer l'égalité matricielle suivante :

$$A = PDP^{-1} \text{ où } P = \begin{pmatrix} 1 & 1 \\ \varphi^+ & \varphi^- \end{pmatrix}, \quad D = \begin{pmatrix} \varphi^+ & 0 \\ 0 & \varphi^- \end{pmatrix} \text{ et } P^{-1} = \frac{1}{\varphi^- - \varphi^+} \begin{pmatrix} \varphi^- & -1 \\ -\varphi^+ & 1 \end{pmatrix}$$

Vérifier que P^{-1} est bien la matrice inverse de P.

5. On définit une suite auxiliaire $(V_n)_{n\in\mathbb{N}}$ à partir de la suite $(U_n)_{n\in\mathbb{N}}$ par la relation : $V_n = P^{-1}U_n$. Écrire la relation de récurrence vérifiée par $(V_n)_{n\in\mathbb{N}}$.

6. Donner une formule **vectorielle explicite** pour la suite $(V_n)_{n\in\mathbb{N}}$.

7. En déduire une formule **vectorielle explicite** pour la suite $(U_n)_{n\in\mathbb{N}}$

8. En déduire la formule explicite suivante pour la suite de Fibonacci :

$$F_n = \frac{1}{\sqrt{5}}\varphi^n - \frac{1}{\sqrt{5}}(1-\varphi)^n$$

On pourra commencer par vérifier qu'elle fonctionne pour les premiers termes de la suite.

9. En déduire un équivalent asymptotique de la suite $(F_n)_{n\in\mathbb{N}}$ quand $n \to +\infty$. *Indication : on justifiera rigoureusement.*

La démonstration que vous venez d'effectuer cache en fait une théorie mathématique extrêmement utile : la **théorie des valeurs propres**, qui consiste à essayer de trouver un décomposition d'une matrice A en $A = PDP^{-1}$ où D est diagonale. Vous la découvrirez l'an prochain et l'utiliserez abondamment, aussi bien en informatique, en mathématiques qu'en physique.

A retenir : pour résoudre une récurrence sur une suite numérique d'ordre strictement supérieur à 1, je transforme le problème en une récurrence sur une suite vectorielle d'ordre 1... Cela marche aussi pour les équations différentielles !