DS1.

Important : Les programmes doivent être écrits en langage Ocaml. Si l'énoncé spécifie que des arguments de fonctions vérifient certaines propriétés, il n'est pas nécessaire de les vérifier dans votre code.

Vous ferez précéder ou suivre tous les codes de plus de 7 lignes d'une phrase expliquant l'idée que vous souhaitez implémenter.

1 Exercices

Exercice 1 Donner le type des fonctions suivantes ; dans le cas où le typage serait incohérent, justifier.

```
let f0 x y = 2*x -.y;;
let rec f1 h = match h with
|0 -> h
|_ -> f1 (h-1);;

let f2 x y = x y;;
let max3 x y z = max (max x y) z;;
```

Exercice 2 On considère la fonction suivante, appelée la fonction `zip` :

```
let rec mystere l1 l2 = match l1, l2 with
|[], [] -> []
|[], _  -> failwith "erreur"
|_, []  -> failwith "erreur"
|t::q, t1::q1 -> (t, t1)::(mystere q q1);;
```

1. Quel est le type de cette fonction ?
2. Avec une phrase, expliquer son rôle.
3. Écrire une fonction `unMystere` qui fait l'opération inverse.

Exercice 3 On considère la fonction suivante :

```
let rec f n = match n with
| n when n>100 -> n - 10
| _ -> f (f (n + 11));;
```

1. Quel est le cas de base de la récursivité ?
2. Pourquoi $f - 37$ renvoie t il une erreur ? Corriger la commande proposée.
3. Calculer la valeur de $f101$, $f100$, $f99$, $f98$, $f97$, $f96$, $f95$, $f94$, $f93$, $f92$, $f91$. Que conjecturez vous ?
4. Démontrer votre conjecture. On pourra par exemple écrire la division euclidienne de $101 - x$ par 11.

2 Problème

Dans ce problème, on ne se préoccupera pas d'un éventuel dépassement du plus grand entier représentable. On pourra utiliser librement les fonctions `List.lenght` et `List.rev` qui respectivement

calculent la longueur d'une liste et la renversent. Pour tout nombre réel positif x, on notera $\lfloor x \rfloor$ la partie entière par défaut de x et $\lceil x \rceil$ sa partie entière par excès.

On considère un ensemble U disposant d'une loi de composition interne associative notée $\times$ que nous appellerons «multiplication». Cette loi possède un élément neutre que nous noterons e. Par exemple, si U est l'ensemble des entier munis de la multiplication usuelle, l'élément neutre est 1. L'ensemble U peut aussi être celui des matrices carrées d'une même dimension d avec le produit matriciel comme multiplication. L'élément neutre est alors la matrice identité.

On définit l'exponentiation de la manière inductive suivante. Soit $a \in U, n \in \mathbb{N}$, on pose :

— $a^0 = e$,

— pour $n \geq 1$, $a^n = a^{n-1} \times a$.

La multiplication étant associative, pour tous entiers positifs i, j, si on pose $n = i + j$, on a alors $a^n = a^i \times a^j$. Un élément $a \in U$ et un entier $n \in \mathbb{N}^*$ étant donnés, on cherche à calculer a^n en minimisant le nombre de multiplications effectuées.

En effet, pour calculer a^{14} par exemple, on peut

1. multiplier 13 fois a par lui-même,
2. calculer $a^2 = a \times a$ (une seule multiplication) puis $a^3 = a^2 \times a$ puis $a^6 = a^3 \times a^3$ puis $a^7 = a^6 \times a$ puis enfin $a^{14} = a^7 \times a^7$. Au total on aura fait 5 multiplications.

Dans toute la suite, a et n désignent respectivement un élément quelconque de U et un élément de $\mathbb{N}^*$.

De manière plus formelle, on appelle *suite pour l'obtention de la puissance n* toute liste croissante non vide d'entiers naturels distincts $(n_0, \ldots, n_r)$ telle que :

— $n_0 = 1$,

— $n_r = n$,

— pour tout indice k vérifiant $1 \leq k \leq r$, il existe deux entiers i et j (éventuellement égaux) vérifiant $0 \leq i \leq k-1, 0 \leq j \leq k-1$ et $n_k = n_i + n_j$. Notez qu'il n'y a pas nécessairement unicité de la paire $\{i, j\}$.

Les deux exemples précédemment évoqués correspondaient aux suites pour l'obtention de la puissance 14 suivante :

1. $(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14)$ de longueur 14,
2. $(1, 2, 3, 6, 7, 14)$ de longueur 6.

On remarque que le nombre de multiplications correspondant à une suite pour l'obtention de la puissance n est égal à la longueur de la suite diminuée de 1.

1. Notons $(n_0, \ldots, n_r)$ une suite pour l'obtention de la puissance n de a. Montrer que $\forall k \in [\![0, r]\!], n_k \leq 2^k$.
2. En déduire que tout calcul de a^n n'utilisant que des multiplications nécessite un nombre de multiplications au moins égal à $\lceil \log_2(n) \rceil$.
3. Donner une famille de valeurs de n qui peuvent être calculées en effectuant exactement ce nombre de multiplications.

2.1 Algorithme par division

On considère un algorithme appelé *parDivision* [1] ayant pour objectif le calcul de a^n par une approche du type «diviser-pour-régner». Sa description récursive succinte est donnée ci-dessous :

— Si $n = 1$ alors $a^n = a$,

— — On calcule la partie entière par défaut, $q = \lfloor \frac{n}{2} \rfloor$,

— On calcule récursivement $b = a^q$,

1. Aussi appelé «algorithme d'exponentiation rapide».

— Si n est pair alors $a^n = b \times b$, sinon $a^n = (b \times b) \times a$.

4. Proposer une fonction `calculParDivision : int -> int -> int` qui implémente le calcul de a^n par l'algorithme décrit ci-dessus. Votre fonction devra ne faire apparaître *qu'un seul* appel récursif.
5. Pour un entier n, combien d'appels récursifs sont ils nécessaires pour atteindre le cas de base ?
6. Montrer que l'algorithme `calculParDivision` effectue au plus $2\lfloor \log_2(n) \rfloor$ multiplications pour le calcul de a^n.
7. Donner sans justifier une famille (infinie) de nombres n nécessitant ce nombre de multiplications pour le calcul de a^n.
8. Donner sans justifier la suite des puissances calculées pour obtenir :
   (a) a^{14} :
   (b) a^{19}, :
   (c) a^{125}. :
9. Ecrire en Ocaml une fonction `parDivision : int -> int list` qui calcule la suite de la puissance n correspondant à l'algorithme `parDivision`. Par exemple : `parDivision 3` renverra la liste `[1,2,3]`.

2.2 Algorithme par décomposition binaire

Nous allons nous intéresser maintenant à un autre algorithme, dit *par décomposition binaire*. On rappelle tout d'abord que tout entier naturel non nul peut se décomposer de manière unique en binaire (base 2), c'est à dire que

$$\forall n \geq 1, \exists k > 0, a_0, a_1, \ldots, a_k \in \{0,1\}, a_k \neq 0, n = \sum_{p=0}^{k} a_p 2^p.$$

Nous allons commencer par donner un pseudo-code vague pour décrire la méthode de décomposition binaire puis nous allons la préciser grâce à des exemples. L'idée de l'algorithme est la suivante pour calculer a^n :

— on décompose n en base 2, $n = \sum_{p=0}^{k} a_p 2^p$.

— on calcule la valeur cible de a^n en effectuant les produits correspondant aux sommes partielles de la somme ci-dessus.

Donnons deux exemples :

— Soit $n = 14$, on écrit $14 = 2 + 2^2 + 2^3 = 2 + 4 + 8$. On a donc $a^14 = (a^2 \times a^4) \times a^8$, ce qui conduit donc à calculer les puissances de a d'exposants $2, 4, 8$ mais également 6 et 14 (à cause des produits intermédiaires). La suite pour l'obtention de la puissance 14 est donc $(1, 2, 4, 6, 8, 14)$.

— Soit $n = 18$. On a $18 = 2^1 + 2^4 = 2 + 16$. L'algorithme consiste à calculer les puissances d'exposants $2, 4, 8, 16$ puis 18. La suite correspondant est donc $(1, 2, 4, 8, 16, 18)$.

— Soit $n = 101 = 1 + 2^2 + 2^5 + 2^6 = 1 + 4 + 32 + 64$. Donc $a^{101} = \left((a \times a^4) \times a^{32}\right) \times a^{64}$. On calcule les puissances $2, 4, 5$ (pour $(a \times a^4)$), $8, 16, 32, 37$ (pour $a^5 \times a^{32}$), $64, 101$ ($a^{37} \times a^{64}$).

1. Donner sans justifier la suite correspondant à l'algorithme *par décomposition binaire* :
   (a) pour l'obtention de la puissance 15,
   (b) pour l'obtention de la puissance 16,
   (c) pour l'obtention de la puissance 125.

On appelle *écriture binaire inverse* d'un entier n la suite $(a_0, \ldots, a_k)$ des termes de la décompositions binaires (on met le bit de poids faible en premier). Par exemple $14 = 2 + 2^2 + 2^3$ a pour écriture binaire inverse $(0, 1, 1, 1)$.

1. Ecrire en CAML une fonction `binaireInverse : int -> int list` qui à un entier $n \geq 1$ donné associe une liste correspondant à son écriture binaire inverse.

2. Ecrire en CAML la fonction `parDecompositionBinaire : int -> int list` qui à un entier $n \geq 1$ associe une liste correspondant à l'algorithme par décomposition binaire.
3. Donner une suite de longueur 6 pour l'obtention de la puissance 15. Qu'en déduire quant aux algorithmes des deux parties précédentes ?