Corrigé : structure de corde (X 2008)

Partie I. Préliminaires sur les mots

Pas de difficulté notable dans cette partie, les hypothèses faites permettent d'utiliser les fonctions `hd` et `tl` (tête et queue d'une liste) sans qu'il soit nécessaire de distinguer le cas de la liste vide.

Question 1.

```
let rec longueurMot = function
  | []   -> 0
  | _::q -> 1 + longueurMot q ;;
```

Question 2.

```
let rec iemeCar i = function
  | x when i = 0 -> hd x
  | x            -> iemeCar (i-1) (tl x) ;;
```

Question 3.

```
let rec prefixe k = function
  | _ when k = 0 -> []
  | x            -> (hd x)::(prefixe (k-1) (tl x)) ;;
```

Question 4.

```
let rec suffixe k = function
  | x when k = 0 -> x
  | x            -> suffixe (k-1) (tl x) ;;
```

Partie II. Opérations sur les cordes

Question 5. L'invariant exigé permet de calculer la longueur d'un mot en coût constant.

```
let rec longueur = function
  | Vide              -> 0
  | Feuille (n, x)    -> n
  | Noeud (n, x1, x2) -> n ;;
```

Question 6. Pour respecter l'invariant exigé, il faut prendre garde à distinguer le cas du mot vide.

```
let nouvelleCorde = function
  | [] -> Vide
  | x  -> Feuille (longueurMot x, x) ;;
```

Question 7. Là encore, il faut penser à distinguer le cas où l'une des deux cordes est vide pour respecter l'invariant.

```
let concat = fun
  | Vide c -> c
  | c Vide -> c
  | c1 c2  -> Noeud (longueur c1 + longueur c2, c1, c2) ;;
```

Question 8.

```
let rec caractere i = function
  | Vide                                -> failwith "caractere"
  | Feuille (n, x)                      -> iemeCar i x
  | Noeud (n, c1, c2) when i < longueur c1 -> caractere i c1
  | Noeud (n, c1, c2)                   -> caractere (i - longueur c1) c2 ;;
```

Question 9. Dans le cas où la corde n'est pas une feuille, il faut distinguer trois cas : le facteur recherché peut être facteur du fils gauche, facteur du fils droit ou le résultat de la concaténation d'un suffixe du fils gauche et d'un préfixe du fils droit.

```
let rec sousCorde i m = function
  | Vide                                  -> failwith "sousCorde"
  | Feuille (n, x)                        -> Feuille (m, prefixe m (suffixe i x))
  | Noeud (n, c1, c2) when i+m <= longueur c1 -> sousCorde i m c1
  | Noeud (n, c1, c2) when i >= longueur c1   -> sousCorde (i-longueur c1) m c2
  | Noeud (n, c1, c2)                     -> Noeud (m,
                                               sousCorde i (longueur c1-i) c1,
                                               sousCorde 0 (m+i-longueur c1) c2) ;;
```

Partie III. Équilibrage

Question 10. Les premiers termes de la suite de Fibonacci sont : $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$; commençons par préciser les contraintes que doivent respecter les cordes non vides pour appartenir à une des cases du tableau **file** :

/	/	hauteur = 0 longueur = 1	hauteur ⩽ 1 longueur = 2	hauteur ⩽ 2 longueur ∈ {3, 4}	hauteur ⩽ 3 longueur ∈ {5, 7}	hauteur ⩽ 4 longueur ∈ {8, 12}	hauteur ⩽ 5 longueur ∈ {13, 20}	⋯

Introduisons maintenant une par une les feuilles de l'arbre donné en exemple :

- Introduction de **Ec** :

/	/	Vide	Ec

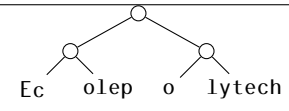
- Introduction de **olep** :

/	/	Vide	Vide	Vide	

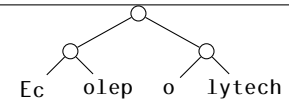
- Introduction de **o** :

/	/	o	Vide	Vide	

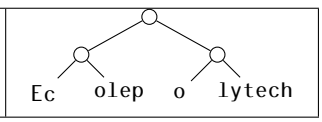
- Introduction de **lytech** :

/	/	Vide	Vide	Vide	Vide	Vide	

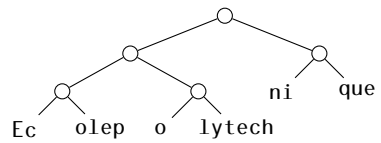
- Introduction de **ni** :

/	/	Vide	ni	Vide	Vide	Vide	

- Introduction de **que** :

/	/	Vide	Vide	Vide		Vide	

Il reste alors à concaténer les différentes cordes de ce tableau pour obtenir la corde suivante :



On peut observer que la corde initiale a un coût égal à 58 alors que la corde rééquilibrée obtenue ci dessus n'a plus qu'un coût égal à 49.

Question 11.

```
let initialiserFib () =
  fib.(1) <- 1 ;
  for k = 2 to tailleMax do fib.(k) <- fib.(k-1) + fib.(k-2) done ;;
```

Question 12. On suppose pour simplifier que l'insertion de la corde c dans le tableau à partir de la case d'indice i est possible, c'est à dire qu'elle ne provoque pas de débordement du tableau **file**.

```
let rec inserer c i =
  let c' = concat file.(i) c in
  match longueur c' < fib.(i+1) with
    | true  -> file.(i) <- c'
    | false -> file.(i) <- Vide ; inserer c' (i+1) ;;
```

Question 13. Notons n la longueur de c et p sa hauteur. Par hypothèse, $p \leqslant i-2$ et $n \geqslant \mathrm{F}_i$.
Soit c_i la corde contenue dans la case d'indice i du tableau **file**, et distinguons deux cas :

- Si c_i est vide, deux sous-cas sont possibles :
  - si $n < \mathrm{F}_{i+1}$ alors c prend la place de c_i et l'invariant exigé pour le tableau **file** reste bien respecté ;
  - sinon, on a $p \leqslant (i+1)-2$ et $n \geqslant \mathrm{F}_{i+1}$ et la fonction **inserer** cherche à insérer c dans la case d'indice $i+1$, avec les mêmes hypothèses au rang $i+1$.
- si c_i n'est pas vide, notons n_i et p_i sa longueur et sa hauteur.
  D'après l'invariant respecté par le tableau, $\mathrm{F}_i \leqslant n_i < \mathrm{F}_{i+1}$ et $p_i \leqslant i-2$. Considérons alors le résultat c' de la concaténation de c_i et de c, et notons n' et p' sa longueur et sa hauteur. Nous avons : $n' = n_i + n$ et $p' = 1 + \max(p_i, p)$.
  On a $n_i \geqslant \mathrm{F}_i$ donc $n' \geqslant 2\mathrm{F}_i \geqslant \mathrm{F}_{i+1}$, et $p' \leqslant i-1 = (i+1)-2$ donc la fonction **inserer** vide la case d'indice i et cherche à insérer c' dans la case d'indice $i+1$, avec les mêmes hypothèses au rang $i+1$.

Si la longueur du tableau **file** est suffisante, la fonction **inserer** va donc bien insérer la corde c en respectant l'invariant de l'énoncé.

Question 14.

```
let equilibrer c =
  let rec aux1 = function
    | Vide                -> ()
    | Feuille (n, x) as f -> inserer f 2
    | Noeud (n, c1, c2)   -> aux1 c1 ; aux1 c2
  and aux2 = function
    | 1 -> Vide
    | i -> concat file.(i) (aux2 (i-1))
  in aux1 c ; aux2 (tailleMax-1) ;;
```

La fonction **aux1** réalise l'insertion de chacune des feuilles de c dans le tableau **file** ; la fonction **aux2** réalise la concaténation des cordes contenues dans le tableau.

Question 15. Notons p le plus grand des indices des cases du tableau **file** qui ne contiennent pas de corde vide ; pour tout $i \in [\![2, p]\!]$, on note c_i la corde se trouvant dans la case d'indice i, et h_i la hauteur de la corde résultant de la concaténation (par la gauche) des cordes $c_2, c_3, \ldots, c_i$.
Montrons par récurrence sur i que $h_i \leqslant i-1$.

- Si $i = 2$, par hypothèse la hauteur de c_2 est inférieure ou égale à 0 ; *a fortiori* $h_2 \leqslant 1$.
- Si $i > 2$, supposons $h_{i-1} \leqslant i-2$. Puisque c_i a par hypothèse une hauteur inférieure ou égale à $i-2$, la dernière concaténation a lieu entre deux arbres de hauteurs inférieures ou égales à $i-2$, et donc $h_i \leqslant i-1$.

De ceci il résulte que $h \leqslant p-1$, et donc que $\mathrm{F}_{h+1} \leqslant \mathrm{F}_p$. Or c_p est une corde de longueur $n_p \geqslant \mathrm{F}_p$ donc $n \geqslant n_p \geqslant \mathrm{F}_p$, ce qui prouve que $\mathrm{F}_{h+1} \leqslant n$.

Le coût de c est à l'évidence majorable par hn (il suffit de majorer la profondeur de chaque feuille par la hauteur h). Or $n \geqslant \mathrm{F}_{h+1} \geqslant \dfrac{\phi^h}{\sqrt{5}}$, donc $h \leqslant \log_\phi(n\sqrt{5})$, ce qui prouve que le coût de c est majoré par $n\Big(\log_\phi(n) + \mathrm{K}\Big)$ avec $\mathrm{K} = \dfrac{1}{2}\log_\phi(5)$.

Partie IV. Équilibrage optimal

Question 16.

```
let initialiserQ c =
  let rec aux k = function
    | Vide                -> k
    | Feuille (n, x) as f -> q.(k) <- f ; k+1
    | Noeud(n, c1, c2)    -> aux (aux k c1) c2
  in aux 0 c - 1 ;;
```

L'argument **k** de la fonction **aux** désigne le premier indice disponible dans le tableau **q**.

Question 17. On commence par écrire une fonction qui cherche l'indice d'une feuille dans le tableau **q** :

```
let cherche_feuille f =
  let rec aux = function
    | i when q.(i) = f -> i
    | i                -> aux (i+1)
  in aux 0 ;;
```

On parcourt ensuite la corde **c1** à la recherche de ses feuilles, le paramètre **p** de la fonction **aux** calculant la profondeur de celles-ci :

```
let initialiserProf c1 =
  let rec aux p = function
    | Vide                -> ()
    | Feuille (n, x) as f -> let i = cherche_feuille f in prof.(i) <- p
    | Noeud (n, c1, c2)   -> aux (p+1) c1 ; aux (p+2) c2
  in aux 0 c1 ;;
```

Question 18.

```
let reconstruire () =
  let rec aux i = function
    | p when p = prof.(i) -> i+1, q.(i)
    | p -> let (j, c1) = aux i (p+1) in
           let (k, c2) = aux j (p+1) in
           k, (concat c1 c2)
  in snd (aux 0 0) ;;
```

La fonction **aux** prend deux paramètres : le premier (**i**) désigne l'indice de la prochaine feuille à insérer ; le second (**p**) la profondeur où on cherche à insérer la feuille. Cette même fonction retourne deux paramètres : l'indice de la feuille suivante à insérer et la corde obtenue.

♦ ♦
♦