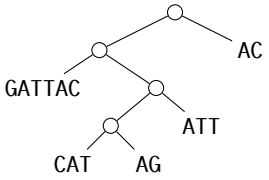
# Structure de corde (X 2008)

Durée : libre

**Introduction** La majorité des langages de programmation fournissent une notion primitive de chaînes de caractères. Si ces chaînes s'avèrent adaptées à la manipulation des mots ou des textes relativement courts, elles deviennent généralement inutilisables sur de très grands textes. L'une des raisons de cette inefficacité est la duplication d'un trop grand nombre de caractères lors des opérations de concaténation ou d'extraction d'une sous-chaîne. Or il existe des domaines où la manipulation efficace de grandes chaînes de caractères est essentielle (représentation du génome en bio-informatique, éditeurs de texte, etc.). Ce problème aborde une alternative à la notion usuelle de chaîne de caractères connue sous le nom de *corde*. Une corde est tout simplement un arbre binaire dont les feuilles sont des chaînes de caractères usuelles et dont les nœuds internes représentent des concaténations. Ainsi la corde :




représente le mot GATTACCATAGATTAC, obtenu par concaténation des cinq mots GATTAC, CAT, AG, ATT, et AC. L'intérêt des cordes est d'offrir une concaténation immédiate et un partage possible de caractères entre plusieurs chaînes, au prix d'un accès aux caractères un peu plus coûteux.

La partie I traite des fonctions préliminaires sur les mots ; la partie II définit les principales opérations sur les cordes. Enfin, les parties III et IV étudient le problème de l'équilibrage des cordes selon deux algorithmes différents dont le second, l'algorithme de Garsia-Wachs, est optimal.

## Partie I. Préliminaires sur les mots

Le mot $x = a_0, a_1, \ldots, a_{n-1}$, de longueur $n$, est représenté dans ce problème par la liste des entiers codant ses caractères. Ainsi, le mot ATT est représenté par la liste $\langle 65, 84, 84\rangle$. Pour ne pas dépendre du codage des caractères, on identifie les caractères et leurs codes. Le type des mots est donc défini par :

```
type mot == int list ;;
```

**Question 1.** Écrire la fonction **longueurMot** qui calcule la longueur d'un mot.

```
longueurMot : mot -> int
```

**Question 2.** Ecrire la fonction **iemeCar** qui prend en arguments un entier $i$ et un mot $x = a_0 a_1 \cdots a_{n-1}$ et qui renvoie le caractère $a_i$. On supposera $0 \leqslant i < n$.

```
iemeCar : int -> mot -> int
```

**Question 3.** Ecrire la fonction **prefixe** qui prend en arguments un entier $k$ et un mot $x = a_0 a_1 \cdots a_{n-1}$ et qui renvoie le mot $a_0 a_1 \cdots a_{k-1}$, c'est-à-dire le mot constitué des $k$ premiers caractères de $x$. On supposera $0 \leqslant k \leqslant n$.

```
prefixe int -> mot -> mot
```

**Question 4.** Ecrire la fonction **suffixe** qui prend en arguments un entier $k$ et un mot $x = a_0 a_1 \cdots a_{n-1}$ et qui renvoie le mot $a_k a_{k+1} \cdots a_{n-1}$, c'est-à-dire le mot obtenu en supprimant les $k$ premiers caractères de $x$. On supposera $0 \leqslant k \leqslant n$.

```
suffixe int -> mot -> mot
```

# Partie II. Opérations sur les cordes

Comme expliqué dans l'introduction, une *corde* est un arbre binaire dont les feuilles sont des mots. Plus précisément, une corde est soit vide, soit constituée d'un unique mot (une feuille), soit un nœud constitué de deux cordes et représentant leur concaténation. Pour des raisons d'efficacité, on conserve dans les feuilles aussi bien que dans les nœuds la longueur de la corde correspondante.
On définit donc le type *corde* suivant :

```
type corde = Vide | Feuille of int * mot | Noeud of int * corde * corde ;;
```

Dans la suite, on garantira l'invariant suivant sur les cordes :

- dans une corde de la forme **Feuille**$(n, x)$ on a $n =$ **longueurMot**$(x)$ et $n > 0$ ;
- dans une corde de la forme **Noeud**$(n, c_1, c_2)$ on a $c_1 \neq$ **Vide**, $c_2 \neq$ **Vide** et $n$ est la longueur totale de la corde, c'est à dire la somme des longueurs de $c_1$ et $c_2$.

On notera en particulier que la corde de longueur 0 est nécessairement représentée par **Vide**.

**Question 5.** Écrire la fonction **longueur** qui renvoie la longueur d'une corde.

```
longueur : corde -> int
```

**Question 6.** Écrire la fonction **nouvelleCorde** qui construit une corde à partie d'un mot.

```
nouvelleCorde : mot -> corde
```

**Question 7.** Écrire la fonction **concat** qui construit la concaténation de deux cordes.

```
concat : corde -> corde -> corde
```

**Question 8.** Écrire la fonction **caractere** qui prend en arguments un entier $i$ et une corde $c$ représentant le mot $a_0 a_1 \cdots a_{n-1}$ et qui renvoie le caractère $a_i$. On supposera $0 \leqslant i < n$.

```
caractere : int -> corde -> int
```

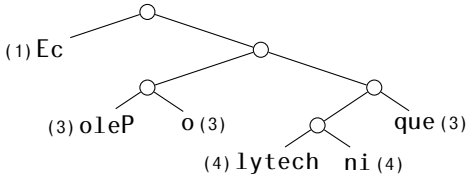
**Question 9.** Écrire la fonction **sousCorde** qui prend en arguments un entier $i$, un entier $m$ et une corde $c$ représentant le mot $a_0 a_1 \cdots a_{n-1}$ et qui renvoie une corde représentant le mot $a_i a_{i+1} \cdots a_{i+m-1}$, c'est-à-dire la sous-corde de $c$ débutant au caractère $i$ et de longueur $m$. On supposera $0 \leqslant i < i + m \leqslant n$.

```
sousCorde : int -> int -> corde -> corde
```

On s'attachera à réutiliser dans la corde résultat autant de sous-arbres de la corde $c$ que possible.

# Partie III. Équilibrage

Le hasard des concaténations peut amener une corde à se retrouver déséquilibrée, c'est-à-dire à avoir certaines de ses feuilles très éloignées de la racine et donc d'accès plus coûteux. Le but de cette partie est d'étudier une stratégie d'équilibrage *a posteriori*.
Considérons une corde $c$ composée de $k+1$ feuilles, et donc de $k$ nœuds internes. Notons ces $k+1$ feuilles $x_0, x_1, \ldots, x_k$ lorsqu'on les considère de la gauche vers la droite, si bien que $c$ représente le mot $x_0 x_1 \cdots x_k$. La profondeur de la feuille $x_i$ est notée **prof**$(x_i)$ et est définie comme la distance de $x_i$ à la racine de $c$. Voici un exemple de corde pour $k = 5$ où la profondeur de chaque feuille est indiquée entre parenthèses :



Le coût de l'accès à un caractère de la feuille $x_i$ est défini comme la profondeur de cette feuille dans $c$, soit $\texttt{prof}(x_i)$ (on ne considère donc pas le coût de l'accès dans le mot $x_i$ lui-même). Le coût total d'une corde est alors défini comme la somme des coûts d'accès à tous ses caractères, et vaut donc :

$$\mathit{Coût}(c) = \sum_{i=0}^{k} \texttt{longueurMot}(x_i) \times \texttt{prof}(x_i).$$

Un rééquilibrage consiste à construire une corde différente, dont les feuilles sont $x_0, x_1, \ldots, x_k$ dans le même ordre (le mot représenté ne doit pas changer) et dont le coût est, éventuellement, meilleur.

L'algorithme proposé est le suivant.

On considère un tableau **file** de cordes dans lequel les feuilles de $c$ vont être successivement insérées dans le sens des indices croissants. Les cases d'indices 0 et 1 ne sont pas utilisées. La case d'indice $i$ contient soit la corde vide (**Vide**), soit une corde de hauteur inférieure ou égale à $i-2$ et dont la longueur est comprise dans l'intervalle $[\mathrm{F}_i, \mathrm{F}_{i+1}[$ où $\mathrm{F}_i$ désigne le $i$-ème terme de la suite de Fibonacci. La hauteur d'une corde $c$, notée $\texttt{hauteur}(c)$, est la profondeur maximale de ses feuilles, c'est-à-dire :

$$\texttt{hauteur}(\texttt{Vide}) = 0$$
$$\texttt{hauteur}\big(\texttt{Feuille}(n,x)\big) = 0$$
$$\texttt{hauteur}\big(\texttt{Noeud}(n,c_1,c_2)\big) = 1 + \max\big(\texttt{hauteur}(c_1), \texttt{hauteur}(c_2)\big)$$

Pour équilibrer la corde $c$ dont les feuilles sont les mots $x_0, x_1, \ldots, x_k$, dans cet ordre, on procède ainsi :

1. On insère successivement chaque feuille $x_j$ dans le tableau **file** à partir de la case 2. L'insertion d'une feuille, et plus généralement d'une corde, à partir de la case d'indice $i$ se fait ainsi :
   (a) La corde à insérer est concaténée à droite de la corde se trouvant dans la case $i$ ; soit $c'$ le résultat. Si la longueur de $c'$ est comprise dans l'intervalle $[\mathrm{F}_i, \mathrm{F}_{i+1}[$ on affecte $c'$ à la case $i$ et on a terminé l'insertion de cette corde.
   (b) Sinon, on affecte **Vide** à la case $i$ et on retourne à l'étape (a) pour effectuer l'insertion de $c'$ à partir de la case d'indice $i+1$.

   On garantit l'invariant suivant : après l'insertion de la feuille $x_j$, la concaténation successive de toutes les cordes contenues dans les cases de **file**, considérées dans le sens des indices décroissants, est égale à une corde représentant le mot $x_0 x_1 \cdots x_j$.
2. Le résultat est alors la corde obtenue en réalisant la concaténation par la gauche de toutes les cordes de **file**, en procédant par indice croissant.

**Question 10.** Calculer le résultat de cet algorithme sur la corde de l'exemple précédent.

**Question 11.** On rappelle que la suite de Fibonacci $(\mathrm{F}_n)_{n\in\mathbb{N}}$ est définie par :

$$\mathrm{F}_0 = 0, \qquad \mathrm{F}_1 = 1, \qquad \forall n \geqslant 0,\ \mathrm{F}_{n+2} = \mathrm{F}_{n+1} + \mathrm{F}_n.$$

Afin d'éviter tout débordement arithmétique en calculant $\mathrm{F}_n$, on limite la taille de **file** à 44 cases indexées de 0 à 43 (les cases 0 et 1 n'étant pas utilisées). On introduit la constante $\texttt{tailleMax} = 44$ et on calcule les valeurs de $\mathrm{F}_n$ pour $0 \leqslant n \leqslant \texttt{tailleMax}$ une fois pour toute dans un tableau **fib** ainsi déclaré :

```
let tailleMax = 44 ;;
let fib = make_vect (tailleMax+1) 0 ;;
```

Écrire la fonction **initialiserFib** qui initialise le tableau **fib**.

**Question 12.** Le tableau **file** utilisé par l'algorithme est déclaré comme un tableau global contenant des cordes :

```
let file = make_vect tailleMax Vide ;;
```

Écrire la fonction **inserer** qui prend en arguments une corde $c$ et un entier $i$ tels que $2 \leqslant i < \texttt{tailleMax}$, $\texttt{hauteur}(c) \leqslant i-2$ et $\texttt{longueur}(c) \geqslant \mathrm{F}_i$ et réalise l'insertion de $c$ dans le tableau **file** à partir de la case d'indice $i$.

```
inserer : corde -> int -> unit
```

**Question 13.** Montrer que l'invariant $\texttt{hauteur}(c_i) \leqslant i-2$ et $\texttt{longueur}(c_i) \geqslant \mathrm{F}_i$ est préservé par cette fonction pour toutes les valeurs $c_i$ non vides des cases du tableau **file** ($2 \leqslant i < \texttt{tailleMax}$).

**Question 14.** Écrire la fonction **`equilibrer`** qui réalise l'équilibrage d'une corde par l'algorithme ci-dessus.

```
equilibrer : corde -> corde
```

**Question 15.** Soit $c$ une corde non vide renvoyée par la fonction **`equilibrer`** ci-dessus. Soit $n$ sa longueur et $h$ sa hauteur. Montrer que l'on a : $n \geqslant F_{h+1}$.
En déduire qu'il existe une constante K (indépendante de $n$) telle que :

$$Coût(c) \leqslant n\Big(\log_\phi(n) + K\Big)$$

où $\phi$ est le nombre d'or $\dfrac{1+\sqrt{5}}{2}$ et $\log_\phi$ désigne le logarithme à base $\phi$. On admettra que l'on a $F_{i+1} \geqslant \dfrac{\phi^i}{\sqrt{5}}$ pour tout $i \geqslant 0$.

# Partie IV. Équilibrage optimal

Bien que satisfaisant en pratique, l'équilibrage étudié dans la partie précédente n'est pas optimal. Le but de cette partie est d'étudier une stratégie optimale de rééquilibrage (algorithme de Garsia-Wachs). Les notations sont celles de la partie III. L'algorithme proposé procède en deux temps : il commence par construire une corde de coût minimal ayant les mêmes feuilles que $c$ mais pas nécessairement dans le même ordre ; puis dans un deuxième temps il transforme cette corde en une autre de même coût où les feuilles sont maintenant dans le même ordre que dans $c$.
La première partie de l'algorithme opère sur une liste de cordes $q = \langle q_0, q_1, \ldots, q_m \rangle$ et procède de la manière suivante :

1. Initialement la liste $q$ est la liste $\langle x_0, x_1, \ldots, x_k \rangle$ des $k+1$ feuilles de $c$.
2. Tant que la liste $q$ contient au moins deux éléments, on effectue l'opération suivante :
   (a) Déterminer le plus petit indice $i$ tel que **`longueur`**$(q_{i-1}) \leqslant$ **`longueur`**$(q_{i+1})$ le cas échéant, et poser $i = m$ sinon.
   (b) Ôter $q_{i-1}$ et $q_i$ de la liste $q$ et former leur concaténation ; soit $c'$ la corde obtenue.
   (c) Déterminer le plus grand indice $j < i$ tel que **`longueur`**$(q_{j-1}) \leqslant$ **`longueur`**$(c')$ le cas échéant, et poser $j = 0$ sinon.
   (d) Insérer $c'$ dans la liste $q$ juste après $q_{j-1}$ (et donc au début de la liste $q$ si $j = 0$).
3. Le résultat est l'unique élément restant dans la liste $q$.

Il est clair que le résultat de cet algorithme est une corde ayant les mêmes feuilles que $c$ mais que ces feuilles ne sont pas nécessairement dans le bon ordre. On admettra le résultat suivant : *l'arbre obtenu est de coût minimal.*

**Question 16.** Pour simplifier le codage, on suppose que le nombre $k$ de feuilles est inférieur à une certaine valeur (ici **`maxf`** = 1000) et que la liste $q$ est représentée dans un tableau global $q$ :

```
let maxf = 1000 ;;
let q = make_vect maxf Vide ;;
```

Écrire la fonction **`initialiserQ`** qui prend en argument une corde $c$, remplit les $k+1$ premiers éléments de $q$ avec les feuilles $x_0, x_1, \ldots, x_k$ de $c$ (c'est à dire des cordes de la forme **`Feuille`**), et renvoie la valeur de $k$. On supposera $c \neq$ **`Vide`**.

```
initialiserQ : corde -> int
```

On admettra avoir effectué le reste de l'algorithme ci-dessus et avoir donc écrit une fonction **`phase1`** qui prend en argument une corde $c$ et renvoie la corde obtenue par l'algorithme ci-dessus.

```
phase1 : corde -> corde
```

La deuxième étape de l'algorithme procède ainsi. Soit $c_1$ la corde obtenue à l'issue de la première étape de l'algorithme. Chaque feuille $x_i$ de $c$ se trouve dans $c_1$ à une certaine profondeur ; notons $p_i$ cette profondeur. On admet la propriété : il existe une corde $c_2$ dont les feuilles sont exactement $x_0, x_1, \ldots, x_k$ dans cet ordre et où la profondeur de chaque $x_i$ est exactement $p_i$. On peut alors construire $c_2$ en ne connaissant que les $p_i$, et $c_2$ est dès lors un rééquilibrage optimal de $c$.

**Question 17.** Les profondeurs $p_i$ seront stockées dans un tableau global **`prof`** :

```
let prof = make_vect maxf 0 ;;
```

Écrire la fonction **`initialiserProf`** qui prend en argument la corde $c_1$ et range dans le tableau **`prof`**, à l'indice $i$, la profondeur de la feuille $x_i$ (de $c$) dans $c_1$ pour $0 \leqslant i \leqslant k$. On pourra avantageusement réutiliser le tableau **`q`** qu'on supposera construit.

```
initialiserProf : corde -> unit
```

**Indication** : on admettra que pour comparer les feuilles de $c$ et de $c_1$ on peut utiliser l'égalité fournie par le langage, *i.e.* le symbole `=`.

**Question 18.** Écrire la fonction **`reconstruire`** qui construit $c_2$ à partir de la seule donnée des tableaux **`q`** et **`prof`**. Attention : pour des profondeurs $p_i$ quelconques, il n'existe pas nécessairement de corde où chaque $x_i$ a la profondeur $p_i$. On demande ici un algorithme qui fonctionne uniquement sous l'hypothèse qu'une telle corde existe (ce qui est le cas ici).

```
reconstruire : unit -> corde
```

♦ ♦
♦