TP n° 3 – Structures de données récursives

vie@jill-jenn.net/tp
thomas.h@ens.fr

1 Oups !

Aujourd'hui, Yorel Reivax fait joujou avec des listes.

```
let rec l = 1::l;;
it_list prefix + 0 l;;
```

Qu'a-t-il cherché à faire et pourquoi est-ce somme toute une mauvaise idée ?

2 Codage de Huffman

Tout néophyte souhaitant implémenter le codage de Huffman, utile en compression de données, doit respecter les deux commandements listés ci-dessous.

1. Les lettres les plus fréquentes avec le moins de bits tu encoderas [1].
2. À ce qu'aucun mot soit le préfixe d'un autre tu veilleras.

Question 1. Pourquoi de tels commandements ? Expliquez.

Il faut d'abord compter les fréquences de chaque lettre du message à encoder.

Question 2. Écrire une fonction `nb_occurrences` telle que `nb_occurrences phrase` retourne un `int vect` de taille 256 dont la i^{e} case contient le nombre d'occurrences du i^{e} caractère [2] dans `phrase`.

Nous allons construire un arbre dont les feuilles seront les caractères de `phrase`. Les lettres les plus fréquentes seront les moins profondes.

```
type 'a arbre = Feuille of 'a | Noeud of 'a arbre * 'a arbre;;
```

Question 3. Écrire la fonction `merge_sort` qui prend en argument une fonction `compare : 'a -> 'a -> bool` telle que `compare a b` doit retourner `true` si vous souhaitez que `a` se trouve avant `b` après le tri, et une liste `l`.

Question 4. Écrire la fonction `list_of_occ` qui prend en argument le résultat de `nb_occurrences phrase` et retourne une liste de couples `(Feuille c, p)` où `c` est un caractère de `phrase` et le poids `p` est son nombre d'occurrences [3].

Question 5. Utiliser la fonction `merge_sort` pour trier la liste de couples par poids croissant.

1. Visiblement, si Huffman savait compresser des mots, il avait en revanche du mal à les ordonner correctement.
2. Le numéro du caractère peut être obtenu avec la fonction `int_of_char`. L'opération inverse peut être réalisée avec la fonction `char_of_int`.
3. Il est bien entendu interdit d'utiliser une boucle `for`, des références ou toute autre forme de programmation impérative.

L'algorithme permettant de construire l'arbre de Huffman consiste à prendre à chaque tour les deux pemiers arbres `a1` et `a2` de la liste de poids minimaux respectivement `p1` et `p2` et à les fusionner en un nouvel arbre de poids `p1 + p2`, qui est ensuite inséré dans la liste à la place de `a1` et `a2` tout en conservant l'ordre croissant des poids tout le long du processus, à la manière d'un tri par insertion. L'algorithme s'arrête lorsqu'il ne reste plus qu'un arbre dans la liste : l'arbre de Huffman.

Question 6. Écrivez la fonction `arbre_huffman`, qui construit l'arbre de Huffman. Elle comportera une fonction auxiliaire récursive `insert`, qui insère un arbre et son poids dans une liste triée de couples `(arbre, poids)`.

Question 7. Arrêtez-vous un instant pour vous rendre compte que c'est quand même hyper bien foutu comme truc, puis tournez-vous vers votre examinateur de TP et témoignez-lui votre gratitude en hochant la tête avec un léger sourire.

Question 8. À présent que l'arbre est planté, écrivez une fonction `table_huffman` qui le prend en argument et retourne une `(char * string) list` qui à chaque lettre x associe son code de Huffman, défini comme suit : soit $x_1 \dots x_n \in \{\text{gauche}, \text{droite}\}^n$ le parcours suivi pour aller de la racine de l'arbre à la feuille associée à x ; le code de Huffman de x est le mot $y_1 \dots y_n \in \{0, 1\}^n$ tel que y_i vaut 0 si x_i vaut gauche, 1 sinon.

Question 9. Écrivez une fonction `encode` qui prend en argument une `phrase` et une `table` de Huffman et retourne le message codé. La fonction `assoc : 'a -> ('a * 'b) list -> 'b` vous sera certainement utile pour cela.

Question 10. Écrivez une fonction `decode` qui prend en argument un `message` codé et un `arbre` de Huffman et retourne la phrase en clair.

Question 11. Trouvez un algorithme qui construit l'arbre de Huffman en temps linéaire, une fois que la liste a été initialisée aux feuilles.

Question 12. Implémenter cet algorithme.

Dans un souci de respecter l'environnement (et de faire une mise en abyme), ce TP a été compressé. S'il vous plaît, n'imprimez pas ce TP.