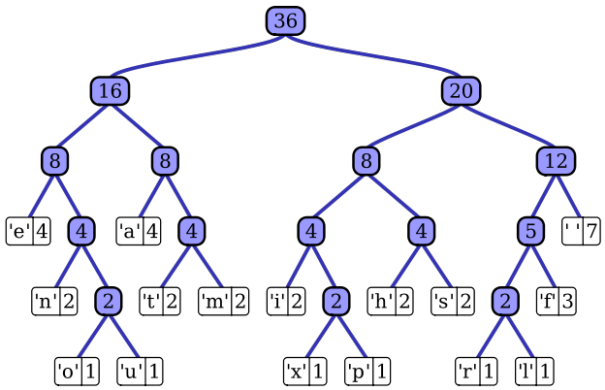
TD 3 - Le codage de Huffman

Retrouvez tous les énoncés et les corrections des TP sur ma page personnelle :

`http://perso.ens-lyon.fr/hadrien.croubois/`

Le codage de Huffman

Le codage de Huffman est un algorithme de compression de données sans perte publié en 1952. Contrairement a certains algorithme de compression telle que `JPEG`, `MPEG` ou `MP3`, il n'exploite pas les caractéristiques du fichier a traiter et permet donc de compresser tout type de donnée. Il est notamment utilisé (en complement d'autres algorithmes) pour la création d'archives, notamment celle au format `zip`, `gzip` ou `LZH`.



Exemple d'arbre de Huffman, généré avec la phrase "this is an example of a huffman tree"

Principe du codage de Huffman

L'objectif de l'algorithme de Huffman est de minimiser le coup du codages des éléments les plus présents au détriment des plus rares. Cela est rendu possible par l'utilisation d'un arbre de codage dans lequel les feuilles représentes de lettre a coder, leur codage étant le chemin binaire (gauche ou droite) de la racine vers la dite feuille. Ainsi plus la profondeur de la feuille est faible, plus le poids de la lettre correspondante sera faible une fois codé.

Question 1

Montrez qu'il existe un arbre tel que le codage de Huffman associé permet de réduire (au sens large) l'espace mémoire nécessaire au stockage des données.

Question 2

Proposer un algorithme permettant d'obtenir l'arbre de Huffman optimal au codage d'un certain texte T passé en entrée.

Il existe un algorithme en temps $\mathcal{O}(|T| + |A|\log(|A|))$ ou $|T|$ est la taille du texte passé en entrée et $|A|$ est la taille de l'alphabet utilisé dans le texte T

Dans toute la suite du TP les arbres seront codé à l'aide du type :

```
type 'a tree =
    Leaf of 'a
  | Node of 'a tree * 'a tree
;;
```

Décodage

Question 3

Écrivez une fonction `sub_tree` qui prend pour arguments un arbre t et un mot binaire w. Cette fonction retournera le sous-arbre de t désigné par w. Si le mot binaire w ne désigne pas un sous-arbre de t, vous lèverez une exception.

```
sub_tree :  'a tree → bool list → 'a tree
```

Question 4

Écrivez une fonction `read` qui prend pour arguments un arbre t et un mot binaire w. Cette fonction parcourera l'arbre t en lisant le mot w jusqu'à arriver sur une feuille. La fonction retournera alors l'étiquette de la feuille atteinte et la partie du mot w qui n'a pas été lue. Si le mot w ne permet pas d'atteindre une feuille, votre fonction lèvera une exception.

```
read :  'a tree → bool list → ( 'a * bool list)
```

Question 5

Écrivez une fonction `décode` qui prend pour argument un code de Huffman et un mot binaire. La fonction retournera la chaîne de caractères représentées par le mot binaire dans le code.

```
decode :  char tree → bool list → string
```

Construction du code

Nous nous intéressons maintenant à la construction d'un code de Huffman permettant de représenter un texte d'une manière la plus concise possible. La difficulté réside dans le fait que, pour obtenir un codage efficace, il faut choisir les codes des différents caractères en tenant compte de leurs fréquences respectives. La méthode que nous proposons nécessite de calculer dans un premier temps le nombre d'occurrences n_c de chaque caractère c présent dans le texte, afin de former la liste m des couples (n_c, c). L'algorithme consiste alors à itérer le processus suivant sur la liste m jusqu'à ce que celle-ci soit réduite à un élément :

- Retirer de la liste m les deux couples (n_1, t_1) et (n_2, t_2) tels que n_1 et n_2 soient minimaux,
- Ajouter à la liste m le couple $(n_1 + n_2, t_1 \circ t_2)$.

L'algorithme se termine lorsque la liste m est réduite à un coupe (n, t) : t est alors l'arbre du code de Huffman recherché. Pour obtenir un implémentation raisonnablement efficace, on maintiendra la liste m triée dans l'ordre des n_c croissants. Ainsi, les éléments minimaux apparaîtront toujours en tête.

Question 6

Quelle est la liste m obtenue si le texte considéré est "Lycée du parc, lyon" ? Quel est l'arbre construit par notre algorithme ?

Question 7

Écrivez une fonction `insert` prenant pour argument un élément e et une liste l supposée triée. La fonction retournera la liste obtenue à partir de l en ajoutant e de manière à maintenir le tri.

```
insert :  'a → 'a list → 'a list
```

Question 8

Écrivez une fonction `merge` prenant pour argument une liste de couples de la forme (n_c, c) supposée triée. Cette fonction réduira la liste comme décrit ci-dessus afin d'obtenir le code de Huffman correspondant.

```
merge :  (int * char tree) list → char tree
```

Question 9

Écrivez une fonction `count` qui, étant donnée une chaîne de caractères, calcule la liste des couples (n_c, c) indiquant le nombre d'occurrences de chaque caractère dans la chaîne.

```
count :  string → (char * int) list
```

Question 10

Écrivez une fonction `huffman` qui calcule l'arbre de Huffman correspondant à liste de couples (n_c, c) indiquant le nombre d'occurrences de chaque caractère.

```
huffman :  (int * char ) list → char tree
```

Codage

Question 11

Écrivez une fonction `extract` qui prend pour argument l'arbre d'un code de Huffman. Cette fonction calculera la liste des couples (c, w_c) ou w_c est le mot binaire représentant le caractère c dans le code.

```
extract :  char tree → (char , bool list) list
```

Question 12

Déduisez-en une fonction qui prend pour argument un code de Huffman ainsi qu'une chaîne de caractères et qui retourne le mot binaire représentant la chaîne de caractères dans le code de Huffman.

```
code :  char tree → string → bool list
```