TP Projet
Codage par la méthode de Huffman

Contenu de votre fichier de programme : votre fichier doit contenir en première ligne vos nom, prénom et numéro d'étudiant. Les commentaires doivent au minimum concerner les types des paramètres et des résultats de vos fonctions, ainsi que les tests réalisés, avec leurs résultats. Cela compte pour une part importante de l'évaluation.

Procédure de rendu du projet : lors de la séance du 9 ou 10 juin, votre enseignant vous demandera de lui présenter votre travail et de fournir des explications. Vous devrez lui envoyer par mail le fichier de votre TP avec comme sujet du mail « LIF3 projet». Le corps du mail doit comporter vos nom, prénom et numéro d'étudiant, ainsi que votre fichier attaché nommé « ProjetLIF3VotreNom.scm ».

Présentation du projet

On souhaite coder une phrase représentée par une liste de caractères, par exemple :

```
("a" "b" "r" "a" "c" "a" "d" "a" "b" "r" "a").
```

Chaque caractère est de type `string`. La fonction `equal?` permet de tester l'égalité de deux caractères.

1 Faire des statistiques sur les caractères

Dans cette première partie, on veut calculer la liste triée des couples (caractère fréquence).

• Écrire la fonction `ajoute1` qui, étant donné un caractère, ajoute 1 à sa fréquence dans une liste de fréquences. Si le caractère n'est pas encore répertorié dans la liste de fréquences, on l'ajoute.

```
(ajoute1 "e" '(("l" 1) ("e" 2) ("x" 1))) → (("l" 1) ("e" 3) ("x" 1))
(ajoute1 "e" '(("a" 4) ("b" 3))) → (("a" 4) ("b" 3) ("e" 1))
```

• Écrire la fonction `statistiques` qui, étant donnée une liste de caractères, calcule la liste des fréquences.

```
(statistiques '("a" "b" "r" "a" "c" "a" "d" "a" "b" "r" "a"))
          → (("a" 5) ("r" 2) ("b" 2) ("d" 1) ("c" 1))
```

• Écrire la fonction `inserefreq` qui, étant donné un couple (caractère fréquence), l'insère dans une liste de couples triée selon les fréquences.

```
(inserefreq '("b" 2) '(("d" 1) ("a" 5))) → (("d" 1) ("b" 2) ("a" 5))
```

• Écrire la fonction `triefreq` qui trie une liste de fréquences par insertions successives.

```
(triefreq '(("a" 5) ("r" 2) ("b" 2) ("d" 1) ("c" 1)))
          → (("d" 1) ("c" 1) ("r" 2) ("b" 2) ("a" 5))
```

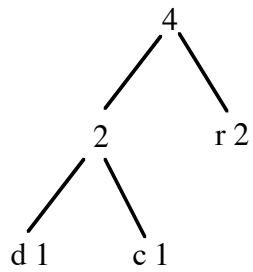
2 Construire l'arbre de Huffman

Dans cette deuxième partie, on veut construire l'arbre de Huffman à partir de la liste triée des couples (caractère fréquence).

On représente un arbre de Huffman de la manière suivante :

- les nœuds internes sont des triplets (poids fils-gauche fils-droit)
- les feuilles sont des couples (caractère poids).

Le poids est un entier qui représente la fréquence d'un caractère dans une feuille, ou la somme des fréquences de tous les caractères regroupés par un nœud interne.



Par exemple, l'arbre ci-dessus est représenté par la liste : `(4 (2 ("d" 1) ("c" 1)) ("r" 2))`

• Écrire les fonctions `fg` et `fd` qui renvoient le fils gauche (respectivement le fils droit) d'un arbre.

• Écrire la fonction booléenne `feuille`? qui teste si une liste représente une feuille.

• Écrire la fonction `poids` qui calcule le poids d'un arbre de Huffman, qu'il s'agisse d'un nœud interne ou d'une feuille.

• Écrire la fonction `fusion`, qui fusionne deux arbres de Huffman.

```
(fusion '("b" 2) '(4 (2 ("d" 1) ("c" 1)) ("r" 2)))
          → (6 ("b" 2) (4 (2 ("d" 1) ("c" 1)) ("r" 2)))
```

• Écrire la fonction `arbre-codage` qui, étant donnée la liste des fréquences triée, construit l'arbre de Huffman. On procédera par insertions successives de la fusion des deux premiers arbres dans le reste de la liste. On modifiera pour cela la fonction `inserefreq`.

```
(arbre-codage '(("d" 1) ("c" 1) ("r" 2) ("b" 2) ("a" 5)))
          → (11 ("a" 5) (6 ("b" 2) (4 (2 ("d" 1) ("c" 1)) ("r" 2))))
```

3 Construire le code

• Écrire la fonction `construit-code` vue en TD qui, étant donné un arbre de Huffman, construit la liste des codes associés à chaque caractère.

```
(construit-code '(11 ("a" 5) (6 ("b" 2) (4 (2 ("d" 1) ("c" 1)) ("r" 2)))))
  → (("a" (0)) ("b" (1 0)) ("d" (1 1 0 0)) ("c" (1 1 0 1)) ("r" (1 1 1)))
```

4 Coder

• Écrire la fonction `codage` qui, étant donnés un code et une liste de caractères, construit la liste de 0 et de 1 correspondant à la phrase codée.

```
(codage '(("a" (0)) ("b" (1 0)) ("d" (1 1 0 0)) ("c" (1 1 0 1)) ("r" (1 1 1)))
       '("a" "b" "r" "a" "c" "a" "d" "a" "b" "r" "a"))
         → (0 1 0 1 1 1 0 1 1 0 1 0 1 1 0 0 0 1 0 1 1 1 0)
```

Indication : on pourra utiliser la fonction prédéfinie `assoc` qui trouve dans une liste de couples celui commençant par l'atome passé en paramètre :

```
(assoc 'a '((c d) (a b) (e a))) → (a b)
```

5 Décoder

• Écrire la fonction `decodage` vue en TD qui, étant donnés un arbre de Huffman et une liste de 0 et de 1, décode la phrase ainsi codée.

```
(decodage '(11 ("a" 5) (6 ("b" 2) (4 (2 ("d" 1) ("c" 1)) ("r" 2))))
    '(0 1 0 1 1 1 0 1 1 0 1 0 1 1 0 0 0 1 0 1 1 1 0))
    → ("a" "b" "r" "a" "c" "a" "d" "a" "b" "r" "a")
```

6 Pour finir

• Écrire la fonction sans argument `compresser` qui demande à l'utilisateur une phrase à coder, et qui affiche la liste des fréquences, l'arbre de Huffman, le code, et la phrase codée.

```
(compresser) →
     Entrez une liste de caractères à coder :
     ("a" "b" "r" "a" "c" "a" "d" "a" "b" "r" "a")
     phrase :
     (a b r a c a d a b r a)
     liste des fréquences :
     ((a 5) (r 2) (b 2) (d 1) (c 1))
     arbre :
     (11 (a 5) (6 (b 2) (4 (2 (d 1) (c 1)) (r 2))))
     code :
     ((a (0)) (b (1 0)) (d (1 1 0 0)) (c (1 1 0 1)) (r (1 1 1)))
     phrase codée :
     (0 1 0 1 1 1 0 1 1 0 1 0 1 1 0 0 0 1 0 1 1 1 0)
```