TP-Info 6 – Huffman (version *Easy*)

1 Qu'est-ce que Huffman ?

1.1 Dans la théorie

`https://fr.wikipedia.org/wiki/Codage_de_Huffman`

1.2 Dans le cadre de ce TP

Dans le cadre de ce TP, nous allons utiliser l'algorithme de Huffman pour compresser du texte simple. Voici les étapes clefs de l'utilisation de l'algorithme de Huffman :

- Génération d'une table de correspondance (aussi appelée dictionnaire) pour associer chaque mot à une valeur
- Compression : Utilisation de la table de correspondance pour compresser le texte en remplaçant les mots connus dans le dictionnaire par leur correspondance numérique
- Décompression : Utilisation de la table de correspondance pour décompresser le texte en remplaçant chaque valeur numérique par sa représentation en mot.

Nous n'avons pas encore toutes les notions pour effectuer un encodage Huffman, nous allons donc mettre en place une version simplifiée pour ce TP (libre à vous d'aller plus loin pour explorer des pistes) :

- Implémentation de l'algorithme de Huffman sans utiliser d'arbre.
- Utilisation d'un système statistique pour associer les mots (on compresse les mots, pas les symboles) à un entier.
- Plus l'entier est petit, plus le mot est présent.
- Nous considérons qu'il n'y a pas de caractères spéciaux dans le texte à compresser : pas d'accent, pas de ponctuation, pas de chiffre, etc. Donc les seuls caractères présents seront alphabétiques (a-zA-Z) et les espaces (espace, tabulation, retour à la ligne).

Concrètement, avec le texte "On ne connait pas la vie des anges dans la vie des anges de la vie", nous pouvons générer la table de correspondance suivante :

- "vie" = 1
- "la" = 2
- "anges" = 3
- "des" = 4
- "de" = 5
- "dans" = 6
- "pas" = 7
- "connait" = 8
- "ne" = 9
- "On" = 10

Voici un exemple de compression de Huffman avec un texte simple en utilisant la table de correspondance ci-dessus :

```
# compress_huffman table "On ne connait pas la vie des anges dans la vie des anges de la vie";;
  - : string = "10 9 8 7 2 1 4 3 6 2 1 4 3 5 2 1"
```

2 Implémentation de l'algorithme d'Huffman pour ce TP

Dans cette version *Easy* du sujet, voici les quelques étapes clefs à réaliser :

2.1 Table de correspondance (ou dictionnaire)

Le but est de réfléchir à la façon de représenter la table de correspondance pour associer un mot à une valeur. Vous pouvez essayer de faire des schémas sur une feuille de papier.

Pour vous aider :
— Quel élément du langage Caml vous connaissez pour associer un string avec un entier ?
— Comment vous pouvez stocker les différentes associations string avec entier ?

2.2 Création du dictionnaire

Il vous faut créer une fonction pour créer le dictionnaire par le biais d'une analyse statistique d'un texte donné. Vous ne pouvez pas tout faire en une fonction, il faut bien penser à découper le problème en plusieurs sous-fonctions.

Pour votre table de correspondance : plus un mot est présent dans le texte, plus il doit avec une valeur faible dans la table.

2.2.1 Fonction *tokenize()*

Pour commencer et tester votre programme, il est plus simple de travailler avec une chaîne de caractère courte.

Par exemple :

```
let input1 = "On ne connait pas la vie des anges dans la vie des anges de la vie";;
```

Vous devez créer la fonction *tokenize()* qui découpe un texte (string) en une liste de mot.

```
# let tokenize str = ...
val tokenize : string -> string list = <fun>

# let word_list = tokenize input1;;
val word_list : string list =
  ["On"; "ne"; "connait"; "pas"; "la"; "vie"; "des"; "anges"; "dans"; "la";
   "vie"; "des"; "anges"; "de"; "la"; "vie"]
```

2.2.2 Fonction *create_stat_dic()*

Maintenant que vous avez la liste de mot, il va falloir faire des statistiques d'utilisation pour chaque mot. C'est simple, on va compter le nombre de fois où chaque mot apparaît.

Le résultat de votre fonction est une liste de couple (chaque couple avec le mot et le nombre d'occurrences du mot).

Pour réaliser cette fonction, vous aurez besoin d'en écrire une autre qui a pour but de créer progressivement votre liste de couple en ajoutant le nouveau mot ou en augmentant le nombre d'occurrences.

Voici un exemple d'utilisation de vos fonctions :

```
%
% Code and use function add_in_list()
%

# let rec add_in_list mot liste = ...
      val add_in_list : 'a -> ('a * int) list -> ('a * int) list = <fun>

# let liste = add_in_list "bonjour" [];;
val liste : (string * int) list = [("bonjour", 1)]
# let liste = add_in_list "les" liste;;
val liste : (string * int) list = [("bonjour", 1); ("les", 1)]
# let liste = add_in_list "sups" liste;;
val liste : (string * int) list = [("bonjour", 1); ("les", 1); ("sups", 1)]
# let liste = add_in_list "bonjour" liste;;
val liste : (string * int) list = [("bonjour", 2); ("les", 1); ("sups", 1)]

%
% Code and use function create_stat_dic()
%

# let create_stat_dic word_list = ...
          val create_stat_dic : 'a list -> ('a * int) list = <fun>

# let stat_dic = create_stat_dic word_list;;
val stat_dic : (string * int) list =
  [("On", 1); ("ne", 1); ("connait", 1); ("pas", 1); ("la", 3); ("vie", 3);
   ("des", 2); ("anges", 2); ("dans", 1); ("de", 1)]
```

2.2.3 Fonction *order_stat_dic()*

Maintenant que le nombre d'occurrences de chaque mot est trouvé, nous allons devoir trier la liste en ordre décroissant selon ce nombre d'occurrences.

Pour réaliser cette fonction, vous allez devoir faire la fonction *insert_order()* qui permet de trier une liste. Nous avons déjà vu cette fonction en TD avec une implémentation simple en utilisant un accumulateur. Il faut parcourir la liste à trier et, pour chaque élément, ajouter l'élément courant à l'accumulateur qui représente la nouvelle liste triée.

Voici un exemple d'utilisation de vos fonctions :

```
%
% Code and use function insert_order()
%

# let rec insert_order dic (str, count) = ...
      val insert_order : ('a * 'b) list -> 'a * 'b -> ('a * 'b) list = <fun>

# let t = insert_order [] ("a", 1);;
val t : (string * int) list = [("a", 1)]
# let t = insert_order t ("b", 4);;
val t : (string * int) list = [("b", 4); ("a", 1)]
# let t = insert_order t ("c", 3);;
val t : (string * int) list = [("b", 4); ("c", 3); ("a", 1)]

%
% Code and use function order_stat_dic()
%

# let order_stat_dic stat_dic = ...
          val order_stat_dic : ('a * 'b) list -> ('a * 'b) list = <fun>

# let order_stat_dict = order_stat_dic stat_dic;;
val order_stat_dict : (string * int) list =
  [("vie", 3); ("la", 3); ("anges", 2); ("des", 2); ("de", 1); ("dans", 1);
   ("pas", 1); ("connait", 1); ("ne", 1); ("On", 1)]
```

2.2.4 Fonction *create_table()*

Maintenant, il reste à créer la fonction *create_table()* qui va créer la base de concordance d'un mot vers le nombre associé.

Voici un exemple d'utilisation de votre fonction :

```
%
% Create a function to ordre stat dict
%

let create_table order_stat_dict = ...
          val create_table : ('a * 'b) list -> ('a * int) list = <fun>

#     let table = create_table order_stat_dict;;
val table : (string * int) list =
  [("vie", 1); ("la", 2); ("anges", 3); ("des", 4); ("de", 5); ("dans", 6);
   ("pas", 7); ("connait", 8); ("ne", 9); ("On", 10)]
```

2.2.5 Tout faire en une fonction *generate_huffman_table()*

Dernière étape : assembler tout votre travail en une seule fonction qui appelle toutes les autres. Nous allons ainsi directement créer *generate_huffman_table()* avec le code suivant :

```
%
% Create the final function generate_huffman_table()
%

# let generate_huffman_table input =
     create_table (order_stat_dic (create_stat_dic (tokenize input)));;
val generate_huffman_table : string -> (string * int) list = <fun>


# let table = generate_huffman_table input1;;
val table : (string * int) list =
  [("vie", 1); ("la", 2); ("anges", 3); ("des", 4); ("de", 5); ("dans", 6);
   ("pas", 7); ("connait", 8); ("ne", 9); ("On", 10)]
```

2.3 Compression avec l'utilisation du dictionnaire

Vous devez écrire une fonction qui fera la compression de votre texte.

Votre fonction utilise la table de correspondance générée à l'étape précédente.

Voici les différentes étapes possibles :
- Lecture de la chaîne de caractère d'entrée pour faire la compression
- Après chaque mot lu, il faut chercher ce mot dans le dictionnaire :
  - si le mot est présent dans le dictionnaire : il faut remplacer le mot par le nombre associé et continuer la compression
  - si le mot n'est pas dans le dictionnaire : écrire le mot en l'état dans le message compressé. Un bonus possible est d'ajouter ce nouveau mot dans le dictionnaire pour faire de la compression en live.

Pour réaliser la compression, seulement deux fonctions sont nécessaires (en plus des fonctions précédemment réalisées) :
- *get_compress()* pour trouver dans le dictionnaire le nombre qui correspond à un mot
- *compress()* pour compresser tout le message

2.3.1 Fonction *get_compress()*

Voici l'exemple d'utilisation de la fonction :

```
%
% Create and use get_compress()
%

# let rec get_compress dic word = ...
      val get_compress : ('a * 'b) list -> 'a -> 'b = <fun>

# table;;
   - : (string * int) list =
[("vie", 1); ("la", 2); ("anges", 3); ("des", 4); ("de", 5); ("dans", 6);
 ("pas", 7); ("connait", 8); ("ne", 9); ("On", 10)]

# get_compress table "anges";;
   - : int = 3
```

2.3.2 Fonction *compress()*

Pour finir, la fonction *compress()* qui va parcourir notre message pour compresser chaque mot pour retourner le message final compresser.

Cette fonction sera très proche de la fonction *tokenize()*. La différence étant qu'elle ne va pas créer de liste de string mais un string avec la concaténation des différents mots compressés.

Voici l'exemple d'utilisation finale de la compression :

```
%
% Create compress()
%

# let compress table str = ...
     val compress : (string * int) list -> string -> string = <fun>


%
% Use compress() on input1
%

# input1;;
#   - : string =
"On ne connait pas la vie des anges dans la vie des anges de la vie"

# let table = generate_huffman_table input1;;
val table : (string * int) list =
  [("vie", 1); ("la", 2); ("anges", 3); ("des", 4); ("de", 5); ("dans", 6);
   ("pas", 7); ("connait", 8); ("ne", 9); ("On", 10)]

# compress table input1;;
# - : string = "10 9 8 7 2 1 4 3 6 2 1 4 3 5 2 1"


%
% Use compress() on input2
%

# input2;;
#   - : string =
"On ne sait pas la vie des anges non plus\nQue des moulins a eaux\nOn se sert un
grand verre de vent\nDe sources de pluie des yeux\nOn ignore comment vivre comme
eux\nOn se sert un grand verre de vin\nOn se sert un grand verre de vin\nDans une
maison avec enfants a venir, chiens"

# let table = generate_huffman_table input2;;
val table : (string * int) list =
  [("On", 1); ("de", 2); ("verre", 3); ("grand", 4); ("un", 5); ("sert", 6);
   ("se", 7); ("des", 8); ("vin", 9); ("a", 10); ("chiens", 11);
   ("venir,", 12); ("enfants", 13); ("avec", 14); ("maison", 15);
   ("une", 16); ("Dans", 17); ("eux", 18); ("comme", 19); ("vivre", 20);
   ("comment", 21); ("ignore", 22); ("yeux", 23); ("pluie", 24);
   ("sources", 25); ("De", 26); ("vent", 27); ("eaux", 28); ("moulins", 29);
   ("Que", 30); ("plus", 31); ("non", 32); ("anges", 33); ("vie", 34);
```

```
   ("la", 35); ("pas", 36); ("sait", 37); ("ne", 38)]

# compress table input2;;
# - : string =
"1 38 37 36 35 34 8 33 32 31 30 8 29 10 28 1 7 6 5 4 3 2 27 26 25 2 24 8 23 1
22 21 20 19 18 1 7 6 5 4 3 2 9 1 7 6 5 4 3 2 9 17 16 15 14 13 10 12 11"
```

2.4 Décompression avec l'utilisation du dictionnaire

Vous avez toutes les cartes en main.

3 Annexe

3.1 Annexe 1 : Une phase simple

```
let input1 = "On ne connait pas la vie des anges dans la vie des anges de la vie";;
```

3.2 Annexe 2 : Un paragraphe

```
let input2 = "On ne sait pas la vie des anges non plus
Que des moulins a eaux
On se sert un grand verre de vent
De sources de pluie des yeux
On ignore comment vivre comme eux
On se sert un grand verre de vin
On se sert un grand verre de vin
Dans une maison avec enfants a venir chiens";;
```

3.3 Annexe 3 : Le texte entier avec la ponctuation

```
let input3 = "On ne connait pas le coeur des gens
Il est tant mal visible que parfois, on cogne dedans
Quelle misere de prendre le train quand
Au bout, il n'y a personne, rien

On ne sait pas la vie des anges non plus
Que des moulins a eaux
On se sert un grand verre de vent
De sources de pluie des yeux
On ignore comment vivre comme eux
On se sert un grand verre de vin
On se sert un grand verre de vin
Dans une maison avec enfants a venir, chiens

Le quai fait des bruits de chaussures
Le quai fait des bruits de valise a roulettes
Des bruits d'avant
Le quai est vide, vide, vide
On bute dans l'air (dans l'air)

Pardon, messieurs-dames
J'ai cru à un nuage
Vous etes innombrables qui personne
Je suis innombrable, et comme vous, presque rien

Prenons donc un pot amical
Au lieu d'un poteau noir, d'un mauvais coup
On ne connait pas d'autres coeurs dans le noir que le notre
Et encore, ni dans le jour non plus, alors : a la bonne votre !
Et nous debarquerons sous le soleil battant

Prenons donc un pot amical
Au lieu d'un poteau noir, d'un mauvais coup
On ne connait pas d'autres coeurs dans le noir que le notre
Et encore, ni dans le jour non plus, alors : a la bonne votre !
Et nous debarquerons sous le soleil battant";;
```

3.4 Annexe 4 : Un début de fonction *tokenize*

```
let tokenize str =
  let length = String.length str in
  let rec r_tok sstr i =
    if i >= length then
      (if sstr = "" then [] else [sstr])
    else if str.[i] = ' ' || str.[i] = '\n' then
      sstr::r_tok "" (i + 1)
    else
      r_tok (sstr ^ Char.escaped (str.[i])) (i + 1)
  in
  r_tok "" 0;;
```