▶ **MPSI – Option Informatique**
Année 2001, Septième TP Caml
Vincent Simonet (`http://cristal.inria.fr/~simonet/`)

Compression LZW

Lorsque l'on stocke ou l'on transmet via *un réseau une grande quantité de données, on cherche souvent à réduire taille des informations manipulées — en particulier pour diminuer le coût du stockage ou la durée de la communication. En utilisant certaines particularités des données considérées — par exemple les redondances dans un texte — il est possible de concevoir des algorithmes de compression permettant dans les représenter sous une forme généralement plus petite.*
Nous nous intéressons aujourd'hui à l'algorithme de compression LZW (du nom de ses inventeurs Abraham Lempel, Jakob Ziv qui l'ont proposé en 1977 et Terry Welch qui l'a finalisé en 1984). Il s'agit d'une compression conservative *dans la mesure où, après un cycle compression et décompression, les données ne sont pas altérées. Cet algorithme est, entre autres, à la base des programmes de compression « ZIP » et des formats d'images « GIF » et « TIFF ».*

1 Gestion d'un dictionnaire

Nous écrivons dans cette partie quelques fonctions préliminaires.
Nous appelons *dictionnaire* un tableau contenant des chaînes de caractères que nous supposerons toutes distinctes. Un dictionnaire pourra toutefois comporter des « *cases vides* » qui contiendront en fait la chaîne vide "".
Étant donné un dictionnaire d et une chaîne s, on appelle *index de s dans d* l'indice de la première case de d contenant la chaîne s. Si la chaîne s n'est pas présente dans d, cet entier n'est pas défini.

▶ **Question 1** *Écrivez une fonction* make_dict *qui prend pour argument un entier n et crée un dictionnaire de taille $256 + n$ initialisé comme suit :*
- *pour $0 \leq i \leq 255$, $d.(i)$ contient la chaîne formée du caractère ayant pour code ASCII l'entier i.*
- *pour $i \geq 256$, $d.(i)$ contient une chaîne vide.*

value make_dict : *int → string vect*

Pour générer une chaîne formée du caractère de code ASCII i, vous pourrez utiliser l'expression *Caml*

```
string_of_char (char_of_int i)
```

Étant donnés un dictionnaire d et une chaîne s, nous nous intéressons particulièrement à deux opérations élémentaires : d'une part obtenir l'index d'une chaîne dans d (si cette chaîne est présente dans le dictionnaire) et d'autre part ajouter une chaîne à un dictionnaire en la plaçant dans la première case libre.

▶ **Question 2** *Écrivez une fonction* index *qui prend pour arguments un dictionnaire d et une chaîne s. Cette fonction retournera l'indice de la première occurence de s dans d. La fonction renverra par convention -1 si s n'apparaît pas dans d.*

value index : *string vect → string → int*

▶ **Question 3** *Programmez une fonction* add *qui prend également pour arguments un dictionnaire d et une chaîne s. Cette fonction ajoutera s au dictionnaire d dans la première case libre. Si le dictionnaire passé en argument est plein, il sera laissé en l'état.*

value add : *string vect → string → unit*

2 Algorithme LZW

2.1 Compression

On s'intéresse dans un premier temps à l'algorithme de compression. L'entrée de cet algorithme est la chaîne de caractères s que l'on souhaite compresser. On note n sa longueur. Le résultat de la compression sera représenté par une liste d'entiers. Ces entiers seront des index correspondant aux entrées d'un dictionnaire.
L'algorithme LZW utilise en effet un dictionnaire initialisé comme indiqué à la question 1. Au fur et à mesure de la lecture de la chaîne à compresser ce dictionnaire est complété par des sous-chaînes de s.

Voici une description informelle de l'algorithme de compression. Dans cette description, on note $a \leftarrow x$ l'action de *mettre la valeur x dans la variable a*, et $s_1 \oplus s_2$ la concaténation des deux chaînes de caractères s_1 et s_2.

```
créer un dictionnaire d
tampon ← s.[0]
resultat ← [ ]
pour i variant de 1 à n − 1 faire
  si (tampon ⊕ s.[i]) est dans d
  alors tampon ← tampon ⊕ s.[i]
  sinon resultat ← index(tampon) :: resultat
        ajouter (tampon ⊕ s.[i]) au dictionnaire
        tampon ← s.[i]
resultat ← index(tampon) :: resultat
renvoyer le miroir de resultat
```

▶ **Question 4** *Appliquez cet algorithme à la main sur la phrase suivante*

maman aime manger

Vous pourrez supposer pour simplifier que le dictionnaire est initialisé avec le caractère d'espacement et les 26 lettres de l'alphabet.

▶ **Question 5** *Écrivez une fonction* compresse *qui prend pour argument une chaîne de caractères et lui applique l'algorithme précédent. Le résultat sera rendu sous la forme d'une liste d'entiers.*

value compresse : *string* → *int list*

2.2 Décompression

Il nous faut maintenant écrire une procédure de décompression. Cette procédure doit prendre en entrée la liste produite par la fonction compresse et retrouver la chaîne de caractères initiale. La difficulté réside dans le fait que l'on ne conserve pas le dictionnaire construit lors de la compression. Il faut donc le reconstruire lors de la décompression.
Soit t une liste d'entiers produite par l'algorithme de compression. On note t_i le i-ème élément de cette liste (pour $0 \leq i < n$). Voici une description de l'algorithme de décompression :

```
créer un dictionnaire d
tampon ← d.(t_0)
resultat ← tampon
pour i variant de 1 à n − 1 faire
  soit p = d.(t_i)
  resultat ← resultat ⊕ p
  soit c le premier caractère de p
  ajouter tampon ⊕ c à d
  tampon ← p
renvoyer la chaîne resultat
```

▶ **Question 6** *Décompressez à la main et à l'aide de cet algorithme le code que vous avez obtenu à la question 4.*

▶ **Question 7** *Écrivez une fonction* decompresse *qui prend pour argument une liste d'entiers (supposée issue de la fonction* compresse*) et lui applique l'algorithme de décompression pour obtenir la chaîne correspondante.*

value decompresse : *int list* → *string*

3 Évaluation du taux de compression

La représentation du résultat de la compression par une liste d'entiers *Caml* n'est pas très réaliste : il faudrait *a priori* 30 bits pour stocker chaque entier. Cependant, on remarque que la taille des entiers produits par l'algorithme de compression croît progressivement au fur et à mesure que l'on avance dans la liste (et que le dictionnaire se remplit). Dans la pratique, on peut donc utiliser la technique suivante pour coder la liste :

- Tant que tous les entiers sont strictement inférieurs à 255, coder ces entiers sur 8 bits.
- Lorsque l'on rencontre le premier entier supérieur ou égal à 255, émettre la séquence 11111111 (huit fois le bit 1) et continuer, tant que les entiers sont strictement inférieurs à 511, en codant les entiers sur 9 bits.
- Lorsque l'on rencontre le premier entier supérieur ou égal à 511, émettre la séquence 111111111 (neuf fois le bit 1) et continuer, tant que les entiers sont strictement inférieurs à 1023, en codant les entiers sur 10 bits.

De manière générale, tant que les entiers considérés sont strictement inférieurs à $n = 2^k - 1$, on peut les représenter sur k bits. Lorsque le premier entier supérieur ou égal à $2^k - 1$ est rencontré, on émet la séquence $1 \ldots 1$ (k fois le bit 1) et on continue en codant les entiers sur $k + 1$ bits.

▶ **Question 8** *Écrivez une fonction* espace *qui étant donnée une liste d'entiers produite par l'algorithme de compression calcule* en octets *l'espace nécessaire pour stocker cette liste en utilisant le codage décrit ci-dessus.*

value espace : *int list* → *int*

En utilisant la fonction espace vous pouvez facilement calculer le taux de compression obtenu par votre algorithme sur différents exemples de votre choix.

▶ **Question 9** *Écrivez une fonction* bin_of_int *qui prend pour argument deux entiers n et k et renvoie une liste de longueur k représentant l'entier n en binaire (bit de poids faible en tête). Cette liste sera tronquée si l'écriture binaire de n comporte plus de k chiffres.*

value bin_of_int : *int* → *int* → *int list*

▶ **Question 10** *Déduisez-en une fonction* code *qui prend pour argument une liste d'entiers produite par la fonction* compresse *et retourne la liste binaire correspondante.*

S'il vous reste du temps, vous pouvez (enfin) programmer une fonction decode qui prend une liste de booléens et retourne la liste d'entiers qu'elle code. Nous avons écrit à peu près toutes les étapes d'un algorithme de compression effectif. Cependant, nos algorithmes manquent beaucoup d'efficacité dans la gestion des dictionnaires. Nous verrons lors de la prochaine séance une structure de donnée permettant de représenter efficacement un tel objet.

▶ **MPSI – Option Informatique**
Année 2001, Septième TP Caml
Vincent Simonet (`http://cristal.inria.fr/~simonet/`)

Compression LZW

Un corrigé

▶ Question 1

```
let make_dict n =
  let d = make_vect (256 + n) "" in
  for i = 0 to 255 do
    d.(i) <- string_of_char (char_of_int i)
  done;
  for i = 256 to n - 1 do
    d.(i) <- ""
  done;
  d
;;
```

▶ Question 2

```
let index d s =
  let n = vect_length d in
  let i = ref 0 in
  while !i < n && d.(!i) <> s do
    i := !i + 1
  done;
  if !i = n then i := -1;
  !i
;;
```

▶ Question 3

```
let add d s =
  let n = vect_length d in
  let i = ref 0 in
  while !i < n && d.(!i) <> "" do
    i := !i + 1
  done;
  if !i < n then d.(!i) <- s
;;
```

▶ Question 5

```
let compresse s =
  let n = string_length s in
  let d = make_dict n in
  let resultat = ref [] in
  let tampon = ref (sub_string s 0 1) in

  for i = 1 to n - 1 do
    let tampon' =
      !tampon ^ (sub_string s i 1)
    in
    if index d tampon' >= 0 then
      tampon := tampon'
    else begin
      resultat := (index d !tampon) :: !resultat;
      add d tampon';
      tampon := sub_string s i 1
    end
  done;

  rev ((index d !tampon) :: !resultat)
;;
```

▶ Question 11

```
let rec list_max = function
    [] -> raise (Invalid_argument "list_max")
  | [x] -> x
  | x :: y :: q -> list_max ((max x y) :: q)
;;

let decompresse = function
    [] -> ""
  | t0 :: queue as liste ->
      let d = make_dict (list_max liste) in
      let rec aux tampon = function
          [] -> ""
        | t :: reste ->
          add d (tampon ^ (sub_string d.(t) 0 1));
          d.(t) ^ (aux d.(t) reste)
      in
      d.(t0) ^ (aux d.(t0) queue)
;;
```

▶ Question 8

```
let espace list =
  let rec aux n k = function
      [] -> 0
    | t :: q as l ->
      if t < n then k + aux n k q
      else k + aux ((1 + n) * 2 - 1) (k + 1) l
  in
  let x = aux 255 8 list in
  x / 8 + (if x mod 8 = 0 then 0 else 1)
;;
```

▶ Question 9

```
let rec bin_of_int = fun
    0 _ ->
      []
  | k 0 ->
      0 :: (bin_of_int (k - 1) 0)
  | k n ->
      (n mod 2) :: (bin_of_int (k - 1) (n / 2))
;;
```

▶ Question 10

```
let rec make_list x = function
    0 -> []
  | n -> x :: (make_list x (n - 1))
;;

let code list =
  let rec aux n k = function
      [] -> []
    | t :: q as l ->
      if t < n then
        (bin_of_int k t)
          @ (aux n k q)
      else
        (make_list 1 k)
          @ (aux ((1 + n) * 2 - 1) (k + 1) l)
  in
  aux 255 8 list
;;
```