```
#load "dynarray.cmo"
#load "heap.cmo"

(* Le codage de Huffman est un code préfixe:
   aucun code d'une lettre n'est préfixe du code d'une autre lettre.
   Si m et m' sont deux message dont le code est z,
   le code du premier caractère de m est un préfixe de z,
   de même que celui du premier caractère de m'.
   La propriété précédente nous donne que ces deux codes sont identiques,
   donc que m et m' commencent par la même lettre.
   Une récurrence immédiate conclut. *)

type huffman =
  | Noeud of huffman * huffman
  | Feuille of char

type direction = Gauche | Droite

let dictionnaire huffman =
  let result = Array.make 256 [] in
  let rec parcours prefixe = function
    | Noeud (gauche, droite) ->
        parcours (Gauche :: prefixe) gauche;
        parcours (Droite :: prefixe) droite
    | Feuille c ->
        result.(Char.code c) <- prefixe in (* qui est donc à l'envers *)
  parcours [] huffman;
  result

let zip huffman string =
  let dico = dictionnaire huffman in
  let result = ref [] in
  String.iter (fun c -> result := dico.(Char.code c) @ !result) string;
  List.rev !result

let rec decode_un_char liste huffman =
  match liste, huffman with
    | Gauche::queue, Noeud (gauche,_) -> decode_un_char queue gauche
    | Droite::queue, Noeud (_,droite) -> decode_un_char queue droite
    | l, Feuille c -> l,c
    | [], Noeud (_, _) -> invalid_arg "decode_un_char"

let list_iteri f l =
  ignore (List.fold_left (fun i x -> f i x; succ i) 0 l)

let unzip huffman directions =
  let chars = ref [] in
  let rec parcours = function
    | [] -> ()
    | liste ->
        let reste,c = decode_un_char liste huffman in
        chars := c :: !chars;
        parcours reste in
  parcours directions;
  let result = String.create (List.length !chars) in
  list_iteri (String.set result) (List.rev !chars);
  result

(* Soit t un arbre de Huffman optimal (pour un message donné),
   et a une lettre de fréquence minimale.
   Si a n'est pas à la profondeur maximale,
   on peut échanger a et une feuille de profondeur maximale
   en diminuant (au sens large) le nombre de bits utilisés pour le codage.
   Soit b une lettre de fréquence minimale après a.
   Si b n'est pas soeur de a,, on peut échanger b et la soeur de a
   en diminuant (au sens large) le nombre de bits utilisés pour le codage.
*)

let (|>) x f = f x (*ainsi [x |> f |> g |> h] signifie [f (g (h x))]*)

let best_huffman string =
  let frequences = Array.make 256 0 in
  String.iter
    (fun c -> let n = Char.code c in frequences.(n) <- succ frequences.(n))
    string;
  let feuilles = ref [] in
  Array.iteri
   (fun code freq -> if freq > 0 then feuilles := (freq,Feuille (Char.chr code)) :: !feuilles)
   frequences;
  let huffmans =
    Heap.of_array (fun (freq1,_) (freq2,_) -> freq1 < freq2) (Array.of_list !feuilles)  in
  while Heap.length huffmans > 1 do
    let freq_g,fils_gauche = Heap.take_min huffmans in
    let freq_d,fils_droite = Heap.take_min huffmans in
    Heap.insere (freq_g+freq_d, Noeud (fils_gauche,fils_droite)) huffmans
```

```
  done;
  snd (Heap.take_min huffmans)

let test =  "Félicitations pour lire les corrigés jusqu'au bout, continuez ainsi et n'hésitez
pas à me faire part de vos commentaires!"
let h = best_huffman test
let zippe = zip h test
let _ = String.length test, List.length zippe /8 (* car les caractères sont codés sur 8 bits.
*)
let _ = unzip h zippe
```