Si vous avez des questions, n’hésitez pas à m’interroger lors d’une séance de TP, ou par courrier électronique :

dmartin@irisa.fr

1 Un corrigé

▶ Question 1

```
let diviser n l =
  let rec aux m f = function
  | [] -> m,(rev f),[]
  | (t::[]) -> m,(rev f),(t]
  | ((i,v)::q) ->
    if m+v >= (n/2) then
      (if n-2*m-v >= 0 then
        m,v,(rev ((i,v)::f)),q
      else
        m,(rev f),((i,v)::q))
    else
      aux (m+v) ((i,v)::f) q
  in
  aux 0 [] l

```

▶ Question 2

```
let creer_arbre_ShannonFano l =
  let occ = liste_occurrences l
  and n = list_length l
  in
  let rec aux m = fun
  | [] -> failwith "Internal error"
  | ((i,v)::[]) -> Feuille(i)
  | (i,v) ->
    let m',g,d = diviser m l
    in
    Noeud((aux m' g),(aux (m-m') d))
  in
  aux n occ
```

▶ Question 3

```
let encoder arbre l =
  let table = table_conversion_of_arbre arbre
  in
  let rec aux = fun
  | [] -> []
  | (t::q) -> table.(t)@(aux q)
  in
  aux l
```

Question 4

```ocaml
let rec decoder_car l = fun
  | (Feuille x) -> x, l
  | (Noeud(g,d)) ->
    if l=[] then failwith "Liste vide"
    else if (hd l) then (decoder_car (tl l) d)
    else (decoder_car (tl l) g)
|;
```

Question 5

```ocaml
let rec decoder arbre l =
  if l = [] then []
  else try
    let x, l' = decoder_car arbre l in
    x :: (decoder arbre l')
  with |
    | (Failure "Liste vide") => []
  |;
```

Question 6

```ocaml
let creer_arbre_Huffman l =
  let rec aux = fun
    | [] -> failwith "Fichier vide"
    | ((a,v)::[]) -> a
    | ((a,v)::(b,w)::q) -> aux (insérer_arbre (Noeud(a,b)) (v+w) q)
  in
  aux (liste_feuilles l)
|;
```

Question 7

```ocaml
let dct_cos x i n =
  cos (((float_of_int (int (2 * x + 1) * i)) *. PI) /. (float_of_int (2 * n)))
|;
let dct_int_ij m i j =
  let res = ref 0.
  and ci = if i = 0 then 1. / / (sqrt 2.) else 1.
  and cj = if j = 0 then 1. / / (sqrt 2.) else 1.
  and n = vect_length m
  in
  for x=0 to n-1 do
    let cix = dct_cos x i n
    in
      for y=0 to n-1 do
        res := !res +. (float_of_int m.(x).(y)) *
        . cix *. (dct_cos y j n)
      done
  done;
|;
let dct_int m =
  let n = vect_length m
  in
    let dct = make_matrix n n 0
```
Question 8

```
let dct_inv_int_xy dct x y =
  let n = vect_length dct
  in
  for i=0 to n-1 do
    let cix = dct_cos i n
    in
    let ci = if i = 0 then 1. / (sqrt 2.) else 1.
        and cj = if j = 0 then 1. / (sqrt 2.) else 1.
        in
      res := ! res +. ci * cj * (float_of_int dct.(i).(j))
        * cix * (dct_cos y n)
    done;
  done;
  int_of_float (! res * 2. / (float_of_int n))
```

```
let dct_inv_int dct =
  let n = vect_length dct
  in
    let m = make_matrix n n 0
    in
      for x=0 to n-1 do
        for y=0 to n-1 do
          m.(x).(y) <- dct_inv_int_xy dct x y
        done;
      done;
  dct
```

Question 9

```
let quantifier q m =
  let n = vect_length m
  in
    for i=0 to n-1 do
      for j=0 to n-1 do
        m.(i).(j) <- m.(i).(j) / q.(i).(j)
      done;
  done;

let dequantifier q m =
  let n = vect_length m
  in
    for i=0 to n-1 do
      for j=0 to n-1 do
        m.(i).(j) <- m.(i).(j) * q.(i).(j)
      done;
  done;
```
Question 10

```
let encoder_jpeg l =
  let nbbyte = 4
  and max_0 = 16
  in
  let rec count_0 = fun |
    [] -> 0,[]
    (0::q) -> let x,l = (count_0 q) in (x+1),l
  in
  let rec aux = fun |
    [] -> []
    (0::q) ->
      let x',l = count_0 (0::q)
      in
      let x = ref x'
      and r = ref (aux l)
      in
      while !x > 0 do
        if !x > max_0 then
          r := (true::(bool_from_int (max_0-1) nbbyte)) @ !r
        else
          r := (true::(bool_from_int (!x-1) nbbyte)) @ !r;
          x := !x - max_0
        done;
        ![r]
      | (t::q) -> false::(encoder_nombre_jpeg t)@(aux q)
      in
      aux l
```

Question 11

```
let rec decoder_jpeg size = fun |
  [] -> [],[]
  | l when (size <= 0) -> [],l
  | (true::q) ->
    let x,l = int_from_bool q 4
    in
    let r',q' = (decoder_jpeg (size-x-1) l)
    in
    let r = ref r'
    in
    for i=0 to x do r := 0::(!r) done;
    ![r,q']
  | (false::q) ->
    try
      let x,l' = decoder_nombre_jpeg q
      in
      let r',q' = (decoder_jpeg (size-1) l')
      in
      (x::r'),q'
    with |
      (Failure "Liste vide") -> [],false::q
      | (Failure "hd") -> [],false::q
    in
  in
```

Question 12

```
let compress_jpeg_one m q =
  let v = dct_int m in
  quantifier q v;
  encoder_jpeg (parcours_zigzag v)
```
Question 13

```ocaml
let decompress_jpeg_one q l =
  let ml, r = (decoder_jpeg ((vect_length q) * (vect_length q)) l) in
  if ml = [] then (make_matrix 0 0 0), r
  else
    let v = retrace_zigzag ml in
    if (vect_length v) != (vect_length q) then (make_matrix 0 0 0), r
    else
      dequantifier q v;
      (dct_inv_int v), r
```

;;