Option Informatique en Sup MPSI
TP : algorithmique des puces à ADN, le corrigé

Question 1 Rien à signaler.

```
let copy_matrix m =
 let nl = vect_length m and nc = vect_length m.(0) in
 let m' = make_matrix nl nc m.(0).(0) in
 for i = 0 to nl - 1 do
  for j = 0 to nc - 1 do
   m'.(i).(j) <- m.(i).(j)
  done
 done ; m' ;;
```

Question 2 Le type de base de la matrice image est l'image par f du type de base de la matrice m.

```
let do_matrix f m =
 let nl = vect_length m and nc = vect_length m.(0) in
 for i = 0 to nl - 1 do
  for j = 0 to nc - 1 do
   m'.(i).(j) <- f(m.(i).(j))
  done
 done ; m' ;;
```

Question 3 Ici, il fallait bien faire attention au fait que p est un indice de ligne, donc une ordonnée !

```
let blit_matrix p q di dj =
 let nl = vect_length p and nc = vect_length p.(0) in
 for i = 0 to nl - 1 do
  for j = 0 to nc - 1 do
   q.(di+i).(dj+j) <- p.(i).(j)
  done
 done ;;
```

Question 4 S'il avait fallu réaliser la symétrie *in situ*, on aurait écrit une boucle échangeant (dans le cas de `sym_x`) les colonnes d'indices i et $nc-1-i$, pour $0 \leqslant i < nc/2$.

```
let sym_x m =
 let nl = vect_length m and nc = vect_length m.(0) in
 let m' = make_matrix nl nc  m.(0).(0) in
 for i = 0 to nl - 1 do
  for j = 0 to nc - 1 do
   m'.(i).(j) <-  m.(nl-1-i).(j)
  done
 done ; m' ;;

let sym_y m =
 let nl = vect_length m and nc = vect_length m.(0) in
 let m' = make_matrix nl nc m.(0).(0) in
 for i = 0 to nl - 1 do
  for j = 0 to nc - 1 do
   m'.(i).(j) <-  m.(i).(nc-1-j)
  done
 done ; m' ;;
```

Question 5 Notons $L_g(n)$ la longueur de bord totale des $4n$ masques requis pour la barication d'une puce à ADN régulière d'ordre n, avec la méthode d'implantation ≪Gray≫. Nous avons $L_g(1) = 8$; et $L_g(n+1) = 4L_g(n) + 8 \cdot 2^n$; en effet, les masques des $4n$ premières étapes exploitent la propriété des codes de Gray, si bien que les ≪raccords≫ entre eux ne coûtent rien. Le terme $8 \cdot 2^n$ provient des 4 derniers masques. La résolution de cette relation de récurrence est immédiate ; il vient $L_g(n) = 4^{n+1} - 2^{n+2}$.

Question 6 ⋆⋆ Notons $L_f(n)$ la longueur de bord totale des $4n$ masques requis pour la barication d'une puce à ADN régulière d'ordre n, avec la méthode d'implantation récursive. Nous avons $L_f(n+1) = 4L_f(n) + 8 \cdot 2^n + 8n \cdot 2^n$. Le premier terme est la contribution des $4n$ premiers masques, avant de les raccorder ; le deuxième terme est la contribution des quatre derniers masques ; enfin, le troisième terme provient des raccords entre les $4n$ premiers masques : on montre avec une récurrence immédiate que, par exemple, le bord inférieur de la zone nord-ouest et le bord supérieur de la zone sud-ouest sont deux mots de longueur 2^n dont la distance de Hamming est égale à 2^n. La résolution de cette relation de récurrence, avec la condition initiale $L_f(1) = 8$, nous donne $L_f(n) = 2 \cdot 4^{n+1} - (n+2)2^{n+2}$.

Question 7 La fonction `pref` servira aussi dans la questionsuivante.

```
let pref x l = x::l ;;

let rec make_fractal_chip = function
 | 1 -> let m = make_matrix 2 2 [A] in
    m.(0).(1) <- [C] ;
    m.(1).(0) <- [G] ;
    m.(1).(1) <- [T] ; m
 | n when n<1 -> failwith "n<1"
 | n -> let m' = make_fractal_chip (n-1) and s = puiss2 n in
   let m = make_matrix s s [] in
  blit_matrix (do_matrix (pref A) m') m 0 0 ;
  blit_matrix (do_matrix (pref C) m') m 0 (s/2) ;
  blit_matrix (do_matrix (pref G) m') m (s/2) 0 ;
  blit_matrix (do_matrix (pref T) m') m (s/2) (s/2) ; m
;;
```

Question 8 La seule différence avec la fonction précédente réside dans l'application des transformations `sym_x` et `sym_y`.

```
let rec make_gray_chip = function
 | 1 -> let m = make_matrix 2 2 [A] in
    m.(0).(1) <- [C] ;
    m.(1).(0) <- [G] ;
    m.(1).(1) <- [T] ; m
 | n when n<1 -> failwith "n<1"
 | n -> let m' = make_gray_chip (n-1) and s = puiss2 n in
   let m = make_matrix s s [] in
  blit_matrix (do_matrix (pref A) (copy_matrix m')) m 0 0 ;
  blit_matrix (sym_y(do_matrix (pref C) (copy_matrix m'))) m 0 (s/2) ;
  blit_matrix (sym_x(do_matrix (pref G) (copy_matrix m'))) m (s/2) 0 ;
  blit_matrix (sym_x(sym_y(do_matrix (pref T) (copy_matrix m')))) m (s/2) (s/2) ; m
;;
```

Question 9

```
let string_of_base = function
 | A -> "A"
 | C -> "C"
 | G -> "G"
 | T -> "T"
;;

let rec kth = function
 | ([],_) -> failwith "liste vide"
 | (t::_,0) -> t
 | (_::q,k) -> kth (q,k-1) ;;

let proj b = function
 | x when x = b -> string_of_base b
 | _ -> "." ;;
```

```
let masque b k m =
 let m' = do_matrix (fun x -> kth(x,k)) m in
 do_matrix (proj b) m' ;;
```

Question 10 Ce n'est pas vraiment un dessin...

```
let string_of_base_list l =
 it_list (prefix ^) "" (map string_of_base l) ;;

let print_chip m =
 let nl = vect_length m and nc = vect_length m.(0) in
 for i = 0 to nl - 1 do
  for j = 0 to nc - 1 do
   print_string((string_of_base_list m.(i).(j)) ^ " ")
  done ; print_newline()
 done ;;

let print_mask m =
 let nl = vect_length m and nc = vect_length m.(0) in
 for i = 0 to nl - 1 do
  for j = 0 to nc - 1 do
   print_string m.(i).(j)
  done ; print_newline()
 done ;;
```

Question 11 Fait unique, nous utilisons une référence ! Mais que je ne vous y prenne pas à faire la même chose trop souvent !

```
let mask_border_length m =
 let nl = vect_length m and nc = vect_length m.(0) in
 let k = ref 0 in
 for i = 0 to nl - 1 do
  for j = 0 to nc - 2 do
   if m.(i).(j) <> m.(i).(j+1) then k := !k + 1
  done ;
 done ;
 for i = 0 to nl - 2 do
  for j = 0 to nc - 1 do
   if m.(i).(j) <> m.(i+1).(j) then k := !k + 1
  done ;
 done ; !k ;;
```

Question 12 Pas trop dure, la dernière question !

```
let rec intervalle p q = if p>q then [] else p::(intervalle (p+1) q) ;;

let sum = it_list (prefix +) 0 ;;

let total_border_length m =
 let iv = intervalle 0 (list_length m.(0).(0) - 1) in
 let p b = map (fun x -> (b,x)) iv in
 let seq = (p A) @ (p C) @ (p G) @ (p T) in
 sum(map (fun (b,k) -> mask_border_length(masque b k m)) seq) ;;
```

FIN