Correction du DS 2

Julien Reichert

La correction des questions de cours étant dans le cours, elle ne sera pas donnée ici. De même, les exercices issus des TD et TP ont leur correction déjà publiée.

Exercices

Exercice 1 :

```
let indice_maximum t =
  let rep = ref 0 in
  for i = 1 to Array.length t - 1 do
    if t.(i) > t.(!rep) then rep := i
  done;
  !rep
```

Exercice 2 :

```
let plus_frequent s1 s2 c =
  let diff = ref 0 in
  for i = 0 to String.length s1 - 1 do
    if s1.[i] = c then incr diff
  done;
  for i = 0 to String.length s2 - 1 do
    if s2.[i] = c then decr diff
  done;
  !diff > 0
```

Exercice 3 :

```
let plus_grand_carre_ou_cube n =
  let racn = sqrt (float_of_int n) in
  let intracn = int_of_float racn in
  let raccubn = (float_of_int n) ** (1. /. 3.) in
  let intraccubn = int_of_float raccubn in
  let candidat1 = intracn * intracn in
  let candidat2 = intraccubn * intraccubn * intraccubn in
  max candidat1 candidat2
```

En pratique, le passage par les flottants occasionne certes une complexité constante (en admettant que les exponentiations soient de coût unitaire), mais apporte un risque d'erreur de précision qu'on peut corriger en testant aussi le carré et le cube des deux voisins de l'entier considéré et parmi les six nombres obtenus on gardera le plus grand qui soit inférieur ou égal à l'argument.

Bien entendu, il restera la possibilité de faire une ou deux boucle(s) conditionnelle(s) et de ne travailler qu'avec des entiers.

Exercice 4 :

```
let dist_min_occ tab x =
  let dernier = ref (-1) in
  let distmin = ref (-1) in
  for i = 0 to Array.length tab - 1 do
    if tab.(i) = x then
      (
        if !dernier <> -1 && (i - !dernier < !distmin || !distmin = -1) then distmin := i - !dernier;
        dernier := i
      )
  done;
  !distmin
```

Exercice 5 :

La terminaison est triviale en raison de l'utilisation d'une boucle inconditionnelle, inutile d'utiliser un variant ici.

Concernant la correction, on utilise l'invariant de boucle suivant : « `dernier` mémorise le dernier indice où `x` se trouve dans `tab` ou `-1` s'il n'y en a pas eu et `distmin` mémorise le plus petit écart entre deux occurrences précédentes de `x` dans `tab` ou `-1` s'il n'y en a pas encore eu deux, le tout jusqu'à l'indice en cours de traitement ».

Cet invariant est vrai avant d'entrer dans la boucle car les deux références ont été initialisées à `-1` et aucun indice du tableau n'a encore été étudié, il est utile car le fait qu'il soit vrai après la boucle permet de conclure que `distmin`, que l'on renvoie, mémorise l'information qu'il fallait calculer, et il est héréditaire au vu du corps de boucle : `dernier` est mis à jour si à l'indice traité on découvre `x`, et `distmin` est mis à jour si de plus `dernier` ne valait pas `-1` (ce qui veut dire que l'occurrence de la valeur `x` que l'on vient de découvrir n'était pas la première) et que l'écart entre les deux dernières occurrences rencontrées, qu'on obtient en soustrayant à l'indice actuel la valeur mémorisée en `dernier`, est inférieur à la valeur actuellement en `distmin` ou qu'il y avait encore `-1`.

Exercice 6 :

```
let rec ecartmincouples l = match l with
| [] -> failwith "Liste vide"
| [(a, b)] -> b -. a
| (a, b)::q -> min (b -. a) (ecartmincouples q)
```

Exercice 7 :

```
let rec phases l = match l with
| [] -> 0
| [_] -> 1
| a::b::q when a = b -> phases (b::q)
| a::b::q ->
  let rec phasesaux cro liste = match liste with
  | [] -> failwith "Ce cas ne peut pas se produire"
  | [_] -> 1
  | a::b::q when a = b -> phasesaux cro (b::q)
  | a::b::q when (a < b) = cro -> phasesaux cro (b::q)
  | a::l -> 1 + phasesaux (not cro) l
  in phasesaux (a < b) (b::q)
```

Problème

Question P1 :

Puisqu'une valeur entre `0` et `255` tient sur un octet, chaque pixel prend trois octets et la taille en mémoire pour une image de n lignes et m colonnes est de $3mn$ octets.

Question P2 :

En appliquant la formule précédente, on obtient un rapport de $\frac{1858000}{24000000}$ soit un peu moins d'un douzième, mais l'exemple est totalement fictif.

Question P3 :

Tout simplement (penser à l'utilisation de la partie entière en mathématiques) :

```
let representant n = n / 16 * 16
```

Question P4 :

Toujours d'après ce qui précède, il y a $3mn$ octets dont la moitié des bits est disponible pour cacher des informations, soit moitié autant d'octets. Avec les valeurs fournies, cela donne donc douze millions d'octets (dont un à la fin pour signaler la fin du fichier).

Question P5 :

En gardant le même principe de séquence de données sans tenir compte de la largeur et de la hauteur de l'image à cacher, le nombre de pixels que l'on peut cacher est la moitié du nombre de pixels dans l'image support. Avec les mêmes dimensions, le nombre de lignes et de colonnes subissent la même réduction d'un facteur $\sqrt{2}$.

Question P6 :

L'utilisation de fonctions auxiliaires est recommandée ici. Il n'est certes pas nécessaire d'en écrire autant que ci-après mais bon...

```
let paquet4 sequence refind =
  let rep = ref 0 in
  try
    for i = 0 to 3 do
      rep := 2 * !rep;
      if sequence.(!refind) then incr rep;
      incr refind
    done;
    (!refind < Array.length sequence, !rep)
  with _ -> (false, !rep)

let trois_paquets sequence refind =
  let (_, rbis) = paquet4 sequence refind in
  let (_, vbis) = paquet4 sequence refind in
  let (test, bbis) = paquet4 sequence refind in
  (* Ne pas l'écrire en une fois, vu l'effet secondaire.
  On rappelle que les triplets s'évaluent de droite à gauche. *)
  (test, (rbis, vbis, bbis))

let nv_trip (r, v, b) (rbis, vbis, bbis) =
  (representant r + rbis, representant v + vbis, representant b + bbis)
```

```
let code matrice sequence =
  let n = Array.length matrice in let m = Array.length matrice.(0) in (* Tant pis si n = 0 ! *)
  let mat2 = Array.make_matrix n m (0, 0, 0) in
  let indseq = ref 0 in
  let modifier = ref true in
  for i = 0 to n-1 do
    for j = 0 to m-1 do
      let trip = matrice.(i).(j) in
      if !modifier then
      (
        let (test, tripbis) = trois_paquets sequence indseq in
        if not test then modifier := false;
        mat2.(i).(j) <- nv_trip trip tripbis
      )
      else mat2.(i).(j) <- matrice.(i).(j)
    done
  done;
  if !modifier then failwith "La séquence n'est pas finie"
  else mat2
```

Question P7 :

```
let action entier buffer parite chaine =
  buffer := 16 * !buffer + entier;
  if !parite then
  ( chaine := !chaine ^ String.make 1 (char_of_int !buffer);
    let code_reponse = !buffer <> 0 in
    buffer := 0; parite := false; code_reponse )
  else ( parite := true; true )

exception Return of string

let decode mat2 =
  let n = Array.length mat2 in let m = Array.length mat2.(0) in
  let res = ref "" in let parite = ref false in let buff = ref 0 in
  try
    for i = 0 to n-1 do
      for j = 0 to m-1 do
        let r, v, b = mat2.(i).(j) in
        let tableau = [| r mod 16 ; v mod 16 ; b mod 16 |] in
        for k = 0 to 2 do
          let continuer = action tableau.(k) buff parite res in
          if (not continuer) then raise (Return !res)
        done
      done
    done; failwith "Chaîne mal formatée"
  with Return s -> s
```

Question P8 :

```
./g > p.txt
```

Question P9 : Les lettres éparpillées en police sans serif et écrites dans une autre nuance de gris ainsi que certains nombres ou noms suggérés étaient des indices faisant référence à Giacomo Puccini.