Correction du DS 2

Option informatique, première année

Julien Reichert

Exercice 1

```
(* somme_tab : int array -> int *)
let somme_tab t =
  let rec aux deb fin =
    if deb > fin then 0
    else if deb = fin then t.(deb)
    else let mil = (deb+fin)/2 in aux deb mil + aux (mil+1) fin
  in aux 0 (Array.length t - 1);;

(* max_tab : int array -> int *)
let max_tab t =
  let rec aux deb fin =
    if deb >= fin then t.(deb)
    else let mil = (deb+fin)/2 in max (aux deb mil) (aux (mil+1) fin)
  in aux 0 (Array.length t - 1);;
(* Si le tableau est vide, on obtient une erreur, c'est volontaire de ma part *)
```

Exercice 2

La multiplication matricielle sera limitée à des matrices (2, 2) ici.

```
(* mm22 : int array array -> int array array -> int array array *)
let mm22 m n =
  [|[|m.(0).(0)*n.(0).(0) + m.(0).(1)*n.(1).(0); m.(0).(0)*n.(0).(1) + m.(0).(1)*n.(1).(1)|];
    [|m.(1).(0)*n.(0).(0) + m.(1).(1)*n.(1).(0); m.(1).(0)*n.(0).(1) + m.(1).(1)*n.(1).(1)|]|];;

(* puiss_rapide_22 : int array array -> int -> int array array *)
let rec puiss_rapide_22 m k =
  if k = 0 then [|[|1; 0|]; [|0; 1|]|]
  else if k mod 2 = 1 then mm22 m (puiss_rapide_22 (mm22 m m) (k/2))
  else puiss_rapide_22 (mm22 m m) (k/2);;

(* fibonacci : int -> int *)
let fibonacci n = let mat = puiss_rapide_22 [|[|0; 1|]; [|1; 1|]|] n in mat.(0).(1);;
```

Exercice 3

```
type 'a arbin = Vide | Noeud of 'a arbin * 'a * 'a arbin;;

(* profondeur : 'a arbin -> int *)
let rec profondeur = function
| Vide -> -1
| Noeud(g, _, d) -> 1 + max (profondeur g) (profondeur d);;
```

Exercice 4

Il s'agit du nombre de branches non triviales d'un arbre binaire, dans la mesure où dans chaque nœud finissent autant de branches que la profondeur du nœud (une par ancêtre qui en serait le point de départ), à l'exclusion des branches réduites à un seul nœud (on pourrait les incorporer en initialisant la variable `i` à `1`, ce qui reviendrait à ajouter la taille de l'arbre). La somme des profondeurs de chaque nœud de l'arbre est donc égale à l'information attendue (l'énoncé demandant la mention explicite de la notion de branche).

Exercice 5

```
type arpasbin = N of int * arpasbin list;;

(* somme_arpasbin : arpasbin -> int *)
let rec somme_arpasbin (N(n, l)) =
  n + List.fold_left (fun total arbre -> total + somme_arpasbin arbre) 0 l;;
```

Exercice 6

```
(* somme_enfants : arpasbin list -> int *)
let somme_enfants l = List.fold_left (fun total (N(n, _)) -> total + n) 0 l;;

(* verif_arpasbin : arpasbin -> bool *)
let rec verif_arpasbin (N(n, l)) =
  n = somme_enfants l && List.for_all verif_arpasbin l;;
```

Exercice 7

Plutôt que d'improviser un programme en cherchant la formule directement, mieux vaut s'intéresser à l'exemple fourni.

On observe que la taille du nouveau tableau est le nombre de lignes (noté ici `n`) plus le nombre de colonnes (noté ici `m`) moins un. En considérant les indices dans le tableau de départ et le tableau d'arrivée, il s'avère que dans le `k`-ième tableau de la réponse on retrouve les éléments dont la somme des deux indices du tableau fourni est `k` (une justification du nombre de lignes total). Leur nombre est au plus le minimum entre `n` et `m` du tableau fourni, en l'occurrence cela commence à un, augmente d'un à chaque étape, atteint provisoirement ce minimum et finit par diminuer d'un jusqu'à revenir à un, d'où l'initialisation étape par étape. L'algorithme est linéaire en la taille du tableau (le produit de `n` et `m`), et on évitera de faire pour chacune des `m + n - 1` lignes du tableau d'arrivée une boucle testant toutes les façons d'obtenir une somme égale à l'indice de la ligne, qui donnerait une complexité quadratique en `m + n` (mais ce ne serait pas pénalisé).

```
(* huitieme_de_tour : 'a array array -> 'a array array *)

let huitieme_de_tour tab =
  let n = Array.length tab and let m = Array.length tab.(0) in (* non nuls, promis *)
  let defaut = tab.(0).(0) in (* pour le typage *)
  let mini = min m n and somme = m + n - 1 in
  let reponse = Array.make somme [||] and taille = ref 1 in
  for i = 0 to mini-1 do reponse.(i) <- Array.make !taille defaut;
    reponse.(somme-1-i) <- Array.make !taille defaut; incr taille done;
  for i = mini to somme-mini-1 do reponse.(i) <- Array.make mini defaut done;
  for indice = 0 to somme-1 do
    let j = ref (max 0 (indice-n+1)) in
    for k = 0 to Array.length reponse.(indice)-1 do
      reponse.(indice).(k) <- tab.(indice - !j).(!j); incr j
    done
  done; reponse;;
```

Exercice 8

```
(* nombre_de_lettres : int -> (int * int) *)
let nombre_de_lettres n =
  let rec aux k vspk somme = (* vspk : _v_ingt-_s_ix _p_uissance _k_ *)
    if somme+vspk >= n then (k, somme)
    else aux (k+1) (26*vspk) (somme + vspk)
  in aux 1 26 0;;

(* clastre_nom : int -> string *)
let clastre_nom n =
  let k, superieurs = nombre_de_lettres n in
  let reste = ref (n - superieurs - 1) in
  let nom_nombres_tab = Array.make k 0 in
    for i = k-1 downto 0 do
      nom_nombres_tab.(i) <- !reste mod 26;
      reste := !reste / 26
    done;
  String.init k (fun i -> char_of_int (nom_nombres_tab.(i) + 65));;

(* sommepuiss : int -> int -> int *)
let sommepuiss vs k = (* somme des vs puissance i pour i de 1 à k inclus *)
  let rec aux i vspk somme =
    if i = k then somme
    else aux (i+1) (vspk*vs) (somme + vspk)
  in aux 0 vs 0;;

(* clastre_rang : string -> int *)
let clastre_rang nom =
  let k = String.length nom in
  let superieurs = ref (sommepuiss 26 (k-1))
  and vspi = ref 1
  and indice = ref (k-1) in
  while !indice >= 0 do
    superieurs := !superieurs + !vspi * (int_of_char nom.[!indice] - 65);
    vspi := !vspi * 26;
    decr indice
  done; !superieurs + 1;;

(* Autre version, grâce à la perspicacité de Tom en 2020 : *)
let clastre_rang_facile nom =
  let rang = ref 0 and vspi = ref 1 in
    for i = String.length nom - 1 downto 0 do
      rang := !rang + (int_of_char nom.[i] - 64) !vspi;
      vspi := !vspi * 26
    done;
  !rang;;

(* Astuce : A compte pour 1 au lieu de 0, de sorte qu'il y ait une augmentation de 26 puissance i
pour chaque indice, correspondant à la somme du nombre de personnes ayant strictement moins de lettres,
dont le 26 puissance 0 car le meilleur a pour rang 1 et non 0 *)
```

Exercice 9

```
let rec formule_vers_arbre = function
| Vrai -> Feuille(V)
| Faux -> Feuille(F)
| Var(b) -> Feuille(Vb(b))
| Non(f) -> Noeud(Not, [formule_vers_arbre f])
| Et(l) -> Noeud(And, List.map formule_vers_arbre l)
| Ou(l) -> Noeud(Or, List.map formule_vers_arbre l);;

let rec arbre_vers_formule = function
| Feuille(V) -> Vrai
| Feuille(F) -> Faux
| Feuille(Vb(b)) -> Var(b)
| Noeud(Not, [f]) -> Non(arbre_vers_formule f)
| Noeud(And, l) -> Et(List.map arbre_vers_formule l)
| Noeud(Or, l) -> Ou(List.map arbre_vers_formule l)
| _ -> failwith "Noeud not avec un nombre différent d'un fils";;
```

La partie logique de CCINP n'est pas corrigée ici.