Correction du DS 3

Julien Reichert

Questions de cours

Question de cours A1 : Le $2n+3$ est vu comme un $\Theta(n)$, ce qui donne la formule classique des tris DPR comme le tri fusion ou le tri rapide avec recherche optimisée de pivot. La complexité associée est alors $\Theta(n \log n)$. On pourra remplacer le Θ par un $\mathcal{O}$ en utilisant l'abus usuel de considérer que cela suggère l'optimalité.

Question de cours A2 : La structure de données abstraite de liste chaînée est une structure dont chaque élément pointe vers le suivant (et le dernier ne pointe sur rien). La liste chaînée elle-même est simplement un pointeur vers le premier élément. Les opérations associées peuvent être diverses, mais on retiendra la possibilité en les utilisant de parcourir l'ensemble de la liste et d'insérer ou de retirer un élément à n'importe quelle position.

Question de cours A3 : Une première implémentation consiste à utiliser un tableau circulaire associé à deux ou trois informations afin de savoir à quelle position effectuer le prochain enfilement (sous réserve de place car dans ce cas la capacité est limitée) et à quelle position effectuer le prochain défilement (sous réserve que la taille, nécessairement une information à maintenir, ne soit pas zéro). Pour ces deux opérations, quand on dépasse à droite en avançant dans le tableau, on revient à l'indice zéro. Le test de vacuité est alors gratuit, et la création revient à initialiser la structure contenant toutes ces informations.

Une deuxième implémentation consiste à utiliser deux piles, une où se feront tous les enfilements, et une où se feront tous les défilements, en transférant (forcément à l'envers) tous les éléments de la pile d'entrée dans la pile de sortie lorsqu'un défilement est nécessaire alors que la pile de sortie est vide. La création est alors simplement une création de deux piles et le test de vacuité vérifie que les deux piles sont vides.

Questions de cours en OCaml

Question de cours B1 :

```
let carte_depuis_entier n = assert (0 <= n && n < 32); { valeur = n / 4 ; couleur = n mod 4 };;
```

Question de cours B2 :

```
assert false;;
```

Question de cours B3 :

```
let est_part_ent_et_different n x = float_of_int n < x && x < float_of_int (n+1);;
```

Questions de cours en C

Question de cours C1 :

Les accolades sont nécessaires pour éviter que la variable `i` ait une portée différente.

```
{
  int i = 0;
  while condition(i)
  {
    corps(i);
    maj(&i);
  }
}
```

Question de cours C2 :

```
char f(int t[], int n)
{
  char* aux = malloc(n+1);
  /* Ici un code qui travaille sur aux et qui ne nous intéresse pas */
  char rep = aux[0];
  free(aux);
  return rep;
}
```

Question de cours C3 :

L'option est `-Wall`.

Exercices en OCaml

Exercice 1 :

```
let dist (i1, j1) (i2, j2) =
  abs (i2 - i1) + abs (j2 - j1);;

let calendrier_avent mat =
  let nl = Array.length mat in
  let nc = Array.length mat.(0) in
  let n = nl * nc in assert (n > 1);
  let pos = Array.make n (0, 0) in
  for i = 0 to nl - 1 do
    for j = 0 to nc - 1 do
      pos.(mat.(i).(j)) <- (i, j)
    done
  done;
  let mini = ref (dist pos.(0) pos.(1)) in
  let maxi = ref (dist pos.(0) pos.(1)) in
  for i = 1 to n-2 do
    let d = dist pos.(i) pos.(i+1) in
    if d < !mini then mini := d
    else if d > !maxi then maxi := d
  done;
  (!mini, !maxi);;
```

Exercice 2 :

On réutilise la fonction `dist` de l'exercice précédent.

```
let maj l mini maxi =
  let rec aux l1 l2 = match l1, l2 with
  | [], _ -> ()
  | [_], [] -> ()
  | a::b::q, [] -> aux (b::q) q
  | a::q, b::qq ->
    let d = dist a b in
    if !mini = -1 || d < !mini then mini := d
    else if d > !maxi then maxi := d;
    aux (a::q) qq
  in match l with
  | [] -> ()
  | a::q -> aux l q;;

let dist_egaux mat =
  let nl = Array.length mat in
  let nc = Array.length mat.(0) in
  let n = nl * nc in assert (n > 1);
  let pos = Hashtbl.create n in
  for i = 0 to nl - 1 do
    for j = 0 to nc - 1 do
      let x = mat.(i).(j) in
      if Hashtbl.mem pos x
        then Hashtbl.replace pos x ((i, j)::Hashtbl.find pos x)
      else
        Hashtbl.add pos x [(i, j)]
    done
  done;
  let mini = ref (-1) and maxi = ref (-1) in
  Hashtbl.iter (fun _ l -> maj l mini maxi) pos;
  (!mini, !maxi);;
```

Exercice 3 : (On pourra encapsuler ceci dans une fonction qui vérifie d'abord que les deux entiers sont positifs, éventuellement nuls.)

```
let rec encheres f1 f2 =
  if f1 = f2 then (0, 0)
  else if f1 > f2 then let (r1, r2) = encheres (f1 - f2 - 1) f2 in (r1 + 1, r2)
  else let (r1, r2) = encheres f1 (f2 - f1 - 1) in (r1, r2 + 1);;
```

Exercices en C

Exercice 4 :

```
int* map(int (*f)(int), int tab[], int taille)
{
  int* rep = malloc(taille * sizeof(int));
  for (int i = 0 ; i < taille ; i += 1) rep[i] = f(tab[i]);
  return rep;
}

int carre(int x) { return x * x; }
```

```
int main()
{
  int t[10] = { 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 };
  int* tcarre = map(carre, t, 10);
  for (int i = 0 ; i < 10 ; i += 1)
  {
    printf("%d ", tcarre[i]);
  }
  free(tcarre);

  return 0;
}
```

Exercice 5 :

```
int motus(char* s1, char* s2, int* bp, int* mp)
{
  int occurrences1[256] = { 0 };
  int occurrences2[256] = { 0 };
  int n = strlen(s1);
  for (int i = 0 ; i < n ; i += 1)
  {
    if (s1[i] == s2[i]) *bp += 1;
    else
    {
      occurrences1[(int) s1[i]] += 1;
      occurrences2[(int) s2[i]] += 1;
    }
  }
  for (int i = 1 ; i < 256 ; i += 1)
  {
    if (occurrences1[i] <= occurrences2[i]) *mp += occurrences1[i];
    else *mp += occurrences2[i];
  }
}

int main()
{
  char s1[] = "ECREME";
  char s2[] = "RECURE";
  int bp = 0;
  int mp = 0;
  motus(s1, s2, &bp, &mp);
  printf("%d et %d\n", bp, mp); // "1 et 3"

  return 0;
}
```

Exercice 6 : Il s'agit de la fonction `medianemax`, numéro 44 de la liste des exercices de l'année. En raison de la création du tableau des occurrences du maximum, la complexité en espace est linéaire en la taille de la liste (borne supérieure pouvant être atteinte). Ceci permet de limiter à un seul passage dans le tableau, mais il aurait mieux valu faire deux passages et avoir une complexité constante en espace (juste les variables créées). En attendant, il manque la libération de la mémoire entre la création de la réponse et la ligne où elle est renvoyée : `free(a);`.

Problème

Question P1 : La condition nécessaire et suffisante est que la propriété soit vraie au moins une fois, c'est-à-dire au dernier indice vu qu'une fois qu'elle est vraie elle le reste.

Question P2 : Pour des tableaux de taille au plus deux c'est toujours vrai (vérification évidente), mais à partir de la taille trois ce n'est plus garanti, et un contre-exemple facile à trouver est un tableau représentant une permutation circulaire.

Question P3 : Le contre-exemple précédent reste valable.

Programmation en OCaml

Question P4 :

```
let premier_inf t =
  let n = Array.length t in assert (t.(n-1) < n-1);
  if t.(0) < 0 then 0 else
  begin
    let debut = ref 0 and fin = ref (n-1) in
    while !debut < !fin - 1 do
      let milieu = (!debut + !fin) / 2 in
      if t.(milieu) > milieu then debut := milieu
      else if t.(milieu) < milieu then fin := milieu
      else (fin := milieu + 1; debut := milieu + 1)
    done; !fin
  end;;
```

Question P5 :

```
exception Trouve of int;;

let compagnons t =
  let n = Array.length t in
  try
    for i = 0 to n-1 do
      let ti = t.(i) in
      if 0 <= ti && ti < n && t.(ti) = i
        then raise (Trouve i)
    done; (-1, -1)
  with Trouve i -> i, t.(i);;
```

Question P6 : On procède par dichotomie. Dans les conditions données ici, la valeur `t.(t.(i)) - i`, là où elle est autorisée, est strictement décroissante.

Une fois qu'on a délimité la zone (par une dichotomie également) où `t.(i)` est entre zéro et la taille du tableau moins un, on cherche le moment où cette valeur passe dans les négatifs.

Par unicité en cas d'existence, on trouve la seule possibilité et on vérifie que c'est le cas.

Astuce pour ne pas avoir à délimiter la zone en pratique : on peut arbitrairement considérer qu'il faut aller à droite si au milieu `t.(i)` est trop grand pour être un indice et à gauche s'il est trop petit.

```
exception Trouve_dicho of int;;

let compagnons_decr t =
  let n = Array.length t in
  let deb = ref 0 and fin = ref (n-1) in
  try
    while !deb <= !fin do
      let milieu = (!deb + !fin) / 2 in
      if t.(milieu) < 0 then fin := milieu - 1
      else if t.(milieu) >= n then deb := milieu + 1
      else if t.(t.(milieu)) = milieu
      then raise (Trouve_dicho milieu)
      else if t.(t.(milieu)) > milieu then deb := milieu + 1
      else fin := milieu - 1
    done; (-1, -1)
  with Trouve_dicho i -> (i, t.(i));;
```

Question P7 : De manière analogue à l'exercice 54 de la liste, on doit chercher la taille des cycles dans la décomposition en cycles de la permutation. La réponse est le PPCM des tailles.

```
let rec pgcd a b =
  if a = 0 && b = 0 then failwith "PGCD de 0 et 0";
  if b = 0 then a else pgcd b (a mod b);;

let ppcm a b = a * b / pgcd a b;; (* tout est positif ici *)

let ppcm_liste l = List.fold_left ppcm 1 l;;

exception Stop;;

let ordre tab =
  let n = Array.length tab in
  let dejavu = Array.make n false in
  let indice = ref 0 in
  let ordres_cycles = ref [] in (* bouh la référence de liste ! *)
  try
    while true do
      while !indice <> n && dejavu.(!indice) do incr indice done;
      if !indice = n then raise Stop;
      dejavu.(!indice) <- true;
      let taille = ref 1 and position = ref tab.(!indice) in
        while not dejavu.(!position) do
          dejavu.(!position) <- true;
          incr taille;
          position := tab.(!position)
        done;
        assert (!position = !indice); (* Vérification qu'on a une bijection *)
        ordres_cycles := !taille :: !ordres_cycles;
        incr indice
    done; failwith "On n'arrive jamais ici"
  with Stop -> ppcm_liste !ordres_cycles;;
```

Programmation en C

Question P8 :

```
int point_fixe(int* t, int taille)
{
  int rep = 0;
  while (rep < taille && t[rep] != rep) rep += 1;
  if (rep == taille) rep = -1;
  return rep;
}
```

La complexité est linéaire.

Question P9 : On peut envisager d'optimiser le programme pour faire une dichotomie (cf. P10).

Ici le principal intérêt de l'hypothèse est qu'on a la garantie d'existence d'un indice où `t[i]` est égal à `i`, car la valeur `t[i] - i` est initialement positive (ou nulle), à la fin elle est négative (ou nulle) et à chaque étape soit elle diminue d'un, soit elle reste la même, soit elle augmente.

Dans tous les cas, en admettant que le premier et le dernier indices ne donnent pas lieu à une valeur nulle (donc un point fixe), le passage de la zone des valeurs strictement positives à la zone des valeurs strictement négatives ne peut se faire que par une décrémentation, donc on ne peut pas sauter par-dessus zéro.

Pour être complet, le passage de la zone des valeurs strictement négatives à la zone des valeurs strictement positives peut se refaire par la suite (sans forcément passer par une valeur nulle), ce qui donnerait forcément lieu à l'apparition d'une deuxième valeur nulle.

Question P10 : Ici aussi on a une information : il ne peut y avoir qu'un point fixe au plus. Pour le déterminer, une dichotomie s'impose, donnant lieu à une complexité logarithmique en la taille du tableau.

Par ailleurs, l'existence n'est plus garantie, on peut prendre pour contre-exemple le tableau restreint à `1` et `0`.

```
int point_fixe_dicho(int* t, int taille)
{
  int deb = 0;
  int fin = taille - 1;
  while (deb < fin)
  {
    int milieu = (deb + fin) / 2;
    if (t[milieu] == milieu) return milieu;
    if (t[milieu] > milieu) deb = milieu + 1;
    else fin = milieu - 1;
  }
  assert(false);
}
```