Correction du TP3 : Structures de données

Implémentation des files en OCaml à l'aide de tableaux

1.

```
type 'a file_array = {contenu : 'a array ; mutable debut : int ;
                      mutable fin : int ; mutable vide : bool}
;;

let file_ex = {contenu = [|9;5;7;3;7;2;0;1;6;1|] ; debut = 6 ;
               fin = 1 ; vide = false}
;;
```

2.

```
let est_vide f = f.vide ;;

est_vide file_ex ;;
- : bool = false
```

3.

```
let est_pleine f = (f.fin = f.debut) && (not f.vide) ;;

est_pleine file_ex ;;
- : bool = false
```

4.

```
let enfile e f =
  if est_pleine f
    then failwith "file pleine"
    else
      begin
        f.contenu.(f.fin) <- e;
        f.fin <- (f.fin+1) mod (Array.length f.contenu);
        f.vide <- false;
        f;
      end
;;

enfile 6 file_ex ;;
- : int file = {contenu = [|9;6;7;3;7;2;0;1;6;1|] ; debut = 6 ;
                fin = 2 ; vide = false}
```

5.

```
let defile f =
  if est_vide f
    then failwith "file vide"
    else
      begin
        f.début <- (f.debut+1) mod (Array.length f.contenu);
        f.vide <- (f.debut = f.fin);
        f;
      end
;;

defile file_ex ;;
- : int file = {contenu = [|9;6;7;3;7;2;0;1;6;1|] ; debut = 7 ;
               fin = 2 ; vide = false}
```

6.

```
let taille f =
  if est_vide f
    then 0
    else
      begin
        let t = f.fin - f.debut in
          if t > 0
            then t
            else t + (Array.length f.contenu)
      end
;;

taille file_ex ;;
- : int = 5
```

7.

```
let appartient x f =
  if est_vide f
    then false
    else
      begin
        let i = ref f.debut and n = Array.length f.contenu in
          while ( !i < f.fin) && (f.contenu.( !i) <> x) do
            i := ( !i+1) mod n
          done;
          !i <> f.fin
      end
;;

appartient 6 file_ex ;;
- : bool = true
```

Implémentation des files en OCaml à l'aide de listes

8.

```
type 'a file_list = {mutable sortie : 'a list ;
                     mutable entree : 'a list}
;;

let file_ex2 = {sortie = [7;6;5;4] ; entree = [1;2;3]}
;;
```

9.

```
let est_vide f = (f.entree = []) && (f.sortie = []) ;;

est_vide file_ex2 ;;
- : bool = false
```

10.

```
let enfile e f =
  f.entree <- e::f.entree;
  f
;;

enfile 8 file_ex2 ;;
- : int file_list = {sortie = [7;6;5;4] ; entree = [8;1;2;3]}
```

11.

```
let rec aux f = match f.entree with
  | []   -> ()
  | t::q -> f.sortie <- t::f.sortie;
            f.entree <- q;
            aux f
;;

let normalise f = if f.sortie = [] then aux f
;;
```

La fonction auxiliaire `aux` se contente d'effectuer le retournement de la liste `entree` dans la liste `sortie`. Cette modification n'est faite que si la liste `sortie` est vide.

12.

```
let defile f =
  normalise f;
  match f.sortie with
    | []   -> failwith "file vide"
    | t::q -> f.sortie <- q; t
;;

defile file_ex2 ;;
- : int = 7
```

13.

```
let rec taille f = match (f.sortie,f.entree) with
  | ([],[])     -> 0
  | ([],t2::q2) -> 1 + taille {sortie = []; entree = q2}
  | (t1::q1,l2) -> 1 + taille {sortie = q1; entree = l2}
;;

taille file_ex2 ;;
- : int = 7
```

14.

```
let rec appartient x f = match (f.sortie,f.entree) with
  | ([],[])     ->false
  | ([],t2::q2) -> (t2 = x) || appartient x {sortie=[]; entree = q2}
  | (t1::q1,l2) -> (t1 = x) || appatient x {sortie=q1; entree = l2}
;;

appartient 4 file_ex2 ;;
- : bool = true
```

Implémentation des dictionnaires en OCaml

15.

```
let creer_dico m = {h = (function x -> x mod m) ;
                    tbl = Array.make m []} ;;
```

16.

```
let chercher t c =
  let rec aux l = match l with
    | [] -> failwith "pas trouve"
    | t::q when t.cle = c -> t.valeur
    | _::q -> aux q
in aux (t.tbl).(t.h c)
;;
```

17.

```
let inserer t e =
  let i = t.h e.cle in
    t.tbl.(i) <- e::(t.tbl.(i))
;;
```

18.

```
let supprimer t c =
  let i = t.h c in
    let rec aux l = match l with
      | [] -> []
      | t::q when t.cle = c -> q
      | t::q -> t::(aux q)
  in t.tbl.(i) <- aux t.tbl.(i)
;;
```

19. Les valeurs de hachage sont les suivantes (sous les collisions sont indiquées les résolutions après sondage) :

c	5	28	19	15	20	33	12	17	10
c mod 9	5	1	1	6	2	6	3	8	1
			2		3	7	4		0

La répartition des clés est donc la suivante :

10	28	19	20	12	5	15	33	17

20.

```
let creer_dico m = {h = (function x -> x mod m) ;
                    tbl = Array.make m Vide} ;;
```

21. On définit ainsi les fonctions demandées :

```
let chercher t c =
  let rec sonde i = match t.tbl.(i) with
    | Vide -> failwith "pas trouve"
    | Element e when e.cle = c -> e.valeur
    | _ -> sonde (t.h (i+1))
  in sonde (t.h c)
;;

let inserer t e =
  let rec sonde i = match t.tbl.(i) with
    | Vide -> t.tbl.(i) <- Element e
    | _ -> sonde (t.h (i+1))
  in sonde (t.h e.cle)
;;
```

22. Si certaines clés sont supprimées de la table, un sondage risque de se terminer avant qu'une clé soit trouvée, même si celle-ci figure dans la table. Pour remédier à ce problème, une solution possible consiste pour une recherche à distinguer les cases

vides (qui occasionnent la fin du sondage) des cases effacées (qui ne mettent pas fin au sondage). Concrètement, il faut redéfinir le type `objet` de la façon suivante :

```
type objet = Vide | Efface | Element of element ;;
```

La fonction `chercher` n'a pas besoin d'être redéfinie, et on lui ajoute :

```
let inserer t e =
  let rec sonde i = match t.tbl.(i) with
    | Vide -> t.tbl.(i) <- e
    | Efface -> t.tbl.(i) <- e
    | _ -> sonde (t.h (i+1))
  in sonde (t.h e.cle)
;;

let supprimer t c =
  let rec sonde i = match t.tbl.(i) with
    | Vide -> ()
    | Element e when e.cle = c -> t.tbl.(i) <- Efface
    | _ -> sonde (t.h (i+1))
  in sonde (t.h c)
;;
```

23. Si on suppose le hachage uniforme, la probabilité qu'une case précédée d'une case vide soit désignée pour accueillir une clé donnée est égale à $\frac{1}{m}$, tandis qu'une case vide précédée de k cases pleines a une probabilité égale à $\frac{k+1}{m}$, puisque la désignation de l'une quelconque des k cases précédentes affecte la clé à cette case.

    Ceci a pour conséquence que plus la succession de cases pleines est longue, plus grande est la probabilité d'agrandir encore cette chaîne, ce qui conduit à la formation d'agrégats. Or plus longue est la chaîne, plus long est le sondage ; ce phénomène contribue donc à ralentir la lecture et l'ajout dans un dictionnaire.