▶ **MP1 – Option Informatique**
1er TP Caml
sl@jachiet.com
http://louis.jachiet.com/tps
Vendredi 27 septembre 2013

Listes, tableaux et tris

1 Listes

1.1 tri par insertion

▶ **Question 1** *Écrire une fonction*

```
insere : 'a -> 'a list -> 'a list
```

qui étant donnés un élément x et une liste triée $[a_1; ...; a_n]$ avec $a_1 \leq a_2 \leq ... \leq a_n$ doit renvoyer $[a_1; ...; a_i; x; a_{i+1}; ...; a_n]$ avec $a_i \leq x \leq a_{i+1}$. En déduire une fonction

```
tri : 'a list -> 'a list
```

qui doit renvoyer la liste triée. On pourra utiliser it_list ou faire une fonction à deux arguments : le premier est un bout de liste déjà trié, le second est ce qu'il reste à trier.

1.2 tri fusion

▶ **Question 2** *Écrire les fonctions :*

```
partition : 'a list -> 'a list * 'a list
```

qui, d'une liste, fabrique une bi-partition la plus équilibrée possible ;

```
reunit : 'a list * 'a list -> 'a list
```

qui fabrique une liste triée à partir de deux listes triées. En déduire la fonction :

```
tri_fusion : 'a list -> 'a list
```

▶ **Question 3**
La fonction :

```
random__int : int -> int
```

prend en paramètre un entier n et renvoie un nombre aléatoire entre 0 et n.
Utilisez cette fonction pour générer des listes aléatoires d'une taille donnée et vérifiez que vos tris trient bien.

▶ **Question 4 *** *Quel est le meilleur des deux algorithmes ? Essayez d'évaluer le nombre d'opérations que va effectuer chacun des algorithmes. Mettez le sous la forme $O(f(n))$, pour de bons f.*

1.3 Tri par sélection

▶ **Question 5** *Le tri par sélection fonctionne de la manière suivante : on extrait le minimum de la liste que l'on veut trier, on trie la liste restante, on ajoute le minimum au début du reste trié.*

2 Tableaux

▶ **Question 6** ***Tri a bulle*** *Faites un programme qui prend un tableau et qui renvoie le tableau trié par la méthode du tri à bulles. (Tant que le tableau n'est pas trié on le parcourt et on inverse les paires d'éléments consécutifs qui ne sont pas dans le bon ordre.)*
Il est conseillé d'écrire une fonction swap qui échange deux valeurs dans un tableau

▶ **Question 7 *** ***QuickSort***
Dans la méthode quicksort, on procède encore récursivement. À chaque étape, on choisit une valeur pivot et on partitionne (i.e. on met les éléments plus petits d'un côté et les plus grands de l'autre) le tableau par rapport à cette valeur et on rappelle récursivement pour chacun des deux sous tableaux.
Il conseillé de manipuler les sous tableaux comme une paire d'indice (début inclus, fin exclue) dans un tableau global. Je vous conseille d'écrire les fonctions `coupe` et `quicksort` :

```
coupe : 'a array -> int -> int -> 'a -> int
```

prend en paramètre un tableau, un indice de début, un indice de fin ainsi qu'un élément pivot (de type `'a`). Cette fonction place les éléments plus grand que le pivot à la fin du sous tableau et les éléments plus petits au début. Elle renvoie l'indice limite entre les éléments plus petits et ceux plus grands.

```
quicksort : 'a array -> unit
```

prend en paramètre un tableau et le trie.

3 Backtrack

Le backtrack c'est une classe d'algorithme pour résoudre des problèmes. L'idée c'est que pour trouver l'existence de solution à un problème, on va distinguer plusieurs cas et tenter de résoudre chacun des cas séparément. Par exemple si je veux résoudre un sudoku, je vais distinguer selon le chiffre qu'il peut y avoir dans tel case, essayer de résoudre le sudoku avec cette valeur et si cela ne marche pas je supposerais une autre valeur pour la case et recommencerais.

3.1 Sudoku solver

Dans cette partie on va écrire un programme capable de résoudre un sudoku. Le sudoku est représenté par un tableau 9×9. Pour construire un tel tableau on utilise la fonction make_matrix tailleX tailleY valeur.

▶ **Question 8 *** *Pourquoi serait-il faux de construire un tableau bidimensionnel avec*
make_vect tailleX (make_vect tailleY valeur) ?

▶ **Question 9** *Écrire un programme qui prend en entrée un tableau 9×9, un x, un y et un entier c avec* $1 \leq c \leq 9$ *renvoie true si poser c en (x,y) ne rentre pas en conflit avec les cases déjà remplies de la grille.*

▶ **Question 10 *** *En vous inspirant de la méthode du backtrack, écrire un programme qui résout le sudoku.*

3.2 Le problème des huit dames

Le problèmes des huit dames est de trouver la position de huit dames sur un échiquier de telle manière qu'elles ne puissent pas se prendre les unes les autres.

▶ **Question 11 *** *En appliquant la même méthode que dans la section précédente, trouvez des solutions au problème des huit dames. Combien de solutions y a-t-il ?*

▶ **MP1 – Option Informatique**
1er TP Caml
sl@jachiet.com
http://louis.jachiet.com/tps
Vendredi 27 septembre 2013

Listes, tableaux et tris

Un corrigé

▶ Question 1

```
let rec insere x = function
  | [] -> [x]
  | a::q ->
    if a < x
    then a:: insere x q
    else x::a::q

let rec complete_tri l = function
  | [] -> l
  | a::q -> complete_tri (insere a l) q
;;
let tri l = complete_tri [] l

let tri2 l = it_list (fun ac el -> insere el ac) [] l
```

▶ Question 2

```
let rec partition = function
  | [] -> ([],[])
  | a::q ->
    let (g,d) = partition q in
    (a::d,g)

let rec reunit = function
  | (a::q,x::t) ->
    if x > a
    then a::reunit (q,x::t)
    else x::reunit (a::q,t)
  | (a,b) -> a@b

let rec tri_fusion = function
  | [] -> []
  | [a] -> [a]
  | l ->
    let (p1,p2) = partition l in
    reunit (tri_fusion p1,tri_fusion p2)
```

▶ Question 3

```
let rec aleatoire = function
  | 0 -> []
  | n -> random__int 100::aleatoire (n-1)

let a = aleatoire 100
let b = tri_fusion a
let c = tri a
let d = c = b
```

▶ Question 4

Dans le tri par insertion, on insère n fois un élément dans une liste. Dans le pire cas on a donc : $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$ Pour le second, si $2^k \leq n \leq 2^{k+1}$, on peut minorer et majorer par les temps mis pour 2^k et pour 2^{k+1}. Quand $n = 2^k$ on montre alors par récurrence que $T(k) = 2^k + 2 * T(k-1) + 2^k$ donc $\frac{T(k)}{2^k} = 1 + \frac{T(k-1)}{2^{k-1}}$. On a donc $T(k) = k * 2^k$ et donc l'algorithme est en $O(n \times ln(n))$. Par ailleurs, il est possible de montrer qu'un algorithme de tri par comparaison ne peut pas avoir une meilleure complexité. Bien évidemment, le premier algorithme est plus lent, en pire cas, que le second.

▶ Question 5

```
let rec extrait_min = function
  | [] -> failwith "liste vide"
  | [a] -> (a,[])
  | a::q ->
    let (mini,reste) = extrait_min q in
    if a < mini
    then (a,q)
    else (mini,reste)

let rec tri_insertion = function
  | [] -> []
  | l ->
    let (mini,reste) = extrait_min l in
    mini::(tri_insertion reste)
```

▶ Question 6

```
let swap t a b =
  let c = t.(a) in
  t.(a) <- t.(b) ;
  t.(b) <- c

let tri tb =
  let modif = ref true in
  while !modif do
    modif := false;
    for i = 0 to vect_length tb - 2 do
      if tb.(i)>tb.(i+1) then
        swap tb i (i+1)
        modif := true
    done
  done
```

▶ Question 7

```
let swap t a b =
  let c = t.(a) in
  t.(a) <- t.(b) ;
  t.(b) <- c

let coupe tableau deb fin pivot =
  let rec coupeR deb fin =
    if deb<fin then
      if tableau.(deb) < pivot then
        coupeR (succ deb) fin
      else
        begin
          swap tableau deb (pred fin) ;
          coupeR deb (pred fin)
        end
    else
      deb
  in
coupeR deb fin
;;


let quicksort t =
  let rec triR deb fin =
    if succ deb < fin then
      begin
        let mil = coupe t (succ deb) fin t.(deb) in
        swap t (pred mil) deb ;
        triR deb mil ;
        triR (succ mil) fin
      end
  in
triR 0 (vect_length t)
;;
```

▶ Question 8

Parce que chaque ligne du tableau pointerait vers la même colonne.

▶ Question 9

```
let peut_poser x y c =
    let peut = ref true in
    for i = 0 to 8 do
      if g.(x).(i) = c || g.(i).(y) =c then peut := false
    done;
    for nx = (x/3)*3 to (x/3)*3+2 do
      for ny = (y/3)*3 to (y/3)*3+2 do
        if g.(nx).(ny) = c then peut := false
    done
    done;
    !peut
  in
```

▶ Question 10

```
let resoudre g =
  let peut_poser x y c =
    let peut = ref true in
    for i = 0 to 8 do
      if g.(x).(i) = c || g.(i).(y) =c then peut := false
    done;
    for nx = (x/3)*3 to (x/3)*3+2 do
      for ny = (y/3)*3 to (y/3)*3+2 do
        if g.(nx).(ny) = c then peut := false
    done
    done;
    !peut
  in

  let rec foo (x,y) =
    if y > 8
    then foo (x+1,0)
    else if (x,y) = (9,0)
    then
      begin
        for x = 0 to 8 do
          for y = 0 to 8 do
            print_int g.(x).(y)
          done ;
          print_newline () ;
        done;
        failwith ''trouve''
      end
    else
      if g.(x).(y) = 0
      then
        for c = 1 to 9 do
          if peut_poser x y c
          then (g.(x).(y) <- c ; foo (x,y+1) ; g.(x).(y) <- 0)
        done
      else
        foo (x,y+1)
  in
  foo (0,0)
```

▶ Question 11

```
let huit_dame ()=
  let g = make_matrix 24 24 0 in
  let aff =
    let s = make_string 72 ' ' in
    for i = 0 to 7 do
      s.[i*9+8] <- '\n'
    done;
    s
  in

  let affiche = let r = ref 0 in fun () ->
  incr r;print_int !r ;print_newline();
  print_string aff; print_newline() in

  let pose x y v =
    for i = -7 to 7 do
      g.(x).(y+i) <- g.(x).(y+i)+ v ;
      g.(x+i).(y) <-g.(x+i).(y)+ v ;
      g.(x+i).(y+i) <-g.(x+i).(y+i)+v ;
      g.(x+i).(y-i) <- g.(x+i).(y-i)+ v
    done ;
    if v > 0
    then aff.[(x-8)*9+y-8]<- '*'
    else aff.[(x-8)*9+y-8]<- ' '
  in
  let rec foo y =
    if y > 15 then affiche ()
    else
      for x = 8 to 15 do
        if g.(x).(y) = 0
        then ( pose x y 1 ; foo (y+1) ; pose x y (-1))
      done
  in
  foo 8
```