algorithmes de tri

7 mai 2003

Pour rechercher un élément x dans un tableau, on suppose à priori que l'élément n'est pas dans le tableau `let trouve = ref false` puis on parcourt tout le tableau et dès qu'un élément du tableau est égal à l'élément cherché, on stocke l'information en changeant la valeur de la variable `trouve` (ligne 4). En sortie de boucle, `!trouve=false` si $x \notin t$ et `!trouve = true` si $x \in t$.

```

1  #let cherche t x =
2  let trouve = ref false in
3      for i = 0 to (vect_length t -1) do
4          if t.(i) = x then trouve := true
5          done;
6      !trouve;;
7  cherche : 'a vect -> 'a -> bool = <fun>
8  #cherche [|5;4;6;7;8;9;1;3;3|] 3;;
9  - : bool = true
10 #cherche [|5;4;6;7;8;9;1;3;3|] 13;;
11 - : bool = false

```

Nous pouvons améliorer le programme ci-dessus en arrêtant le parcours du tableau t dès que nous avons trouvé l'élément x dans le tableau t .

```

1  #let cherche t x =
2  let trouve = ref false and l = vect_length t and i = ref 0 in
3      while not(!trouve) && !i < l do
4          if t.(!i) = x then trouve := true;
5          incr i;
6          done;
7      !trouve;;
8  cherche : 'a vect -> 'a -> bool = <fun>
9  #cherche [|5;4;6;7;8;9;1;3;3|] 3;;
10 - : bool = true
11 #cherche [|5;4;6;7;8;9;1;3;3|] 13;;
12 - : bool = false

```

Fonction d'échange de deux éléments d'un tableau.

```
1 #let echange t i j =
2 let temp = t.(i) in
3   t.(i) <- t.(j);
4   t.(j) <- temp;;
5 echange : 'a vect -> int -> int -> unit = <fun>
```

Fonction pour trouver l'indice d'un sous-tableau où se trouve le plus petit élément.

```
1 #let min t a b =
2 let min = ref a in
3 for i = a+1 to b do
4   if t.(i) < t.(!min) then min := i
5   done;
6 !min;;
7 min : 'a vect -> int -> int -> int = <fun>
```

Le tri par sélection

```
1 #let tri_selection t =
2 let l = vect_length t in
3 for i=0 to (l-2) do
4   echange t i (min t i (l-1))
5   done;;
6 tri_selection : 'a vect -> unit = <fun>
7 #let t = [|1;3;4;5;7;9;-6;-4;-6;96;45;12;145;4|];;
8 t : int vect = [|1; 3; 4; 5; 7; 9; -6; -4; -6; 96; 45; 12; 145; 4|]
9 #tri_selection t;;
10 - : unit = ()
11 #t;;
12 - : int vect = [| -6; -6; -4; 1; 3; 4; 4; 5; 7; 9; 12; 45; 96; 145|]
```

On écrit une fonction d'affichage pour pouvoir suivre les transformations du tableau dans l'exécution de la fonction `tri_selection`.

```
1 #let affiche t =
2 let l = vect_length t in
3 print_string "[";
4 for i = 0 to l-2 do
5   print_int t.(i); print_char ',';
6   done;
7 print_int t.(l-1); print_string "|";
8 print_newline();;
9 affiche : int vect -> unit = <fun>
```

Ce qui sur un exemple nous donne :

```
1  #let tri_selection t =
2  let l = vect_length t in
3  for i=0 to (l-2) do
4      echange t i (min t i (l-1));
5      affiche t
6      done;;
7  tri_selection : int vect -> unit = <fun>
8  #let t = [|1;3;4;5;7;9;-6;-4;-6;96;45;12;145;4|];;
9  t : int vect = [|1; 3; 4; 5; 7; 9; -6; -4; -6; 96; 45; 12; 145; 4|]
10 #tri_selection t;;
11 [| -6;3;4;5;7;9;1;-4;-6;96;45;12;145;4|]
12 [| -6;-6;4;5;7;9;1;-4;3;96;45;12;145;4|]
13 [| -6;-6;-4;5;7;9;1;4;3;96;45;12;145;4|]
14 [| -6;-6;-4;1;7;9;5;4;3;96;45;12;145;4|]
15 [| -6;-6;-4;1;3;9;5;4;7;96;45;12;145;4|]
16 [| -6;-6;-4;1;3;4;5;9;7;96;45;12;145;4|]
17 [| -6;-6;-4;1;3;4;4;9;7;96;45;12;145;5|]
18 [| -6;-6;-4;1;3;4;4;5;7;96;45;12;145;9|]
19 [| -6;-6;-4;1;3;4;4;5;7;96;45;12;145;9|]
20 [| -6;-6;-4;1;3;4;4;5;7;9;45;12;145;96|]
21 [| -6;-6;-4;1;3;4;4;5;7;9;12;45;145;96|]
22 [| -6;-6;-4;1;3;4;4;5;7;9;12;45;145;96|]
23 [| -6;-6;-4;1;3;4;4;5;7;9;12;45;96;145|]
24 - : unit = ()
```

Le tri par insertion

Vous avez certainement déjà pratiqué le tri par insertion. C'est un tri naturellement pratiqué par les joueurs de cartes. Si les i premières cartes sont rangées (en ordre croissant de valeurs par exemple), on prend la $(i+1)$ ième et on la fait «descendre» jusqu'à sa place dans le jeu (on s'arrête dès que l'on compare la carte à une carte de valeur inférieure ou bien si on est arrivé en première position).

```
1  #let tri_insertion t =
2  let l = vect_length t in
3  for i = 1 to l-1 do
4      let j = ref i in
5      while !j > 0 && t.(!j-1) > t.(!j) do
6          echange t !j (!j-1);
7          decr j;
8          done;
9      affiche t
10     done;;
11  tri_insertion : int vect -> unit = <fun>
12  #let t = [|1;3;4;5;7;9;-6;-4;-6;96;45;12;145;4|];;
13  t : int vect = [|1; 3; 4; 5; 7; 9; -6; -4; -6; 96; 45; 12; 145; 4|]
14  #tri_insertion t;;
15  [|1;3;4;5;7;9;-6;-4;-6;96;45;12;145;4|]
16  [|1;3;4;5;7;9;-6;-4;-6;96;45;12;145;4|]
```

```

17 [|1;3;4;5;7;9;-6;-4;-6;96;45;12;145;4|]
18 [|1;3;4;5;7;9;-6;-4;-6;96;45;12;145;4|]
19 [|1;3;4;5;7;9;-6;-4;-6;96;45;12;145;4|]
20 [| -6;1;3;4;5;7;9;-4;-6;96;45;12;145;4|]
21 [| -6;-4;1;3;4;5;7;9;-6;96;45;12;145;4|]
22 [| -6;-6;-4;1;3;4;5;7;9;96;45;12;145;4|]
23 [| -6;-6;-4;1;3;4;5;7;9;96;45;12;145;4|]
24 [| -6;-6;-4;1;3;4;5;7;9;45;96;12;145;4|]
25 [| -6;-6;-4;1;3;4;5;7;9;12;45;96;145;4|]
26 [| -6;-6;-4;1;3;4;5;7;9;12;45;96;145;4|]
27 [| -6;-6;-4;1;3;4;4;5;7;9;12;45;96;145|]
28 - : unit = ()

```

On peut améliorer la fonction `tri_insertion` en diminuant le nombre d'échanges, en réécrivant le programme de la manière suivante :

```

1  let tri_insertion t =
2  let l = vect_length t in
3  for i = 1 to l-1 do
4    let v = t.(i) and j = ref i in
5    while !j > 0 && t.(!j-1) > v do
6      t.(!j) <- t.(!j-1);
7      decr j;
8    done;
9    t.(!j) <- v
10   done;;

```

Le tri à bulles.

Le tri par bulles résulte d'une idée originale pour trier un tableau. On part du début du tableau et on remonte, on compare deux éléments successifs et si le premier est plus grand que le suivant, on les permute. Une fois que le tableau a été balayé le plus grand élément se retrouve en place, en dernière position. On recommence le processus une deuxième fois puis autant de fois qu'il le faut pour ranger le tableau.

```

1  let tri_bulle1 t =
2  let n = vect_length t in
3  for i = 0 to n-1 do
4    for j = 0 to n-2 do
5      if t.(j) > t.(j+1) then echange t j (j+1);
6    done;
7  affiche t;
8  done;;

```

Nous pouvons améliorer l'algorithme précédent en remarquant qu'à la fin de la $i^{\text{ème}}$ étape, les i derniers éléments du tableau sont en place.

```

1  let tri_bulle2 t =
2  let n = vect_length t in
3  for i = 0 to n-1 do
4    for j = 0 to n-i-2 do
5      if t.(j) > t.(j+1) then echange t j (j+1);
6      done;
7  affiche t;
8  done;;

```

Une dernière amélioration est encore possible. Si, lorsque nous balayons le tableau à la $i^{\text{ème}}$ étape, nous n'avons effectué aucune permutation, c'est que le tableau est bien rangé et si la dernière permutation s'est effectuée entre les positions i_0 et $i_0 + 1$, nous sommes sûrs que les éléments à partir de la position $i_0 + 1$ sont rangés dans le bon ordre.

```

1  let tri_bulle3 t =
2  let n = vect_length t in
3  let i = ref (n-2) and imax = ref 0 in
4  while !i <> 0 do
5    for j = 0 to !i do
6      if t.(j) > t.(j+1) then (echange t j (j+1); imax := j)
7      done;
8    i := !imax; imax := 0;
9    affiche t;
10 done;;

```

La variable `imax` permet de repérer l'emplacement de la dernière permutation. Dans cette dernière version, si le tableau est déjà rangé en ordre croissant, nous ne parcourons qu'une seule fois le tableau ce qui présente un avantage par rapport aux tris par sélection et par insertion. Par contre, si le tableau est rangé mais en ordre décroissant, le nombre de permutations d'éléments sera égal à $\frac{n(n-1)}{2}$ ce qui le rend dans ce cas moins intéressant que le tri par sélection par exemple.

Fonction de recherche dans un tableau trié.

Si on suppose que le tableau est trié, nous pouvons utiliser un algorithme de recherche par dichotomie pour déterminer si un élément appartient ou non à un tableau.

```

1  (* d : indicateur de début
2    f : indicateur de fin
3    m : indicateur de milieu *)
4  let cherche t x =
5  let n = vect_length t in
6  let d = ref 0 and f = ref (n-1) in
7  let m = ref ((n-1)/2) in
8  while (t.(!m) <> x && f >= d) do
9    if t.(!m) < x
10     then (d := !m+1; m := (!d + !f)/2;)
11     else (f := !m-1; m := (!d + !f)/2;);
12 done;
13 t.(!m) = x; (* égal à false si t.(m) <> x et à true sinon *)
14 ;;

```

Et voici avec un petit peu d'anticipation, la version récursive de la recherche dichotomique dans un tableau trié.

```
1 let cherche t x =
2 let n = vect_length t in
3 let rec cherche_rec t m M x =
4     let p = (m+M)/2 in
5     match x with
6     | _ when M < m -> false
7     | x when x = t.(p) -> true
8     | x when x < t.(p) -> cherche_rec t m (p-1) x
9     | x -> cherche_rec t (p+1) M x
10    in
11 cherche_rec t 0 (n-1) x;;
```