Université Bordeaux 1 Licence Informatique 2006/2007

INF102 Initiation à l'algorithmique

Corrigé devoir surveillé

Durée : 1h20. Aucun document autorisé. Les algorithmes qu'on vous demandera d'écrire dans les exercices suivants doivent utiliser le langage EXALGO défini en cours.

Exercice 1

1. Sans le reécrire, expliquer le principe de l'algorithme de tri par insertion.
   L'algorithme de tri par insertion est un algorithme itératif. Il utilise le mécanisme d'insertion dans un tableau trié. Aprés i itération les i premiers éléments du tableau initial sont triés entre eux. L'itération suivante consiste à insérer le (i+1)ieme élément parmi ces i éléments en conservant l'ordre établi. On procède ainsi jusqu'à avoir traité les n éléments du tableau
2. Faites le tourner sur un tableau contenant la suite d'entier $4, 7, 2, 8, 6, 1, 5$ afin de trier le tableau en ordre croissant.

```
[ 4|7 2 8 6 1 5]
  ^
[ 4 7|2 8 6 1 5]
    ^
[ 2 4 7|8 6 1 5]
  ^
[ 2 4 7 8|6 1 5]
        ^
[ 2 4 6 7 8|1 5]
      ^
[ 1 2 4 6 7 8|5]
  ^
[ 1 2 4 5 6 7 8]
        ^
```

3. Donner un ordre de grandeur du nombre d'affectations et du nombre de comparaisons nécessaires au déroulement de cet algorithme pour le pire des cas, préciser quel est ce cas.
   Lorsqu'on veut effectuer un tri par insertion par ordre croissant le pire des cas est celui où le tableau initial est trié dans l'ordre décroissant. Pour ce cas il y aura de l'ordre de n^2 affectations et n^2 comparaisons.
4. Même question pour le cas au "mieux" *Pour le même genre de tri le meilleur des cas sera celui où le tableau est déjà trié dans l'ordre croissant. Pour ce cas il y aura de l'ordre de n affectations et n comparaisons*

Exercice 2

Soit `Mes` un tableau de `N` réels ($1 \leq N \leq NMAX$). En stockant par ordre croissant dans ce tableau une série de mesures on remarque que le tableau contient beaucoup de valeurs identiques qui sont contigües puisque le tableau est trié.

Exemple : avec $N = 16$

`Mes` =

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1.3	1.3	1.5	1.8	1.8	1.8	1.8	2.0	2.0	2.0	2.0	2.0	2.1	2.5	2.5	2.5

On souhaite représenter ce tableau de façon plus compacte en faisant figurer seulement les mesures dont les valeurs sont distinctes et en leur associant le nombre de fois où elles apparaissent dans le tableau `Mes`. Pour cela on utilise un tableau `MesCompact` dont les `M` éléments ($1 \leq M \leq KMAX$) sont des structures ayant deux champs :

- Un champ `mesure` contenant la valeur de la mesure
- Un champ `nb` contenant le nombre de fois que cette mesure figure dans le tableau `Mes`

Exemple : avec $M = 6$

`MesCompact` =

1	2	3	4	5	6
1.3	1.5	1.8	2.0	2.1	2.5
2	1	4	5	1	3

1. Les expressions suivantes ont-elles un sens ? Si oui donner leur type et leur valeur dans l'exemple précédent :
   - `MesCompact[2]` *Oui. C'est le* 2^{ieme} *éléments du tableau MesCompact. Il s'agit donc d'une structure dont le champs* `mesure` *est un réel qui vaut* 1.5 *et le champs* `nb` *est un entier qui vaut* 1.
   - `mesure[4]` *Non*
   - `MesCompact.mesure[3]` *Non*
   - `MesCompact[1].mesure` *Oui. Il s'agit d'un réel qui vaut* 1.3. *C'est la première mesure.*
   - `nb.MesCompact[5]` *Non*
   - `MesCompact[5].nb` *Oui. Il s'agit d'un entier qui vaut* 1. *C'est le nombre de fois où la* 5^{ieme} `mesure` *est recensée*
2. Ecrire une fonction `compacte` qui construit le tableau `MesCompact` à partir du tableau `Mes`. Les deux tableaux seront des paramètres de la fonction.

```
type Element = structure
        mesure : reel ;
        nb : entier ;
           finstructure

TabCompact = tableau[1..KMAX] de Element
```

```
fonction compacte(ref MesCompacte:TabCompact,
                  ref Mes:tableau[1..NMAX] d'entier,
                  ref m:entier, val n:entier):vide

var i, j :entier;

debut
    j = 1;
    MesCompacte[1].mesure = Mes[1];
    MesCompacte.nb = 1;
    pour i = 2 à n faire
        si Mes[i] == MesCompacte[i].mesure alors
            MesCompacte[j].nb =  MesCompacte[j].nb + 1;
        sinon
            j = j+1;
            Mescompacte[j].mesure = Mes[i];
            MesCompacte[j].nb = 1;
        finsi
    finpour
    m = j;
fin finfonction
```

3. On suppose maintenant que le tableau `MesCompact` a été construit.
   – Ecrire une fonction `moyenne` qui calcule la moyenne de toutes les mesures à partir du tableau `MesCompact`.

```
fonction moyenne(ref MesCompacte:TabCompact, val m:entier) : reel
var i : entier;
    SomMes, SomNb : reel;

debut
    SomMes = 0;
    SomNb = 0;
    pour i = 1 à m faire
        SomMes = SomMes + MesCompacte[i].mesure * MesCompacte[i].nb;
        SomNb = SomNb + MesCompacte[i].nb;
    finpour
    retourner SomMes/SmNb;
fin
finfonction
```

– Ecrire une fonction `freqmax` qui renvoie la valeur de la mesure qui apparait le plus souvent (en cas d'égalité, elle renvoie la plus petite mesure).

```
fonction freqmax(ref MesCompacte:TabCompact, val m:entier) : reel
var i, imax : entier;

debut
    imax = 1;
    pour i = 2 à m faire
        si MesCompacte[i].nb > MesCompacte[imax].nb alors
            imax = i;
        finsi
    finpour
    retourner MesCompacte[imax].mesure;
fin
finfonction
```

Exercice 3

On stocke des valeurs réelles strictement positives toutes distinctes dans un tableau `Vect` contenant `N` éléments ($1 \leq N \leq NMAX$).
Ce tableau est souvent modifié par des suppressions ou des ajouts d'éléments ; au lieu de faire des décalages, pour supprimer un élément on convient de le remplacer par 0 et pour ajouter un nouvel élément on l'ajoute à la fin du tableau s'il y a de la place. Il y a donc dans `Vect` des éléments « significatifs » : les réels strictement positifs, et des éléments « non significatifs » : les 0.

1. Ecrire une fonction `ajout-elt` qui ajoute dans le tableau `Vect` contenant déjà N éléments, l'élément dont la valeur est x (on supposera qu'il y a assez de place).

   ```
   fonction ajout-elt(ref Vect:tableau[1..NMAX] de reel, ref N:entier ,
   val x:reel) : vide /* on suppose N<NMAX */

   debut
       n = n+1;
       Vect[n] = x;
   fin
   finfonction
   ```

   Quelle est la complexité de cette fonction ? *La complexité de cette fonction est en temps constant ou $O(1)$*

2. Ecrire une fonction `sup-elem` qui « supprime » du tableau `Vect` un élément x, s'il est présent, en le rendant non significatif, sinon ne fait rien.

   ```
   fonction sup-elem(ref Vect:tableau[1..NMAX] de reel,
                     val N:entier) : vide;
   var i:entier;

   debut
       i = 1;
       tant que i < N et Vect[i] != x faire
           i = i+1
       fintantque
       si Vect[i] == x alors
           Vect[i] = 0;
       finsi
   fin
   finfonction
   ```

   Quelle est la complexité de cette fonction ? *La complexité de cette fonction est linéaire par rapport à N ou $O(N)$*

3. Lorsque le nombre de 0 devient trop important, cela peut devenir gênant. Il est alors nécessaire de « tasser » le tableau `Vect` en supprimant les 0. Sans utiliser de tableau supplémentaire, écrire une fonction `tasser` qui supprime tous les 0 de `Vect`.

   ```
   fonction tasser(ref Vect:tableau[1..NMAX] de reel,
                   ref N:entier):vide;
   var i , j : entier;

   debut
       j = 0;
       pour i = 1 à N faire
           si Vect[i] != 0 alors
               j = j+1;
               Vect[j] = Vect[i];
           finsi
       finpour
       N = j;
   fin
   finfonction
   ```

Exemple :
  `Vect` $= (0, 0, 0, 11, 0, 23, 0, 0, 0, 17, 0, 0, 0, 0, 38, 0, 0, 0,)$ avec $N = 18$
devient
  `Vect` $= (11, 23, 17, 38)$ avec $N = 4$