Session 2008-2009 : ING 1 : Examen d'algorithmique I

Durée 2heures : aucun document autorisé

Algorithmique de base

Dans cet exercice, nous considérons un fichier ***Liste.txt*** contenant des Etudiants. Un Etudiant est caractérisé par son nom, son prénom et sa moyenne.

1. **Un peu de statistiques simples.**

1.1 Créez une structure permettant de représenter un Etudiant. Dans la suite de l'exercice, vous supposerez que **Liste.txt** contient des Etudiants de ce type.
Réponse (0,5 pt)

```
type Etudiant = Structure
nom : Chaine
prenom : Chaine
moyenne : Reel
FinType
```

1.2 Ecrivez un algorithme qui récupère tous les Etudiants du fichier et qui affiche des statistiques sur les moyennes des Etudiants :

Réponse (4pts)

```
procedure afficherStatistiques()
variables locales
moyenne : Reel
variance : Relle
compteur : Entier

    moyenne ← 0
    compteur ← 0
    ouvrir(fl, "Liste.txt ")
    TantQue non eof(fl) Faire
        lire(fl,val)
        moyenne ← moyenne +val
        compteur ← compteur + 1
    FinTantQue
    fermer(fl)
    moyenne ← moyenne / compteur
    ouvrir(fl, "Liste.txt ")
    variance ← 0
    TantQue non eof(fl) Faire
        lire(fl,val)
```

```
            variance ← variance + (val-moyenne)  *  (val-moyenne)
        FinTantQue
        fermer(fl)
        variance ← variance / compteur
        ecrire("la moyenne vaut ",moyenne)
        ecrire("la variance vaut ",variance)
        fermer(fl)
    FinProcedure
```

- La moyenne arithmétique

  Rappel : Pour un vecteur X = (x_1,x_2,...,x_k), la moyenne arithmétique est

$$\bar{x} = \left(\sum_{i=1}^{k} x_i\right)/k$$

- La variance

  Rappel : Pour un vecteur X = (x_1,x_2,...,x_k), la variance est :

$$V(X) = \left(\sum_{i=1}^{k} (x_i - \bar{x})^2 \right)/k$$

2. Calcul de racine carrée.

La méthode de Héron ou **méthode babylonienne** est une méthode efficace d'extraction de racine carrée. Elle porte le nom du mathématicien **Héron d'Alexandrie** mais certains calculs antérieurs semblent prouver que la méthode est plus ancienne.

Pour déterminer la racine carrée du nombre A, on choisit un nombre $x_0 > 0$, puis on construit une suite définie par récurrence par $x_{n+1} = \frac{1}{2} * (x_n + \frac{A}{x_n})$

La suite ainsi obtenue est une suite décroissante à partir du second terme, convergeant vers $\sqrt{A}$.

2.1 Créez une fonction racine **récursive** en suivant la définition précédente. Essayez de limiter au maximum le nombre d'appels récursifs. Cette fonction calculera le terme x_n de la suite. La fonction aura donc *n* pour paramètre.

Réponse (2 pts)

```
Fonction racineCarreeHeron(a : Reel, n : Entier, x : Reel) : Reel
        Variables locales
        y : Reel
        Si  n > 0 Alors
                y ← racineCarreeHeron(a,n-1,x)
```

```
            retourner 0.5 * (y + a / y))
        Sinon
            retourner x
Fin Fonction
```

2.2 Utilisez la fonction précédente pour ajouter l'affichage de l'Ecart-type à votre algorithme. On prendra $n = 100$ pour obtenir une bonne précision.

Rappel : Pour un vecteur $X = (x_1,x_2,...,x_k)$, l'écart-type est : $\sqrt{V(X)}$

Réponse (0,5pt)

On ajoute l'instruction

```
ecrire("L'écart-type vaut ", racineCarreeHeron(variance,100,1))
```

3. Complexité

3.1 Calculez séparément la complexité des différentes fonctions/procédures que vous avez écrites et déduisez-en la complexité de votre algorithme.

Réponse

Complexité de la fonction afficherStatistiques : 1,5 pt

On note n le nombre d'enregistrements dans le fichier liste.txt.

- Pour le premier parcours du fichier, on effectue 2*n affectations et 2*n additions.
- Pour le deuxième parcours du fichier, on effectue n affectations, n produits, n soustractions et n additions.
- Pour la partie fixe, on effectue 5 affectations et 2 divisions

On a donc effectué 8*n + 7 opérations élémentaires. La complexité est d'ordre O(n).

Complexité de la fonction racineCarreeHeron : 1,5 pt

A chaque exécution de la fonction racineCarreeHeron, on effectue, dans le cas général, 1 affectation, 1 soustraction, 1 produit, 1 addition, 1 division et 1 retour. Le nombre d'appels dans le cas général est égal à n. On a donc effectué 6*n opérations élémentaires. La complexité est d'ordre O(n).

Tri d'une matrice

Représentons une page d'un logiciel de courrier électronique par une matrice (n*p) de chaînes de caractères. Chaque ligne représente un mail. Pour chaque mail, on a p colonnes. Chaque colonne représente un renseignement du mail.

On désire pouvoir trier ces mails. Pour trier, l'utilisateur sélectionne une colonne j. Pour comparer deux mails, on compare les contenus des colonnes dans l'ordre suivant :

j, j+1, …,p, 1, …, j-1

On note $m_{i,k}$ le $k^{ème}$ renseignement du $i^{éme}$ mail. Pour comparer deux mails m_i et $m_{i'}$, on cherche le premier indice de colonne noté k où les renseignements sont différents. Pour cet indice k,

on compare les deux chaînes de caractères $m_{i,k}$ et $m_{i',k}$. Le mail m_i est plus petit que le mail $m_{i'}$ si et seulement si $m_{i,k} < m_{i',k}$.
Exemple p = 4 et j = 3.
On considère les deux mails suivants :

	1	2	3	4
m_1	Titi	Zozo	Alpha	Beta
m_2	Toto	Tutu	Alpha	Beta

$m_1 < m_2$ car

$m_{1,3} = m_{2,3}$ = 'Alpha'
$m_{1,4} = m_{2,4}$ = 'Beta'
$m_{1,1}$ = 'Titi' < $m_{2,1}$ = 'Toto'

1. Ecrire l'algorithme qui reçoit la matrice des mails et l'indice de la colonne sélectionnée et qui trie cette matrice de mails comme indiqué ci-dessus. Vous avez le choix pour la technique de tri.

Réponse (7 pts)

■ Cette procédure trie la matrice **em** en adaptant le tri par sélection

```
Procedure triMatrice(ES em : tableau() : Entier, E n Entier, p Entier, no Entier)
    Variables locales
    jmax, i, j : Entier
    Pour i ← n à 2 pas -1
        jmax ← 1
        Pour j ← 2 à i pas 1
                Si infLigne(em,n,p,no,jmax,j) Alors
                        jmax ← j
                FinSi
        FinPour
        Si i /= jmax Alors
                inverserLigne(em, n, p, jmax, i)
        FinSi
    FinPour
FinProcedure
```

■ Cette fonction renvoie vrai si la ligne i de la matrice **em** est strictement inférieure à la ligne j de la matrice **em**, faux sinon

```
Fonction infLigne(E em : tableau() : Entier, E n : Entier, E p : Entier, E no :
Entier, E  i : Entier, E j : Entier) : Booleen
    Variables locales
    k : Entier
    res : booleen

    k ← no
    res ← faux
    Tantque (non res) et k <= p Faire
```

```
            Si em(i,k) < em(j,k) Alors
                    res ← vrai
            Sinon
                    k ← k + 1
            FinSi
        FinTantQue
        k← 1
        Tantque (non res) et k < no Faire
            Si em(i,k) < em(j,k) Alors
                    res ← vrai
            Sinon
                    k ← k + 1
            FinSi
        FinTantQue
        Retourner res
FinFonction
```

■ Cette procédure intervertit les lignes **i** et **j** de la matrice **em**

```
Procedure inverserLigne(ES em : tableau() : Entier, E n : Entier, E p : Entier, E i :
Entier, E j : Entier)
    Variables locales
    k, val : Entier

    Pour k ← 1 à p pas 1
        val ← em(i,k)
        em(i,k) ← em(j,k)
        em(j,k) ← val
    FinPour
FinProcedure
```

2. Calculer la complexité de votre tri.

Réponse (3pts)

La procédure triMatrice :

- effectue, dans la boucle principale, n-1 affectations, n-1 comparaisons et dans le pire des cas n-1 appels de la procédure inverserLigne
- effectue, dans la boucle secondaire, n*(n-1)/2 appels à la fonction infLigne et dans le pire des cas n*(n-1)/2 affectations.

La fonction infLigne

- effectue deux affectations
- effectue dans le pire des cas p comparaisons, p additions et p affectations

La procédure inverserLigne effectue 3*p affectations.

En tout le nombre d'opérations élémentaires est égal à

2*(n-1) + (n-1)*(2+3*p) + n*(n-1)/2 *(2+3*p) + n*(n-1)/2. En conclusion, la complexité est $O(n^2*p)$.