UNIVERSITE IBN TOFAIL
Faculté des sciences
Département d'Informatique
Kenitra

Année: 2017/2018
Filières : SMI
Semestre : 3

Algorithmique II

Examen final

Corrigé

Exercice 1 : (Sur 7 points)

Un établissement universitaire organise les résultats finaux des étudiants d'une filière sous la forme d'un tableau *Filiere* composé de *N* éléments. Chaque élément du tableau est une structure contenant les champs suivants :

- *Code_Etudiant* qui est une chaine de 8 caractères
- *Nom* qui est une chaine de 15 caractères
- *Prénom* qui est une chaine de 15 caractères
- *Moyenne* qui est un nombre réel

1. Donner la déclaration du type *Etudiant* en tant que structure regroupant les champs ci-dessus et la déclaration de la variable *Filiere* en tant que tableau composé de N *Etudiant*
2. Ecrire la procédure *Afficher_Résultat*() qui permet d'afficher les résultats des étudiants de la filère comme suit :

Numéro	Code_Etudiant	Nom	Prénom	Moyenne	Mention

Où Numéro est le numéro d'ordre dans la liste, Code_Etudiant est le code de l'étudiant, Nom est le nom de l'étudiant, Prénom est le prénom de l'étudiant, Moyenne est la moyenne de l'étudiant et Mention est égale à :

"Ajourné"	si	Moyenne < 10
"Passable"	si	10 ≤ Moyenne < 12
"A.Bien"	si	12 ≤ Moyenne < 14
"Bien"	si	14 ≤ Moyenne < 16
"T.Bien"	si	16 ≤ Moyenne

Corrigé

1. Déclarations

```
        Type  Etudiant = Structure
                Code_Etudiant : Caractere[1..8]
                Nom           : Caractere[1..15]
                Prenom        : Caractere[1..15]
                Moyenne       : Reel
        Fin Structure
Var     Filiere :       Etudiant[1..N]
```

2.

```
Procedure Afficher_Resultat()
Var     i, j : Entier
Debut
  Ecrire(”\nNumero\t Code_Etudiant \t Nom \t Prenom \t Moyenne \t Mention”)
  Pour  (i←1 a N) Faire
      Ecrire(”\n”, i, ” \t ”, Filiere[i].Code_Etudiant, ” \t ”, Filiere[i].Nom, ” \t ”)
      Ecrire(Filiere[i].Prenom, ” \t ”, Filiere[i].Moyenne, ” \t ”)
      Si  (Filiere[i].Moyenne < 10) Alors
         Ecrire(”Ajourne”)
      Sinon
         Si (Filiere[i].Moyenne < 12  Et  Filiere[i].Moyenne >= 10) Alors
            Ecrire(”Passable”)
         Sinon
            Si (Filiere[i].Moyenne < 14  Et  Filiere[i].Moyenne >= 12) Alors
               Ecrire(”A.Bien”)
             Sinon
               Si (Filiere[i].Moyenne < 16  Et  Filiere[i].Moyenne >= 14) Alors
                  Ecrire(”Bien”)
               Sinon
                   Ecrire(”T.Bien”)
               Fin Si
            Fin Si
         Fin Si
     Fin Si
  Fin Pour
Fin
```

Exercice 2 : (Sur 8 points)

```
Fonction Calcul(A : Entier[1..n]) : Entier
Var     i, j : Entier
        Count, MaxCount :  Entier
Debut
        i ←1
        j ←1
        MaxCount ←0
        Count ←0
        Tant que (i  <= n)  Faire
                Si (A[i] = A[j]) Alors
                        Count ← Count + 1
                Fin Si
                j ←j + 1
                Si  (j > n) Alors
                        Si  (count > MaxCount)  Alors
                                MaxCount ← Count
```

Fin Si
Count ← 0
i ← i + 1
j ← i
Fin Si
Fin Tant Que
Retourner MaxCount

Fin

1. **Expliquez brièvement ce que fait la fonction *Calcul***
2. **Déterminer la complexité temporelle dans le pire des cas de la fonction *Calcul*.**
3. **Ecrire une fonction *Meilleur()* faisant le même travail que la fonction *Calcul* et qui est de complexité temporelle, dans le pire des cas, strictement inférieure à celle de la fonction *Calcul* [indication : Vous pouvez exploiter un des algorithmes vus au cours, sans donner sa description].**

Corrigé

But de la fonction Calcul

Pour i=1, on effectue n passages dans la boucle tant que, ceci à la recherche du nombre d'occurrences de A[1]. Après ces n passage, le nombre obtenu (valeur de Count) est affecté à la variable MaxCount et une nouvelle série de passages dans la boucle tant que commence avec i=2. Après n-1 passages on compare la valeur de Count (qui égal au nombre d'occurrences de A[2] dans le tableau A[2..n]) ; si (Count>MaxCount) alors on affecte à MaxCount la valeur de Count. On continue ainsi jusqu'à la sortie de la boucle Tant que. La valeur retournée par la fonction Calcul est alors le nombre d'occurrences de l'élément le plus fréquent du tableau A.

Complexités temporelles dans le pire des cas de la fonction Calcul.

Soit t(n) la complexité temporelle de la fonction Calcul dans les pires des cas on a :
Pour chaque valeur de i, on effectue (n-i+1) passage dans la boucle Tant que (pour j=i à j=n). Pour les n-i premiers passages, on effectue les mêmes opérations, à savoir :
3*tcomp + 2*taffect + 2*tadd
Pour le (n-i+1)^{ème} passage, on effectue de plus :
tcomp + 4*taffect + tadd (associé à la partie en vert)
D'où la complexité dans les pires des cas de la fonction Calcul est :

$$\mathrm{t(n)} = 4 * \mathrm{taffect} + \mathrm{tretour} + \sum_{\mathrm{i=1}}^{n}\big((n-i)*(3*tcomp + 2*taffect + 2*tadd) + tcomp + 4*taffect + tadd\big) + tcomp = \theta(n^2)$$

Fonction équivalente à la fonction Calcul et qui est de complexité temporelle strictement inférieure à celle de la fonction Calcul

Fonction Meilleur()
Var i, j, count, MaxCount
Debut
//Trie du tableau A[1..n]

On applique l'algorithme Trie Rapide au tableau A[1..n]
//Calcul de la valeur de MaxCount
i⟵1
MaxCount⟵0
Tant que(i<n) Faire
    count⟵0
    j⟵i
    Tant que (j<=n et A[i]=A[j]) Faire
        count⟵count+1
        j⟵j+1
    Fin Tant que
    Si (count>MaxCount) Alors
        MaxCount⟵count
    Fin Si
    i⟵j
Fin Tant que
Retourner MaxCount
Fin

Complexité temporelle de la fonction Meilleur()
On connait la complexité temporelle de l'algorithme trie rapide qui est θ(nlog n)
On montre que la complexité temporelle t1(n) de la partie qui calcule MaxCount est θ(n).

En effet, si $i_1, i_2, .., i_k$ sont les valeurs prises par i dans la boucle externe Tant que et $j_1, j_2, .., j_k$ sont les nombres de passage pour chacune de ces valeurs dans la boucle interne Tant que, on a :
$j_1 + j_2 + .. + j_k = n$, avec $k \le n$ et t1(n) est donnée par :

$$\mathrm{t1(n)} = 2 * \mathrm{taffect} + \sum_{\mathrm{i}=1}^{k}\left(4 * taffect + 4 * tcomp + (j_i) * (3 * tcomp + 2 * taffect + 2 * tadd)\right) + tcomp + \mathrm{tretour}$$

$$= C_0 + C_1 k + C_2 \sum_{i=1}^{k} j_i$$

$$= \theta(n)$$

Donc finalement, la complexité temporelle de la fonction *Meilleur* est θ(n log n) + θ(n) = θ(n log n)

Exercice 3 : (Sur 5 points)
La fonction récursive d'Ackerman f est la fonction définie de IN x IN dans IN par :

$$\mathrm{f(n,\ m)} = \begin{cases} \mathrm{m} + 1 & Si\ \ n = 0 \\ f(n-1, 1) & Si\ \ m = 0\ \ et\ \ n \ge 1 \\ f(n-1, f(n, m-1)) & Si\ \ n > 0\ \ et\ \ m > 0 \end{cases}$$

1. **Calculer f(1,0) et f(2,0)**
2. **Ecrire en pseudo_code la fonction récursive *Ackerman(n : Entier, m : Entier)* qui retourne la valeur de f(n,m)**

Corrigé

1. f(1,0) = f(0,1), d'après la deuxième égalité
= 2, d'après la première égalité

f(2,0) = f(1,1), d'après la deuxième égalité
= f(0,f(1,0)), d'après la troisième égalité
= f(1,0)+ 1, d'après la première égalité
= 2 + 1 = 3

f(3,0) = f(2,1), d'après la deuxième égalité
= f(1,f(2,0)), d'après la troisième égalité
= f(1,3) = f(0,f(1,2)), d'après la troisième égalité
= f(1,2) + 1, d'après la première égalité
= f(0,f(1,1)) + 1, d'après la troisième égalité
= f(1,1) + 1 + 1, d'après la première égalité
= f(0,f(1,0)) + 2, d'après la troisième égalité
= f(1,0) + 1 + 2, d'après la première égalité
= 2 + 3 = 5

2.

```
Fonction Ackerman(n : Entier, m : Entier) : Entier
Debut
        Si (n<0 ou m<0) Alors
                Retourner
        Fin Si
        Si  ( n=0 )  Alors
                Retourner (m+1)
        Sinon
                Si ( m=0 )  Alors
                        Retourner( Ackerman(n-1, 1) )
                Sinon
                        Retourner( Ackerman(n-1, Ackerman(n, m-1)) )
                Fin Si
        Fin Si
Fin
```