UNIVERSITE IBN TOFAIL
Faculté des sciences
Département d'Informatique
Kenitra

Année : 2021/2022
Filières : SMI
Semestre : 3

Algorithmique II

Examen final

Corrigé

Exercice 1 : (Sur 7 points)

On rappelle qu'un nombre entier positif n>1 est dit premier si les seuls diviseurs de n sont 1 et n.
Un nombre entier positif n>1 est dit semi-premier si n est le produit de deux nombres premiers non nécessairement distincts.

1. **Ecrire une fonction Premier(n : Entier) qui retourne Vrai si n est premier, Faux sinon.**
2. **Ecrire une fonction SemiPremier(n : Entier) qui retourne Vrai si n est semi-premier, Faux sinon, en utilisant la fonction Premier.**
3. **En utilisant la fonction *SemiPremier,* définie ci-dessus, écrire l'algorithme *AfficheNombreSemiPremier* qui détermine et affiche tous les nombres semi-premiers inférieurs à un entier N saisi par l'utilisateur.**

Corrigé :

```
1.  Fonction Premier(n : Entier) : Boolleen
    Var         i, M : Entier
    Début
        Si (n<=1) Alors
            Retourner (Faux)
        Fin Si
        M←n Div 2
        Pour (i←2 à M) Faire
            Si (n Mod i=0) Alors
                Retourner Faux
            Fin Si
        Fin pour
        Retourner Vrai
    Fin
```

```
2.  Fonction SemiPremier(n : Entier) : Boolleen
    //Nous insérons ici la fonction Premier décrite ci-dessus
    Var         i, nbdiv, produit, M : Entier
    Début
        Si (n<=3) Alors
            Retourner (Faux)
        Fin Si
        M←n Div 2
        i←2             //pour parcourir tous les entiers inférieurs ou égaux à M
        nbdiv←0         //nombre de diviseurs premiers de n
        produit←1       //produit des diviseurs premiers de n, initialisé à 1
        Tant que (i <= M Et nbdiv<2) Faire
            Si (n Mod i=0  Et  Premier(i)) Alors
                Si (i*i=n) Alors
                    Retourner Vrai
                Fin Si
                nbdiv←nbdiv+1
                produit←produit * i
```

```
            Fin Si
            i←i+1
    Fin Tant que
    Si (produit=n) Alors
            Retourner Vrai
    Sinon
            Retourner Faux
    Fin Si
Fin
```

Ci-dessous une autre manière d'écriture de la fonction *SemiPremier,* mais elle est moins rapide que la précédente

```
Fonction SemiPremier(n : Entier) : Boolleen
//Nous insérons ici la fonction Premier décrite ci-dessus
Var         i, j, M : Entier
            SemiPremier : Boolleen←Faux
Début
    Si (n<=3) Alors
            Retourner (Faux)
    Fin Si
    M←n Div 2
    i←2             //pour parcourir tous les entiers inférieurs ou égaux à M
    Tant que (i <= M Et SemiPremier=Faux) Faire
        Si (n Mod i=0 Et Premier(i)) Alors
            j←i
            Tant que (j <= M Et SemiPremier=Faux) Faire
                Si (i*j=n Et Premier(j)) Alors
                    SemiPremier ←Vrai
                Fin Si
                j←j+1
            Fin Tant que
        Fin Si
        i←i+1
    Fin Tant que
    Retourner SemiPremier
Fin
```

3. Algorithme *AfficheNombreSemiPremier* ()

```
//Nous insérons ici la fonction SemiPremier décrite ci-dessus
Var         i, N : Entier
Début
    //Saisi d'un entier
    Répéter
        Ecrire("\nDonner la valeur d'un entier supérieur à 3")
        Lire(N)
    Jusqu'à N>3
    Ecrire("\nLes nombres semi-premiers inférieurs à ",N, " sont : \n")
    Pour (i←2 à N) Faire
        Si (SemiPremier(i)) Alors
            Ecrire(i,"\t")     //affichage de la valeur de i suivi de quelques espaces
        Fin Si
    Fin Pour
Fin
```

Exercice 2 : (Sur 6 points)

Étant donnés un texte *t* et un mot *m* sous forme de chaînes de caractères. La fonction suivante permet de déterminer le nombre de fois où le mot *m* apparaît dans le texte *t*.

Exemples :

– le mot "elle" apparaît 2 fois dans le texte "quelle belle journee",
– le mot "aa" apparait 4 fois dans le texte "aaaaa".

```
FONCTION chercheMot(m, t : chaine) : ENTIER
VAR     lt, lm, i, j, n, d : ENTIER
        Trouve : BOOLEEN
DEBUT
        lt ← longueur(t)        // longueur du texte
        lm ← longueur(m)        // longueur du mot
        n ← 0              // nombre d'occurrences trouvées
        i ← 1              // indice du caractère courant du texte
        d← lt-lm+1
        TANT QUE (i <= d) FAIRE
                j ← 1   // indice du caractère courant du mot
                Trouve ← VRAI
                TANT QUE (j <= lm ET Trouve) FAIRE
                        SI (t[i+j-1] <> m[j]) ALORS
                                Trouve ← FAUX
                        FIN SI
                        j ← j+1
                FIN TANT QUE
                SI (Trouve) ALORS
                        n ← n+1
                FIN SI
                i ← i+1
        FIN TANT QUE
        RETOURNER (n)
FIN
```

1. **Appliquer chercheMot(m, t) aux chaines t="Quels bonbons !" et m="bon"**
2. **Déterminer la complexité *C(lm, lt)* en <u>nombre de comparaisons</u> de la fonction *chercheMot(m, t),* ceci en fonction de *lm* et *lt*, où *lm* et *lt* sont respectivement les longueurs des deux chaînes *m* et *t.***

Corrigé

1. **<u>Application de la fonction cherchMot aux deux chaines m et t données ci-dessus</u>**

   m="bon" et t="Quels bonbons !"
   On a : lt=15 et lm=3.
   Donc d=lt – lm + 1=13
   On entre dans la boucle externe "Tant que" pour parcourir la chaine t de l'indice i=1 à l'indice i=13.

   - **Aucun des caractères de la chaine t correspondant aux valeurs de i allant de 1 jusqu'à 6 ne coïncide avec le premier caractère de la chaine m. D'où on passe une seule fois dans la boucle interne "Tant que" pour affecter la valeur "Faux" à la variable "Trouve", sans incrémentation de n à la sortie de cette boucle.**
   - **Les caractères de la chaine t correspondant aux valeurs de i allant de 7 jusqu'à 9 coïncident avec ceux de la chaine m, donc on passe trois fois dans la boucle interne, sans toucher à la valeur de "Trouve". D'où la *<u>première incrémentation de n</u>* à la sortie de cette boucle (n=1).**
   - **Les caractères de la chaine t correspondant aux valeurs de i allant de 10 jusqu'à 12 coïncident avec ceux de la chaine m, donc on passe trois fois dans la boucle interne, sans toucher à la valeur de "Trouve". D'où la *<u>deuxième incrémentation de n</u>* à la sortie de cette boucle (n=2).**

- Le caractère de la chaine t correspondant à la valeur i=13 ne coïncide pas avec le premier caractère de la chaine m. D'où on entre une seule fois dans la boucle interne pour affecter la valeur "Faux" à la variable "Trouve", sans incrémentation de n à la sortie de cette boucle.

Ensuite on sort de la boucle externe "Tant que" pour retourner la dernière valeur de n qui est égale à 2.

2. Calcul de la complexité en nombre de comparaisons de la fonction chercheMot

C(lm,lt) = d*[comp + lm*(3*comp + comp) + 4*comp] + comp
C(lm,lt) = c0 + c1 * d + c2 * d * lm, avec d = lt-lm+1, c0 = comp, c1 = 5*comp et c2 = 4*comp

Donc   C(lm,lt) = c0 + c1 *(lt-lm+1) + c2 *(lt-lm+1)*lm = θ(lm(lt-lm+1))

Exercice 3 : (Sur 7 points)

On considère la fonction récursive *BinRecursive(n : Entier)*, retournant une chaine de caractères, qui est donnée par :

```
Fonction BinRecursive(n : Entier) : chaine
Var     r : Entier
Début
        Si (n<0) Alors            //Si n<0 BinRecursive retourne la chaine vide
                Retourner ""
        Fin Si
        Si (n<2) Alors
                Si (n=0) Alors
                        Retourner "0"
                Sinon
                        Retourner "1"
                Fin Si
        Sinon
                r=n Mod 2
                Si (r=0) Alors
                        Retourner BinRecursive(n Div 2)+"0"
                Sinon
                        Retourner BinRecursive(n Div 2)+"1"
                Fin Si
        Fin Si
Fin
```

1. La récursivité de la fonction *BinRecursive(n)* est-elle terminale ou non terminale ? justifier votre réponse.
2. Donner les étapes d'exécution de *BinRecursive(13)*, ainsi que la chaine retournée par cet appel.
3. Sachant que la fonction *BinRecursive(n) retourne le codage binaire d'un entier positif,* écrire une fonction itérative *BinIterative(n)* équivalente à cette fonction.

Corrigé

1. La récursivité de la fonction *BinRecursive(n)* est non terminale, car le retour de valeur est suivi de traitements. En effet, le retour de la valeur de BinRecursive(n Div 2) est accompagné de la concaténation avec la chaine "0" ou avec la chaine "1"

2. Etapes d'exécution de *BinRecursive(13)*

   1er appel : BinRecursive(13) = BinRecursive (6)+"1"   (car 13=2*6+1)
   2ème appel : BinRecursive(6) = BinRecursive (3)+"0"   (car 6=2*3+0)
   3ème appel : BinRecursive (3)= BinRecursive (1)+"1"   (car 3=2*1+1)
   4ème appel : BinRecursive (1)="1"

Remonté après le dernier appel
BinRecursive (3)= BinRecursive (1)+"1" = "11"
BinRecursive(6)= BinRecursive (3)+"0" = "110"
BinRecursive(13)= BinRecursive(6)+"1" = "1101"
D'où finalement l'appel de BinRecursive(13) va retourner la chaine "1101"

3. **Comme la récursivité de la fonction BinRécursive n'est pas terminale, alors on ne peut pas utiliser la méthode qu'on a vu dans le cours pour transformer la fonction BinRécursive en une fonction itérative équivalente. D'où on va créer une fonction itérative qui effectue le meme travail que BinRécursive.**
Une fonction itérative équivalente à la fonction BinRecursive(n) est donnée par :

```
Fonction BinItérative(n : Entier) : Chaine
Var     q, r : Entier
        codeBinaire : Chaine
Début
        codeBinaire ← ""        //initialement codeBinaire est la chaine vide
        q ← n
        Tant que (q>1) Faire
                r ← q Mod 2
                q ← q Div 2
                Si (r=0 ) Alors
                        codeBinaire ← "0" + codeBinaire        //concaténation des deux chaines
                Sinon
                        codeBinaire ← "1" + codeBinaire
                Fin Si
        Fin Tant que
        Si (q=0 ) Alors
                codeBinaire ← "0" + codeBinaire
        Sinon
                codeBinaire ← "1" + codeBinaire
        Fin Si
        Retourner codeBinaire
Fin
```