1ALGO Algorithmique et logique

Gaël Roustan (Argonaultes)

2025

Abstract

TD 6

Contents

TD6 **3**
Problème du nombre parfait 3
Palindrome 5
Anagrammes 6
Convertisseurs entre nombres romains et nombres arabes 7
Traitement 1 7
Traitement 2 8
Tri à bulle (bubble sort) 9

Argonaultes, RCS PARIS 978 665 909, NAF : 85.59A
SARL au capital de 3 000 euros,
enregistrée sous le numéro 11756794275 auprès du préfet de région d'Ile-de-France

TD6

Problème du nombre parfait

Écrire un algorithme affichant tous les nombres parfaits jusqu'à un entier n saisi par l'utilisateur. Pour rappel, un nombre est parfait s'il est égal à la somme de ses diviseurs stricts (c'est à dire excepté lui-même).Nous découperons le problème en construisant des procédures et fonction selon le plan suivant :

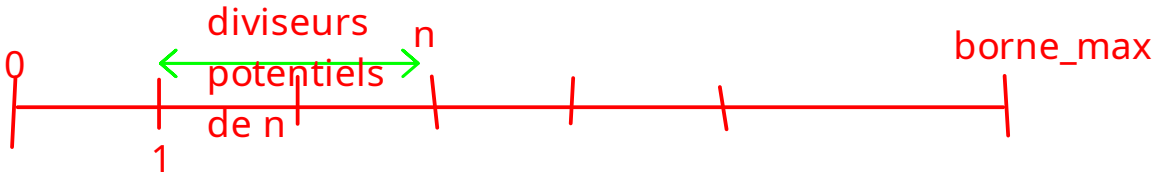
- afficherDesNombresParfaits
  - obtenirBorneMax
  - afficherNombresParfaitsJusquA
    - estParfait
    - sommeDesDiviseurs
    - estUnDiviseur

Exemple d'un nombre parfait : 6.

Les diviseurs de 6 sont : 1, 2, 3 et 6

La somme des diviseurs exceptés lui-même est : 1 + 2 + 3 = 6

```
ALGORITHME: nombre parfait
FONCTION : obtenirBorneMax() : ENTIER
VARIABLES :
    borne_max : ENTIER
DEBUT
    borne_max <-- LIRE()
    RETOURNER : borne_max
FIN

FONCTION : estUnDiviseur(ENTREE : n, m : ENTIER) : BOOLEEN
DEBUT
    RETOURNER : MOD(n, m) = 0
FIN

FONCTION : sommeDesDiviseurs(n : ENTIER) : ENTIER
VARIABLES :
    sommeDiviseurs <-- 0 : ENTIER
    d : ENTIER
DEBUT
    POUR d ALLANT DE 1 A  n - 1 AU PAS DE 1 FAIRE
        SI (estUnDiviseur(n, d)) ALORS
            sommeDiviseurs <-- sommeDiviseurs + d
        FINSI
    FINPOUR
FIN
```



```
FONCTION : estParfait(n : ENTIER) : BOOLEEN
DEBUT
    RETOURNER : sommeDesDiviseurs(n) = n
FIN

PROCEDURE : afficherNombresParfaitsJusquA(borne_max : ENTIER)
VARIABLES :
    n : ENTIER
DEBUT
    POUR n ALLANT DE 0 A borne_max AU PAS DE 1 FAIRE
        SI (estParfait(n)) ALORS
            ECRIRE(n)
        FINSI
    FINPOUR
FIN

PROGRAMMEPRINCIPAL : les nombres parfaits
VARIABLES :
    borne_max : ENTIER
DEBUT
    borne_max <-- obtenirBorneMax()
    afficherNombresParfaitsJusquA(borne_max)
FIN
```

Palindrome

Écrire un algorithme qui détermine si une chaine de caractères donnée est un palindrome. Rappelons qu'un palindrome est un mot qui se lit "dans les deux sens", comme par exemple "Kayak".

elle

```
FONCTION : palindrome(ENTREE : mot : CHAINE) : BOOLEN
VARIABLES :
    limite, longueur : ENTIERS
    indice <-- 0 : ENTIER
    lettre_reference, lettre_comparaison : CARACTERES
DEBUT
    longueur <-- LONGUEUR(mot)
    limite <-- DIV(longueur, 2)

    TANTQUE (indice <= limite) FAIRE
        lettre_reference <-- EXTRACTION(mot, indice)
        lettre_comparaison <-- EXTRACTION(mot, longueur - indice - 1)
        SI (lettre_reference = lettre_comparaison) ALORS
            indice <-- indice + 1
        SINON
            RETOURNER : Faux
        FINSI
    FINTANTQUE

    RETOURNER : Vrai
FIN
```

```
FONCTION : palindrome(ENTREE : mot : CHAINE) : BOOLEN
VARIABLES :
    limite, longueur : ENTIERS
    indice <-- 0 : ENTIER
    lettre_reference, lettre_comparaison : CARACTERES
    derniere_comparaison <-- Vrai : BOOLEEN
DEBUT
    longueur <-- LONGUEUR(mot)
    limite <-- DIV(longueur, 2)

    TANTQUE (derniere_comparaison ET indice <= limite) FAIRE
        lettre_reference <-- EXTRACTION(mot, indice)
        lettre_comparaison <-- EXTRACTION(mot, longueur - indice - 1)
        SI (lettre_reference = lettre_comparaison) ALORS
            indice <-- indice + 1
        SINON
            derniere_comparaison <-- Faux
        FINSI
    FINTANTQUE

    RETOURNER : derniere_comparaison
FIN
```

```
FONCTION : palindrome(ENTREE : mot : CHAINE) : BOOLEN
VARIABLES :
    limite, longueur : ENTIERS
    indice <-- 0 : ENTIER
    lettre_reference, lettre_comparaison : CARACTERES
    derniere_comparaison <-- Vrai : BOOLEEN
DEBUT
    longueur <-- LONGUEUR(mot)
    limite <-- DIV(longueur, 2)

    TANTQUE (derniere_comparaison ET indice <= limite) FAIRE
        lettre_reference <-- EXTRACTION(mot, indice)
        lettre_comparaison <-- EXTRACTION(mot, longueur - indice - 1)
        indice <-- indice + 1
        derniere_comparaison <-- lettre_reference = lettre_comparaison
    FINTANTQUE

    RETOURNER : derniere_comparaison
FIN
```


Argonaultes, RCS PARIS 978 665 909, NAF : 85.59A
SARL au capital de 3 000 euros,
enregistrée sous le numéro 11756794275 auprès du préfet de région d'Ile-de-France

Anagrammes

Écrire un algorithme qui détermine si deux mots donnés sont des anagrammes. Rappelons que des anagrammes sont des mots ayant exactement les mêmes lettres, comme par exemple “logarithme” et “algorithme”.

```
ALGORITHME: anagramme_avec_destruction_lettre
FONCTION: anagramme(ENTREE : mot1, mot2 : CHAINES) : BOOLEEN
VARIABLES :
    TABLEAU : tab[0..1][0..LONGUEUR(mot1) - 1] : CARACTERES
    lettre_a_detruire : CARACTERE
    indice, indice2 : ENTIER
    lettre_pas_detruite, est_anagramme : BOOLEEN
DEBUT
    SI (LONGUEUR(mot1) <> LONGUEUR(mot2)) ALORS
        RETOURNER : Faux
    FINSI
    POUR indice ALLANT DE 0 A LONGUEUR(mot1) - 1 AU PAS DE 1
        tab[0][indice] <-- EXTRACTION(mot1, indice,1)
        tab[1][indice] <-- EXTRACTION(mot2, indice,1)
    FINPOUR
    POUR indice ALLANT DE 0 A LONGUEUR(mot1) - 1 AU PAS DE 1 FAIRE
        lettre_a_detruire <-- tab[0][indice]
        tab[0][indice] <-- ''
        lettre_pas_detruite <-- Vrai
        indice2 <-- 0
        TANTQUE (lettre_pas_detruite ET indice2 < LONGUEUR(mot2)) FAIRE
            SI (lettre_a_detruire = tab[1][indice2]) ALORS
                tab[1][indice2] <-- ''
                lettre_pas_detruite <-- Faux
            FINSI
        FINTANTQUE
    FINPOUR
    POUR indice ALLANT DE 0 A LONGUEUR(mot1) - 1 AU PAS DE 1 FAIRE
        est_anagramme <-- tab[0][indice] = tab[1][indice] ET tab[0][indice] = ''
    FINPOUR
    RETOURNER est_anagramme
FIN
```


Argonaultes, RCS PARIS 978 665 909, NAF : 85.59A
SARL au capital de 3 000 euros,
enregistrée sous le numéro 11756794275 auprès du préfet de région d’Ile-de-France

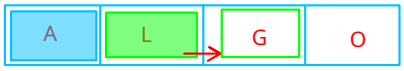
lettre_a_detruire <-- 'L'

''	O	G	A
A	L	G	O

Anagrammes

Écrire un algorithme qui détermine si deux mots donnés sont des anagrammes. Rappelons que des anagrammes sont des mots ayant exactement les mêmes lettres, comme par exemple “logarithme” et “algorithme”.


```
ALGORITHME: anagramme_avec_tri_selection
```




```
FONCTION: tri_alphabetique(ENTREE : mot : CHAINE) : CHAINE
VARIABLES:
    indice, compteur, indice_lettre_min : ENTIERS
    lettre_ref, lettres_avant_ref, lettre, lettre_avant_min, lettre_apres_min : CHAINES
DEBUT
    POUR indice ALLANT DE 0 A LONGUEUR(mot) - 1 AU PAS DE 1 FAIRE
        lettre_ref <-- EXTRACTION(mot, indice, 1)
        indice_lettre_min <-- indice
        POUR compteur ALLANT DE 1 A LONGUEUR(mot) - 1 - indice AU PAS DE 1 FAIRE
            lettre_a_tester <-- EXTRACTION(mot, indice + compteur, 1)
            SI (lettre_a_tester < lettre_ref) ALORS
                indice_lettre_min <-- indice + compteur
            FINSI
        FINPOUR
        lettres_avant_ref <-- EXTRACTION(mot, 0, indice - 1)
        lettre <-- EXTRACTION(mot, indice_lettre_min, 1)
        lettre_avant_min <-- EXTRACTION(mot, indice + 1, indice_lettre_min - indice)
        lettre_ref <-- lettre_ref
        lettre_apres_min <-- EXTRACTION(mot, indice_lettre_min + 1, LONGUEUR(mot) - indice_lettre_min)
```

```
        mot <-- CONCATENER(
            CONCATENER(
                CONCATENER(
                    CONCATENER(lettres_avant_ref, lettre),
                    lettre_avant_min),
                lettre_ref),
            lettre_apres_min)
    FINPOUR
    RETOURNER : mot
FIN

FONCTION: anagramme(ENTREE : mot1, mot2 : CHAINES) : BOOLEEN
DEBUT
    mot1 <-- tri_alphabetique(mot1)
    mot2 <-- tri_alphabetique(mot2)
    RETOURNER : LONGUEUR(mot1) = LONGUEUR(mot2) ET mot1 = mot2
FIN
```

Convertisseurs entre nombres romains et nombres arabes

Traitement 1

Écrire un algorithme qui convertit un nombre écrit en chiffres arabes en un nombre écrit en chiffres romains.

Argonaultes, RCS PARIS 978 665 909, NAF : 85.59A
SARL au capital de 3 000 euros,
enregistrée sous le numéro 11756794275 auprès du préfet de région d'Ile-de-France

Traitement 2

Écrire un algorithme qui réalise l’opération inverse.

Argonaultes, RCS PARIS 978 665 909, NAF : 85.59A
SARL au capital de 3 000 euros,
enregistrée sous le numéro 11756794275 auprès du préfet de région d’Ile-de-France

Tri à bulle (bubble sort)

Pour effectuer le tri du tableau, nous allons utiliser l'algorithme appelé `tri à bulles`. Nous ne l'avons pas vu en cours mais l'explication ci-après ainsi que les schémas détaillant son fonctionnement pas à pas doivent vous permettre d'implémenter ce tri en structure algorithmique.

Soit un tableau de n éléments non triés. Les n éléments sont saisis par l'utilisateur.

3	1	6	9	7

Le principe est de parcourir le tableau à la recherche du plus grand élément et de le placer à la fin du tableau.

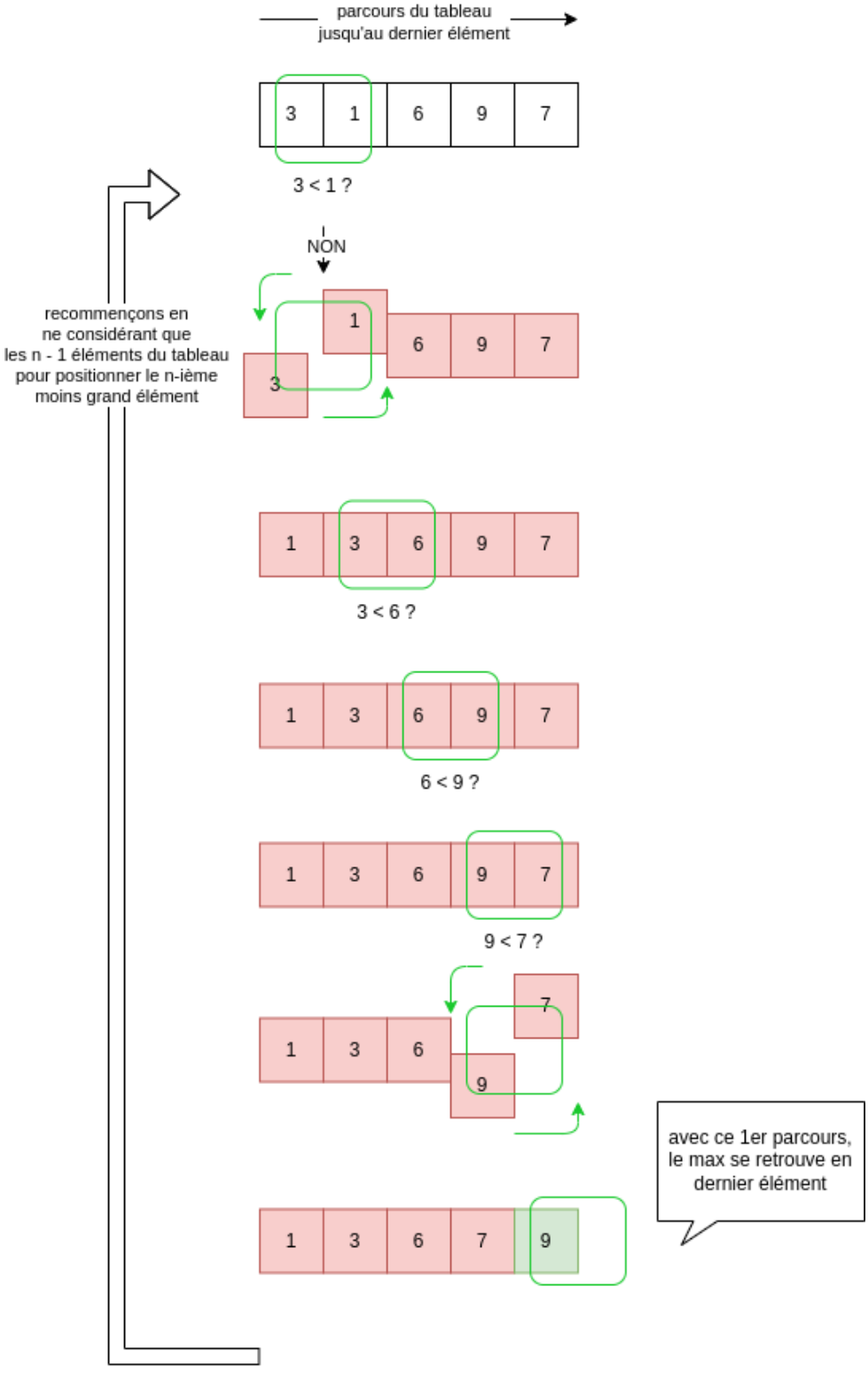
Pour cela, il faut considérer chaque élément et le comparer avec l'élément suivant. Si l'élément suivant est plus petit, alors il faut échanger sa place avec l'élément en cours. Il faut répéter cette opération jusqu'à atteindre la dernière case du tableau. Nous avons trouvé le plus grand élément du tableau.

Nous recommençons le parcours du tableau pour identifier le 2ème plus grand élément du tableau, il faudra donc s'arrêter à l'avant dernière case, puisque nous savons déjà que la dernière case contient le plus grand élément.

Nous arrêtons de parcourir le tableau lorsque la case du plus grand élément à chercher correspond l'indice à la première position du tableau.

Proposer une implémentation de cet algorithme de tri à bulles. Sur la page suivante vous est présenté en image l'algorithme de tri à bulles.

Argonaultes, RCS PARIS 978 665 909, NAF : 85.59A
SARL au capital de 3 000 euros,
enregistrée sous le numéro 11756794275 auprès du préfet de région d'Ile-de-France

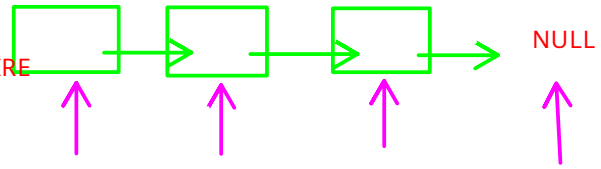


Argonaultes, RCS PARIS 978 665 909, NAF : 85.59A
SARL au capital de 3 000 euros,
enregistrée sous le numéro 11756794275 auprès du préfet de région d'Ile-de-France

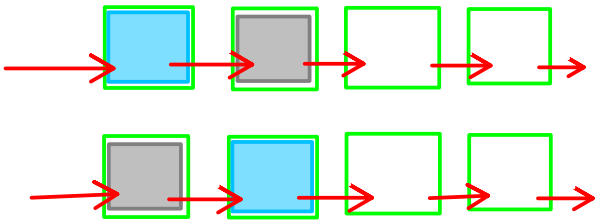
Nom	Adresse	Valeur
nb_elements	0x0	5
var_element	0x1	0x4
compteur	0x2	1
valeur	0x3	3
e1.valeur	0x4	3
e1.element_suivant	0x5	NULL
e2.valeur	0x6	1
e2.element_suivant	0x7	NULL
dernier_element_ajoute	0x8	0x6

```
ALGORITHME: bubble_sort

ENREGISTREMENT : Element
DEBUTENREGISTREMENT
    valeur : Entier
    *element_suivant <-- NULL : Element
FINENREGISTREMENT

FONCTION : initialiser_liste() : *Element
VARIABLES
    nb_elements, compteur, valeur : ENTIER
    *var_element <-- NULL : Element
    *dernier_element_ajoute <-- NULL : Element
DEBUT
    nb_elements <-- LIRE()
    POUR compteur ALLANT DE 0 A nb_elements - 1 AU PAS DE 1 FAIRE
        valeur <-- LIRE()
        SI (var_element = NULL) ALORS
            var_element <-- NOUVEAU : Element
            (*var_element).valeur <-- valeur
            dernier_element_ajoute <-- var_element
        SINON
            SI (dernier_element_ajoute <> NULL) ALORS
                (*dernier_element_ajoute).element_suivant <-- NOUVEAU : Element
                dernier_element_ajoute <-- (*dernier_element_ajoute).element_suivant
                (*dernier_element_ajoute).valeur <-- valeur
            FINSI
        FINSI
    FINPOUR
    RETOURNER : var_element
FIN

FONCTION : obtenir_element(ENTREE : *liste : Element, indice : ENTIER) : *Element
VARIABLES:
    *maillon_liste <-- liste : Element
DEBUT
    POUR element ALLANT DE 0 A indice - 1 AU PAS DE 1 FAIRE
        SI (maillon_liste <> NULL) ALORS
            maillon_liste <-- (*maillon_liste).element_suivant
        FINSI
    FINPOUR
    RETOURNER maillon_liste
FIN
```

```
FONCTION : obtenir_element(ENTREE : *liste : Element, indice : ENTIER) : *Element
VARIABLES:
    *maillon_liste <-- liste : Element
    position_element <-- 0 : ENTIER
DEBUT
    TANTQUE maillon_liste <> NULL ET  position_element <> indice FAIRE
            maillon_liste <-- (*maillon_liste).element_suivant
            position_element <-- position_element + 1
    FINTANTQUE

    RETOURNER : maillon_liste
FIN
```




```
PROCEDURE : echanger_2_elements(ENTREE : *element1 : Element, *element2 : Element)
VARIABLES:
    tmp : ENTIER
DEBUT
    SI (element1 <> NULL ET element2 <> NULL) ALORS
        tmp <-- (*element1).valeur
        (*element1).valeur <-- (*element2).valeur
        (*element2).valeur <-- tmp
    FINSI
FIN
```



```
PROCEDURE : echanger_2_elements(ENTREE : position1, position2 : ENTIERS, *liste : Enregistrement)

VARIABLES :
    *element_bleu, *element_gris, *element_tmp : Element
DEBUT
    element_bleu <-- obtenir_element(liste, position1)
    element_gris <-- obtenir_element(liste, position2)

    SI (element_bleu = liste) ALORS
        liste <-- element_gris
    FINSI

    element_tmp <-- (*element_gris).element_suivant
    (*element_gris).element_suivant <-- element_bleu
    (*element_bleu).element_suivant <-- element_tmp

FIN


PROCEDURE : tri_a_bubulle()
VARIABLES:
    *liste : Element
DEBUT

    liste <-- initialiser_liste()
    TANTQUE maillon_liste <> NULL (*maillon_liste).element_suivant <> NULL

FIN
```