

***L2 X22I020***

*Algorithmique et*

*Structures de données 2*

*Projet*

*2020\_2021*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ATALLA Salim

L2 – Informatique\_Groupe\_485k

Sujet

L’objective de ce projet est de réaliser une structure de donnée *fint* pour les factorisations des entiers.

Le travail est constitué de trois branches :

1. La définition d’une SDA (Structure de Donnée Abstraite) munie des opérations.
2. La réalisation d’une SDC (Structure de Donnée Concrète) en visant les meilleures complexités possibles.
3. La programmation d’une classe *fint* en C++ en respectant les codes de la programmation orientée objets.

Les opérations de cette classe sont :

Etant donnés deux *fint* a et b et un entier strictement positif n :

* Création d’un *fint* à partir de n
* Tester si a divise b
* Calculer a × b
* Calculer an
* Calculer a div b
* Calculer a mod b
* Calculer a div b
* Calculer le plus grand diviseur commun : gcd (a, b)
* Calculer le plus petit multiple commun : lcm (a, b)

Remarque : le fichier source des opérations de la classe *fint* est nommé : fint.cpp

**Fint** est la liste des facteurs premiers d'un entier avec leur multiplicité, il est constitué d’un entier nb\_couples, et d’un vecteur de couples tab\_couples,

**Un couple** est un enregistrement constitué d’un grand entier strictement positif int\_t fact, et d’un entier strictement positif mult\_t mult.

Les types utilisés :

|  |  |
| --- | --- |
| Type int\_t = grand entier positif  Type mult\_t = entier positif | Type Couple = enregistrement  int\_t fact  mult\_t mult  Fin |

|  |
| --- |
| Classe fint = enregistrement  // Attributs  entier nb\_couples  vecteur de Couple tab\_couples  // Constructeurs  fint (int\_t d n)  fint (liste de int\_t d lf, liste de mult\_t d lm)  // Destructeur  ~fint ()  // Méthodes  fonction to\_int () : int\_t  fonction divides (d a : fint) : booléen  fonction is\_prime () : booléen  fonction operator== (d a : fint, d b : fint) : booléen  fonction operator!= (d a : fint, d b : fint) : booléen  fonction lcm (d a : fint, d b : fint) : fint  fonction gcd (d a : fint, d b : fint) : fint  fonction operator\* (d a : fint, d b : fint) : fint  fonction operator/ (d a : fint, d b : fint) : fint  fonction pow (d a : fint, d n : entier positif) : fint  fonction operator<< (os : ostream, d a : fint) : ostream  Fin |

Variables globales :

MAX\_INT\_T : int\_t // la limite autorisée pour l’objet fint

Remarque : au-dessous, et pour calculer la complexité temporelle, on suppose que n est la taille de la liste.

// Constructeur de l’objet fint à partir d’un entier n strictement positif

SDA :

* Signature : fint (d n : entier)
* Rôle : construire un objet fint à partir d’un entier n donné.
* Précondition : n ≥ 1
* Sortie : un objet fint contient la liste des facteurs premiers de n et la taille de cette liste.

SDC :

Complexité : Ω(1), O(2n + n) ∈ O(2n)

Variables : f : vecteur de int\_t, m : vecteur de mult\_t,

taille, i : entier, couple : Couple

Début

Si (n < 1 ou n > MAX\_INT\_T) alors

ERREUR

Sinon

decomposition (n, f, m) // Θ(2n)

taille 🡨 f.longueur

nb\_couples 🡨 taille

Pour i de 0 à taille-1 faire

couple.fact = f[i]

couple.mult = m[i]

tab\_couples.push\_back(couple)

finPour

finSi

Fin

-------------------------------------------------------------------------------------

// Destructeur

SDA :

* Signature : ~fint ()
* Rôle : détruire l’objet fint
* Précondition : aucune
* Sortie : aucune

SDC :

Complexité : Ω(1), O(n)

Début

tab\_couples.supprimer()

Fin

-------------------------------------------------------------------------------------

// Constructeur de l'objet font à partir de deux listes prés initialiser selon la // condition de l’ordonnancement croissant de la liste des facteurs

SDA :

* Signature : fint (d lf, lm : liste de int\_t)
* Rôle : construire un objet fint à partir d’une liste de facteurs et une liste des multiplicités.
* Précondition : les listes sont ordonnées selon la liste des facteurs par un ordre croissant.
* Sortie : un objet fint contient le vecteur des couples et le nombre des couples.

SDC :

Complexité : Ω(1), O(2n) ∈ O(n)

Variables : i, taille : entier, couple : Couple

Début

i 🡨 0

Tantque (i < lf.longueur

et lf(i) < lf(i+1)) faire

i 🡨 i + 1

finTanque

Si (i != lf.longueur-1) alors

ERREUR

finSi

taille 🡨 lf.longueur

nb\_couples 🡨 taille

Pour i de 0 à taille-1 faire

couple.fact 🡨 lf(i)

couple.mult 🡨 lm(i)

tab\_couples.push\_back(couple)

finPour

Fin

-------------------------------------------------------------------------------------

// Décomposer un entier à ses facteurs premiers

SDA :

* Signature : procédure decomposition (d n : int\_t, m lf : vecteur de int\_t, m lm : vecteur de mult\_t)
* Rôle : Décomposer un entier à ses facteurs premiers.
* Précondition : lf et lm sont vides et n > 0
* Sortie : faire des modifications sur les vecteurs lf et lm.

SDC :

Complexité : Ω(1), O(2n)

Variables : i, j, prec : int\_t, checked : booléen

Début

i 🡨 2

prec 🡨 0

j 🡨 0

checked 🡨 faux

Tantque (n ≥ 2) faire

Tantque (n mod i = 0) faire

Si (i != prec) alors

if.push\_back(i)

checked 🡨 vrai

finSi

n 🡨 n div i

prec 🡨 i

j 🡨 j + 1

finTantque

Si (checked = vrai) alors

lm.push\_back(j)

checked 🡨 faux

finSi

j 🡨 0

i 🡨 i + 1

finTanque

Fin

-------------------------------------------------------------------------------------

// Transformation en décimale

SDA :

* Signature : fonction to\_int () : int\_t
* Rôle : transformer l’objet fint en décimale.
* Précondition : le nombre fint ne doit pas dépasser la limite.
* Sortie : un entier du type int\_t représente le nombre fint en décimale.

SDC :

Complexité : Ω(1), O(m) où m est la somme des multiplicités

Variables : val, fVal, mVal : int\_t

taille, i, j : entier

Début

val 🡨 1

taille 🡨 nb\_couples

Pour i de 0 à taille-1 faire

fVal 🡨 tab\_couples[i].fact

mVal 🡨 tab\_couples[i].mult

Pour j de 0 à mVal-1 faire

val 🡨 val \* fVal

Si (val > MAX\_INT\_T) alors

ERREUR

finSi

finPour

finPour

retourner val

Fin

-------------------------------------------------------------------------------------

// Tester si l’objet divise a (où a est un autre objet fint)

SDA :

* Signature : fonction divides (d a : fint) : booléen
* Rôle : tester si l’objet divise un autre objet du même type.
* Précondition : aucune
* Sortie : vrai si l’objet divise a, faux sinon.

SDC :

Complexité : Ω(1), O(n × m) où m est le longueur de la liste de a

Variables : i, j : entier

Début

Si (a.nb\_couples ≥ nb\_couples) alors

Pour i de 0 à nb\_couples-1 faire

j 🡨 0

Tantque (j ≤ a.nb\_couples

et tab\_couples[i].fact != a.tab\_couples[j].fact) faire

j 🡨 j + 1

finTantque

Si (j ≥ a.nb\_couples

ou tab\_couples[i].mult > a.tab\_couples[j].mult) alors

retourner faux

finSi

finPour

sinon

retourner faux

finSi

retourner vrai

Fin

-------------------------------------------------------------------------------------

// Tester si l’objet est premier

SDA :

* Signature : fonction is\_prime () : booléen
* Rôle : tester si l’objet est premier.
* Précondition : aucune
* Sortie : vrai si l’objet est premier, faux sinon.

SDC :

Complexité : Ω(1), O(1)

Variables : aucune

Début

retourner nb\_couple = 1

et tab\_couples[0].mult = 1

et tab\_couples[0].fact ≥ 2

Fin

-------------------------------------------------------------------------------------

// Les opérations :

// Opérateur ==

SDA :

* Signature : fonction operator==(d a : fint, d b : fint) : booléen
* Rôle : tester si a et b sont égaux.
* Précondition : aucune
* Sortie : vrai si a et b sont égaux, faux sinon.

SDC :

Complexité : Ω(1), O(n)

Variables : i : entier

Début

Si (a.nb\_couples = b.nb\_couples) alors

i 🡨 a.nb\_couples-1

Tantque (a.tab\_couples[i].fact = b.tab\_couples[i].fact

et a.tab\_couples[i].mult = b.tab\_couples[i].mult

et i ≥ 0) faire

i 🡨 i – 1

finTantque

Si (i < 0) alors

retourner vrai

finSi

finSi

retourner faux

Fin

-------------------------------------------------------------------------------------

// Opérateur !=

SDA :

* Signature : fonction operator!=(d a : fint, d b : fint) : booléen
* Rôle : tester si a et b ne sont pas égaux.
* Précondition : aucune
* Sortie : vrai si a et b ne sont pas égaux, faux sinon.

SDC :

Complexité : Ω(1), O(n)

Variables : aucune

Début

retourner !(a = b)

Fin

-------------------------------------------------------------------------------------

// Le plus petit commun multiple PPCM ou LCM

SDA :

* Signature : fonction lcm (d a : fint, d b : fint) : fint
* Rôle : calculer le plus petit commun multiple entre a et b.
* Précondition : aucune
* Sortie : retourner lcm(a, b).

SDC :

Complexité : Ω(1), O(n × m) ssi il n’y a aucun facteur commun entre a et b

Variables : v\_A, v\_B, v\_c : vecteur de Couple

c : Couple, nb, i, j : entier, tmp : fint

Début

v\_A 🡨 a.tab\_couples

v\_B 🡨 b.tab\_couples

nb 🡨 0

i 🡨 0

j 🡨 0

Tantque (v\_A.longueur > i ou v\_B.longueur > j) faire

Si (v\_A[i].fact = v\_B[j].fact) alors

c.fact 🡨 v\_A[i].fact

Si (v\_A[i].mult > v\_B[j].mult) alors

c.mult 🡨 v\_A[i].mult

Sinon

c.mult 🡨 v\_B[j].mult

finSi

i 🡨 i + 1

j 🡨 j + 1

Sinon

Si (v\_A[i].fact < v\_B[j].fact) alors

c 🡨 v\_A[i]

i 🡨 i + 1

Sinon

c 🡨 v\_B[j]

j 🡨 j + 1

finSi

finSi

v\_c.push\_back(c)

nb 🡨 nb + 1

finTantque

tmp.nb\_couples 🡨 nb

tmp.tab\_couples 🡨 v\_c

retourner tmp

Fin

-------------------------------------------------------------------------------------

// Le plus grand commun diviseur PGCD ou GCD

SDA :

* Signature : fonction gcd (d a : fint, d b : fint) : fint
* Rôle : calculer le plus grand commun diviseur entre a et b.
* Précondition : aucune
* Sortie : retourner gcd(a, b).

SDC :

Complexité : Ω(1), O(n × m)

Variables : nb, i, j : entier, tmp : fint

v\_c : vecteur de Couple, c : Couple

Début

tmp 🡨 a

Pour i de 0 à a.nb\_couples-1 faire

j 🡨 0

Tantque (j ≤ a.nb\_couples

et a.tab\_couples[i].fact != b.tab\_couples[j].fact) faire

j 🡨 j + 1

finTantque

Si (j < b.nb\_couples) alors

Si (a.tab\_couples[i].mult > b.tab\_couples[j].mult) alors

tmp.tab\_couples[i].mult 🡨 b.tab\_couples[j].mult

finSi

Sinon

tmp.tab\_couples[i].mult 🡨 0

finSi

finPour

// nettoyage le vecteur

nb 🡨 0

Pour i de 0 à tmp.nb\_couples-1 faire

Si (tmp.tab\_couples[i].mult != 0) alors

c 🡨 tmp.tab\_couples[i]

v\_c.push\_back(c)

nb 🡨 nb + 1

finSi

finPour

tmp.tab\_couples 🡨 v\_c

tmp.nb\_couples 🡨 nb

retourner tmp

Fin

-------------------------------------------------------------------------------------

// Opérateur ×

SDA :

* Signature : fonction operator\*(d a : fint, d b : fint) : fint
* Rôle : calculer la multiplication pour deux fint.
* Précondition : aucune
* Sortie : retourner a \* b

SDC :

Complexité : Ω(1), O(n × m) ssi il n’y a aucun facteur commun entre a et b

Variables : v\_A, v\_B, v\_c : vecteur de Couple

c : Couple, nb, i, j : entier, tmp : fint

Début

v\_A 🡨 a.tab\_couples

v\_B 🡨 b.tab\_couples

nb 🡨 0

i 🡨 0

j 🡨 0

Tantque (v\_A.longueur > i ou v\_B.longueur > j) faire

Si (v\_A[i].fact = v\_B[j].fact) alors

c.fact 🡨 v\_A[i].fact

c.mult 🡨 v\_A[i].mult + v\_B[j].mult

i 🡨 i + 1

j 🡨 j + 1

Sinon

Si (v\_A[i].fact < v\_B[j].fact) alors

c 🡨 v\_A[i]

i 🡨 i + 1

Sinon

c 🡨 v\_B[j]

j 🡨 j + 1

finSi

finSi

v\_c.push\_back(c)

nb 🡨 nb + 1

finTantque

tmp.nb\_couples 🡨 nb

tmp.tab\_couples 🡨 v\_c

retourner tmp

Fin

-------------------------------------------------------------------------------------

// Opérateur div

SDA :

* Signature : fonction operator/(d a : fint, d b : fint) : fint
* Rôle : calculer la division pour deux fint.
* Précondition : a ≥ b
* Sortie : retourner a / b

SDC :

Complexité : Ω(1) en cas d’erreur, O(2n + m + 2k) ∈ O(2k)

Variables : i : entier, val\_A, val\_B, f : int\_t, m : mult\_t

Début

Si (a = b) alors // Θ(n) au pire a.longueur = b.longueur mais a ≠ b

retourner fint (1)

Sinon

val\_A 🡨 a.to\_int() // Θ(m) où m supérieur ou égale à a.longueur

val\_B 🡨 1

Pour i de 0 à b.nb\_couples-1 faire // Θ(n) où n égale à b.longueur

val\_B 🡨 fm

val\_A 🡨 val\_A div val\_B

Si (val\_A < 1) alors

ERREUR

finSi

finPour

retourner fint (val\_A) // Θ(2k) où k est le nombre des facteurs de

// l’objet créé

finSi

Fin

-------------------------------------------------------------------------------------

// Opérateur mod

SDA :

* Signature : fonction operator%(d a : fint, d b : fint) : fint
* Rôle : calculer le modulo pour deux fint.
* Précondition : aucune
* Sortie : retourner a % b

SDC :

Complexité : Ω(n × m) en cas d’erreur, O(n + m + 2k) ∈ O(2k)

Variables : val\_A, val\_B : int\_t

Début

Si (b.divides(a)) alors // Θ(n × m)

ERREUR

finSi

val\_A 🡨 a.to\_int() // Θ(n) où n le somme des multiplicités de a

val\_B 🡨 b.to\_int() // Θ(m) où m le somme des multiplicités de b

retourner fint (val\_A % val\_B) // Θ(2k) où k est le nombre des facteurs

// de l’objet créé

Fin

-------------------------------------------------------------------------------------

// Opérateur pow

SDA :

* Signature : fonction pow (d a : fint, d n : entier positif) : fint
* Rôle : calculer la puissance pour un fint.
* Précondition : aucune
* Sortie : retourner an

SDC :

Complexité : Ω(1), O(n)

Variables : i : entier, tmp : fint

Début

Si (n = 1) alors

retourner a

finSi

tmp 🡨 a

Pour i de 0 à a.nb\_couples-1 faire

tmp.tab\_couples[i].mult 🡨 tmp.tab\_couples[i].mult \* n

finPour

retourner tmp

Fin

Conclusion :

Ce projet a pour but de représenter les nombres entiers sous forme d‘une liste contenant les facteurs premiers et leurs multiplicités,

Cette liste peut être représenter en différentes façons, par exemple : par un tableau dynamique, un dictionnaire, ou encore une liste chaînée.

La différence entre les tableaux dynamiques et les listes chaînées est au niveau de la complexité temporelle. Les tableaux dynamiques sont idéals pour récupérer les éléments du tableau, mais ils sont complexes pour ajouter ou supprimer un élément.

Par contre, à l’inverse des tableaux dynamiques, les listes chaînées sont idéals pour ajouter ou supprimer les éléments d’une liste mais elles sont complexes pour récupérer un élément.

En regardant le sujet donné, on peut remarquer que les tableaux dynamiques sont plus compatibles avec le sujet que les listes chaînées, parce que, pour faire les opérations sur l’objet fint, on aura souvent besoin de récupérer des éléments de la liste, et donc en conséquence, on a utilisé l’objet « vector » qui représente un tableau dynamique.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_