HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Matrices binaires

LABORATOIRE 5

Quentin Forestier, Herzig Melvyn

POO1 – 19.10.2020

Table des matières

Introduction

Ce projet met en œuvre une classe Matrice permettant de créer des matrices de taille M x N avec des valeurs modulo P.

Pour des raisons de simplification, le terme « utilisateur » sera utilisé dans ce document pour parler de l'utilisateur de la classe.

Attributs

Une matrice est définie par son contenu que nous stockons dans un tableau d'entier à 2 dimensions et son modulo en entier.

Trois méthodes ont été mises en place pour récupérer la hauteur et la largeur de la matrice ainsi que le modulo.

Constructeurs

Une matrice peut être construite de deux manières :

- **Contenu aléatoire** : L'utilisateur passe la taille MxN et le modulo P en argument. Le contenu est généré aléatoirement modulo P.

  En extension, un constructeur à deux arguments est fourni. Il prend une taille M et le modulo P. Son comportement est similaire, mais il permet de construire une matrice carrée. Il appelle le premier constructeur en dupliquant la taille M.

- **Contenu déterminé** : L'utilisateur passe une matrice m (tableau 2D de int) en argument ainsi que le modulo P. Le constructeur corrige la matrice reçue modulo P.

Nous avons permis les matrices vides (de taille 0x0). Les matrices modulo 1 sont admises, ce qui crée une matrice nulle.

Exceptions

Les constructeurs lèvent une *RuntimeException* si une dimension négative est reçue où en cas de modulo plus petit que 1.

Affichage

Le contenu de la matrice peut être affiché via la méthode toString surchargée (voir figure 1).

Pour un affichage détaillé des attributs de la matrice, l'utilisateur peut combiner la fonction toString avec les fonctions getWidth, getHeight et getModulus. Nous n'avons pas jugé pertinent de traiter ce cas nous-mêmes.

```
1143
0151
5101
```

Figure 1 Affchage d'une matrice

Opérations

Les opérations matricielles s'effectuent composante par composante.

Si les matrices opérantes sont de tailles différentes, la matrice résultante sera de taille max(M1,M2) x max(N1,N2) où M1,N1,M2,N2 sont les dimensions des matrices opérantes.

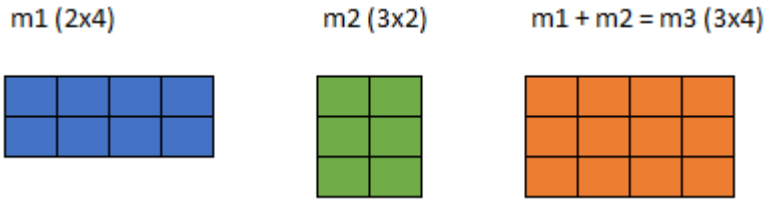

Figure 2 Exemple de redimensionnement sous opération

Le système d'opérations est modulable. Nous avons créé trois méthodes publics (add, sub, prod) qui appellent opération avec un objet operator en paramètre qui définit le traitement à effectuer entre les deux composantes des matrices.

L'objet operator est une énumération qui définit une méthode abstraite apply. Pour chaque élément de l'enum, la méthode apply est redéfinie pour le traitement souhaité. De cette manière, le traitement des opérations est centralisé et extensible.

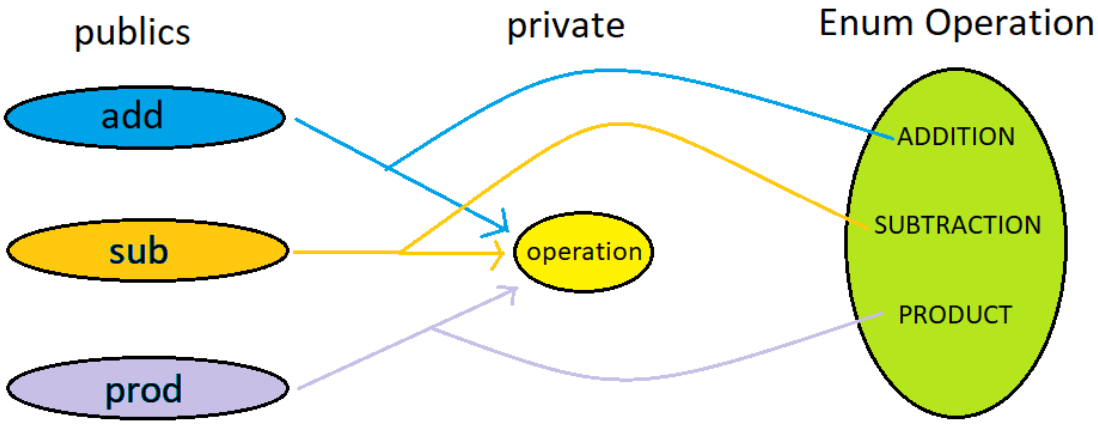

Figure 3 Modélisation des opérations matricielles

Exceptions

Si les matrices à traiter ont un modulo différent, une *RuntimeException* est levée.

Tests

Une classe Test est fournie. Une série de tests avec des matrices connues est effectuée. Une autre série de tests est effectuée selon les arguments que le programme reçoit. Il attend 5 arguments pour créer deux matrices, <M1> <N1> <M2> <N2> <modulo>. Si un argument est invalide ou manquant, une exception est levée. Finalement, des tests « limites » sont réalisés.

Résultats

Cette partie reprend les tests de la donnée. Nous avons reproduit les deux matrices one et two manuellement puis nous avons effectué les opérations à disposition et nous avons affiché le résultat.

Ce test est réussi, l'affichage toString s'effectue correctement ainsi que add, sub et prod.

```
------------ Test de la donnée ------------
The modulus is 5

one
1311
3242
1010


two
14232
01042
00202


one + two
22342
33412
10302


one - two
04433
31433
10403


one x two
12230
02030
00200
```

Pour ce test, nous avons passé au programme les arguments 2 2 3 4 6.

Ainsi, deux matrices three (2x2) four(3x4) ont été générées aléatoirement modulo 6.

La génération est correcte (valeur entre 0 et 5) ainsi que le résultat des opérations.

Ce test est réussi, l'affichage toString, les deux constructeurs de contenu aléatoire, les opérations.

```
------------ Test de l'entrée utilisateur ------------
The modulus is 6

three
41
24


four
3043
4351
5101


three + four
1143
0151
5101


three - four
1123
4115
1505


three x four
0000
2000
0000
```

Les tests suivants ont permis de vérifier le comportement de la classe avec des cas limites.

Ici encore, ils sont réussis, créer une matrice avec une taille négative où un module inférieur à 1 génère une *RuntimeException.*

Effectuer une opération entre deux matrices de modules différents lève une *RuntimeException.*

Créer (aléatoirement ou à partir d'un tableau 2D) et utiliser une matrice vide fonctionne.

```
------------ Test de cas limite ------------
Test de matrice avec une taille négative :
java.lang.RuntimeException: Argument(s) invalide(s)


Test de matrice avec un modulo incorrecte :
java.lang.RuntimeException: Le modulo est < 1


Test d'une opération avec un modulo différent :
java.lang.RuntimeException: Les modulos ne sont pas égaux


Test création d'une matrice vide + addition :
Matrice vide
1233
3231
0032

1233
3231
0032



Test création d'une matrice vide à partir d'un tableau vide
Matrice vide
```

Tous les tests ont été validés manuellement et jugés conformes.

Annexes

- Code source en pdf
  - Test.java
  - Matrice.java