TP sur la méthode du simplexe

Le travail consiste à modifier une classe nommée `Dictionniaire.java` qui modélise un dictionnaire. Les attributs de cette classe servant à coder le problème sont :

```
int nbVarBase;
int nbVarHorsBase;
int[] tabVarBase;
int[] tabVarHorsBase;
double[][] D;
```

Dans les tableaux `tabVarBase` et `tabVarHorsBase`, **on n'utilise pas les cases d'indice 0**.
Le tableau `tabVarBase` a pour dimension `nbVarBase + 1`.
Le tableau `tabVarHorsBase` a pour dimension `nbVarHorsBase + 1`.
La matrice `D` a pour dimension `(nbVarBase + 1) × (nbVarHorsBase + 1)`.

Les données du problème sont lues dans un fichier et correspondent à un problème mis sous forme standard. Il est nécessaire, pour travailler, de comprendre le codage retenu.
Si le problème est (dont le codage se trouve dans pb1.txt) :

maximiser $z = 4x_1 + 3x_2$

avec

$2x_1 + 3x_2 \leq 24$

$5x_1 + 3x_2 \leq 30$

$x_1 + 3x_2 \leq 18$

$x_1 \geq 0, x_2 \geq 0$

alors le fichier contient (première ligne : le nombre de variables et le nombre de contraintes autres que les contraintes de signe ; pour chacune des lignes suivantes sauf la dernière : les coefficients et la constante d'une contrainte ; dernière ligne : les coefficients des variables dans z) :

```
2 3
2 3 24
5 3 30
1 3 18
4 3
```

Le premier dictionnaire réalisable est :

$x_3 = 24 - 2x_1 - 3x_2$

$x_4 = 30 - 5x_1 - 3x_2$

$x_5 = 18 - x_1 - 3x_2$

$z = 0 + 4x_1 + 3x_2$

et, après initialisation des variables du programme (l'initialisation à partir du fichier est déjà programmée) :

```
nbVarBase = 3
nbVarHorsBase = 2
```

`tabVarHorsBase` est de dimension 3 et contient aux indices 1 et 2 les numéros 1, 2 (pour les variables x_1 et x_2, qui sont au départ les variables hors-bases)

`tabVarBase` est de dimension 4 et contient aux indices 1, 2 et 3 les numeros 3, 4, 5 (pour les variables x_3, x_4, x_5, qui sont au départ les variables en base)

La matrice `D` est de dimension 4×3 et contient :

```
 0  4  3
24 -2 -3
30 -5 -3
```

```
18 -1 -3
```

Apres avoir pivoté en faisant entrer la variable x_1 et sortir la variable x_4, c'est-à-dire entrer la variable d'indice 1 dans `tabVarHorsBase` et sortir la variable d'indice 2 dans `tabVarBase`, le nouveau dictionnaire doit être :

$x_3 = 12 + 0{,}4\ x_4 - 1{,}8\ x_2$
$x_1 = 6 - 0{,}2\ x_4 - 0{,}6\ x_2$
$x_5 = 12 + 0{,}2\ x_4 - 2{,}4\ x_2$
$z = 24 - 0{,}8\ x_4 + 0{,}6\ x_2$

Le codage doit alors être :
`tabVarHorsBase` contient aux indices 1 et 2 les numéros 4 et 2
`tabVarBase` contient aux indices 1, 2 et 3 les numéros 3, 1 et 5
La matrice `D` contient :

```
24 -0,8  0,6
12  0,4 -1,8
 6 -0,2 -0,6
12  0,2 -2,4
```

Avec ce codage, une étape consiste à chercher dans `tabVarHorsBase` l'indice d'une variable entrante, dans `tabVarBase` l'indice d'une variable sortante, puis à construire le nouveau dictionnaire obtenu en pivotant avec ces deux choix d'indices ; pour cela, il faut calculer pour le nouveau dictionnaire la matrice `D`, les tableaux `tabVarBase` et `tabVarHorsBase`.

REMARQUES

- **ATTENTION : Dans les commentaires du programme, on dit que la variable x1 est la variable de numéro 1, la variable x2 est la variable de numéro 2, ... Quand on parle d'indice dans la programmation, il s'agit toujours des indices dans des tableaux et non des indices de variable.**
- **ATTENTION** : ne pas utiliser l'attribut `incomplet` de la classe `Dictionnaire`
- Les variables de décision (aussi appelées variables principales ou variables initiales) ont des numéros compris entre 1 et `nbVarHorsBase`; les variables d'écart ont des numéros compris entre `nbVarHorsBase + 1` et `nbVarBase + nbVarHorsBase`.
- Pour ce qui concerne un dictionnaire de la première phase recherchant une solution réalisable, une variable de numéro 0 est ajoutée.

Il faut commencer par télécharger le fichier simplexeACompleter.jar ainsi que le fichier pbs.jar contenant les dictionnaires. Si vous travaillez avec Eclipse, il faudra importer ces deux fichiers. Pour chacun des deux fichiers (sur une version d'Eclipse en langue anglaise) :

- créer un nouveau projet (dans File, faire New puis Java Project),
- en sélectionnant **le nom du projet**, cliquer avec le bouton de droite puis choisir "import" ;
- dans la fenêtre obtenue, dans "General", choisir "Archive File" ;
- après avoir fait "Next", choisir le fichier à téléchargé ;
- appuyer sur finish.

Vous êtes alors prêt à travailler. Vous pouvez déjà exécuter le programme ; la méthode `main` est dans le fichier Main.java du paquetage `simplexe`. Vous pouvez choisir n'importe lequel des fichiers de données

fournis dans le répertoire nommé `pbs`.

ATTENTION : durant tout le travail, ne rien modifier en dehors de la classe `Dictionnaire`.

Précisions sur les différents fichiers :

Les dictionnaires codés dans pb1.txt à pb15.txt sont réalisables.
Les problèmes correspondant à pb1.txt jusqu'à pb13.txt sont bornés.
Les problèmes correspondant à pb14.txt et pb15.txt sont non bornés.
Si on effectue une itération à partir du dictionnaire codé par pb12.txt en utilisant la variable entrante de plus grand coefficient, on obtient un dictionnaire dégénéré.
Le dictionnaire codé par pb13.txt est dégénéré. Si on utilise la variable entrante de plus grand coefficient, sans la règle de Bland, il y a cyclage.
Les solutions basiques associées aux dictionnaires codés dans pb16.txt jusqu'à pb20.txt sont non réalisables mais les problèmes correspndants admettent une solution réalisable : la première phase de la méthode du simplexe permet d'avoir un dictionnaire réalisable pour le problème initial.
Les problèmes correspondant à pb21.txt et pb22.txt ne sont pas réalisables.

Vous pouvez trouver une documentation sur le code du projet écrit en Java.

Travail à effectuer

Il faut implémenter (et tester au fur et à mesure) une partie des méthodes de la classe `Dictionnaire`.

1. Implémenter les méthodes dans l'ordre suivant:

- `estRealisable`
- `chercherPremierIndiceVariableEntrante`
- `chercherIndiceVariableSortante`
- `pivoter`

Vous pouvez déjà essayer votre programme avec tous les fichiers de données donnant un dictionnaire initial réalisable avec le choix de la variable entrante selon le critère de la variable de plus petit indice dans le tableau `tabVarHorsBase`. Nous vous conseillons d'essayer au moins pb1.txt et pb14.txt (qui n'est pas borné).

2. Implémenter :

- `chercherIndiceVariableEntranteGrandCoeff` : vous pouvez tester sur pb3.txt ; vous devriez aussi constater qu'il y a cyclage avec cette méthode et le fichier pb13.txt
- `chercherIndiceVariablEntranteAvantageuse` : vous pouvez tester sur pb4.txt.

afin de pouvoir appliquer la règle de Bland :

- `chercherIndiceVariableEntrantePlusPetitNumero`,
- `chercherIndiceVariableSortantePlusPetitNumero` (attention à bien prendre **parmi les variables sortantes** celle de plus petit numéro).

Vous pouvez tester la règle de Bland sur pb13.txt pour constater qu'il n'y a plus de cyclage avec le choix de la variable entrante de plus grand coefficient en sélectionnant "Appliquer la regle de Bland"..