Travail en binôme, langage : Python

1 Prise en main de `lp_solve`

Dans tout le TP, nous allons utiliser le programme `lp_solve`, qui est un solveur de programmation linéaire. La documentation se trouve « ICI »

Voilà un exemple de programme linéaire sous le format de lp_solve :

```
max: 2 x1 + x2;
x1 + 2 x2 <= 20;
3.2 x1 + x2 <= 15;
```

Pour faire tourner le solveur, il suffit de taper la commande :

```
$ lp_solve fichier.lp
```

où le fichier `fichier.lp` contient l'exemple ci-dessus. Le résultat est affiché comme ceci :

```
Value of objective function: 12.77777778

Actual values of the variables:
x1                            1.85185
x2                            9.07407
```

Pour restreindre les solutions à des nombres entiers, on peut ajouter la ligne suivante :

```
int x1, x2;
```

► **Exercice 1** ◄ Vérifiez que tout fonctionne correctement dans l'exemple ci-dessus. Quelles sont les valeurs optimales de `x1` et `x2` si on se restreint à des nombres entiers ?

► **Exercice 2** ◄ lp_solve impose automatiquement des contraintes de non-négativité pour toutes les variables. Essayez de changer l'objectif de l'exemple à `max: -2x1+x2;` et notez que x1 ne descend pas au-dessous de 0. Vous pouvez enlever les contraintes de non-négativité en ajoutant la ligne `free x1, x2;`. Essayez avec l'objectif modifié.

► **Exercice 3** ◄ Vérifiez que vous trouvez bien les mêmes solutions à l'exercice 1 de la feuille de TD 1 avec le solveur que celles que l'on a déterminées graphiquement.

2 Un problème générique

On considère un problème plus général où l'on dispose de m types de ressources $R_1, \ldots, R_m$ pour fabriquer n types de produits $P_1, \ldots, P_n$. Chaque produit j rapporte un bénéfice de c_j pour chaque unité vendue.

Les différentes statistiques associées aux produits sont données dans un fichier texte formaté de la façon suivante :

```
3 2
250 400 490
P1 10 20 10 150
P2 10 10 25 100
```

- La première ligne indique le nombre de ressources m et de produits n.
- La deuxième ligne indique les limites sur les m ressources (ex: au plus 250 unités de la première ressource).
- Les n lignes qui suivent décrivent les produits. La première colonne est le nom du produit (ex: `P2`). Ensuite, suivent m colonnes avec les besoins en ressources (ex: `10 10 25`). La dernière colonne est le bénéfice réalisé par unité de ce produit qui est vendue (ex: 100).

On peut fabriquer des fractions d'unité de produits, et on considère que tout sera vendu.

Dézipper le fichier `data.zip` que vous trouvez sur elearning. Vous y trouvez quelques exemples.

► **Exercice 4** ◄ Écrivez un programme `generic.py` qui prend en entrée un tel fichier et écrit sur un autre fichier de sortie le programme linéaire associé. Les deux noms de fichier seront fournis sur la ligne de commande :

```
$ python3 generic.py exemple1.txt output.lp
```

Avec l'exemple ci-dessus (le fichier `exemple1.txt`), le contenu du fichier `output.lp` devrait ressembler au suivant :

```
max: 150P1+100P2;
10P1+10P2 <= 250;
20P1+10P2 <= 400;
10P1+25P2 <= 490;
```

Ajoutez à votre programme la possibilité de donner l'argument `-int` sur la ligne de commande pour rajouter des contraintes d'intégralité sur toutes les variables.

► **Exercice 5** ◄ Modifiez le programme `generic.py` pour qu'il lance également le solveur `lp_solve` avec le programme linéaire comme entrée, récupère la solution et affiche l'optimum ainsi que les variables non-null et leur valeurs.

Exemple avec le fichier `exemple1.txt` :

```
$ python3 generic.py exemple1.txt output.lp
opt = 3250
P1 = 15
P2 = 10
```

► **Exercice 6** ◄ Essayez votre programme avec le fichier `data.txt`.

(a) Quel est le bénéfice optimal ?

(b) Combien de produits différents faut-il fabriquer ?

(c) Combien de temps a mis le solveur pour calculer la solution optimale ?

► **Exercice 7** ◄ Rajoutez les contraintes d'intégralité. Répondez aux questions de l'exercice précédent. Expliquez le résultat.

► **Exercice 8** ◄ Essayez maintenant avec le fichier (beaucoup) plus volumineux `bigdata.txt`, sans et avec les contraintes d'intégralité. Expliquez le résultat.

3 Un problème de découpe

► **Exercice 9** ◄ Reprenez l'exercice 4 de la feuille de TD 2 sur la découpe de barres d'aluminium et trouvez la solution optimale avec le solveur. Comment faut-il découper les barres de 3m pour minimiser le nombre utilisé ?

► **Exercice 10** ◄ Écrivez une fonction qui prend en argument la longueur des barres de base (en cm) ainsi qu'une liste de longueurs souhaitées. La fonction génère tous les différents découpages maximaux possibles.

Par exemple, avec les entrées 300 et `[120, 100, 50]` comme dans l'exercice précédent, la fonction utilise 0, 1 ou 2 éléments de longueur 120 et continue de façon récursive avec les éléments de longueurs 100 et ensuite 50. **Pour que les découpages soient maximaux, il faut toujours placer la plus petite longueur à la fin et en prendre le maximum possible.**

Pour cet exemple, les 7 différents découpages maximaux possibles seront alors (chaque liste donne le nombre de chacune des longueurs 120, 100 et 50 qu'on obtient par ce découpage) :

```
[0, 0, 6]
[0, 1, 4]
[0, 2, 2]
[0, 3, 0]
[1, 0, 3]
[1, 1, 1]
[2, 0, 1]
```

Essayez avec les entrées 500 et `[200, 120, 100, 50]`. Combien de différents découpages obtenez-vous ?

► **Exercice 11** ◄ En utilisant la fonction de l'exercice précédent, écrivez un programme qui prend en entrées la longueur des barres de base, une liste des longueurs souhaitées et une liste avec le nombre d'éléments de chaque longueur souhaitée. Le programme génère le programme linéaire correspondant et lance le solveur pour trouver l'optimum et la façon dont il faut découper les barres.

Exemple avec réponse : Essayez avec des barres de base de 500 cm qu'on veut découper en 60 éléments de longueurs 200 cm, 100 éléments de 120 cm, 150 éléments de 100c m et 350 éléments de 50 cm.

Réponse : une solution optimale utilise 114 barres de longueur 500 cm.