Les algorithmes gloutons - Correction des exercices

▷ **Exercice 1** - Exercice débranché - ⋆

On suppose dans cet exercice, qu'on prend le système de pièces de monnaie européen (en centimes)

1. On suppose qu'on dispose d'autant d'unités que l'on veut. Quel est le rendu de monnaie proposé à l'aide d'un algorithme glouton de monnaie

   (a) si on doit rendre 2€47 ?

   solution :

   Le rendu proposé à l'aide d'un l'algorithme glouton de rendu de monnaie est 200,20,20,5,2 soit 5 pièces.

   (b) si on doit rendre 6€36 ?

   solution :

   Le rendu proposé à l'aide d'un l'algorithme glouton de rendu de monnaie est 200,200,200,20,10,5,1 soit 7 pièces.

   (c) si on doit rendre 3 €68 ?

   solution :

   Le rendu proposé à l'aide d'un l'algorithme glouton de rendu de monnaie est 200,100,50,10,5,2,1 soit 7 pièces.

2. On suppose à présent qu'on a un nombre limité de pièces réparti de la façon suivante :

Pièces en centimes	200	100	50	20	10	5	2	1
Quantité	3	5	9	8	6	5	4	1

   Quel est le rendu de monnaie proposé à l'aide d'un algorithme glouton de monnaie si on doit rendre successivement 2€47, 6€36 et 3€68 ?

   solution :

   Voici l'évolution du stock de la répartition des pièces de monnaie :

Pièces en centimes	200	100	50	20	10	5	2	1	
Quantité initiale	3	5	9	8	6	5	4	1	Rendu de 2€47 : 200,20,20,5,2
Quantité après avoir rendu 2€47	2	5	9	6	6	4	3	1	Rendu de 6€36 : 200,200,100,100,20,10,5,1
Quantité après avoir rendu 6€36	0	3	9	5	5	3	3	0	Rendu de 3€68 : impossible

▷ **Exercice 2** - Exercice débranché - ⋆

On considère un bureau de poste américain qui ne dispose pas de machine à affranchir mais des timbres de valeurs faciales :

1 cent, 10 cents, 21 cents, 34 cents, 70 cents et 100 cents

Un client veut affranchir un courrier pour 1$ et 40 cents.

1. Quel serait le rendu proposé à l'aide d'un l'algorithme glouton de rendu de monnaie ?

   solution :

   Le rendu proposé à l'aide d'un l'algorithme glouton de rendu de monnaie est 100,34,1,1,1,1,1,1 soit 8 pièces.

2. A-t-on obtenu une solution optimale ?

solution :

On n'a pas obtenu de solution optimale. La solution optimale est : 70,70 soit 2 pièces.

▷ **Exercice 3** - Exercice débranché - ⋆

« Ils sont fous ces Bretons ! »

Obélix dans Astérix chez les Bretons, Goscinny et Uderzo
(Source)

Nous sommes en 1966 après Jésus-Christ. Toutes les monnaies européennes sont décimalisées… Toutes ? Non ! Un royaume peuplé d'irréductibles Bretons résiste encore et toujours à la décimalisation de leur chère livre sterling. En effet, jusqu'en 1971, le Royaume-Uni ne possédait pas un système de monnaies décimalisées [1].
La livre(£) valait 20 shillings(s) et un shilling 12 pence(d). Il y avait d'autres sous-unités du penny et du shilling : voici le récapitulatif des "petits" billets et des pièces qui étaient utilisés en 1966.

	billets		Pièces							
Nom	a Fiver	a Quid	a Half-Crown	a Florin	a Bob	a Tanner	three Pence	a Copper	a half-penny	a farthing
Valeur	5 £	1 £	2 shillings and 6 pence	two Shillings	one Shilling	six Pence	$\frac{1}{4}$ shilling	one Penny	$\frac{1}{2}$ de penny	$\frac{1}{4}$ de penny
Valeur en pence	1200	240	30	24	12	6	3	1	0,5	0,25

1. Compléter les dernières lignes du tableau.

solution :

cf ci-dessus pour tableau complété

2. On se propose de tester un algorithme glouton de rendu de monnaie en prenant pour répartition de monnaie toutes les unités en pence décrites ci-dessus. On considère que l'on dispose d'autant d'unités que l'on veut.
   (a) On doit rendre 2 £ et 34 pences. Quel serait le rendu proposé par l'algorithme ?

   solution :

   Le rendu proposé à l'aide d'un l'algorithme glouton de rendu de monnaie est 1£, 1 £,a half crown, three pence et a copper soit 5 pièces.

   (b) On doit rendre 3 £ et 4 shillings. Quel serait le rendu proposé par l'algorithme ?

   solution :

   Le rendu proposé à l'aide d'un l'algorithme glouton de rendu de monnaie est 1£, 1 £,1 £,a half crown, a bob et a Tanner soit 6 pièces.

   (c) A-t-on obtenu des solutions optimales ?

   solution :

   On a obtenu une solution optimale pour le premier rendu mais pas pour le second. En effet, la solution optimale est 1£, 1 £,1 £,a Florin, a Florin soit 5 pièces.

1. Décimalisée signifie ici que la livre sterling est divisée en cent sous-unités : cent pence (au singulier : un penny). Un penny vaut un centième de livre.

▷ **Exercice 4** - Exercice branché - ⋆⋆

Une route comporte n stations-service, numérotées dans l'ordre du parcours, de 0 à $n-1$. Les distances entre chaque stations-service sont données par un tableau de données d $[d_0, d_1, ...; d_n]$ telle que :
la première est à une distance d_0 du départ, la deuxième est à une distance d_1 de la première, la troisième à une distance d_2 de la deuxième, etc. La fin de la route est à une distance d_n de la n-ième et dernière station-service.
Un automobiliste prend le départ de la route avec une voiture dont le réservoir d'essence est plein. Sa voiture est capable de parcourir une distance r avec un plein.

1. Donner une condition nécessaire et suffisante pour que l'automobiliste puisse effectuer le parcours. On la supposera réalisée par la suite.

solution :

Une condition nécessaire et suffisante pour que l'automobiliste puisse effectuer le parcours est que la distance entre le départ et la station 0 ou entre deux stations consécutives ou entre la station n et l'arrivée soit inférieure ou égal à n.

2. En considérant 17 stations-service avec les distances d = [23, 40, 12, 44, 21, 9, 67, 32, 51, 30, 11, 55, 24, 64, 32, 57, 12, 80] et r = 100.
   L'automobiliste désire faire le plein le moins souvent possible. Écrire une fonction en Python nommée rapide de paramètre d et r qui détermine à quelles stations-service il doit s'arrêter.

solution :

Un script possible est le suivant :

fonction rapide

```
>>> #ou faire le plein ?
>>> #d : distances entre les stations-service
>>> #r : distance que l'on peut parcourir au maximum
>>> d = [23, 40, 12, 44, 21, 9, 67, 32, 51, 30, 11, 55, 24, 64, 32, 57, 12, 80]
>>> r = 100

>>> def rapide(d,r) :
...     i=0
...     dmax=r
...     while i<len(d):
...         dmax=dmax-d[i]
...         if dmax<0:
...             i=i-1
...             print("il faut faire le plein a la station-service no ",i)
...             dmax=r
...         i=i+1
...
>>> rapide(d,r)
il faut faire le plein a la station-service no  2
il faut faire le plein a la station-service no  5
il faut faire le plein a la station-service no  7
il faut faire le plein a la station-service no  10
il faut faire le plein a la station-service no  12
il faut faire le plein a la station-service no  14
il faut faire le plein a la station-service no  16
```

▷ **Exercice 5** - Exercice débranché - ⋆

Le problème dit "du sac à dos" possède une importance majeure en algorithmique. Depuis plus d'un siècle, il fait l'objet de recherches actives et sa résolution trouve de nombreuses applications pratiques dans le domaine de la finance par exemple. Dans les années 1970, il fut a l'origine des premiers algorithmes de cryptographie à clé publique/privée. Nous tenterons de le résoudre par différentes "méthodes" gloutonnes.

Enoncé du problème

"Imaginons un voleur entrant de nuit dans un magasin ; il est équipé d'un sac pour réaliser son délit et y placer les objets volés. Malheureusement pour lui, son sac est usé et au-delà d'un certain poids, celui-ci risque de craquer. Dès lors, pour maximiser son profit, le voleur cherche à placer dans son sac les objets de plus grande valeur possible, tout en veillant à ne pas dépasser le poids maximal autorisé."

1. S'agit-il d'un problème d'optimisation ? Justifier la réponse.

solution :

Il s'agit bien d'un problème **d'optimisation** puisque le voleur chercher à maximiser le montant de son butin.

Notations

On parle souvent du **KP** pour évoquer le problème du sac à dos **(KP = "Knapsack Problem")**.
On parle parfois du **0/1-KP** car les objets sont volés en entier (1) ou laissés dans le magasin (0) : le voleur ne peut pas prendre un bout d'objet ! Supposons que quatre objets(A, B, C, et D) présents dans le magasin, et que le voleur vole les quatre : la solution du **0/1-KP** peut-être notée {1, 1, 1, 1}

2. En utilisant les notations précédentes, quelle est la solution si le voleur vole les objets A et C ?

solution :

Si A et C sont volés (1) alors B et D restent en magasin (0) : la solution est {1,0,1,0}

3. Soient trois objets de 4kg, 2kg et 1kg ainsi qu'un sac supportant jusqu'à 6kg. La solution {1, 0, 1} respecte-t-elle la condition sur la masse totale autorisée ?

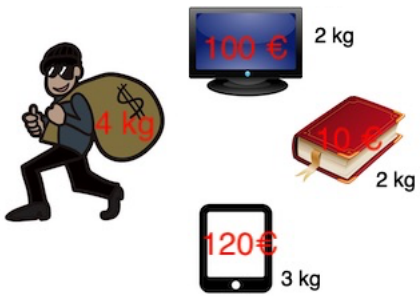
solution :

Si on désigne les objets par leur masse, on noterait la liste d'objets $\{m_1, m_2, m_3\}$. La solution {1, 0, 1} au KP correspond à une masse de $4 \times 1 + 2 \times 0 + 1 \times 1$ soit 5 kg. La solution {1, 0, 1} respecte la condition sur la masse totale autorisée.

▷ **Exercice 6** - Exercice débranché - ⋆

Prenons l'exemple d'un sac supportant 4 kg au maximum et un magasin avec trois objets A, B, C(Voir Figure 1). On supposera que le sac est suffisamment grand pour accueillir tous les objets. Pour la suite nous représenterons, les trois objets sous la forme d'un couple **(valeur en €, poids en kg) :**

Objet A : (100, 2) Objet B : (10,2) Objet C : (120,3)



1. En respectant le poids maximal, déterminer ("sur papier") le nombre de façons de remplir le sac (on pourra présenter les résultats dans un tableau).

solution :

Possibilités	n°1	n°2	n°3	n°4
Objets	A	B	C	A+B
Valeur (€)	100	10	120	100+10=110
poids (kg)	2	2	3	2+2=4

2. Indiquer quel est le choix le plus profitable au voleur.

solution :

Parmi les quatre possibilités, la n°3 est la plus profitable au voleur (car 120€ > 110€ > 100€ > 10€).

▷ **Exercice 7** Exercice débranché - ⋆ mais accès à la documentation officielle Python3
Soit la liste d'objets suivante où chaque objet est caractérisé par (un poids, une valeur) et un sac tel que $p_{max} = 2,7$:
L=[(0.2,300),(0.2,500),(0.1,250),(0.3,500),(1.3,2300),(0.8,2000),(1.3,2500)]

1. Ecrire une liste triée par valeurs (décroissantes) puis une liste triée par rapports décroissants $\frac{\text{valeur}}{\text{poids}}$

solution :

Attention le poids est donné avant la valeur!
valeurs (décroissantes) : [(1.3,2500),(1.3,2300),(0.8,2000),(0.2,500),(0.3,500),(0.2,300),(0.1,250)]
$\frac{\text{valeur}}{\text{poids}}$: [(0.8,2000),(0.2,500),(0.1,250),(1.3,2500),(1.3,2300),(0.3,500),(0.2,300)]
En cas d'égalité, on peut placer les objets au hasard

.

2. Appliquer l'algorithme glouton sur ces deux listes et écrire une liste réponse uniquement constituée de 0 ou de 1. Commenter.

solution :

Algorithme Sac appliqué à la liste 1 :
→ [(1.3,2500),(1.3,2300),(0.8,2000),(0.2,500),(0.3,500),(0.2,300),(0.1,250)]
→ Reponse=[1,1,0,0,0,0,1] par rapport à la liste triée
→ Reponse=[0,0,1,0,1,0,1] par rapport à la liste initiale
→ Poids du Sac n°1 : 2,7 & Valeur du Sac n°1 : 5050
→ Le Sac n°1 est maximal ce qui ne signifie nécessairement que 5050 soit la solution optimale.

Algorithme Sac appliqué à la liste 2 :
→ [(0.8,2000),(0.2,500),(0.1,250),(1.3,2500),(1.3,2300),(0.3,500),(0.2,300)]
→ Reponse=[1,1,1,1,0,1,0] par rapport à la liste triée
→ Reponse=[0,1,1,1,0,1,1] par rapport à la liste initiale
→ Poids du Sac n°2 : 2,7 & Valeur du Sac n°2 : 5750
→ Le Sac n°2 est également maximal mais de valeur supérieur au Sac n°1 (ce qui ne prouve toujours rien sur son optimalité!). On peut néanmoins conclure que l'algorithme glouton donne une meilleure réponse avec le second critère.

3. En Python, que renvoie les instructions `L.sort(key = lambda a : a[1])` et `L.sort(key = lambda a : a[0], reverse=True)`?

solution :

https ://docs.python.org/3/howto/sorting.html :
"list.sort() and sorted() have a key parameter to specify a function to be called on each list element prior to making comparisons." [...]accept a reverse parameter with a boolean value. This is used to flag descending sorts.

Dès lors, on peut appeler une fonction **lambda** (nom donné à une fonction nommée "à la volée") pour trier des objets à un certain indice dans l'élément courant (si cet élément est une liste, un tuple,...) En conséquence, le tri sur **a[1]** signifie que l'on s'intéresse au deuxième élément du tuple, c'est à dire à la valeur dans cet exercice. Par ailleurs, **reverse=True** indique qu'il s'agit d'un tri "décroissant".
→ **L.sort(key = lambda a : a[1])**→ Tri par valeur croissante : [(0.1, 250), (0.2, 300), (0.2, 500), (0.3, 500), (0.8, 2000), (1.3, 2300), (1.3, 2500)]
→ **L.sort(key = lambda a : a[0], reverse=True)**→ Tri par poids décroissant :[(1.3, 2300), (1.3, 2500), (0.8, 2000), (0.3, 500), (0.2, 300), (0.2, 500), (0.1, 250)]
Dans le second tri, on pourra remarquer que Python place l'objet de valeur 2500 après celui de valeur 2300 .

▷ **Exercice 8** - exercice branché - ⋆⋆

Il s'agit d'exécuter un algorithme "naïf" qui consiste à tester toutes les choix d'objets acceptables pour finir par choisir la meilleure.

1. Télécharger sur l'ENT le fichier **ex8.py**.
2. Le code télechargé comporte plusieurs fonctions. La fonction **possibilites** de paramètre un entier n permet d'afficher le nombre de choix possibles de n objets (dans le sac du voleur).
   Tester la fonction **possibilites** pour $n \in \{3,4,5,6,7\}$. Conjecturer le nombre de façons de remplir le sac avec n objets.

   solution :

   $2^3 = 8, 2^4 = 16, ...2^7 = 128$ donc avec n objets, on peut conjecturer que le nombre de façons de remplir le sac soit 2^n.

3. La fonction **KPnaif** de paramètre une liste de couples **(valeur en €, poids en kg)** et un entier n retourne un triplet constitué de la solution optimale obtenue par l'algorithme naïf pour le sac, la valeur du sac et le poids du sac. Tester l'algorithme et noter les durées d'exécution sur les 5 premiers jeux de données du fichier datas.txt, c'est à dire pour $n \in \{4,7,8,10,15\}$.

solution :

n=4 [(7,13),(4,12),(3,8),(3,10)] 30
n=7 [(442, 41),(525, 50),(511, 49),(593, 59),(546, 55),(564, 57),(617, 60)] 170
n=8 [(15, 2),(100, 20),(90, 20),(60, 30),(40, 40),(15, 30),(10, 60),(1, 1)] 102
n=10 [(92, 23),(57, 31),(49, 29),(68, 44),(60, 53),(43, 38),(67, 63),(84, 85),(87, 89),(72, 82)] 165
n=15 [(214, 113),(229, 118),(192, 98),(150, 80),(173, 90),(139, 73),(240, 120),(156, 82),(135, 70),(163, 87),(221, 115),
(201, 106),(149, 77),(184, 94),(210, 110)] 750

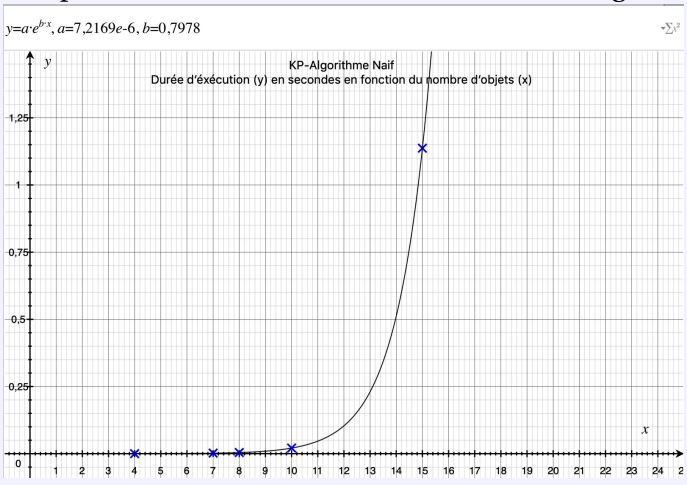
KPnaif : Réponses et Durée d'exécution (en s)					
Objets	n=4	n=7	n=8	n=10	n=15
Durée	1.291e-4	2.129e-3	4.348e-3	2.100e-2	1.137
Réponse	([1,1,0,0], 11, 25)	([0, 1, 0, 1, 0, 0, 1], 1735, 169)	([1, 1, 1, 1, 0, 1, 0, 0], 280, 102)	([1, 1, 1, 1, 0, 1, 0, 0, 0, 0], 309, 165)	([0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0] , 1458, 749)

* Calculs effectués sur 1000 itérations pour chaque fonction ; MacBook Pro IntelCore i5 2.4 GHz double coeur

4. A l'aide du logiciel de votre choix, représenter $t = f(n)$ où t représente la durée d'exécution du programme. Donner une équation d'une courbe de tendance de la forme $t = a \times e^{b \times n}$ (donner les valeurs des paramètres a, b).

solution :

Evolution exponentielle de la durée d'exécution de l'algorithme naïf



5. Quel est le temps d'exécution prédit par ce modèle lorsque n=50 ? Commenter le résultat obtenu.

solution :

On peut modéliser la durée d'exécution t en fonction de n par $t(n) = 7,217 \times 10^{-6} \times e^{0,7978 \times n}$ où n est le nombre d'objets à voler.
Ce modèle prédit, $t(50) = 7,217 \times 10^{-6} \times e^{0,7978 \times 50} \approx 1521821054913$s
Il s'agit d'une durée proche de 50 000 ans!! Il n'est donc pas réaliste d'envisager une recherche de solution optimale par "force brute" même pour des ensembles relativement petits.