Projet de validation du bloc 2

Proposition de cours

Thème choisi : les algorithmes gloutons

Aude Duhem et Patrice Nicolas

juillet 2019

Table des matières

Chapitre 1

Support pédagogique

I Prérequis

- Types de base, listes, dictionnaires ;
- instruction IF, boucles FOR et WHILE ;
- spécification, définition et appel de fonctions ;
- importation de modules tels que random et time.
- connaissance des notions de complexité, terminaison et correction des algorithmes
- algorithmes de tri (dont complexité) (car les algorithmes gloutons supposent d'avoir des valeurs triées)

II Notions abordées dans le cadre du programme NSI Première

Contenus	Capacités attendues	Commentaires
Algorithmique		
Algorithmes gloutons	Résoudre un problème grâce à un algorithme glouton	Exemples : problèmes du sac à dos ou du rendu de monnaie. Les algorithmes gloutons constituent une méthode algorithmique parmi d'autres qui seront vues en terminale.

III Situation dans l'année scolaire

Cette séquence vient en milieu d'année, au minimum après les prérequis .
Durée : 8 heures de 55 min.

IV Compétences attendues en fin de séance

- connaissance du principe des algorithmes gloutons et de ses faiblesses
- sensibilisation à la notion d'optimisation
- connaissance des deux algorithmes de référence au programme :
- Approfondissement des types tableaux de données et dictionnaires
- Approfondissement des notions de complexité, terminaison, correction d'un algorithme

V Modalité d'évaluation

- Évaluation des corrections orales des élèves en séances
- Interrogation sommative en séance 6
- Évaluation des compte-rendu des TP1 et TP2

Chapitre 2

Progression envisagée

I séance 1 - 1 h - séance débranchée

1 Activités d'introduction du principe des algorithmes gloutons (15-20 min) - cf. annexe A

introduction faite à partir de deux activités : un exemple pour introduire optimisation/méthode gloutonne et un problème du rendu de monnaie.
Travail proposé en groupe pour amener les élèves à découvrir les notions de problèmes d'optimisation, d' algorithme glouton et la "solution" du problème de rendu de monnaie.

2 synthèse sur le cahier de cours (20 min)- - cf. annexe B

Cours jusqu'au III A (inclus) Définition concept de problème d'optimisation et d'algorithme glouton
Principe de l'algorithme du rendu de monnaie
énoncé en pseudo-code +

3 Exercice d'application (15 min) - cf. annexe C

exercice 1 - exercice où l'algorithme glouton trouve la solution optimale

Devoirs pour la séance suivante : finir l' exercice

II séance 2 - 2 h - TP

1 Retour sur la séance précédente (10 min)

Réactivation des connaissances acquises lors de la séance précédente.
Projection et explications à l'oral de l'exercice 1 (annexe C) par un élève

2 Première partie de la séance (1 h 30 min)

TP de rendu de monnaie à programmer en Python
Difficultés croissantes
Exercice 4 (de la feuille d'exercices) à proposer pour les élèves rapides

3 Deuxième partie de la séance (0 h 20 min)

Exercices 2 et 3 - Exercices de rendu de monnaie où l'algorithme glouton ne trouve pas la solution optimale

Devoirs pour la séance suivante :
- finir le compte-rendu de TP
- finir les exercices 2 et 3

III séance 3 - 1h - séance débranchée

1 Retour sur la séance précédente (10 min)

Réactivation des connaissances acquises lors de la séance précédente.
Projection et explications à l'oral des exercices 2 et 3 (annexe C) par un élève

2 synthèse sur le cahier de cours (35 min)- - cf. annexe B

Paragraphes III B et III C :Complexité, terminaison et correction de l'algorithme du rendu de monnaie

3 Découverte du principe de l'algorithme du sac à dos (10 min)

Travail de groupe : exercice 5 - exercice d'introduction de l'algorithme du sac a dos

Devoirs pour la séance suivante : finir l'activité

IV séance 4 - 1h - séance débranchée

1 Retour sur la séance précédente (5 min)

Réactivation des connaissances acquises lors de la séance précédente.
Projection et explications à l'oral de l'activité par chaque groupe

2 synthèse sur le cahier de cours (30 min)- - cf. annexe B

Algorithme du sac à dos - Paragraphe IV définition de façon "littérale", sensibilisation au formalisme si le groupe s'y prête et si le groupe a vu les notations en cours de mathématiques.
critères
énoncé en pseudo-code+ complexité et terminaison

3 Exercice d'application (15 min) - cf. annexe C

exercice 6 - exercice à la main pour vérifier que l'algorithme du sac à dos est compris
Exercice 7 - exercice moins guidé de l'algorithme du sac et permettant de découvrir des méthodes à employer en Python pour effectuer des tris selon des critères.

Devoirs pour la séance suivante : Retravailler le cours sur le sac à dos et finir l'exercice 7

V séance 5 - 2 h - TP

1 Retour sur la séance précédente (10 min)

Projection et explications à l'oral de l'exercice 7 (annexe C) par un élève

2 Travail de groupe - 15 min

Élaboration d'une fiche mémorisation/résumé de cours, en groupe (forme libre : carte mentale, résumé linéaire, ...)
Le professeur circule pour vérifier que les compétences attendues sont bien cernées.
Les productions ainsi produites sont mises à disposition du groupe classe sur l'ENT via un espace partagé.

3 Fin de la séance (1 h 30 min)

TP de sac à dos
Exercice 8 (de la feuille d'exercices) à proposer pour les élèves rapides

Devoirs pour la séance suivante : se préparer pour l'évaluation et demander à chacun une petite recherche ciblée (2 élèves par attendu) afin de pouvoir remplir, ensemble, la frise chronologique des algorithmes gloutons

VI séance 6 - 1h

1 Évaluation (30 min) (cf. annexe E)

2 Correction à l'oral avec les élèves de l'évaluation (10 min) (cf. annexe E)

3 Point bilan sur les nouveaux termes rencontrés dans cette séance (10 min) (cf. annexe F)

Au fur et à mesure de l'année un lexique est constitué avec les élèves. Projection et explications de l'annexe F qui viendra amender ce lexique.

4 Frise chronologique du chapitre à compléter (10 min) (cf. annexe H)

En fonction des apports des élèves, esquisser une frise chronologique du chapitre

Devoirs pour la séance suivante : rendre le compte-rendu du TP2

Chapitre 3

Répartition des tâches

Après avoir respectivement fait des recherches individuellement, nous avons en commun réfléchi à une progression envisageable et nous sommes donnés un premier axe d'approche :

— Patrice est le responsable principal de l'algorithme du sac à dos

— Aude est la responsable principale de l'algorithme du rendu de monnaie

Voici la répartition du travail de chacun :

	Aude Duhem	Patrice Nicolas
Support pédagogique	Recherche et conception du support	
Progression envisagée	Réflexion conjointe Finalisation saisie et mise en page	Réflexion conjointe
Actvivité d'introduction	Recherche, conception et mise en page de l'activité problème de monnaie	Recherche, conception de l'activité maître nageur
Support de cours	Recherche et conception conjointe	Recherche et conception conjointe
Feuille d'exercices	Recherche et conception des exercices 1 à 4	Recherche et conception des exercices 5 à 8
TP1 rendu de monnaie	recherche, concept, tests, corrigé et compléments pédagogiques	
TP sac à dos		recherche, concept, tests, corrigé et compléments pédagogiques
Evaluation	Recherche et conception conjointe	Recherche et conception conjointe
Notions mises au lexique	Recherche et conception conjointe	Recherche et conception conjointe
Frise chronologique	Recherche du site ou de l'application adéquate, recherches et élaboration de la frise	

Annexe A

Activités d'introduction

Ce support propose deux activités d'introduction aux principes de problème d'optimisation, d'algorithme glouton et du problème de rendu de monnaie.

I Déroulé envisagé

1 introduction orale

Dans la vie courante, il y a beaucoup de situations où on cherche la solution la plus optimale face à un problème donné tel que :

- Rendre la monnaie avec le moins de pièces : problème du caissier ;
- ranger son sac de façon optimisée : problème du sac à dos ;
- ranger des objets avec un nombre minimum de boîtes : bin packing ;
- trouver le plus court chemin : Algorithme de Djikstra

Nous allons pour introduire ce chapitre, nous familiariser à la notion de problème d'optimisation et au premier problème cité.

2 Travail de recherche en groupe

Le professeur distribue l'énoncé des deux activités à des élèves répartis en groupe de 3
Les laisser chercher vingt minutes par groupe de 3/4 élèves et faire une synthèse tous ensemble.

Intérêt : fixer sur des exemples concrets des principes de problème d'optimisation et d'algorithme glouton et du problème du rendu de monnaie

---

II Compléments pédagogiques de l'activité 1 :

Activité permettant de sensibiliser les élèves aux notions de problèmes d'optimisation en abordant deux approches : l'approche naïve et l'approche gloutonne. Les élèves n'obtiendront pas la solution optimale car on a volontairement pas donné suffisamment d'informations.

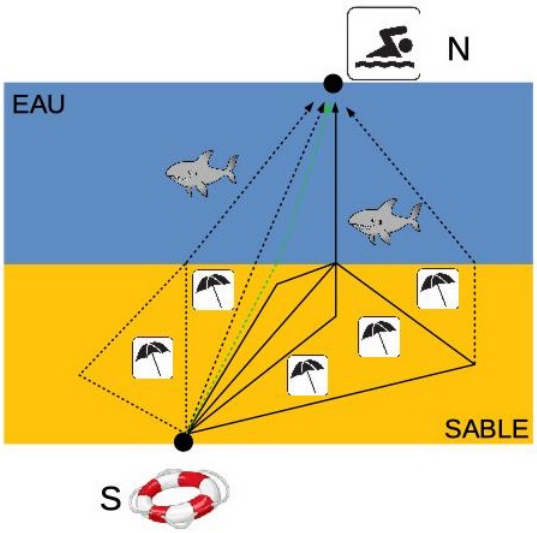
III Compléments pédagogiques de l'activité 2 :

Il est possible, que pour le problème du rendu de monnaie, que deux méthodes "sortent" : l'algorithme glouton naïf et celui où on fait des divisions au lieu des soustractions.
Les faire conclure qu'on arrivera de toute façon au même rendu de monnaie.
Et les faire trouver par tâtonnement une définition avec leurs mots des principes de problème d'optimisation et d'algorithme glouton.

Activités d'introduction au chapitre

Activité 1 :

Sur la plage, un surveillant de baignade (S) aperçoit un nageur (N) en difficulté. Le surveillant dispose d'un ordinateur qui lui "suggère" plusieurs chemins possibles pour atteindre et tenter de sauver le baigneur.
La Figure ci-dessus schématise la situation ; le sauveteur peut utiliser plusieurs chemins : ceux en traits pleins assurent un temps de nage minimum pour rejoindre le nageur. Le chemin en pointillés vert est celui "de moindre temps" : **c'est la solution optimale!**



Analysons cette situation plus en détail :

— Le problème est de sauver une personne.

— Il existe peut-être une solution pour résoudre ce problème...

— Si plusieurs solutions existent, on peut affirmer que la solution **"chemin de moindre temps"** fait partie de cet ensemble de solutions.

Partie A - Approche naïve

Le surveillant se dit :"je vais étudier **un à un tous les chemins** puis, calculer les temps de parcours et je garderai le chemin correspondant au minimum ; **je suis certain alors de trouver le meilleur chemin!!"**

1. Citer un avantage et un inconvénient de la méthode proposée par le surveillant ?
2. Quel critère pourrait-on se donner pour éliminer certains chemins ?
3. Avec votre critère, le chemin optimal serait-il conservé ?

Partie B - Approche "gloutonne"

L'ordinateur propose le critère suivant : "Comme vous avancez moins vite dans l'eau que sur le sable, je vous propose de classer les chemins de façon à rester le moins longtemps possible dans l'eau. **On élimine tous les autres!! C'est la méthode gloutonne, on élimine beaucoup de chemins!!** ; les chemins conservés sont ceux représentés en traits plein sur la figure.

1. Citer un avantage et un inconvénient de la méthode proposée par l'ordinateur ?

Activité 2 :

Supposons que vous êtes un commerçant. Une cliente vous achête pour 134,34 € de marchandise. Elle vous tend un billet de 100 € et un billet de 50 €.
Votre caisse n'est constituée que de pièces et possède autant de pièces que l'on veut.

CC by Pixabay

1. Quel montant devez-vous rendre ?
2. Proposez au moins quatre façons différentes de rendre la monnaie.
3. Quelle est la méthode où vous rendriez le plus de pièces possibles ?
4. Est-ce est selon vous, la méthode employée par un caissier expérimenté ?
   Si non, quelle est sa méthode ? La décrire en langage naturel.
5. Quel est le rendu proposé par votre méthode ?

Pistes de solutions pour le professeur :
Activité 1 :

Partie A - Approche naïve

1. La méthode proposée par le sauveteur assure de trouver la meilleure solution possible mais elle est très couteuse en temps!
2. Supposons que les 9 chemins soient numérotés de gauche à droite. Quelques critères possibles :
   — "Chemin(s) laissant tous les parasols à droite" : chemins possibles {n°1}
   — "Chemin(s) où le temps sur le sable est minimum" : chemins possibles {n°2}
   — "Chemin(s) où le temps dans l'eau est minimum" : chemins possibles {n°5, n°6, n°7}
   — ...
3. Aucun des trois critères précédents ne contient la solution finale optimale. Avec ces critères, la méthode de l'ordinateur est dite sous-optimale!

Partie B - Approche "gloutonne"

1. La méthode gloutonne assure une certaine rapidité d'exécution mais sa réponse dépend du critère de décision choisi et en pratique elle est sous-optimale.

Activité 2 :

1. on doit rendre 100€+50€-134,34€soit 15,66€.
2. première façon : 15 pièces de 1€, 3 pièces de 20 centimes et 3 pièces de deux centimes soit 21 pièces
   deuxième façon : 7 pièces de 2 €, 1 pièce de 1€, 3 pièces de 20 centimes et 3 pièces de deux centimes soit 14 pièces
   troisième façon : 7 pièces de 2 €, 1 pièce de 1€, 1 pièce de 50 centimes, 1 pièce de 10 centimes et 3 pièces de deux centimes soit 13 pièces
   quatrième façon : 7 pièces de 2 €, 1 pièce de 1€, 6 pièces de 10 centimes et 3 pièces de deux centimes soit 17 pièces
3. La méthode avec le plus de pièces possibles est 1566 pièces de 0,01 €
4. Cela n'est pas la méthode du caissier expérimenté, son but est de rendre le moins de pièce possible afin d'aller au plus vite pour passer au client suivant.
   Il est possible que deux méthodes "sortent" : l'algorithme glouton naïf et celui où on fait des divisions au lieu des soustractions.
   (cf cours pour leur écriture).
   leur faire écrire en pseudo code les deux afin d'introduire le cours
5. La méthode avec le moins de pièces possibles décrite ci-dessus est :
   7 pièces de 2 €, 1 pièce de 1€, 1 pièce de 50 centimes, 1 pièce de 10 centimes , 1 pièce de 5 centimes, 1 pièce de 1 centimes soit 12 pièces.

Annexe B

Support de cours

Les algorithmes gloutons

I Notion de problème d'optimisation

Le cadre de notre étude cette année se limite à des problèmes possédant toujours au moins une solution et un nombre fini de solutions. Dans ce contexte, nous nous interesserons ici plus particulièrement aux **problèmes d'optimisation, c'est à dire des problèmes dans lesquels, il faut minimiser ou maximiser une certaine quantité.** Il faut donc trouver la meilleure solution parmi l'ensemble des solutions possibles. Si on n'y parvient pas, nous tenterons d'estimer notre écart à la solution optimale.
Pour atteindre cet objectif, il y a plusieurs méthodes au choix : méthode exhaustive ou brute , méthode diviser pour régner (vue précédemment), programmation dynamique (vue ultérieurement) et méthode gloutonne (présentée ici)

II La méthode gloutonne

Définition

Un algorithme glouton est un algorithme qui fait un choix optimum local à chaque étape, dans le but d'obtenir une solution optimale globale au problème. Il n'y a pas de retour en arrière : à chaque étape de décision dans l'algorithme, le choix effectué est définitif.

But d'un algorithme glouton

Un algorithme glouton sert donc à résoudre des problèmes d'optimisation.

Vocabulaire

un algorithme glouton ne permettra pas toujours d'obtenir la solution optimale : il fait donc partie du groupe des méthodes approchées ou heuristiques gloutonnes.

Exemple 1

Approche "gloutonne" pour le surveillant de baignade et problème de rendu de monnaie (activités)

III Le problème du rendu de monnaie

A Principe

Enoncé classique du problème du rendu de monnaie

«étant donné un système de monnaie, comment rendre une somme donnée de façon optimale, c'est-à-dire avec le nombre minimal de pièces et billets ?»

A l'issue de l'activité d'introduction, nous avons écrit deux versions en pseudo-code d'une fonction **monnaie** implémentant l'algorithme :

- **Pré-conditions :**
  ⋆ somme s (entier positif) à rendre
  ⋆ liste P $[v_1, v_2, ..., v_n]$ des valeurs des pièces disponibles triées par valeur décroissante
- **Post-condition :** résultat = liste Q $[q_1; q_2; ..., q_n]$ du nombre de pièces par valeur à rendre

Version 1

```
1  Fonction( monnaie(P,s))
2  n ← taille de la liste P ;
3  Q ← [0,.... 0];
4  k ← 1;
5  tant que s > 0 et k < n faire
6      tant que s ⩾ v_k faire
7          q_k ← q_k + 1;
8          s ← s − v_k
9      fin
10     k← k+1
11 fin
12 retourner liste Q des pièces à rendre
```

Version 2

```
1  Fonction( monnaie(P,s))
2  n ← taille de la liste P ;
3  Q ← [0,.... 0];
4  k ← 1;
5  tant que s > 0 et k < n faire
6      si s ⩾ v_k alors
7          q_k ← quotient de la division euclidienne
             de s par v_k;
8          s ← reste de la division euclidienne de s
             par v_k;
9      fin
10     k← k+1 ;
11 fin
12 retourner liste Q des pièces à rendre
```

Exemple 2

Rendre la somme de 2,43€ en disposant du système de pièces de monnaie européen, à savoir (en cents) : [200,100,50,20,10,5,2,1]

B Complexité et terminaison de l'algorithme

La version 2 est plus efficace que la version 1, plus naturelle. Elle permet d'en déduire aisément la **complexité** de l'algorithme de rendu de monnaie :
Les pièces du système de monnaie étant supposées triées par ordre décroissant de valeur et au nombre de n, alors la boucle Tant que correspond à une **complexité** de l'ordre de n. Par contre si la liste des pièces n'est pas triée on passe à une **complexité** de l'ordre $n \log n$.
La **terminaison** de l'algorithme est garantie : en effet, les variants de boucle sont s et k
s est un entier positif ou nul qui décroit strictement et k est un entier positif qui augmente de 1 à chaque étape jusqu'à devenir égal à n qui est un nombre fini. La boucle Tant que s'arrête donc.

C Correction de l'algorithme

Exemple 3

Rendre la somme de 6⋆ en disposant du système de pièces de monnaie (en ⋆) : {4,3,1}

Pistes pour l'exemple 3

Le faire à la main et constater que la méthode gloutonne rendra 1 pièce de 4 ⋆ et 2 pièces de 1 ⋆ alors que la solution optimale rendra 2 pièces de 3 ⋆

L'algorithme glouton du rendu de monnaie ne fournit donc pas toujours la solution optimale :

Vocabulaire

On appelle canonique un système de pièces pour lequel l'algorithme glouton est optimal.

Le système de monnaie européen est canonique alors que le système anglais avant 1971 ne l'était pas.
Pour montrer qu'un système de monnaie n'est pas canonique, il suffit de donner un contre-exemple, par contre démontrer la correction de l'algorithme glouton pour un système de monnaie donné est moins aisé.

preuve de correction pour un système simple

Voici une démonstration de la correction de l'algorithme du rendu de monnaie dans le cas d'un système de monnaie composé des pièces 1, 2 et 5.
La liste P est donc [5,2,1] et la somme à rendre est notée s
La solution produite par l'algorithme glouton est telle que :
$s = q_1 \times 5 + r_1$ avec $0 \leqslant r_1 < 5$
$r_1 = q_2 \times 2 + r_2$ avec $0 \leqslant r_2 < 2$
$r_2 = q_3 \times 1 + r_3$ avec $\underbrace{0 \leqslant r_3 < 1}_{\text{donc } r_3=0}$ d'où $r_2 = q_3 \times 1$
On retrouve que : $s = q_1 \times 5 + r_1 = q_1 \times 5 + q_2 \times 2 + r_2 = q_1 \times 5 + q_2 \times 2 + q_3 \times 1$
On vérifie ainsi que la répartition obtenue permet d'obtenir s. Montrons qu'elle est optimale :
Soit o_1, o_2, o_3 une solution optimale dans ce système canonique.
Cette solution est donc telle que $s = o_1 \times 5 + o_2 \times 2 + o_3 \times 1$
$o_3 < 2$ sinon on pourrait remplacer 2 pièces de 1 par une pièce de 2.
De même, $o_2 < 3$ sinon on pourrait remplacer 3 pièces de 2 par une pièce de 5 et une de 5.
On a de plus $2 \times o_2 + o_1 < 5$ car sinon on pourrait remplacer par une pièce de 5.
On a donc $s = o_1 \times 5 + o_2 \times 2 + o_3 \times 1$ avec $2 \times o_2 + o_1 < 5$
D'où o_1 est le quotient q_1 de la division euclidienne de s par 5 et $o_2 \times 2 + o_3 \times 1$ est le reste r_1 de la division euclidienne de s par 5.
De même, $o_3 < 2$, d'où o_2 est le quotient q_2 de la division euclidienne de r_1 par 2 et $o_3 \times 1$ est le reste r_2 de la division euclidienne de r_1 par 2.
et on en déduit que o_3 est le quotient q_3 de la division euclidienne de r_2 par 1 ;
D'où $o_1 = q_1$, $o_2 = q_2$ et $o_3 = q_3$
L'algorithme glouton est donc correct pour le système {1,2,5} et le système {1,2,5} est canonique.

IV Le problème du sac à dos

A principe

Le problème du "Sac à Dos" ou KP(Knapsack Problem) peut s'énoncer de plusieurs façons, parfois même sans parler de sacs ! Néanmoins, voici un énoncé classique :

Enoncé classique du KP

«Étant donné plusieurs objets possédant chacun un poids et une valeur, et étant donné un poids maximum pour le sac, quels objets faut-il mettre dans le sac de manière à maximiser la valeur totale sans dépasser le poids maximal autorisé ?»

Note pour l'enseignant

Si le groupe s'y prête alors, une présentation plus formelle du problème pourra être réalisée .

Présentation formelle

Soit un ensemble E $\{(v_1; p_1), (v_2; p_2), ..., (v_i; p_i), ..., (v_n; p_n)\}$ de n éléments et un sac à dos. On note :
v_i : la valeur de l'élément i p_i : le poids de l'élément i p_{max} le poids maximal autorisé dans le sac à dos
Une solution au KP est un ensemble $\{x_1, x_2, ..., x_i, ..., x_n\}$ tel que :

$$x_i = \begin{cases} 1 : \text{si l'élément } (v_i; p_i) \text{ est placé dans le sac} \\ 0 : \text{sinon} \end{cases} \quad \text{et} \sum_{i=1}^{n} p_i \times x_i \leqslant p_{max}$$

Une solution optimale est une solution au KP telle que $\sum_{i=1}^{n} v_i \times x_i$ soit maximale

B Critère de décision

Pour résoudre le KP avec cette stratégie, il nous faut définir **un critère de décision**; on rappelle qu'à chaque étape décisionnelle un choix optimal doit être réalisé sur la base d'un critère défini. Il s'agit d'un choix optimal, local.

Exemple 4

Soit un ensemble E de 5 objets $\{(1;2),(2;3),(9;4),(5;3),(4;6)\}$ où chaque objet est sous la forme (valeur en €, poids) ; voici quelques exemples de critères de décision triviaux :

— Critère n°1 : priorité pour les objets les plus légers

— Critère n°2 : priorité pour les objets les plus lourds

— Critère n°3 : priorité pour les objets de faible valeur

— Critère n°4 : priorité pour les objets de grande valeur

— ....

Le KP contiendra donc nécessairement un tri, qui peut représenter un coût important et augmenter significativement la complexité en temps de l'algorithme.

Par exemple, si on trie la liste selon le premier critère, elle devient $\{(1;2),(2;3),(5;3),(9;4),(4;6)\}$ ou $\{(1;2),(5;3),(2;3),(9;4),(4;6)\}$ puisque les objets C et D ont le même poids.

Un seul choix (poids ou valeur) peut être source d'aléas. En cas d'égalité sur le poids, il semble "naturel" de choisir comme critère secondaire "la plus grande valeur" puisque notre but est de maximiser la valeur totale du sac.

En 1957, G.Dantzig (1914-2005) introduit un critère souvent utilisé pour le KP :

— **Critère n°5 : priorité pour les objets de rapport $\frac{\text{valeur}}{\text{poids}}$ élevé**

Le choix du critère peut grandement influencer la réponse donnée (voir TP n°2) ; reprenons l'exemple précédent avec une contrainte supplémentaire inhérente au KP : le poids maximal dans le sac. On supposera $p_{max} = 10$:

— Liste initiale à trier selon un critère → $\{(1;2),(2;3),(9;4),(5;3),(4;6)\}$

— Critère n°1 → Liste triée $\{(1;2),(5;3),(2;3),(9;4),(4;6)\}$ → Réponse $\{1,1,0,1,0\}$ car $2+3+3 \leq 10 < 2+3+3+4$

— Critère n°5 → Liste triée $\{(9;4),(5;3),(4;6),(2;3),(1;2)\}$ → Réponse $\{0,1,1,1,0\}$ car $4+3+3 \leq 10 < 4+3+3+2$

Remarque : Les deux critères conduisent à des sacs avec 3 objets. La réponse indique avec des '1' la position dans la liste initiale des objets retenus et des '0' sinon. Par ailleurs, le poids total du 2^e sac est égale à p_{max} , **on dit parfois que le sac est maximal.** Enfin, la valeur du second sac est de 16 contre seulement 8 pour le premier : néanmoins, à ce stade, nous ne savons pas si 16 correspond à la solution optimale.

C Pseudo-Code : Algorithme "glouton"

voici une écriture en pseudo-code d'une fonction **KP** implémentant l'algorithme :

- **Pré-conditions :**
  ⋆ poids max p_{max} (réel positif)
  ⋆ Liste "Objets" triée $[(v_1,p_1),(v_2,p_2),...,(v_n,p_n)]$
- **Post-condition :** résultat = Liste "Reponse" contenant des 0 ou des 1 selon que l'objet est sélectionné ou pas

On suppose ici que l'on se dote d'un critère de décision et d'une liste **L d'objets triés par ordre décroissant selon notre critère.** La réponse donne la position dans la liste triée.

La complexité est d'ordre n et la terminaison est assurée par le variant i de la boucle « Pour ». .

```
1  Fonction(KP(P,s))
2  Sac ← 0 ;
3  Reponse← [  ] ;
4  pour i=0 à n-1 faire
5      si p_i+Sac ≤ p_max alors
6          Reponse[i] ← 1 ;
7          Sac ← Sac+p_i ;
8      sinon
9          Reponse[i] ← 0 ;
10     fin
11 fin
12 retourner Reponse
```

Annexe C

Feuille d'exercices

Sur cette feuille d'exercices, sont proposés des exercices portant sur le thème des algorithmes gloutons

I Compléments pédagogiques des exercices 1 à 3 :

Ces trois exercices sont débranchés.

Ces trois premiers exercices portent sur le problème du rendu de monnaie.
L'exercice 1 permet de tester à la main l'algorithme du rendu de monnaie avec un système canonique et de préparer les élèves au TP1. Les exercices 2 et 3 sont des exercices où les systèmes de monnaie ne sont pas canoniques. On a veillé dans ces exercices à leur donner un contexte.

II Compléments pédagogiques de l'exercice 4

Cet exercice est branché.

Cet exercice est un exercice plus ouvert car il ne fait pas appel aux deux algorithmes gloutons vus en classe et sera donc proposé aux élèves les plus rapides.

III Compléments pédagogiques de l'exercice 5

Cet exercice est débranché.

Cet exercice est un exercice d'introduction au problème du sac à dos et permet d'introduire le TP2.

IV Compléments pédagogiques des exercices 6 à 7 :

Ces deux exercices sont débranchés.

L'exercice 6 permet de tester à la main l'algorithme du sac à dos et de vérifier si le principe est bien compris. L'exercice 7 est un exercice moins guidé de l'algorithme du sac et qui permet la découverte des méthodes à employer en Python pour effectuer des tris selon des critères.

V Compléments pédagogiques de l'exercice 8

Cet exercice est branché.

Cet exercice est un exercice plus ouvert permettant de comparer la méthode brute du sac à dos avec les critères de l'algorithme du sac à dos.

Les algorithmes gloutons - Feuille d'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?
   (b) si on doit rendre 6€36?
   (c) si on doit rendre 3 €68?
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?

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

▷ **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										

1. Compléter les dernières lignes du tableau.
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?

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.

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

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

▷ **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.
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.

▷ **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.

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}

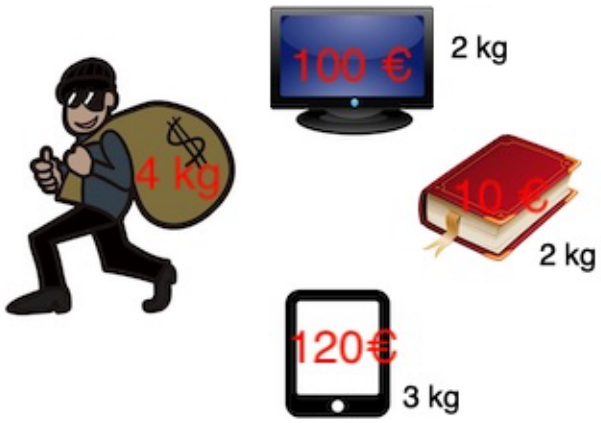
2. En utilisant les notations précédentes, quelle est la solution si le voleur vole les objets A et C ?
3. Soient trois objets de 4kg, 2kg et 1kg ainsi qu'un sac supportant jusqu'à 6kg. La solution {1, 0, 1} respecte-t-elle la condition sur la masse totale autorisée ?

▷ **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).

FIGURE 1 –



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

▷ **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 (décroissante) puis une liste triée par rapport décroissant $\frac{\text{valeur}}{\text{poids}}$
2. Appliquer l'algorithme glouton sur ces deux listes et écrire une liste réponse uniquement constituée de 0 ou de 1. Commenter.
3. En Python, que renvoie les instructions `L.sort(key = lambda a : a[1])` et `L.sort(key = lambda a : a[0], reverse=True)` ?

▷ **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.
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 $n \in \{4,7,8,10,15\}$.
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).
5. Quel est le temps d'exécution prédit par ce modèle lorsque n =50 ? Commenter le résultat obtenu.

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?

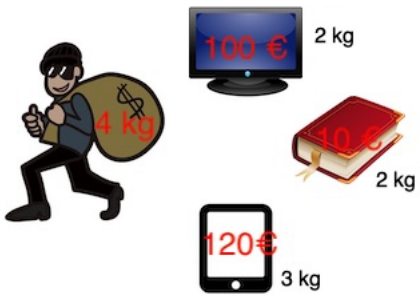
solution :

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

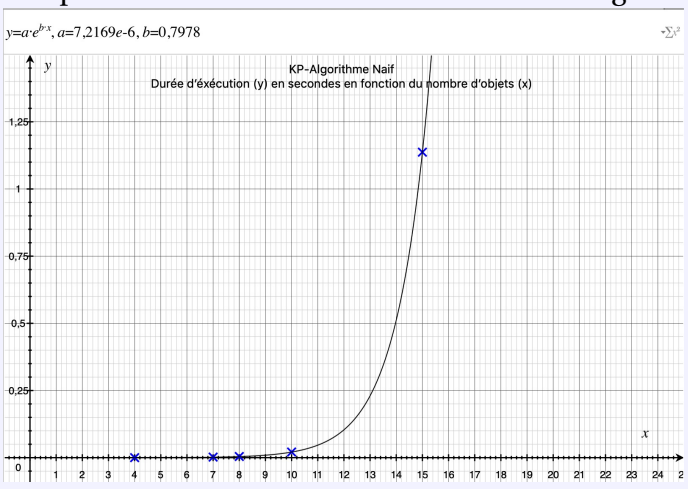
KPnaif : Réponses et Durée d'exécution (en s)					
Objets	n=4	n=7	n=8	n=10	n=15
Durée	1.291e-4	2.129e-3	4.348e-3	2.100e-2	1.137
Réponse	([1,1,0,0], 11, 25)	([0, 1, 0, 1, 0, 0, 1], 1735, 169)	([1, 1, 1, 1, 0, 1, 0, 0], 280, 102)	([1, 1, 1, 1, 0, 1, 0, 0, 0, 0], 309, 165)	([0, 1, 1, 0, 1, 0, 1, 1, 1, 0,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

Annexe D

TP

Deux TP sont prévus :
— TP1 : rendu de monnaie
— TP2 : sac à dos

I Compléments pédagogiques du TP1 :

Le TP1 s'instaure dans la continuité de ce qui a été fait en classe avant.
Prérequis : les mêmes que pour toute la séance.
Les parties sont en ordre croissant de difficulté et ont été préparées via l'exercice 1 fait en classe.
Remarque : le système de monnaie étant généralement de petite taille, les listes seront supposées prétriées

1 Principe général du TP

But du TP : programmer un appareil qui rend automatiquement la monnaie à l'aide d'un algorithme glouton de rendu de monnaie.
Dans les parties I et II : La caisse est sous la forme d'une liste contenant les valeurs des pièces par ordre décroissant. On suppose que l'on dispose d'autant de pièces que l'on veut.
Partie I :
but : écrire une caisse sous forme d'une liste et écrire une fonction qui retourne un rendu de monnaie sous la forme d'une liste
Partie II :
but : écrire une fonction qui retourne un rendu de monnaie sous la forme d'une liste de liste
Partie III :
La caisse est à présent sous la forme d'un dictionnaire qui prend en compte le nombre de pièces de chaque catégorie
but : écrire une caisse sous forme d'un tel dictionnaire et une fonction qui retourne un rendu de monnaie sous la forme d'un dictionnaire indiquant les quantités prises pour chaque pièce si c'est possible ou un message si le rendu est impossible

2 Objectifs

- vérifier la compréhension du principe de l'algorithme glouton du rendu de monnaie
- Revoir les différentes structures de données (de la liste simple, aux dictionnaires via les listes de listes)
- réinvestir les boucles et les tests
- Mettre en oeuvre un algorithme glouton en Python

3 Difficultés anticipées

1. comprendre la démarche à suivre pour résoudre le problème
2. Manipuler une liste dans une liste
3. Manipuler un dictionnaire et confusion clé/valeur dans un dictionnaire

4 Remédiation envisagées

1. ré-explication du principe
2. donner des exemples de cas simples
3. prendre des exemples où la clé n'est pas un nombre, où la manipulation est sans équivoque

II Compléments pédagogiques du TP2 :

1 Principe général du TP

Ce TP arrive en fin de séquence sur les gloutons ; il permet de réinvestir la notion de tri et viser davantage à comparer les réponses des algorithmes gloutons qu'à écrire les algorithmes eux-mêmes.
Partie I le tri :
On revient sur un principe de tri (supposé connu et maitrisé) ; l'élève propose une modification possible de son Tri Classique afin de l'adapter à un tableau à 2 dimensions. On rappelle brièvement le caractère immuable des tuples qui rend plus compliqué le tri d'une liste de tuples. On demande à l'élève de s'emparer de quelques lignes de la documentation officielle Python (éventuellement en anglais) afin d'analyser une syntaxe nouvelle. La notion de fonction lambda n'est pas introduite en tant que telle.
Partie II : Exécution de l'algorithme du sac à dos selon trois critères.
l'élève écrit ses trois algorithmes qui ne différent que par le critère de décision. il teste les codes sur un ensemble de données fourni dans un fichier annexe. On pourrait demander à un élève très à l'aise, de lire directement dans le fichier. On compare et commente la réponse des algorithmes : coût en temps, optimalité.

2 Objectifs

- réinvestissement des tris
- vérifier la compréhension du principe de l'algorithme glouton du sac à dos
- Revoir les différentes structures de données
- réinvestir les boucles et les tests
- Mettre en oeuvre un deuxième algorithme glouton en Python

3 Difficultés anticipées

1. Remobiliser ses connaissances sur les tris
2. comprendre la démarche à suivre pour effectuer l' étude des performances des algorithmes

4 Remédiation envisagées

1. ré-explication du principe
2. donner des exemples

TP1 - Le problème du rendu de monnaie

Le but de ce TP est de programmer un appareil qui rend automatiquement la monnaie avec des pièces.
Pour ce faire, on aura recours à l'algorithme de rendu de monnaie vu la séance précédente.

On considère dans les parties I et II que l'on dispose d'autant de pièces que l'on veut.

I Une première approche

On se propose de coder en Python l'algorithme de rendu de monnaie vu la séance précédente. Pour faciliter les calculs, on va exprimer les valeurs de toutes les pièces en centimes que l'on stockera dans une liste.

1. Ecrire sous forme d'une liste nommée ***caisse*** contenant les valeurs par ordre décroissant.
2. Ecrire une fonction ***rendmonnaie*** qui prend en paramètre la somme à rendre ***s*** en € et une caisse ***c*** de type list et qui renvoie une liste contenant la quantité de monnaie à rendre pour chaque valeur.
   Par exemple, `rendmonnaie(2.34,caisse)` doit afficher : [1, 0, 0, 1, 1, 0, 2, 0]
   où [1, 0, 0, 1, 1, 0, 2, 0] indique que l'on a rendu une pièce de 2 €, une pièce de 20 centimes, une pièce de 10 centimes et deux pièces de 2 centimes.

II Une première amélioration

Ecrire une fonction ***rendmonnaie2*** qui prend en paramètre la somme à rendre ***s*** en € et une caisse ***c*** de type list et qui renvoie une liste constituée par les coupes [pièces,quantité rendue]
Par exemple, `rendmonnaie2(2.34,caisse)` doit afficher : [[200, 1], [100, 0], [50, 0], [20, 1], [10, 1], [5, 0], [2, 2], [1, 0]]

A partir de la partie III, on disposera pour chaque pièce d'une certaine quantité.

III Avec un dictionnaire

1. On représente maintenant le contenu contenu dans le monnayeur par un dictionnaire :
   - la valeur faciale de la pièce comme cle
   - le nombre de pièces de cette valeur contenue dans le monnayeur

   Créer un dictionnaire nommé ***caisse2*** répondant à ces conditions.
   La quantité prise par la pièce sera générée aléatoirement avec le module random.
2. Ecrire une fonction ***rendmonnaie3*** qui prend en paramètre la somme à rendre ***s*** en € et une caisse ***c*** de type dictionnaire et qui renvoie un dictionnaire constituée par les couples [pièces,quantité rendue] si c'est possible et le message 'il n'y a plus de monnaie disponible' si le rendu est impossible
   Par exemple, `rendmonnaie3(2.34,caisse2)` doit afficher :
   {200 : 1, 100 : 0, 50 : 0, 20 : 1, 10 : 1, 5 : 0, 2 : 2, 1 : 0}
   si la caisse avait la répartition {200 : 6, 100 : 0, 50 : 1, 20 : 9, 10 : 1, 5 : 8, 2 : 9, 1 : 6}
   il n'y a plus assez de monnaie si la caisse avait la répartition suivante {200 : 0, 100 : 1, 50 : 1, 20 : 1, 10 : 2, 5 : 2, 2 : 0, 1 : 1}.

Pour aller plus loin ...

Faire l'exercice 4 de la feuille d'exercices

Correction du TP 1

Partie I

question 1

In [1]:
```
#définition de la caisse
caisse=[200,100,50,20,10,5,2,1]
caisse
```

Out[1]:
```
[200, 100, 50, 20, 10, 5, 2, 1]
```

question 2

In [2]:
```
def rendmonnaie(s,c):
    n=len(c)
    rendu=len(c)*[0]
    somme_a_rendre=s*100
    i=0
    while somme_a_rendre>0 and i<=n-1:
        while somme_a_rendre>=c[i] :
            somme_a_rendre=somme_a_rendre-c[i]
            rendu[i]=rendu[i]+1
        i=i+1
    return rendu
```

Tests

In [3]:
```
rendmonnaie(2.34,caisse)
```

Out[3]:
```
[1, 0, 0, 1, 1, 0, 2, 0]
```

In [4]:
```
rendmonnaie(5.71,caisse)
```

Out[4]:
```
[2, 1, 1, 1, 0, 0, 0, 1]
```

In [5]:
```
#autre version
def rendmonnaiebis(s,c):
    n=len(c)
    rendu=len(c)*[0]
    somme_a_rendre=s*100
    i=0
    while somme_a_rendre>0 and i<=n-1:
        if somme_a_rendre>=c[i] :
            rendu[i]=int(somme_a_rendre//c[i])
            somme_a_rendre=somme_a_rendre%c[i]
        i=i+1
    return rendu
```

Tests

In [6]:
```
rendmonnaiebis(2.34,caisse)
```

Out[6]:
```
[1, 0, 0, 1, 1, 0, 2, 0]
```

In [7]:
```
rendmonnaiebis(5.71,caisse)
```

Out[7]:
```
[2, 1, 1, 1, 0, 0, 0, 1]
```

Partie II

In [8]:
```
def rendmonnaie2(s,c):
    n=len(c)
    rendu=[[c[i],0] for i in range(len(c))]
    somme_a_rendre=s*100
    i=0
    while somme_a_rendre>0 and i<=n-1:
        while somme_a_rendre>=c[i] :
            somme_a_rendre=somme_a_rendre-c[i]
            rendu[i][1]=rendu[i][1]+1
        i=i+1
    return rendu
```

Tests

In [9]:
```
rendmonnaie2(2.34,caisse)
```

Out[9]:
```
[[200, 1], [100, 0], [50, 0], [20, 1], [10, 1], [5, 0], [2, 2], [1, 0]]
```

In [10]:
```
rendmonnaie2(5.71,caisse)
```

Out[10]:
```
[[200, 2], [100, 1], [50, 1], [20, 1], [10, 0], [5, 0], [2, 0], [1, 1]]
```

In [11]:
```
#version bis
def rendmonnaie2bis(s,c):
    n=len(c)
    rendu=[[c[i],0] for i in range(len(c))]
    somme_a_rendre=s*100
    i=0
    while somme_a_rendre>0 and i<=n-1:
        if somme_a_rendre>=c[i] :
            rendu[i][1]=int(somme_a_rendre//c[i])
            somme_a_rendre=somme_a_rendre%c[i]
        i=i+1
    return rendu
```

In [12]:
```
rendmonnaie2bis(2.34,caisse)
```

Out[12]:
```
[[200, 1], [100, 0], [50, 0], [20, 1], [10, 1], [5, 0], [2, 2], [1, 0]]
```

In [13]:
```
rendmonnaie2bis(5.71,caisse)
```

Out[13]:
```
[[200, 2], [100, 1], [50, 1], [20, 1], [10, 0], [5, 0], [2, 0], [1, 1]]
```

Partie III

question 1

In [14]:
```
import random
#Création d'un dictionnaire contenant une quantité aléatoire de pièce
caisse2={caisse[i] : random.randint(0,5)  for i in range(len(caisse))}
caisse2
```

Out[14]:
```
{200: 0, 100: 2, 50: 4, 20: 1, 10: 4, 5: 0, 2: 4, 1: 0}
```

Question 2

In [15]:
```
#Version avec dictionnaire et contrôle de la quantité de pièces disponible
def rendmonnaie3(s,d):
    somme_a_rendre=s*100
    rendu={i:0 for i in d}
    for i in caisse:
        while somme_a_rendre>=i and caisse2[i]>0:
            somme_a_rendre=somme_a_rendre-i
            rendu[i]=rendu[i]+1
            caisse2[i]=caisse2[i]-1
    if somme_a_rendre>0:
           return print("il n'y a plus assez de monnaie")
    else :
        return rendu
```

Remarque : pour le moment, cette partie est codée avec une boucle pour car il y a de très grandes chances que ce soit la solution choisie par les élèves. En effet le recours au dictionnaire rend l'utilisation de la boucle while plus complexe.
On pourra pour les élèves plus rapides et donc les plus à l'aise leur demander d'y réflechir.

Tests

In [16]:
```
print("caisse à notre disposition :",caisse2)
print("rendu : ",rendmonnaie3(2.34,caisse2))
```

```
caisse à notre disposition : {200: 0, 100: 2, 50: 4, 20: 1, 10: 4, 5: 0, 2:
4, 1: 0}
rendu :  {200: 0, 100: 2, 50: 0, 20: 1, 10: 1, 5: 0, 2: 2, 1: 0}
```

In [17]:
```
print("caisse à notre disposition :",caisse2)
print("rendu :",rendmonnaie3(4.54,caisse2))
```

```
caisse à notre disposition : {200: 0, 100: 0, 50: 4, 20: 0, 10: 3, 5: 0, 2:
2, 1: 0}
il n'y a plus assez de monnaie
rendu : None
```

In [ ]:
```

```

TP2 - Le problème du rendu de monnaie

I Retour sur les Tris

1. Ecrire une fonction **extraire** qui accepte en paramètres, une liste de liste **l** et un entier **n** dans [0 ; 2] .La fonction doit retourner la liste triée en fonction du n-ième élément de chaque sous-liste. On pourra choisir le tri insertion par exemple.

Console Python

```
>>> L= [ [1, 5, 6], [8, 10, 2], [3, 3, 5],[4, 8, 1] ]
>>> extraire(L,0)
[[1, 5, 6], [3, 3, 5], [4, 8, 1], [8, 10, 2]]
>>> extraire(L,1)
[[3, 3, 5], [1, 5, 6], [4, 8, 1], [8, 10, 2]]
>>> extraire(L,2)
[[4, 8, 1], [8, 10, 2], [3, 3, 5], [1, 5, 6]]
```

2. Pourquoi le code précédent ne fonctionne-t-il pas (tel quel) avec une liste de tuples ?
3. A l'aide de l'exercice 7, expliquer le sens de l'instruction `List.sort(key = lambda a : a[1])`

II Algorithme du sac à dos

A Critère de Poids

Nous allons écrire un premier algorithme glouton de critère **"placer d'abord dans le sac, les objets les plus lourds."** En cas d'égalité, on prendra "au hasard" un objet parmi les indécis. On pourra utiliser la méthode **list.sort()** comme définie sur la documentation officielle Python

1. Ecrire une fonction **gloutonP** de paramètres **l** et **maxi**, respectivement une liste de tuples (valeur en €, poids en kg) et un entier correspondant au poids maximal supporté par le sac. La fonction doit retourner un triplet **reponse, valeur, poids** comme dans l'exemple ci-dessous :

Console Python

```
>>> liste=[(7,13), (4,12), (3,8), (3,10)]
>>> gloutonP(liste,30)
([1, 1, 0, 0], 11, 25)
```

   Le fichier **Datas.txt** contient un ensemble de données correspondant à 5 magasins fictifs : t4, t7, t8, t10 et t15. Chaque ligne du fichier contient, le nom du magasin, une liste d'objets et le poids maximal autorisé dans le sac du voleur.
2. A l'aide de la fonction **perf_counter** du module **time**, calculer la durée d'exécution sur les ensembles de données fournis dans le fichier Datas.txt. Les durées seront obtenues en réalisant une moyenne sur 100 appels de la fonction **gloutonP**. Noter vos résultats dans les tableaux en annexe A (avec 4 chiffres significatifs sur le temps).

B Critère de Valeur

Pour notre deuxième algorithme glouton, le critère retenu est **"placer d'abord dans le sac, les objets de plus grande valeur"**. Ecrire une fonction **gloutonV** qui accepte les mêmes paramètres que gloutonP et qui retourne le triplet **reponse, valeur, poids** comme dans l'exemple ci-après ; réaliser ensuite les mêmes tests à l'aide du fichier **datas.txt** pour compléter les tableaux de l'annexe A.

Console Python

```
>>> liste=[(7,13), (4,12), (3,8), (3,10)]
>>> gloutonV(liste,30)
([1, 1, 0, 0], 11, 25)
```

C Critère de Valeur Pondérée par le Poids

Pour notre dernier algorithme glouton, le critère retenu est **"placer d'abord dans le sac, les objets dont le rapport $\frac{\text{valeur}}{\text{poids}}$ est le plus élevé**. Ecrire une fonction **gloutonVPP** qui accepte les mêmes paramètres que gloutonP et retourne le triplet **reponse, valeur, poids** comme dans l'exemple ci-après; réaliser ensuite les mêmes tests à l'aide du fichier **datas.txt** pour compléter les tableaux en annexe A.

Console Python

```
>>> liste=[(7,13), (4,12), (3,8), (3,10)]
>>> gloutonVPP(liste,30)
([1, 1, 0, 0], 11, 25)
```

D Comparaison des trois critères

1. Comparer les durées d'exécution des trois algorithmes gloutons. Commenter.
2. Discuter du choix du critère de sélection des algorithmes gloutons en fonction de n.

Annexe A : comparaison des performances des différents algorithmes

Durée d'exécution (en s)* de différents algorithme en fonction du nombre d'objets pour le KP					
Algorithme	n=4	n=7	n=8	n=10	n=15
GLOUTONP					
GLOUTONV					
GLOUTONVPP					
Algorithme Naif	1.291e-4	2.129e-3	4.348e-3	2.100e-2	1.137

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

Réponses** des différents algorithmes en fonction du nombre d'objets pour le KP					
Algorithme	n=4	n=7	n=8	n=10	n=15
GLOUTONP					
GLOUTONV					
GLOUTONVPP					
Sol. Optimale	([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)

Pour aller plus loin ... : exercice 8 de la feuille d'exercices

Correction du TP n°2: Le problème du Sac à Dos

4 juillet 2019

I Retour sur les Tris

1. Ecrire une fonction **extraire** qui accepte en paramètres, une liste de liste **l** et un entier **n** dans [0 ; 2] .La fonction doit retourner la liste triée en fonction du n-ième élément de chaque sous-liste. On pourra choisir le tri insertion par exemple.

Console Python

```
>>> L= [ [1, 5, 6], [8, 10, 2], [3, 3, 5],[4, 8, 1] ]
>>> extraire(L,0)
[[1, 5, 6], [3, 3, 5], [4, 8, 1], [8, 10, 2]]
>>> extraire(L,1)
[[3, 3, 5], [1, 5, 6], [4, 8, 1], [8, 10, 2]]
>>> extraire(L,2)
[[4, 8, 1], [8, 10, 2], [3, 3, 5], [1, 5, 6]]
```

```
1 def extraire(t,n):
2     for i in range(1,len(t)):
3         v = t[i][n]
4         j = i-1
5         while(j>=0 and t[j][n]>v):
6             t[j+1],t[j] = t[j],t[j+1]
7             j = j-1
8         t[j+1][n] = v
9     return t
```

2. Pourquoi le code précédent ne fonctionne-t-il pas (tel quel) avec une liste de tuples ?

Solution: Le tuple est un objet que l'on ne peut pas modifier donc une instruction comme celle de la ligne 8 par exemple n'est pas autorisée. Il faudrait par exemple, copier la liste de tuples en liste de liste puis trier et retourner une liste de tuples.

3. A l'aide de l'exercice 7, expliquer le sens de l'instruction `List.sort(key = lambda a : a[1])`

Solution: "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. Par ailleurs, reverse=True indique qu'il s'agit d'un tri "décroissant".

II Algorithme du sac à dos

A Critère de Poids

Nous allons écrire un premier algorithme glouton de critère **"placer d'abord dans le sac, les objets les plus lourds."** En cas d'égalité, on prendra "au hasard" un objet parmi les indécis. On pourra utiliser la méthode **list.sort()** comme définie sur la documentation officielle Python

4. Ecrire une fonction **gloutonP** qui accepte deux paramètres **l** et **maxi**, respectivement une liste de tuples (valeur en €, poids en kg) et un entier correspondant au poids maximal supporté par le sac. La fonction doit donc retourner un triplet **reponse, valeur, poids** comme dans l'exemple ci-dessous :

Console Python

```
>>> liste=[(7,13), (4,12), (3,8), (3,10)]
>>> print(gloutonP(liste,30))
([1, 1, 0, 0], 11, 25)
```

Solution: (voir Annexe B) On crée une copie de la liste de tuples **lig(2 et 3)** puis on trie la liste "temporaire" selon le 2e élément (tup[1]) dans l'ordre décroissant (reverse=True) ; **lig(8)** on rentre dans une boucle "de la longueur" du tableau. Si le poids actuel du sac plus le poids du prochain objet est inférieur ou égal au poids maxi **lig(9)**, alors on ajoute cet objet à la solution **lig(10)**, puis on augmente le poids et la valeur du sac **lig(11 et 12)**. Enfin, on réalise une compréhension en utilisant la liste initiale pour afficher soit 1 soit 0 selon que l'élément est dans la solution ou pas.

Le fichier **Datas.txt** contient un ensemble de données correspondant à 5 magasins fictifs : t4, t7, t8, t10 et t15. Chaque ligne du fichier contient, le nom du magasin, une liste d'objets et le poids maximal autorisé dans le sac du voleur.

5. A l'aide de la fonction **perf_counter** du module **time**, calculer la durée d'exécution sur les ensembles de données fournis dans le fichier Datas.txt. Les durées seront obtenues en réalisant une moyenne sur 100 appels de la fonction **gloutonP**. Noter vos résultats dans les tableaux en annexe A (avec 4 chiffres significatifs sur le temps).

Solution: (voir Annexe A)

B Critère de Valeur

Pour notre deuxième algorithme glouton, le critère retenu est **"placer d'abord dans le sac, les objets de plus grande valeur"**. Ecrire une fonction **gloutonV** qui accepte les mêmes paramètres que gloutonP et retourne le triplet **reponse, valeur, poids** comme dans l'exemple ci-après ; réaliser ensuite les mêmes tests à l'aide du fichier **datas.txt** pour compléter les tableaux de l'annexe A.

Console Python

```
>>> liste=[(7,13), (4,12), (3,8), (3,10)]
>>> gloutonV(liste,30)
([1, 1, 0, 0], 11, 25)
```

C Critère de Valeur Pondérée par le Poids

Pour notre dernier algorithme glouton, le critère retenu est **"placer d'abord dans le sac, les objets dont le rapport $\frac{valeur}{poids}$ est le plus élevé**. Ecrire une fonction **gloutonVPP** qui accepte les mêmes paramètres que gloutonP et retourne le triplet **reponse, valeur, poids** comme dans l'exemple ci-après ; réaliser ensuite les mêmes tests à l'aide du fichier **datas.txt** pour compléter les tableaux en annexe A.

Solution: 6. - 7. (voir Annexe B) Algorithmes identiques à gloutonP. Seule la clé du tri change : la liste "temporaire" est triée soit selon le 1er élément (tup[0]) soit selon le rapport $\frac{tup[0]}{tup[1]}$ et dans l'ordre décroissant (reverse=True).

Console Python

```
>>> liste=[(7,13), (4,12), (3,8), (3,10)]
>>> print(gloutonVPP(liste,30))
([1, 1, 0, 0], 11, 25)
```

D Comparaison des trois critères

6. Comparer les durées d'exécution des trois algorithmes gloutons. Commenter.

Solution: (voir Annexe A) gloutonP, gloutonV et gloutonVPP possèdent des durées d'exécution comparables (de l'ordre de $1 \times 10^{-5} s$) et ceci quel que soit n (pour notre petit échantillon). Les algorithmes gloutons semblent posséder une certaine "stabilité en temps".

7. Discuter du choix du critère de sélection des algorithmes gloutons en fonction de n.

Solution: (voir Annexe A) gloutonP semble le plus mauvais des trois car il ne fait jamais mieux que les deux autres ; gloutonV semble le meilleur pour des petits échantillons mais à partir de n=10 gloutonVPP domine les deux autres. Il faudrait toutefois pouvoir réaliser des tests sur des valeurs de n bien supérieures pour "solidifier" notre hypothèse.

A Performances des différentes algorithmes

Durée d'exécution (en s)* de différents algorithme en fonction du nombre d'objets pour le KP					
Algorithme	n=4	n=7	n=8	n=10	n=15
GLOUTONP	8.490e-6	1.284e-5	1.967e-5	1.857e-5	2.559e-5
GLOUTONV	1.059e-5	8.202e-6	1.487e-5	1.209e-5	3.243e-5
GLOUTONVPP	2.071e-5	7.543e-6	1.990e-5	1.431e-5	2.716e-5
Algorithme Naif	1.291e-4	2.129e-3	4.348e-3	2.100e-2	1.137

TABLE 1 – Durée d'exécution

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

Réponses** des différents algorithmes en fonction du nombre d'objets pour le KP					
Algorithme	n=4	n=7	n=8	n=10	n=15
GLOUTONP	([1,1,0,0], 11,25)	([0, 1, 0, 1, 0, 0, 1], 1735, 169)	([1, 0, 0, 0, 1, 0, 1, 0], 65, 102)	([0, 0, 0, 0, 0, 0, 1, 0, 1, 0], 154, 152)	([1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1], 1315, 682)
GLOUTONV	([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, 0, 0, 1, 0, 0, 0, 0, 1, 0], 247, 156)	([1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1], 1315, 682)
GLOUTONVPP	([1, 0, 1, 0], 10,21)	([1, 1, 1, 0, 0, 0, 0], 1478, 140)	([1, 1, 1, 1, 0, 0, 0, 1], 266, 73)	([1, 1, 1, 1, 0, 1, 0, 0, 0, 0], 309, 165)	([0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0], 1441, 740)
Sol. Optimale	([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)

TABLE 2 – Comparaison des réponses obtenues

** Code couleur pour les algorithmes gloutons
Réponse optimale ou réponse approchée la plus proche
Réponse approchée "intermédiaire"
Réponse approchée "éloignée"

B Question 4,6,7 - Proposition d'algorithmes

```python
def gloutonP(l,maxi):
    tmp=[x for x in l]
    tmp.sort(key=lambda tup: tup[1],reverse=True)
    valeur=0
    poids=0
    i=0
    solution=[]
    while i<len(tmp):
        if (poids+tmp[i][1]) <=maxi:
            solution.append(tmp[i])
            poids +=tmp[i][1]
            valeur+=tmp[i][0]
        i+=1
    reponse=[1 if x in solution else 0 for x in l]
    return reponse,valeur,poids
```

```python
def gloutonV(l,maxi=30):
    tmp=[x for x in l]
    l.sort(key=lambda tup: tup[0],reverse=True)
    valeur=0
    poids=0
    solution=[]
    i=0
    while i<len(tmp):
        if (poids+tmp[i][1]) <=maxi:
            solution.append(tmp[i])
            poids +=tmp[i][1]
            valeur+=tmp[i][0]
        i+=1
    reponse=[1 if x in solution else 0 for x in tmp]
    return reponse,valeur,poids
```

```python
def gloutonVPP(l,maxi=30):
    tmp=[x for x in l]
    l.sort(key=lambda tup: tup[0]/tup[1],reverse=True)
    valeur=0
    poids=0
    solution=[]
    i=0
    while i<len(tmp):
        if (poids+tmp[i][1]) <=maxi:
            solution.append(tmp[i])
            poids +=tmp[i][1]
            valeur+=tmp[i][0]
        i+=1
    reponse=[1 if x in solution else 0 for x in tmp]
    return reponse,valeur,poids
```

Annexe E

Notions mises au lexique

Les algorithmes gloutons - Lexique

Résoudre un problème d'optimisation	rechercher la(les) solution de valeur optimale du problème
Algorithme glouton	Un algorithme glouton est un algorithme qui fait un choix optimum local à chaque étape, dans le but d'obtenir une solution optimale globale au problème. Il n'y a pas de retour en arrière : à chaque étape de décision dans l'algorithme, le choix effectué est définitif.
Heuristique gloutonne	stratégie pour résoudre un problème en effectuant des choix locaux optimaux dans le but d'obtenir une solution optimale globale au problème mais sans pouvoir le garantir
Algorithme du rendu de monnaie	étant donné un système de monnaie, rendre une somme donnée avec le nombre minimal de pièces et billets.
Système de monnaie canonique	un système de pièces pour lequel l'algorithme glouton est optimal
Algorithme du sac à dos	Étant donné plusieurs objets possédant chacun un poids et une valeur et étant donné un poids maximum pour un sac donné, déterminer quels objets, il faut mettre dans le sac de manière à maximiser la valeur totale sans dépasser le poids maximal autorisé pour le sac.
recherche exhaustive	on explore systématiquement tous les choix possibles

Les notions de :

— complexité d'un algorithme

— terminaison d'un algorithme

— correction d'un algorithme

ont déjà été amendées au lexique lors des chapitres précédents.

Annexe F

Evaluation

NOM : ..... PRENOM : ..... Classe : .....

Evaluation - Les algorithmes gloutons

1. On suppose dans cette question, que l'on dispose des pièces de monnaie suivantes (sans limite d'effectif) : 5 €, 2 €, 1 €, 50 centimes, 20 centimes, 10 centimes, 5 centimes, 2 centimes et 1 centime.
   Pour 11,67 € payé avec un billet de 20 €, le rendu de monnaie en pièces obtenu à l'aide d'un algorithme glouton est :
   Cocher la bonne réponse :
   - ◯ Réponse A :
     2 pièces de 5 €, 1 pièce de 1 €, 1 pièce de 50 centimes, 1 pièce de 10 centimes, 1 pièce de 5 centimes et 1 pièce de 2 centimes.
   - ◯ Réponse B :
     1 pièce de 5 €, 3 pièces de 1 €, 1 pièce de 20 centimes, 1 pièce de 10 centimes, 1 pièce de 2 centimes et 1 pièce de 1 centime.
   - ◯ Réponse C :
     1 pièce de 5 €, 1 pièce de 1 €, 2 pièce de 2 €, 3 pièces de 10 centimes, 1 pièce de 2 centimes et 1 pièce de 1 centime.
   - ◯ Réponse D :
     1 pièce de 5 €, 1 pièce de 1 €, 2 pièce de 2 €, 1 pièce de 20 centimes, 1 pièce de 10 centimes, 1 pièce de 2 centimes et 1 pièce de 1 centime.

2. On dispose d'un système de monnaie (sans limite d'effectif) de 4⋆, 3⋆ et 1⋆, on cherche la façon de payer 6 ⋆ selon le principe glouton. La répartion est :
   Cocher la bonne réponse :
   - ◯ Réponse A : 3 ⋆ , 3 ⋆
   - ◯ Réponse B : 3 ⋆ , 1 ⋆, 1 ⋆, 1 ⋆
   - ◯ Réponse C : 4 ⋆ , 1 ⋆, 1 ⋆
   - ◯ Réponse D : 1 ⋆, 1 ⋆, 1 ⋆, 1 ⋆,1 ⋆, 1 ⋆

3. L'algorithme glouton (Cocher la bonne réponse) :
   - ◯ Réponse A : assure de trouver une solution optimale
   - ◯ Réponse B : entraîne un coût en temps exponentiel
   - ◯ Réponse B : donne parfois une solution optimale

4. Soit une liste L=[(7,13),(4,12),(3,8),(3,10)] ; en python, l'instruction **L.sort(key = lambda a : a[1], reverse=True)** :
   Cocher la bonne réponse :
   - ◯ Réponse A : trie les tuples par ordre décroissant en fonction de l'élément d'indice1
   - ◯ Réponse B : trie à partir premier élément de la liste
   - ◯ Réponse C : trie les tuples par ordre croissant en fonction de l'élément d'indice 0

5. Quels mots complètent (dans l'ordre) la définition du problème du sac à dos :
   «Étant donné plusieurs objets possédant chacun un ..... et une ..... et étant donné un poids ..... pour le sac, quels objets faut-il mettre dans le sac de manière à ..... la valeur totale sans dépasser le poids maximal autorisé pour le sac ?»
   Cocher la bonne réponse :
   - ◯ Réponse A : indice, liste, minimal, minimiser
   - ◯ Réponse B : poids, valeur, minimal, maximiser
   - ◯ Réponse C : poids, valeur, maximal, maximiser

6. Soit une liste de tuples (valeur,poids) L=[(15, 2),(100, 20),(90, 20),(60, 30),(40, 40),(15, 30),(10, 60),(1, 1)] . On dispose d'un sac de poids maximal autorisé $p_{max} = 90$ et on utilise l'algorithme glouton avec un critère sur la valeur. L'algorithme propose une solution avec un sac de valeur totale :
   Cocher la bonne réponse :
   - ◯ Réponse A : 245
   - ◯ Réponse B : 265
   - ◯ Réponse C : 266

NOM : ..... PRENOM : ..... Classe : .....

Evaluation - Les algorithmes gloutons

1. On suppose dans cette question, que l'on dispose des pièces de monnaie suivantes (sans limite d'effectif) : 5 €, 2 €, 1 €, 50 centimes, 20 centimes, 10 centimes, 5 centimes, 2 centimes et 1 centime.
   Pour 11,67 € payé avec un billet de 20 €, le rendu de monnaie en pièces obtenu à l'aide d'un algorithme glouton est :
   Cocher la bonne réponse :
   - ◯ Réponse A :
     2 pièces de 5 €, 1 pièce de 1 €, 1 pièce de 50 centimes, 1 pièce de 10 centimes, 1 pièce de 5 centimes et 1 pièce de 2 centimes.
   - ◯ Réponse B :
     1 pièce de 5 €, 3 pièces de 1 €, 1 pièce de 20 centimes, 1 pièce de 10 centimes, 1 pièce de 2 centimes et 1 pièce de 1 centime.
   - ◯ Réponse C :
     1 pièce de 5 €, 1 pièce de 1 €, 2 pièce de 2 €, 3 pièces de 10 centimes, 1 pièce de 2 centimes et 1 pièce de 1 centime.
   - ◯ Réponse D :
     1 pièce de 5 €, 1 pièce de 1 €, 2 pièce de 2 €, 1 pièce de 20 centimes, 1 pièce de 10 centimes, 1 pièce de 2 centimes et 1 pièce de 1 centime.

   Solution: C'est la réponse D qui convient.

2. On dispose d'un système de monnaie (sans limite d'effectif) de 4⋆, 3⋆ et 1⋆, on cherche la façon de payer 6 ⋆ selon le principe glouton. La répartion est :
   Cocher la bonne réponse :
   - ◯ Réponse A : 3 ⋆ , 3 ⋆
   - ◯ Réponse B : 3 ⋆ , 1 ⋆, 1 ⋆, 1 ⋆
   - ◯ Réponse C : 4 ⋆ , 1 ⋆, 1 ⋆
   - ◯ Réponse D : 1 ⋆, 1 ⋆, 1 ⋆, 1 ⋆,1 ⋆, 1 ⋆

   Solution: C'est la réponse C qui convient.

3. L'algorithme glouton (Cocher la bonne réponse) :
   - ◯ Réponse A : assure de trouver une solution optimale
   - ◯ Réponse B : entraîne un coût en temps exponentiel
   - ◯ Réponse B : donne parfois une solution optimale

   Solution: C

4. Soit une liste L=[(7,13),(4,12),(3,8),(3,10)] ; en python, l'instruction **L.sort(key = lambda a : a[1], reverse=True)** :
   Cocher la bonne réponse :
   - ◯ Réponse A : trie les tuples par ordre décroissant en fonction de l'élément d'indice1
   - ◯ Réponse B : trie à partir premier élément de la liste
   - ◯ Réponse C : trie les tuples par ordre croissant en fonction de l'élément d'indice 0

   Solution: A

5. Quels mots complètent (dans l'ordre) la définition du problème du sac à dos :
   «Étant donné plusieurs objets possédant chacun un ..... et une ..... et étant donné un poids ..... pour le sac, quels objets faut-il mettre dans le sac de manière à ..... la valeur totale sans dépasser le poids maximal autorisé pour le sac ?»
   Cocher la bonne réponse :
   - ◯ Réponse A : indice, liste, minimal, minimiser
   - ◯ Réponse B : poids, valeur, minimal, maximiser
   - ◯ Réponse C : poids, valeur, maximal, maximiser

Solution: C

6. Soit une liste de tuples (valeur,poids) L=[(15, 2),(100, 20),(90, 20),(60, 30),(40, 40),(15, 30),(10, 60),(1, 1)] . On dispose d'un sac de poids maximal autorisé $p_{max} = 90$ et on utilise l'algorithme glouton avec un critère sur la valeur. L'algorithme propose une solution avec un sac de valeur totale :
   Cocher la bonne réponse :
   - ◯ Réponse A : 245
   - ◯ Réponse B : 265
   - ◯ Réponse C : 266

Solution: C

Annexe G

Frise chronologique