Chapitre 11 : Algorithmes avancés

Dans ce chapitre, nous allons travailler sur des algorithmes complexes de manière guidée : en raison du confinement, chaque algorithme se fera à la maison et la correction sera donnée en cours de semaine.

Le but n'est pas de recopier ou d'attendre la correction mais d'essayer de comprendre le fonctionnement de ces algorithmes un peu plus complexes que les algorithmes vus chapitre 7.

I. Algorithmes gloutons

1) Introduction : problème du sac à dos

Un cambrioleur possède un sac à dos d'une contenance maximum de 30 kg. Au cours d'un de ses cambriolages, il a la possibilité de dérober 4 objets A, B, C et D qui ont les caractéristiques suivantes :

objet	A	B	C	D	E
masse	13 kg	12 kg	6 kg	6 kg	5 kg
valeur	700 €	650 €	250 €	400 €	100 €

❐ — Exercice 1 —

Déterminez les objets que le cambrioleur aura intérêt à dérober, sachant que :

- tous les objets dérobés devront tenir dans le sac à dos (30 kg maxi)
- le cambrioleur cherche évidemment à obtenir **le gain maximum**.

Votre résultat est-il le meilleur ? Pouvez-vous en être certain ?

❒ — Exercice **2** —

Calcul de la complexité et détermination de a :

- Combien de combinaisons possibles y-a-t'il pour 3 objets ? pour 4 objets ? Dans ces deux cas, on écrira une formule faisant intervenir les combinaisons $\binom{n}{k}$. La définition des combinaisons est : $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ avec $k! = k \times (k-1) \times (k-2) \ldots \times 2 \times 1$ la factorielle de k .
- À partir de la formule trouvée dans les deux exemples, en déduire une formule générale faisant intervenir le symbole somme Σ .
- Écrire une fonction Python **bruteForce** permettant de trouver le nombre de combinaisons possibles pour un problème de taille n (variable d'entrée) si on résout le problème du sac à dos de manière dite "brute force". On devra écrire une fonction **factorielle** et une fonction **combinaison**.
- Représentez la courbe avec matplotlib en échelle semi-logarithmique selon l'axe des ordonnées, en utilisant cette échelle de données :

  listeX = range(3,50)

  listeY = [bruteForce(i) for i in listeX]
- Confirmez vous la complexité exponentielle ?

$\binom{n}{1}$ est le nombre de combinaisons que l'on peut faire avec une lettre parmi n.

$\binom{n}{2}$ est le nombre de combinaisons que l'on peut faire avec une lettre parmi n.

$$\binom{n}{1} + \binom{n}{2} + \binom{n}{3} + \ldots + \binom{n}{n} = \sum_{k=1}^{n} \binom{n}{k} = 2^n - 1$$

À la place de cette méthode "brute force", il est possible d'utiliser une méthode dite **gloutonne** (greedy en anglais).

2) Les méthodes gloutonnes

La résolution d'un problème dont l'on cherche un optimum se fait généralement par étapes : à chaque étape, on doit faire un choix. Par exemple, dans le problème du sac à dos, nous ajoutons les objets un par un. Chaque ajout d'un objet constitue une étape: à chaque étape, on doit choisir un objet à mettre dans le sac.

Principe : une méthode gloutonne fait, à chaque étape de la résolution du problème, le choix qui semble le plus pertinent **sur le moment. Ce choix n'est jamais remis en question.** On espère que cela nous conduira vers la meilleure solution du problème à résoudre.

Rem : on fait des choix **localement** optimaux dans l'espoir que ces choix mèneront à une solution **globalement** optimale. "Localement" signifie ici "à chaque étape de la résolution du problème".

Appliquons une méthode gloutonne à la résolution du problème du sac à dos :

- Sachant que l'on cherche à maximiser le gain, commençons par établir un tableau nous donnant la "valeur massique" de chaque objet (pour chaque objet on divise sa valeur par sa masse) :

objet	A	B	C	D	E
valeur massique					

- On classe ensuite les objets par ordre décroissant de valeur massique :

- Enfin, on remplit le sac en prenant les objets dans l'ordre et en s'arrêtant **dès que la masse limite** est atteinte. C'est ici que se fait "le choix glouton", à chaque étape, on prend l'objet ayant le rapport "valeur-masse" le plus intéressant au vu des objectifs :
  - 1re étape :
  - 2e étape :
  - 3e étape :
  - 4e étape :

Le sac est donc composé de …… objets : _____

_____

Cette méthode gloutonne peut être "automatisée", il est donc possible d'écrire un algorithme glouton afin de trouver une solution au problème du sac à dos avec n'importe quelles valeurs (nombre d'objets, masse des objets, valeur des objets, masse maximum).

❒ — Exercice **3** —

Algorithme glouton

- On se place dans un cas où les objets à dérober sont représentés dans un dictionnaire appelé **dicoObjet**. Chaque objet est représenté par une liste dont le premier élément est la poids et le second élément est la valeur marchande.

  Par exemple : **dicoObjet = { "Joconde" : [10, 1000000], "Livre" : [2, 22]}**

  dicoObjet["Joconde"] renvoie [10, 1000000]

  dicoObjet["Joconde"][0] renvoie 10
- Commencez par utiliser une liste d'objets correspondant à la partie I-1).
- En Python, implémenter l'algorithme glouton que vous testerez sur votre dictionnaire listeObjets avec des objets appelés A, B, C, D et E.

3) Problème d'optimisation

Ce genre de problème est un grand classique en informatique, on parle de problème **d'optimisation**. Il existe toujours de nombreuses solutions possibles à un problème d'optimisation. Dans le problème du sac à dos, on peut choisir A+B ou A+C ou B+C+D (ou même A!) : toutes les combinaisons sont possibles à partir du moment où la masse totale ne dépasse pas 30 kg.

Mais ici, **on ne cherche pas n'importe quelle solution, on cherche la solution optimale** : dans notre exemple, on cherche le plus grand gain possible.

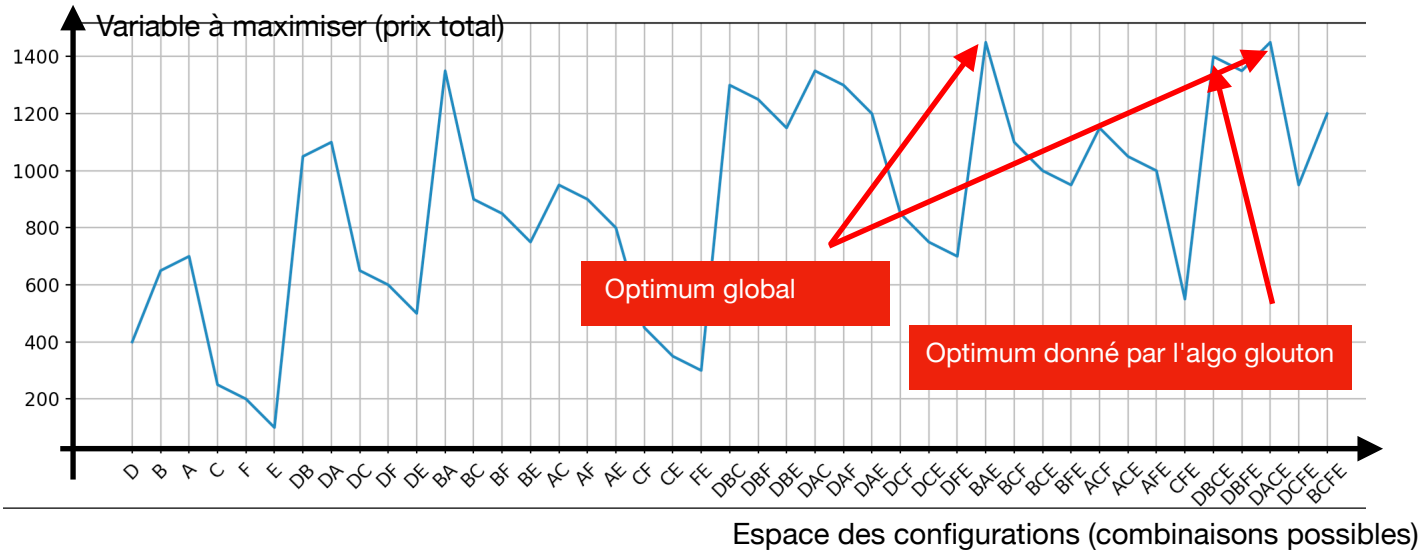
La méthode gloutonne nous permet-elle de faire cela ? Dis autrement : la solution trouvée dans le cas du sac à dos est-elle optimale ?

❒ — Exercice **4** —

- En jouant un peu avec les valeurs, pouvez-vous trouver une solution meilleure que celle proposée par l'algorithme glouton ?

Il est donc important de comprendre qu'un algorithme glouton ne donne pas forcément la solution optimale. Toutefois, on peut accepter que l'on n'ait pas la meilleure solution mais une des meilleures solutions : c'est ce que l'on appelle une **heuristique**.

On réduit nos exigences de précision afin de résoudre un problème compliqué de manière très rapide. On peut représenter la situation par le graphique ci-dessous :

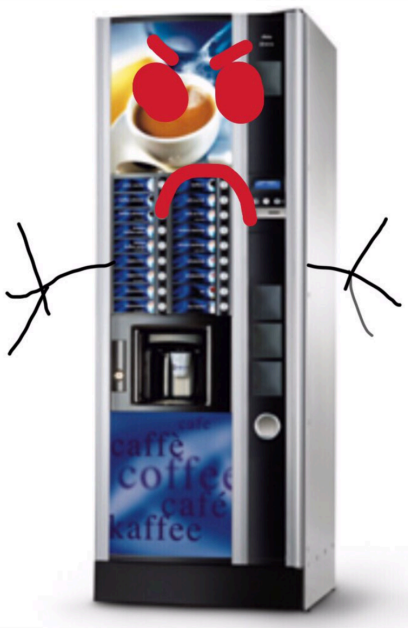


Espace des configurations (combinaisons possibles)

Il existe différentes méthodes algorithmiques permettant d'approcher une solution optimale à un problème d'optimisation : C'est un champ de recherche assez foisonnant où les chercheurs regorgent d'idées.

4) <u>Autre exemple : problème du rendu de monnaie</u>

Nous sommes un distributeur automatique (bip bip), nous avons à notre disposition un nombre **illimité** de pièces de :



- 1 centime
- 2 centimes
- 5 centimes
- 10 centimes
- 20 centimes
- 50 centimes
- 1 €
- 2 €

Nous devons rendre la monnaie à un client à l'aide de ces pièces. **La contrainte est d'utiliser le moins de pièces possible** (personne n'a envie d'avoir 251 pièces de 1 centime).

❐ — Exercice **5** —

- Trouvez une méthode gloutonne permettant d'effectuer un rendu de monnaie (en utilisant le moins possible de pièces).
- Vous devez rendre la somme de 2,63 €, appliquez la méthode que vous venez de mettre au point.
- Combien de pièces avez-vous utilisées ?

❐ — Exercice **6** —

- À partir de la méthode gloutonne que vous avez élaborée ci-dessus, écrivez un algorithme glouton qui permettra de déterminer le nombre minimal de pièces à utiliser pour une somme donnée.
- Vous proposerez ensuite une implémentation en Python de votre algorithme. Vous testerez votre programme avec une somme à rendre de 2 euros et 63 centimes.