Algorithmique - Partie 4

Les algorithmes gloutons

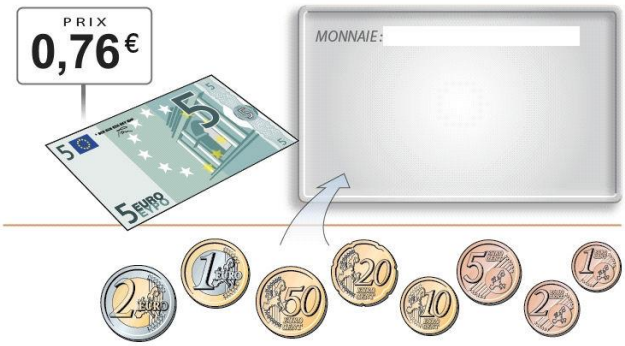
Objectifs pédagogiques :

✓ Comprendre la notion d'algorithme glouton
✓ Résoudre un problème grâce à un algorithme glouton.

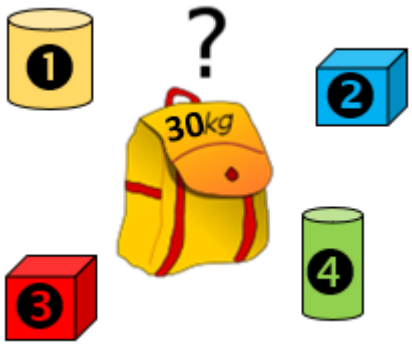
1. Optimisation d'un problème

Optimiser un problème, c'est déterminer les conditions dans lesquelles ce problème présente une caractéristique spécifique. Par exemple, dans le domaine des mathématiques, déterminer le minimum ou le maximum d'une fonction est un problème d'optimisation. Dans la vie de tous les jours, de nombreuses situations relèvent d'un problème d'optimisation :

- ✓ recherche du plus court chemin entre deux points géographiques par un GPS
- ✓ rendu de monnaie par un distributeur de boisson
- ✓ chargement d'un sac à dos …
- ✓ …



Problème du rendu de monnaie



Problème du sac à dos

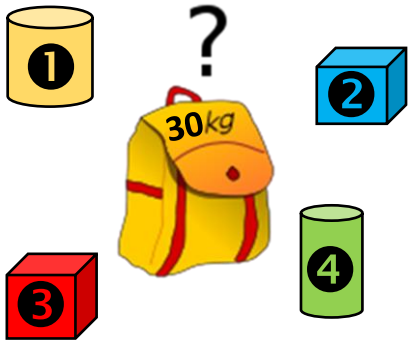
De nombreuses techniques informatiques sont susceptibles d'apporter une solution exacte ou approchée à ces problèmes. Certaines de ces techniques, comme l'énumération exhaustive de toutes les solutions, ont un coût machine qui les rend souvent peu pertinentes au regard de contraintes extérieures imposées (temps de réponse de la solution imposé, moyens machines limités).

Les **algorithmes gloutons** constituent une méthode possible de résolution de ce type de problème. Néanmoins, le résultat auquel ils conduisent n'est cependant pas toujours la solution optimale : il s'agit souvent d'un optimum possible mais pas de l'optimum. Plus précisément, ces algorithmes déterminent un optimum en effectuant successivement des choix locaux, jamais remis en cause. Au cours de la construction de la solution, l'algorithme résout une partie du problème puis se focalise ensuite sur le sous-problème restant à résoudre. Le principal avantage des algorithmes gloutons est leur facilité de mise en œuvre.

Une différence essentielle avec la **programmation dynamique**, plus efficace mais plus compliquée à implémenter, est que celle-ci peut remettre en cause des solutions déjà établies. Au lieu de se focaliser sur un seul sous-problème, elle explore les solutions de tous les sous-problèmes pour les combiner finalement de manière optimale.

2. Le problème du sac à dos

2.1. Situation problème



Le problème du sac à dos : quelles boîtes choisir afin de maximiser la somme emportée tout en ne dépassant pas les 30 kg autorisés ?

Richard Karp

En algorithmique, le **problème du sac à dos**, noté également **KP** (en anglais, *Knapsack problem*) est un problème d'optimisation combinatoire. Il modélise une situation analogue au remplissage d'un sac à dos, ne pouvant supporter plus d'une certaine masse, avec tout ou partie d'un ensemble donné d'objets ayant chacun une valeur spécifique en euros. Les objets mis dans le sac à dos doivent maximiser la valeur totale en euros, sans dépasser la masse maximale. Le problème du sac à dos est l'un des 21 problèmes NP-complets du mathématicien américain **Richard Karp**, exposés dans son article de 1972. Ce dernier reçu le prix Turing en 1985 pour ses travaux.

Objets	❶	❷	❸	❹
Valeurs v_i (€)	70	40	30	30
Masse m_i (kg)	13	12	8	10

2.2. Résolution exacte par la méthode de « force brute»

La solution naïve consiste à énumérer toutes les combinaisons possibles des objets puis choisir la solution optimale, c'est-à-dire celle qui maximise la valeur totale des objets emportés et dont la masse totale ne dépasse pas 30 kg. Elle conduit à coup sûr à l'optimum mais est gourmande en tant de calculs.

Q1. Recopier et compléter l'extrait de tableau ci-dessous rendant compte de toutes les combinaisons possibles des objets. On définit la variable x_i associée à un objet i de la façon suivante : $x_i = 1$ si l'objet i est mis dans le sac, et $x_i = 0$ si l'objet i n'est pas mis dans le sac.

Objets / Combinaisons	❶	❷	❸	❹	Valeur totale (€)	Masse totale (kg)
C1	0	0	0	0	0	0
C2						
C3						
C4						
…	…	…	…	…	…	…

Q2. De combien de manière peut-on combiner les 4 objets entre eux ?
Q3. En déduire la solution optimale au problème.
Q4. Quel est l'avantage de cette technique et son inconvénient ?

FOCUS : mise en équation mathématique du problème

La formulation mathématique du problème passe par une reformulation de ses données sous la forme d'expressions littérales. Dans notre cas, nous avons un sac à dos de **masse maximale M = 30 kg** et une collection de **n = 4 objets (**$n \in \mathbb{N}*$). Pour chaque objet **i**, nous avons une masse $\mathbf{m_i}$ et une valeur $\mathbf{v_i}$. On définit la variable x_i associée à un objet i de la façon suivante : $x_i = 1$ si l'objet i est mis dans le sac, et $x_i = 0$ si l'objet i n'est pas mis dans le sac.

Objectif 1 : respecter la contrainte de la masse totale → $\sum_{i=1}^{i=n} x_i \times m_i \leq M$

Objectif 2 : maximiser la valeur totale des objets → $max\left(\sum_{i=1}^{i=n} x_i \times v_i\right)$

Vérifions sur l'exemple suivant la validité des notations :

Objets	❶	❷	❸	❹	Valeur totale (€)	Masse totale (kg)
	0	1	1	1	0	0

Masse totale : $\sum_{i=1}^{i=n} x_i \times m_i = 0 \times 13 + 1 \times 12 + 1 \times 8 + 1 \times 10 = 30\ kg \leq M = 30\ kg$

⇨ La solution est réalisable, mais ce n'est pas nécessairement la meilleure solution.

Valeur totale : $\sum_{i=1}^{i=n} x_i \times v_i = 0 \times 70 + 1 \times 40 + 1 \times 30 + 1 \times 30 = 100\ €$

⇨ La valeur totale contenue dans le sac est égale à 100 €. Cette solution n'est pas la meilleure, car il existe une autre solution de valeur plus grande que 100 € : il faut prendre seulement les objets ❶ et ❷ qui donneront une valeur totale de 110 €. Il n'existe pas de meilleure solution que cette dernière, nous dirons alors que cette solution est optimale.

2.3. Résolution approchée par la méthode de l'algorithme glouton

Une méthode approchée a pour but de trouver une solution avec un bon compromis entre la qualité de la solution et le temps de calcul. Pour le problème du sac à dos, voici un exemple d'algorithme de ce type :

- calculer le rapport (v_i / m_i) pour chaque objet i ;
- trier tous les objets par ordre décroissant de cette valeur ;
- sélectionner les objets un à un dans l'ordre du tri et ajouter l'objet sélectionné dans le sac si le poids maximal reste respecté.

Effectuons le déroulé de cet algorithme sur notre exemple :

■ **Première étape :**

Objets	❶	❷	❸	❹
v_i (€)	70	40	30	30
m_i (kg)	13	12	8	10
v_i / m_i	5,4	3,3	3,7	3,0

■ **Deuxième étape :**

Objets	❶	❸	❷	❹
v_i (€)	70	30	40	30
m_i (kg)	13	8	12	10
v_i / m_i	5,4	3,7	3,3	3,0

■ **Troisième étape :**

Objet ❶ : on le met dans le sac vide, la masse du sac est alors de 13 kg ≤ 30 kg.

Objet ❸ : on le met dans le sac car 13 + 8 = 21 kg ≤ 30 kg.

Objet ❷ : on ne le met pas dans le sac car 21 + 12 = 33 kg ≥ 30 kg.

Objet ❹ : on ne le met pas dans le sac car 21 + 10 = 31 kg ≥ 30 kg.

La solution trouvée est donc de mettre les objets ❶ et ❸ dans le sac, ce qui donne une valeur de 100 € . Cette solution n'est pas optimale, puisqu'une solution avec une valeur totale de 110 € existe comme on l'a vu au paragraphe précédent. Néanmoins, cet algorithme reste rapide même si le nombre d'objets augmente considérablement.

Ce type d'algorithme est appelé algorithme glouton, car il ne remet jamais en cause une décision prise auparavant. Ici, lorsque l'objet ❷ ne peut pas entrer dans le sac, l'algorithme n'essaie pas d'enlever l'objet ❸ du sac pour y mettre l'objet ❷ à sa place.

Remarque :

Une heuristique est un terme donné à une classe d'algorithmes utilisées pour déterminer en temps réel une solution réalisable à un problème d'optimisation complexe. En optimisation combinatoire, théorie des graphes et théorie de la complexité, une heuristique est un algorithme qui fournit rapidement une solution réalisable, mais pas nécessairement optimale, pour un problème d'optimisation complexe. L'algorithme glouton est à ce titre un algorithme euristique.

Q5. Quel avantage principal revêt l'utilisation d'un algorithme glouton par rapport à une méthode par force brute ? Quel est son principal inconvénient ?
Q6. Pourquoi ce type d'algorithme est-il qualifié de glouton ?
Q7. Pourquoi l'algorithme glouton est-il qualifié d'heuristique ?
Q8. Pourquoi, selon vous, dans la méthode gloutonne, avoir classé les objets par valeurs de v_i / m_i décroissantes ?

FOCUS : l'analyse heuristique des antivirus

■ Notion d'analyse heuristique

L'**analyse heuristique** en temps réel est un argument « marketing » souvent mis en avant par les éditeurs d'antivirus. Cette méthode permet aux logiciels antivirus de détecter les nouveaux virus, ainsi que les nouvelles variantes d'un virus déjà connu. L'analyse heuristique est une méthode basée sur le comportement supposé d'un programme afin de déterminer si ce dernier est ou non un virus. Cette méthode se différencie de l'analyse statistique qui se base quant à elle sur des comparaisons du programme avec des virus connus référencés dans une bibliothèque du logiciel antivirus.

Les analyses heuristiques appliquées aux antivirus sont... l'art de la divination, de la supposition, de l'hypothèse, de l'incertitude... C'est d'ailleurs assez surprenant dans un monde fait de certitudes, de zéros ou de uns, de portes logiques ouvertes ou fermées mais jamais d'à-peu-près, de plus ou moins, d'approximations...

■ Les hypothèses heuristiques

L'hypothèse heuristique doit permettre de produire une idée directrice de recherche et, par mesure de prudence, une alarme, indépendamment de la vérité absolue.

Les analyses heuristiques (ou algorithmes heuristiques, méthodes heuristiques etc. ...) d'un objet (un fichier, un flux...) servent à aider à la découverte de nouveaux parasites (virus...) par la recherche de "faits" (actions conduites par le code du programme analysé et, mieux encore, combinaisons d'actions) qui :

- ✓ dans des cas qui se sont présentés antérieurement, ont généralement conduit à l'affirmation que l'objet était ou contenait un parasite ;
- ✓ habituellement, ne se produisent pas.

Les "faits" recherchés sont, par exemples, des bouts de code dont les actions trouvées simultanément dans un même objet le rende suspicieux, sans pouvoir affirmer qu'il s'agit réellement d'un parasite.

Par exemple, le cryptage du code peut être considéré comme un élément de suspicion. Les méthodes heuristiques utilisent un système de "poids" (de "notation") des suspicions et la somme de ces notes, pondérées par leurs interactions, fait basculer, en fin d'analyse, l'objet en suffisamment "suspicieux" pour alerter l'utilisateur ou "inoffensif".

Les analyses heuristiques servent donc à écarter ou appuyer un doute. Tout le problème de ces outils d'intelligence artificielle qui intègrent sans arrêt les résultats précédents (ces outils sont en perpétuelles phases d'auto-apprentissage) et donc évoluent sans cesse, est le poids qu'ils accordent à leurs suspicions - à partir de quand leur susceptibilité leur fait déclarer suspicieux un objet complètement anodin (et donc leur fait produire une fausse alarme - un faux positif) ou l'inverse (absence de production d'une alarme sur un objet hostile - faux négatif).

Les analyses heuristiques permettent, plus ou moins, de dire qu'un objet analysé embarque ou non les conditions de comportement pour qu'il soit un objet hostile mais ne permettent pas d'affirmer qu'il s'agit d'un objet hostile ou inoffensif.

■ **Principe de fonctionnement**
Les logiciels antivirus exécutent en temps réel le code ou le script de fichier à analyser dans un environnement virtuel, tout en analysant les instructions du programme. Cela permet de connaître le comportement du programme tout en isolant le code du fichier suspect de la machine réelle. Si l'antivirus détecte des instructions suspectes comme la suppression de fichiers ou le lancement de processus multiples, le fichier sera reconnu comme un virus et l'utilisateur sera alerté. Une autre technique consiste à décompiler le programme en question et analyser son code source. Le code est comparé aux codes d'autres virus connus. Si une grande partie du code se retrouve dans d'autres virus, le programme sera reconnu comme une menace.

■ **Fiabilité de la prédictibilité**
Bien que l'analyse heuristique permette de détecter de nouveaux virus et les nouvelles variantes de virus préexistants, l'efficacité est vraiment faible au regard du nombre de faux positifs. C'est parce que les virus informatiques, tout comme les virus biologiques, changent constamment et évoluent. Comme l'analyse heuristique repose sur la comparaison du fichier suspect avec les autres virus déjà connus, il est fréquent qu'elle rate certains virus qui contiennent de nouveaux codes ou de nouvelles méthodes de fonctionnement.

Les analyses heuristiques, très complexes, ont permis de détecter, dans le meilleur des cas, de 40 à 60% de parasites dans des lots de parasites totalement nouveaux (inconnus des bases de signatures). C'est peu et beaucoup à la fois. Une suspicion soulevée par une méthode heuristique doit être suivie d'une analyse approfondie de l'objet suspicieux par des méthodes produisant des certitudes et seul l'être humain est capable de le faire, même s'il s'appuie de plus en plus sur des outils plus avancés que les méthodes heuristiques, comme les sandbox et les machines virtuelles.

Q9. Quel est le principe de l'analyse heuristique pour un antivirus ?
Q10. Cette méthode permet-elle de détecter 100 % des virus ?
Q11. Citer deux inconvénients à cette méthode.

2.3. Résolution exacte par la méthode de séparation et évaluation (PSE)

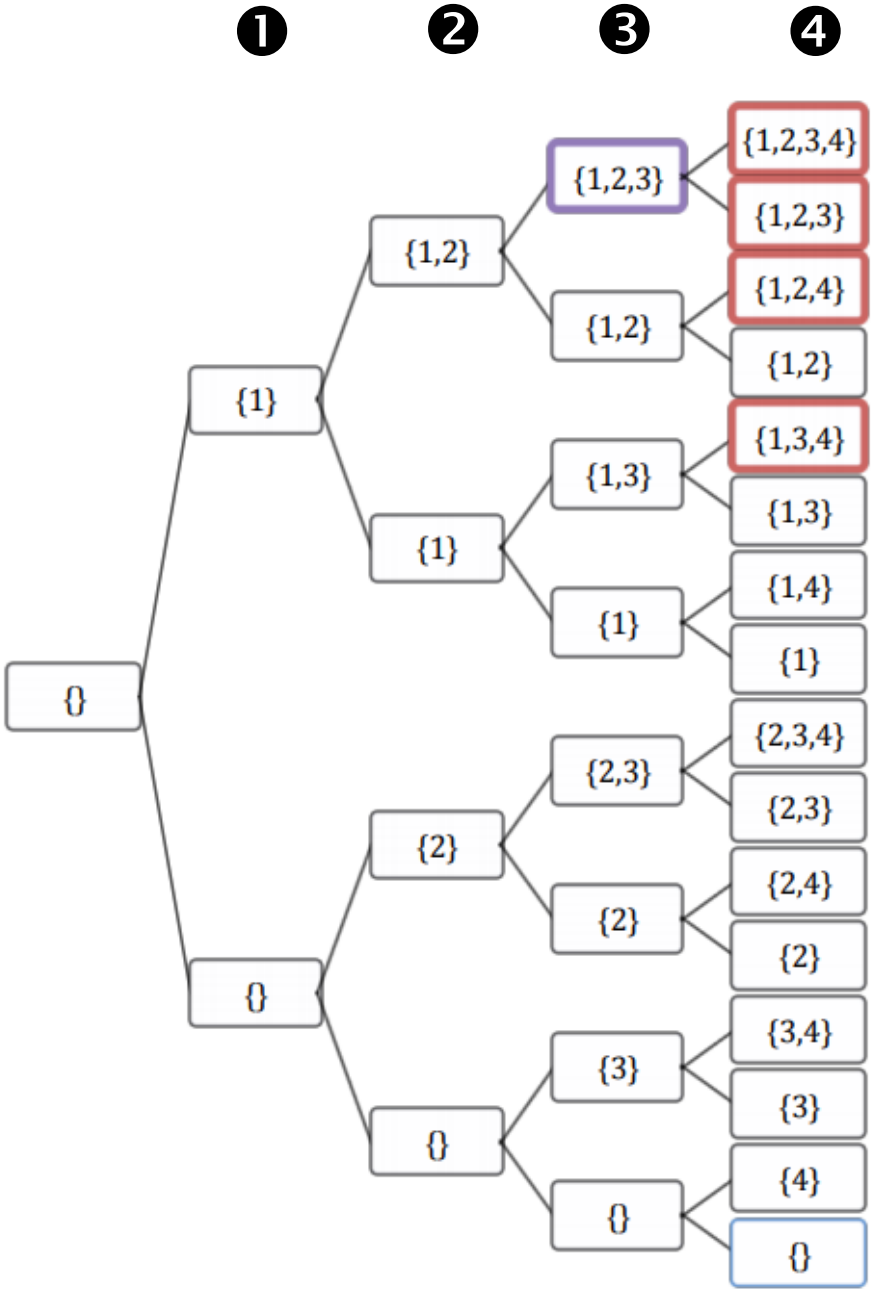
Pour trouver la solution optimale, et être certain qu'il n'y a pas mieux, il faut utiliser une méthode exacte, qui demande un temps de calcul beaucoup plus long (si le problème est difficile à résoudre). Il n'existe pas une méthode exacte universellement plus rapide que toutes les autres. Chaque problème possède des méthodes mieux adaptées que d'autres. Nous allons présenter un exemple d'algorithme de ce type, nommé **procédure par séparation et évaluation** (**PSE**), ou en anglais **Branch and Bound (BAB)**. Nous n'exposerons ici qu'une version simplifiée d'une **PSE**.

Une PSE est un algorithme qui permet d'énumérer intelligemment toutes les solutions possibles. En pratique, seules les solutions potentiellement de bonne qualité seront énumérées, les solutions ne pouvant pas conduire à améliorer la solution courante ne sont pas explorées.

Pour représenter une PSE, nous utilisons un « **arbre de recherche** » constitué :

- ✓ de nœuds ou sommets, où un nœud représente une étape de construction de la solution
- ✓ d'arcs pour indiquer certains choix faits pour construire la solution.

Dans notre exemple, les nœuds représentent une étape pour laquelle des objets auront été mis dans le sac, d'autres auront été laissés en dehors du sac, et d'autres, encore, pour lesquels aucune décision n'aura encore été prise. Chaque arc indique l'action de mettre un objet dans le sac ou, au contraire, de ne pas le mettre dans le sac. La figure suivante représente l'arbre de recherche du problème donné en exemple.



On associe l'ensemble vide à la racine (dont la profondeur est 0) et pour un nœud de profondeur i-1, on construit deux fils : celui du haut où l'on ajoute l'objet i dans le sac et celui du bas où le sac reste tel quel. L'arbre de recherche achevé, chaque feuille représente alors une solution potentielle mais pas forcément réalisable. Dans le schéma, les feuilles au bord épais représentent les propositions irréalisables car supérieures au poids maximal à ne pas dépasser. Pour déterminer la solution, il suffit de calculer la valeur du sac pour chaque nœud feuille acceptable et de prendre la solution ayant la plus grande valeur.

Cependant la taille de l'arbre de recherche est exponentielle en le nombre d'objets. Aussi il existe de nombreuses techniques algorithmiques de parcours de ce type d'arbre. Ces techniques ont pour but de diminuer la taille de l'arbre et d'augmenter la rapidité du calcul. Par exemple, on peut remarquer que le poids du nœud interne {1,2,3} dépasse déjà le poids maximal, il n'était donc pas nécessaire de développer l'étape suivante avec l'objet 4. Les procédures par séparation et évaluation (PSE) permettent d'élaguer encore plus cet arbre en utilisant des bornes inférieures et supérieures de la fonction objectif :

- ✓ Une borne inférieure est une valeur minimum de la fonction objectif. Autrement dit, c'est une valeur qui est nécessairement inférieure à la valeur de la meilleure solution possible. Dans notre cas, une configuration du sac réalisable quelconque fournit une borne inférieure.
- ✓ Une borne supérieure est une valeur maximale de la fonction objectif. Autrement dit, nous savons que la meilleure solution a nécessairement une valeur plus petite. Dans notre cas, nous savons que la valeur de la meilleure solution est nécessairement inférieure à la somme des valeurs de tous les objets (comme si on pouvait tous les mettre dans le sac).

Supposons maintenant que la borne inférieure soit initialisée par l'algorithme heuristique vu précédemment. Pendant la recherche à chaque nœud, nous pouvons déterminer une borne supérieure : la somme de toutes les valeurs de tous les objets déjà mis dans le sac plus la somme des valeurs des objets restants dont on ne sait pas encore s'ils seront dans le sac. À partir d'un nœud et de sa borne supérieure, nous savons que les solutions descendantes de ce nœud ne pourront pas avoir une valeur plus grande que cette borne supérieure. Si jamais, à un nœud donné, la valeur de la borne supérieure est inférieure à la valeur de la borne inférieure, alors il est inutile d'explorer les nœuds descendants de celui-ci. On dit qu'on coupe l'arbre de recherche. En effet, si à partir d'un nœud, nous savons que nous ne pourrons pas faire plus de 10 (borne supérieure calculée) et que la borne inférieure existante est à 11 (on a déjà une solution de valeur 11), alors les solutions descendantes de ce nœud ne sont pas intéressantes. Enfin, dernière remarque, la valeur de la borne inférieure peut être actualisée lorsqu'est trouvée une solution réalisable qui possède une valeur plus grande.

Ce système de calcul de bornes demande un faible coût de temps de calcul, et permet d'augmenter la rapidité de la PSE puisqu'elle coupe des branches d'arbre pour ne pas perdre de temps à les explorer. D'autres techniques servent à diminuer la taille de l'arbre et à augmenter la rapidité. Par exemple, elles sont basées sur l'ordre dans lequel on prend les décisions sur les objets, ou sur une évaluation à chaque nœud, ou encore sur des propriétés du problème qui permettent de déduire des conclusions sur certains objets. Sur notre exemple initial, il est possible d'obtenir de façon exacte la solution optimale en n'examinant que 9 nœuds au lieu de 31.

Q12. Quel outil utilise-t-on pour mettre en œuvre la méthode exacte de résolution d'un problème par séparation et évaluation (PSE) ?
Q13. Quel est le principe de la PSE ?
Q14. Pourquoi, selon vous, peut-on dire que la PSE est un bon compromis entre la force brute et la méthode gloutonne ?

3. Le problème du rendu de monnaie

3.1. Situation problème

Le **problème du rendu de monnaie** est un problème d'algorithmique. Il s'énonce de la façon suivante : étant donné un système de monnaie (pièces et billets), comment rendre une somme donnée de façon optimale, c'est-à-dire avec le nombre minimal de pièces et billets ?
Dans la zone euro, le système de pièces et de billets en vigueur est, en mettant de côté les centimes d'euros : **{1, 2, 5, 10, 20, 50, 100, 200 €}**

Hypothèses : on dispose d'autant d'exemplaires qu'on souhaite de chaque pièce et billet. Par la suite nous désignerons par monnaie, indifféremment une pièce ou un billet.

3.2. Résolution exacte par la méthode de « force brute »

Q15. Hugo veut acheter une nouvelle souris de gamer pour son PC. Celle-ci coûte 53 euros. A la caisse automatique d'une grande surface qui accepte le paiement en monnaie, il paye avec un billet de 100 euros. La caisse automatique doit lui rendre 47 euros. Indiquer quelques-unes des combinaisons possibles de pièces et / ou de billets permettant de le faire. Est-il facile de toutes les dénombrer ?
Q16. Parmi toutes les combinaisons précédentes possibles, quelle est, selon vous, celle permettant de minimiser le nombre de pièces et / ou de billets utilisés ? Cette solution sera qualifiée d'optimale.
Q17. Citer un avantage et un inconvénient à cette méthode.

3.3. Résolution approchée par la méthode de l'algorithme glouton

Utiliser un algorithme glouton consiste à optimiser la résolution d'un problème avec contrainte en utilisant l'approche suivante : on procède étape, par étape, en faisant à chaque étape, le choix qui semble être le meilleur, sans jamais remettre en question les choix précédents. Dans de nombreuses situations de la vie quotidienne, nous employons intuitivement, sans le savoir, des algorithmes gloutons. Le problème du rendu de monnaie en est une illustration typique.

Q18. Écrire en langage naturel l'algorithme glouton permettant de résoudre le problème du rendu de monnaie.
Q19. Appliquer l'algorithme glouton au problème de rendu de monnaie de Hugo (cf **Q15**).

L'algorithme glouton permet de trouver une solution optimale locale au problème, mais pas toujours une solution optimale globale.

Q20. Supposons une caisse automatique devant rendre une somme de 63 euros. Nous formulons en outre comme hypothèse qu'elle ne dispose plus de billets de 5 et 10 euros. Appliquer l'algorithme glouton pour résoudre le problème.
Q21. La solution précédente est-elle la solution optimale globale ? Si tel n'est pas le cas, quelle est-elle ?
Q22. En déduire quelle hypothèse formulée au départ influence fortement l'efficacité de la méthode gloutonne.

FOCUS : qu'est qu'un système canonique pour un algorithme glouton ?

Rendre par exemple 49 € avec un minimum de pièces est un problème d'optimisation. En pratique, sans s'en rendre compte généralement, tout individu met en œuvre un algorithme glouton. Il choisit d'abord la plus grandeur valeur de monnaie, parmi 1, 2, 5, 10, contenue dans 49 €. En l'occurrence, quatre fois une pièce de 10 €. La somme de 9 € restant à rendre, il choisit une pièce de 5 €, puis deux pièces de 2 €. Cette stratégie gagnante pour la somme de 49 € l'est-elle pour n'importe quelle somme à rendre ? On peut montrer que la réponse est positive pour le système monétaire européen. Pour cette raison, un tel système de monnaie est qualifié de canonique. D'autres systèmes ne sont pas canoniques. L'algorithme glouton ne répond alors pas de manière optimale. Par exemple, avec le système {1, 3, 6, 12, 24, 30}, l'algorithme glouton répond en proposant le rendu 49 € = 30 + 12 + 6 + 1, soit 4 pièces alors que la solution optimale est 49 € = 2 × 24 + 1, soit 3 pièces. La réponse à cette difficulté passe par la programmation dynamique, thème abordé en spécialité NSI de classe de terminale.

4. Implémentation en langage Python

4.1. Algorithme glouton pour le problème du sac à dos

Dans un IDE PYTHON (JupyterNoteBook, Edupython ou Spyder), exécuter le programme suivant :

```
# -*- coding: utf-8 -*-

"""
Mise en oeuvre d'un algorithme glouton pour le sac à dos
"""
# Masse maximum pouvant être emportée dans le sac
masse_max_sac = 30

# Définition du système d'objets : liste de sous-listes
# Sous-liste : ["Nom_objet",valeur_objet,masse_objet,valeur_objet/masse_objet]
systeme_objets=[["Objet1",70,13,70/13],["Objet2",40,12,40/12],["Objet3",30,8,
30/8],["Objet4",30,10,30/10]]

# Tri du système d'objets par valeurs décroissantes de
(valeur_objet/masse_objet)
# Utilisation de key et d'une fonction lambda pour faire porter le tri sur le
# dernier élément de chaque sous-liste
systeme_objets.sort(key=lambda colonne:colonne[-1],reverse = True)

# Définition de la fonction gloutonne
def sac_a_dos(masse_max_sac, systeme_objets):
    '''Fonction gloutonne'''
    # Initialisation de la masse temporaire
    masse = 0
    # Initialisation de la liste d'objets à sélectionner
    liste_objets = []
    for i in range(len(systeme_objets)):
        if masse + systeme_objets[i][-2] <= masse_max_sac:
            masse = masse + systeme_objets[i][-2]
            liste_objets.append(systeme_objets[i])
    return liste_objets

print(sac_a_dos(30, systeme_objets))
```

Affichage : [['Objet1', 70, 13, 5.384615384615385], ['Objet3', 30, 8, 3.75]]

L'idée à suivre, si on veut développer une méthode gloutonne, est d'ajouter en premier les objets dont le quotient (valeur / masse) est élevé, jusqu'à saturation du sac. Cette méthode est parfois efficace, mais parfois pas : prenons deux exemples, afin d'illustrer cela.

Exemple 1 : supposons qu'on dispose d'un sac de capacité maximale 26 kg et du « set » d'objets suivant :

Objets	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Valeurs vi (€)	4	3	8	5	10	7	1	7	3	3	6	12	2	4
Masse mi (kg)	2	2	5	2	7	4	1	4	2	1	4	10	2	1

Q23. Appliquer le programme glouton pour rechercher une solution. Celle-ci est-elle optimale ?

Exemple 2 : supposons maintenant que l'on dispose d'un sac de capacité maximale 40 kg et du « set » d'objets suivant :

Objets	A	B	C	D	E	F
Valeurs vi (€)	30	12	12	12	12	4
Masse mi (kg)	39	10	10	10	10	1

Q24. Appliquer le programme glouton pour rechercher une solution. Celle-ci est-elle optimale ?
Q25. À la lumière des résultats des deux exemples précédents, selon vous, l'algorithme glouton donne-t-il de bons résultats lorsque les masses des différents objets sont relativement proches les unes des autres ou si elles sont très déséquilibrées ?
Q26. Proposer des améliorations possibles pour rendre le programme glouton plus ergonomique.

4.2. Algorithme glouton pour le problème du rendu de monnaie

Dans un IDE PYTHON (JupyterNoteBook, Edupython ou Spyder), exécuter le programme suivant :

```
# -*- coding: utf-8 -*-

"""
Mise en oeuvre d'un algorithme glouton pour le rendu de monnaie
"""
# Somme devant être rendue
somme_a_rendre = 69

# Définition du système de monnaie (liste)
# Hypothèse : pièces et billets disponibles en nombre infini
systeme_monnaie = [1,2,5,10,20,50,100]

# Initialisation de la liste de la monnaie (pièces + billets) à rendre
liste_monnaie_rendue = []

# Indice de la première monnaie à comparer à la somme à rendre
# On commence par la monnaie de plus grande valeur (dernier élément de la liste)
i = len(systeme_monnaie) - 1

# Définition de la fonction gloutonne
def monnaie_a_rendre(somme_a_rendre, systeme_monnaie):
    '''Fonction gloutonne'''
    # Liste de la monnaie à rendre
    liste_monnaie_rendue = []
    # Indice de la première monnaie à comparer à la somme à rendre
    i = len(systeme_monnaie) - 1
    while somme_a_rendre > 0:
        valeur = systeme_monnaie[i]
        if somme_a_rendre < valeur:
            i = i - 1
        else:
            liste_monnaie_rendue.append(valeur)
            somme_a_rendre = somme_a_rendre - valeur
    return liste_monnaie_rendue

# Programme principal (appel de la fonction)
print(monnaie_a_rendre(somme_a_rendre,systeme_monnaie))
```

Affichage : [50, 10, 5, 2, 2]

Q27. Testez le programme glouton dans plusieurs cas de figure. La solution proposée est-elle toujours optimale ? Donnez quelques exemples.

5. Pour conclure …

Comment faire pour obtenir une solution optimale à un problème ?

Il n'y a pas de réponses simples à cette question, tout dépend du problème. Dans certains cas, les algorithmes gloutons produisent d'excellents résultats et sont appropriés au problème, dans d'autres cas, ils ne le sont pas…Généralement, si les « poids » respectifs des objets sont très déséquilibrés, les algorithmes gloutons produiront une solution non optimale car de tels algorithmes ont une mauvaise « vision globale » du problème.

Les exemples qui ont été traités ici peuvent très bien être résolus à l'aide de la programmation dynamique ou à l'aide du paradigme de programmation « divide-and-conquer » (diviser pour régner) que nous verrons en terminale spé NSI. On aurait pu également utiliser une solution de brute-force, mais ce genre de solutions est à éviter en raison de leur très mauvaise complexité algorithmique (temps de calculs très longs).

Il convient donc, avant toute chose, de réfléchir au type d'algorithme à adopter en fonction de la nature du problème à résoudre !

Alan Turing est un mathématicien et cryptologue britannique, auteur de travaux qui fondent scientifiquement l'informatique. En homage à ces travaux, depuis 1966, l'Association for Computing Machinery décerne annuellement le prix Turing ("Turing Award") à des personnes ayant apporté une "contribution majeure d'importance durable" au domaine de la recherche informatique. Cette récompense est souvent considérée comme l'équivalent du prix Nobel pour l'informatique