Optimisation de problèmes : Algorithmes Gloutons

Définitions:

Les algorithmes gloutons répondent au problème d'optimisation de problèmes. Optimiser un problème revient à trouver la meilleure solution à un problème en utilisant un minimum de ressources en fonction de certains critères : une solution optimale.

Il existe plusieurs manières de trouver des solutions dites optimales :

- Par Bruteforce : On cherche toutes les solutions possibles, ce qui requiert beaucoup de temps.
- Par Gloutons : On cherche à obtenir la meilleure solution à chaque étape du programme, c'est à dire des meilleures solutions locales.

On s'interessera ici aux algorithmes Gloutons qui ont pour principal avantage leur facilité de mise en place et quelques problèmes qui les implémente.

Il existes divers problèmes pouvant être résolus à l'aide d'algorithmes gloutons :

- Problème de rendu de monnaie
- Problème du sac à dos
- Problème du choix d'activité
- Problème du voyageur de commerce

Le problème du rendu de monnaie

Le problème du rendu de monnaie est un problème simple : On cherche à rendre une certaine somme en limitant le nombre de pièce (ou billet) à rendre.

C'est un problème pouvant être résolu par un algorithme glouton : on cherche une solution globalement satisfaisante en fonction d'une contrainte précise : le nombre de billet ou pièces à rendre.

On dispose d'un système monétaire :

1	2	5	10	20	50	100	200	500

Et on veut rendre de la monnaie en fonction du système proposé, prenons l'exemple de 42€

Étapes	Liste de monnaies rendues	Somme restante à vendre
Initialisation	monnaie = [ ]	Monnaie_restante = 42
Étape 1	monnaie = [20]	Monnaie_restante = 22
Étape 2	monnaie = [20, 20]	Monnaie_restante = 2
Étape 3	Monnaie = [20, 20, 2]	Monnaie_restante = 0

Exercice :

Sur papier, réaliser l'algorithme de rendu de monnaie avec les systèmes proposés en détaillant chaque étapes :

1. Système 1 = {1,2,5,10,20}, monnaie à rendre = 84
2. Système 2 = {1,3,6,12,24,30}, monnaie à rendre = 53
3. Ajouter une pièce de 3 euros dans le Système 1. Que peut on déduire ?

Les solutions données sont-elles toujours les bonnes ?

On appelle **système canonique**, un système dans lequel un algorithme glouton donne la solution globalement optimale.

Le problème du sac à dos

Le problème du sac à dos consiste à trouver la combinaison optimale d'objets à placer dans un sac à dos, en maximisant la valeur totale des objets tout en respectant la capacité maximale du sac. Les algorithmes gloutons tentent de résoudre ce problème en choisissant à chaque étape l'objet le plus "rentable" en termes de valeur par unité de poids.

Il existe différentes manières de remplir ce sac à dos :

- Favoriser le poids : mettre le plus d'éléments de plus petit poids.
- Favoriser la valeur : mettre le plus d'élément de grande valeur.
- Prendre le plus d'objets ayant les meilleurs rapports valeur / masse.

Exercice :

Sur papier, réaliser l'algorithme du problème du sac à dos avec les capacités de sacs et objets proposés en prenant en compte les 3 façons de remplir le sac.

1. On considère un sac à dos d'une capacité de 30kg et les objets suivants :

Objet	1	2	3	4
Masse	5	13	6	10
Valeur	7	15	14	5

2. On considère un sac à dos d'une capacité de 14kg et les objets suivant

Objet	1	2	3	4
Masse	4	11	5	9
Valeur	7	15	10	9

Travail Pratique : Implémentation des problèmes :

Exercice 1 : Le problème du rendu de monnaie

Un distributeur automatique de snacks doit rendre la monnaie aux clients lorsqu'ils paient en espèces. Écrivez un algorithme qui, étant donné un montant à rendre et une liste des valeurs des pièces et billets disponibles, détermine la façon optimale de rendre la monnaie en utilisant le moins de pièces et de billets possible.

Votre algorithme devrait prendre en compte les contraintes suivantes :

- Vous disposez d'un nombre illimité de chaque type de pièce et billet.
- Vous devez utiliser les valeurs les plus élevées de pièces et billets disponibles en priorité, afin de minimiser le nombre total de pièces et de billets rendus.

Votre algorithme devrait retourner la liste des pièces et billets à rendre, ainsi que le nombre total de pièces et de billets rendus.

Exercice 2 : Le problème du sac à dos

Réaliser une fonction *sac_a_dos_1* qui prendra en paramètre un sac à dos (on le définira par une liste) et une liste d'objets et renvoie la solution proposée en privilégiant la masse des objets. On considère la liste d'objets : ***objets = [(2, 5), (3, 8), (4, 10), (5, 7), (6, 3), (7, 6)]***.

Réaliser une fonction *sac_a_dos_2* qui prendra en paramètre un sac à dos et une liste d'objets et renvoie la solution proposée en priviégiant la valeur des objets. On considère la lisete : d'objets : ***objets = [(2, 4), (3, 6), (4, 8), (5, 2), (6, 7), (7, 9)]***

Réaliser une fonction *sac_a_dos_3* qui prendra en paramètre un sac à dos et une liste d'objet et renvoie la solution proposée en privilégiant le rapport valeur/masse des objets. On considère la liste d'objets : ***objets_2 = [(3, 9), (5, 3), (6, 5), (7, 4), (8, 7), (9, 2)]***

Exercice 3 : La plannification d'activités

Vous êtes responsable de la planification d'un événement qui comporte un ensemble d'activités avec des horaires de début et de fin différents. Vous devez sélectionner un sous-ensemble d'activités qui ne se chevauchent pas, de manière à maximiser le nombre total d'activités que vous pouvez inclure dans l'événement.

Écrivez un algorithme appelé `choix_activites` qui prendra en paramètre une liste d'activités représentées par des tuples, où chaque tuple contient l'horaire de début et l'horaire de fin d'une activité. L'algorithme devra retourner le sous-ensemble d'activités qui maximise le nombre d'activités sans chevauchement.

Vous devrez implémenter une stratégie gloutonne pour résoudre ce problème. L'idée est de trier les activités par ordre croissant d'horaire de fin.