CentraleSupélec

Algorithmique et complexité

TD 7-8/8 – Résolution exacte de problèmes NP-difficiles

L'objectif de ce TD est de résoudre un problème NP-difficile.

Exercice 1 : Problème du Sac à dos

Considérons trois différents problèmes pratiques.

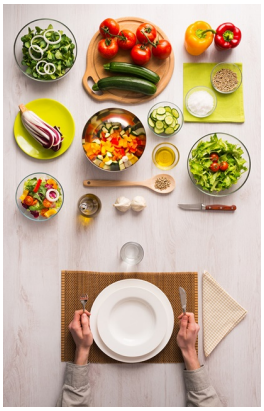
Bagage de cabine

Vous vivez une vie nomade, voyageant tout le temps autour du monde. Vous avez un ensemble d'éléments auxquels vous tenez plus ou moins selon vos besoins, par exemple votre ordinateur portable est quelque chose de très important, il a donc une grande valeur. Au contraire, vous aimez votre t-shirt SF, mais vous pouvez toujours en acheter un à destination, alors votre t-shirt SF a une petite valeur. Vous ne transportez que des bagages de cabine. Différentes compagnies aériennes imposent des limites différentes au poids des bagages à main, par exemple 7kg ou 10kg. Comment maximiser la valeur des articles que vous pouvez embarquer avec vous, tout en restant en dessous du poids maximum autorisé ?

Cambriolage

Un voleur parvient à pénétrer chez un bijoutier. Il y a beaucoup d'objets qu'il peut voler : montres, bagues, colliers ... Tous ces éléments ont des valeurs et des poids différents. Le voleur ne peut transporter qu'un poids limité. Supposons que les articles peuvent être divisés en catégories, chaque catégorie ayant un poids, une valeur et un certain nombre d'éléments disponibles. Comment le voleur devrait-il choisir quoi prendre ?

Repas équilibré



Vous recevez une liste de valeurs nutritionnelles pour une portion de chaque élément servi dans le menu quotidien de votre restaurant universitaire préféré. Pour chaque élément, vous obtenez le nombre de calories (ou poids) et la valeur protéique d'une partie de l'article. Qu'est-ce que vous avez choisi de manger pour maximiser la prise de protéines sans dépasser une certaine quantité de calories ?

Complexité

Question 1

Les trois problèmes peuvent être formalisés en utilisant le même format. Donnez une description formelle de ce problème : quelles sont les entrées, quelle est la question du problème de décision correspondant, quelle est la question du problème d'optimisation ?

Question 2

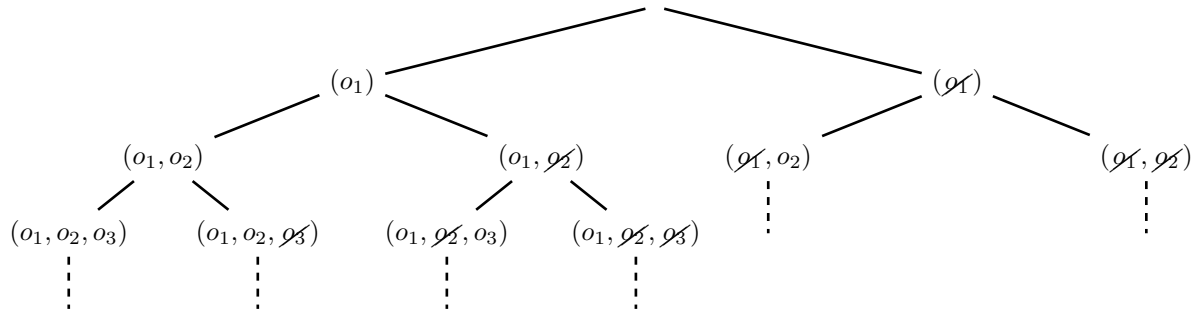
Montrez que ce problème (connu comme **KNAPSACK problem**) est NP-complet. Vous pouvez utiliser un problème NP-complet classique SUBSET-SUM :

SUBSET SUM
Entrée :
— un ensemble A d'entiers positifs $a_1, \ldots, a_n$
— une valeur $t \in \mathbb{N}$

Question : Existe-il un sous-ensemble $S \subseteq [1, n]$ dont la somme satisfait $\sum_{i \in S} a_i = t$?

Backtracking

On considère l'algorithme de **backtracking** qui décide à chaque étape la présence ou l'absence d'un objet dans la solution partielle courante. On construit ainsi l'espace de solutions qui prend la forme d'un arbre binaire de hauteur n :



L'algorithme de *backtracking* va énumérer toutes les solutions possibles pour trouver la meilleure (ici ayant la plus grande valeur).

L'algorithme de *backtracking* évitera d'explorer une branche si elle dépasse déjà la limite de poids.

Question 3

Écrivez en Python le code de cet algorithme. Pour vous aider, rendez-vous sur la page de pratique du TD :

https://wdi.centralesupelec.fr/1CC2000/TD8ProgEn

Branch & Bound

Un algorithme **branch and bound** améliore le *backtracking* en évitant d'énumérer toutes les configurations possibles. Il analyse les propriétés du problème et combine deux principes :
— une stratégie de **séparation** (branching) pour que l'algorithme rencontre la solution optimale le plus tôt possible.
— une stratégie de **coupe** pour éviter d'explorer des branches qui ne peuvent pas conduire à une meilleure solution.

Question 4

Proposez une stratégie de séparation pour que l'algorithme rencontre la solution optimale le plus tôt possible lors de l'exploration de l'espace des solutions.

Question 5

Nous avons besoin d'un critère pour arrêter l'exploration de la branche courante si elle ne peut pas mener à une meilleure solution que celle déjà trouvée. L'algorithme prend en paramètre :
— v la valeur de la meilleure solution trouvée jusqu'ici,
— $I_p \subset [1, n]$ les indices des objets sélectionnés comme présents avec $\sum_{i \in I_p} w_i \leq B$,
— $I_a \subset [1, n]$ les indices des objets sélectionnés comme absents et qui ne feront pas partie de la solution en cours de construction.

Donnez la condition à implémenter ainsi que la fonction *bound* correspondante qui permet d'estimer une borne sur les meilleures solutions vers lesquelles une solution partielle peut mener.

Question 6

Retournez sur la page de pratique du TD. En partant du code de l'algorithme de *backtracking*, réalisez un *branch & bound*. Un *benchmark* est fourni pour comparer le *backtracking* et le *branch & bound* sur de grandes instances aléatoires.

Programmation dynamique

Nous voulons proposer un algorithme de programmation dynamique pour résoudre le problème du sac à dos. On note $V(i, j)$ la valeur totale maximale qu'on peut mettre dans un sac à dos de taille j en embarquant que des objets parmi $o_1, \ldots, o_i$.

Question 7

Quelle est la formule de récursion qui calcule $V(i, j)$ pour tout i et j dans $\mathbb{N}$?

Question 8

Quelle est la complexité en temps et en espace d'un algorithme de programmation dynamique implémentant la formule précédente ?

Question 9

Cela signifie-t-il que $P = NP$?

Question 10

Retournez sur la page de pratique du TD. Écrivez en Python puis tester l'algorithme de résolution basé sur la programmation dynamique.