Université Larbi Ben Mhidi — Niveau : 1ère année Master AD

Département MI — Module : Méthodes algorithmiques

Année : 2024-2025

Solution Série de TD 4 : La programmation

Solution Exercice 01 :

```
def rendu_monnaie(somme, pieces):

    dp = [float('inf')] * (somme + 1) // Initialise avec l'infini pour indiquer qu'il n'y a pas de
solution par défaut
    track = [-1] * (somme + 1)         // Pour suivre les pièces utilisées
    dp[0] = 0


    for j in range(1, somme + 1):
        for piece in sorted(pieces, reverse=True):  //Trier les pièces pour privilégier les valeurs
plus grandes
            if j >= piece and dp[j - piece] + 1 < dp[j]:
                dp[j] = dp[j - piece] + 1
                track[j] = piece

    if dp[somme] == float('inf'):
        return None, None  // Aucun moyen de rendre la somme avec les pièces disponibles

    P OUR Récupérer des pièces utilisées pour obtenir le montant optimal

    rendu = {}
    montant = somme
    while montant > 0:
        piece = track[montant]
        if piece is None:
            return None, None
        if piece in rendu:
            rendu[piece] += 1
        else:
            rendu[piece] = 1
        montant -= piece

    return dp[somme], rendu

print("Nombre minimal de pièces :", nb_pieces)
print("Composition des pièces :", rendu)
```

Solution Exercice 02 :

Pour résoudre ce problème, on va utiliser une approche de programmation dynamique en construisant un tableau table où chaque cellule table[i][j] représente la valeur maximale obtenue en utilisant les premiers i objets et une capacité maximale de j.

```
def sac_a_dos(valeurs, poids, capacite):
    nombre_objets = len(valeurs)

    table = [[0] * (capacite + 1) for _ in range(nombre_objets + 1)] //Créer une matrice pour stocker les résultats intermédiaires

    for objet in range(1, nombre_objets + 1) //Remplir la table de manière dynamique

        for capacite_actuelle in range(1, capacite + 1):
            if poids[objet - 1] <= capacite_actuelle // Vérifier si l'objet peut être ajouté dans le sac

            table[objet][capacite_actuelle] = max(table[objet - 1][capacite_actuelle],
                valeurs[objet - 1] + table[objet - 1][capacite_actuelle - poids[objet - 1]])
//Calculer la meilleure valeur entre inclure ou non l'objet

            else:
                table[objet][capacite_actuelle] = table[objet - 1][capacite_actuelle] // Ne pas inclure l'objet
    return table[nombre_objets][capacite] //La valeur maximale obtenue est dans la dernière cellule de la table
```

Exemple d'exécution détaillée :

1. **Initialisation de la table** :

Table	0	1	2	3	...	50
Objet 0	0	0	0	0	...	0
Objet 1	0	0	0	0	...	0
Objet 2	0	0	0	0	...	0
Objet 3	0	0	0	0	...	0

2. **Remplissage de la table** :

a. **Objet 1** (valeur 60, poids 10) :

Premier objet (valeur 60, poids 10) :
Pour chaque capacité j :

- Si j < 10, table[1][j] = 0 (on ne peut pas prendre cet objet).
- Si j >= 10, table[1][j] = 60 (on prend l'objet)

Table	0	1	2	...	10	...	50
Objet 1	0	0	0	...	60	...	60

b. **Objet 2** (valeur 100, poids 20) :

Pour chaque capacité j :

- Si j < 20, table[2][j] = table[1][j]
- Si j >= 20, on compare deux options

1. Ne pas prendre l'objet, donc table[2][j] = table[1][j].
2. Prendre l'objet, donc table[2][j] = 100 + table[1][j - 20]

Table	**0**	**1**	**2**	**...**	**20**	**...**	**50**
Objet 2	0	0	0	...	100	...	160

c. **Objet 3** (valeur 120, poids 30) :

Pour chaque capacité j :

- Si j < 30, table[3][j] = table[2][j]
- Si j >= 30, de même on compare deux options

1. Ne pas prendre l'objet, donc table[3][j] = table[2][j].
2. Prendre l'objet, donc table[3][j] = 120 + table[2][j - 30]

Table	**0**	**1**	**2**	**...**	**30**	**...**	**50**
Objet 3	0	0	0	...	160	...	220

La meilleure valeur possible dans un sac de capacité 50 est **220**.