**Programmation dynamique – Activités d’introduction**

**Activité 1 : la suite de Fibonacci**

La célèbre suite de Fibonacci est définie par F0 = 0, F1 = 1 et pour tout entier naturel *n* ≥ 2, F*n* = F*n*-1 + F*n*-2.

Pour les non matheux, il s’agit de la suite de nombres 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 …

Chaque nombre s’obtient en ajoutant les deux précédents. Le premier terme de cette liste est dit terme de rang ou d’indice 0, le suivant terme de rang 1, … Ainsi, le terme de rang 7 est égal à 13. On le note F7.

Écrire une fonction récursive fibo(n) renvoyant le nombre d’indice *n* de la suite de Fibonacci, le premier de la liste ayant pour indice 0.

Utiliser votre fonction pour calculer les termes d’indice 10, 20, 30, 40… quel est le problème ?

**Activité 2 : le sac à dos**

On rappelle le problème du sac à dos déjà vu en première : on dispose de *n* objets assimilables à des couples (valeur, poids) et d’un sac à dos qui peut porter un poids maximum *w*. L’objectif est de maximiser la valeur des objets contenus dans le sac.

Nous avons vu deux stratégies en première :

* force brute : tester toutes les combinaisons possibles, envisageable avec 20 objets par exemple, mais pas avec 60 objets.
* algorithmes gloutons :
  + glouton 1 : on prend d’abord les objets de valeurs maximales
  + glouton 2 : on prend d’abord les objets maximisant le rapport valeur/poids.

Les algorithmes gloutons sont très rapides, en O(*n log2*(*n*)) si on trie les objets suivant le critère choisi avec un bon algorithme de tri, mais ne garantissent pas d’obtenir la meilleure solution.

**Résolution par programmation dynamique**

On peut construire une solution optimale du problème à *i* objets à partir d’une résolution du problème à *i* – 1 objets.

Supposons qu’on a résolu le problème à *i* – 1 objets pour un poids maximal *p* allant de 0 à *w*.

On rajoute un *i*-ème objet (*vi*, *pi*). Alors, une solution optimale du problème à *i* objets avec un poids maximal de *w* est :

* soit une solution optimale du problème à *i* – 1 objets avec le poids maximal *w*
* soit une solution optimale du problème à *i* – 1 objets avec le poids maximal *w* – *pi* à laquelle on ajoute le *i*-ème objet.

On résout donc successivement les problèmes à 1 objet, 2 objets, 3 objets, … pour les poids allant de 0 à *w*. On présente les solutions dans un tableau. Le contenu du tableau dépend de l’ordre des objets mais pas la dernière ligne.

**Exemple**

Résolution du problème du sac à dos avec la liste objets = [(3, 2), (8, 10), (2, 2), (8, 1), (4, 6), (6, 6)] et le poids maximal *w* = 10 kg. Les objets sont au format (valeur, poids).

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| objets\poids | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 2 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 8 |
| 3 | 0 | 0 | 3 | 3 | 5 | 5 | 5 | 5 | 5 | 5 | 8 |
| 4 | 0 | 8 | 8 | 11 | 11 | 13 | 13 | 13 | 13 | 13 | 13 |
| 5 | 0 | 8 | 8 | 11 | 11 | 13 | 13 | 13 | 13 | 15 | 15 |
| 6 | 0 | 8 | 8 | 11 | 11 | 13 | 13 | 14 | 14 | 17 | 17 |

La valeur maximale est 17, atteinte avec un poids de 9 kg. Puisque cette valeur n’est pas atteinte avec 5 objets, on a pris l’objet n°6, qui pèse 6 kg, donc il reste 9 – 6 = 3 kg pour 5 objets. Pour 5 objets, la valeur maximale atteinte avec 3 kg est égale à 11, c’est la même avec 4 objets. On n’a donc pas pris l’objet n°5, mais on a pris l’objet n°4 qui pèse 1 kg, donc il reste 2 kg pour 3 objets, ce qui permet une valeur égale à 3, déjà atteinte avec l’objet n°1.

On obtient donc la valeur optimale de 17 avec les objets 1, 4, 6.

**Exercice**

Même exercice avec objets = [(5, 3), (9, 2), (10, 5), (6, 4), (7, 1), (9, 3)]et*w* = 10.

**Algorithme**

1. Écrire l’algorithme en langage naturel permettant, à partir d’une liste d’objets au format (valeur, poids) et d’un poids maximal *w* de construire le tableau des solutions du problème du sac à dos comme ci-dessus.
2. Écrire l’algorithme renvoyant une solution optimale à partir du tableau précédent.

***ou***

Expliquer la démarche en français le plus précisément possible.

1. Quelle est la complexité, en temps et en mémoire, de cette méthode de résolution ?

**Programmation**

Ouvrir le fichier sacados\_eleve.py.

1. Écrire la fonction tableau\_kp\_dynamique(objets, w) qui renvoie le tableau donnant les solutions optimales pour 0 à len(objets) objets et des poids de 0 à w.

Exécuter le code pour tester votre fonction.

1. Écrire la fonction kp\_dynamique(objets, w), qui utilise la fonction tableau\_kp\_dynamique(objets, w) et renvoie la valeur maximale et une liste d’objets réalisant cette valeur.

Exécuter la fonction test\_dynamique() pour tester votre fonction.