Chapitre 2 : Programmation dynamique

Nicky Sonigo

Lycée Victor Hugo
PC*

2023/2024

Tables des matières

Tables des matières

Introduction

En algorithmique, le principe du **diviser pour régner** repose sur une idée simple :

- ⋆ découper un problème compliqué en sous-problèmes plus simples (de manière récursive)
- ⋆ résoudre les sous-problèmes et combiner leurs solutions pour résoudre le problème initial

Cette méthode excelle dans de nombreux domaines (dichotomie, tri fusion, tri rapide) mais elle atteint rapidement ses limites lorsque les sous-problèmes rencontrés ne sont pas uniques (chevauchements de sous-problèmes). Dans ce cas, on en vient à recalculer les solutions de sous-problèmes déjà rencontrés ce qui rend notre programme très lent, voire inutilisable.

Introduction

Le paradigme de la **programmation dynamique** apporte un moyen de pallier ce problème d'efficacité en stockant les solutions intermédiaires afin de pouvoir les réutiliser sans les recalculer. C'est un exemple parfait de compromis entre complexité temporelle et complexité spaciale.

La programmation dynamique est fréquemment employée pour résoudre des problèmes d'optimisation : elle s'applique dès lors que la solution optimale peut être déduite des solutions optimales des sous-problèmes (c'est le principe d'optimalité de Bellman). Dans les années 1950, Richard Bellman introduit le concept de *dynamic programmation*. À l'époque, « programming » signifie « optimisation » ou « ordonnancement ».

Tables des matières

Tables des matières

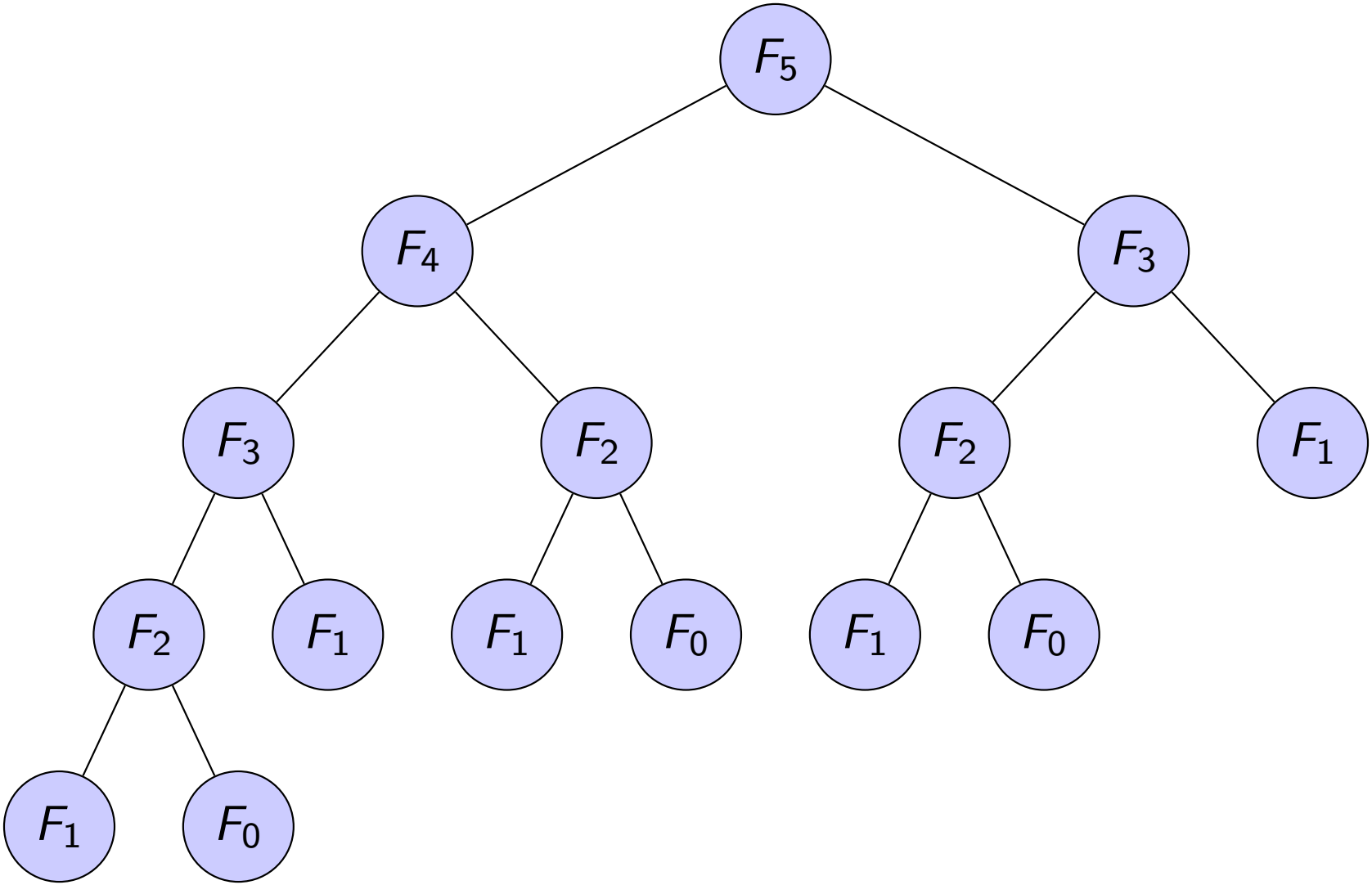
La suite de Fibonacci est la suite $(F_n)_{n\in\mathbb{N}}$ définie par

$$F_0 = F_1 = 1 \qquad \text{et} \qquad \forall n \in \mathbb{N}, \quad F_{n+2} = F_{n+1} + F_n$$

Pour calculer les termes de la suite de Fibonacci, une solution naïve consiste à utiliser la programmation récursive :

```
1 def Fibonacci(n):
2     if n == 0 or n == 1:
3         return 1
4     return Fibonacci(n-1) + Fibonacci(n-2)
```

Le problème est que ce type de fonction est très inefficace comme le montre l'arbre des appels récursifs nécessaires pour calculer F_5 :



On constate que de nombreux noeuds (et leur sous-arbre engendré) sont répétés : F_3 apparait deux fois, F_2 apparait trois fois. La complexité de cet algorithme récursif est exponentielle (voir cours de 1re année).

Tables des matières

Définition 1 : Mémoïsation

La **mémoïsation** consiste à améliorer un algorithme récursif naïf en sauvegardant les solutions des sous-problèmes déjà résolus en vue de les réutiliser plus tard si besoin.

Pour calculer les termes de la suite de Fibonacci, on construit un dictionnaire dont les clés sont les nombres de 0 à n et les valeurs sont les termes de la suite correspondants :

```
1  M = {}
2
3  def Fibonacci(n):
4      if n in M:
5          return M[n]
6
7      if n == 0 or n == 1:
8          f = 1
9      else:
10         f = Fibonacci(n-1) + Fibonacci(n-2)
11     M[n] = f    # On enregistre chaque résultat calculé (mémoisation)
12
13     return f
```

Calcul de la complexité

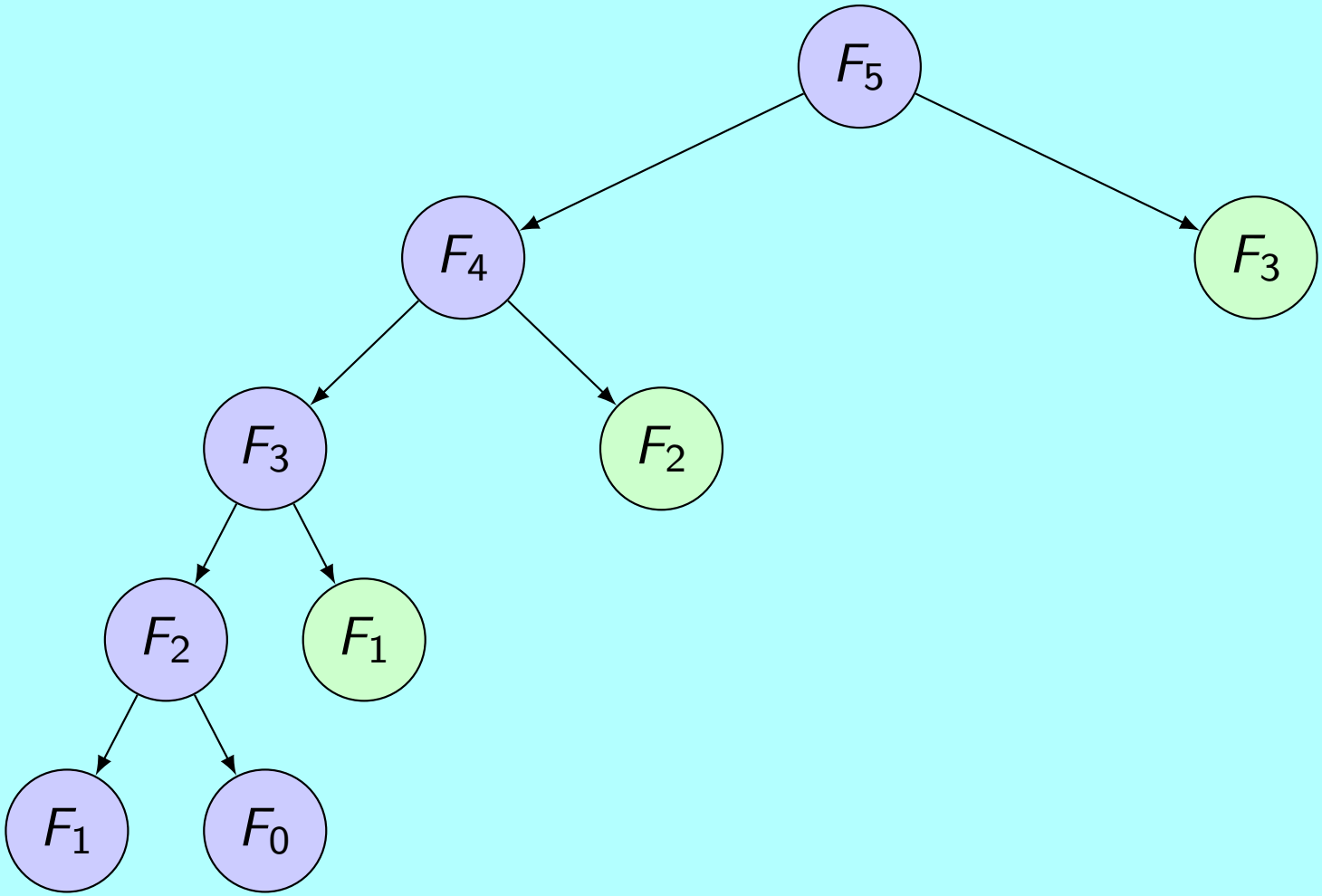
Soit $C(n)$ le nombre d'opérations pour calculer `Fibonacci(n)`. On a

$$C(n) = C(n-1) + 1 + O(1)$$

car F_{n-2} aura été mémoïsé lors du calcul de F_{n-1} et la recherche d'un élément d'un dictionnaire se fait en $O(1)$.

On a donc $\boxed{C(n) = O(n)}$

Calcul de la complexité



Pour chaque noeud en vert, il n'y a pas d'appel récursif car le problème a déjà été résolu avant (le parcours du graphe se fait en profondeur).

Remarque

Pour évaluer rapidement la complexité de ce type d'algorithme, on peut utiliser le principe suivant :

$$C(n) = \text{Nombre de sous-problèmes} \times \text{Complexité par sous-problème}$$

en ignorant les appels récursifs pour la complexité des sous-problèmes.
Pour Fibonacci, nous avons n sous-problèmes et pour chacun d'entre eux, on effectue $O(1)$ opérations en ignorant les appels récursifs.

Exercice 1

Écrire une fonction Python `binomial_memo(n, p)`, basée sur le principe de la programmation dynamique et de la mémoïsation, calculant le coefficient binomial $\binom{n}{p}$ à l'aide de la formule de Pascal

$$\binom{n}{p} = \binom{n-1}{p-1} + \binom{n-1}{p}$$

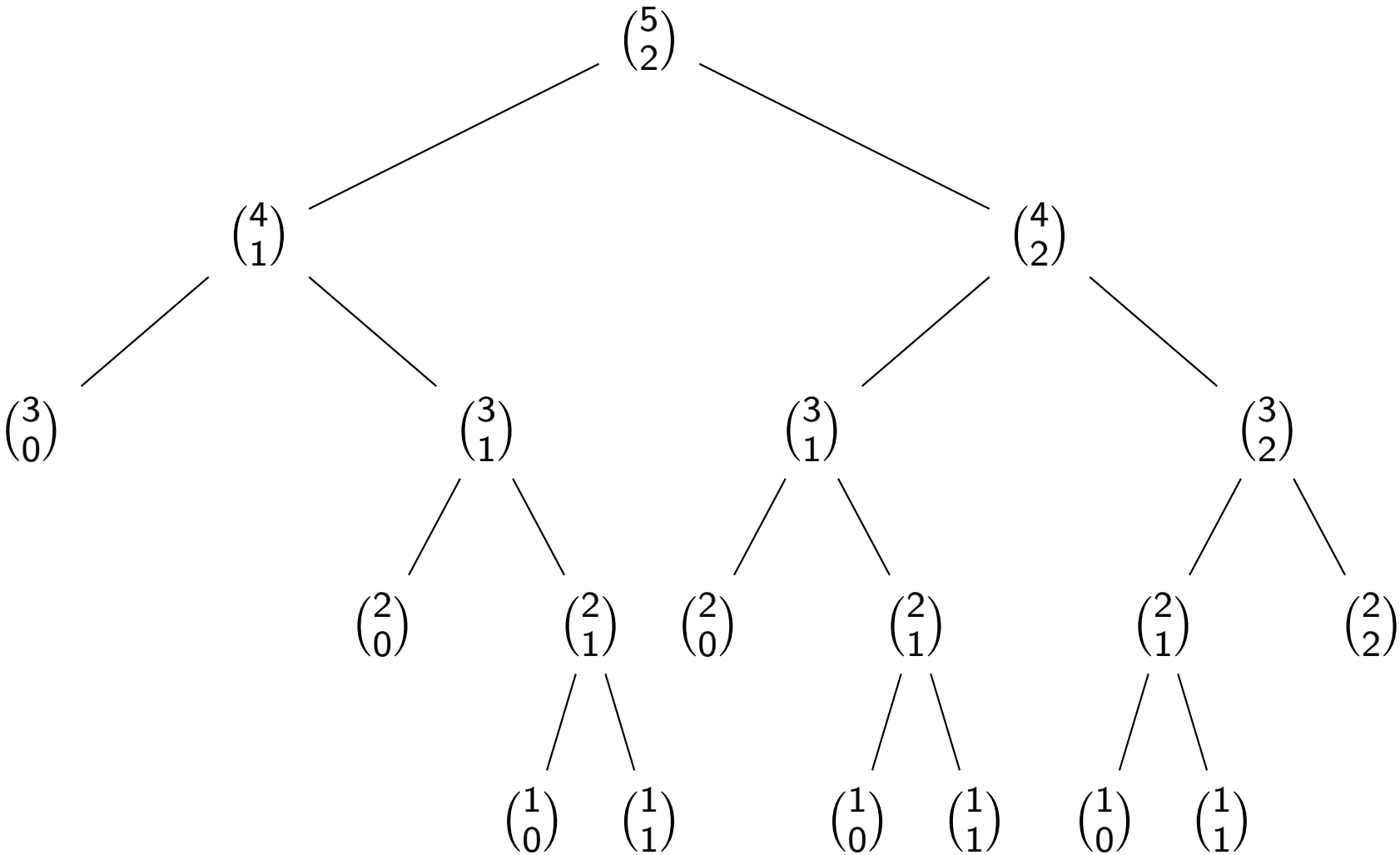
Quelle est sa complexité ?

On implémente d'abord un algorithme récursif naïf :

```
def binomial_naif(n, p):
    # Cas de base
    if p == 0:
        return 1
    elif n == 0:
        return 0

    return binomial_naif(n-1, p-1) + binomial_naif(n-1, p)
```

Comme pour Fibonacci, cet algorithme a une complexité exponentielle :



On « mémoïse » l'algorithme naïf :

```
1  M = {}
2
3  def binomial(n, p):
4      # Calcul déjà fait :
5      if (n, p) in M:
6          return M[(n, p)]
7
8      # Algorithme récursif sans renvoyer le résultat :
9      if p == 0:
10         resultat = 1
11     elif n == 0:
12         resultat = 0
13     else:
14         resultat = binomial(n-1, p-1) + binomial(n-1, p)
15
16     # Mémoisation du résultat pour de futurs appels
17     # récursifs :
18     M[(n, p)] = resultat
19
20     return resultat
```

On remarque que l'algorithme de programmation dynamique par mémoïsation calcule exactement les coefficients binomiaux nécessaires pour calculer $\binom{n}{p}$:

```
In : binomial(8, 4)
Out: 70
In : M
Out: {(4, 0): 1, (3, 0): 1, (2, 0): 1, (1, 0): 1, (0, 0): 1,
(0, 1): 0, (1, 1): 1, (2, 1): 2, (3, 1): 3, (4, 1): 4,
(5, 1): 5, (0, 2): 0, (1, 2): 0, (2, 2): 1, (3, 2): 3,
(4, 2): 6, (5, 2): 10, (6, 2): 15, (0, 3): 0, (1, 3): 0,
(2, 3): 0, (3, 3): 1, (4, 3): 4, (5, 3): 10, (6, 3): 20,
(7, 3): 35, (0, 4): 0, (1, 4): 0, (2, 4): 0, (3, 4): 0,
(4, 4): 1, (5, 4): 5, (6, 4): 15, (7, 4): 35, (8, 4): 70}
In : len(M)
Out: 35
```

Si on calcule plusieurs coefficients binomiaux consécutivement, le calcul sera de plus en plus efficace.

Tables des matières

Définition 2

La **méthode ascendante (bottum-up en anglais)** consiste à résoudre d'abord les plus petits sous-problèmes puis de combiner leurs solutions pour résoudre des sous-problèmes de plus en plus grands jusqu'à atteindre le problème initial. L'algorithme est alors itératif.

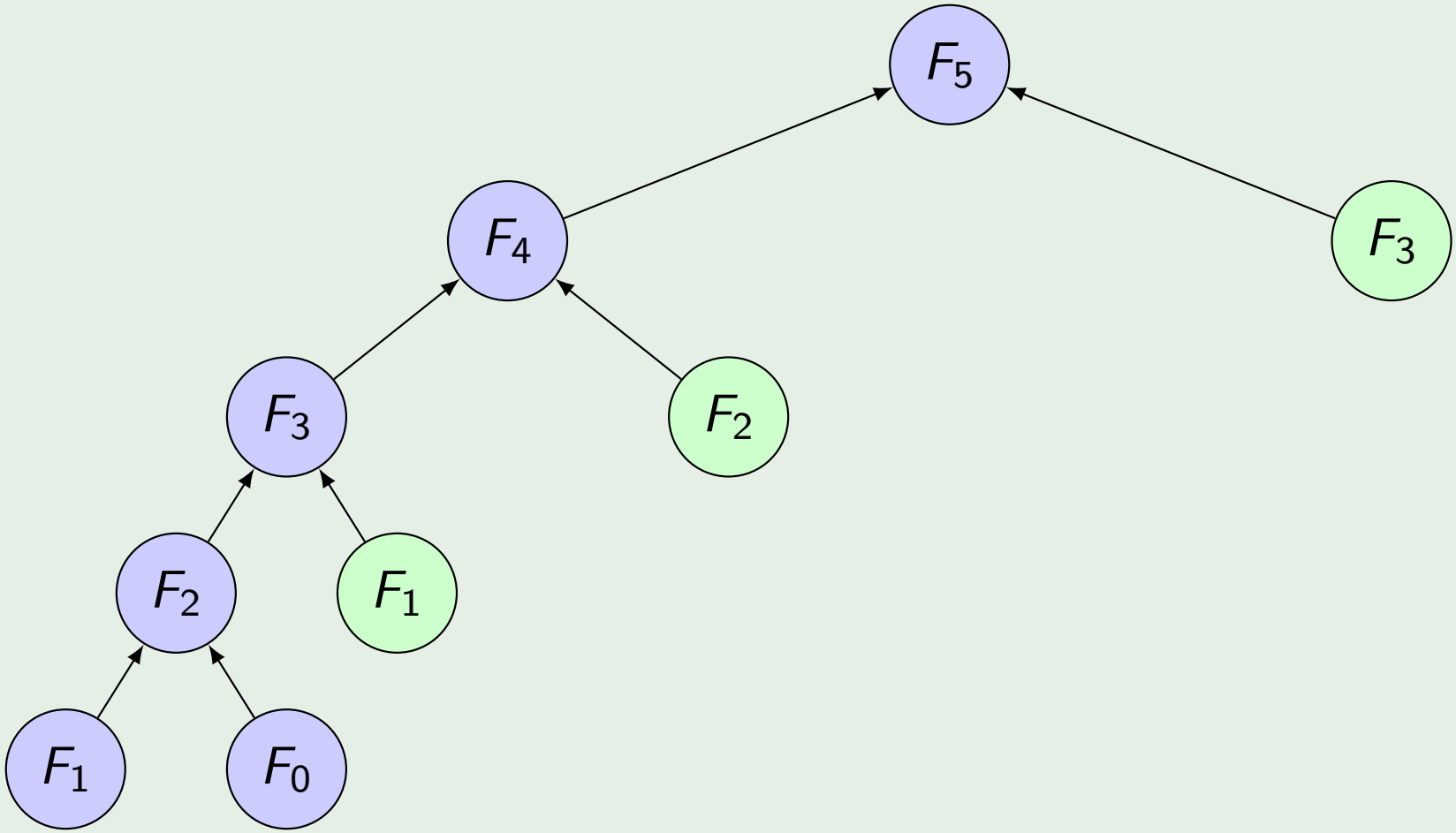
```python
def Fibonacci(n):
    F = [1, 1]

    for i in range(2, n):
        F.append(F[-1] + F[-2])

    return F[n]
```

Remarques

⋆ La complexité de la méthode ascendante est plus facile à calculer que celle de la mémoïsation car l'algorithme n'est pas récursif : ici encore, on a $\boxed{C(n) = O(n)}$



Remarques

⋆ Dans certains cas, on peut encore optimiser la méthode ascendante pour diminuer la complexité spaciale. Par exemple ici nous pourrions garder seulement les deux derniers termes calculés car nous n'aurons plus besoin des précédents :

```
1 def Fibonacci(n):
2     a, b = 1, 1
3
4     for i in range(2, n):
5         a, b = b, a + b
6
7     return b
```

⋆ L'inconvénient de la méthode ascendante (par rapport à la mémoïsation) est qu'il faut s'assurer que les calculs sont fait dans le bon ordre et de ne pas faire trop de calculs inutiles : pour l'exemple de la suite de Fibonacci c'est assez simple mais dans le cas général il faut étudier les dépendances des différents sous-problèmes.

Exercice 2

Écrire une fonction Python `binomial_asc(n, p)`, basée sur le principe de la programmation dynamique et de la méthode ascendante, calculant le coefficient binomial $\binom{n}{p}$ à l'aide de la formule de Pascal

$$\binom{n}{p} = \binom{n-1}{p-1} + \binom{n-1}{p}$$

Quelle est sa complexité ?

On transforme l'algorithme mémoïsé en calcul de bas en haut :

```
1  M = {}
2
3  def binomial(n, p):
4      # Initialisation de la première ligne et de la première
       colonne :
5      for i in range(n+1):
6          M[(i, 0)] = 1
7      for j in range(1, p+1):
8          M[(0, j)] = 0
9
10     # Calcul des lignes suivantes :
11     for i in range(1, n+1):
12         for j in range(1, p+1):
13             M[(i, j)] = M[(i-1, j-1)] + M[(i-1, j)]
14
15     return M[(n, p)]
```

On remarque que cet algorithme calcule 10 coefficients binomiaux « inutiles » : on peut améliorer analysant plus précisemment les dépendances des coefficients.

```
In : binomial(8, 4)
Out: 70
In : M
Out: {(0, 0): 1, (1, 0): 1, (2, 0): 1, (3, 0): 1, (4, 0): 1,
(5, 0): 1, (6, 0): 1, (7, 0): 1, (8, 0): 1, (0, 1): 0,
(0, 2): 0, (0, 3): 0, (0, 4): 0, (1, 1): 1, (1, 2): 0,
(1, 3): 0, (1, 4): 0, (2, 1): 2, (2, 2): 1, (2, 3): 0,
(2, 4): 0, (3, 1): 3, (3, 2): 3, (3, 3): 1, (3, 4): 0,
(4, 1): 4, (4, 2): 6, (4, 3): 4, (4, 4): 1, (5, 1): 5,
(5, 2): 10, (5, 3): 10, (5, 4): 5, (6, 1): 6, (6, 2): 15,
(6, 3): 20, (6, 4): 15, (7, 1): 7, (7, 2): 21, (7, 3): 35,
(7, 4): 35, (8, 1): 8, (8, 2): 28, (8, 3): 56, (8, 4): 70}
In : len(M)
Out: 45
```

Tables des matières

Tables des matières

On dispose de N objets numérotés de 1 à $N \in \mathbb{N}^*$. Chaque objet i a un poids $w_i \in \mathbb{N}^*$ et une valeur $v_i \in \mathbb{N}^*$. On souhaite remplir un sac à dos de capacité maximale W_{max} avec ces objets.

Comment choisir les objets de sorte à maximiser la valeur emportée $\sum_{i \in I} v_i$ *tout en respectant la contrainte* $\sum_{i \in I} w_i \leqslant W_{max}$ *?*

Ce type de problème fait partie des problèmes dits d'optimisation.

Exemple

On dispose d'un sac à dos de capacité $W_{max} = 20$ et de $N = 10$ objets :

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

Comment remplir le sac à dos sans dépasser sa capacité tout en maximisant la valeur totale du sac ?

Tables des matières

Découpage en sous-problèmes

Pour avoir un problème plus simple à résoudre, on peut limiter le nombre d'objets disponibles. Pour tous $i \in [\![0, N]\!]$ et $W \in [\![0, W_{max}]\!]$, notons $V(i, W)$ la valeur maximale d'un sac à dos de capacité W en n'utilisant que les objets de numéros 1 à i.

Découpage en sous-problèmes

Pour avoir un problème plus simple à résoudre, on peut limiter le nombre d'objets disponibles. Pour tous $i \in [\![0, N]\!]$ et $W \in [\![0, W_{max}]\!]$, notons $V(i, W)$ la valeur maximale d'un sac à dos de capacité W en n'utilisant que les objets de numéros 1 à i.

Exemple

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

Découpage en sous-problèmes

Pour avoir un problème plus simple à résoudre, on peut limiter le nombre d'objets disponibles. Pour tous $i \in [\![0, N]\!]$ et $W \in [\![0, W_{max}]\!]$, notons $V(i, W)$ la valeur maximale d'un sac à dos de capacité W en n'utilisant que les objets de numéros 1 à i.

Exemple

Objet	1	2	3	4	5
Poids	9	8	15	2	13
Valeur	17	10	5	1	19

Dans cet exemple, le sous-problème avec $i = 5$ et $W = 10$ est très facile à résoudre.

Découpage en sous-problèmes

Pour avoir un problème plus simple à résoudre, on peut limiter le nombre d'objets disponibles. Pour tous $i \in [\![0, N]\!]$ et $W \in [\![0, W_{max}]\!]$, notons $V(i, W)$ la valeur maximale d'un sac à dos de capacité W en n'utilisant que les objets de numéros 1 à i.

Exemple

Objet	1	2	3	4	5
Poids	9	8	15	2	13
Valeur	17	10	5	1	19

Dans cet exemple, le sous-problème avec $i = 5$ et $W = 10$ est très facile à résoudre : les seuls choix crédibles sont de prendre seulement l'objet 1 (valeur : 17) ou de prendre les objets 2 et 4 (valeur : 11). La solution optimale est donc de prendre seulement l'objet 1 et on a $V(5, 10) = 17$.

⋆ Pour $i = 0$, il n'y a aucun objet disponible donc $V(0, W) = 0$ pour tout $W \in [\![0, W_{max}]\!]$.

- ⋆ Pour $i = 0$, il n'y a aucun objet disponible donc $V(0, W) = 0$ pour tout $W \in [\![0, W_{max}]\!]$.
- ⋆ Pour $W = 0$, aucun objet ne peut être choisi (les poids sont tous strictement positifs) donc $V(i, 0) = 0$ pour tout $i \in [\![0, N]\!]$.

- ⋆ Pour $i = 0$, il n'y a aucun objet disponible donc $V(0, W) = 0$ pour tout $W \in [\![0, W_{max}]\!]$.
- ⋆ Pour $W = 0$, aucun objet ne peut être choisi (les poids sont tous strictement positifs) donc $V(i, 0) = 0$ pour tout $i \in [\![0, N]\!]$.
- ⋆ Étant donné $i \in [\![1, N]\!]$:

- ⋆ Pour $i = 0$, il n'y a aucun objet disponible donc $V(0, W) = 0$ pour tout $W \in [\![0, W_{max}]\!]$.
- ⋆ Pour $W = 0$, aucun objet ne peut être choisi (les poids sont tous strictement positifs) donc $V(i, 0) = 0$ pour tout $i \in [\![0, N]\!]$.
- ⋆ Étant donné $i \in [\![1, N]\!]$:
  - ⋆ Si l'objet i est dans la solution optimale de ce sous-problème, alors $w_i \leqslant W$ et la valeur du sac à dos sera

$$V(i, W) = v_i + V(i - 1, W - w_i)$$

- ⋆ Pour $i = 0$, il n'y a aucun objet disponible donc $V(0, W) = 0$ pour tout $W \in [\![0, W_{max}]\!]$.
- ⋆ Pour $W = 0$, aucun objet ne peut être choisi (les poids sont tous strictement positifs) donc $V(i, 0) = 0$ pour tout $i \in [\![0, N]\!]$.
- ⋆ Étant donné $i \in [\![1, N]\!]$:
  - ⋆ Si l'objet i est dans la solution optimale de ce sous-problème, alors $w_i \leqslant W$ et la valeur du sac à dos sera

$$V(i, W) = v_i + V(i - 1, W - w_i)$$

  - ⋆ Sinon, la valeur du sac à dos sera

$$V(i, W) = V(i - 1, W)$$

- ⋆ Pour $i = 0$, il n'y a aucun objet disponible donc $V(0, W) = 0$ pour tout $W \in [\![0, W_{max}]\!]$.
- ⋆ Pour $W = 0$, aucun objet ne peut être choisi (les poids sont tous strictement positifs) donc $V(i, 0) = 0$ pour tout $i \in [\![0, N]\!]$.
- ⋆ Étant donné $i \in [\![1, N]\!]$:
  - ⋆ Si l'objet i est dans la solution optimale de ce sous-problème, alors $w_i \leqslant W$ et la valeur du sac à dos sera

$$V(i, W) = v_i + V(i - 1, W - w_i)$$

  - ⋆ Sinon, la valeur du sac à dos sera

$$V(i, W) = V(i - 1, W)$$

Comme on cherche à maximiser la valeur du sac à dos, on a

$$V(i, W) = \begin{cases} \max\left(v_i + V(i - 1, W - w_i); V(i - 1, W)\right) & \text{si } w_i \leqslant W \\ V(i - 1, W) & \text{sinon} \end{cases}$$

Exemple

Objet	1	2	3	4	5
Poids	9	8	15	2	13
Valeur	17	10	5	1	19

On a vu que $V(5, 10) = 17$ avec une composition optimale de sac n'utilisant que l'objet 1.

Exemple

Objet	1	2	3	4	5	6
Poids	9	8	15	2	13	2
Valeur	17	10	5	1	19	11

On a vu que $V(5, 10) = 17$ avec une composition optimale de sac n'utilisant que l'objet 1. Si on autorise maintenant l'utilisation de l'objet 6, alors on a deux possibilités :

Exemple

Objet	1	2	3	4	5	6
Poids	9	8	15	2	13	2
Valeur	17	10	5	1	19	11

On a vu que $V(5, 10) = 17$ avec une composition optimale de sac n'utilisant que l'objet 1. Si on autorise maintenant l'utilisation de l'objet 6, alors on a deux possibilités :

- ⋆ Soit on ne prend pas l'objet 6 dans la solution optimale, auquel cas la valeur totale du sac sera de $V(5, 10) = 17$.

Exemple

Objet	1	2	3	4	5	6
Poids	9	8	15	2	13	2
Valeur	17	10	5	1	19	11

On a vu que $V(5, 10) = 17$ avec une composition optimale de sac n'utilisant que l'objet 1. Si on autorise maintenant l'utilisation de l'objet 6, alors on a deux possibilités :

- ⋆ Soit on ne prend pas l'objet 6 dans la solution optimale, auquel cas la valeur totale du sac sera de $V(5, 10) = 17$.
- ⋆ Soit on prend l'objet 6 dans la solution optimale. Dans ce cas, les autres objets à prendre dans la solution optimale doivent être parmi les objets de 1 à 5 et ne doivent pas dépasser le poids de $10 - 2 = 8$. Ainsi la valeur optimale de ces autres objets est de $V(5, 8) = 10$ (on ne prend que l'objet 2). La valeur totale du sac sera alors de $11 + V(5, 8) = 21$.

Exemple

Objet	1	2	3	4	5	6
Poids	9	8	15	2	13	2
Valeur	17	10	5	1	19	11

On a vu que $V(5, 10) = 17$ avec une composition optimale de sac n'utilisant que l'objet 1. Si on autorise maintenant l'utilisation de l'objet 6, alors on a deux possibilités :

- ⋆ Soit on ne prend pas l'objet 6 dans la solution optimale, auquel cas la valeur totale du sac sera de $V(5, 10) = 17$.
- ⋆ Soit on prend l'objet 6 dans la solution optimale. Dans ce cas, les autres objets à prendre dans la solution optimale doivent être parmi les objets de 1 à 5 et ne doivent pas dépasser le poids de $10 - 2 = 8$. Ainsi la valeur optimale de ces autres objets est de $V(5, 8) = 10$ (on ne prend que l'objet 2). La valeur totale du sac sera alors de $11 + V(5, 8) = 21$.

La meilleure solution est la deuxième donc $V(6, 10) = 11 + V(5, 8) = 21$

Exemple

Objet	1	2	3	4	5	6	7
Poids	9	8	15	2	13	2	9
Valeur	17	10	5	1	19	11	6

Ajoutons maintenant l'objet numéro 7. On a toujours deux possibilités :

Exemple

Objet	1	2	3	4	5	6	7
Poids	9	8	15	2	13	2	9
Valeur	17	10	5	1	19	11	6

Ajoutons maintenant l'objet numéro 7. On a toujours deux possibilités :

- ⋆ Soit on ne prend pas l'objet 7 dans la solution optimale, auquel cas la valeur totale du sac sera de $V(6, 10) = 21$.

Exemple

Objet	1	2	3	4	5	6	7
Poids	9	8	15	2	13	2	9
Valeur	17	10	5	1	19	11	6

Ajoutons maintenant l'objet numéro 7. On a toujours deux possibilités :

- ⋆ Soit on ne prend pas l'objet 7 dans la solution optimale, auquel cas la valeur totale du sac sera de $V(6, 10) = 21$.
- ⋆ Soit on prend l'objet 7 dans la solution optimale. Dans ce cas, les autres objets à prendre dans la solution optimale doivent être parmi les objets de 1 à 6 et ne doivent pas dépasser le poids de $10 - 9 = 1$. Ainsi la valeur optimale de ces autres objets est de $V(6, 1) = 0$ (on ne peut rien mettre de plus dans le sac). La valeur totale du sac sera alors de $6 + V(6, 1) = 6$.

Exemple

Objet	1	2	3	4	5	6	7
Poids	9	8	15	2	13	2	9
Valeur	17	10	5	1	19	11	6

Ajoutons maintenant l'objet numéro 7. On a toujours deux possibilités :

- ⋆ Soit on ne prend pas l'objet 7 dans la solution optimale, auquel cas la valeur totale du sac sera de $V(6, 10) = 21$.
- ⋆ Soit on prend l'objet 7 dans la solution optimale. Dans ce cas, les autres objets à prendre dans la solution optimale doivent être parmi les objets de 1 à 6 et ne doivent pas dépasser le poids de $10 - 9 = 1$. Ainsi la valeur optimale de ces autres objets est de $V(6, 1) = 0$ (on ne peut rien mettre de plus dans le sac). La valeur totale du sac sera alors de $6 + V(6, 1) = 6$.

La meilleure solution est la première donc $V(7, 10) = V(6, 10) = 21$.

Tables des matières

Pour qu'un problème soit résoluble par programmation dynamique, il doit impérativement respecter deux propriétés :

- ⋆ **Chevauchement des tâches** : l'algorithme doit être amené à résoudre plusieurs fois le même sous-problème.
- ⋆ **Sous-structure optimale** : un problème dispose d'une sous-structure optimale lorsque les solutions des sous-problèmes qui composent la solution optimale sont elles-mêmes optimales. Pour le problème du sac à dos, si par exemple l'objet numéro N se trouve dans la solution optimale, alors en enlevant cet objet du sac on doit obtenir une solution optimale au sous-problème avec les objets de 1 à $N-1$ et le poids maximal $W_{max} - w_N$.

Tables des matières

```python
w = [9, 8, 15, 2, 13, 2, 9, 7, 15, 1]
v = [17, 10, 5, 1, 19, 11, 6, 12, 18, 4]

def V(i, W):
    if i == 0 or W == 0:
        return 0
    elif w[i-1] <= W:
        return max(v[i-1] + V(i-1, W - w[i-1]), V(i-1, W))
    else:
        return V(i-1, W)
```

```
In[] : print(V(10, 20))
Out[]: 44
```

```python
D = {}

def V(i, W):
    if (i, W) in D:
        return D[(i, W)]

    if i == 0 or W == 0:
        r = 0
    elif w[i-1] <= W:
        r = max(v[i-1] + V(i-1, W - w[i-1]), V(i-1, W))
    else:
        r = V(i-1, W)

    D[(i, W)] = r
    return r
```

```python
def memoisation(v: list, w: list, W_max: int):
    D = {}

    def V(i, W):
        if (i, W) in D:
            return D[(i, W)]

        if i == 0 or W == 0:
            r = 0
        elif w[i-1] <= W:
            r = max(v[i-1] + V(i-1, W - w[i-1]), V(i-1, W))
        else:
            r = V(i-1, W)

        D[(i, W)] = r
        return r

    return V(len(v), W_max)
```

Tables des matières

Étude des dépendances

Ici, il faut commencer par réfléchir aux dépendances des différents $V(i, W)$: pour calculer $V(i, W)$, il est nécessaire de connaître $V(i-1, W-w_i)$ et $V(i-1, W)$. Comme w_i peut prendre n'importe quelle valeur selon le problème, on va devoir calculer $V(i-1, 0), V(i-1, 1), \ldots, V(i-1, W)$ avant de pouvoir calculer $V(i, W)$.

```python
def bottom_up(v: list, w: list, W_max: int):
    N = len(v)

    D = {}

    for i in range(N + 1):
        D[(i, 0)] = 0

    for W in range(W_max + 1):
        D[(0, W)] = 0

    for i in range(1, N + 1):
        for W in range(1, W_max + 1): # L'ordre des deux boucles est important !
            if w[i-1] > W:
                D[(i, W)] = D[(i-1, W)]
            else:
                D[(i, W)] = max(v[i-1] + D[(i-1, W-w[i-1])], D[(i-1, W)])

    return D[(N, W_max)]
```

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7																					
6																					
5																					
4																					
3																					
2																					
1																					
0																					

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7																					
6																					
5																					
4																					
3																					
2																					
1																					
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7																					
6																					
5																					
4																					
3																					
2																					
1	0	0	0	0	0	0	0	0	0												
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7																					
6																					
5																					
4																					
3																					
2																					
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7																					
6																					
5																					
4																					
3																					
2	0	0	0	0	0	0	0	0													
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7																					
6																					
5																					
4																					
3																					
2	0	0	0	0	0	0	0	0													
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7																					
6																					
5																					
4																					
3																					
2	0	0	0	0	0	0	0	0	10												
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7																					
6																					
5																					
4																					
3																					
2	0	0	0	0	0	0	0	0	10												
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7																					
6																					
5																					
4																					
3																					
2	0	0	0	0	0	0	0	0	10	17											
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7																					
6																					
5																					
4																					
3																					
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17				
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7																					
6																					
5																					
4																					
3																					
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27			
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7																					
6																					
5																					
4																					
3																					
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7																					
6																					
5																					
4																					
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7																					
6																					
5																					
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7																					
6																					
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8																					
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9																					
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10																					
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	6 31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	6 31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

135 valeurs sont réellement nécessaires pour le calcul de $V(10, 20)$ mais le calcul de bas en haut en calcule $11 \times 21 = 231$.

Tables des matières

Il est facile ensuite de récupérer la liste optimale des objets à mettre dans le sac à dos : pour cela, il suffit de modifier la fonction précédente pour qu'elle renvoie le dictionnaire complet au lieu de la valeur de celui-ci sur le couple (N, W_{max}).

```python
def bottom_up(v: list, w: list, W_max: int):
    N = len(v)

    D = {}

    for i in range(N + 1):
        D[(i, 0)] = 0

    for W in range(W_max + 1):
        D[(0, W)] = 0

    for i in range(1, N + 1):
        for W in range(1, W_max + 1): # L'ordre des deux boucles est important !
            if w[i-1] > W:
                D[(i, W)] = D[(i-1, W)]
            else:
                D[(i, W)] = max(v[i-1] + D[(i-1, W-w[i-1])], D[(i-1, W)])

    return D
```

Ensuite, la fonction suivante reconstitue la liste des objets à emporter :

```
1  def sac_a_dos(v: list, w: list, W_max: int):
2      D = bottom_up(v, w, W_max)
3      sac = []
4      N = len(v)
5      W = W_max
6
7      while N > 0:
8          if D[(N, W)] > D[(N-1, W)]:
9              sac.append((v[N-1], w[N-1]))
10             W -= w[N-1]
11         N -= 1
12
13     return sac
```

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objets :

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objets : 10

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objets : 10

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objets : 10, 8

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objets : 10, 8

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objets : 10, 8, 6

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objets : 10, 8, 6

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objets : 10, 8, 6

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objets : 10, 8, 6

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objets : 10, 8, 6

Objet	1	2	3	4	5	6	7	8	9	10
Poids	9	8	15	2	13	2	9	7	15	1
Valeur	17	10	5	1	19	11	6	12	18	4

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0	4	11	15	15	16	16	16	16	23	27	28	32	32	33	33	34	34	40	44	44
9	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
8	0	0	11	11	12	12	12	12	12	23	23	28	28	29	29	30	30	33	40	40	41
7	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
6	0	0	11	11	12	12	12	12	12	17	21	28	28	29	29	30	30	31	31	38	38
5	0	0	1	1	1	1	1	1	10	17	17	18	18	19	19	20	20	27	27	28	28
4	0	0	1	1	1	1	1	1	10	17	17	18	18	18	18	18	18	27	27	28	28
3	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
2	0	0	0	0	0	0	0	0	10	17	17	17	17	17	17	17	17	27	27	27	27
1	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	17	17	17	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Objets : 10, 8, 6, 1

Valeur : $4 + 12 + 11 + 17 = 44$ Poids total : 19

Tables des matières

Programmation dynamique

La programmation dynamique garantit d'obtenir la meilleure solution au problème étudié, mais dans un certain nombre de cas sa complexité temporelle reste trop importante pour pouvoir être utilisée dans la pratique.

Programmation dynamique

La programmation dynamique garantit d'obtenir la meilleure solution au problème étudié, mais dans un certain nombre de cas sa complexité temporelle reste trop importante pour pouvoir être utilisée dans la pratique.

Programmation gloutonne

Dans ce type de situation, on se résout à utiliser un autre paradigme de programmation, la **programmation gloutonne**. Alors que la programmation dynamique se caractérise par la résolution par taille croissante de tous les problèmes locaux, la stratégie gloutonne consiste à choisir à partir du problème global un problème local et un seul en suivant une *heuristique* (c'est à dire une stratégie permettant de faire un choix rapide mais pas nécessairement optimal). On ne peut en général garantir que la stratégie gloutonne détermine la solution optimale, mais lorsque l'heuristique est bien choisie on peut espérer obtenir une solution proche de celle-ci.

Pour le problème du sac à dos, il faut définir une heuristique, c'est-à-dire une méthode arbitraire (mais réfléchie) pour évaluer quel objet est prioritaire par rapport à un autre : par exemple, nous pouvons choisir en priorité les objets dont le rapport $\frac{v_i}{w_i}$ est maximal et remplir le sac autant que possible :

```
1  def glouton(v, w, W_max):
2      W = W_max
3      priorites = [(v[i] / w[i], i) for i in range(len(v))]
4      priorites.sort(reverse = True)
5
6      V = i = 0
7      while W >= 0 and i < len(v):
8          objet = priorites[i][1]
9          if w[objet] <= W:
10             W -= w[objet]
11             V += v[objet]
12         i += 1
13
14     return V
```

Comparaison des résultats

En exécutant les deux fonctions précédentes sur différents jeux de données, on constate que dans environ 40% des cas, l'algorithme glouton donne la même solution que l'algorithme par programmation dynamique (qui donne toujours la solution optimale).

Lorsque l'algorithme glouton ne donne pas la solution optimale, celui-ci donne une solution au moins égale à 98% de la solution optimale.