En route vers la Terminale !

Contenus	Capacités attendues	Commentaires
Récursivité.	Écrire un programme récursif. Analyser le fonctionnement d'un programme récursif.	Des exemples relevant de domaines variés sont à privilégier.
Programmation dynamique.	Utiliser la programmation dynamique pour écrire un algorithme.	Les exemples de l'alignement de séquences ou du rendu de monnaie peuvent être présentés. La discussion sur le coût en mémoire peut être développée.

Sac à dos (suite…)

On dispose d'un sac à dos pouvant contenir 9 kg et de 4 objets
- A : 2 kg / 4 €
- B : 2 kg / 2 €
- C : 6 kg / 5 €
- D : 3 kg / 6 €

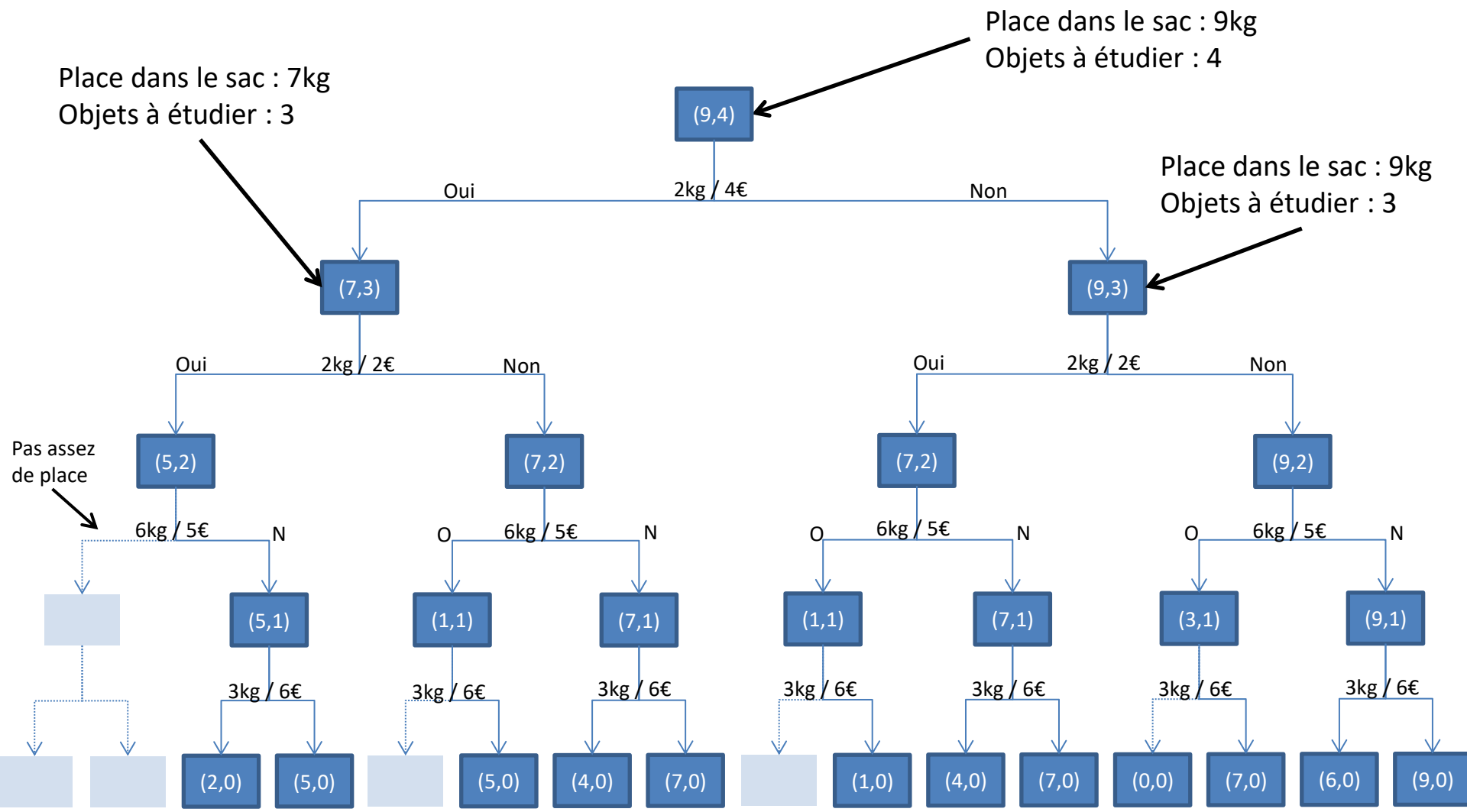
Que prendre pour avoir le plus d'argent dans son sac à dos ?

- le faire à la main : faisable car il n'y a que 4 objets, mais plus compliqué avec plus d'objets
- le programmer avec un algorithme glouton : faisable même si le nombre d'objets est important
  - Critère = prix, on trouve : C-D pour 11 €
  - Critère = ratio, on trouve : A-B-D pour 12 €
  - Critère = poids, on trouve : A-B-D pour 12 €

  Mais on ne sait pas si 12 € est la solution optimale !
- Pour trouver la meilleure combinaison, il n'y a pas le choix : il faut TOUT calculer !

Sac à dos

Rappels :
Sac : 9 kg
A : 2 kg / 4 €
B : 2 kg / 2 €
C : 6 kg / 5 €
D : 3 kg / 6 €



Sac à dos

On pose : V[p][i] : valeur du sac à dos

i : nombre d'objets à étudier
p : poids disponible dans le sac à dos

Ainsi, on veut calculer V[9][4]

Relation de récurrence

Si $i = 0$: il n'y a pas d'objets à étudier, donc valeur = 0

Si $p_i > p$: l'objet étudié est trop lourd, on ne le prend pas.
la valeur du sac reste la même, tout comme le poids disponible

$$V[p][i] = \begin{cases} 0 & si\ i = 0 \\ V[p][i-1] & si\ p_i > p \\ \max(V[p-p_i][i-1] + v_i\ ;\ V[p][i-1]) & si\ p_i \leq p \end{cases}$$

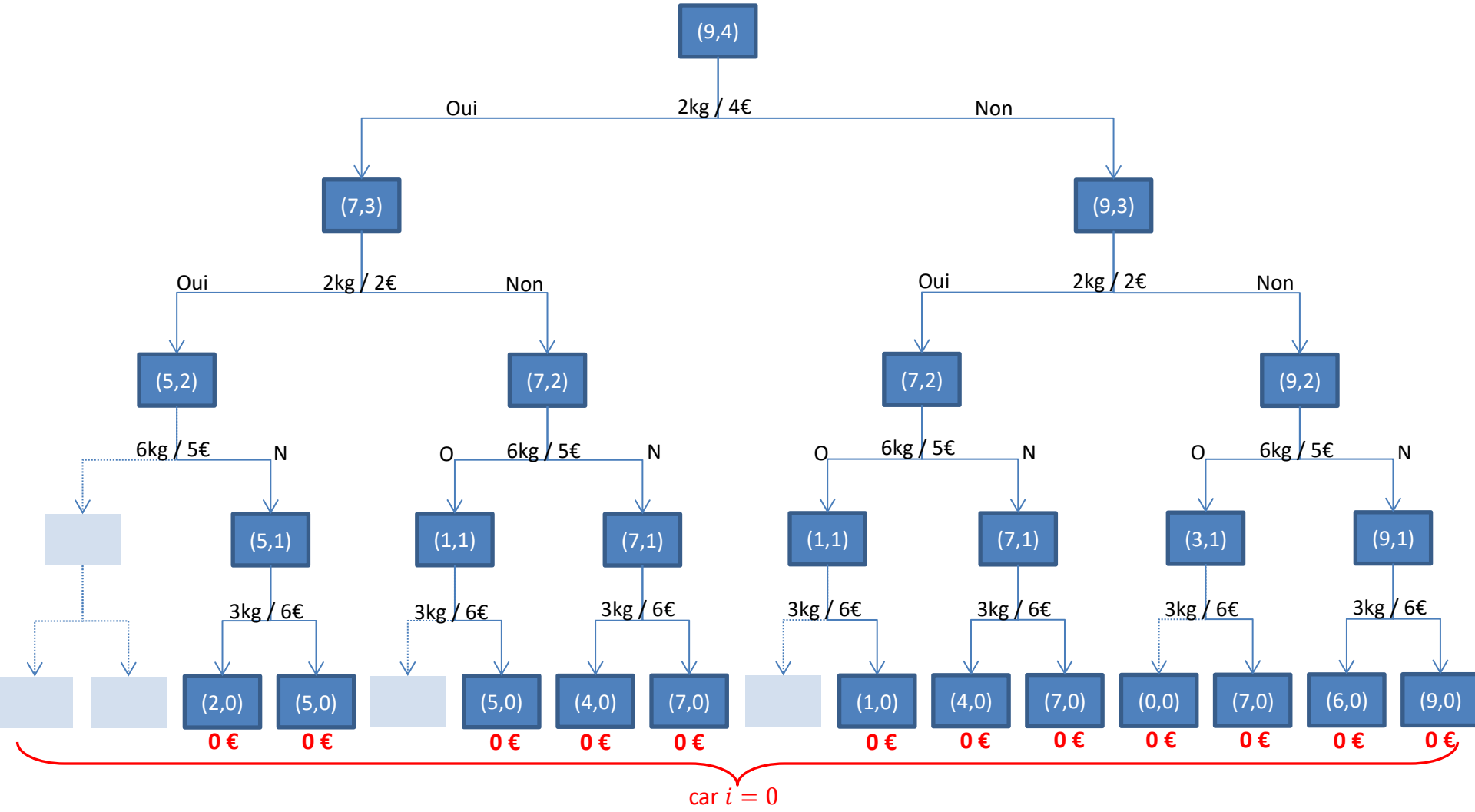
soit on le prend :
la valeur du sac est augmentée de v_i
le poids disponible est diminué de p_i

soit on ne le prend pas (car il sera plus intéressant de prendre un autre objet plus tard).
la valeur du sac reste la même,
le poids disponible reste le même

Si $p_i \leq p$: l'objet étudié est mettable dans le sac,

Sac à dos

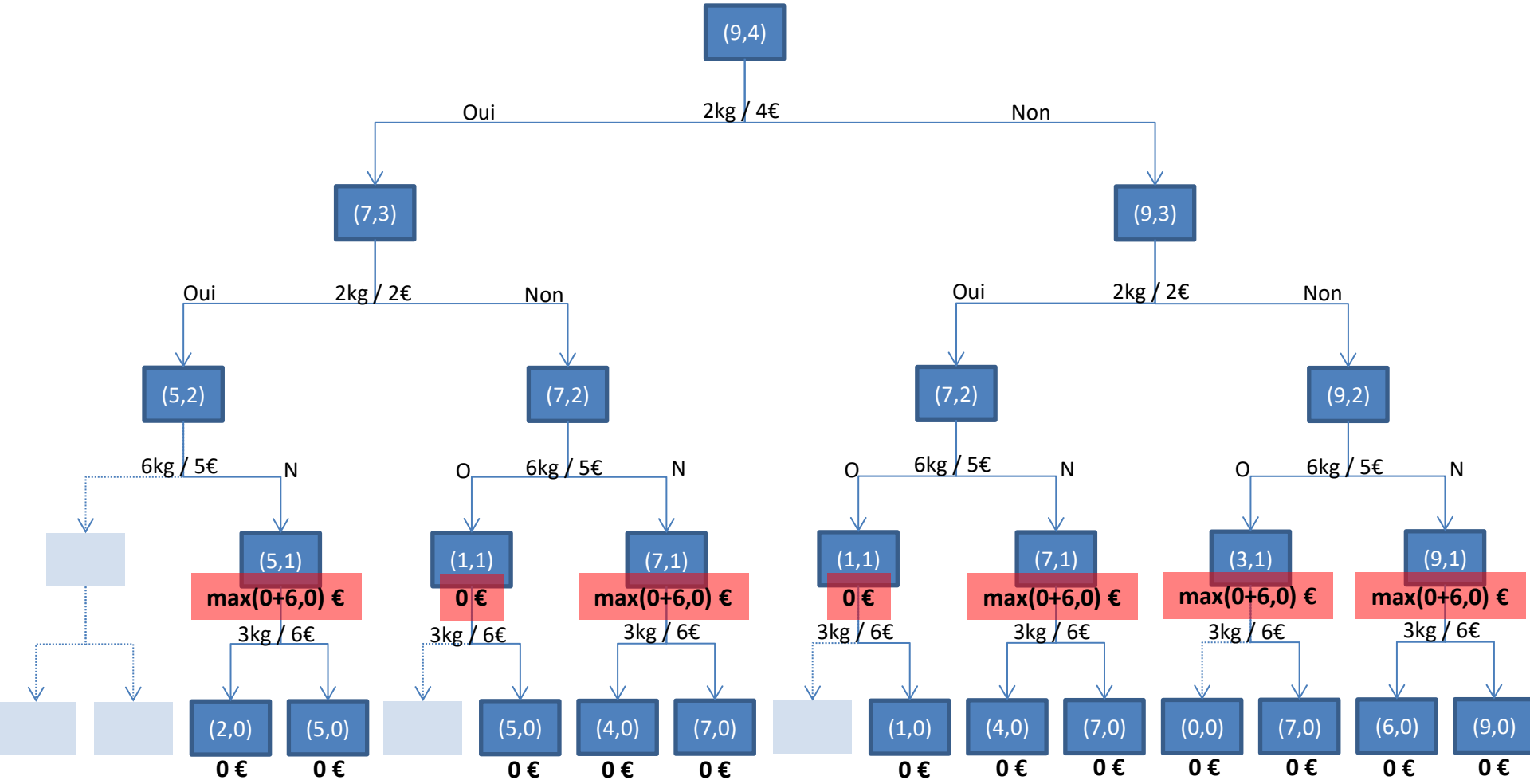
Rappels :
Sac : 9 kg
A : 2 kg / 4 €
B : 2 kg / 2 €
C : 6 kg / 5 €
D : 3 kg / 6 €



Sac à dos

Rappels : Sac : 9 kg

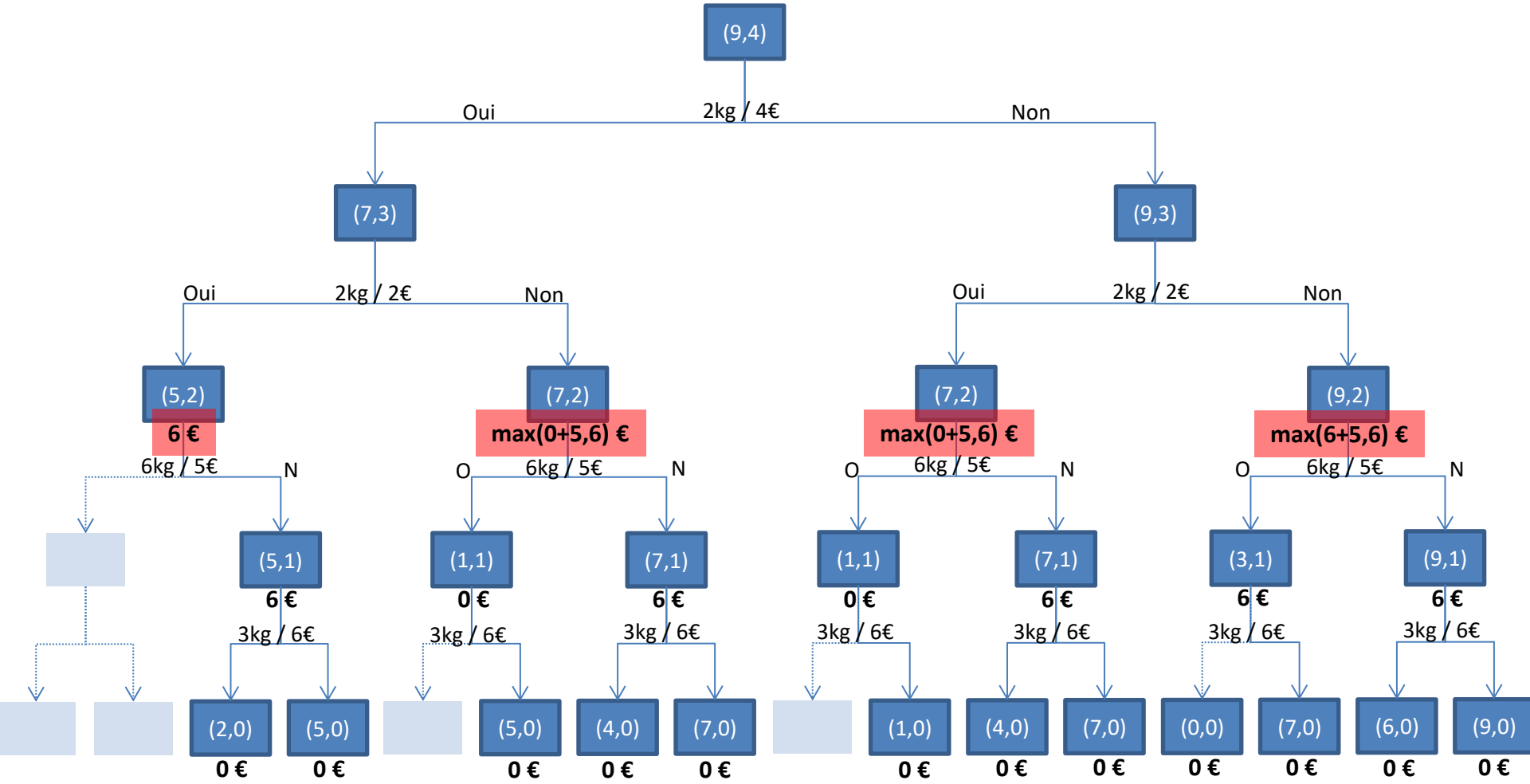
A : 2 kg / 4 €
B : 2 kg / 2 €
C : 6 kg / 5 €
D : 3 kg / 6 €



Sac à dos

Rappels : Sac : 9 kg

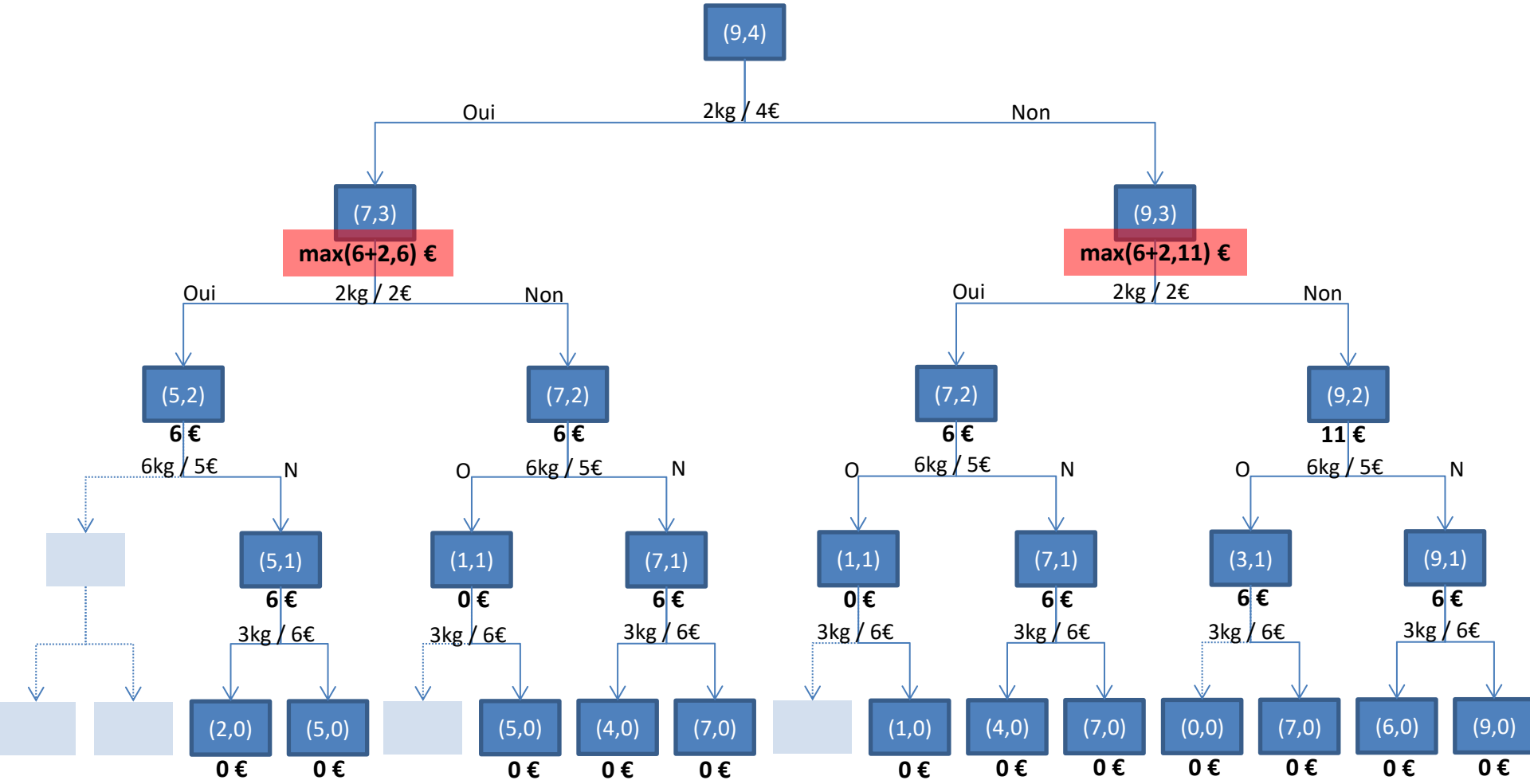
A : 2 kg / 4 €
B : 2 kg / 2 €
C : 6 kg / 5 €
D : 3 kg / 6 €



Sac à dos

Rappels : Sac : 9 kg

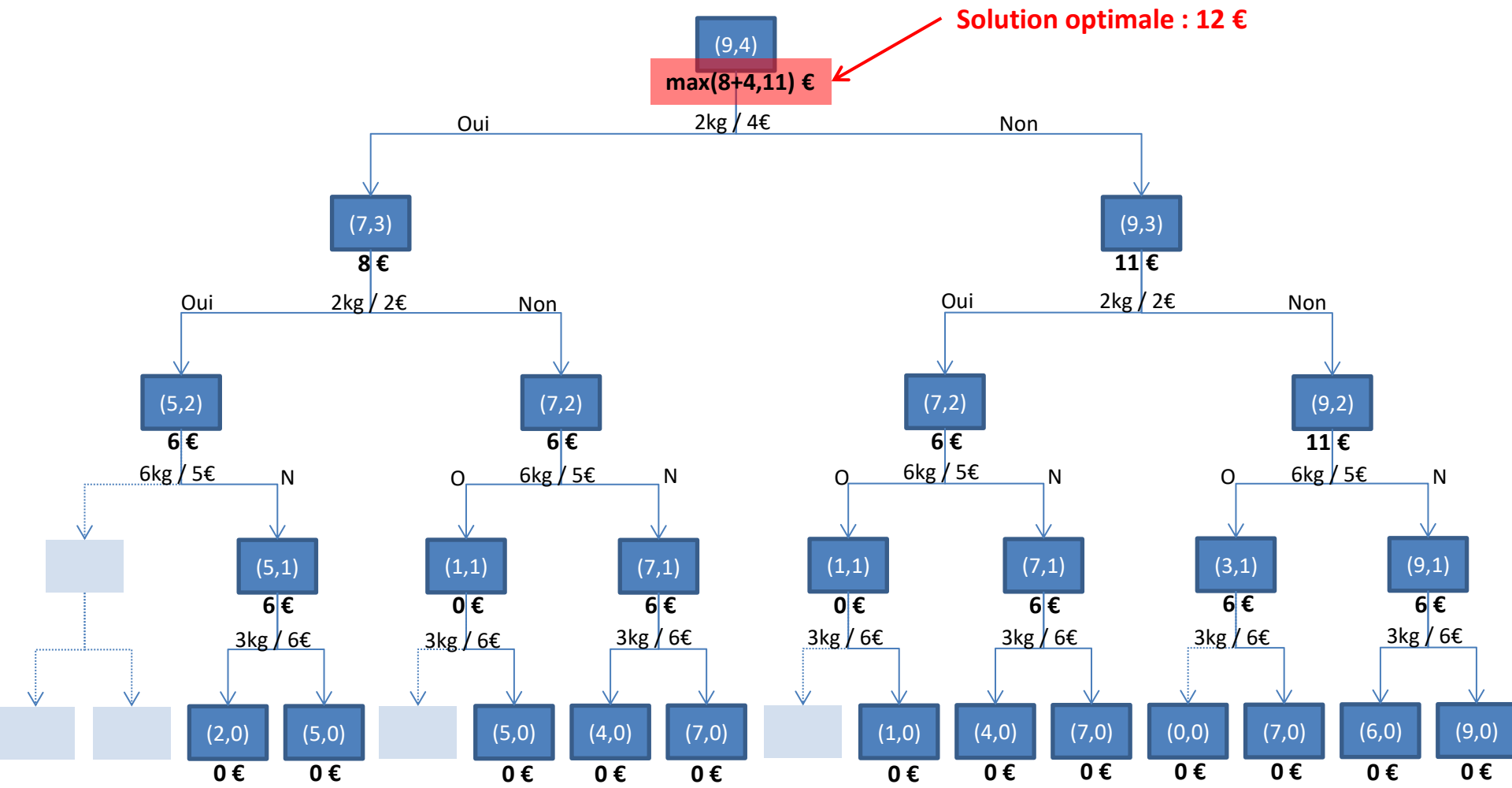
A : 2 kg / 4 €
B : 2 kg / 2 €
C : 6 kg / 5 €
D : 3 kg / 6 €



Sac à dos

Rappels : Sac : 9 kg

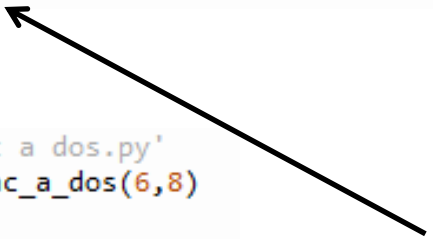
A : 2 kg / 4 €
B : 2 kg / 2 €
C : 6 kg / 5 €
D : 3 kg / 6 €



Sac à dos

Programme du sac à dos (récursif) en Python

```
1  def valeur_sac_a_dos(p,i):
2      if i==0:
3          return 0
4      if Objets[i][0] > p:
5          return valeur_sac_a_dos(p,i-1)
6      else:
7          return max(valeur_sac_a_dos(p,i-1),Objets[i][1]+valeur_sac_a_dos(p-Objets[i][0],i-1))
8
9  Objets=((0,0),(5,8),(1,7),(5,2),(1,9),(5,4),(1,3),(5,6),(1,5))
```



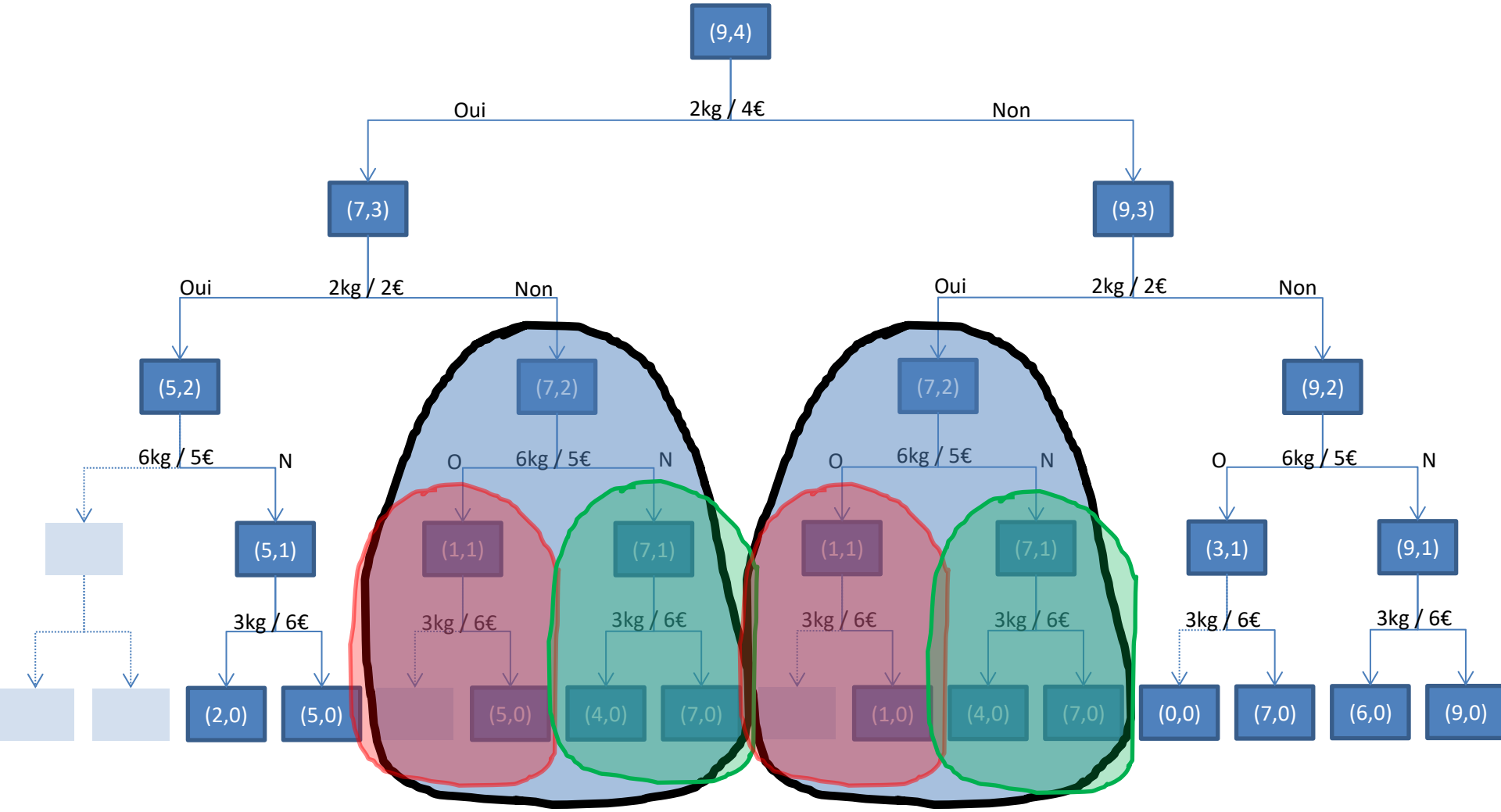
"Obligation" de mettre un premier objet dans la liste pour faire coïncider numéro de l'objet et place dans la liste

```
>>> %Run 'sac a dos.py'
>>> valeur_sac_a_dos(6,8)
24

>>> valeur_sac_a_dos(11,8)
32

>>> valeur_sac_a_dos(15,8)
38

>>> valeur_sac_a_dos(17,8)
38

>>> valeur_sac_a_dos(20,8)
42
```

Sac à dos

Rappels : Sac : 9 kg

A : 2 kg / 4 €
B : 2 kg / 2 €
C : 6 kg / 5 €
D : 3 kg / 6 €



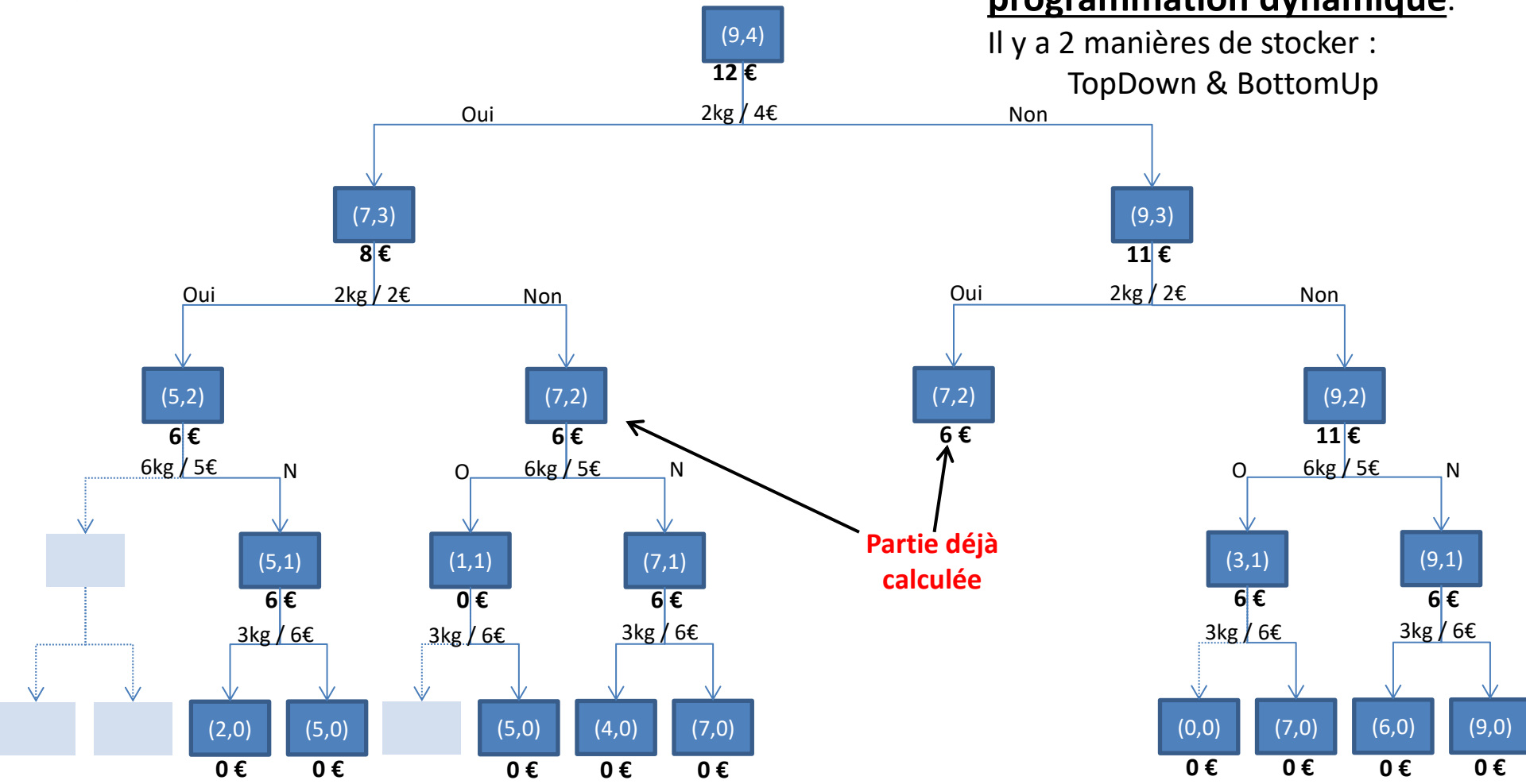
Sac à dos

Rappels : Sac : 9 kg
A : 2 kg / 4 €
B : 2 kg / 2 €
C : 6 kg / 5 €
D : 3 kg / 6 €

Principe : les calculs déjà faits ne sont pas à refaire !

A chaque fois qu'on calcule la valeur d'une case, on la stocke dans un tableau, on parle alors de **programmation dynamique**.

Il y a 2 manières de stocker :
TopDown & BottomUp



Sac à dos

Intérêt de la programmation dynamique :

Au départ, sur la ligne numéro k, il y a 2^k cases. Chaque case est de la forme (poids restant, $n-k$)

Mais le poids restant est dans $[\![0;p]\!]$, il ne peut y avoir au maximum que $p+1$ cases différentes. Un (très) grand nombre de cases vont être redondantes. Si elles ont été stockées en mémoire, on aura au maximum $p+1$ cases sur chaque ligne.

Bilan :

- Avant : n lignes de 2^k cases, donc $\sum_{k=0}^{n} 2^k = \Theta(2^{n+1})$ cases à traiter
- Après : au pire n lignes de $p+1$ cases , donc $\Theta(p \times n)$ cases à traiter

On a donc transformé un problème de complexité 2^n en du $p \times n$, c'est-à-dire linéaire pour chacune des entrées n et p.

Si p est du même ordre que n, on obtient du quadratique.

(le stockage des valeurs revient à remplir un tableau de taille $n \times p$)

donc cela ne change pas la complexité qui reste quadratique (généralement).

Un autre exemple de sac à dos

Avec davantage de redondances

Un autre exemple : Sac : 9 kg

A : 2 kg / 2 €
B : 2 kg / 1 €
C : 2 kg / 3 €
D : 4 kg / 6 €