Evaluation de NSI

Programmation dynamique

Exercice : récursivité et programmation dynamique

L'entreprise Sun and Steel produit et vend des barres métalliques de différentes longueurs dont voici les prix de vente (en fonction des longueurs):

Longueur (m)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Prix (euros)	1	5	8	9	10	17	17	20	24	30

Quand l'entreprise produit une barre de longueur *n*, elle a deux possibilités :

- Vendre la barre telle quelle.
- La découper en plusieurs parties et les vendre.

Par exemple, pour une barre de 3 mètres, l'entreprise peut agir ainsi :

- Vendre la barre de 3 mètres : 8 euros de revenus.
- Produire une barre de 2 mètres et une barre d'un mètre : 5 + 1 soient 6 euros de revenus.
- Produire trois barres de 1 mètre chacune : 3 * 1 soient 3 euros de revenus.

Question 1 : Donner la liste de toutes les possibilités pour l'entreprise lors de la vente d'une **barre de 4 mètres**. Quel est le **revenu optimal** ?

Dans la suite du problème, **une barre mesure désormais *n* mètr**es, avec *n* compris entre 1 et 10 mètres.

- Pour chaque longueur, on détermine ainsi le **revenu maximal** possible, noté ***r(n)***, ainsi, r(3) vaut 8 euros par exemple.
- Le prix de chaque barre métallique (non découpée) est notée ***p(n)***, ainsi, p(3) vaut 8 euros (voir le tableau).

L'idée est d'écrire un programme en Python permettant de calculer le revenu maximal engendré pour chaque longueur de barre métallique.

On peut remarquer qu'il est aussi possible de calculer **r(5)** de la méthode suivante :

r(5) = max(p(0) + r(5), p(1)+ r(4), p(2) + r(3), p(3) + r(2), p(4) + r(1), p(5) + r(0)) avec p(0) et r(0) valant zéro euro puisqu'il s'agit de barres métalliques de longueur nulle.

On peut généraliser au rang *n* avec la relation suivante :

r(n) = max(p(0) + r(n), p(1)+ r(n-1), p(2) + r(n-2), … , p(n-1) + r(1), p(n) + r(0))

Question 2 : En s'aidant des informations ci-dessus, **compléter** (sur la copie) la fonction *revenu_barre(n)*.

```
def revenu_barre(n):
    # Les indices de prix représentent la longueur
    # de la barre.
    prix = [0,1,5,8,9,10,17,17,20,24,30]

    # Cas d'arrêt
    if n == 0:
        return ...

    # Cas général
    else :
        revenu = 0
        for i in range(1, n+1) :
            revenu = max(revenu, prix[i] + ...)

        return revenu

# Jeu de test
print(revenu_barre(10)) # Attendu : 30
```

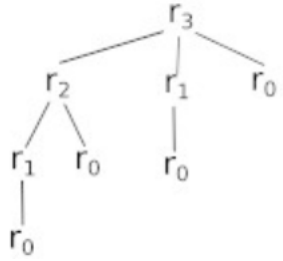
Question 3 : L'entreprise a modifié ses tarifs :

Longueur (m)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Prix (euros)	1	6	9	11	12	19	20	23	24	26

Que doit-on **changer** dans le programme ci-dessus pour qu'il corresponde aux nouveaux tarifs ?

Question 4 :

Voici l'arbre d'appel récursif du calcul de r(3) :



1/ **Combien de fois** est calculé r1 ? Quel type de problèmes cela pourra-t-il poser ?

2/ **Dessiner** l'arbre d'appels récursifs de r(4).

Question 5 : **Proposer** alors une version de la fonction précédente (on gardera la programmation dynamique descendante) pour corriger les problèmes précédents : on recopiera et complètera le programme sur la copie.

Une approche ascendante …

Question 6 : **Compléter** le programme suivant, version non récursive de la précédente :

```
def revenu_barre(n):
    # Les indices de prix représentent la longueur
    # de la barre.
    prix = [0,1,5,8,9,10,17,17,20,24,30]
    # Stocke les revenus optimisés
    revenu = [0]*11

    for i in range(1, n+1) :
        for j in range(...) :
            revenu[i] = max(revenu[i], prix[j] + ...)

    return ...

# Jeu de test
print(revenu_barre(5)) # Attendu : 13
```