École Centrale Lyon

Info TC1
Rapport

Rendu de monnaie

Élèves :
Hugo Puybareau
Luc Guidi

Enseignant :
Auday Berro

31 mai 2023

0.1 Hypothèse

On suppose que la liste des pièces S est triée dans l'ordre croissant. Si ce n'était pas le cas, il suffirait seulement de rajouter la ligne S.sort() au début des différentes fonctions.

1 Question 1

```
def Monnaie31(S,M):
    mat=np.zeros([len(S)+1,M+1]) #Matrice de zéros de la taille demandée
    #print(mat)
    for i in range (len(S)+1):
        for m in range(M+1):     #Parcours de la matrice
            if m==0:
                mat[i][m]=0      #Première colonne de 0
            elif i==0:
                mat[i][m]=inf    #Première ligne de inf sauf indice 0.0
            else :
                mat[i][m]=min(1+mat[i][m-S[i-1]] if m-S[i-1]>=0 else inf,mat[i-1][m] if i>=1 else inf) #remplissage
    return mat[len(S)][M] #Je fais le choix de renvoyer le nombre de pièce minimal au lieu de ce qui est précisé da
```

FIGURE 1 – Code de la fonction Monnaie31

Afin de déterminer le nombre minimal de pièces nécessaires pour atteindre un montant M en utilisant les pièces de la liste S, nous créons une fonction appelée "monnaie". Cette fonction prend en compte la liste des pièces et le montant à atteindre en tant qu'arguments.

Dans un premier temps, nous créons une matrice où les coefficients sont remplis selon une formule spécifiée dans l'énoncé. Le coefficient [i,m] représente le nombre minimal de pièces nécessaires pour atteindre la somme m en utilisant les pièces S[:i]. Chaque coefficient, à l'exception de ceux de la première ligne et de la première colonne, est calculé en prenant le minimum entre le nombre minimal de pièces utilisées pour atteindre m avec S[:i-1] et le nombre minimal de pièces utilisées pour atteindre m-S[i-1] avec S[:i-1], auquel on ajoute 1 pour tenir compte de l'utilisation de la pièce S[i-1] afin d'atteindre le montant m. En construisant cette matrice de cette manière, le coefficient situé à la dernière ligne et dernière colonne correspond au nombre minimal de pièces nécessaires pour atteindre le montant M.

```
In [4]: Monnaie31([1,3,4],7)
Out[4]: 2.0
```

FIGURE 2 – Test de la fonction Monnaie31 pour les pièces [1,3,4] et le montant 7€

2 Question 2

Dans cette deuxième question, nous souhaitons que notre fonction renvoie à la fois le nombre minimal de pièces utilisées pour atteindre le montant M, ainsi que les pièces spécifiquement utilisées. Pour cela, nous utiliserons une liste de la même taille que S, où le coefficient i représente le nombre de fois que la pièce S[i] est utilisée.

```python
def Monnaie32(S,M):
    n=len(S)
    mat=[[(0,[0]*n) for j in range (M+1)] for i in range (n+1)]    #On crée la matrice à partir de la longueur de S
    #print(mat)
    for i in range (len(S)+1):
        for m in range(M+1):
            if i==0:
                mat[i][m]=inf,[0]*n                    #Pour le cas ou il n'y a aucune pièce
            elif m==0:
                mat[i][m]=0,[0]*n                      #Pour le cas ou il n'y a rien à rembourser
            else:
                if m-S[i-1]>=0 :                       #la pièce S[i-1] est utilisée + 1 fois et le nombre de pièces utilisées prend +1 aussi quand on ajoute S[i-1] au montant de remboursement m-S[i-1]
                    nb_piece,T=mat[i][m-S[i-1]]        #T prend le détail des pièces
                    T1=T.copy()
                    T1[i-1]+=1
                    a=nb_piece+1
                else :
                    a,T1=inf,[0]*n
                if i>=1:                               #T2 et nb_piece ne varient pas si on rembourse m en prennant des pièces placées avant i-1 dans S
                    nb_piece,T=mat[i-1][m]
                    T2=T.copy()
                    b=nb_piece
                else:
                    b,T2=inf,[0]*n
                if a<=b:                               #Choix de la méthode qui prend le moins de pièces
                    mat[i][m]=a,T1
                else:
                    mat[i][m]=b,T2
    return mat[n][M]     #Si on test avec [1,3,4] pour 7€ on obtient bien 2,[0,1,1] soit 2 pièces dont une de 3 et une de 4 donc ça marche
```

FIGURE 3 – Code de la fonction Monnaie32

Pour ce faire, nous utilisons essentiellement le même algorithme que précédemment, à une différence près. Chaque coefficient [i,m] de la matrice comprend maintenant deux éléments : le nombre minimal de pièces nécessaires pour atteindre la somme m en utilisant les pièces S[:i], ainsi que la liste des pièces utilisées pour atteindre ce montant.

De plus, en fonction de la manière dont le nombre minimal de pièces est obtenu (soit en ajoutant à nouveau la pièce S[i-1] au nombre minimal de pièces utilisées pour atteindre m-S[i-1], soit en utilisant le nombre minimal obtenu avec les pièces S[:i-1]), nous ajustons la liste des pièces utilisées en conséquence.

Enfin, pour obtenir le résultat souhaité, nous renvoyons encore une fois le coefficient situé à la dernière ligne et dernière colonne de la matrice. Cela nous donne à la fois le nombre minimal de pièces nécessaires pour atteindre le montant M et la liste des pièces spécifiquement utilisées.

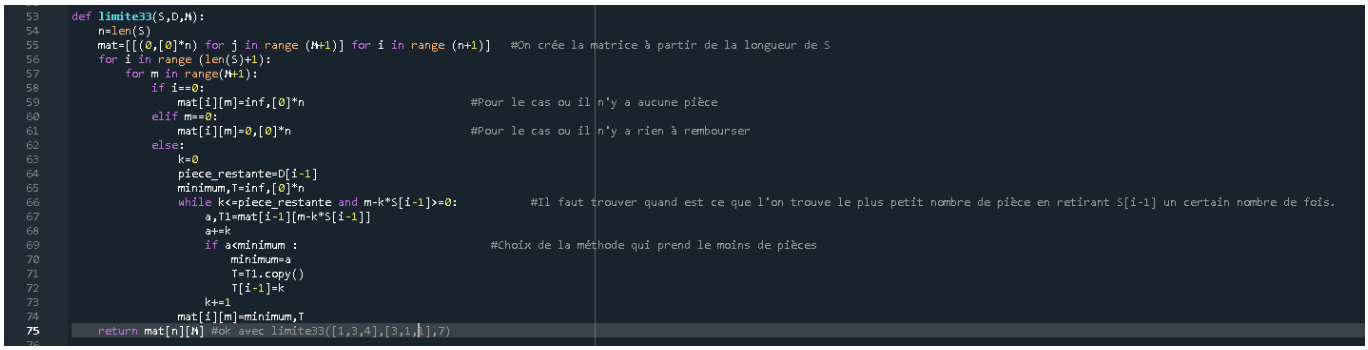
```
In [5]: Monnaie32([1,3,4],7)
Out[5]: (2, [0, 1, 1])
```

FIGURE 4 – Test de la fonction Monnaie32 pour les pièces [1,3,4] et le montant 7€

3 Question 3

Pour cette question, nous devons prendre en compte la disponibilité des pièces. Ainsi, nous ajoutons un argument supplémentaire par rapport aux fonctions précédentes : une liste de disponibilité des pièces, où le coefficient i correspond au nombre de pièces disponibles pour la pièce S[i].

Afin de déterminer le nombre minimal de pièces à utiliser pour rembourser le montant m avec les pièces S[:i], nous examinons différentes possibilités pour la valeur de k, allant de 0 jusqu'à la disponibilité de la pièce S[i-1]. Pour chaque valeur de k, nous calculons le nombre de pièces en ajoutant k pièces de S[i-1] au nombre minimal de pièces nécessaires



```python
def limite33(S,D,M):
    n=len(S)
    mat=[[(0,[0]*n) for j in range (M+1)] for i in range (n+1)]   #On crée la matrice à partir de la longueur de S
    for i in range (len(S)+1):
        for m in range(M+1):
            if i==0:
                mat[i][m]=inf,[0]*n                          #Pour le cas ou il n'y a aucune pièce
            elif m==0:
                mat[i][m]=0,[0]*n                            #Pour le cas ou il n'y a rien à rembourser
            else:
                k=0
                piece_restante=D[i-1]
                minimum,T=inf,[0]*n
                while k<=piece_restante and m-k*S[i-1]>=0:          #Il faut trouver quand est ce que l'on trouve le plus petit nombre de pièce en retirant S[i-1] un certain nombre de fois.
                    a,T1=mat[i-1][m-k*S[i-1]]
                    a+=k
                    if a<minimum :                           #Choix de la méthode qui prend le moins de pièces
                        minimum=a
                        T=T1.copy()
                        T[i-1]=k
                    k+=1
                mat[i][m]=minimum,T
    return mat[n][M] #ok avec limite33([1,3,4],[3,1,1],7)
```

FIGURE 5 – Code de la fonction Limite33

pour rembourser m-k*S[i-1].

En cherchant la valeur de l qui minimise ce nombre, nous ajustons ensuite la liste des pièces utilisées en conséquence, en ajoutant les pièces S[i-1] utilisées à la liste correspondante.

En résumé, nous prenons en compte la disponibilité des pièces en itérant sur différentes valeurs de l, de 0 à la disponibilité de la pièce S[i-1], afin de déterminer le nombre minimal de pièces à utiliser pour rembourser le montant m. Nous ajustons ensuite la liste des pièces utilisées en fonction de la valeur optimale de l trouvée.

```
In [7]: limite33([1,3,4],[3,1,1],7)
Out[7]: (2, [0, 1, 1])
```

FIGURE 6 – Test de la fonction Limite33 avec 3 pièces de 1€, 1 pièce de 3€ et 1 pièce de 4€, pour le montant 7€

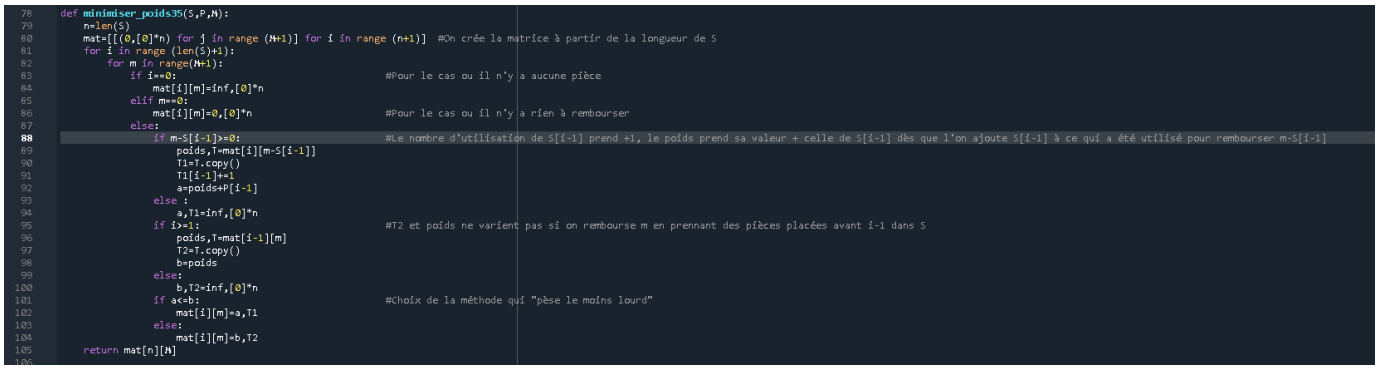
4 Question 4

En utilisant une matrice similaire à celle fournie dans l'énoncé, nous avons déterminé que le poids minimal pour un montant M de 7 est égal à 6,98.

5 Question 5

Dans cette partie, l'objectif n'est plus de minimiser le nombre de pièces utilisées, mais plutôt le poids total des pièces nécessaires pour rembourser le montant M. Par conséquent, notre fonction prend un argument supplémentaire, qui est une liste contenant les poids des pièces, où le coefficient i correspond au poids de la pièce S[i].

Pour résoudre ce problème, nous utilisons la même méthode que celle utilisée dans la question 2. La différence réside dans le choix de la méthode qui minimise maintenant le poids total plutôt que le nombre de pièces utilisées.



```
def minimiser_poids35(S,P,M):
    n=len(S)
    mat=[[(0,[0]*n) for j in range (M+1)] for i in range (n+1)]  #On crée la matrice à partir de la longueur de S
    for i in range (len(S)+1):
        for m in range(M+1):
            if i==0:                                #Pour le cas ou il n'y a aucune pièce
                mat[i][m]=inf,[0]*n
            elif m==0:
                mat[i][m]=0,[0]*n                   #Pour le cas ou il n'y a rien à rembourser
            else:
                if m-S[i-1]>=0:                     #Le nombre d'utilisation de S[i-1] prend +1, le poids prend sa valeur + celle de S[i-1] dès que l'on ajoute S[i-1] à ce qui a été utilisé pour rembourser m-S[i-1]
                    poids,T=mat[i][m-S[i-1]]
                    T1=T.copy()
                    T1[i-1]+=1
                    a=poids+P[i-1]
                else :
                    a,T1=inf,[0]*n
                if i>=1:                            #T2 et poids ne varient pas si on rembourse m en prennant des pièces placées avant i-1 dans S
                    poids,T=mat[i-1][m]
                    T2=T.copy()
                    b=poids
                else:
                    b,T2=inf,[0]*n
                if a<=b:                            #Choix de la méthode qui "pèse le moins lourd"
                    mat[i][m]=a,T1
                else:
                    mat[i][m]=b,T2
    return mat[n][M]
```

FIGURE 7 – Code de la fonction MinimiserPoids35

Par ailleurs, nous constatons qu'il existe une différence entre les pièces utilisées par le programme qui minimise le poids total et celui qui minimise le nombre de pièces utilisées (monnaie32). Par exemple, pour un montant M de 7 avec la liste des pièces S=[1,3,4] et les poids correspondants P=[10,50,100], les pièces utilisées ne sont pas les mêmes entre les deux programmes.

```
In [9]: minimiser_poids35([1,3,4],[10,50,100],7)
Out[9]: (70, [7, 0, 0])
```

FIGURE 8 – Test de la fonction minimiserpoids35 avec les valeurs précisées ci-dessus

6 Question 6

```
def Poids_Gloutonne36(S,P,M):
    L=[(P[i]/S[i],S[i],P[i]) for i in range (len(P))]
    L.sort()
    Mprime=M
    res=0
    while Mprime !=0:
        k=0
        while L[k][1]>Mprime:
            k=k+1
        s=L[k][1]
        p=L[k][2]
        res=res+p*(Mprime//s)
        Mprime=Mprime%s
    return res
```

FIGURE 9 – Code de la fonction PoidsGloutonne

Dans cette dernière question, nous devions implémenter l'algorithme Poidsgloutonne qui résout le problème de minimisation du poids de manière gloutonne. Cependant, nous

avons constaté que cet algorithme renvoie des solutions différentes de celles proposées par l'algorithme de Programmation Dynamique pour certains montants.

Le programme suivant a été conçu pour nous montrer pour quels montants les deux programmes diffèrent.

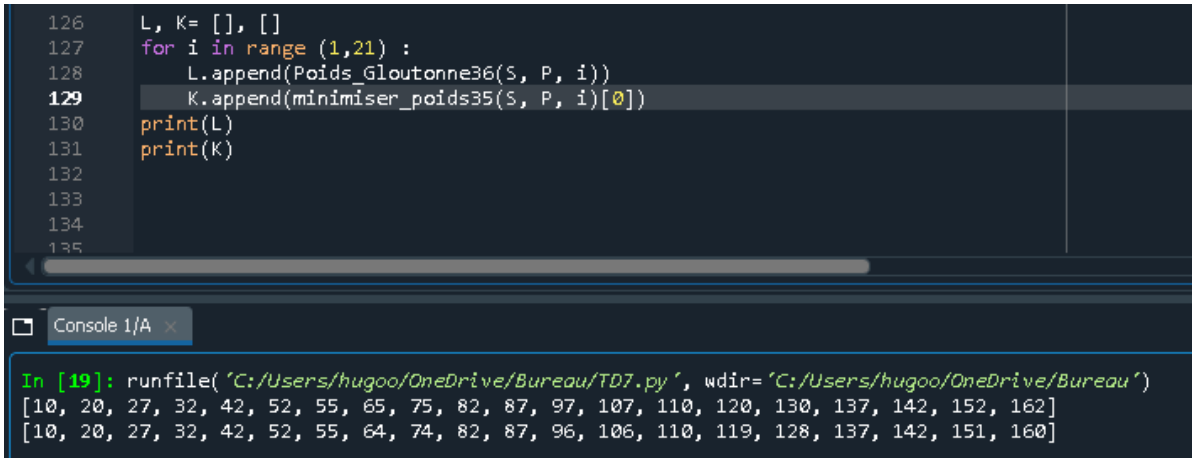


FIGURE 10 – Recherche de résultats différents

Les valeurs pour lesquelles les résultats des deux fonctions diffèrent sont donc 8, 9, 12, 13, 15, 16, 19, 20.