TD5

Commencez par récupérer le fichier `TD5_eleves.py` sur le réseau et enregistrez le sur votre répertoire pour pouvoir le modifier et l'utiliser.

Le problème

On suppose disposer de différentes pièces dont les valeurs sont stockées, dans l'ordre décroissant, dans une liste `Pieces`.
On souhaite trouver le moyen de rendre une certaine somme `s` en utilisant le moins de pièces possibles.
On supposera que le stock de pièces disponible est infini (et donc on n'aura toujours toutes les pièces disponibles pour ce rendu de monnaie).
On supposera également que la plus petite pièce possède une valeur 1 ce qui fait que le rendu de monnaie sera toujours possible.

1. Approche gloutonne

L'approche gloutonne est définie ainsi : pour rendre la monnaie, on commence par rendre la pièce dont la valeur est la plus grande possible (et on recommence sur la somme restante).

1. Écrire une fonction `glouton(s:int,P:list)->list` qui prend en argument la somme `s` à rembourser (supposée $\in \mathbb{N}$) et la liste `P` des valeurs des pièces, et qui renvoie la liste des pièces à utiliser pour ce remboursement.

2. Tester votre fonction lorsque `s = 49` dans le cas où le système de pièces est `Pieces1 = [50,20,10,5,2,1]` puis dans le cas où le système de pièces est `Pieces2 = [30,24,12,6,3,1]` et commenter le résultat.

2. Programmation dynamique

Si on note N_s le nombre de pièces minimal nécessaire pour rembourser la somme s et $p_0 > \cdots > p_{k-1}$ la valeurs des pièces disponibles. On a, pour $s \in \mathbb{N}$,

$$N_s = \begin{cases} 0 & \text{si } s = 0 \\ 1 + \min\{N_{s-p_i}, i \in [\![1,k]\!], s - p_i \geqslant 0\} & \text{sinon} \end{cases}$$

3. On considère le code suivant (*que vous pouvez récupérer sur le réseau*)

```
1 def N(s,P) :
2     if s == 0 :
3         return 0
4     else :
5         P_poss = []
6         for k in P :
7             if k <= s :
8                 P_poss.append(k)
9         return 1+min([N(s-k,P_poss) for k in P_poss])
```

a) Expliquer le rôle des lignes 5 à 8.

b) Calculer N_s pour $s = 49$ et le système de pièces `Pieces2` et commenter.

4. Écrire une fonction `dynamique(s:int,P:list)->int` qui prend en argument la somme `s` à rembourser (supposée $\geqslant 0$) et la liste `P` des valeurs des pièces, et qui renvoie le nombre minimal de pièces à utiliser pour ce remboursement. Cette fonction devra utiliser la mémoïsation avec un dictionnaire

Tester votre fonction pour $s = 49$ et le système de pièces `Pieces2` et commenter.

5. Écrire une fonction `BasEnHaut(s:int,P:list)->int` qui effectue le même calcul que la fonction précédente mais en version itérative. *On pourra introduire une liste `L` de longueur $s+1$ telle que `L[k]` $= N_k$.*

Évaluer la complexité de cette dernière fonction en fonction de s et $k =$ `len(P)`.

6. On suppose que la liste remplie par la fonction précédente est, avec $s = 20$ et le système de pièces `Pieces2` :

```
[0, 1, 2, 1, 2, 3, 1, 2, 3, 2, 3, 4, 1, 2, 3, 2, 3, 4, 2, 3, 4]
```

Comment peut-on à partir de cette liste déterminer la valeur des pièces à rendre ?

7. Modifier la fonction précédente de façon à ce qu'elle renvoie la liste des pièces pour un rendu optimal et non seulement le nombre de pièces pour un tel rendu.

8. Expliquer la relation de récurrence vérifiée par N_s au début de cette partie.