On souhaite comparer l'efficacité de la résolution du problème du rendu de monnaie par une méthode gloutonne et par programmation dynamique.

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 récursive `glouton(s:int,P:list)->list` qui prend en argument la somme `s` à rembourser (supposée ≥ 1) et la liste `P` des valeurs des pièces, et qui renvoie la liste des pièces à utiliser pour ce remboursement.
2. Tester votre fonction dans le cas où `Pieces = [10,5,2,1]` puis `Pieces = [30,24,12,6,3,1]` lorsque `s = 49`.

2. Programmation dynamique

3. Si on note N_s le nombre de pièces minimal nécessaire pour rembourser la somme `s` et $p_1, \dots, p_k$ la valeurs des pièces disponibles. Quelle est la valeur de N_s en fonction des termes $(N_i)_{i \leq s-1}$?
4. Quels sont les sous problèmes à résoudre pour déterminer la valeur de N_s ?
Donner un exemple de système de pièces et une valeur de `s` conduisant à un problème de chevauchement.
5. Écrire une fonction `dynamique(s:int,P:list)->list` qui prend en argument la somme `s` à rembourser (supposée ≥ 1) et la liste `P` des valeurs des pièces, et qui renvoie la liste des pièces à utiliser pour ce remboursement. Cette fonction devra utiliser la programmation dynamique et la mémorisation.
6. Tester votre fonction sur les mêmes cas que dans la partie 1.
7. Écrire une fonction `BasEnHaut(n:int,P:list)->list` qui effectue le même calcul que la fonction précédente mais en allant de bas en haut (programmation itérative).
8. Évaluer la complexité de cette dernière fonction.

3. Et si on enlève la pièce 1 ?

Imaginons qu'il n'y ait plus la pièce 1, certaines sommes ne seront plus remboursables (la somme 1 par exemple).

9. Que renvoie la démarche gloutonne dans le cas où `s = 8`, si `Pieces = [5,2]` ?
Et la démarche par programmation dynamique ?
10. Adapter votre fonction `dynamique` à cette situation ; on choisira par exemple de renvoyer la liste `[-1]` pour un rendu de monnaie impossible.