Lycée Jean Moulin **Draguignan**
NSI *Terminale* Année .....

TP - Rendu de monnaie - Programmation dynamique

Introduction

Le problème :
Vous avez à votre disposition un nombre illimité de pièces de 2 cts, 5 cts, 10 cts, 50 cts et 1 euro (100 cts). Vous devez rendre une certaine somme (rendu de monnaie). Le problème est le suivant : "Quel est le nombre minimum de pièces qui doivent être utilisées pour rendre la monnaie"
La résolution "gloutonne" de ce problème peut être la suivante :

- On prend la pièce qui a la plus grande valeur (il faut que la valeur de cette pièce soit inférieure ou égale à la somme restant à rendre)
- On recommence l'opération ci-dessus jusqu'au moment où la somme à rendre est égale à zéro.

? Exercice 1 :

Appliquer cette méthode pour une somme de 1€77 (177cts) à rendre.

.....
.....
.....
.....
.....
.....

Au final il faut rendre ..... pièces

? Exercice 2 :

Appliquer cette méthode à la somme de 11 centimes

.....
.....
.....

Au final .....

Évidemment, le fait que notre algorithme glouton ne soit pas "capable" de trouver une solution ne signifie pas qu'il n'existe pas de solution

? Exercice 3 :

Donner une solution pour la somme de 11 centimes

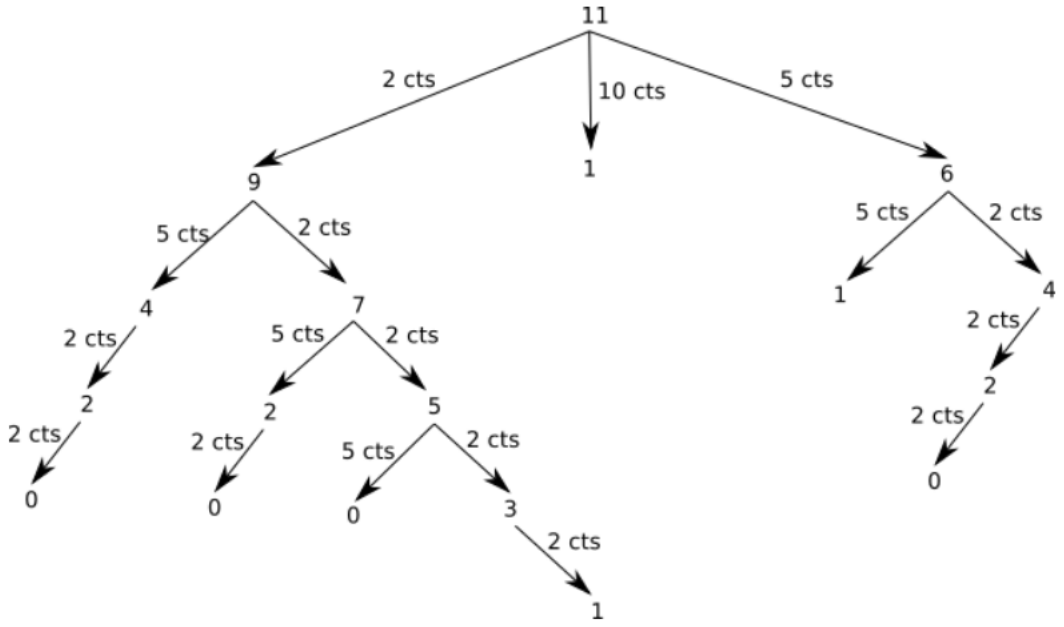
.....
.....

Mise au point d'un algorithme (récursif)

Soit X la somme à rendre, on notera $nb(X)$ le nombre minimal de pièces nécessaires pour rendre la somme X.
On note p_1,p_2,p_3,...,p_n les pièces à notre disposition.

Remarque :
Si on est capable de rendre la somme de X cts avec $nb(X)$ pièces, on est alors capable de rendre la somme de $X - p_i$ avec $nb(X - p_i) + 1$ pièces (avec la valeur de p_i inférieure à X)
On a :
Si $X = 0$ alors $nb(X) = 0$
Si $X > 0$ alors $nb(X) = nb(X - pi) + 1$ avec $1 \leq i \leq n$ et $p_i \leq X$
Plus précisément :
Si $X > 0$ alors $nb(X) = min(nb(X - pi)) + 1$ avec $1 \leq i \leq n$ et $p_i \leq X$
Le "min" présent dans la formule de récurrence exprime le fait que le nombre de pièces à rendre pour une somme $X - p_i$ *doit être le plus petit possible.*

Examinons en détails le processus pour la somme de 11cts à l'aide d'un arbre



Plusieurs remarques s'imposent :

- Tous les cas sont "traités" (quand un algorithme "traite" tous les cas possibles, on parle souvent de méthode de "force brute".
- Pour certains cas, on se retrouve dans une "impasse" (cas où on termine par un "1")
- La profondeur minimum de l'arbre (avec une feuille 0) est de 4, la solution au problème est donc 4 (il existe plusieurs parcours : (5,2,2,2), (2,5,2,2)... qui donne à chaque fois 4)

Voici le programme qui réalise ce processus :

```
def rendu_monnaie_rec(P,X):
  if X==0:
    return 0
  else:
    mini = 1000
  for i in range(len(P)):
    if P[i]<=X:
      nb = 1 + rendu_monnaie_rec(P,X-P[i])
      if nb<mini:
        mini = nb
  return mini

pieces = (2,5,10,50,100)
```

Pour être sûr de renvoyer le plus petit nombre de pièces, on attribue dans un premier temps la valeur 1000 à la variable mini (cette valeur 1000 est arbitraire, il faut juste une valeur suffisamment grande : on peut partir du principe que nous ne rencontrerons jamais de cas où il faudra rendre plus de 1000 pièces), ensuite, à chaque appel récursif, on "sauvegarde" le plus petit nombre de pièces dans cette variable mini.

À FAIRE 1:

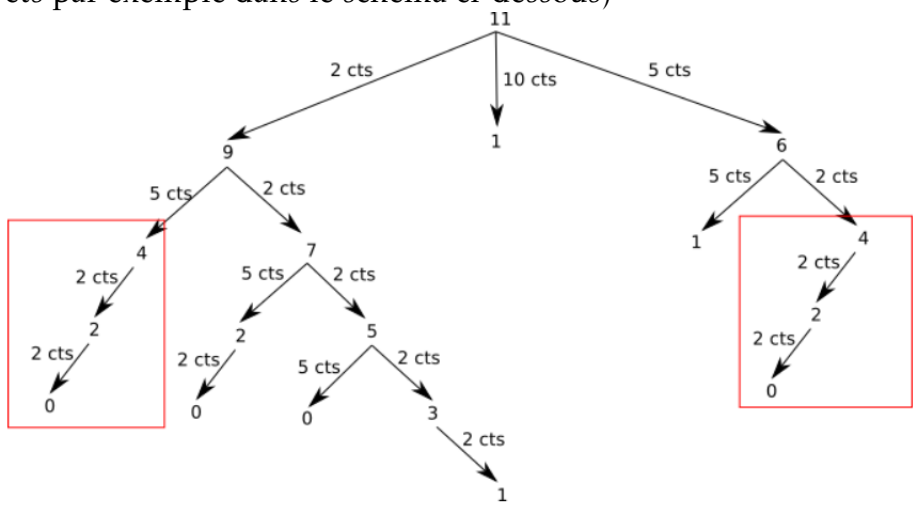
Écrire ce programme et le faire fonctionner pour la somme de 11cts.
Faire évoluer la somme à rendre 12 cts, 15 cts, ...
À partir de quelle somme le programme est-il visiblement lent ?
.....

Comme vous l'avez sans doute constaté le programme ne permet pas toujours d'obtenir une solution, pourquoi ?
Parce que les appels récursifs sont trop nombreux, on dépasse la capacité de la pile.

---

la méthode dynamique

En y regardant de plus près, on s'aperçoit que certains calculs se font plusieurs fois (le rendu de 4 cts par exemple dans le schéma ci-dessous)



On va pouvoir appliquer la même méthode que pour Fibonacci :
la programmation dynamique.

l'algorithme :

```
Données : X est un entier (somme à rendre en cts)
P ← pièces
m est un tableau contenant X+1 zéros
fonction rendu_monnaie_rec_ mem(P,X,m):
Si X = 0 alors
    renvoyer 0
Sinon si  m[X] > 0 :
    renvoyer m[X]
Sinon
    mini ← 1000
    Pour i allant de 0 à len(P) faire
        Si  P[i] <= X alors
            nb ← 1 + rendu_monnaie_rec_mem(P,X-P[i],m)
            Si  nb < mini alors
                mini ← nb
                m[X] ← mini
renvoyer mini
```

À FAIRE 2:

Implémenter cet algorithme et vérifier que cette fois-ci, on trouve bien les solutions pour des sommes à rendre comme 177cts, 289cts.