Chap 13. Programmation dynamique

Livre p 233 – Chap 14 Programmation dynamique
Livre p 249 – Chap 15 Recherche textuelle

1. « Du haut vers le bas » : La mémoïsation

a. Principe

Lors de la résolution de problème par un algorithme récursif, une situation « explosive » est apparue : si on appelle plusieurs fois la fonction récursive, on se trouve avec une complexité exponentielle et le temps de traitement devient exagérément grand même pour un cas relativement simple…
La cause de cette situation : les appels récursifs de la fonction ne gardent pas en mémoire les calculs déjà effectué !
La solution : conserver en mémoire tous les résultats obtenus afin de pouvoir les réutiliser sans avoir besoin de les recalculer.
Remarque : En Python, il existe un module à charger pour garder en mémoire tous les appels récursifs, il suffit d'insérer en début de programme les deux lignes suivantes :

```
from functools import lru_cache
@lru_cache(maxsize=None)
```

b. Exemple : Suite de Fibonacci

- Version récursive sans mémoïsation :
  32 secondes pour calculer `fibo(40)`
  avec 165 580 141 appels à la fonction récursive !

```
def fibo(n) :
    """ Calcul des termes de la suite de Fibonacci """
    if n == 0 or n == 1 :
        return 1
    else :
        return fibo(n-2) + fibo(n-1)
```

- Version récursive avec mémoïsation :
  0 seconde pour calculer `fibo(40)`
  avec seulement 79 appels à la fonction récursive.

```
memo = {0:1, 1:1} # Utilisation d'un dictionnaire
def fibo(n) :
    """ Calcul des termes de la suite de Fibonacci """
    if n in memo :
        return memo[n]
    else :
        memo[n] = fibo(n-2) + fibo(n-1)
        return memo[n]
```

c. Exemple : Coefficients binomiaux

- Version récursive sans mémoïsation :
  150 secondes pour calculer `combinaisons(35, 12)`
  avec 1 668 903 599 appels à la fonction récursive !
- Version récursive avec mémoïsation :
  32 seconde pour calculer `combinaisons(35, 12)`
  avec seulement 553 appels à la fonction récursive.

2. « Du bas vers le haut » : La programmation dynamique

a. Principe

Plutôt que de reprendre l'algorithme récursif, on peut aussi utiliser un algorithme itératif en partant du cas le plus simple et en calculant tous les cas jusqu'à aboutir au cas cherché en utilisant les cas précédemment calculés.

Remarque : On va peut-être calculer des cas inutiles pour notre problème, mais ce n'est pas grave !

b. Exemple : Suite de Fibonacci

```
def fibo(n) :
    """ Calcul des termes de la suite de Fibonacci """
    u = [1, 1]
    for i in range(2, n+1) :
        u.append(u[i-2] + u[i-1])
    return u[n]
```

3. Rendu de monnaie

a. Rappel version gloutonne

Avec le problème du rendu de monnaie, si la somme est grande, le nombre de cas à étudier est trop important pour envisager de les comparer tous, on s'est alors tourné vers un algorithme glouton qui donne une solution intéressante mais pas toujours optimale. (Exemple : un rendu de 38 avec [1, 6, 10])

b. Version dynamique pour rendre un minimum de pièces.

En fait, il n'est pas nécessaire de calculer tous les cas, mais il suffit de connaître les cas optimaux pour toutes les sommes inférieures ou égales à la somme cherchée.

Exemple avec [1, 6, 10] :

S	1	6	10
0	0	0	0
1	1	0	0
2	2	0	0
3	3	0	0
4	4	0	0
5	5	0	0
6	0	1	0
7	1	1	0
8	2	1	0
9	3	1	0

S	1	6	10
10	0	0	1
11	1	0	1
12	0	2	0
13	1	2	0
14	2	2	0
15	3	2	0
16	0	1	1
17	1	1	1
18	0	3	0
19	1	3	0

S	1	6	10
20	0	0	2
21	1	0	2
22	0	2	1
23	1	2	1
24	0	4	0
25	1	4	0
26	0	1	2
27	1	1	2
28	0	3	1
29	1	3	1

S	1	6	10
30	0	0	3
31	1	0	3
32	0	2	2
33	1	2	2
34	0	4	1
35	1	4	1
36	0	1	3
37	1	1	3
38	0	3	2
39	1	3	2

c. Version dynamique pour compter le nombre de façons de rendre la monnaie.

On peut même compter le nombre total de façons de rendre la monnaie pour une somme cherchée.
Pour cela, on compte le nombre de façon de rendre la monnaie pour toutes les sommes inférieures ou égales à la somme cherchée en utilisant : 1, 2, 3, 4 types de pièces…(voir [CG] « Ways to make change »)

Exemple avec [1, 2, 5, 10] :

S	1	2	5	10
0	1	1	1	1
1	1	1	1	1
2	1	2	2	2
3	1	2	2	2
4	1	3	3	3
5	1	3	4	4
6	1	4	5	5
7	1	4	6	6
8	1	5	7	7
9	1	5	8	8

S	1	2	5	10
10	1	6	10	11
11	1	6	11	12
12	1	7	13	15
13	1	7	14	16
14	1	8	16	19
15	1	8	18	22
16	1	9	20	25
17	1	9	22	28
18	1	10	24	31
19	1	10	26	34

S	1	2	5	10
20	1	11	29	49
21	1	11	31	43
22	1	12	34	49
23	1	12	36	52
24	1	13	39	58
25	1	13	42	64
26	1	14	45	70
27	1	14	48	76
28	1	15	51	82
29	1	15	54	88