NSI · TERMINALE / COURS

La programmation dynamique

Astrolab Cours · astrolabcours.fr

01 Le problème

La suite de Fibonacci : $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$. La fonction récursive directe met plusieurs secondes pour F_{40} et des années pour F_{100}. Pourtant le calcul à la main est rapide. Où passe le temps ?

01 L'idée : le carnet de calculs

Pour calculer F_{40}, la fonction naïve calcule F_{38} dans une branche, puis le recalcule entièrement dans une autre, et ainsi de suite : des milliards de fois les mêmes calculs. Avec un carnet où l'on note chaque résultat la première fois, on n'a jamais à le refaire : on regarde dans le carnet.

01 Le principe

La **programmation dynamique** s'applique quand un problème se découpe en **sous-problèmes qui se répètent**. On calcule chaque sous-problème **une seule fois** et on **mémorise** son résultat.

01 Version descendante : la mémoïsation

```
memo = {}

def fibo(n):
    if n <= 1:
        return n
    if n not in memo:
        memo[n] = fibo(n - 1) + fibo(n - 2)
    return memo[n]
```

Chaque valeur de 0 à n est calculée une fois : environ n opérations au lieu d'un nombre qui double à chaque rang.

Version ascendante : le tableau

On part des petits cas et on remplit un tableau jusqu'au cas voulu :

```
def fibo(n):
    f = [0, 1]
    for i in range(2, n + 1):
```

© Astrolab Cours · astrolabcours.fr/nsi/terminale/programmation-dynamique/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits

```
        f.append(f[i - 1] + f[i - 2])
    return f[n]
```

Le rendu de monnaie optimal

Avec des pièces de 1, 6 et 10, comment rendre 12 avec le moins de pièces ? L'algorithme glouton prend d'abord la plus grosse pièce : 10 + 1 + 1, soit 3 pièces. Or 6 + 6 n'en utilise que 2.

En programmation dynamique, `nb[s]` est le nombre minimal de pièces pour la somme `s` :

```
def rendu(pieces, somme):
    nb = [0] + [float("inf")] * somme
    for s in range(1, somme + 1):
        for p in pieces:
            if p <= s:
                nb[s] = min(nb[s], nb[s - p] + 1)
    return nb[somme]
```

On essaie toutes les pièces pour chaque somme, mais chaque sous-somme n'est calculée qu'une fois.

Exemple guidé

On monte un escalier de 5 marches, une ou deux marches à la fois. Combien y a-t-il de façons de monter ?

1. **Découper** : pour arriver à la marche n, le dernier pas vient de $n-1$ ou de $n-2$. Donc $w(n) = w(n-1) + w(n-2)$, avec $w(1) = 1$ et $w(2) = 2$.
2. **Remplir le tableau** dans l'ordre croissant : $w(3) = 2 + 1 = 3$, $w(4) = 3 + 2 = 5$, $w(5) = 5 + 3 = 8$. Il y a 8 façons.
3. **En Python** :

```
def escalier(n):
    w = [0, 1, 2] + [0] * n
    for k in range(3, n + 1):
        w[k] = w[k - 1] + w[k - 2]
    return w[n]
```

Chaque case est calculée une seule fois, à partir de deux cases déjà remplies.

Retour au problème

La récursion directe recalcule sans cesse les mêmes valeurs : pour F_5, elle fait déjà 15 appels. En notant chaque résultat, F_{100} ne demande que 100 additions, exactement comme à la main.

Les pièges

- Appliquer la mémoïsation à un problème dont les sous-problèmes ne se répètent pas : aucun gain.
- Croire que le glouton donne toujours la meilleure réponse au rendu de monnaie : cela dépend des pièces.
- Une variable `memo` partagée garde ses valeurs d'un appel à l'autre : c'est voulu, mais attention si le problème change.

© Astrolab Cours · astrolabcours.fr/nsi/terminale/programmation-dynamique/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits

© Astrolab Cours · astrolabcours.fr/nsi/terminale/programmation-dynamique/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits