STRUCTURES DE DONNÉES ET ALGORITHMES · CHAPITRE 22

Mémoriser les solutions de sous-problèmes

Calculez un nombre minimal de pièces par états, récurrence et table. Comparez le résultat à une stratégie gloutonne qui peut échouer.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Définir un état, une récurrence et un ordre de calcul.
- Un état doit désigner un sous-problème sans ambiguïté.
- La récurrence utilise des états plus petits et le cas de base zéro.

Prérequis. Avoir étudié les chapitres DSA précédents ; connaître recherche, tris et raisonnement sur les coûts.

Partir d'un besoin

Pour former 6 avec des pièces de valeurs 1, 3 et 4 en quantité illimitée, prendre toujours la plus grande donne 4 + 1 + 1. Pourtant 3 + 3 utilise moins de pièces. Comment explorer les choix sans résoudre sans cesse les mêmes sous-problèmes ?

Définir la structure et les conventions

La **programmation dynamique** mémorise la meilleure réponse à des sous-problèmes qui se recouvrent. Un **état** définit exactement un sous-problème. Ici, best[a] est le nombre minimal de pièces pour une somme entière a. Le cas de base vaut best[0] = 0 ; un état impossible vaut l'infini.

Pour a > 0, la dernière pièce possède une valeur c ≤ a. La récurrence est `best[a] = min(best[a − c] + 1)` sur les pièces autorisées. Il faut seulement utiliser un sous-état atteignable. Comme les valeurs des pièces sont des entiers strictement positifs, a - c est plus petit que a. Remplir la table par sommes croissantes respecte les dépendances.

Python utilise l'indice a pour la somme a. MATLAB décale d'une case : la somme a est stockée à l'indice a + 1, car les indices commencent à 1. Cette correspondance est essentielle pour le cas zéro. Les deux versions reçoivent une somme entière non négative et des pièces entières strictement positives.

Suivre les états

Table pour les pièces 1, 3 et 4

Somme a	0	1	2	3	4	5	6
Nombre minimal	0	1	2	1	1	2	2

Comparer les réalisations

Le résultat 2 correspond à deux pièces de 3. min choisit la plus petite des deux valeurs proposées ; notre pseudo-code écrit MIN pour cette opération. Ajouter 1 à l'infini laisse l'infini dans les versions exécutables, ce qui conserve un état impossible. MATLAB attend ici un vecteur ligne de pièces.

Pseudo-code

```
// example.pseudo
// minimum_coins
// best[0] = 0; other states initially infinity.
FOR value FROM 1 TO amount INCLUSIVE DO
    FOR EACH coin IN coins DO
        IF coin <= value THEN
            best[value] <- MIN(best[value], best[value - coin] + 1)
        END IF
    END FOR
END FOR
RETURN best[amount]
```

Python

```
# minimum_coins.py
def minimum_coins(coins, amount):
    best = [float("inf")] * (amount + 1)
    best[0] = 0
    for value in range(1, amount + 1):
        for coin in coins:
            if coin <= value:
                best[value] = min(best[value], best[value - coin] + 1)
    return best[amount]
result = minimum_coins([1, 3, 4], 6)
print(result)
```

MATLAB

```
% minimum_coins_demo.m
result = minimum_coins([1, 3, 4], 6);
disp(result);
function result = minimum_coins(coins, amount)
    best = inf(1, amount + 1);
    best(1) = 0;
    for value = 1:amount
        for coin = coins
            if coin <= value
                best(value + 1) = min(best(value + 1), best(value - coin + 1) +
1);
            end
        end
    end
    result = best(amount + 1);
end
```

Justifier et compter

Pour k valeurs de pièces et une somme A, le calcul effectue O(Ak) travail et utilise O(A) mémoire, sous un modèle de coût constant des opérations. Ce coût dépend de la valeur A, pas seulement du nombre de chiffres qui l'encodent : on parle de coût pseudo-polynomial. Une très grande somme peut donc demander une table trop volumineuse.

Prévoir puis expérimenter

Testez la somme 0 (résultat 0), les pièces [3, 4] avec somme 2 (impossible, infini) et [2] avec somme 6 (3). Avant d'exécuter, identifiez quels sous-états alimentent best[6].

Exercice corrigé

▼ Comment retrouver les pièces, pas seulement leur nombre ?

Lors d'une amélioration de best[a], conservez la pièce choisie pour cet état. En partant de A, soustrayez cette pièce puis répétez jusqu'à zéro ; ne lancez pas la reconstruction si best[A] est infini.

À retenir

- Un état doit désigner un sous-problème sans ambiguïté.
- La récurrence utilise des états plus petits et le cas de base zéro.
- Le coût O(Ak) dépend de la valeur de la somme A : il est pseudo-polynomial.

Version web : https://www.clearcode.fr/courses/programming/dsa/strategies/dynamic-programming