STRUCTURES DE DONNÉES ET ALGORITHMES · CHAPITRE 23

# Explorer un choix puis restaurer l’état

Construisez une recherche de sous-ensembles par retour sur trace. Séparez choix, appel récursif, restauration et copie d’une solution.

Par Ali ZAINOUL · Cours gratuit

### Ce que vous allez apprendre

- Explorer des choix puis restaurer l’état partagé.
- Chaque branche fixe un choix et progresse vers un cas terminal.
- Après un retour, restaurez l’état partagé avant d’explorer une autre branche.

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

## Partir d’un besoin

Parmi les valeurs 2, 3 et 5, quels sous-ensembles ont une somme de 5 ? Conserver seulement un meilleur choix ne suffit pas si l’on veut toutes les solutions.

## Définir la structure et les conventions

Le **retour sur trace**, ou backtracking, explore des choix successifs puis annule leurs effets pour explorer une autre branche. Ici, à chaque position, on choisit de prendre ou de ne pas prendre la valeur. Chaque position est utilisée au plus une fois ; les entrées sont des entiers strictement positifs et la cible est un entier non négatif.

Un état contient l’indice de la prochaine position, la somme restante et le choix partiel. Si le reste est zéro, le choix est une solution. Si la fin est atteinte sans somme correcte, cette branche échoue. La positivité permet de ne pas prendre une valeur supérieure au reste : aucun nombre

négatif ultérieur ne pourrait compenser.

Python utilise une liste chosen partagée : append ajoute un choix, pop le retire après le retour. La copie chosen.copy() conserve une solution indépendante avant les futurs changements. MATLAB transmet une nouvelle suite chosen pour la branche qui prend la valeur et réunit les cellules de solutions. Il ne réalise donc pas la même restauration par mutation, mais explore les mêmes choix.

## Suivre les états

**Branches pour la cible 5**

| Choix | Reste | Résultat |
|---|---|---|
| Prendre 2, prendre 3 | 0 | Solution [2, 3] |
| Prendre 2, ignorer 3, ignorer 5 | 3 | Échec |
| Ignorer 2, ignorer 3, prendre 5 | 0 | Solution [5] |

## Comparer les réalisations

Les solutions sont [2, 3] et [5]. La branche qui prend une valeur est explorée avant celle qui l'ignore. Le script MATLAB utilise des cellules pour stocker des solutions de longueurs différentes. Des valeurs identiques à des positions distinctes peuvent produire des solutions de mêmes valeurs : le contrat distingue les positions et ne promet pas une déduplication.

**Pseudo-code**

```
// example.pseudo
// subset_solutions
// Explore two branches: take current value, then skip it.
// Advance the index in both branches.
// Restore a shared choice list after returning from the take branch.
// Store a COPY when remaining == 0.
```

**Python**

```python
# subset_solutions.py
def subset_solutions(values, target):
    solutions = []
    chosen = []
    def explore(index, remaining):
        if remaining == 0:
            solutions.append(chosen.copy())
            return
        if index == len(values):
            return
        if values[index] <= remaining:
            chosen.append(values[index])
            explore(index + 1, remaining - values[index])
            chosen.pop()
        explore(index + 1, remaining)
    explore(0, target)
    return solutions
solutions = subset_solutions([2, 3, 5], 5)
print(solutions)
```

MATLAB

```
% subset_solutions_demo.m
solutions = explore([2, 3, 5], 1, 5, []);
disp(solutions);
function solutions = explore(values, index, remaining, chosen)
    if remaining == 0
        solutions = {chosen};
        return;
    end
    if index > numel(values)
        solutions = {};
        return;
    end
    solutions = {};
    if values(index) <= remaining
        solutions = explore(values, index + 1, remaining - values(index),
[chosen, values(index)]);
    end
    skipped = explore(values, index + 1, remaining, chosen);
    solutions = [solutions, skipped];
end
```

## Justifier et compter

Il existe $2^n$ choix de sous-ensembles de n positions au pire. Explorer et recopier des solutions peut demander $O(n \times 2^n)$ temps et autant de stockage pour la sortie dans le pire cas ; l'état récursif Python hors sortie utilise O(n) mémoire. La version MATLAB ajoute des copies de chemins et de collections de résultats : son allocation ne suit pas la même garantie. L'élagage réduit certains cas, sans supprimer le pire cas exponentiel.

## Prévoir puis expérimenter

Testez une cible impossible, une liste vide et une cible nulle. Pour une cible nulle, l'unique solution avec des valeurs positives est le choix vide. Les appels avancent toujours l'indice ; l'arrêt ne dépend donc pas uniquement de la somme restante.

## Exercice corrigé

**▼ Pourquoi conserver chosen directement dans solutions serait-il incorrect en Python ?**

Toutes les entrées pourraient désigner la même liste ensuite modifiée par pop et append. Chaque solution doit figer l'état du choix au moment de sa découverte.

## À retenir

- Chaque branche fixe un choix et progresse vers un cas terminal.
- Après un retour, restaurez l'état partagé avant d'explorer une autre branche.
- Copiez une solution avant de modifier le choix courant ; l'exploration peut rester exponentielle.

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