Programmation Dynamique

Denis Trystram – oct. 2019

Case study : a grand mother wants to give some monney to her two grand children (Alice and Bob). She has some coins in her wallet, and she would like to give the same amount to each of them. If it is impossible, she would like to be as fair as possible.

La méthode

Nous avons vu qu'il était possible de résoudre ce problème de façon exacte avec une énumération « brute force » ou un Branch-and-Bound (plus efficace). Pour déterminer toutes les combinaisons possibles, il existe un moyen que de lister toutes les configurations de façon exhaustive (même avec énumération implicite) :
D'une certaine manière, on traite le problème « à l'envers » du B-and-B.

Prenons un cas concret. Dans un sac de pièces, nous avons : 5, 9, 3, 8, 2, 5.
La somme est égale à 32, on doit donc répartir 16 à chacun (si c'est possible, sinon, le plus proche possible de 16). On notera S cette somme dans la suite.

Nous allons rechercher une sous-liste de pièces optimale (c'est-à-dire dont la somme se rapprochera le plus possible de la moitié de la somme de toutes les pièces).

Avec la première pièce, il n'y a qu'un seul assemblage possible. Avec les deux premières pièces, quels sont les assemblages possibles ? Quelle est la meilleure solution ?
Et ainsi de suite jusqu'à épuisement des pièces.

En pratique, on peut connaître les assemblages possibles avec deux pièces à partir des assemblages possibles avec une pièce, et ainsi de suite. On constate que, pour travailler, le programme a besoin de connaître les données déjà trouvées et comment il les a déterminées. C'est le fondement même de la programmation dynamique : **éliminer les recalculs en mémorisant ce qui a déjà été calculé**.

Mise en oeuvre

Passons maintenant à l'implémentation qui utilise un tableau booléen T où Chaque ligne représente les combinaisons possibles avec les i premières pièces.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1re pièce	F	F	F	F	V	F	F	F	F	F	F	F	F	F	F	F
2 premières pièces	F	F	F	F	V	F	F	F	V	F	F	F	F	V	F	F
3 premières pièces	F	F	V	F	V	F	F	V	V	F	F	V	F	V	F	F
4 premières pièces	F	F	V	F	V	F	F	V	V	F	V	V	V	V	F	V

Dans notre cas, on s'arrête à la ligne 4 car la solution idéale a été trouvée. Sinon, on continuerait à remplir le tableau jusqu'à la dernière ligne, et nous connaîtrons de ce fait la solution optimale : **la valeur maximale de la dernière ligne du tableau représente l'une des deux sous-sommes optimales** (la plus petite), l'autre sous-somme optimale étant alors la somme globale moins cette valeur.

Récupérer la sous-liste

Nous allons reconstruire la sous-liste obtenue en suivant le procédé à l'envers.
Analysons le tableau

- En T[4,16], nous avons le résultat optimal. Or le quatrième nombre (piece[4]) est 8.
- Passons alors en colonne 16-8 = 8 ; à nouveau, le résultat de T[3,16-8] est optimal (car la case au-dessus est false). De plus, nous savons que le troisième nombre (piece[3]) est 3.
- On passe à T[2,8-3] ; le résultat n'est pas optimal (puisque la case du dessus est encore V).
- On passe alors à T[1,5] ; le résultat est optimal, donc on ajoute le premier nombre (piece[1]) à notre sous-liste.
- On retrouve bien une sous-liste optimale : 8, 3 et 5.