CORRIGÉ

Tle

Programmation dynamique — Mémoïsation

Corrigé détaillé
4 exercices

> Ce qui est noté au contrôle, ce n'est pas le résultat seul : c'est le raisonnement posé, puis le calcul mené jusqu'au bout avec l'unité.

1 Vocabulaire. Répondre en une phrase précise.

1. a. Le **chevauchement** des sous-problèmes : les mêmes calculs doivent être demandés **plusieurs fois**.
2. b. La **mémoïsation** est **descendante** — on garde la récursion et on mémorise. La **tabulation** est **ascendante** — on remplit un tableau des petits cas vers les grands.
3. c. La saturation de la **pile d'appels**, puisqu'elle n'emploie pas de récursion.
4. d. Choisir parmi des objets ayant un **poids** et une **valeur**, de façon à **maximiser la valeur** sans dépasser une **capacité**.
5. e. Qu'une solution **optimale** se compose de solutions **optimales** de sous-problèmes.

2 Un développeur ajoute une mémoïsation à une fonction de tri fusion. Analyser.

1. a. **Non**. Chaque appel porte sur une **portion différente** du tableau : aucune portion n'est triée deux fois.
2. b. Des entrées **consultées zéro fois** : chaque résultat stocké ne sera jamais redemandé.
3. c. **Non** : il devient **plus lent**. Aucun calcul n'est supprimé, et deux opérations sont ajoutées à chaque appel.
4. d. Le **stockage** de la table, sa **consultation** avant chaque calcul et son **écriture** après. On paie donc la mémoire **et** le temps.
5. e. Vérifier, sur la version **récursive naïve**, si les mêmes **paramètres** reviennent dans plusieurs branches. La réponse était non, et la **mémoïsation** était donc exclue d'emblée.

3 Le calcul récursif naïf de Fibonacci recalcule massivement les mêmes valeurs.

1. a. Parce que le terme n demande n−1 **et** n−2, et que n−1 redemande n−2 : la branche se duplique à chaque niveau, et le nombre d'appels croît de façon **exponentielle**.
2. b. **n** au plus : un par valeur de 0 à n. Le nombre d'appels naïfs est exponentiel, mais le nombre de **calculs distincts** est linéaire.
3. c. **O(n)** : chaque sous-problème n'est calculé **qu'une fois**, les autres appels étant des consultations.
4. d. Parce qu'on passe d'un nombre **exponentiel** d'appels à un nombre **linéaire** de calculs. L'écart entre les deux croît si vite qu'il rend utilisable, en une fraction de seconde, un calcul qui demandait auparavant des durées inatteignables.

4 Mémoïsation ou tabulation. Raisonner sur le choix.

1. a. Elle ne calcule que les sous-problèmes **effectivement nécessaires**, là où la **tabulation** peut remplir des cases dont on n'avait pas besoin.
2. b. Elle n'emploie **pas de récursion**, donc n'empile aucun appel : elle évite la saturation de la **pile d'appels** sur les grandes entrées.
3. c. Il faut trouver le bon **ordre de remplissage** : chaque case ne doit dépendre que de cases **déjà remplies**. C'est parfois le vrai travail du problème.
4. d. Parce que ses sous-problèmes — la meilleure valeur avec les i premiers objets et une capacité c — se **chevauchent** massivement, et qu'une solution optimale s'y compose de **sous-solutions optimales**.
5. e. Un problème où un choix **localement optimal interdit** le meilleur choix global : la seconde propriété tombe, et décomposer en sous-problèmes optimaux ne reconstruit plus l'optimum. La propriété se **vérifie**, elle n'est pas automatique.

Corrige-toi toi-même. Compte tes points exercice par exercice : c'est ainsi qu'on voit où l'on perd, et ce n'est presque jamais là où on le croit.

Ce qui est évalué	Points	Ce qui fait perdre le point
Énoncer la condition de chevauchement	5 pts	Traiter la mémoïsation comme une optimisation générale
Montrer qu'une mémoïsation inutile coûte	4 pts	Conclure qu'elle est au pire neutre
Distinguer mémoïsation et tabulation	4 pts	Les présenter comme équivalentes
Estimer le gain par le nombre de sous-problèmes distincts	4 pts	Annoncer un gain sans le mesurer

Vérifier la sous-structure optimale	3 pts	La supposer acquise