Série 4 : Solutions

1 Une meilleure version du tri par fusion ?

A priori, il peut sembler plus efficace de diviser la liste en 4 que de la couper en 2, car ainsi le nombre d'étapes passe de $\log_2(n)$ à $\log_4(n)$. Cependant, remarquez que :

- $\log_4(n) = \frac{\log_2(n)}{\log_2(4)} = \frac{1}{2} \cdot \log_2(n)$: le gain n'est que d'un facteur constant…
- d'un autre côté, la procédure de fusion de 4 listes est plus complexe que celle de la fusion de 2 listes : en effet, à chaque étape, pour décider quel élément rajouter à la liste fusionnée, il faut effectuer 3 tests successifs au lieu d'un seul : comparer $L_1(1)$ et $L_2(1)$, puis comparer $L_3(1)$ et $L_4(1)$, puis comparer encore $\min\{L_1(1), L_2(1)\}$ et $\min\{L_3(1), L_4(1)\}$.

L'un dans l'autre, ces deux effets se compensent et c'est même la première version qui gagne ! Dans le même ordre d'idées, diviser la liste en 8, en 16, deviendrait encore moins efficace…

Question subsidiaire : Si on divise directement la liste en n unités, alors la procédure de fusion devient beaucoup plus complexe ! Plus précisément, pour fusionner (en une fois) les n unités, il faut d'abord chercher le plus petit élément parmi n, puis cherche le 2e plus petit élément parmi les $n-1$ restants, etc. Ceci demande de l'ordre de

$$n + (n-1) + (n-2) + \cdots + 3 + 2 + 1 = \Theta(n^2) \text{ opérations}$$

donc on obtient une complexité temporelle similaire au tri par insertion. Cette nouvelle méthode de tri a en fait un nom : elle s'appelle le *tri par sélection.*

2 Rendu de pièces de monnaie

Avec P_1, le rendu glouton n'atteint pas le montant exact et rend 1 fr. + 1 fr. + 40 cts, alors qu'un rendu exact serait obtenu avec 70 cts + 70 cts + 40 cts + 40 cts + 40 cts.

Avec P_2, le rendu glouton n'atteint pas le montant exact et rend 1 fr. + 1 fr. + 50 cts, alors qu'un rendu exact serait obtenu avec 70 cts + 70 cts + 70 cts + 50 cts.

Avec P_3, le rendu glouton atteint le montant exact et rend 1 fr. + 1 fr. + 40 cts + 10 cts + 10 cts, alors qu'un rendu optimal (avec moins de pièces) serait obtenu avec 1 fr. + 1fr. + 30 cts + 30 cts.

Avec P_4, le rendu glouton atteint le montant exact avec un nombre minimal de pièces et rend 1 fr. + 1 fr. + 40 cts + 20 cts.

3 Comptage de décompositions

Solutions

a)

- [ ] 2
- [ ] 3
- [x] 4
- [ ] 5

Pour déterminer le nombre de façons d'obtenir la valeur 5 en utilisant 1 ou 3, on note $f(z)$ le nombre total de possibilités. Chaque choix retire soit 1, soit 3 de la valeur cible. Cela conduit à la relation $f(z) = f(z-1) + f(z-3)$, avec $f(0) = 1$. En calculant progressivement : $f(1) = 1$, $f(2) = 1$, $f(3) = 2$, $f(4) = 3$, $f(5) = 4$. On obtient donc 4 façons différentes.

b)

- [ ] $\Theta(z)$
- [x] $\Theta(k^z)$
- [ ] $\Theta(z^2)$
- [ ] $\Theta(z \log z)$

La version naïve explore toutes les suites possibles de retraits successifs parmi les k valeurs de l'ensemble. Après chaque choix, k nouvelles possibilités apparaissent, et cela pendant environ z étapes. Le nombre total de chemins possibles augmente donc comme k multiplié par lui-même z fois, ce qui correspond à une croissance exponentielle.

c)

- ☐ $\Theta(z)$
- ☒ $\Theta(kz)$
- ☐ $\Theta(k^z)$
- ☐ $\Theta(z^2)$

Avec la mémoïsation, chaque valeur entre 0 et z est calculée au plus une seule fois. Pour chacune d'elles, on essaie tous les choix possibles dans l'ensemble, soit k vérifications. On effectue donc un total d'environ $k \cdot z$ opérations.

d)

- ☐ Algorithme glouton
- ☒ Programmation dynamique et mémoïsation
- ☐ Calculabilité
- ☐ Recherche dichotomique

Cet exercice montre qu'une fonction récursive coûteuse peut être largement accélérée en enregistrant les résultats déjà calculés. Cette approche, où l'on évite de calculer plusieurs fois les mêmes valeurs, est une forme de programmation dynamique appliquée de manière descendante.

4 Gloutons vs. dynamique 1

a)

- ☐ `[8, 3, 15, 9, 7, 21]`
- ☐ `[1, 3, 15, 8, 7, 20]`
- ☒ `[8, 3, 15, 5, 7, 1]`
- ☐ `[1, 2, 5, 8, 7, 10]`

L'algorithme glouton choisit toujours la prochaine valeur strictement plus grande en se basant uniquement sur une décision immédiate. Dans l'exemple correct, il sélectionne 8 puis 15, ce qui donne une sous-séquence de longueur 2. L'algorithme fondé sur la programmation dynamique examine toutes les possibilités d'extension de sous-séquences et peut trouver une suite de longueur 3, comme 3, 5, 7.

b)

- ☒ Glouton : $\Theta(n)$, Dynamique : $\Theta(n^2)$
- ☐ Glouton : $\Theta(n \log n)$, Dynamique : $\Theta(n^2)$
- ☐ Glouton : $\Theta(n)$, Dynamique : $\Theta(n \log n)$
- ☐ Glouton : $\Theta(n^2)$, Dynamique : $\Theta(n^2)$

L'approche gloutonne se contente d'un seul parcours de la liste, ce qui donne un temps proportionnel à sa taille. L'algorithme utilisant la programmation dynamique compare chaque élément à tous ceux qui le précèdent, ce qui conduit à un nombre d'étapes proportionnel à n multiplié par n.

5 Gloutons vs. dynamique 2

a)

- [X] $\begin{pmatrix} 1 & 3 & 1 \\ 1 & 5 & 1 \\ 4 & 2 & 1 \end{pmatrix}$
- [ ] $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 8 & 2 \\ 1 & 5 & 3 \end{pmatrix}$
- [ ] $\begin{pmatrix} 1 & 10 & 1 \\ 1 & 1 & 100 \\ 10 & 1 & 1 \end{pmatrix}$
- [ ] $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 100 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

L'algorithme glouton choisit toujours l'étape immédiatement la moins coûteuse, mais cela peut l'amener sur un chemin qui devient plus cher par la suite. Dans la première matrice, ce raisonnement local ne conduit pas au chemin optimal. L'algorithme fondé sur la programmation dynamique calcule pour chaque case le coût minimal pour y arriver, ce qui lui permet de reconstruire un chemin complet dont le coût total est minimal.

b)

- [X] Glouton : $\Theta(n+m)$, Dynamique : $\Theta(nm)$
- [ ] Glouton : $\Theta(nm)$, Dynamique : $\Theta(nm \log(nm))$
- [ ] Glouton : $\Theta(nm)$, Dynamique : $\Theta(n+m)$
- [ ] Glouton : $\Theta(n+m)$, Dynamique : $\Theta(n+m)$

L'algorithme glouton effectue environ $(n-1)+(m-1)$ déplacements puisqu'il avance toujours soit vers la droite soit vers le bas ; son temps est donc proportionnel à $n+m$. L'algorithme utilisant la programmation dynamique remplit un tableau contenant une valeur pour chacune des nm cases, ce qui nécessite nm calculs.