Série 4

1 Une meilleure version du tri par fusion ?

Au cours, nous avons vu le tri par fusion, dont la complexité temporelle est $\Theta(n \cdot \log_2(n))$.

Voici maintenant une proposition d'amélioration de cet algorithme : au lieu de diviser la liste à trier en deux parties à chaque étape, divisons-la en quatre parties, trions chacune de ces quatre parties séparément (au moyen de notre nouvel algorithme), et fusionnons ensuite ces quatre parties en une seule. Quelle sera alors la complexité temporelle de ce nouvel algorithme, si on suppose que l'algorithme de fusion de quatre listes L_1, L_2, L_3, L_4 a une complexité temporelle $\Theta(m_1 + m_2 + m_3 + m_4)$, où m_i est la taille de la liste L_i ?

Question subsidiaire : Et si on pousse l'idée encore plus loin et qu'on divise la liste en une seule fois en n unités au départ, puis qu'on fusionne ensuite ces n unités également en une fois, quelle complexité temporelle obtient-on ?

2 Rendu de pièces de monnaie

Pour un montant à rendre valant $z = 2.60$ frs et pour chacun des ensembles de pièces de monnaie ci-dessous :

$P_1 = (40 \text{ cts}, 70 \text{ cts}, 1 \text{ fr.})$ $P_2 = (50 \text{ cts}, 70 \text{ cts}, 1 \text{ fr.})$

$P_3 = (10 \text{ cts}, 30 \text{ cts}, 40 \text{ cts}, 1 \text{ fr.})$ $P_4 = (20 \text{ cts}, 40 \text{ cts}, 1 \text{ fr.})$

déterminez si l'algorithme de rendu "glouton" vu au cours :

a) rend le montant exact ;

b) utilise le nombre minimum de pièces pour cela.

Dans les cas où la réponse à l'une des deux questions ci-dessus est négative, déterminez quelle est la solution optimale (si une telle solution existe).

3 Comptage de décompositions

On souhaite calculer le nombre de manières d'atteindre exactement la valeur cible z en utilisant un ensemble de nombres positifs $P = \{p_1, \ldots, p_k\}$. L'ordre des éléments compte.

nb_manieres_naif

entrée : *entier z, ensemble P*
sortie : *entier res*

```
Si z = 0
    Sortir : 1
Si z < 0
    Sortir : 0
res ⟵ 0
Pour chaque p dans P
    res ⟵ res + nb_manieres_naif(z − p, P)
Sortir : res
```

nb_manieres_memo

entrée : *entier z, ensemble P, dictionnaire memo*
sortie : *entier res*

```
Si z = 0
    Sortir : 1
Si z < 0
    Sortir : 0
Si z ∈ memo
    Sortir : memo[z]
res ⟵ 0
Pour chaque p dans P
    res ⟵ res + nb_manieres_memo(z − p, P, memo)
memo[z] ⟵ res
Sortir : res
```

a) Pour $P = \{1, 3\}$ et $z = 5$, combien de façons existe-t-il ?

☐ 2 ☐ 3 ☐ 4 ☐ 5

b) Quelle est la complexité temporelle en fonction de z et $k = |P|$ de `nb_manieres_naif` ?

☐ $\Theta(z)$ ☐ $\Theta(k^z)$ ☐ $\Theta(z^2)$ ☐ $\Theta(z \log z)$

c) Quelle est la complexité temporelle (en notant $k = |P|$) de `nb_manieres_memo` ?

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

d) Ce que cet exercice illustre surtout :

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

4 Gloutons vs. dynamique 1

On souhaite déterminer la longueur de la plus longue sous-séquence strictement croissante dans une liste d'entiers.

Voici deux implémentations : une approche gloutonne et une approche dynamique.

plus_longue_chaine_glouton

entrée : *tableau L de taille n*
sortie : *nombre entier res*

```
Si n = 0
  Sortir : 0
res ⟵ 1
dernier ⟵ L[1]
Pour i ⟵ 2 to n
  Si L[i] > dernier
    res ⟵ res + 1
    dernier ⟵ L[i]
Sortir : res
```

plus_longue_chaine_dynamique

entrée : *tableau L de taille n*
sortie : *nombre entier res*

```
Si n = 0
  Sortir : 0
Initialiser un tableau dp[1:n] tel que ∀i, dp[i] ⟵ 1
Pour i ⟵ 1 to n
  Pour j ⟵ 1 to i − 1
    Si L[j] < L[i]
      dp[i] ⟵ max(dp[i], dp[j] + 1)
res ⟵ max(dp[1:n])
Sortir : res
```

a) Laquelle des entrées suivantes produit une réponse non optimale avec l'algorithme glouton mais optimale avec l'algorithme dynamique ?

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

b) Quelle est respectivement la complexité temporelle de chaque algorithme, en fonction de n ?

- ☐ 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 : $\Theta(n^2)$

5 Gloutons vs. dynamique 2

On souhaite déterminer le chemin de coût minimal pour atteindre la case en bas à droite d'un tableau à deux dimensions contenant des entiers positifs. On ne peut se déplacer que vers le bas ou vers la droite. Voici deux implémentations : une approche gloutonne et une approche dynamique.

chemin_minimal_glouton

entrée : *tableau tab de taille $n \times m$*
sortie : *nombre entier res*

```
Si n = 0 or m = 0
    Sortir : 0
i ⟵ 1
j ⟵ 1
res ⟵ tab[1][1]
Tant que i < n or j < m
    Si i = n
        j ⟵ j + 1
    Sinon, si j = m
        i ⟵ i + 1
    Sinon, si tab[i + 1][j] < tab[i][j + 1]
        i ⟵ i + 1
    Sinon
        j ⟵ j + 1
    res ⟵ res + tab[i][j]
Sortir : res
```

chemin_minimal_dynamique

entrée : *tableau tab de taille $n \times m$*
sortie : *nombre entier res*

```
Si n = 0 or m = 0
    Sortir : 0
Initialiser un tableau dp[1:n][1:m]
dp[1][1] ⟵ tab[1][1]
Pour i ⟵ 2 to n
    dp[i][1] ⟵ dp[i − 1][1] + tab[i][1]
Pour j ⟵ 2 to m
    dp[1][j] ⟵ dp[1][j − 1] + tab[1][j]
Pour i ⟵ 2 to n
    Pour j ⟵ 2 to m
        dp[i][j] ⟵ tab[i][j] + min(dp[i − 1][j], dp[i][j − 1])
res ⟵ dp[n][m]
Sortir : res
```

a) Laquelle des entrées suivantes produit une réponse non optimale avec l'algorithme glouton mais optimale avec l'algorithme dynamique ?

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

b) Quelle est respectivement la complexité temporelle de chaque algorithme ?

- ☐ 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 : $\Theta(n+m)$