TD21 : Algorithmique

1 Cible à atteindre

On se propose d'atteindre n'importe quel nombre à partir de 1, en s'autorisant les opérations suivantes :
- incrémentation de 1,
- doublement,

en un nombre minimal d'étapes.
exemple : $1 \xrightarrow{\times 2} 2 \xrightarrow{+1} 3 \xrightarrow{\times 2} 6 \xrightarrow{\times 2} 12$

Question 1 : Donner un algorithme glouton pour résoudre ce problème.

Question 2 : Prouver la correction de cet algorithme.

2 Mot bien parenthésé

L'ensemble des mots bien parenthésés est défini inductivement par :
- assertion : le mot vide est bien parenthésé
- règles d'inférences : si u et v sont bien parenthésés, alors $u(v)$ est bien parenthésé.

Question 3 : Trouver un algorithme glouton qui prend en argument une suite de parenthèses, dont on suppose qu'elle contient autant de parenthèses ouvrantes que de parenthèses fermantes, et détermine le nombre d'inversions à faire pour obtenir un mot bien parenthésé.

3 Étagères

On considère une liste de n livres dans un ordre prédéfini qu'on ne peut pas modifié et une bibliothèque contenant des étagères.
Chaque livre est caractérisé par son épaisseur e_i et par sa hauteur h_i. La largeur L de la bibliothèque n'est pas modifiable et chaque livre doit reposer entièrement sur une étagère.

Question 4 : Supposons que tous les livres ont la même hauteur h et que toutes les tablettes peuvent stocker des livres de cette hauteur. Donner un algorithme glouton pour stocker les livres en utilisant le moins de tablettes possibles. Montrer sa correction.

Question 5 : Supposons maintenant que les livres peuvent avoir des hauteurs différentes et qu'on peut ajuster chaque tablette pour accueillir le livre le plus haut sur cette tablette. En particulier, la hauteur de chaque tablette vide peut être 0. Le problème est de stocker les livres de sorte que la somme totale des hauteurs des tablettes soit le plus petit possible. Montrer que l'algorithme glouton précédent ne donne pas toujours la meilleure solution.

Question 6 : Donner un algorithme pour trouver la meilleure solution.

4 Détecter de la fausse monnaie

On dispose de n pièces dont exactement une est fausse. On peut détecter cette fausse pièce car on sait qu'elle est plus légère que les autres (les autres font toutes le même poids) et on dispose d'une balance à plateaux.

Question 7 : Proposer un algorithme de complexité linéaire permettant de repérer la fausse pièce.

Question 8 : En utilisant la stratégie "diviser pour régner", proposer un algorithme de complexité logarithmique pour résoudre le même problème.

Question 9 : Montrer que votre algorithme possède une correction totale et calculer sa complexité dans le pire des cas. L'algorithme est-il optimal ?

5 Produit matriciel

Soit un entier $n \in \mathbb{N}^*$. On s'intéresse à la complexité des opérations entre deux matrices carrées de taille n.

Question 10 : Combien de sommes de nombre fait-on pour calculer la somme de deux matrices carrés de taille n ?

Question 11 : Combien de sommes et combien de produits de nombres fait-on pour calculer de le produit de deux matrices carrés de taille n (avec l'algorithme naïf) ?

Question 12 : On découpe chaque matrice en 4 blocs :

$$M_i = \begin{pmatrix} A_i & B_i \\ C_i & D_i \end{pmatrix} \quad i \in \{1,2\}.$$

Exprimez le produit $M_1 M_2$ en fonction des produits des blocs.

Question 13 : Si on utilise la stratégie "diviser pour régner" pour calculer récursivement le produit $M_1 M_2$, quelle est la complexité de l'algorithme de multiplication ? Est-ce mieux que l'algorithme naïf ?

Question 14 : L'algorithme de Strassen est plus subtil dans ses appels récursifs :

$$M_1 M_2 = \begin{pmatrix} X & Y \\ Z & T \end{pmatrix},$$

où

$$\begin{aligned} S_1 &= (B_1 - D_1)(C_2 + D_2) \\ S_2 &= (A_1 + D_1)(A_2 + D_2) \\ S_3 &= (A_1 - C_1)(A_2 + B_2) \end{aligned} \qquad \begin{aligned} S_4 &= (A_1 + B_1)D_2 \\ S_5 &= A_1(B_2 - D_2) \\ S_6 &= D_1(C_2 - A_2) \\ S_7 &= (C_1 + D_1)A_2 \end{aligned} \qquad \begin{aligned} X &= S_1 + S_2 - S_4 + S_6 \\ Y &= S_4 + S_5 \\ Z &= S_6 + S_7 \\ T &= S_2 - S_3 + S_5 - S_7 \end{aligned}$$

Déterminer la complexité de cet algorithme

6 Produits matriciels

Question 15 : Si on fait le produit de deux matrices, A de taille $p \times q$ et B de taille $q \times r$, de manière naïve, combien de produits entre scalaires effectue-t-on ?

Question 16 : On souhaite calculer le produit de n matrices

$$M = M_1 M_2 \dots M_n.$$

en jouant sur l'associativité du produit matriciel pour minimiser le nombre de produits effectués entre scalaires. Pour $n = 4$ et $M_1 : 13 \times 5$, $M_2 : 5 \times 89$, $M_3 : 89 \times 3$ et $M_4 : 3 \times 34$, donner toutes les possibilités de parenthésages et le nombre de produits entre scalaires effectués à chaque fois.

Question 17 : Pour tout i, notons $d_{i-1} \times d_i$ la dimension de la matrice M_i, et $m_{i,j}$ le nombre minimal de produits entre scalaires pour calculer le produit matriciel $M_i \dots M_j$ (avec $i \leq j$).

1. Quelle est la valeur de $m_{i,i}$?
2. Donner une relation de récurrence entre les $m_{i,j}$ (pour $i \leq j$).

Question 18 : Proposer une stratégie de type programmation dynamique avec calcul de bas en haut pour déterminer les $m_{i,j}$ (pour $i \leq j$), en spécifiant l'ordre dans lequel on doit les calculer.

Question 19 : Cette stratégie s'adapte-t-elle si on utilise le produit matriciel de la partie 2 ?