Algorithmique — M1

Partiel du 28 novembre 2008

Université Paris Diderot

Chaque exercice DOIT être rendu sur une feuille différente (4 en tout) Document autorisé : UNE feuille de papier format A4 Durée : 1h30

Exercice 1 : applications de cours

1.1. Récurrence

Étant donné que

$$T(n) = 9T(\lfloor n/3 \rfloor) + n^2 + 3$$

trouvez le comportement asymptotique de $T(n)$. Justifiez votre réponse.

1.2. Cinéma

Les séances au cinéma "PRG-Algo" ont lieu aux horaires suivants :

A :08h10-11h ; B :9h15-10h20 ; C :10h-13h ; D :11h-13h20 ; E :13h10-15h ; F :13h15-14h45 ; G :14h-17h ; H :14h50-18h ;

L'étudiant a une journée libre et il veut voir le maximum de films pendant cette journée.

1. Quel algorithme de cours résout ce problème ? Décrivez l'algorithme en français ou en pseudo-code.
2. Appliquez l'algorithme pour choisir les films à voir.

Exercice 2 : Greedy - un algorithme facile à inventer

Sur la rue Gloutonne il y a n maisons avec les coordonnées $x_1, x_2, \ldots, x_n$ (données du problème). Une ligne de bus passe par cette rue. On cherche à placer les arrêts de bus de manière que pour chaque maison il y ait un arrêt à moins de 100 m. Trouver le nombre minimal m et les coordonnées de tels arrêts $a_1, \ldots, a_m$.

1. Pour $x_1 = 180, x_2 = 200, x_3 = 370, x_4 = 390, x_5 = 590$ faire un dessin et trouver une solution optimale.
2. Proposer un choix glouton pour le premier arrêt.
3. Proposer un algorithme glouton qui résout le problème
4. Énoncer et démontrer le lemme du choix glouton.
5. Analyser la complexité de votre algorithme.

Exercice 3 : Divide & Conquer - adaptation d'un algorithme de cours

Tout le monde connaît la séquence de Fibonacci 1,1,2,3,5,8,13,21,... définie par la récurrence :

$$\begin{cases} F(1) &= 1 \\ F(2) &= 1 \\ F(n) &= F(n-2) + F(n-1) \text{ pour } n > 2 \end{cases}$$

Il est très facile de calculer $F(n)$ en temps $O(n)$. Dans cet exercice on trouve un algorithme beaucoup plus efficace.

1. Montrer que le calcul de la suite de Fibonacci se ramène à celui de la résolution d'une équation matricielle récurrente. On notera $\vec{V}(n) = \begin{pmatrix} F(n) \\ F(n-1) \end{pmatrix}$ un vecteur colonne à deux éléments et $\mathbf{F}$ une matrice carrée (2×2). On cherche $\mathbf{F}$ telle que :

$$\begin{cases} \vec{V}(2) &= \text{constante} \\ \vec{V}(n) &= \mathbf{F} \times \vec{V}(n-1) \end{cases}$$

2. Montrer que la solution de cette équation de récurrence s'écrit $\vec{V}(n) = \mathbf{F}^{n-2} \times \vec{V}(2)$.
3. En déduire un algorithme (qui ressemble beaucoup à un algorithme de cours) de type « diviser pour régner » calculant $F(n)$ pour tout n, et prouver que sa complexité est $O(\log(n))$ en terme de nombre d'opérations arithmétiques.
4. Montrer par un cas particulier que cet algorithme d'élévation à la puissance n n'est pas optimal. Prendre par exemple : $n = 15$.

Exercice 4 : Backtracking

On considère le problème (NP-complet) classique de 3-matching. Il y a n garçons $G = \{g_1, \ldots, g_n\}$, qui veulent épouser n filles $F = \{f_1, \ldots, f_n\}$ et s'installer dans n maisons $M = \{m_1, \ldots, m_n\}$. Un ensemble S contient plusieurs triplets de la forme ijk qui représentent les souhaits. Ainsi un triplet ijk signifie que g_i et f_j sont d'accord de se marier et de s'installer à la maison m_k.

Le problème algorithmique est le suivant : étant donné n et S trouver une façon de marier et loger tous les couples en respectant tous les souhaits (ou dire que c'est impossible).

1. Trouver un 3-matching pour $n = 3; S = \{111, 132, 221, 233, 321, 333, 322\}$
2. Décrire les solutions partielles du problème (configurations prometteuses)
   - Qu'est ce qu'une configuration, qu'est-ce qu une configuration prometteuse ? Quelle est une structure de données adaptée pour représenter une configuration.
   - Comment tester si une configuration est prometteuse ? Comment tester si une configuration prometteuse est une solution ?
3. Décrire un graphe de configurations prometteuses.
4. Proposer un algorithme (de type retour arrière) pour résoudre le problème de 3-matching.
5. Analyser la complexité de votre algorithme.