Algorithmique — M1 — Université Paris Diderot

Coorigé (béta 1.1) du partiel du 28 novembre 2008

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.

Correction : On applique le Master Theorem. On trouve d'abord l'exposant $k = \log_3 9 = 2$. On remarque que la perturbation est $n^2 + 3 = \Theta(n^2) = \Theta(n^k)$ - c'est donc le cas de perturbation moyenne. D'où, par le Master Theorem, on obtient que

$$T(n) = \Theta(n^2 \log n).$$

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.

   Correction : C'est le même problème que l'affectation de salle (vue en cours). L algorithme glouton, vu en cours, est le suivant :

   Trier les requêtes (d_i, f_i) en ordre ascendant de f

   T=0
   pour i de 1 à n faire
   si $d_i >= T$
   aller au séance i
   $T = f_i$

2. Appliquez l'algorithme pour choisir les films à voir.

   Correction : On trie les séances selon l' heure de la fin : BACDFEGH
   On prend B, T=10 :20, on saute A et C, on prend D, T=13 :20, on saute F et E, on prend G, T=17, on saute H. On verra donc 3 films seulement : B,D,G.

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 (ou=) 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.

   Correction : Deux arrêts suffisent : $a_1 = 280$ desservira trois premières maisons, $a_2 = 490$ les deux autres

2. Proposer un choix glouton pour le premier arrêt.

   Correction : On suppose que les maisons sont placé dans l'ordre croissant de numéros $x_1 < x_2 < \cdots < x_n$ (si ce n'est pas le cas, il faut faire un tri en pré-traitement). Choix glouton : on met le premier arrêt à 100 mètres après la 1ère maisons (c.-à-d. $a_1 = x_1 + 100$).

3. Proposer un algorithme glouton qui résout le problème,

   Correction : On trie si nécessaire. On fait le choix glouton pour placer le premier arrêt. On saute toutes les maison desservis par cet arrêt, on refait la même procédure pour les maisons qui restent non desservis.
   Pseudo-code (variable Z désigne la fin de zone desservie, m=le nombre d' arrêts déjà faits)

   Trier les maisons en ordre ascendant de x_i

   $Z = -\infty$
   pour i de 1 à n faire
   si $x_i > Z$ //première maison pas encore desservie
   m++ ; //on ajoute un arrêt
   $a_m = x_i + 100$; //100 mètres après la maison
   $Z = a_m + 100$; //et on déplace la frontière de la zone

4. Énoncer et démontrer le lemme du choix glouton.

   Correction :

   Lemme 1 *Il existe une solution optimale où le premier arrêt (le plus à gauche) a la position* $a_1 = x_1 + 100$

   Démonstration : Soit $A = a_1, \ldots, a_m$ une solution optimale quelconque (on suppose que les arrêts sont classés dans l' ordre croissant des coordonnées. Forcément la maison la plus à gauche x_i est desservie par un arrêt a_i (on a donc $a_i <= x_1 + 100$). Si on avance ce premier arrêt utile vers la droite à la nouvelle position $x_1 + 100$, il desservira encore la maison x_1 et toutes les autres maison qui étaient desservies par a_i. Tous les arrêts plus à gauche : $a_1..a_{i-1}$ ne servent à rien, on peut les supprimer.
   Ainsi (en poussant le premier arrêt utile et en supprimant les inutiles) on obtient une nouvelle solution optimale A' avec $a'_1 = x_1 + 100$. CQFD

   Remarque Pour démontrer la correction de l'algorithme il faut aussi établir la sous-structure- optimale : après avoir placé le premier arrêt comme décrit à $a_1 = x_1 + 100$, il reste à resoudre le même problème, mais seulement pour toutes les maisons après la position $a_1 + 100$.

5. Analyser la complexité de votre algorithme.

   Correction : La boucle principale fait O(n). Le tri fait $O(n \log n)$. D'où la réponse : si les maisons sont déjà classés (et le tri est inutile) la complexité de l' algorithme est O(n). S'ils ne le sont pas la complexité est de $O(n \log n)$.

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 :

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

   Correction :
   On a $\vec{V}(2) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, ce qui est une constante.
   D' autre part, pour $n > 2$ on a

$$\vec{V}(n) = \begin{pmatrix} F(n) \\ F(n-1) \end{pmatrix} = \begin{pmatrix} F(n-1) + F(n-2) \\ F(n-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} F(n-1) \\ F(n-2) \end{pmatrix} = \mathbf{F} \cdot \vec{V}(n-1),$$

   avec

$$\mathbf{F} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

   CQFD
2. Montrer que la solution de cette équation de récurrence s'écrit $\vec{V}(n) = \mathbf{F}^{n-2} \cdot \vec{V}(2)$.

   Correction : Preuve par récurrence. Base :$\vec{V}(2) = \mathbf{F}^0 \vec{V}(2)$ est trivial.
   Pas inductif : Supposons que c'est vrais pour $n-1$: $\vec{V}(n-1) = \mathbf{F}^{n-3} \cdot \vec{V}(2)$. Alors $\vec{V}(n) = \mathbf{F}\vec{V}(n-1) = \mathbf{F} \cdot \mathbf{F}^{n-3} \cdot \vec{V}(2) = \mathbf{F}^{n-2} \cdot \vec{V}(2)$ CQFD.
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.

   Correction : On a vu en cours un algorithme "diviser pour régner" de calcul de puissances. Il s'adapte parfaitement aux puissances de la matrice $\mathbf{F}$.

```
int[.,.] fonction Puissance (F,n)
si n=0
        retourner I
sinon si n=1
        retourner F
sinon si n est pair
            M= Puissance (F, n/2)
            retourner Mult(M,M)
sinon // si n est impair
            M= Puissance (F, (n-1)/2)
            retourner Mult (Mult(M,M),F)
```

En sachant que pour les matrices de taille fixe 2×2, l'opération de multiplication a un coût fixe O(1) (en nombre d'opérations arithmétiques), on obtient pour la complexité de cet algorithme la récurrence

$$T(n) = T(\lfloor n/2 \rfloor) + O(1),$$

d'où, par Master Theorem $T(n) = O(\log n)$.

Finalement, pour calculer le n-ème élément de la séquence de Fibonacci $F(n)$ on applique la formule prouvée $\vec{V}(n) = \mathbf{F}^{n-2} \cdot \vec{V}(2)$. La complexité est $O(\log(n-2)) + O(1)$ ce qui se simplifie à $O(\log n)$.

4. Montrer par un cas particulier que cet algorithme d'élévation à la puissance n n'est pas optimal. Prendre par exemple : $n = 15$.

   Correction : L'algorithme "diviser pour régner" donne pour $n = 15$ le calcul suivant

   $$M^{15} = M * (M^7)^2 = M * (M * (M^3)^2)^2 = M * (M * (M * M^2)^2)^2.$$

   Ça fait 6 opération : 3 carrés et 3 multiplications avec M. Or, en réfléchissant on trouve un calcul avec 5 multiplications

   $$A = M * M * M; B = A * A; C = B * A; M^{15} = B * C.$$

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 $n = 3; S = \{111, 132, 221, 233, 321, 333, 322\}$

   Correction : M={111,233,322}

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 ?

   Correction : Une configuration (de niveau k) donne une fiancée et une maison à chacun des k premiers garçons. On peut la représenter par deux tableaux des entiers : ff[1..k] (les numéros des fiancées de g_1,.., g_k) et mm[1..k] (numéros de leurs maisons). Une configuration est prometteuse si

– tous les triplets (i,ff[i],mm[i]) (pour i de 1 à k) appartiennent à l'ensemble de souhaits S ;
– tous les ff[i] sont différents ;
– tous les mm[i] sont différents.

Le test de config prometteuses est immédiat

```
fonction booléenne test(ff,gg,k)
pour x de 1 à k
        si (x,ff[x],mm[x]) ¬ ∈ S
           retourner faux
        pour y de x+1 à k
           si (ff[x]==ff[y] ou mm[x]==mm[y])
              retourner faux
retourner vrai
```

Par ailleurs, si on obtient une configuration de manière incrémentale (en ajoutant un triplet (k+1,i,j)$\in S$ à une configuration prometteuse de niveau k (comme on fera dans l'algorithme complet, le test se simplifie - il faut juste tester que la nouvelle fille f_i et la nouvelle maison m[j] ne sont pas prises.

```
fonction booléenne testIncrémental(ff,gg,k, i,j)
pour x de 1 à k
           si (ff[x]==i ou mm[x]==mm[j])
              retourner faux
retourner vrai
```

Une configuration prometteuse est une solution si elle est de niveau n.

3. Décrire un graphe de configurations prometteuses.

   Correction : En fait c'est un arbre. Racine : Configuration vide. Sommets : toutes les configurations prometteuses. Arcs : soit u une configuration de niveau k. Il y a un arc de u vers chaque config. prometteuse de forme $v = u \cup (k+1, i, j)$. En autres mots, $u \rightarrow v$ si on obtient v à partir de u en mariant et logeant le garçon g_{k+1}

4. Proposer un algorithme (de type retour arrière) pour résoudre le problème de 3-matching.
   L'algorithme sera un DFS dans l'arbre précédemment décrit.

```
fonction DFS(conf)
k=niveau(conf)

si (k=n)
        imprimer(conf)et s'arrêter
sinon
        pour tout triplet t dans S de forme (k+1,i,j) faire
                si testIncrémental()
                   nouvelle = conf ∪ (k+1,i,j)
                   DFS(nouvelle)
```

Et pour trouver un 3-MATCHING on appelle DFS(vide).

5. Analyser la complexité de votre algorithme.

   Correction : Les configurations prometteuses (le noeuds de l'arbre exploré) qu'on essaye sont des sous-ensembles de S qui contiennent jusqu'à n éléments. Soit C le nombre de telles configurations. Le DFS d'un arbre a la complexité $O(C)$. On a O(n) opérations pour tester la cohérence de chaque noeud (si on ne teste que le dernier triplet). D'où la complexité $O(nC)$.

On peut majorer C par le nombre de tous les sous-ensembles de S de $0, 1, \ldots, n$ éléments (en réalité il y en a beaucoup moins) :

$$C <= C_|S|^0 + C_|S|^1 + C_|S|^2 + \cdots + C_|S|^n \leq O(n|S|^n)$$

On obtient donc une estimation (exponentielle) $O(n^2|S|^n)$.