Algorithmique — M1

Partiel du 16 novembre 2010

Corrigé

On applique les cours

Exercice 1 (1 point) – *Récurrence*

Étant donné que

$$T(n) = 27T(\lfloor n/3 \rfloor) + 3n^3 + \log n$$

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

Correction. On applique le Master theorem avec $a = 27, b = 3, f(n) = 3n^3 + \log n$. L'exposant est $k = \log_b a = \log_3 27 = 3$, et puisque $f(n) = \Theta(n^k)$ on a la perturbation moyenne. On conclut par Master theorem que :

$$T(n) = \Theta(n^k \log n) = \Theta(n^3 \log n).$$

Exercice 2 (2 points) – *La science*

Un chercheur souhaite assister à un nombre maximum de conférences durant le mois de décembre. Les dates prévues sont :

Le colloque des savants 1-12 décembre.

Le colloque des ignorants 8-10 décembre.

La petite conférence sur les grands graphes 14-15 décembre.

La grande conférence sur les petits graphes 3-19 décembre.

La semaine des automates 18-25 décembre.

La journée des grammaires 11 décembre.

Le workshop de Noël : 24-31 décembre.

1. C'est une instance d'un problème vu en cours. Lequel ?

Correction. Ordonnancement des séances de cinéma dans une salle.

2. Expliquez l'algorithme de cours pour ce problème. (inutile de prouver sa correction)

Correction.
Trier les conférences selon la date de fin.
Plan= vide ; fin=0 ;
pour i de 1 à n :
    Si d[i] >fin
        Plan = Plan U {i} ;
        fin = f[i] ;
retourner Plan

3. Appliquez cet algorithme et trouvez la solution du problème.

Correction. On classe les conférences selon la fin : 8-10 ; 11-11 ; 1-12 ; 14-15 ; 3-19 ; 18-25 ; 24-31. On prend 8-10, puis 11-11, on laisse 1-12 (en conflit), on prend 14-15, on laisse 3-19, on prend 18-25, on laisse la dernière.
Conclusion : on peut assister à 4 conférences maximum : 8-10, 11-11, 14-15 et 18-25

On invente des algorithmes très simples

Exercice 3 (2 points) – *Méthode imposée pour un problème trivial*

Écrire une fonction qui compte le nombre d'occurrences d'un élément e dans un tableau $A[s..f]$.

1. Écrivez un algorithme de type diviser-pour-régner qui résout ce problème.

Correction. On coupera le tableau en deux moitiés, comptera les e dans chaque moitie, et additionnera les résultats :

```
int Compter(A,s,f,e)
    si s=f
        si (A[s]=e) retourner 1, sinon retourner 0
    m=(s+f)/2
    gauche=Compter(A,s,m,e)
    droite=Compter(A,m+1,f,e)
    retourner gauche +droite
```

2. Analysez sa complexité.

Correction. Le problème pour un tableau de taille n est réduit aux deux sous-problèmes de taille $n/2$, et encore $O(1)$ opérations, d'où la récurrence sur la complexité :

$$T(n) = 2T(n/2) + O(1).$$

Par Master theorem (avec $k = \log_2 2$ et perturbation petite), on déduit que $T(n) = O(n)$.

3. Comparez-la avec celle de l'algorithme naïf vu en L1.

Correction. L'algo naïf comporte une boucle de n itération, le corps de boucle se fait en O(1). Sa complexité est donc $O(n)$. Conclusion : l'algo Diviser-pour-régner a la même complexité que l'algo naïf.

```
int CompterL1(A,s,f,e)
    acc=0
    pour i de s à f
        si (A[i]=e) acc++
    retourner acc
```

Exercice 4 (3 points) – *Encore un sac à dos (monotone)*

Nous avons n objets de poids $p[1] \leqslant p[2] \leqslant \cdots \leqslant p[n]$ et de valeur $v[1] \geqslant v[2] \geqslant \cdots \geqslant v[n]$. Il faut trouver un sous-ensemble des objets de poids total $\leqslant W$kg et de valeur maximale.

1. Écrivez un algorithme glouton qui résout ce problème.

Correction. On prendra les objets dans l'ordre tant qu'ils rentrent dans le sac :

```
i=0 ; P=0 ;
tant que i<n et P+p[i+1]<W
        P=P+p[i+1]
        i++
retourner {1,2,...,i}
```

2. Analysez sa complexité.

Correction. L'algo consiste en une simple boucle pour qui se répète n fois, et dons le corps est en O(1). D'où la complexité O(n).

3. Justifiez sa correction.

Correction. Supposons qu'on a une solution optimale S qui contient k éléments. En les remplaçant par $S' = \{1, 2, \ldots, k\}$, on obtient une autre solution optimale, puisque S' pèse moins (ou autant) que S (et donc rentre dans le sac) et vaut plus (ou autant) que S. Il existe donc une solution optimale S' qui serait trouvée par notre algo glouton.

On invente des algorithmes moins simples

Exercice 5 (6 points) – *Un problème NP-complet : chemin d'Hamilton*

Les n villes du pays Back-And-Track-Land sont reliés par un réseau routier. La matrice d'adjacence $M[i,j]$ (de taille $n \times n$) représente ce réseau de façon habituelle : $M[i,j] = 1$ s'il y a une route directe de la ville i vers la ville j ; et $M[i,j] = 0$ s'il n'y en a pas. Mr Hamilton, explorateur célèbre, veut visiter toutes les villes, en passant par chaque ville une seule fois [1]. Aidons-le à planifier son voyage.

Dans cet exercice il faut trouver un algorithme retour-arrière qui trouve un itinéraire si celui-ci existe. On va appeler une solution partielle un chemin de longueur k (qui utilise les routes existantes et ne revisite jamais une même ville). On représentera cette solution partielle par un tableau de numéros de villes $R = [i_1, \ldots, i_k]$.

1. Écrivez une fonction booléenne $test(R, k, M, n)$ qui teste si le tableau $R[k]$ est une solution partielle pour un réseau routier représenté par un tableau (matrice d'adjacence) $M[n, n]$.

Correction. Je préfère que R,M et n soient des variables globales et non les paramètres de la fonction test. Je suppose que tous les éléments de R sont des entiers entre 1 et n.

```
bool fonction test(k)
    pour j de 2 à k
        si R[j] appartient à R[1..j-1] retourner faux (déjà vu)
        si M[R[j-1],R[j]]=0 retourner faux (pas de route)
    retourner vrai
```

Pour notre algorithme de backtracking il vaut mieux utiliser la version incrémentale de la même fonction, en supposant que jusqu'à la position $k-1$ on a une solution partielle. Sous cette hypothèse il suffit de tester seulement la dernière ville visitée :

```
bool fonction testInc(k)
    si k=1 retourner vrai
    si R[k] appartient à R[1..k-1] retourner faux
    si M[R[k-1],R[k]]=0 retourner faux
    retourner vrai
```

2. Quelles sont les solutions partielles de taille 1 ? Comment à partir d'une solution partielle de taille k passer à ses extensions de taille $k+1$? Comment détecter si on a déjà trouvé un chemin pour Mr Hamilton ?

Correction.

–On visite une seule ville. Il y a n solutions partielles suivantes : [1], [2], ..., [n].

–Il faut essayer toutes les valeurs possibles pour R[k+1] : de 1 à n. Lorsque ça passe le testInc(k+1), on a une solution partielle de taille $k+1$.

–Il faut vérifier que $k = n$.

3. Écrivez un algorithme retour-arrière de recherche de chemin d'Hamilton.

Correction. On programme d'abord une fonction récursive DFS(k) qui cherche en profondeur une solution qui étende un chemin partiel R[1..k] déjà trouvé.

```
DFS(k)
    si k=n
      imprimer R
      stop
    pour x de 1 à n
      R[k+1]=x
      si testInc(k+1)
        DFS(k+1)
```

1. Ne pas confondre avec le chemin eulerien : Mr Euler veut parcourir *chaque route* une fois et une seule, tandis que Mr Hamilton veut visiter *chaque ville* une fois et une seule.

Et puis on fait le programme principal qui cherche à partir de chaque ville

```
pour x de 1 à n
    R[1]=x
    DFS(1)
Imprimer(le chemin n'existe pas)
```

4. Estimez la complexité de votre algorithme.

Correction. Au pire cas l'algorithme parcourt toutes les permutations de n villes. Ça donne la complexité $n!$.

Exercice 6 (6 points) – *Encore un cinéma : multiplex*

Demain dans le "Ciné Glouton" il y aura n séances et plusieurs salles. La séance numéro i commence à l'heure $d[i]$ et se termine à l'heure $f[i]$. Il faut planifier toutes les séances en utilisant le moins de salles possible.

1. Écrivez un algorithme qui résout le problème dans le cas général. On pourrait le faire dans l'ordre croissant de $d[i]$.

Correction. On parcourt tous les séances et on les met dans chaque fois dans la première salle disponible. Pour réaliser cette idée on maintient un tableau lib[i] qui contient l'heure de libération de chaque salle i.

```
lib[1..n]=0
pour i de 1 à n
    salle=0
    tant que d[i]<lib[salle]
            salle++
    s[i]=salle
Imprimer ("la liste de salles pour les séances de 1 à n")
Imprimer (s)
```

2. Analysez sa complexité.

Correction. La boucle principale est itérée n fois. A l'intérieur, la recherche de la première salle disponible prend $O(n)$ opérations. D'où la complexité $O(n^2)$.

3. Justifiez sa correction.

Correction. Le problème et les sous-problèmes qu'on doit considérer ont toutes la forme suivante : étant donné la liste de séances $(d[i] - f[i]$, en ordre croissant de $d[i])$ et les heures de libération de chaque salle $lib[j]$, trouver une affectation de tous les films aux salles, qui respecte l'heure de libération de salles, l'exclusion mutuelle pour les films de la même salle, et qui minimise le nombre de salles utilisées.
La preuve de correction doit se baser sur les deux lemmes :
Lemme 1 (Choix glouton). *Il existe une solution optimale qui est compatible avec le choix glouton : film numéro 1 est affecté à la 1ère salle qui convient (son numéro est* $s_1 = \min\{s | lib[s] \leqslant d[1]\}$*).*
Idée de preuve : Supposons qu'on a une solution optimale S qui ne correspond pas au choix glouton. C'est-à dire le 1er film va dans une autre salle s_2 (elle doit aussi satisfaire la contrainte $lib[s_2] \leqslant d[1]$). On peut permuter les salles s_1 et s_2 (tous les films de s_1 seront déplacés vers s_2 et vice versa). Le planning S' ainsi obtenu est correct :
–tout va bien dans la salle s_1 puisque $lib[s_1] \leqslant d[1]$;
–tout va bien dans la salle s_2, puisque la première séance qu'on a mis dans cette salle démarre après $d[1]$ (les films sont triés), et $lib[s_2] \leqslant d[1]$.
On a obtenu un planning S' qui est optimal (il utilise le même nombre de salles que S), et qui correspond au choix glouton. Q.e.d.

Lemme 2 (Sous-structure optimal)*Après le choix glouton du film 1 en salle* s *le problème se réduit au même problème pour les films 2-n et les heures de libération de salle* $lib'[j]$ *définis comme suit :*

$$lib'[j] = \begin{cases} lib[j], & si\ j \neq s; \\ f[s], & si\ j = s. \end{cases}$$

Se lemme est presque évident.
La correction de l'algorithme glouton se déduit de deux lemmes par un raisonnement standard.