Majeure Informatique
Analyse et Conception d'Algorithmes
Contrôle classant

Mercredi 21 Mars 2001
Documents autorisés : polycopiés et notes de cours
Les élèves devront résoudre le Problème 1 et au choix le Problème 2 ou le Problème 3.

1 Ordonnancement

On considère dans cet exercice des tâches $T_1, T_2, \ldots, T_n$ à réaliser sur un processeur séquentiel. Pour chaque tâche T_i, sa durée d'exécution d_i et une date limite souhaitée ℓ_i pour sa réalisation sont données.

On cherche un ordre de réalisation de celles-ci (c'est-à-dire une permutation des indices $T_{\sigma(1)}, T_{\sigma(2)}, \ldots, T_{\sigma(n)}$) qui maximise le nombre de tâches effectuées dans le délai imparti.

1. On suppose que toutes les tâches ont la même date limite. Donner un algorithme glouton pour ce problème. Montrer que c'est un cas particulier du problème du sac à dos, quelle est sa complexité dans ce cas ?

Pour le cas des ℓ_i quelconques, on utilise l'algorithme consistant d'abord à ordonner les tâches par ordre de dates limites croissantes ; l'algorithme gère un sous-ensemble X de $\{1, 2, \ldots, n\}$ contenant les numéros de tâches qui seront réalisées dans le délai souhaité, initialisé à l'ensemble vide. L'algorithme considère itérativement chaque tâche T_i, ajoute i à X, puis vérifie si T_i peut être réalisée dans le délai souhaité ; si ce n'est pas le cas on supprime de X la tâche ayant la durée la plus grande.

2. Appliquer l'algorithme à l'exemple suivant :

Tache	1	2	3	4	5	6
durée	5	8	6	4	7	5
limite	9	15	18	19	21	25

3. Proposer des structures de données pour l'algorithme, calculer sa complexité.

4. Démontrer que l'algorithme donne bien la solution optimale.

On suppose que la durée de chaque tâche est 1 et on se donne un ordre partiel $\prec$ sur les tâches. Une réalisation σ est acceptable si $T_i \prec T_j \quad \Rightarrow \quad \sigma(i) < \sigma(j)$. On appelle ORD-PREC le problème de la recherche d'une réalisation acceptable dont p tâches sont dans les délais. On se propose de démontrer que ce problème est NP-complet. Pour cela, on va montrer que $CLIQUE$ se réduit à ce problème.

Une instance de $CLIQUE$ est donnée par un graphe $G = (X, E)$ et un entier k ; il s'agit alors de savoir si le graphe admet une clique de taille k. A une instance de $CLIQUE$ pour laquelle le graphe a n sommets et m arêtes, on associe un ensemble de $n + m$ tâches $T_e, e \in E$ et $T_x, x \in X$. L'ordre $\prec$ est tel que pour tout $e = \{x, y\}$ on a $T_x \prec T_e$ et $T_y \prec T_e$. Le délai souhaité de tous les T_e est $k(k+1)/2$ et celui des T_x est $n + m$.

5. Quel doit être le nombre de tâches à réaliser dans les délais pour que le graphe G admette une clique de taille k ? Justifier votre réponse. En déduire que ORD-PREC est NP-complet.

2 Algorithme approché pour un recouvrement minimal

Le problème RECOUVREMENT PAR ENSEMBLES s'énonce de la façon suivante :

Données : un ensemble fini E, une famille $(S_i)_{1\leq i\leq k}$ de sous-ensembles de E et pour chaque S_i un poids $w(S_i)$ qui est un entier positif.

Il s'agit de déterminer p sous-ensembles $S_{j_1}, \ldots S_{j_p}$ dont l'union est E et dont la somme des poids est minimale. Soit plus formellement $J = \{j_1, j_2, \ldots, j_p\}$ tel que

$$\cup_{j\in J} S_j = E, \text{ et } \sum_{j\in J} w(S_j) \text{ minimal.}$$

I. Recouvrement par ensembles est NP-complet

1. En utilisant un paramètre supplémentaire w, donner une version de décision de ce problème.

On se propose de montrer que cette version est NP-complète. Pour cela on réduit RECOUVREMENT PAR SOMMETS à ce problème. Etant donné un graphe $G = (V, E)$, on considère le RECOUVREMENT PAR ENSEMBLES de l'ensemble E des arêtes, en associant à chaque sommet v de V un sous-ensemble S_v de poids 1.

2. Comment choisir les ensembles S_v pour que le recouvrement de E par les S_v détermine un recouvrement par sommets du graphe G ?

3. Démontrer que le problème posé est NP-complet.

II. Algorithme approché glouton

On considère l'algorithme glouton suivant qui calcule un recouvrement J (où $|X|$ dénote le nombre d'éléments de X et $X \setminus Y$ l'ensemble des éléments de X qui ne sont pas dans Y) :

- Pour i de 1 à k Calculer $c(S_i) = w(S_i)/|S_i|$.
- $F \leftarrow \emptyset$, $J \leftarrow \emptyset$.
- Tant que $F \neq E$ Faire
  - Soit i tel que $c(S_i)$ soit minimal
  - $F \leftarrow F \cup S_i$, $J \leftarrow J \cup \{i\}$
  - Recalculer les $c(S_i)$ par $c(S_i) = w(S_i)/|S_i \setminus F)|$.

Soit $(T_i)_{1\leq i\leq k}$ les ensembles du recouvrement ordonnés dans l'ordre dans lequel ils sont ajoutés par l'algorithme glouton.

1. Soit W_0 la somme des poids des sous-ensembles S_i qui figurent dans la solution optimale.
Montrer qu'au moins un de ces sous ensembles S_i satisfait $\frac{w(S_i)}{|S_i|} \leq \frac{W_0}{n}$

2. En déduire que $\frac{w(T_1)}{|T_1|} \leq \frac{W_0}{n}$

Soit $x_1, \ldots, x_n$ les éléments de E dans l'ordre dans lequel ils sont ajoutés dans F.

3. Montrer que le sous-ensemble T_i contenant x_j à l'étape où il entre dans F satisfait lors de cette étape : $c(T_i) \leq W_0/(n-j+1)$.

4. En déduire que l'algorithme glouton est H_n approché, où $H_n = 1 + 1/2 + \ldots + 1/n$.

5. Soit les $n+1$ sous-ensembles suivants de $E = \{1, 2, \ldots, n\}$, $S_i = \{i\}$ pour $1 \leq i \leq n$ et $S_{n+1} = E$ et soit $\varepsilon > 0$. Donner des poids à ces sous-ensembles pour que le résultat de l'algorithme glouton diffère de l'optimum d'un facteur $H_n/(1+\varepsilon)$.

III. Recouvrement de mots

Soit Σ un alphabet fini, un mot est une suite finie de lettres de Σ, la longueur d'un mot f est notée $\ell(f)$. On dit qu'un mot g est *facteur* du mot f si $f = f'gf''$, c'est à dire si f contient une suite consécutives de lettres qui forment un mot égal à g.

On s'intéresse dans la suite au problème P suivant :

Soit un ensemble $S = \{f_1, \ldots, f_n\}$ de mots, déterminer le plus court mot f contenant tous les f_i comme facteurs. On se propose d'appliquer les résultats qui précèdent à ce problème.

Deux mots f_i et f_j tels que les k dernieres lettres de f_i forment un mot égal à celui constitué par les k premiéres de f_j sont dits *k-compatibles.* On note dans ce cas σ_{ijk} le mot de longueur $n_i + n_j - k$ obtenu par concaténation de f_i et des $n_j - k$ derniers caractères de f_j (n_i et n_j sont les longueurs de f_i et f_j).

1. Soit $S = \{f_1 = ab^k, f_2 = b^k c, f_3 = b^{k+1}\}$. Que vaut σ_{12k} ? Quel est le f optimal ?

2. On considère l'algorithme glouton qui à chaque étape, détermine deux mots f_i et f_j de S qui sont k-compatibles pour un k maximal, et qui remplace alors f_i et f_j dans S par σ_{ijk}.

L'algorithme termine quand S est réduit à un seul mot. Montrer sur un exemple que cet algorithme est au mieux (1/2)-approché.

On associe à un problème P, donné par $S = \{f_1, \ldots, f_n\}$, un problème Q de recouvrement par sous-ensembles. Pour Q on pose $E = S$ et on construit des sous-ensembles S_u pour tous les mots u obtenus de la façon suivante : u est ou bien dans S ou bien égal à σ_{ijk} pour un couple (f_i, f_j) k-compatible ; S_u est alors l'ensemble des facteurs de u qui sont dans S, il a pour poids $w(S_u) = \ell(u)$ la longueur de u.

3. Soit $S_{\theta_1}, \ldots, S_{\theta_p}$ les ensembles donnant une solution optimale au problème Q, et W_0 le poids de cette solution. Donner une solution de longueur W_0 au problème P.

Soit f le mot solution optimale du problème P, dans lequel on suppose que deux mots de S ne sont pas facteur l'un de l'autre. On renumérote les f_i de sorte que f_i apparaît pour la première fois avant f_j dans f si et seulement si $i < j$. *Par abus de notation, f_i désignera uniquement sa première occurence dans f.*

On pose $b_1 = 1$, et soit e_1 le plus grand indice d'un mot de S qui a une intersection avec f_1 dans f. De même, si $1 \leq e_i < n$, on pose $b_{i+1} = e_i + 1$ et e_{i+1} le plus grand indice tel que $f_{e_{i+1}}$ ait une intersection avec $f_{b_{i+1}}$. On continue jusqu'à ce que $e_t = n$.

4. On note k_i la longueur de l'intersection entre f_{b_i} et f_{e_i}, et $\pi_i = \sigma_{b_i e_i k_i}$. Montrer que $S_{\pi_1}, \ldots, S_{\pi_t}$ est une solution au problème Q. Quel est son poids ?

5. Montrer que π_i et π_{i+2} sont des facteurs de f qui ne s'intersectent pas dans f. En déduire que $\sum_i \ell(\pi_i) \leq 2\ell(f)$.

6. En déduire un algorithme $2H_n$ approché qui trouve un mot f contenant tous les f_i comme facteurs.

3 Arbre de poids minimum en temps linéaire probabiliste

On se propose d'étudier un algorithme probabiliste qui permet de calculer l'arbre couvrant de poids minimum en temps linéaire. Dans tout le problème on suppose que $G = (X, E, w)$ est un graphe ayant n sommets et m arêtes et où chaque arête e a un poids $w(e)$. Pour simplifier on supposera que tous les poids sont différents, il existe ainsi un seul arbre recouvrant de poids minimum.

I. Calcul par contraction d'arêtes

On rappelle que la contraction de l'arête $E = \{x, y\}$ consiste à remplacer les deux sommets x et y par un seul sommet z et les arêtes incidentes à x ou y par des arêtes incidentes à z. Noter que des arêtes multiples peuvent être crées par une contraction.

Pour chaque sommet x de G on note $e_{min}(x)$ l'arête de poids minimum qui est incidente à x et V_{min} l'ensemble de toutes les $e_{min}(x)$.

1. Montrer que toute arête de V_{min} appartient à l'arbre recouvrant de poids minimal de G.

2. Déterminer des valeurs de α_n et β_n telles que

$$\alpha_n \leq |V_{min}| \leq \beta_n$$

Proposer deux exemples de poids sur le cycle à 6 sommets pour lesquels ces bornes sont atteintes.

Soit $Contr(G)$ le graphe obtenu à partir de G en contractant toutes les arêtes de V_{min}. On admettra dans la suite qu'il existe un algorithme en $O(m + n)$ qui calcule $Contr(G)$.

3. Montrer que la donnée de l'arbre couvrant de poids minimum pour $Contr(G)$ permet de construire l'arbre de poids minimum pour G.

4. En déduire un algorithme récursif qui construit l'arbre de poids minimum d'un graphe. Quelle est la complexité de cet algorithme ?

II. Echantillonage

A partir d'un graphe G on construit un graphe $G_{/2}$ en effectuant un tirage à pile ou face équilibré sur chaque arête de G : une arête de G est dans $G_{/2}$ avec probabilité $1/2$.

1. Quel est le nombre moyen d'arêtes dans $G_{/2}$?

Le graphe $G_{/2}$ n'étant pas nécessairement connexe on considère sa *forêt couvrante* F de poids minimal ; elle est constituée des arbres couvrants de poids minimal de chacune de ses composantes connexes. Une arête $e = \{x, y\}$ de G est dite *lourde* pour $G_{/2}$ s'il existe dans F un chemin de x à y composé d'arêtes toutes de poids inférieur à $w(e)$. Un arête qui n'est pas lourde est dite *légère*.

2. Que peut-on dire d'une arête lourde par rapport à l'arbre recouvrant de poids minimal de G ? et d'une arête légère ?

On considère le processus suivant qui considère les arêtes e de G par ordre de poids croissants : à chaque pas un bit aléatoire est tiré ; l'arête e est ajoutée dans $G_{/2}$ si et seulement si le bit est égal à 1 (probabilité $1/2$).

3. Donner un algorithme qui construit itérativement au cours du tirage les ensembles d'arêtes suivants : F, la forêt couvrante minimale de $G_{/2}$ et L, l'ensemble des arêtes légères de G qui ne sont pas dans $G_{/2}$.

On peut considérer le processus aléatoire ci-dessus comme un processus composé de k phases, chacune constituée des tirages entre deux ajouts d'un élément dans F. Au cours de chaque phase sont tirées des arêtes lourdes et des arêtes légères.

4. Montrer que le nombre d'éléments de $F \cup L$ est majoré en moyenne par $2n$ (*les non spécialistes de probabilité pourront admettre ce résultat et passer à la partie suivante*).

III. Algorithme

On utilise l'algorithme suivant de détermination d'un arbre recouvrant de poids minimal :

`MSTProba(G)`

1. Appliquer 3 fois l'algorithme de contraction de V_{min} ; on obtient un graphe G_1 et des ensembles d'arêtes contractées V_1, V_2, V_3.
2. Construire $G_2 = G_{1/2}$.
3. Soit $F_1 =$ `MSTProba(`G_2`)`.
4. Soit G_3 obtenu à partir de G_2 en ne conservant que F_1 et les arêtes légères de G_1.
5. $T_2 =$ `MSTProba(`G_3`)`.
6. `return` $T_2 \cup V_1 \cup V_2 \cup V_3$.

1. Exprimer un majorant du nombre de sommets de G_2, G_3 et du nombre moyen d'arêtes de G_3 en fonction de n. Majorer le nombre moyen d'arêtes de G_2 en fonction de m.

2. On admet que le nombre d'opérations de l'étape 4 de l'algorithme est linéaire en $(n+m)$. En déduire une formule de majoration du nombre d'opérations moyen $T(m, n)$ de `MSTProba(G)` qui fait intervenir des $T(m', n')$ et une fonction linéaire $c(m + n)$.

3. Montrer par récurrence que $T(m, n) \leq 2c(m + n)$.