Devoir Surveillé d'Algorithmique - Maitrise d'Informatique

21 Novembre 2001

Durée: 2 heures.
Documents: Seuls sont autorisés les notes manuscrites de cours et TD.
NB: Toutes les réponses doivent être clairement justifiées.

Exercice 1

Question 1: Comment utiliser l'algorihme de Floyd-Warshall pour détecter la présence de circuits de poids négatifs dans un graphe pondéré G ?

Question 2: Modifier l'algorithme de Dijkstra pour qu'il puisse calculer le nombre des chemins de poids minimal entre s et chaque sommet du graphe. Comment modifier l'algorithme pour qu'il produise le chemin le plus court de poids minimal entre s et les autres sommets.

Exercice 2

L'algorithme de Floyd-Warshall vu en cours permet de calculer les chemins de poids minimaux entre tous les couples de sommets d'un graphe sans circuits de poids négatifs.

Supposons que l'ensemble des sommets est $S = \{1, \ldots, n\}$. On veut maintenant construire, pour chaque couple de sommets (i, j), un chemin de poids mininmum les reliant. Pour cela, on cherche à définir une matrice Π telle que, $\forall i, j \in \{1, \ldots, n\}$, Π_{ij} est le prédécesseur (père) de j dans un chemin de poids minimal allant de i à j. Noter que si $i = j$, on a $\Pi_{ij} = Nil$.

On adopte la même approche que celle utilisé dans l'algorithme de Floyd-Warshall. On construit la matrice Π en calculant itérativement une séquence de matrices $\Pi^0, \Pi^1, \ldots, \Pi^n$ avec $\Pi = \Pi^n$, où, Π_{ij}^k est le prédécesseur de j dans un chemin de poids minimal allant de i à j dont tous les sommets intermédiaires sont dans $\{1, \ldots, k\}$.

Question 1: Donner une définition récursive de Π_{ij}^k.

Question 2: Intégrer le calcul de la matrice Π dans le code de l'algorithme de Floyd-Warshall.

Exercice 3

Soit $G = (S, A)$ un graphe orienté et p une fonction de pondération $p : A \to \mathbb{Z}$, où $\mathbb{Z}$ est l'ensemble des entiers (positifs ou négatifs). On suppose que G n'a pas de circuits, et pour simplifier, qu'il a un unique sommet source s (sommet n'ayant aucun arc incident).

Question 1: Pourquoi l'algorithme de Dijkstra peut etre appliqué à G pour calculer les chemins de poids minimum entre le sommet s et tous les autres ?

On rappelle qu'un tri topologique des sommets d'un graphe sans circuits est une séquence $s_1, \ldots, s_n$ ordonnant totalement l'ensemble S, telle que, $\forall i, j \in \{1, \ldots, n\}$, si $(s_i, s_j) \in A$ alors $i < j$.

On considère une variante de l'algorithme de Dijkstra où le choix des sommets se fait selon un ordre topologique (au lieu d'utiliser l'opération Extract-min).

DAG-POIDS-MIN(G, p, s)

> Trier topologiquement les sommets de G
>
> Initialisation (comme dans Dijkstra)
>
> **Pour tout** sommet u dans l'ordre du tri topologique **faire**
>
> > **Pour tout** sommet $v \in Adjacents(u)$ **faire** $Relaxation(u, v)$

Question 2: Montrer que cet algorithme est correct, c'est-à-dire, que lorsqu'il termine, on a pour tout sommet $v \in S$, $d[v] = \delta(s, v)$. (indication: Soit un chemin de poids min entre s et v. Prouver que pour tous les sommets v_i de ce chemin, on a $d[v_i] = \delta(s, v_i)$. La preuve est par induction, et utilise un des lemmes vus en cours).

Question 3: Evaluer la complexité de cet algorithme.