Année 2007-2008

Examen d'algorithmique – L3

Notes de cours autorisées

10 janvier 2008 – 3h

1 Arbres binaires

1. Etant donné un arbre binaire A contenant des entiers (avec par exemple un champ "`val`"), donner une procédure qui affiche tous les entiers contenus dans A et compris entre deux bornes entières a et b données en arguments.
   (on pourra supposer, comme en cours, que les valeurs entières sont stockées uniquement dans les noeuds internes et que les feuilles sont vues comme des arbres vides (sans valeur)).
2. Adapter la procédure précédente pour le cas où A est un *arbre binaire de recherche*.
   Evaluer sa complexité.
3. Etant donnés un arbre binaire de recherche A (contenant des entiers) et deux entiers a et b, écrire une procédure qui retourne un *arbre binaire de recherche* contenant les valeurs de A comprises entre a et b.
   Evaluer sa complexité.

2 Location d'un camion

On dispose d'un camion que l'on souhaite louer à différentes personnes.

Chaque personne intéressée fait une demande de location (une requête) en donnant : une date [1] de début de location d et une date de fin de location f (avec bien sûr $d < f$).

On suppose qu'il y a n personnes : P_1, ..., P_n. Chaque personne P_i fait une requête $R_i = (d_i, f_i)$ pour louer le camion entre la date d_i et la date f_i (ces deux dates sont inclues dans la période de location). Soit $\mathcal{R}$ l'ensemble des n requêtes considérées $\{R_1, \ldots, R_n\}$.

Il se peut que toutes les requêtes ne puissent pas être satisfaites ensemble : par exemple supposons que $R_1 = (6, 16)$, $R_2 = (4, 7)$ et $R_3 = (20, 38)$, alors si on loue le camion à P_1, il n'est plus possible de le louer à P_2 (et inversement) : il y a chevauchement des requêtes R_1 et R_2.

On dit qu'un sous-ensemble S de $\mathcal{R}$ est compatible si et seulement si il n'y a aucun chevauchement des requêtes de S, c'est-à-dire ssi pour toutes requêtes $R_i = (d_i, f_i)$ et $R_j = (d_j, f_j)$ choisies dans S, on a : soit $i = j$, soit $f_i < d_j$, soit $f_j < d_i$.

L'objectif est de décider comment louer "le mieux possible" (voir ci-dessous) le camion en fonction des n requêtes faites par les différentes personnes, c'est-à-dire de choisir le "meilleur" sous-ensemble S compatible de $\mathcal{R}$.

On distingue deux critères pour le choix de S : on peut chercher à optimiser :
- le nombre de personnes dont on satisfait la requête (*i.e.* le cardinal de l'ensemble S), ou
- la durée totale de location du camion (*i.e.* la somme de la durée des requêtes de S).

[1] Dans cet exercice, on suppose que les dates sont des entiers positifs.

2.1 Exemple

Considérons l'ensemble de requêtes suivant : $\mathcal{R} = \{R_1 = (4,7), R_2 = (1,2), R_3 = (16,19), R_4 = (9,14)\}$.

Est-il possible de satisfaire toutes les requêtes de $\mathcal{R}$?

Et si l'on ajoute la requête $R_5 = (6,8)$?

2.2 Algorithmes

1. Donner un algorithme efficace pour décider s'il est possible de satisfaire **toutes** les requêtes d'un ensemble $\mathcal{R}$. Expliquer votre algorithme et évaluer sa complexité.
2. **Maintenant on souhaite écrire un algorithme efficace pour construire un sous-ensemble compatible de $\mathcal{R}$ qui maximise le *nombre* de requêtes traitées.**
   Pour cela on souhaite utiliser le schéma d'algorithme `Location-1` suivant (NB : il s'agit juste d'un schéma dans la mesure où l'on ne dit pas comment on ordonne les requêtes et cet ordre est crucial pour l'application de l'algorithme car il fixe la manière dont les requêtes seront énumérées) :

Procédure `Location-1`($\mathcal{R}$)
`//`$\mathcal{R} = \{R_1, \ldots, R_n\}$`, avec` $R_i = (d_i, f_i)$ `pour tout` i
begin
  $S := \emptyset$
  Ordonner les requêtes de $\mathcal{R}$ (on utilise cet ordre pour l'énumération ci-dessous.)
  pour chaque $R \in \mathcal{R}$ **faire**
    si $S \cup \{R\}$ *est compatible* **alors**
      $S := S \cup \{R\}$
    retourner S
end

   (a) A quelle famille d'algorithme appartient `Location-1` ?

   (b) On se propose de considérer les trois ordres d'énumération des requêtes suivants :
   - on trie les requêtes par date de début d croissante.
   - on trie les requêtes par date de fin f croissante.
   - on trie les requêtes par durée $f - d + 1$ croissante

   Pour chaque ordre ci-dessus, dire si, utilisé dans l'Algorithme `Location-1`, il permet d'obtenir un algorithme correct pour le problème (donner une preuve si il est correct et évaluer sa complexité ; donner un contre-exemple si il n'est pas correct).

3. **On souhaite maintenant construire un sous-ensemble compatible de $\mathcal{R}$ qui maximise la *durée* totale de location du camion.**

   (a) Montrer que l'algorithme précédent n'est plus correct pour ce nouveau critère quel que soit l'ordre choisi (parmi les 3 cités ci-dessus).

   (b) **Donner un algorithme efficace pour construire un sous-ensemble compatible de $\mathcal{R}$ qui maximise la *durée* totale de location du camion.**
   (On pourra utiliser des graphes valués.)
   Justifier votre algorithme, donner sa complexité.

3 Arbre couvrant minimal

1. Donner un exemple de graphe non-orienté et valué admettant plusieurs *arbres couvrants minimaux différents*.
2. Soit $G = (S, A, w)$ un graphe non-orienté, valué et connexe. Soit T un sous-ensemble d'arêtes de A. On appelle la *liste de poids* de T l'ensemble ordonné des valeurs $\{w(x, y) \mid (x, y) \in T\}$.
   Montrer que tous les arbres couvrants minimaux de G ont la même liste de poids.

   Pour cette démonstration, on pourra **prouver** (et ensuite utiliser) le résultat suivant : Soient deux arbres couvrants distincts T et T' tels que $E = T \cap T' \neq \emptyset$, et soit $(x, y) \in T' \backslash E$. Alors l'algorithme `AC` suivant vérifie les deux propriétés suivantes :
   – une seule arête de $T \backslash E$ ne sera pas ajoutée à S
   – le résultat final S est un arbre couvrant (distinct de T et T').

Algorithme `AC`
begin
  $S := E \cup \{(x, y)\}$
  pour chaque $(u, v) \in T \backslash E$ **faire**
    si $S \cup (u, v)$ *est acyclique* **alors** $S := S \cup \{(u, v)\}$
  retourner S
end

3. Montrer que si la fonction poids w d'un graphe non-orienté, valué et connexe $G = (S, A, w)$ est telle que le poids de chaque arête de A est unique, alors il existe un unique arbre couvrant minimal pour G.

4 Relation d'accessibilité

On considère un graphe orienté $G = (S, A)$ avec $S = \{x_1, \ldots, x_n\}$. On note M la matrice d'adjacence de G, c'est à dire la matrice booléenne $(\alpha_{ij})_{1 \leq i,j \leq n}$ telle que :

$$\alpha_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathit{Vrai} & \text{si } (x_i, x_j) \in A \\ \mathit{Faux} & \text{sinon} \end{cases}$$

On définit la somme et le produit de matrices booléennes de manière standard : la multiplication est remplacée par la conjonction ($\wedge$), et l'addition par la disjonction ($\vee$). Ainsi soient A, B et C trois matrices booléennes $n \times n$ avec les coefficients a_{ij}, b_{ij} ou c_{ij}, on a :

– $C \stackrel{\text{def}}{=} A + B$ ssi $c_{ij} = a_{ij} \vee b_{ij}$;
– $C \stackrel{\text{def}}{=} A \cdot B$ ssi $c_{ij} = \bigvee_{k=1\ldots n} (a_{ik} \wedge b_{kj})$;

On note `Id` la matrice booléenne $(\gamma_{ij})_{1 \leq i,j \leq n}$ telle que $\gamma_{ii} \stackrel{\text{def}}{=} \mathit{Vrai}$ et $\gamma_{ij} \stackrel{\text{def}}{=} \mathit{Faux}$ si $i \neq j$.

On définit la suite de matrice $M^{(k)}$ pour $k = 0, \ldots, n-1$ de la manière suivante :

- $M^{(0)} \stackrel{\text{def}}{=} \texttt{Id}$
- $M^{(k)} \stackrel{\text{def}}{=} M \cdot M^{(k-1)}$ pour $k = 1, \ldots, n-1$.

1. Montrer que $M^{(k)}$, pour $k = 0, \ldots n-1$, représente la matrice d'accessibilité en exactement k transitions de G.
2. Soit M^* la matrice booléenne correspondant à la fermeture réflexive et transitive de la relation induite par G : le coefficient γ_{ij} de M^* est *Vrai* si et seulement si il existe un chemin (de longueur quelconque) de x_i à x_j dans G.
   Montrer que $M^* = \sum_{i=0}^{n-1} M^{(i)}$
   Quelle est la complexité de l'algorithme qui calcule M^* à partir des $M^{(k)}$?
3. Adapter l'algorithme de Floyd pour calculer M^* plus efficacement.
   Donner sa complexité.