Examen d'algorithmique

Mardi 5 janvier 2010 8h30/11h30 – Aucun document autorisé

Questionnaire : Un questionnaire est à remplir sur `http://www.liafa.jussieu.fr/~francoisl/l3algo.html`. Merci d'y répondre rapidement !

Exercice 1 : Arbres binaires de recherche – (4 points [1] :0,5/1,5/2)

On considère des arbres binaires de recherche (munis des opérations clé, FilsG, FilsD et EstVide).

1. Ecrire une fonction `Compter-Clé`(a) qui retourne le nombre de clés contenues dan l'ABR de racine a. Quelle est sa complexité ?
2. Ecrire une fonction `Compter-Clé-Inf`(a, k) qui retourne le nombre de clés inférieures à k contenues dan l'ABR de racine a. Quelle est sa complexité ?
3. Ecrire une fonction `Test-ABR`(a) qui retourne `Vrai` si et seulement si l' arbre binaire de racine a est bien un arbre binaire de recherche. (on pourra définir d'autres fonctions pour résoudre cette question)

Exercice 2 : Graphes – (3 points :0,5/0,5/0,5/1,5)

On considère un graphe orienté $G = (S, A)$ **représenté sous la forme d'une matrice booléenne d'adjacence** (*i.e.* le coefficient $\alpha_{i,j}$ vaut `Vrai` ssi $(q_i, q_j) \in A$).
Un sommet p est un *puits* de G si (1) pour chaque $q \in S$ différent de p, on a : $(q, p) \in A$ et $(p, q) \notin A$; et (2) $(p, p) \notin A$.

1. Peut-il y avoir plusieurs plusieurs puits dans un graphe ?
2. Ecrire une fonction `Test-puits`(p) qui teste si p est un puits.
3. En déduire un algorithme qui détecte la présence d'un puits dans G. Evaluer sa complexité.
4. Proposer un autre algorithme plus efficace (en $O(|S|)$).

Exercice 3 : Relation d'accessibilité – (4 points :1/1/2)

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 $(\alpha_{ij})_{1 \leqslant i,j \leqslant n}$ telle que :

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

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 $c_{ij} = \bigvee_{k=1\ldots n} (a_{ik} \wedge b_{kj})$;

1. Le barème est donné à titre indicatif.

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

Problème : Plus courts chemins – (9 points : 2/4/3)

On s'intéresse au calcul des distances des plus courts chemins (PCC) dans un graphe orienté et valué $G = (S, A, w)$. L'objectif est ici de calculer ces distances pour toutes les paires de sommets (comme le fait l'algorithme de Floyd) : à la fin de l'algorithme, on veut connaître $\delta_G(x, y)$ pour tout $x, y \in S$. (NB : comme en cours, on note $\delta_G(x, y)$ la distance d'un PCC entre x et y dans le graphe G).

On rappelle en annexe les algorithmes de Dijkstra et de Bellman-Ford.

Dans toute la suite on suppose fixé un graphe $G = (S, A, w)$. **On demande donc de résoudre les questions pour un tel graphe G général, à l'exception des questions "Application" qui portent sur le graphe exemple M de la figure 1.**

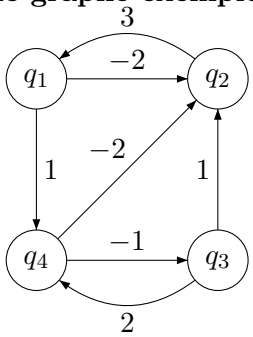

FIGURE 1 – Graphe M.

1. Questions de cours.

1. Etant donnés deux sommets u et v, donner les conditions nécessaires et suffisantes pour qu'il existe un plus court chemin entre u et v dans G.
2. Donner la complexité des algorithmes de Dijkstra et de Bellman-Ford.
   Quelle est la différence importante entre ces deux algorithmes ?
3. Que signifie le booléen renvoyé par l'algorithme de Bellman-Ford ?

2. Propriétés. Pour toute fonction $f : S \to \mathbb{R}$ qui associe à chaque sommet de S une valeur dans $\mathbb{R}$, on définit le graphe $G_f = (S, A, w_f)$ où $w_f(u,v) \stackrel{\text{def}}{=} w(u,v) + f(u) - f(v)$. G_f a donc les mêmes sommets et les mêmes arêtes que G, seuls les poids des arêtes changent.

1.a Application : En prenant le graphe M de la figure 1 et en prenant $f(q_1) = 2$, $f(q_2) = -1$, $f(q_3) = 4$ et $f(q_4) = 0$, construire le graphe M_f.

1.b Soit $\rho = u \to \ldots \to v$ un chemin de u à v dans G (et G_f).
Montrer que ρ est un plus court chemin entre u et v dans G **si et seulement si** ρ est un plus court chemin de u à v dans G_f.

1.c Montrer que G contient un cycle strictement négatif si et seulement si G_f contient un cycle strictement négatif.

On définit maintenant le graphe $\overline{G} = (S', A', w')$ de la manière suivante :

- $S' \stackrel{\text{def}}{=} S \cup \{x_0\}$ où x_0 est un nouveau sommet ;
- $A' \stackrel{\text{def}}{=} A \cup \{(x_0, u) \mid u \in S\}$; et
- $w'(u,v) \stackrel{\text{def}}{=} \begin{cases} w(u,v) & \text{si } u, v \in S \\ 0 & \text{si } u = x_0 \text{ et } v \in S \end{cases}$

Pour tout sommet $u \in S$, on note alors $h(u)$ la distance d'un plus court chemin de x_0 à u dans $\overline{G}$ (c'est-à-dire $h(u) \stackrel{\text{def}}{=} \delta_{\overline{G}}(x_0, u)$).

2.a Application : Dessiner $\overline{M}$ pour le graphe M de la figure 1. Donner la valeur de $h(q_1)$, $h(q_2)$, $h(q_3)$ et $h(q_4)$.

2.b Montrer que G contient un cycle strictement négatif si et seulement si $\overline{G}$ contient un cycle strictement négatif.

On suppose pour les 4 questions ci-dessous (2.c, 2.d, 3 et 4) qu'il n'y a pas de cycle strictement négatif dans G. Etant donnée la fonction h donnant la distance des PCC entre x_0 et les sommets de S dans $\overline{G}$, on va utiliser le graphe $G_h = (S, A, w_h)$.

2.c Application : Dessiner le graphe M_h de l'exemple.

2.d Montrer que l'on a toujours $w_h(u,v) \geqslant 0$ pour tout $(u,v) \in A$ (NB : on rappelle qu'il n'y a pas de cycle strictement négatif dans G).

3. Quel algorithme faut-il utiliser pour trouver la distance d'un PCC entre u et v dans G_h ?

4. Que vaut la distance d'un PCC entre u et v dans G (*i.e.* $\delta_G(u,v)$) en fonction de la distance d'un PCC entre u et v dans G_h (*i.e.* $\delta_{G_h}(u,v)$) et de la fonction h ?

3. Algorithme. L'algorithme de Johnson est décrit par l'algorithme 1.

1. Application : Appliquer l'algorithme de Johnson sur le graphe de la figure 1.
2. Expliquer ce que fait cet algorithme. Que calcule-t-il ? Pourquoi utilise-t-il d'abord l'algorithme de Bellman-Ford ? Quelle est sa complexité ?
3. Pour le même objectif, quelle serait la complexité d'un algorithme qui n'utiliserait que Bellman-Ford pour les différents calculs ?
4. Quand l'algorithme de Johnson est-il plus intéressant que l'algorithme de Floyd ?

Procédure `Algo-Johnson`(G)
`//`$G = (S, A, w)$ `: un graphe orienté, valué avec` $w : A \to \mathbb{R}$.
begin
  $D \stackrel{\text{def}}{=}$ `matrice`$(S \times S)$ telle que $d_{uv} \stackrel{\text{def}}{=} \infty$ pour tout $u, v \in S$
  $\overline{G} \stackrel{\text{def}}{=}$ `Graphe`$(S \cup \{x_0\}, A \cup \{(x_0, u) \mid u \in S\}; \overline{w})$ avec $\overline{w}(u,v) \stackrel{\text{def}}{=} \begin{cases} w(u,v) & \text{si } u, v \in S \\ 0 & \text{si } u = x_0 \end{cases}$
  si `PCC-Bellman-Ford`$(\overline{G}, x_0, \mathtt{d}) ==$ `Faux` **alors**
    Afficher("le graphe G contient un cycle strictement négatif.")
  sinon
    soit $h(v) \stackrel{\text{def}}{=} \mathtt{d}(x_0, v)$ pour tout $v \in S$
    soit $w_h(u,v) \stackrel{\text{def}}{=} w(u,v) + h(u) - h(v)$ pour tout $u, v \in S$
    $G_h \stackrel{\text{def}}{=}$ `Graphe`(S, A, w_h)
    pour chaque $u \in S$ **faire**
      `PCC-Dijkstra`$(G_h, u, \mathtt{d})$
      pour chaque $v \in S$ **faire** $d_{uv} := \mathtt{d}[v] + h(v) - h(u)$
  return D
end

Algorithme 1 : algorithme de Johnson

Annexe

On rappelle ici les algorithmes de Dijkstra et de Bellman-Ford vus en cours et en TD. Ici on ne s'intéresse qu'aux distances et non aux chemins, on n'utilise donc pas le tableau des prédécesseurs Π.

Pour ces deux algorithmes, on suppose que $\mathsf{d}[-]$ est une variable (un tableau indicé par les éléments de S) utilisée pour stocker les résultats, c'est-à-dire les distances des PCCs : après l'appel de ces fonction, $\mathsf{d}[v]$ contient la distance d'un PCC entre s et v.

L'algorithme 2 décrit l'algorithme de Dijkstra, il utilise une file de priorité F avec une fonction de mise à jour "étendue" (MaJ-F-Dijkstra) lorsque l'on diminue la priorité d'un élément (le tableau IndiceDansF sert alors à accéder directement à la position dans F de cet élément). Cette opération se fait en temps $O(\log(n))$ où n est le nombre d'éléments dans F. Et la création de la file se fait en temps $O(n)$.

Procédure `PCC-Dijkstra`(G, s, d)
`//`$G = (S, A, w)$ `: un graphe orienté, valué avec` $w : A \to \mathbb{R}_+$.
`//`$s \in S$ `: un sommet origine.`
`//d : un tableau indicé par` S `pour stocker les résultats`
begin
  pour chaque $u \in S \backslash \{s\}$ **faire** $\mathsf{d}[u] := \infty$
  $\mathsf{d}[s] := 0$
  $F :=$ File$(S, d,$ IndiceDansF$)$
  tant que $F \neq \emptyset$ **faire**
    $u :=$ Extraire-Min(F)
    IndiceDansF$[u] := -1$
    pour chaque $(u, v) \in A$ **faire**
      si $d[v] > d[u] + w(u, v)$ **alors**
        $\mathsf{d}[v] := \mathsf{d}[u] + w(u, v)$
        MaJ-F-Dijkstra$(F, d, v,$ IndiceDansF$)$
end

Algorithme 2 : algorithme de Dijkstra

Fonction `PCC-Bellman-Ford`(G, s, d) : booléen
`//`$G = (S, A, w)$ `: un graphe orienté, valué avec` $w : A \to \mathbb{R}$.
`//`$s \in S$ `: un sommet origine.`
`//d : un tableau indicé par` S `pour stocker les résultats`
begin
  pour chaque $u \in S \backslash \{s\}$ **faire** $\mathsf{d}[u] := \infty$
  $\mathsf{d}[s] := 0$
  pour chaque $i = 1 \ldots |S|$ **faire**
    pour chaque $(u, v) \in A$ **faire** $\mathsf{d}[v] := \min(\mathsf{d}[v], \mathsf{d}[u] + w(u, v))$
  pour chaque $(u, v) \in A$ **faire**
    si $d[v] > d[u] + w(u, v)$ **alors**
      return *Faux*
  return *Vrai*
end

Algorithme 3 : algorithme de Bellman-Ford