Examen d'algorithmique

Lundi 14 juin 2010 8h30/11h30 – Aucun document autorisé

Exercice 1 : Arbres binaires de recherche – (6 points [1] :1/1/1/1/2)

On considère des arbres binaires avec les opérations habituelles : clé, FilsG, FilsD, EstVide.

1. Ecrire une procédure `Affiche-Clés-Feuilles`(a) qui affiche les clés contenues dans les feuilles (c.-à-d. les noeuds contenant une clé et dont les deux sous-arbres sont des arbres vides) d'un arbre binaire a.
2. Appliquer l'algorithme `Truc` ci-dessous à l'arbre binaire de la figure 1.

```
Procédure Truc(a)
begin
    si EstVide(a) alors
        retourner 0;
    sinon
        clé(a) := clé(a) + Truc(FilsG(a)) + Truc(FilsD(a));
        retourner clé(a);
end
```

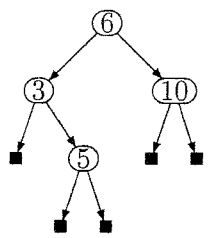


FIGURE 1 – Exemple pour la question 2.

3. Ecrire une fonction `Moyenne-Clé`(a) qui retourne la *valeur moyenne* des clés contenues dans un ABR non vide a. Quelle est sa complexité ?
4. Etant donnés un arbre binaire de recherche a (contenant des entiers) et deux entiers k_1 et k_2, écrire une fonction `Extraire-ABR`(a, k_1, k_2) qui retourne un *arbre binaire de recherche* contenant les clés de a comprises entre k_1 et k_2.
   Evaluer sa complexité.
5. Soit a un arbre binaire contenant des clés distinctes dans chaque sommet. Montrer que a est un arbre binaire de recherche si et seulement si un parcours infixe de a affiche les clés dans l'ordre croissant.

Exercice 2 : Graphes – (4 points :2/2)

1. Proposer un algorithme renvoyant, s'il existe, un circuit de longueur minimale passant par un sommet s donné dans un graphe G. Evaluer sa complexité.
2. Proposer un algorithme qui vérifie si un graphe non orienté et connexe possède un cycle. Evaluer sa complexité.

1. Le barème est donné à titre indicatif.

Problème : Plus courts chemins – (10 points : 7/3)

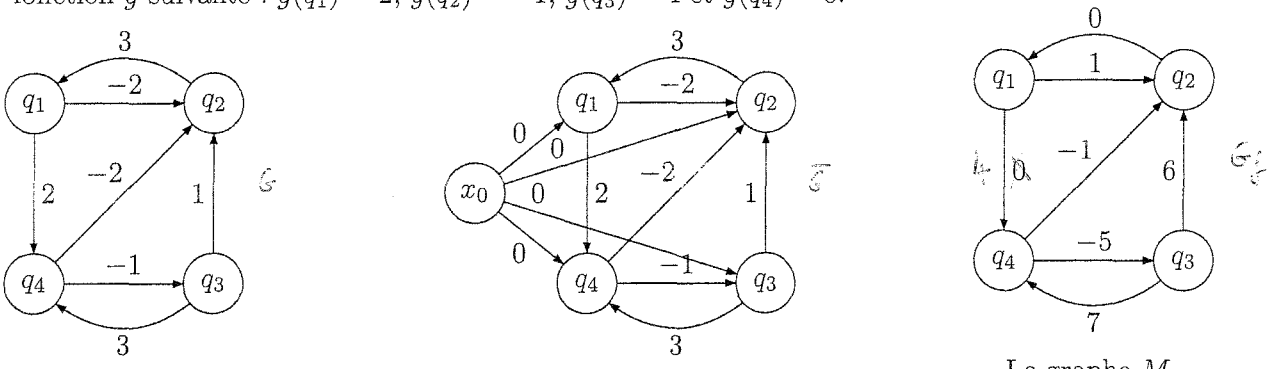
On s'intéresse au calcul des distances des plus courts chemins (PCC) dans un graphe orienté et valué $G = (S, A, w)$. L'objectif est de calculer, **lorsqu'elles existent**, ces distances pour toutes les paires de sommets : à la fin de l'algorithme, on veut connaître $\delta_G(x, y)$ pour tout $x, y \in S$ lorsque des PCC existent (comme en cours, on note $\delta_G(x, y)$ la distance d'un PCC entre x et y dans G). On rappelle en annexe les algorithmes de Dijkstra et de Bellman-Ford.

Etant donné un graphe $G = (S, A, w)$, on définit le graphe $\overline{G} = (S', A', w')$ tel que $S' \stackrel{\text{def}}{=} S \cup \{x_0\}$ où x_0 est un nouveau sommet, $A' \stackrel{\text{def}}{=} A \cup \{(x_0, u) \mid u \in S\}$ et :

$$w'(u, v) \stackrel{\text{def}}{=} \begin{cases} w(u, v) & \text{si } u, v \in S \\ 0 & \text{si } u = x_0 \text{ et } v \in S \end{cases}$$

Et si f est une fonction qui associe à chaque sommet de S une valeur dans $\mathbb{R}$ (c.-à-d. $f : S \to \mathbb{R}$), 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 transitions que G, seuls les poids des arcs changent.

La figure 2 présente un graphe M, son graphe associé $\overline{M}$ ainsi que le graphe M_g pour la fonction g suivante : $g(q_1) = 2$, $g(q_2) = -1$, $g(q_3) = 4$ et $g(q_4) = 0$.



Le graphe M — Le graphe $\overline{M}$ — Le graphe M_g avec $g(q_1) = 2$, $g(q_2) = -1$ et $g(q_3) = 4$ et $g(q_4) = 0$

FIGURE 2 – Exemple

1. Propriétés. Pour les questions suivantes, on suppose donnés un graphe $G = (S, A, w)$ et une fonction $f : S \to \mathbb{R}$.

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

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

2.a Donner les valeurs : $\delta_{\overline{M}}(x_0, q_1)$, $\delta_{\overline{M}}(x_0, q_2)$, $\delta_{\overline{M}}(x_0, q_3)$, et $\delta_{\overline{M}}(x_0, q_4)$.

2.b Construire le graphe M_h où h est la fonction suivante : $h(q_i) \stackrel{\text{def}}{=} \delta_{\overline{G}}(x_0, q_i)$ pour $i = 1, \ldots, 4$.

Pour les 3 questions suivantes (3.a, 3.b, et 3.c), on suppose donné un graphe G sans cycle strictement négatif.

On note $h : S \to \mathbb{R}$ la fonction donnant la distance des PCC entre x_0 et les sommets de S dans $\overline{G}$: c'est-à-dire $h(u) = \delta_{\overline{G}}(x_0, u)$ pour tout $u \in S$. Et on va utiliser cette fonction pour le graphe $G_h = (S, A, w_h)$ défini comme précédemment.

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

3.c Exprimer 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.

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

1. Appliquer l'algorithme de Johnson sur le graphe de la figure 2 (on de détaillera pas les appels des procédures `PCC-Bellman-Ford` et `PCC-Dijkstra`.
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`$(\overline{G}, x_0, \mathsf{d}) ==$ `Faux` **alors**
    Afficher("le graphe G contient un cycle strictement négatif.")
  sinon
    soit $h(v) \stackrel{\text{def}}{=} \mathsf{d}(x_0, v)$ 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, d)
      pour chaque $v \in S$ **faire** $d_{uv} := \mathsf{d}[v] + h(v) - h(u)$
  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 fonctions, $\mathsf{d}[v]$ 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