Examen d'algorithmique

Vendredi 19 juin 2020
A déposer sur Moodle avant samedi 20 juin 10h.
Document autorisé : le polycopié.

Mode d'emploi : Le barème est donné à titre indicatif. **La qualité de la rédaction sera très fortement prise en compte pour la note.** On peut toujours supposer une question résolue et passer à la suite.

Exercice 1 : Circulez ! – (5 points)

Dans cet exercice, on dispose d'un graphe représentant des axes routiers : il s'agit d'un graphe orienté doublement valué $G = (S, A, \ell, v)$: S est un ensemble de ville, $A \subseteq S \times S$ un ensemble d'arcs et :

— $\ell : A \to \mathbb{N}$ associe à chaque arc sa distance (en km) ;
— $v : A \to \mathbb{N}$ associe à chaque arc la vitesse maximale autorisée sur la route représentée par cet arc (en km/h).

1. Etant donné un noeud (une ville) x, on veut calculer les distances des plus courts chemins entre x et toutes les autres villes de S. Comme d'habitude, on cherche construire une table dist telle que $\mathsf{dist}[y]$ contienne la distance d'un PCC entre x et y, ainsi qu'une table Π telle que $\Pi[y]$ donne le prédécesseur de y le long d'un PCC de x à y. On veut donc écrire un algorithme `distance-min`(x, G) qui renvoie (dist, Π).
   Suggérer un algorithme pour résoudre ce problème ? Donner sa complexité.
2. On souhaite maintenant calculer la durée minimale pour aller de x à n'importe quelle autre ville du graphe (on suppose qu'on voyage à la vitesse indiquée par v). On veut donc construire une table temps telle que $\mathsf{temps}[y]$ corresponde au temps minimal pour aller de x à y, et une table Γ telle que $\Gamma[y]$ indique un prédécesseur de y le long d'un chemin de temps minimal entre x et y.
   Proposer un algorithme `temps-min`(x, G) pour résoudre ce problème. Donner sa complexité.
3. Donner un exemple où les résultats pour Π et Γ ne sont pas identiques.
4. On suppose désormais que des routes peuvent être fermées. Etant donnés un tableau de booléen ferme qui indique si l'arc (y, z) est fermé ou non ($\mathsf{ferme}[(y, z)]$ vaut $\top$ ou $\bot$), une table des prédécesseurs Θ représentant des chemins depuis un sommet x (comme Π ou Γ) et un sommet y, écrire une procédure $\mathsf{test}(\mathsf{ferme}, \Theta, y)$ qui renvoie $\top$ ssi on peut aller de x à y en utilisant uniquement des arcs de la table Θ qui ne sont pas fermés. Donner sa complexité.

Exercice 2 : Expliquez ! - (4 points)

Si la valeur de *votre-numéro-d'étudiant*%4 est égale à...

0 alors expliquer l'algorithme de Prim
1 alors expliquer l'algorithme de Bellman-Ford
2 alors expliquer l'algorithme de parcours en largeur
3 alors expliquer l'algorithme de recherche de composantes fortement connexes

Par « expliquer », on entend une présentation en 10 ou 15 lignes en français, suivie d'un exemple, puis d'un problème réel (de la vie de tous les jours !) pour lequel cet algorithme peut être utilisé pour trouver des solutions.

Exercice 3 : Flots - (7 points)

Ici on veut modifier l'algorithme de Ford-Fulkerson sur des réseaux de transport pour essayer de garantir que les chemins améliorants choisis fournissent toujours des valeurs (δ) assez grandes. Pour cela on cherchera des chemins améliorant [1] dans le graphe des augmentations en se restreignant aux arcs ayant des poids supérieurs à une certaine borne. Dans la suite, pour un nombre Δ fixé, on note $G_{\varphi,\Delta}$ le graphe des augmentations G_φ restreint aux arcs de poids supérieur ou égal à Δ (donc tous les arcs ayant des "petits" poids sont oubliés dans $G_{\varphi,\Delta}$).

On commence par initialiser Δ avec la plus grande puissance de 2 pour laquelle il existe au moins un arc issu de e ayant une capacité supérieure à Δ. Puis on cherche des chemins améliorants dans les graphes $G_{\varphi,\Delta}$ pour augmenter φ et quand il n'y en a plus, on remplace Δ par $\Delta/2$, *etc.* L'objectif est de renvoyer un flot maximal (dont la valeur est notée $V_{\varphi_{max}}$).

On décrit ci-dessous l'algorithme Δ-FF :

Algorithme Δ-FF(G)
//$G = (S, A, \mathcal{C})$ `réseau de transport`
begin

1 $\varphi := \{(x, y) \mapsto 0 \mid (x, y) \in A\}$;
2 $\Delta := \max\{2^p \mid \exists (e, v) \in A \,.\, \mathcal{C}(e, v) \geqslant \Delta\}$;
3 **tant que** $\Delta \geqslant 1$ **faire**
4 **tant que** *il existe un chemin* $\rho : e \to^*_{G_{\varphi,\Delta}} s$ **faire**
5 Améliorer φ avec le chemin ρ ;
6 $\Delta := \Delta/2$;
7 **return** (φ) ;

end

1. Appliquer l'algorithme Δ-FF sur le réseau de transport de la figure 1. On détaillera à chaque étape la valeur courante de Δ, les graphes $G_{\varphi,\Delta}$ considérés et les chemins améliorants trouvés.

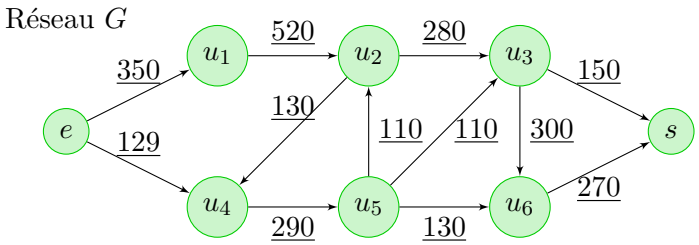


FIGURE 1 – Exemple de réseau pour l'algorithme Δ-FF.

2. Expliquer pourquoi l'algorithme est correct, c'est-à-dire qu'il donne un flot maximal.
3. On veut désormais montrer que la complexité de l'algorithme Δ-FF est en $O(|A|^2 \cdot \log_2(C_{max}))$ pour un réseau $G = (S, A, \mathcal{C})$ où C_{max} est la plus grande capacité utilisée dans G. Pour cela, on va montrer plusieurs résultats.
   (a) Donner une borne maximale sur le nombre de fois que l'on peut franchir le test de la boucle ligne 3 ("Tant que $\Delta \geqslant 1$").

1. Ici un chemin améliorant pour un flot φ d'un réseau de transport G est un chemin de e à s dans le graphe des augmentations G_φ.

(b) Donner une borne minimale sur l'accroissement du flot φ que permet un chemin améliorant dans le graphe des augmentations modifié $G_{\varphi,\Delta}$.

(c) On note Δ_0 la valeur initiale pour Δ fixée à la ligne 2 de l'algorithme. Montrer que le nombre de tours de boucle interne (ligne 4) lorsque Δ vaut Δ_0 est borné par $|A|$.

(d) On cherche maintenant à borner le nombre de tours de boucle interne (ligne 4) pour les valeurs $\Delta \neq \Delta_0$. Pour cela, on va admettre le résultat suivant : **lorsque l'on sort** de la boucle interne de la ligne 4, la valeur du flot obtenu V_φ vérifie : $V_{\varphi_{max}} - V_\varphi \leqslant |A| \cdot \Delta$, c'est-à-dire que la différence entre le flot maximal et le flot courant est inférieure à $|A| \cdot \Delta$.
En déduire que le nombre de tours de boucle interne (ligne 4) pour une valeur $\Delta \neq \Delta_0$ est borné par $2 \cdot |A|$.

(e) En déduire que la complexité de Δ-FF est bien en $O(|A|^2 \cdot \log_2(C_{max}))$.

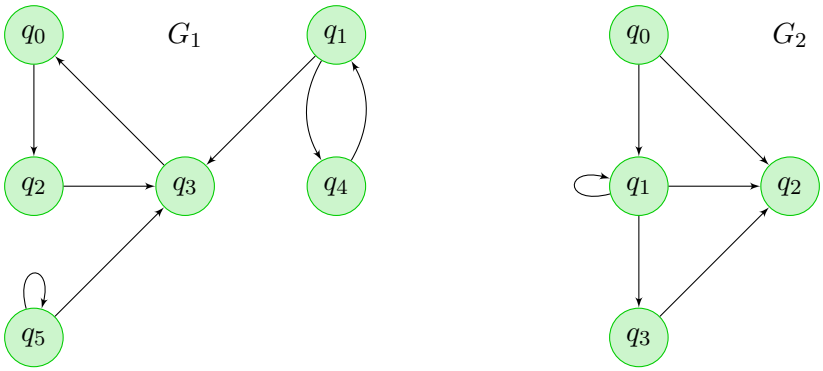
Exercice 4 : Parcours – (4 points)

Dans cet exercice, on va étudier l'algorithme A ci-dessous, où $G = (S, A)$ est un graphe orienté.

Algorithme A(G)
//$G = (S, A)$ `graphe orienté`
begin
  $Z = \emptyset$;
  pour chaque $x \in S$ **faire**
    Couleur[x] := blanc ;
  pour chaque $x \in S$ **faire**
    si *Couleur*[x] = *blanc* **alors**
      $Z := Z \cup$ `Aux`(G, x) ;
  $G' = (S, A \setminus Z)$;
  return (G') ;
end

Procédure `Aux`(G, s)
begin
  res := $\emptyset$;
  Couleur[s] := gris;
  pour chaque $(s, u) \in A$ **faire**
    si *Couleur*[u] = *blanc* **alors**
      res := res $\cup$ `Aux`(G, u) ;
    sinon si *Couleur*[u] = *gris* **alors**
      res := res $\cup \{(s, u)\}$;
  Couleur[s] := noir;
  return res ;
end

1. Appliquer l'algorithme A sur les deux graphes G_1 et G_2.



2. Expliquer ce que fait l'algorithme.
3. A quelle condition sur G, l'algorithme A appliqué à G est déterministe, c'est-à-dire renvoie toujours le même résultat quelle que soit la manière d'énumérer les sommets dans l'algorithme.
4. G' a une propriété particulière : laquelle ? Justifier votre réponse.