Algorithmique II
Année 2007-2008

Jean-François Pineau
Jean-Baptiste Rouquier

Partiel

Relations diplomatiques

Chaque réponse sera soigneusement justifiée. La correction prendra en compte la concision des réponses et la complexité des algorithmes.

Les graphes seront codés, au choix, par matrice d'adjacence ou par liste d'adjacence. Ils sont finis.

Même s'il y a un fil conducteur, les 6 exercices sont indépendants.

Il est demandé de rendre 2 copies séparées à la fin de ce partiel.

1 Marquons les frontières (à rendre sur la copie 1)

Exercice 1 En visitant les capitales du continent, nous voici à la frontière d'un pays. Après une longue période de paix, un grand réseau de routes s'est développé dans cette région, permettant d'aller dans toutes les villes de tous les pays. Si développé qu'il existe pour chaque route r deux circuits disjoints passant par cette route. « Disjoint » signifie que les deux circuits n'ont pas de route en commun (sauf bien sûr r).

Cependant, on voudrait à présent reconstruire des postes de douanes à la frontière.

Montrer qu'il existe au moins 3 routes qui traversent la frontière.

Solution :

Première Solution : *Soit $G = (X, E)$ le graphe représentant la région. Soit $(x, y) \in X$, et μ un chemin de x à y. Soit $(e_1, e_2) \in E$, $e_1 = (u_1, v_1), e_2 = (u_2, v_2)$.*

- *$(e_1, e_2) \notin \mu$: alors x et y reste dans la même composante connexe de $G \setminus \{e_1, e_2\}$*
- *$e_1 \in \mu, e_2 \notin \mu$: Par hypothèse, il existe 2 chemins arêtes disjoints p_1 et $p_2 \neq \{e_1\}$ reliant u_1 à v_1.*
  1. *Si $e_2 \in p_1$, alors $\mu_{x \leftarrow u_1}.p_2.\mu_{v_1 \leftarrow y}$ est un chemin de x à y.*
  2. *Sinon, $\mu_{x \leftarrow u_1}.p_1.\mu_{v_1 \leftarrow y}$*
- *Si e_1 et $e_2 \in \mu$: on trouve p_1 de u_1 à v_1 ne contenant pas e_2, et p_2 de u_2 à v_2 ne contenant pas e_1. Alors, le chemin $\mu_{x \leftarrow u_1}.p_1.\mu_{v_1 \leftarrow u_2}.p_2.\mu_{v_2 \leftarrow y}$ est un chemin reliant x à y dans $G \setminus \{e_1, e_2\}$.*

Comme le choix de x et y est quelconque, $G \setminus \{e_1, e_2\}$ reste connexe.

Deuxième Solution : *Soit S le sous-ensemble de X représentant l'un des pays, $S \neq X, S \neq \varnothing$. Montrons qu'il faut couper plus de 3 arêtes pour séparer S de $X \setminus S$. Comme S contient au moins un sommet, et que le graphe est connexe, il y a au moins une arête entre S et $X \setminus S$. Soit $e = (u, v)$ une telle arête avec $u \in S$. Par hypothèse, il existe deux chemins p_1 et p_2, arêtes disjoints entre u et v, ne comportant pas e. Soient, pour $i \in \{1, 2\}$, (u_i, v_i) l'arête de p_i telle que u_i soit le dernier sommet de S dans p_i. Ainsi ces arêtes passent d'un sommet de S à un sommet de $X \setminus S$. Par hypothèse, elles sont différentes de e, donc il existe 3 arêtes connectant S à $X \setminus S$.*

□

2 War's graphs (à rendre sur la copie 1)

Exercice 2 C'est la guerre!!! Rien ne va plus entre les deux pays, et une armée s'apprête à débarquer. Afin d'arrêter cette invasion, il faut détruire soit les routes, soit les carrefours, afin que le réseau ne soit plus connexe. Mais ne disposant pas de beaucoup de temps (et pour penser déjà à l'après guerre), il faut minimiser le nombre d'endroits à détruire. (*Note : On ne suppose plus ici avoir 2 circuits disjoints passant par une route. Le coût de destruction d'une route est identique à celui d'un carrefour*).

1. Est-il préférable de détruire des routes, ou des carrefours ? Y a-t-il un cas où la différence est flagrante ? Peut-on transformer un plan visant à détruire n routes en un plan visant à détruire n' carrefours, $n' \leqslant n$?

   Solution : *Soit $G = \{X, E\}$ un graphe. On note :*

   $$k(G) = \min\{|S|, S \subseteq X \text{ tel que } G \setminus S \text{ non connexe}\}$$

   $$k'(G) = \min\{|F|, F \subseteq E \text{ tel que } G \setminus F \text{ non connexe}\}$$

   Si G est une clique, alors $k(G)$ n'est pas défini.
   Si G n'est pas une clique, alors soit $F \subseteq E$ de cardinal minimum $k'(G)(< n-1)$ qui déconnecte G. Puisque G n'est pas une clique, soit x et y à distance supérieure ou égale à 2 déconnecté par F (Si ce n'est pas le cas, alors tous les éléments d'une composante connexe sont reliés aux éléments de l'autre, ce qui représente au moins $n-1$ arêtes. Or, G n'est pas une clique, donc il existe un sommet ayant au plus $n-2$ arêtes qu'il suffirait de déconnecter pour avoir un ensemble F' de cardinal inférieur, ce qui entraine une contradiction).
   $\forall e = (u, v) \in F$, on note :

   $$\nu(e) = \begin{cases} u & \text{si } v = x \text{ ou } y. \\ v & \text{si } u = x \text{ ou } y. \\ u & \text{sinon} \end{cases}$$

   et $S = \{\nu(e), e \in F\}$. Alors $|S| \leqslant |F|$, et S déconnecte x et y. Donc $k(G) \leqslant |S| \leqslant k'(G)$.
   La différence est flagrante dans le cas d'un nœud papillon : deux cliques à n sommets C_1 et C_2, un $(2n+1)^e$sommet x, tous les sommets de C_1 et de C_2 reliés à x. Il faut couper $n = \Omega(\sqrt{|E|})$ arêtes pour déconnecter le graphe, mais il suffit de supprimer un sommet (x).

   □

2. Quelqu'un réalise alors que chaque carrefour relie exactement 3 routes. Y a-t-il encore une différence de coût entre détruire les routes ou les carrefours ?

   Solution :
   Soient $S \subseteq V$ qui réalise $k(G)$, et A et B 2 composantes connexes de $G - S$. $S = \Gamma(A) = \Gamma(B)$. Soit $x \in S$, et $\nu(x)$ l'arête de x à A si x n'a qu'un voisin dans A, l'arête de x à B sinon (G est 3-régulier). Alors $|F| \leqslant |S|$. Soit μ de a à b, $a \in A$, $B, \in B$. $\exists x \in S \cap \mu$. $\nu(x) \in \mu$ donc μ est déconnecté par F. μ étant quelconque, $k'(G) \leqslant k(G)$. □

3 Déminage (à rendre sur la copie 1)

Au beau milieu de la guerre, on soupçonne un batiment d'avoir été miné par l'ennemi. Le batiment en question est composé de pièces reliées par des couloirs, chaque couloir reliant exactement deux pièces.

On place un robot de déminage dans une des pièces. Le seul inconvénient, c'est qu'en ces temps de pénurie, le seul robot que l'on a pu trouver est un robot à deux bras, avec des pistolets à peinture au bout, un rouge et un bleu. Il a le droit de faire des marques sur les portes qu'il traverse. La reconnaissance des couleurs est un peu archaïque. Du coup, il risque de griller sur place s'il ne sait pas décider si une porte est plutôt avec plus de bleu ou plutôt avec plus de rouge. Et comme il vise très mal, on n'est pas certain que le robot réussira à bien recouvrir totalement une vieille peinture avec une nouvelle couleur et donc il vaut mieux ne l'autoriser à marquer qu'une seule fois chaque porte. Il ne peut marquer qu'un côté de chaque porte (celui qui donne sur la pièce), le côté couloir est trop sombre pour être lisible.

Le but est d'écrire un algorithme permettant au robot d'explorer toutes pièces et de s'arrêter lorsqu'il a la garantie qu'aucune bombe ne se trouve dans le batiment (ou bien qu'il a explosé sur une mine).

Solution : *On réalise un parcours (ni en profondeur ni en largeur). Dans l'algorithme général, les nœuds sont blancs (jamais visités), gris (en cours de visite) ou noir (visite terminée). Ici*

- *« jamais visité » sera représenté par « aucune porte marquée »,*
- *« en cours de visite » par « au moins une porte non marquée »,*
- *« visite terminée » par « Toutes les portes marquées ».*

À chaque instant, chaque pièce a au plus une porte marquée en rouge, cette dernière est celle qui mène au père du nœud. L'algorithme est alors le suivant :

- *En entrant dans un couloir, le traverser jusqu'à l'autre bout.*
- *En entrant dans un pièce où aucune porte n'est marquée, peindre en rouge la porte qu'il vient de franchir.*
- *Dans une pièce où il y a une porte sans marque, en choisir une, la peindre en bleu et la franchir.*
- *Dans une pièce où toutes les portes sont marquées, sortir par la porte rouge si elle existe, sinon s'arrêter (on a alors tout visité).*

□

4 Contrebande (à rendre sur la copie 2)

On souhaite envoyer discrètement des colis vers divers lieux. On dispose du plan des routes que l'on peut emprunter. Le coût d'un trajet est essentiellement le coût des pots-de-vin (positifs...) pour acheter le silence des postes de garde à chaque carrefour. On suppose que l'on connaît le prix f de chacun des carrefours. Le prix d'un trajet est donc la somme des prix des carrefours traversés.

Pour se ramener à un autre problème, on remplace les poids sur les carrefours par des poids sur les routes. Pour une route de u à v, on définit le coût de la route par $g(u,v) := \frac{f(u)+f(v)}{2}$.

Question 1 Montrer que les chemins optimaux selon f (i.e. de coût minimal selon f) entre deux points quelconques sont les mêmes que les chemins optimaux selon g.

Solution :

Soit $u_1, ..., u_k$ un chemin. $\sum_{i=1}^{k-1}(g(u_i, u_{i+1})) = \sum_{i=1}^{k} f(u_i) - \frac{f(u_1)+f(u_k)}{2}$. Le coût d'un chemin de u_1 à u_k est donc le même selon f et g à une constante près. Les chemins optimaux selon f sont donc les chemins optimaux selon g. □

Question 2 On utilise bien sûr l'algorithme de Dijkstra pour trouver le coût de toutes les destinations. On utilise la fonction de coût f (et non g). Montrer que dans ce cas particulier, pendant l'exécution de l'algorithme, le coût d'une destination n'est mis à jour qu'une seule fois.

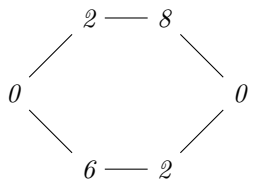
Solution :

Pour tout sommet v, soit $\hat{f}(v)$ le coût d'un chemin optimal vers v.

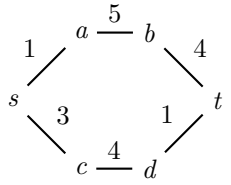
Le coût d'un sommet x est potentiellement mis à jour quand un de ses voisins sort du tas. Or un sommet u sort du tas avant v si et seulement si $\hat{f}(u) \leqslant \hat{f}(v)$. Le premier voisin u de x qui sort du tas entraîne une mise à jour de coût non définitif de x : il passe de ∞ à $\hat{f}(u) + f(x)$. Les voisins suivant v proposeront un chemin de coût $\hat{f}(v) + f(x)$ pour atteindre x, le coût de x ne sera donc pas mis à jour.

Remarque : c'est faux si l'on utilise la fonction de coût g au lieu de f, comme le montre l'exemple suivant :

fonction de coût f :



ce qui donne donne pour la fonction g :



et pour suite de mises à jour (une étape par ligne) :

sommet qui sort du tas	*coûts non définitifs*					
	s	a	b	c	d	t
	0	∞	∞	∞	∞	∞
s		*1*	∞	*3*	∞	∞
a			*6*	*3*	∞	∞
c			*6*		*7*	∞
b					*7*	*10*
d						*8*

Le coût du sommet t est donc mis à jour deux fois. □

Question 3 En déduire la nouvelle complexité de l'algorithme en précisant les structures de données utilisées.

Solution : *On utilise toujours un tas pour stocker les sommet ayant un coût provisoire, ce qui permet insertion, mise à jour et recherche du minimum en temps $O(\log n)$. Puisque le coût d'un sommet n'est mis à jour qu'une fois, la manipulation du tas ne coûtera au total plpus que $O(n \log n)$ (et non $O(m \log n)$). Le coût total est $O(m + n \log n)$.*

Remarques :
- *une fonction de poids sur les sommets est plus contrainte qu'une fonction de poids sur les arêtes.*
- *les (difficiles) tas de Fibonacci permettent une complexité $O(m + n \log n)$ pour l'algorithme de Dijkstra même dans le cas général des poids sur les sommets.*

□

5 Navigation **(à rendre sur la copie 2)**

Il n'est pas demandé de démonstration dans cette partie.

Un agent des services secrets se promène sur le graphe du web à la recherche d'informations.

Question 4 Dans les deux principaux navigateurs web, un clic du milieu sur un lien l'ouvre dans un nouvel onglet. Mais leurs comportements précis sont différents :
- Dans Firefox, l'onlet est placé après tous les onglets déjà ouverts.

– Dans Internet Explorer®, les onglets ouverts depuis une même page sont placés juste après l'onglet de cette page.

Notre agent clique (du milieu) sur tous les liens d'une page, et quand il a lu une page en entier il ferme l'onglet de cette page. Faire un lien avec le cours.

Solution : *Si l'on clique sur chaque lien rencontré, on fait un parcours en profondeur avec Internet Explorer® (la liste des onglets est LIFO) et un parcours en largeur avec Firefox (la file des onglets est FIFO).* □

Question 5 Sur certains sites web de webmestres incompétents, les liens déjà visités sont de la même couleur que les liens jamais visités. Faire à nouveau un lien avec le cours.

Solution : *Un parcours a besoin de se rappeler des nœuds déjà visités (sinon il parcourt indéfiniment un cycle). C'est le navigateur qui s'en charge (grâce à son historique) et affichant en bleu les liens vierges et en violet les liens déjà visités (ce sont les couleurs par défaut). Certains webmestres définissent tous les liens de la même couleur et demandent donc au navigateur d'effacer cette information. J'ai même vu un site avec les couleurs opposées des couleurs habituelles !*

(Tant que l'on garde en tête la facilité de navigation, assortir les jolies couleurs d'un site est utile.)

Remarques intéressantes trouvées dans les copies :
- *S'il y a une page d'un webmestre compétent dans chaque cycle, on est sauvé.*
- *Avec un parcours un largeur, même si on ne marque pas les sommets visités, on finira par visiter tout le graphe (mais on ne saura pas s'arrêter si on ne connaît pas la taille du graphe).*
- *Quand le webmestre est compétent et que le liens vers des sommets déjà visités sont marqués, on a supposé que, lors de la visite d'une page, tous les liens vers cette page changent de couleur dans tous les onglets.*
- *Certains ont précisé « on lit les onglets de gauche à droite » ou bien « et réciproquement pour un agent écrivant de droite à gauche ». Noter que le sujet ne dit pas « à droite » ou « à gauche » mais seulement « après ». Quelqu'un écrivant de droite à gauche interprétera naturellement « après » par « à gauche », ces précisions ne sont donc pas nécessaires.*

□

6 Pour bien finir **(à rendre sur la copie 2)**

Question 6 Quand on évalue la complexité d'algorithmes s'appliquant sur les graphes représentés par des listes d'adjacences, on considère souvent que ces listes sont triées. Si elles ne le sont pas, peut-on les trier rapidement ?

Solution : *On peut bien sûr trier chaque liste d'adjacence indépendamment, le tri de toutes les listes se fait alors globalement en* $O\big(\sum_{i=1}^{n} deg(i)\log(deg(i))\big) = O(m \log n)$. *Mais on peut faire mieux :*

- *On crée une seconde structure de graphe* G' *sans arêtes (listes vides).*
- *Pour chaque sommet* i *de* G *(dans l'ordre décroissant), on parcourt sa liste d'adjacence et pour chaque voisin* v *de* i *on ajoute* i *comme voisin de* v *dans* G' *(en tête de la liste d'adjacence de* v*).*
- *Les listes d'adjacence de* G' *sont alors triées (car on a ajouté les sommets en tête de liste en ordre décroissant).*
- *Les arêtes on été renversées (l'arête* (i, v) *est devenue* (v, i)*), donc si le graphe est orienté il suffit de le renverser une seconde fois.*

Au total, on a traité en temps constant chaque élément des listes d'adjacence, soit $O(n + m)$. □