TD #17 — Graphes #2

Exercice 1. *Détection de triangles de poids négatifs*

Un *triangle* d'un graphe orienté $G = S, A$ est tout triplet d'arcs $(u, v), (v, w), (w, u) \in A$. Dans le cas d'un graphe dont les arcs sont pondérés (par des poids $\in \mathbb{Z} \setminus \{0\}$), le *poids* d'un triangle est la somme des poids de ses arcs.

1. Écrivez une fonction C de signature :
   ```
   bool has_negative_triangle(size_t n, int g[n][n])
   ```
   qui décide si le graphe pondéré de `n` sommets représenté par la matrice d'adjacence `g` contient un triangle de poids négatif. (On suppose le graphe sans boucle : `g[i][i]` vaut zéro pour tout `i`.)
2. Analysez le coût pire-cas de votre fonction en fonction de `n`.
3. Montrez que l'existence d'un algorithme de détection de triangle de poids négatif dans un graphe de n sommets de coût $O(n^{2.99})$ (dans un modèle RAM transdichotomique) implique celle d'un algorithme calculant le produit de deux matrices carrées de dimension n dans le semi-anneau $(\min, +)$ en temps $O(n^{3-\delta})$ pour un certain $\delta > 0$.
4. Proposez un algorithme de détection de triangles de poids négatifs etc. de coût $O(n^{2.99})$.

Exercice 2. *Composantes fortement connexes*

Dans cet exercice, on considère un graphe orienté $G = S, A$ quelconque représenté par tableau de listes d'adjacence. Pour $v, w \in S$, on note $v \rightsquigarrow w$ la relation binaire indiquant qu'il existe un chemin de v à w dans G, et R la relation binaire définie par : $v R w$ ssi. $(v = w) \vee (v \rightsquigarrow w \wedge w \rightsquigarrow v)$. On rappelle que G est dit fortement connexe ssi. $\forall v, w \in S.\ v\ R\ w$.

On définit les *composantes fortement connexes* de G comme ses sous-graphes induits fortement connexes maximaux pour l'inclusion, ou alternativement comme les sous-graphes induits par les classes d'équivalences de R.

1. Montrez que R est une relation d'équivalence.
2. Montrez l'équivalence des deux caractérisations ci-dessus des composantes fortement connexes d'un graphe.
3. Soit $v \in S$, esquissez un algorithme pour calculer $S_v := \{w \in S \mid v \rightsquigarrow w\}$. Celui-ci devra être de coût linéaire en la taille de la représentation de G.
4. Soit $v \in S$ et S_v, esquissez un algorithme calculant l'ensemble des sommets de la composante fortement connexe de v. Celui-ci devra être de coût $O(\#S_v \times (\#A + \#S))$.

Pour améliorer le coût de l'algorithme impliqué par les deux questions précédentes, on introduit le *graphe transposé* $G^{\leftarrow}$ de G comme le graphe obtenu depuis G en inversant l'orientation de chaque arc $u \to v$ en $v \to u$.

5. Esquissez un algorithme de coût $O(\#S^2)$ (resp. $O(\#A + \#S)$) calculant une représentation de $G^{\leftarrow}$ depuis une représentation de G par matrice (resp. tableau de listes) d'adjacence.
6. Montrez que les composantes fortement connexes de G et $G^{\leftarrow}$ sont les mêmes.
7. Esquissez un algorithme qui calcule l'ensemble $S_v^{\leftarrow}$ des sommets de G **depuis** lesquels v est accessible (c'est à dire des sommets w t.q. $w \rightsquigarrow v$) en temps $O(\#A + \#S)$ (pour un graphe représenté par listes d'adjacence).
8. De tout ce qui précède, déduisez un algorithme qui calcule la partition des sommets de G correspondant à ses composantes fortement connexes en temps $O(\#S(\#A + \#S))$ (et en fait $O(\#S(\#A + 1))$).

L'algorithme précédent (et les résultats intermédiaires y ayant mené) peuvent se résumer de façon plus élégante (?) en utilisant la *fermeture transitive* G^C de G. Celle-ci est définie comme le graphe S, A^C t.q. $(v, w) \in A^C$ ssi. dans G l'on a $v = w \ \vee \ (v \rightsquigarrow w)$ Autrement dit, c'est le graphe représentant la fermeture réflexive-transitive de la relation $\rightsquigarrow$ (qui n'est autre que la relation R).
On admettra le résultat suivant (démontré après les vacances) : si G est représenté par matrice d'adjacence, la représentation par matrice d'adjacence de sa fermeture transitive peut être calculée en temps $O(\#S^3)$.

9. Esquissez un algorithme qui étant donnée une matrice d'adjacence représentant G^C calcule les composantes fortement connexes de G en temps $O(\#S^2)$.

Exercice 3. *Fermeture transitive d'un graphe non-orienté*

On considère à nouveau la *fermeture transitive* d'un graphe $G = S, A$ quelconque, définie à la fin de l'exercice précédent.
On s'intéresse ici uniquement au cas de graphes **non-orientés** représentés par tableau de listes d'adjacence.

1. Esquissez un algorithme qui calcule une « information équivalente » au graphe de la fermeture transitive G^C en temps linéaire en la taille de la représentation de G.
2. Pouvez-vous esquisser une modification de votre algorithme pour qu'il renvoie une représentation de G^C de type OCaml `int list array`, toujours en temps linéaire ?

Exercice 4. *Handshake lemma*

1. Montrez le *handshake lemma* : un graphe non orienté possède un nombre pair de sommets de degré impair.