Connexité et Cheminement

Gilles Simonin

8 octobre 2021

Résumé

Votre travail lors de cette séance de 2h30 est de tester vos connaissances théoriques sur les graphes, d'approfondir certaines notions, puis de juger votre capacité à concevoir un algorithme, démontrer certaine propriété relative à un algorithme, et enfin [**dans le cadre d'un projet noté**] mettre en œuvre un algorithme de graphe à partir de la représentation en machine qui été étudiée lors de la dernière séance.

1 Retour sur les définitions de base

Soit un graphe orienté $G = (X, A)$ d'ordre 10, dont les sommets sont numérotés de 1 à 10. On donne A sous forme d'une liste de couples : $A = \{(2,1),(3,2),(3,5),(4,1),(5,2),(5,4),(5,6),(6,8),(7,4),(8,5),(8,7),(8,9),(8,10),(9,6),(10,7),(10,9)\}$.

Question 1 *Donner une représentation graphique de G.*

Question 2 *Citer un chemin et une chaîne de longueur 5 de G. Faire de même pour un chemin élémentaire et une chaîne élémentaire de G. Puis donner tout d'abord un circuit et un cycle de G (n'importe quelle longueur), et ensuite une version élémentaire.*

Question 3 *Que valent $d^+(9)$ (degré sortant du sommet 9) et $d^-(10)$ (degré entant du sommet 10) ? G est-il symétrique ? Complet ?*

Question 4 *Le graphe G est-il fortement connexe ? Qu'en est-il de la version non orienté, est-il connexe ?*

Question 5 *Expliciter le sous-graphe engendré par le sous-ensemble de sommets $V' = \{4,7,8,10\}$. Donner également un graphe partiel sans cycle de G, puis un sous-graphe partiel engendré par le sous-ensemble de sommets $V'' = \{5,6,8,9,10\}$.*

2 Autour des chemins et des circuits

Reprenons le graphe G précédent.

Définition 1 *Un graphe non-orienté est eulérien (hamiltonien) s'il possède un cycle eulérien (hamiltonien). Il en est de même pour les graphes orientés avec les circuits.*

Question 6 *G admet-il un chemin hamiltonien ? Est-il hamiltonien ?*

Question 7 *G admet-il une chaîne eulérienne ? Est-il eulérien ?*

Question 8 *Montrer que tout graphe connexe G est Eulérien si et seulement si l'ensemble de ses arêtes peut être décomposé en cycles.*

Algorithm 1 Pseudocode d'un parcours en largeur d'abord

Require: Un graphe $G = (V, E)$, d'ordre n, un sommet $s \in V$.
Ensure: Un marquage des sommets de V dans l'ordre de parcours "largeur d'abord".

```
1: boolean mark[] = new boolean[n] ;
2: for all v ∈ V do
3:    mark[v] ← faux ;
4: end for
5: mark[s] ← vrai ;
6: Queue toVisit ; {premier ajouté, premier sorti}
7: toVisit.push(s) ;
8: while !toVisit.isEmpty() do
9:    int v ← toVisit.pop() ;
10:   for all w tel que (v, w) ∈ E do
11:      if mark[w] == faux then
12:         mark[w] ← vrai ;
13:         toVisit.push(w) ;
14:      end if
15:   end for
16: end while
```

3 Parcours d'un graphe en largeur

L'algorithme 1 est un implémentation non récursive d'une exploration en largeur d'un graphe $G = (V, E)$.

Question 9 *Prenez un exemple de graphe non orienté et donnez une trace de l'exécution de cet algorithme. Pourquoi parle-t-on de parcours en largeur ? Quel est le type de graphe partiel induit par ce parcours ?*

Question 10 *Comment modifier l'algorithme d'exploration en largeur proposé ci-dessous de façon à calculer les plus courts chemins en nombre d'arêtes du sommet de départ de l'exploration à tous les autres sommets ?*

Question 11 *Quelle est la complexité de l'algorithme initial ? de sa version modifiée ?*

4 Une version impérative d'un parcours en profondeur

Considérons maintenant l'algorithme `explorerGraphe`() introduit dans le cours au transparent 51 du cours Partie 1.

Question 12 *Pouvez-vous en proposer une version impérative (c'est à dire non récursive) en partant de l'algorithme proposé dans la section 3 ? Quelle est la complexité ?*

Question 13 *Quel est le type de graphe partiel induit par ce parcours ?*

5 Extension au cas des composantes fortement connexes

Un graphe orienté $G = (V, A)$ est dit *fortement connexe* si, étant donné deux sommets x et y de V, il existe un chemin de x vers y ET il existe un chemin de y vers x dans G. La relation ainsi définie entre x et y forme une relation d'équivalence. Les classes d'équivalences induites par cette relation sur V forment une partition de V appelée *composantes fortement connexes*. Nous nous proposons ici d'étudier une méthode efficace pour calculer les composantes fortement connexes d'un graphe orienté $G = (V, A)$. Pour cela, nous repartirons de l'algorithme `explorerGraphe`() proposé au transparent 53 du cours.

Question 14 *Montrez que le graphe partiel induit par les arcs parcourus lors d'un appel à l'algorithme* `explorerGraphe()` *est une forêt couvrante.*

Nous souhaitons modifier l'algorithme afin de pouvoir distinguer trois classes de sommets : les non visités, les sommets en cours de visite et les sommets totalement visités. Un sommet s est *non visité* s'il n'a pas encore été examiné par la procédure `explorerSommet`(s) ; un sommet s est dit *en cours de visite* s'il a déjà fait l'objet d'un appel à la procédure `explorerSommet`(s) mais que cet appel ne s'est pas encore terminé ; enfin, un sommet s est dit *totalement visité* lorsque l'appel à `explorerSommet`(s) est terminé (c'est à dire que tous les successeurs de s ont été visités).

Question 15 *Proposez une modification de l'algorithme intégrant cette classification dynamique à partir d'un tableau d'entier* `visite`[]*, pour lequel* $\forall x \in V$, `visite`$[x] \in \{0, 1, 2\}$ *suivant l'état de* x.

Nous souhaitons maintenant être en mesure de *dater* les évènements subis par chaque sommet pour cet algorithme. Cela va permettre de créer un lien de *filiation* entre les sommets du graphe. En pratique, pour sommet $x \in V$, nous voulons un connaitre le moment où le sommet a été rencontré pour la première fois (noté `debut`$[x]$), et le moment où tous les successeurs de x ont été visités (noté `fin`$[x]$). Ainsi, on dira que y est un descendant de x dans notre parcours si et seulement si on rencontre y, et termine sa visite, avant d'avoir terminé de visiter x.

Question 16 *À partir d'une variable globale incrémentée judicieusement et deux tableaux d'entiers* `debut`[] *et* `fin`[]*, proposez une modification de l'algorithme.*

Question 17 *Quelle est la complexité de l'algorithme ainsi modifié ?*

Question 18 *Que peut-on dire de* `debut`$[x]$, `fin`$[x]$, `debut`$[y]$ *et* `fin`$[y]$ *pour deux sommets* x *et* y *du graphe* G *? Vous supposerez dans votre observation que* `debut`$[x] <$ `debut`$[y]$ *(sinon inversez les notations), remarquez que deux cas peuvent se produire :* y *est un descendant de* x *ou non.*

Question 19 *Étant donné deux sommets* x *et* y *d'un graphe orienté* $G = (V, A)$*, montrez que* y *est un descendant de* x *(il existe un chemin de* x *à* y *dans le parcours) si et seulement si, à l'instant* `debut`$[x]$*, il existe un chemin de* x *à* y *n'empruntant que des sommets* z *tels que* `visite`$[z] = 0$.

6 Mise en œuvre - Contrôle continue partie 1 - Travail en Duo

Reprenez votre implémentation de la structure de graphe proposée lors de la précédente séance. Un contrôle continue partie 2 arrivera en TP 4 sur les plus courts chemins. Vous rendrez votre travail final (parties 1 et 2) avant le 30 novembre au soir dans un dépôt sur Moodle.

Question 20 *Proposez une mise en œuvre de l'algorithme de parcours en profondeur d'abord.*

Question 21 *Proposez une mise en œuvre de l'algorithme de parcours en largeur d'abord.*

Question 22 *Proposez une mise en œuvre algorithmique du calcul des composantes fortement connexe d'un graphe* $G = (V, A)$*, à partir des éléments introduits à la section 5 :*

1. *Exécuter l'algorithme* `explorerGraphe()` *sur le graphe* G *et mémoriser* `fin`[].
2. *Inverser le graphe* G *par la procédure introduite durant le cours (transparent 32). Soit* G^{-1} *le graphe inverse de* G.
3. *Exécuter l'algorithme* `explorerGraphe()` *sur le graphe* G^{-1} *en appelant dans la boucle principale les sommets de* G^{-1} *par ordre décroissant sur* `fin`[].

Question 23 *Comparez les performances de ces algorithmes suivant la représentation utilisée pour la structure de graphe.*