CentraleSupélec

Algorithmique et complexité

TD 1/7 – Parcours de graphes

Exercices complémentaires

Remarque: Les éléments de correction fournis dans le sujet ne sont pas forcément complets. Ils sont là pour vous guider dans votre travail personnel. Nous vous invitons à rédiger une solution comme vous le feriez à l'examen et, si vous avez des questions, à vous tourner vers votre chargé de TD.

Exercice 1 : Par ici la sortie !

Ce premier exercice, relativement facile, doit pouvoir être réalisé par tous les étudiants en autonomie, comme à l'examen.

Dans le cours 1, nous avons vu les algorithmes de parcours en profondeur et en largeur pour décider si un sommet est atteignable. Ces algorithmes ne peuvent pas être utilisés directement pour trouver le chemin de sortie d'un labyrinthe. L'objectif de cet exercice est de palier ce manque.

Question 1

Modifiez l'algorithme BFS vu au cours 1 pour qu'il mémorise dans un dictionnaire python le somment antécédent (n dans l'algorithme) chaque fois qu'un voisin (v) est ajouté dans la file. L'algorithme doit retourner ce dictionnaire au lieu d'une valeur booléenne.

Élements de correction :

Il faut penser à créer un dictionnaire `parent` initialement vide et ajouter une instruction lors de l'ajout d'un voisin, si celui-ci n'est pas déjà présent dans le dictionnaire, pour lui associer le sommet antécédent.

Écrivez complètement l'algorithme en Python, ne vous contentez pas d'un "ah oui, j'ai compris" !

Question 2

Écrivez maintenant un deuxième algorithme qui calcule le chemin entre s et t en utilisant la structure de données obtenue par l'algorithme précédent. Le résultat renvoyé est une liste de sommets de V dont le premier élément est s et le dernier t.

Élements de correction :

En partant du sommet t, il faut écrire une boucle qui se termine lorsqu'on atteint s et qui regarde dans le dictionnaire des parents pour « remonter » dans le graphe.

Question 3

Quelle est la complexité de votre algorithme de recherche de chemin ?

Élements de correction :

Si vous n'avez pas fait de bêtise, vous devriez toujours être en $\mathcal{O}(|E|)$. Si vous n'êtes pas sûr de votre réponse, nous vous invitons à en discuter avec votre chargé(e) de TD.

Exercice 2 : Ordre topologique

Un processus de fabrication, un projet, une scolarité, etc. sont composés de tâches (modules, etc.). Certaines tâches peuvent être réalisées uniquement après l'exécution de certaines autres tâches. Nous voulons établir l'ordre d'exécution de toutes les tâches en respectant ces contraintes.

Un ordre topologique d'un graphe orienté $G = (V, E)$ est un ordre de sommets de V (une liste) $v_1, v_2, \ldots, v_n$ tel que pour tout arc (v_i, v_j) nous avons $i < j$.

Question 1

Proposez un algorithme qui ordonne topologiquement un DAG.

Élements de correction :

Il existe plusieurs algorithmes célèbres pour résoudre ce problème dont l'algorithme de Kahn (1962) et l'algorithme de Tarjan (1979). D'autres algorithmes ont été proposés dans les années 80 pour utiliser le calcul parallèle. Vous pouvez trouver le détail de ces algorithmes sur la page wikipedia :

`https://en.wikipedia.org/wiki/Topological_sorting`

L'algorithme de Tarjan est basé sur le parcours en profondeur en utilisant une pile pour garder l'ordre. En effet, si G est un DAG, il suffit alors d'effectuer un parcours en profondeur du graphe, au cours duquel on empile chaque sommet une fois ses successeurs visités (c-à-d après tous les appels récursifs portant sur ces successeurs). En désempilant, on obtient un tri topologique.

L'algorithme de Kahn est plus immédiat : un sommet sans arcs entrants (il en existe forcément un dans un DAG) reçoit le numéro 1. On le retire (avec ses arcs sortants), ce qui reste est toujours un DAG. Et on recommence.

Vous devriez essayer d'écrire vous-même ces algorithmes en Python, sans regarder la solution.

Question 2

Prouvez le théorème suivant :

$$G \text{ admet un ordre topologique} \iff G \text{ est un DAG.}$$

Élements de correction :

Dans un sens, vous pouvez utiliser une preuve par l'absurde en observant le sommet qui a le plus petit rang dans le cycle et son prédécesseur, puis dans l'autre sens, une preuve par récurrence en reprenant l'approche constructive de l'algorithme de Kahn (l'hypothèse de récurrence est que tout DAG à n sommets a un ordre topologique).

Nous vous invitons à écrire proprement ces preuves pour vous entraîner à l'examen.