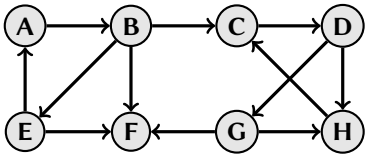
Outils logiques et algorithmiques – TD 6 – Terminaison

Exercise 1 (Parcours en profondeur et détection de cycles) Voici un algorithme de parcours en profondeur pour des graphes orientés. L'algorithme manipule une pile avec les notations suivantes : `[]` désigne la pile vide, `p.empiler(s)` place l'élément `s` au sommet de la pile `p`, et `p.depiler()` retire l'élément au sommet de la pile `p`. Notez que dans sa forme présentée ici l'algorithme ne consulte jamais la pile : il se contente d'y enregistrer certaines informations.

```
 1   exploration(s):
 2       en_cours := []
 3       visiter(s)

 4   visiter(s):
 5       si s est déjà marqué:
 6           ne rien faire
 7       sinon:
 8           en_cours.empiler(s)
 9           marquer s
10           pour chaque successeur s' de s:
11               visiter(s')
12           en_cours.depiler()
```

1. Exécuter l'algorithme `exploration` sur le sommet A du graphe suivant. On pourra supposer que toute énumération de sommets est faite par ordre alphabétique.



2. Décrire l'effet de `visiter(s)` sur la pile `en_cours`, et notamment la relation entre l'état de la pile en entrée et l'état de la pile en sortie. Justifier que cette spécification est correcte et dire quel est le sommet retiré du sommet de la pile par l'instruction (ligne 12)

   ```
   en_cours.depiler()
   ```

3. Montrer qu'à tout moment de l'exécution de `exploration(s)`, les sommets contenus dans la pile `en_cours` décrivent un chemin dans le graphe exploré.
4. On propose la technique suivante pour tester la présence d'un cycle :

   > *Si au cours de la visite d'un sommet* `s` *le test de la ligne 5 s'évalue à vrai, alors on déclare que le graphe contient un cycle passant par* `s`*. Si en revanche le parcours se termine sans que ce cas se soit produit alors on déclare que la graphe ne contient pas de cycle.*

   Montrer que ce critère est invalide et le corriger.

□

Exercise 2 (Terminaison de parcours de graphe) Voici un algorithme de parcours de graphe.

```
explorer(G, s)
    à_explorer = {s}
    tant que à_explorer n'est pas vide
        retirer un sommet s de à_explorer
        ...
        pour chaque voisin v de s
            si v n'est pas marqué alors
                marquer v
                ajouter v à à_explorer
```

1. Lors d'un tour de la boucle `tant que`, comment évoluent le nombre de sommets dans l'ensemble `à_explorer` et le nombre de sommets marqués ?
2. Justifier que l'exploration termine pour tout graphe `G` fini.

□

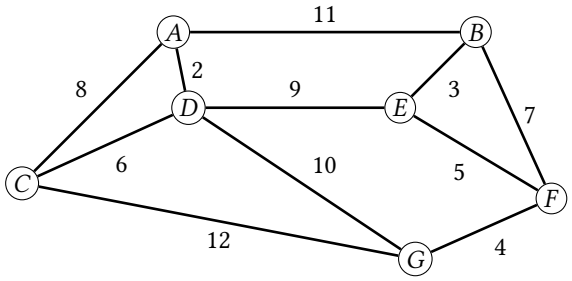
Exercise 3 (Tournée générale !) Dans cet exercice, on regarde des graphes non orientés dont les arêtes ont des longueurs (strictement positives). Un graphe est *connexe* si pour tous deux sommets s_1 et s_2 il existe un chemin entre s_1 et s_2. Une *couverture connexe* d'un graphe $G = (S, A)$ est un graphe connexe $G' = (S', A')$ avec $S = S'$ et $A' \subseteq A$. Une *couverture connexe minimale* est une couverture connexe pour laquelle la somme des longueurs des arêtes est minimale.

1. Montrer qu'un graphe admettant une couverture connexe est lui-même connexe.
2. Montrer qu'une couverture connexe minimale n'a pas de cycle.
3. Donner un graphe pour lequel il existe plusieurs couvertures connexes minimales.
4. Considérons un graphe G, un cycle ρ dans G, et une arête a de ce cycle dont la longueur est strictement supérieure aux longueurs des autres arêtes de ρ. Montrer qu'une couverture connexe de G contenant a ne peut pas être minimale.
5. En déduire que si toutes les arêtes ont des longueurs différentes, alors la couverture minimale connexe est unique.

L'algorithme de Kruskal construit une couverture connexe minimale :

```
G' = (S, ensemble vide)
trier A par ordre de longueurs croissantes
pour chaque a dans A
    si a relie deux sommets qui ne sont pas joignables par un chemin dans G'
    alors ajouter a à G'
renvoyer G'
```

6. Appliquer l'algorithme au graphe suivant.



7. Quels éléments définissent la complexité en temps de cet algorithme ?

Considérons maintenant un ensemble E de points du plan à deux dimensions. Une *tournée* de E est un itinéraire partant de l'un des points de E et y revenant après être passé par chacun des autres points au moins une fois. Trouver une tournée de longueur minimale est un problème algorithmiquement difficile [1]. On va montrer ici que l'on peut en revanche trouver efficacement une tournée dont la longueur ne dépasse pas le double de la longueur minimale.

8. Proposer une manière d'associer un graphe à l'ensemble E, et de traduire notre objectif dans le vocabulaire des graphes.
9. Montrer que toute tournée a une longueur strictement plus grande que la somme des longueurs des arêtes d'une couverture connexe minimale.
10. Montrer comment, à partir d'une couverture connexe minimale, on peut construire une tournée dont la longueur est deux fois la somme des longueurs des arêtes de cette couverture.

□

1. Le problème est *NP-complet*, et on ne connaît que des algorithmes dont la complexité est exponentielle en le nombre de points.