TD RO : Théorie des Graphes

Éléments de correction

Exercice 1

1. **Énoncé:** Soit G un graphe connexe. Montrer que si x est un sommet de degré 1, alors $G \setminus \{x\}$ est encore un graphe connexe.

   Démonstration: Soit x un sommet de degré 1 dans un graphe connexe G. Soit y l'unique voisin de x. Si on supprime x, aucun chemin entre deux sommets distincts de y ne peut passer par x car x est une extrémité. Donc, la connexité entre les sommets de $G \setminus \{x\}$ est préservée. $G \setminus \{x\}$ est donc connexe.

2. **Énoncé:** Montrer que si G est connexe d'ordre $n \geq 2$, alors il doit au moins avoir $n-1$ arêtes. Qu'en est-il de la réciproque?

   Démonstration (par récurrence):
   Cas de base: Pour $n = 2$, un graphe connexe a au moins une arête.
   Hérédité: Supposons qu'un graphe connexe d'ordre k a au moins $k-1$ arêtes. Considérons un graphe connexe G d'ordre $k+1$. Il existe un sommet v tel que $G \setminus \{v\}$ est connexe (par exemple, une feuille d'un arbre couvrant). $G \setminus \{v\}$ a au moins $k-1$ arêtes. Puisque G est connexe, v doit être connecté à au moins un sommet de $G \setminus \{v\}$, donc G a au moins k arêtes.

   Réciproque: La réciproque est fausse. Un graphe avec $n-1$ arêtes n'est pas forcément connexe (exemple : deux composantes connexes, une de taille $n-1$ avec $n-2$ arêtes et un sommet isolé).

3. **Énoncé:** Soit un graphe G d'ordre n et ayant strictement plus de $(n-1)(n-2)/2$ arêtes. Montrez que ce graphe est connexe.

   Démonstration (par l'absurde):
   Supposons que G est non connexe. Alors G a au moins deux composantes connexes. Le nombre maximal d'arêtes pour un graphe non connexe d'ordre n est atteint quand il y a une composante connexe de taille $n-1$ et un sommet isolé, soit $(n-1)(n-2)/2$ arêtes. Si G a plus d'arêtes, il doit être connexe.

4. **Énoncé:** Montrer que dans un graphe non orienté la somme des degrés des sommets est un nombre pair. En déduire que tout graphe possède un nombre pair de sommets de degré impair.

   Démonstration: Le lemme des poignées de main stipule que la somme des degrés des sommets est égale à deux fois le nombre d'arêtes, donc un nombre pair. Par conséquent, le nombre de sommets de degré impair doit être pair pour que la somme soit paire.

5. **Énoncé:** Proposer un algorithme qui vérifie si un graphe est connexe (supposons que BFS existe).

   Algorithme:

   (a) Appliquer BFS à partir d'un sommet arbitraire s.

   (b) Si tous les sommets sont visités, le graphe est connexe. Sinon, il est non connexe.

   Justification : BFS explore tous les sommets accessibles depuis un sommet de départ. Si tous les sommets sont visités, alors il existe un chemin entre n'importe quelle paire de sommets, et donc le graphe est connexe.

Exercice 2

1. **Formalisation:**

   On peut modéliser ce problème avec un graphe non orienté $G = (V, E)$, où :

   - V est l'ensemble des modules ($|V| = N$).
   - Une arête $\{u, v\} \in E$ existe si et seulement si il existe au moins un étudiant inscrit aux modules u et v (ces examens ne peuvent être simultanés).

   Le problème revient alors à colorier les sommets de G avec un nombre minimal de couleurs, chaque couleur représentant un jour d'examen. Les sommets adjacents (reliés par une arête) doivent avoir des couleurs différentes. Ce problème est connu comme le problème de coloration de graphe.

Solution

On utilise la fonction de coloration séquentielle donnée en cours :

```
fonction coloration-séquentielle (G graphe) : entier ;
début
  pour i ← 1 à n faire {
    c ← 1 ;
    tant que il existe t adjacent à s_i avec f(t) = c faire
      c ← c + 1 ;
    f(s_i) ← c ;
  }
  retour max (f(s_i), i = 1, ..., n) ;
fin
```

Où :

- G est le graphe représentant les conflits entre examens.
- n est le nombre de sommets (modules).
- s_i est le i-ème sommet du graphe.
- $f(s_i)$ est la couleur (jour) assignée au sommet s_i.

Contrainte du nombre de salles (K)

Pour gérer la contrainte de salles, on modifie l'algorithme :

```
fonction coloration-séquentielle (G graphe, K constante) : entier ;
début
  NB[] = {0}
  pour i ← 1 à n faire {
    c ← 1 ;
    tant que il existe t adjacent à s_i avec f(t) = c OU NB[c] == K faire {
        c ← c + 1 ;
        f(s_i) ← c ;
        NB[c] ← NB[c] + 1 ;
    }

  }
  retour max (f(s_i), i = 1, ..., n) ;
fin
```