Planche 5

MP2I – l'EISAM

www.crvr.fr

Introduction

Cette planche de colle porte sur les bases des graphes.

Les exercices sont de difficulté variable et ont pour objectif de vérifier votre compréhension et votre maîtrise de ces notions.

Exercices

Cours

Exercice

Déclarer un type graphe dans le langage de votre choix. Déclarer une variable graphe qui correspond au graphe suivant :

- Paris est connectée à Lyon.
- Lyon est connectée à Marseille.
- Marseille est connectée à Montpellier.
- Montpellier est connectée à Paris.

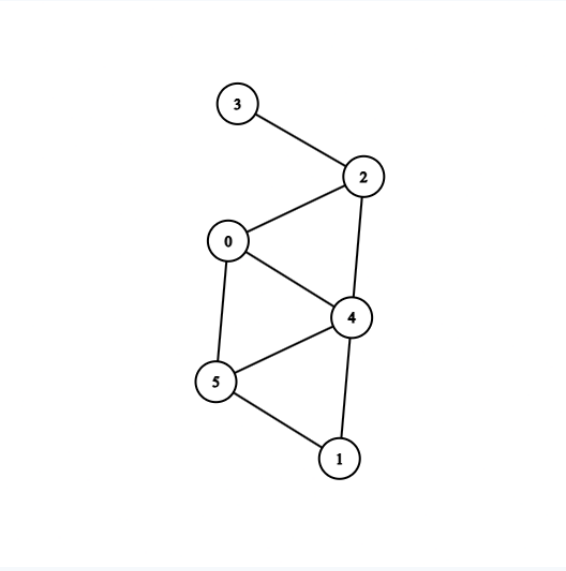
Exercice

Un graphe peut être codé soit par liste d'adjacence, soit par matrice d'adjacence. Quelle type optimise :

- L'espace mémoire.
- Le temps pour trouver le nombre de voisins d'un sommet.
- Le temps pour trouver la liste des voisins d'un sommet.

Exercice

Soit le graphe suivant :



Est-il cyclique ? Quel est le plus corut chemin de 3 à 1 ? Est-il unique ?

Donner la plus grande composante connexe possible du graphe.

Exercices de colle

Exercice

On propose le type graphe non-orienté suivant (par convention, les arêts sont encodés uniquement sur les parties $i < j$ de la matrice, pour tester si il y a une arête (i, j), il faut tester si il y a $(min(i, j), max(i, j))$ dans le graphe) :

```
type graphe = bool array array
```

1. Problème : le type actuel n'encode pas la taille du graphe. Proposer un nouveau type `graphe_bis` qui contient le champ `taille` qui donne le nombre de sommets et `adj` qui donne la matrice d'adjacence.
2. Créer une variable qui contient un graphe non vide.
3. Ecrire une fonction `existe_cycle: graphe_bis -> int -> int -> bool` qui dit si un graphe contient un cycle qui passe par i et j.

Exercice

On propose le type graphe orienté suivant :

```
type graphe = bool array array
```

1. Problème : le type actuel n'encode pas la taille du graphe. Proposer un nouveau type `graphe_bis` qui contient le champ `taille` qui donne le nombre de sommets et `adj` qui donne la matrice d'adjacence.
2. Créer une variable qui contient un graphe non vide.
3. Ecrire une fonction `existe_chemin: graphe_bis -> int -> int -> bool` qui dit si un graphe contient un chemin qui passe par de i à j.

Exercice

On propose le type graphe orienté suivant :

```
type graphe = bool array array
```

1. Problème : le type actuel n'encode pas la taille du graphe. Proposer un nouveau type `graphe_bis` qui contient le champ `taille` qui donne le nombre de sommets et `adj` qui donne la matrice d'adjacence.
2. Créer une variable qui contient un graphe non vide. On dit qu'un graphe G_1 est inclus dans G_2 si $|G_1| \leq |G_2|$ et $\forall (i, j) \in G_1, (i, j) \in G_2$.
3. Ecrire une fonction `inclus: graphe_bis -> graphe_bis -> bool` qui renvoie `true` si et seulement si le premier graphe est inclus dans le deuxième.

Exercice

Source : TP que j'ai donné à Saint-Louis (Paris) en 2024.

Définition 3.1 niveau de connexité

Soit G un graphe non-orienté, on note $\xi(G)$ son niveau de connexité, à savoir le nombre minimal d'arêtes qu'il faut lui retirer pour qu'il ne soit plus connexe.

Question 8 à l'écrit

Indiquer $\xi(G)$ pour chacun des graphes de la question 2. Que dire de $\xi(G)$ pour un graphe non-connexe ?

On a la borne triviale: $\xi(G) \leq n$ avec n le nombre de sommets.

Question 9 à l'écrit

En sachant qu'on doit retirer au plus $|\mathcal{V}|$ arêtes de G pour trouver $\xi(G)$, combien (au $+$) de combinaisons d'arêtes à retirer allez-vous tester ?

Question 10 code

Déduire de la borne un algorithme `int xi(graphe g)` qui renvoie $\xi(G)$. Quelle est sa complexité ?

Exercice

Nous considérons maintenant des graphes pondérés, où chaque arête a un poids (coût) positif ou nul, représenté par un flottant. Une représentation possible est une liste d'adjacence où chaque voisin est une paire `(sommet, poids)`.

```
type poids = float
type sommet = int
type arete_ponderee = sommet * poids
type graphe_pondere = arete_ponderee list array
```

L'objectif est d'implémenter l'algorithme de Dijkstra pour trouver les plus courts chemins depuis une source donnée.

1. Donnez la représentation `graphe_pondere` pour le graphe suivant :
   — Sommets : 0, 1, 2, 3
   — Arêtes : (0, 1, poids=1.0), (0, 2, poids=4.0), (1, 2, poids=2.0), (1, 3, poids=5.0), (2, 3, poids=1.0)
   L'algorithme de Dijkstra utilise une file de priorité pour sélectionner le sommet non visité le plus proche de la source.
2. Sans l'implémenter, donnez les signatures OCaml que vous attendriez pour les opérations suivantes sur une file de priorité `PrioQueue` contenant des paires `(sommet, distance)` (priorité sur la distance) :
   — Créer une file vide.
   — Ajouter/mettre à jour un élément avec sa priorité.
   — Extraire l'élément de plus petite priorité.

— Vérifier si la file est vide.
Coder une file de priorité prend du temps, supposez que vous avez la structure dans la suite.

3. Ecrivez la fonction OCaml `dijkstra : graphe_pondere -> sommet -> (float array * sommet array)`. Cette fonction prend le graphe et le sommet source `s`, et retourne une paire de tableaux :
   — `distances : distances.(i)` contient la longueur du plus court chemin de s à i. Initialiser à ∞ (utiliser 'infinity'), sauf pour la source (0.0).
   — `predecesseurs : predecesseurs.(i)` contient le sommet précédant i dans le plus court chemin depuis s. Initialiser -1.

Barême

Les exercices sont de difficultés variables. Je me réserve le droit de fixer la note en fonction de votre performance sans barême fixe (car je ne connais pas encore votre niveau sur les graphes).