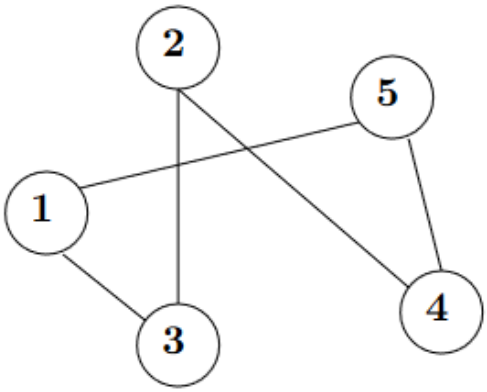
Graphes et algorithmes : TD2

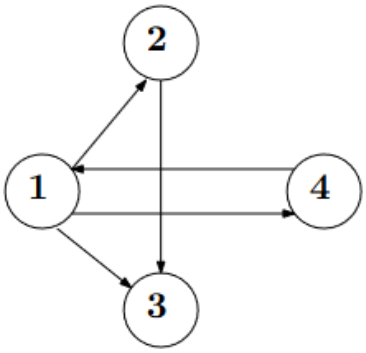
Exercice 1

Sujet

Donner les représentations par matrice d'incidence, matrice d'adjacence et listes d'adjacence des deux graphes suivants, puis déterminer le degré de chaque sommet.



1



2

Résolution

Graphe 1 :

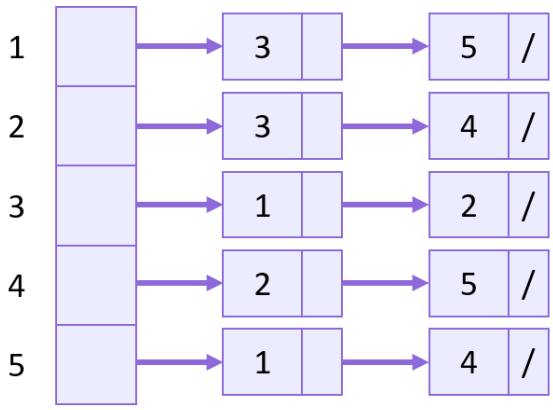
- **Matrice d'incidence** :

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} (1,3) & (1,5) & (2,3) & (2,4) & (4,5) \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

- **Matrice d'adjacence** :

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

- **Listes d'adjacence** :



Graphe 2 :

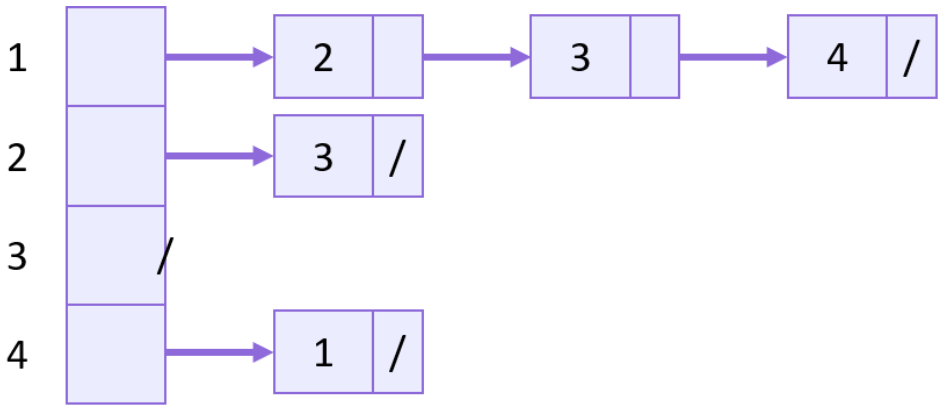
- **Matrice d'incidence** :

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} (1,2) & (1,3) & (1,4) & (2,3) & (4,1) \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & -1 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{pmatrix} \end{array}$$

- **Matrice d'adjacence** :

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

- **Listes d'adjacence** :



Exercice 2

Sujet

Donner des algorithmes (en pseudo-langage) pour insérer et supprimer un arc dans un graphe orienté représenté par sa matrice d'adjacence, puis par listes d'adjacences.

Résolution

Matrice d'adjacence

```
CONST MAX = 1000;
TYPE Matrice = tableau[MAX][MAX] d'Entiers;

/*INSERTION*/

/*retourne 0 si (i,j) existe déjà, 1 sinon*/
fonction ajout_matrice(Entrée/Sortie M : Matrice, Entrée i,j : Entier) : Ent
début
    si M[i][j] = 0 alors
        M[i][j] ← 1;
        retourner 1;
    fin si
    retourner 0;
fin

/*SUPPRESSION*/

/*retourne 0 si (i,j) n'existe pas, 1 sinon*/
fonction supp_matrice(Entrée/Sortie M : Matrice, Entrée i,j : Entier) : Enti
début
    si M[i][j] = 1 alors
        M[i][j] ← 0;
        retourner 1;
    fin si
    retourner 0;
fin
```

Listes d'adjacence

```
CONST MAX = 1000;
TYPE Listes = tableau[MAX] de Liste;

/*INSERTION*/

/*retourne 0 si (i,j) existe déjà, 1 sinon*/
fonction ajout_liste(Entrée/Sortie L : Listes, Entrée i,j : Entier) : Entie
début
    si j ∉ L[i] alors
        L[i] ← L[i] + {j};
        retourner 1;
    fin si
```

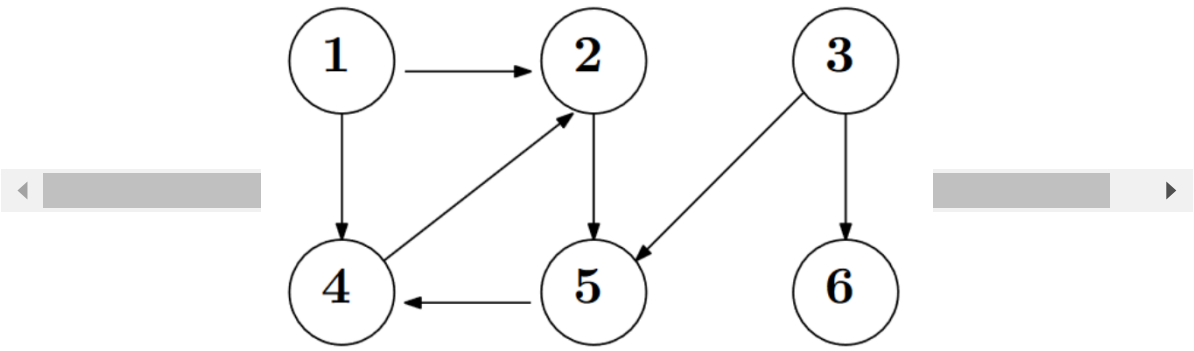
```
    retourner 0;
fin

/*SUPPRESSION*/

/*retourne 0 si (i,j) n'existe pas, 1 sinon*/
fonction supp_liste(Entrée/Sortie L : Listes, Entrée i,j : Entier) : Entier
début
    si j ∈ L[i] alors
        L[i] ← L[i] - {j};
        retourner 1;
    fin si
    retourner 0;
fin
```

Exercice 3

Sujet

Le carré d'un graphe orienté $G = (S, A)$ est le graphe $G^2 = (S, A^2)$ tel que $(x, z) \in A2 \Leftrightarrow \exists y \in S$ tel que $(x, y) \in A$ et $(y, z) \in A$.

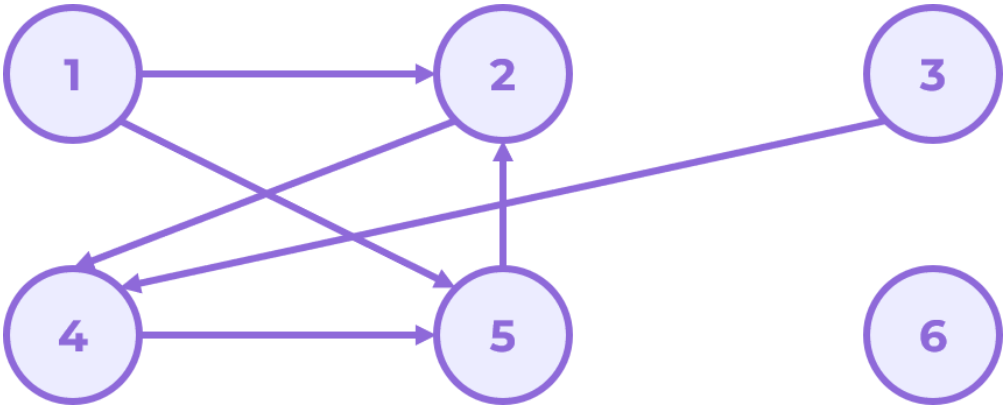
1. Donner le carré du graphe orienté ci-dessous :



2. Donner une interprétation de G^2 en termes de chemin.
3. Décrire un algorithme (pseudo-langage) efficace permettant de calculer le graphe G^2 d'un graphe G représenté par listes d'adjacence.
4. Même question si G est représenté par matrice d'adjacence.
5. Analyser le temps d'exécution des algorithmes décrits dans les questions précédentes.

Résolution

Question 1



Question 2

Pour que l'arc (x, z) existe dans G^2, il faut qu'il existe au moins un chemin de longueur 2 dans G reliant x à z.

Question 3

```
/*retourne le carré du graphe représenté par la liste
d'adjacence L*/
fonction carre_listes(Entrée L : Listes, n : Entier) : Listes
début
    pour i allant de 0 à n faire
        L²[i] ← ∅;
    fin pour

    pour i allant de 0 à n faire
        pour j ∈ L[i] faire
            pour k ∈ L[j] faire
                ajout_liste(L², i, k);
            fin pour
        fin pour
    fin pour
    retourner L²;
fin
```

> Complexité : $O(n^3)$