C.U.F.R. J-F. Champollion
Albi

L2 Informatique S5
2015

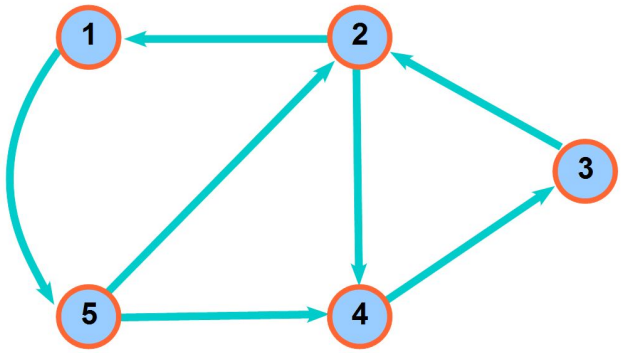
Théorie et Algorithmique des Graphes

TP (Java – Trois séances)
Représentations d'un graphe. Premières méthodes.

Le but de ce premier TP est de mettre en œuvre en Java la représentation d'un graphe par matrice ou par listes d'adjacence et de commencer à utiliser cette représentation pour écrire nos premières fonctions simples de traitement des graphes.

1 Graphes orientés

Les fonctions suivantes seront testées sur le graphe G1 (page 8 du polycopié de cours)



On rappelle que ce graphe peut être représenté :

- par liste d'arêtes par
  L=[[1,5],[2,1],[2,4],[3,2],[4,3],[5,2],[5,4]]
- par liste d'adjacence par
  G=[[],[5],[1,4],[2],[3],[2,4]]
- et par matrice d'adjacence par

```
M=[[0, 0, 0, 0, 0, 0],
   [0, 0, 0, 0, 0, 1],
   [0, 1, 0, 0, 1, 0],
   [0, 0, 1, 0, 0, 0],
   [0, 0, 0, 1, 0, 0],
   [0, 0, 1, 0, 1, 0]]
```

Remarque : Les éléments d'un objet de type ArrayList ou d'un tableau d'entiers sont indexées à partir de 0 en Java. Or nous souhaitons, pour faciliter la compréhension des algorithmes, que G[i] soit la liste d'adjacence du sommet i, on prendra donc l'habitude de commencer notre vecteur de listes d'adjacence par une première liste vide (indexée 0 et qui pourra éventuellement servir à mémoriser des résultats de calculs auxiliaires.)

Pour les mêmes raisons, la matrice d'adjacence sera indexée de 0 à n, la colonne et la ligne indexées 0 étant perdues et l'élément M[i][j] concernant bien l'arête entre les sommets i et j.

1 Compléter la classe GrapheOrienteMat

- Attributs :
  private int n; // le nombre de sommets
  private int m; // Le nombre d'arcs
  private int[][] mat; // La matrice d'adjacence
- Méthodes :
  - Constructeur du graphe (par défaut, construction d'un graphe sans arc à n sommets)
  - Constructeur du graphe (à partir des listes d'adjacence, sous forme d'un tableau à deux dimension{{succ de 1},{succ de 2},..,{succ de n}})

On testera les constructeurs et les différentes méthodes sur le graphe G1
Compléter les méthodes suivantes :

- *public int nbSommets() :* qui retourne le nombre de sommets du graphe
- *public int nbArcs() :* qui retourne le nombre d'arcs du graphe
- *public void affiche()* : qui affiche la matrice représentant le graphe
- *ajoutArc(G,i,j):* qui ajoute l'arc (i,j) au graphe G
- *enleveArc(G,i,j):* qui enlève l'arc (i,j) du graphe G (si elle existe)
- *public int degreS(int i) :* qui calcule le degré sortant du sommet i
- *public int[] degreS() :* qui retourne un vecteur D tel que D[i-1] soit le degré sortant du sommet i.
- *public int degreE(int i) :* qui calcule le degré entrant du sommet i
- *voisinageE(G,i):* qui calcule le voisinage entrant du sommet i
- *public int[] degreE() :* qui retourne un vecteur D tel que D[i-1] soit le degré entrant du sommet i.
- *public void affiche() :* qui affiche la matrice représentant le graphe.

TP2

Prérequis Graphe : Définition d'un graphe, Représentation par liste d'adjacence, degrés, voisinage.
Prérequis Java : Structures de contrôle impératives classiques, notions de classe, méthodes, tableaux, classe ArrayList.

On a créée, à partir de la classe prédéfinie ***ArrayList***, une classe **Liste** mettant en oeuvre les primitives classiques de traitement des listes.

- Attributs :
  private ArrayList L;
- Méthodes :
  public Liste() // Constructeur par défaut
  public Liste(int[] T) // Constructeur à partir d'un tableau.

```
public int taille ()              // taille de la liste
public int  tete ()               // premier élément
public void ajoutFin (int a)// liste où a est ajouté en fin de liste
public void ajoutTete (int a)     // liste où a est ajouté en tête de liste
public void reste ()              // on enlève l'élément de tête
public void concatene (Liste l2)// Concaténation de deux listes
public int elt (int i)            // l'élément d'indice i
public int indice (int a)         // indice de l'élément dans la liste (-1 si absent)
public boolean vide ()            // vrai ssi la liste est vide
public void enleveInd (int i)     // enlève l'élément d'indice i
public void enleveElt (int a)     // enlève l'élément a de la liste
public void tri ()                // renvoie la liste triée
public void affiche ()            // affiche le contenu de la liste
```

On pourra utiliser ces méthodes pour la gestion des listes d'adjacence.

2 Compléter la classe GrapheOrienteList

Cette classe définit un graphe par liste d'adjacence :

- Attributs :

```
private int n;     // le nombre de sommets
private int m;     // Le nombre d'arcs
private Liste[]  G; // Le vecteur des liste d'adjacence, G[i] est la liste des
                       successeurs   du sommet i.
```

- Méthodes :
  - Constructeur du graphe (par défaut, construction d'un graphe sans arc à n sommets)
  - Constructeur du graphe (à partir des listes d'adjacence, sous forme d'un tableau à deux dimension{{succ de 1},{succ de 2},..,{succ de n}})

On testera les constructeurs et les différentes méthodes sur le graphe G1.
Compléter les méthodes suivantes :

- *public void affiche()* : qui affiche le vecteur des listes d'adjacence
- *public void ajoutArc(int i, int j) :* qui ajoute un arc entre les sommets i et j
- *public void enleveArc(int i, int j) :* qui enlève l'arc entre i et j
- *public int degreS(int i) :* qui calcule le degré sortant du sommet i
- *public int[] degreS() :* qui calcule le vecteur des degrés sortant de tous les sommets
- *public int[] degreE() :* qui calcule le vecteur des degrés entrant de tous les sommets

TP3

3 Compléter les classes GrapheOrienteList, et GrapheOrienteMat

Écrire dans les deux classes une méthode main (programme principal), permettant de créer un graphe de test sur lequel vous pourrez tester toutes les méthodes écrites.

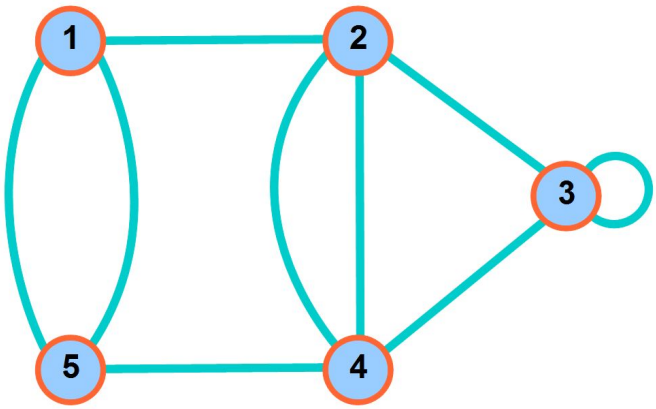
public static void main ()

Écrire les méthodes de conversions entre Mat et List :

- public GrapheOrienteList toList() dans la classe GrapheOrienteMat.
- public GrapheOrienteMat toMat() dans la classe GrapheOrienteList.

4 Créer les classes GrapheNonOrienteList, et GrapheNonOrienteMat

On testera les méthodes de cette classe sur le multigraphe non orienté G2 (page 8 du polycopié de cours)



Le graphe G2 sera représenté

- par liste d'arêtes par
  L=[[1,2],[1,5],[1,5],[2,3],[2,4],[2,4],[3,3],[3,4],[4,5]]
- par liste d'adjacence par
  G=[[],[2,5,5],[1,3,4,4],[2,3,4],[2,2,3,5],[1,1,4]]
- et par matrice d'adjacence par

M=[[0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 2],
[0, 1, 0, 1, 2, 0],
[0, 0, 1, 1, 1, 0],
[0, 0, 2, 1, 0, 1],
[0, 2, 0, 0, 1, 0]]

On testera également ces méthodes sur le graphe de Petersen et les graphes complets de Kuratowski.

Copier les classes orienté et adapter les méthodes pour qu'elles fonctionnent maintenant avec des graphes non orientés.

- Écrire dans la classe GrapheNonOrienteList un constructeur de graphe qui construise le vecteur des listes d'adjacences représentant le graphe de Kuratowski à n sommets.

A l'issue de ces TP, vous devez disposer du package complet vous permettant d'écrire les algorithmes des graphes vus en cours.