Révisions sur les graphes

1 Modélisation d'un graphe

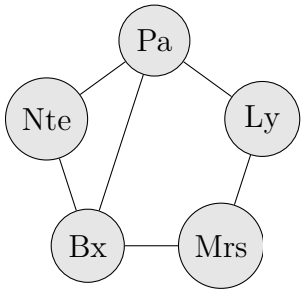
1.1 Vocabulaire

Graphe Un graphe (orienté ou non-orienté) G est un couple constitué de deux ensembles (S, A) :

- un ensemble S de points nommés **sommets** ;
- un ensemble A de traits nommées **arêtes** ou **arcs**.

Vocabulaire des graphes non-orientés :

- Arête
- Chaine
- Cycle
- Extrémité



Vocabulaire des graphes orientés :

- arc
- chemin
- circuit
- sucesseur
- prédecesseur

1.2 Modélisation en machine

Dictionnaire d'adjacence :

```
G={'Pa':['Nte','Bx','Ly'],'Nte':['Pa','Bx'],
   'Bx':['Pa','Nte','Mrs'],'Mrs':['Bx','Ly'],'Ly':['Pa','Mrs']}
```

Liste d'adjacence :

```
G=[[1,2,4],[0,2],[0,1,3],[2,4],[0,3]]
```

Les noms doivent être mémorisés indépendamment : `Label=['Pa','Nte', 'Bx','Mrs','Ly']`

Matrice d'adjacence

$$\mathbb{A}_G = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

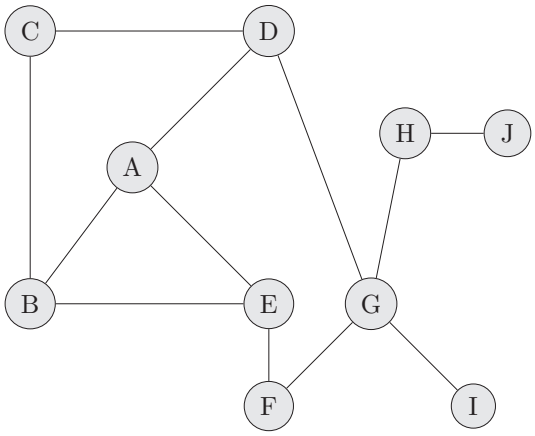
Ajout d'informations :

Graphe pondéré chaque arête ou arc a un poids ;

Graphe coloré chaque nœuds et/ou chaque arête a une catégorie (couleur).

2 Algorithmes à connaitre

2.1 Parcours en largeur



Algorithme 1 : Algorithme de parcours de graphe en largeur

```
1  Initialisation;
2  Initialiser la liste file avec le nœud départ ;
3  Initialiser la liste noeuds_connus avec le nœud départ ;
4  tant que file n'est pas vide faire
5      noeud_courant ← Défiler(file) ;
6      pour chacun des voisins v de noeud_courant faire
7          si v n'appartient pas à la liste noeuds_connus alors
8              Ajouter v à la liste noeuds_connus;
9              Enfiler v à la fin de la liste file;
10         fin
11     fin
12 fin
13 return noeuds_connus
```

2.2 Parcours en profondeur

Algorithme 2 : Algorithme de parcours de graphe en profondeur itératif

```
Initialisation;
Initialiser la pile avec le nœud de départ;
Initialiser la liste noeuds_connus;
tant que la pile n'est pas vide faire
    nœud courant ← Dépiler(pile) ;
    si il existe un voisin du nœud courant non connu alors
        Ajouter le voisin à la liste noeuds_connus ;
        Empiler le nœud courant ;
        Empiler ce voisin ;
    fin
fin
return noeuds_connus
```

2.3 Version recursive

Le pseudo-code de la version de parcours de graphe en profondeur est alors la suivante :

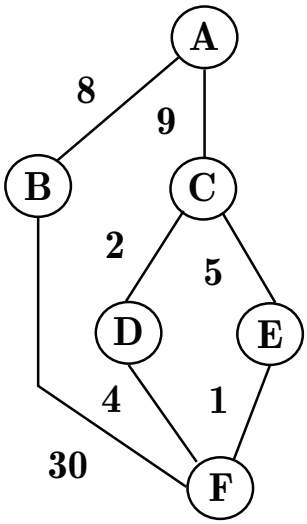
Algorithme 3 : Algorithme de parcours de graphe en profondeur récursif

```
Initialisation;
Initialiser la liste noeuds_connus avec le nœud départ ;
Visiter(noeud_depart,noeuds_connus);
return noeuds_connus
```

Algorithme 4 : Fonction récursive Visiter(u,noeuds_connus)

```
si u n'est pas dans la liste noeuds_connus alors
    Ajouter u à la liste noeuds_connus;
fin
pour chaque sommet v voisin de u non connu faire
    Visiter(v,noeuds_connus);
fin
```

2.4 Chemin le plus court



Algorithme 5 : Algorithme de Dijsktra

```
Initialisation;
Initialiser la liste DO des distances avec l'origine (=nœud de départ) : 0 et ∞;
Initialiser la liste Pré des prédésseurs : l'origine et ∅ ;
Initialiser la liste NT des nœuds à traiter comme la liste de tous les sommets;
tant que la liste NT n'est pas vide faire
    Trier la liste NT par ordre croissant des distances à l'origine (en utilisant DO);
    nœud courant ← le premier sommet de NT ainsi triée ;
    pour tous les voisins de nœud courant faire
        si si la distance entre un voisin et l'origine en passant par le nœud courant
          est inférieure à celle mémorisée alors
            Mettre à jour la distance dans DO;
            Mettre à jour le prédecesseur dans Pré
        fin
        Retirer le nœud courant de la liste NT
    fin
fin
return DO & Pré
```