8 - Exemples d'algorithmes opérant sur un graphe. Applications.

I- Quelques notions sur les Graphes

a- Notations et généralité

Un graphe est un ensemble de points nommés *sommets* reliés par des traits ou flèches nommées *arêtes* (ou *arcs*). L'ensemble des arêtes entre nœuds forme une figure similaire à un réseau.

L'ensemble des sommets est souvent noté V (*Vertices* en anglais) et l'ensemble des arêtes est noté E (*Edges*). E appartient à VxV et est même souvent bien inférieur car VxV signifie que tous les sommets sont reliés à tous les autres et à eux-mêmes.

Les arêtes peuvent être orientées (flèches) ou non orientées (traits). Si les arêtes sont orientées, la relation va dans un seul sens et est donc asymétrique, et le graphe lui-même est dit orienté. Sinon, si les arêtes sont non orientées, la relation va dans les deux sens et est symétrique, et le graphe est dit non orienté.

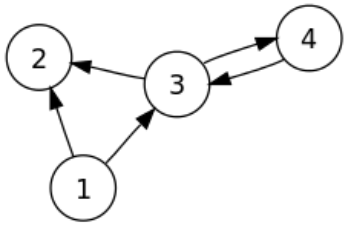


Figure 1 : graphe orienté

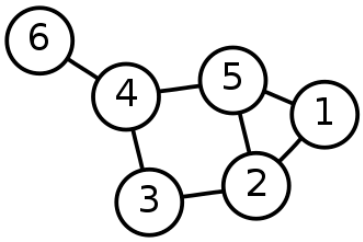


Figure 2 : graphe non orienté

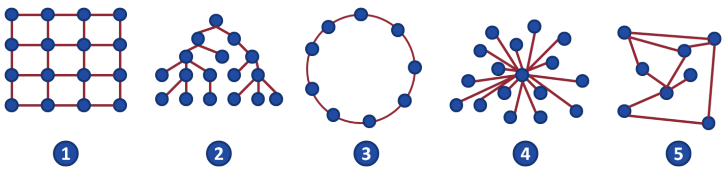
Les arrêtes sont notés 3 -> 2 où 3 est un prédécesseur de 2 et 2 est un successeur de 3.

Un graphe est donc un couple de sommets et d'arrêtes : G= (V,E).

Quelques définitions :

- Un chemin non vide d'un sommet *x* à ce même sommet *x* est un cycle.
- Si pour tout x, y, il existe un chemin x~~~> y (direct ou indirect) alors le graphe est dit fortement connexe.
- Dans un graphe pondéré, on a en plus une application de w.E dans R qui à chaque arrêtes associe sont poids : w(x->y)

Typologies : homogène (1), hiérarchique (2), cyclique (3) , centralisé (4), quelconque (5)



Quelques exemples de graphe dans la vraie vie :

- Un réseau informatique local
  - V = {machines}
  - E = {Câbles entre les machines}
  - Non orienté
- Métro
  - V = {Stations}
  - E = {tronçons entre deux stations}
  - Orienté par endroit
  - Pondération possible avec le temps de trajet
- Web
  - V = {Pages web}
  - E = {hyperliens}
  - Non orienté

Activité/développement : Construction d'un graphe

Jeu du berger, du loup, de la chèvre et du chou

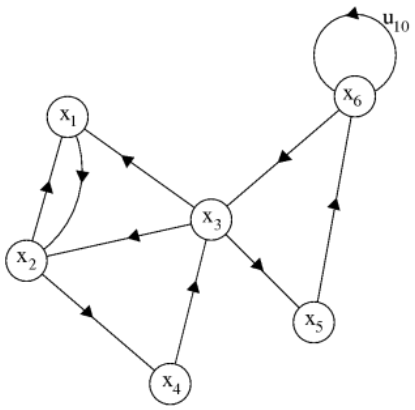
Un berger, un loup, une chèvre et un chou sont sur la rive d'une rivière.
Le berger possède un bateau qui peut transporter un seul des 3 à la fois.
Si le berger n'est pas présent : le loup mange la chèvre et la chèvre mange le choux.

Comment faire traverser la rivière aux 3 sans incidents ?

b- Exemple d'implémentation

- Matrice d'adjacence

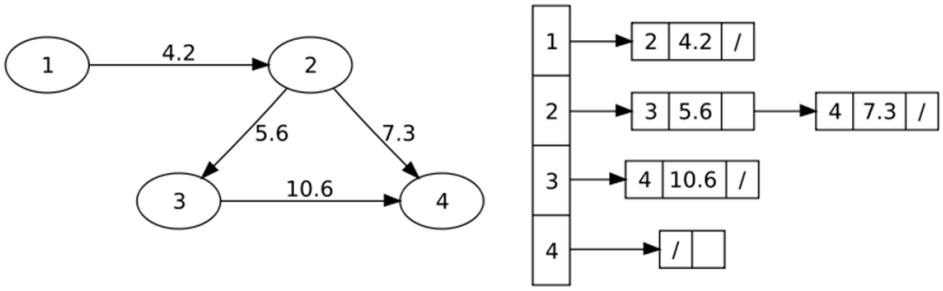
C'est une matrice carrée |V| X |V|, où les lignes représentent les sommets de départ et les colonnes les sommets d'arrivée. Une entrée peut désigner la présence d'un arc, ou peut porter une valeur d'arc (graphe pondéré). Dans le cas des graphes non orientés, la matrice est symétrique.



	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	1	0	0	0	0
x_2	1	0	0	1	0	0
x_3	1	1	0	0	1	0
x_4	0	0	1	0	0	0
x_5	0	0	0	0	0	1
x_6	0	0	1	0	0	1

- Liste d'adjacence

Les sommets sont stockés dans un tableau à une dimension. Chaque case du tableau est un objet sommet avec comme attribut : un nom et une liste chainée contenant les sommets auxquels il est relié, avec un poids pour l'arrête dans le cas d'un graphe pondéré.



Quelques opérations sur les graphes (on suppose le graphe immutable)

Opérations	Matrice d'adjacence	Liste d'adjacence
Afficher les sommets	O(|V|)	O(|V|)
Afficher les arrêtes d'un sommet	O(|V|)	O(|E|)
Afficher toutes les arrêtes	$O(\|V\|^2)$	O(|V|+|E|)
Test d'adjacence entre deux sommets	O(1)	O(|E|)

II- Algorithmes de parcours de graphes

a- Parcours en largeur

L'algorithme de parcours en largeur permet le parcours d'un graphe ou d'un arbre de la manière suivante : on commence par explorer un nœud source, puis ses successeurs, puis les successeurs non explorés des successeurs, etc.

```
ParcoursLargeur(Graphe G, Sommet s):
        f = File();
        f.enfiler(s);
        marquer(s);
        tant que la file est non vide
                s = f.defiler();
                afficher(s);
                pour tout voisin t de s dans G
                        si t non marqué
                                f.enfiler(t);
                                marquer(t);
```

La complexité en temps dans le pire cas est en O(|V| + |E|) où |V| est le nombre de sommets et |E| est le nombre d'arcs. En effet, chaque arc et chaque sommet est visité au plus une seule fois.

b- Parcours en profondeur

L'algorithme de parcours en profondeur (ou DFS, pour Depth First Search) est un algorithme de parcours d'arbre, et plus généralement de parcours de graphe, qui se décrit naturellement de manière récursive. Son application la plus simple consiste à déterminer s'il existe un chemin d'un sommet à un autre.

L'algorithme en version récursive :

```
ParcoursProfondeur(graphe G, sommet s)
       marquer le sommet s
       afficher(s)
       pour tout sommet t fils du sommet s
             si t n'est pas marqué alors
                    ParcoursProfondeur(G, t);
```

Version itérative avec une pile

```
ParcoursLargeur(Graphe G, Sommet s):
        p = Pile();
        p.empiler(s);
        marquer(s);
        tant que la pile est non vide
                s = p.dépiler();
                afficher(s);
                pour tout successeur t de s dans G
                        si t non marqué
                                p.empiler(t);
                                marquer(t);
```

c- <u>détection d'arc arrière</u>

Si G est un graphe, un arc arrière dans un des ses parcours en profondeur est un arc qui relie x avec une de ses ancêtres y dans l'arborescence.

Pour détecter ces arcs arrières, on va utiliser des couleurs lors d'un parcours en profondeur :

- Vert : pas traité
- Bleu : en cours de traitement
- Rouge : Fini de traiter

```
PP(G)
        Initialiser les sommets à VERT
        pour tout sommet s de G :
                si s.couleur == VERT alors
                        ParcoursProfondeur(G,s)


ParcoursProfondeur(graphe G, sommet s)
        s.couleur=BLEU
        pour tout sommet t successeur du sommet s
                si t.couleur == BLEU alors
                        // il y a un arc arrière s -> t
                Sinon si t.couleur == VERT alors
                        ParcoursProfondeur(G,t)
        s.couleur = ROUGE
```

III- Algorithmes d'arbre couvrant de poids minimum

a- Qu'est-ce qu'un arbre couvrant de poids minimum

En théorie des graphes, étant donné un graphe non orienté connexe dont les arêtes sont pondérées, un arbre couvrant de poids minimal de ce graphe est un arbre couvrant (sous-ensemble qui est un arbre et qui connecte tous les sommets ensemble) dont la somme des poids des arêtes est minimale.

Application : Le graphe d'un quartier où les sommets sont des maisons et les arrêtes les câbles du réseau internet avec une distance en mètre. En cherche à relier toutes les maisons entre elle en utilisant le moins de câble possible.

Un graphe peut comporter plusieurs arbres couvrants minimum différents, mais si tous les poids sont différents, alors il est unique.

b- L'algorithme de Kruskal

https://fr.wikipedia.org/wiki/Algorithme_de_Kruskal

```
Kruskal(G) :
        A = GrapheVide()
        L  = listeArrete()
        pour chaque arrete a de G :
                ajouter a à L
        trier les arêtes de L par poids croissant
   pour chaque arête a de L prise par poids croissant :
      si G.CreerCycle(a) est Faux :
         ajouter l'arête a au graphe A
      Supprimer a de L
   retourner A
```

Complexité : Ajout des arrêtes dans L : O(|E|)

Trie : O(|E| log |E|)

Parcours avec détection de cycle : $O(|E|^2 + |V|)$

La complexité est donc quadratique mais avec une structure union-find on peut réduire la complexité à celle du tri.

c- L'algorithme de Prim

https://fr.wikipedia.org/wiki/Algorithme_de_Prim

L'algorithme7 consiste à faire croître un arbre depuis un sommet. On commence avec un seul sommet puis à chaque étape, on ajoute une arête de poids minimum ayant exactement une extrémité dans l'arbre en cours de construction. En effet, si ses deux extrémités appartenaient déjà à l'arbre, l'ajout de cette arête créerait un deuxième chemin entre les deux sommets dans l'arbre en cours de construction et le résultat contiendrait un cycle.

IV- Algorithme du plus cours chemin : Dijkstra

Développer à la main