2019-2020, semestre automne
L3, Licence Sciences et Technologies

LIFAP6: Algorithmique, Programmation et Complexité

Chaine Raphaëlle (responsable semestre automne)
E-mail : raphaelle.chaine@liris.cnrs.fr
http://liris.cnrs.fr/membres?idn=rchaine

Graphe

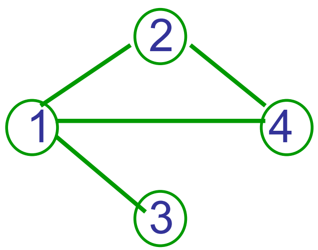
- Idée générale
  - Notion plus générale que la notion d’arbre
  - Arbre = cas particulier d’un graphe

- Graphe
  - Modélisation de relations binaires entre des éléments

Remarque : En théorie des graphes un arbre n’est pas forcément orienté!

Définitions

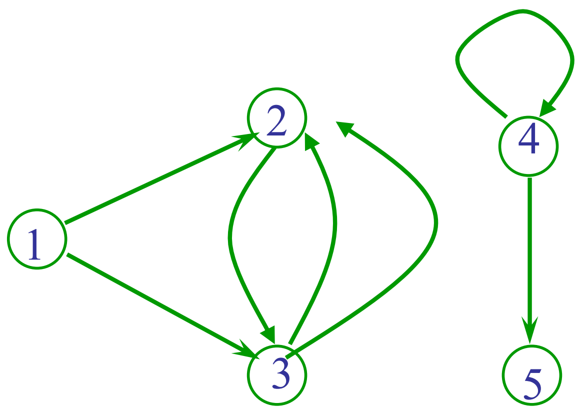
- Un **graphe non orienté** est un couple <S, A> où
  - S est un ensemble fini de **sommets**
  - A est un ensemble fini de paires de sommets, appelées **arêtes**



$S = \{ 1, 2, 3, 4 \}$
$A = \{ \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 4\} \}$

Définitions

- Un **graphe orienté** est un couple <S, A> où
  - S est un ensemble fini de **nœuds**
  - A un ensemble fini de paires ordonnées de nœuds, appelées **arcs**



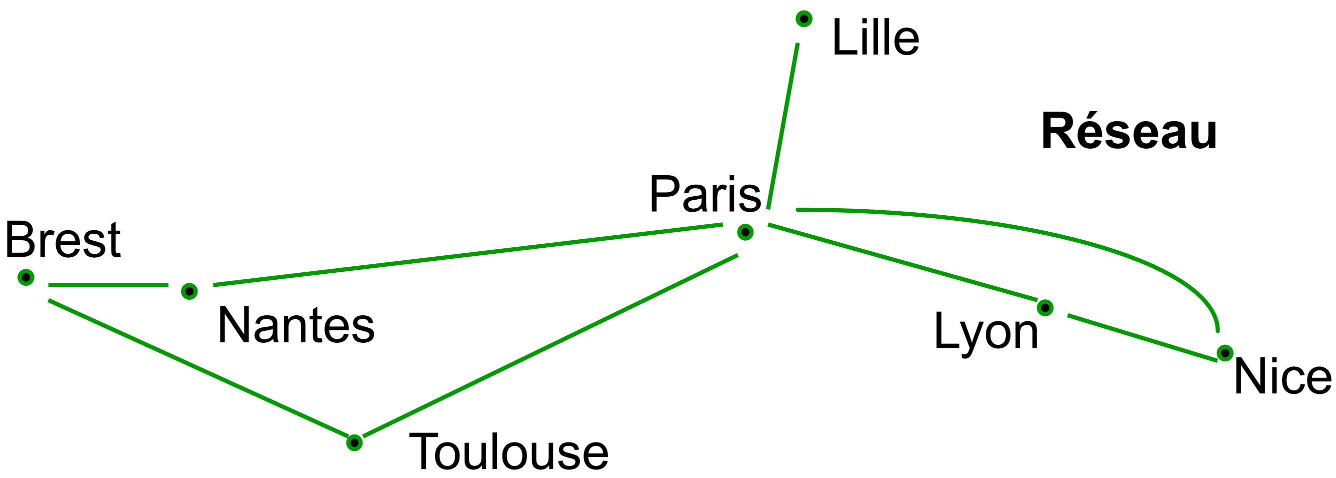
$S = \{ 1, 2, 3, 4, 5 \}$
$A = \{ (1, 2), (1, 3), (2, 3), (3, 2), (3, 2), (4, 4), (4, 5) \}$

Définitions

- Il arrive qu'une information de coût soit associée aux arcs (ou arêtes), voire aux nœuds (resp. sommets)
- Un **graphe valué** est un triplet <S, A, C> où
  - S est un ensemble fini de nœuds (ou sommets),
  - A un ensemble fini d'arcs (ou d'arêtes)
  - C une fonction de A dans R appelée **fonction de coût**

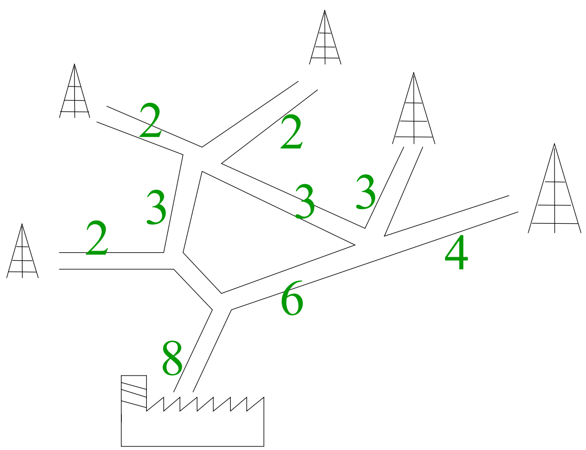
Exemples

- GPS : Localités reliées par des voies de communication (routes, voies ferrées, lignes aériennes, …), la fonction de coût pouvant correspondre à une distance, ou un temps de parcours, le prix du trajet...
  → Recherche de plus courts chemins



Exemples

- Représentation de localités reliées par des canalisations (eau, gaz, électricité) caractérisées par leur débit et leur capacité. Certains nœuds pouvant correspondre à des stations de distribution ou de pompage
  → Problème de flux maximal



Pipelines

Exemples

- Représentation
  - des configurations possibles d'un jeu tactique par des nœuds,
  - des coups légaux par des arcs permettant de passer d'une configuration à une autre

→Recherche position gagnante, séquences de coups forcés, etc.

Algorithmes

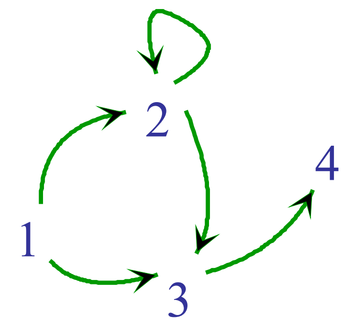
- **Explorations**
  – Parcours en profondeur, en largeur
  – Tri topologique
  – Composantes fortement connexes, ...
- **Recherche de chemins**
  – Clôture transitive
  – Chemin de coût minimal
  – Circuits Eulériens (passer 1 et 1 seule fois sur chaque arête) et Hamiltoniens (passer 1 et 1 seule fois sur chaque sommet), ...
- **Arbres couvrants**
  – Algorithmes de Kruskal et Prim

Algorithmes

- **Réseaux de transport**
  - Flot maximal
- **Classification**
  - Coupures de graphes
- **Divers**
  - Coloration d'un graphe
  - Test de planéarité, ...

Terminologie

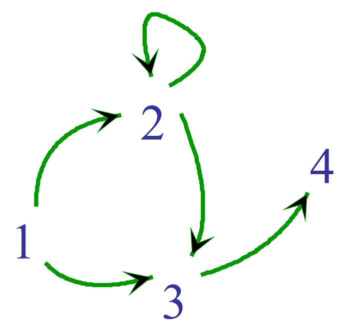
- Etant donné un arc (x, y)
  - x est l'**extrémité initiale** de l'arc,
  - y l'**extrémité terminale**



- On dit que
  - x et y sont **adjacents**
  - y est un **successeur** de x
  - x est un **prédécesseur** de y
- Les arcs qui partent d'un nœud lui sont **incidents extérieurement**, ceux qui y arrivent lui sont **incidents intérieurement**

Terminologie

- Dans un graphe orienté (resp. non orienté) G, on appelle **degré** d'un nœud (resp. sommet) x et on note d° (x) le nombre d'arcs (resp. d'arêtes) dont x est une extrémité
- **Demi-degré extérieur** d'un nœud : nombre d'arcs incidents extérieurement
- **Demi-degré intérieur** d'un nœud : nombre d'arcs incidents intérieurement



Terminologie

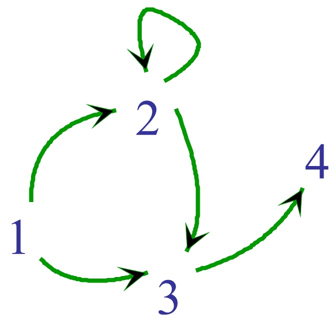
- Dans un graphe orienté (resp. non orienté) G, on appelle **chemin** (resp. **chaîne**) de longueur l une suite de l+1 nœuds (resp. sommets) (s_0, …, s_l) tels que

$$\forall i, 0 \leq i \leq l-1, s_i \rightarrow s_{i+1} \text{ est un arc (resp. arête) de } G$$

- Un chemin (resp. une chaîne) est dit **élémentaire** s’il ne contient pas plusieurs fois le même nœud (resp. sommet)
- Un **circuit** (resp. un **cycle**) est un chemin (resp. une chaîne) tel (resp. telle) que les 2 sommets aux extrémités coïncident

Terminologie

- Un graphe orienté est dit **fortement connexe** si pour toute paire de noeuds distincts (u,v) il existe un chemin de u vers v et un chemin de v vers u.
- Un graphe non orienté est dit **connexe** si pour toute paire de sommets distincts (u,v) il existe une chaîne entre u et v



Terminologie

- Un **arbre** est un graphe non orienté, connexe et sans cycle

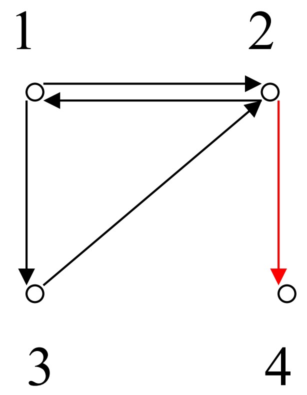
- Etant donné un graphe orienté G = <S,A>, on appelle **racine** de G un nœud r de S tel que tout autre nœud puisse être atteint par un chemin d’origine r
- On appelle **arborescence** un graphe orienté admettant une racine et tel que le graphe non orienté associé soit un arbre

Type Abstrait Graphe

- Prévoir des procédures
  - d'initialisation, de testament, (d'affectation)
  - d'ajout / de retrait d'un nœud ou d'un arc
- Prévoir des fonctions
  - testant l'existence d'un nœud ou d'un arc
- Prévoir des routines
  - Permettant d'itérer
    - sur l'ensemble des arcs issus d'un nœud (par exemple une fonction ième arc)
    - Sur l'ensemble des successeurs d'un nœud (par exemple une fonction ième successeur)
  - De connaître l'origine (resp. l'extrémité) d'un arc

Représentations possibles

- Matrices d'adjacence
  - Chaque sommet est représenté par un indice dans 1..n
  - Ensemble des arcs représenté par un tableau de booléens de dimension n*n, où n est le nombre de nœuds



	1	2	3	4
1	F	V	V	F
2	V	F	F	V
3	F	V	F	F
4	F	F	F	F

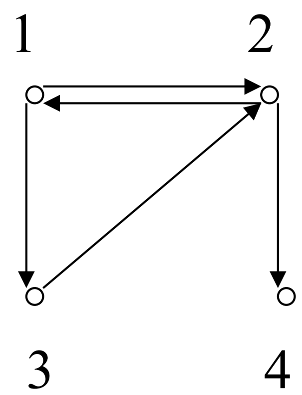
- Si le graphe est valué on remplace les booléen par le coût associé à l’arc (en réservant une valeur spécifique pour signifier l’absence d’arc)
- Types
  - Nœud = 1..n
  - Graphe = tableau [Nœud, Noeud] de valeurs
- Utilisable si pas plus d’un arc entre 2 nœuds

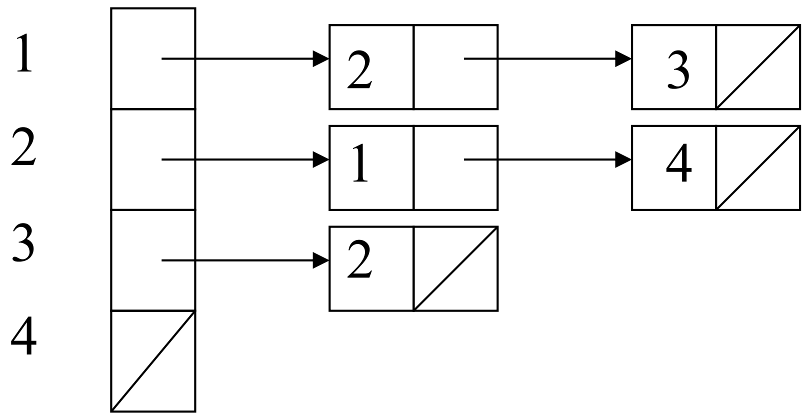
- Quelle propriété est vérifiée par la matrice d’adjacence d’un graphe non orienté?
- Quel est la complexité du parcours des successeurs d’un nœud?

- Matrice d’adjacence
  - Représentation commode pour tester l’existence d’un arc, pour ajouter ou supprimer un arc (en $\Theta(1)$)
  - Parcours de tous les successeurs ou de tous les prédécesseurs d’un sommet en $\Theta(n)$
- Une consultation de l’ensemble des arcs du graphe requiert un temps en $\Theta(n^2)$ et cette représentation exige un espace mémoire de $\Theta(n^2)$

Représentations possibles

- Listes d'adjacence
  - Etant donnée une représentation des nœuds, on associe à chaque nœud la **liste de ses successeurs** rangés dans un certain ordre
  - Cette représentation utilise un espace mémoire en $\Theta(n+p)$ pour un graphe avec n nœuds et p arcs



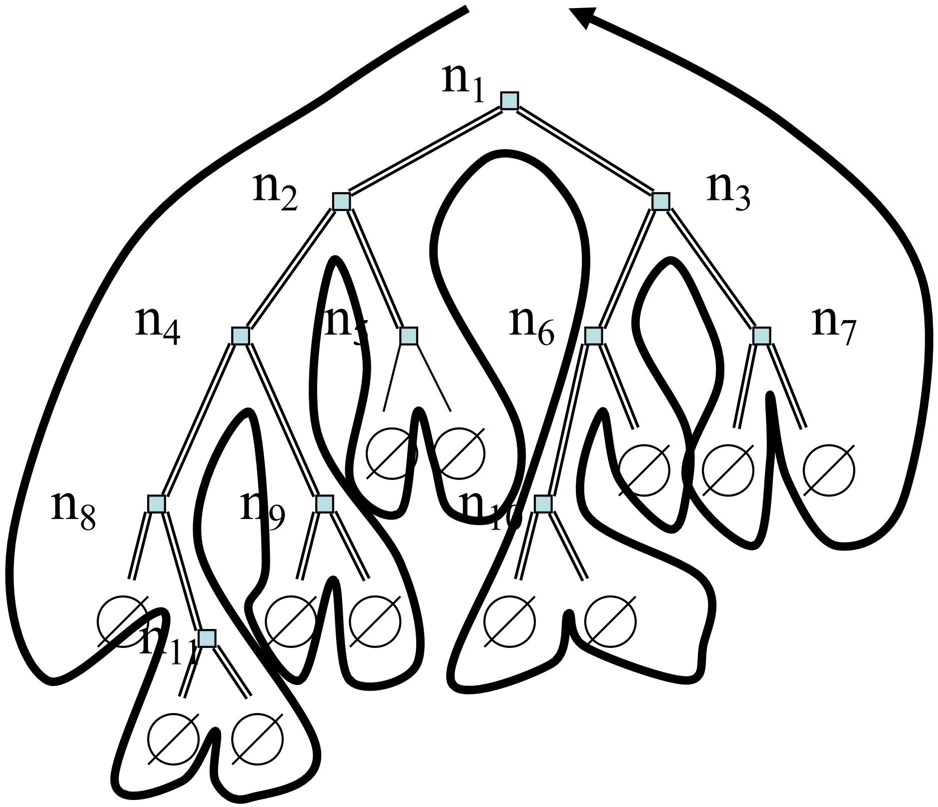


- Parcours des successeurs d’un nœud n_i en $\Theta(deg^+(n_i))$
- Mais parcours des prédécesseurs d’un nœud n_i en $\Theta(p)$

Parcours de graphe

- But : parcourir l’ensemble des nœuds du graphe, pour effectuer une certaine action en chacun des nœuds (ex. affichage d’information)
- Exploration d’un graphe plus compliquée que celle d’un arbre, mais elle s’en inspire!
- Il existe 2 grandes stratégies de parcours :
  – le parcours en profondeur (*depth-first search*)
  – le parcours en largeur (*breadth-first search*)

Rappel : Parcours d'un arbre (binaire)



Parcours en profondeur « à main gauche »

Se généralise à un arbre n-aire :

on rencontre les nœuds n+1 fois

Parcours de graphe

- Lors du parcours, on matérialisera l’état d’un nœud par une couleur
  - Blanc : nœud non découvert
  - Gris : nœud découvert
  - Noir : nœud dont tous les successeurs ont été parcourus

Parcours en profondeur

- On considère un graphe orienté dont tous les nœuds sont initialement non découverts (blanc)
- Le parcours en profondeur consiste à
  - choisir un nœud de départ s et le marquer comme découvert (gris)
  - suivre un chemin issu de s **aussi loin que possible** en marquant les nœuds en gris au fur et à mesure qu'on les découvre
  - en fin de chemin (marquage en noir), **revenir au dernier choix fait** et prendre une autre direction.

Parcours en profondeur

- Utilisation de propriétés associées aux nœuds
  - Couleur (père, ordre de découverte, fin de découverte …)
- Pour y stocker des informations sur l'exploration
- Certaines de ces informations ne sont pas indispensables à l'exploration et dépendront des besoins de l'application
- Seule l'info de couleur est indispensable
- Possibilité de stocker ces infos de manière non invasive
  - dans des Tableaux ou des Tables

Parcours en profondeur (DFS)

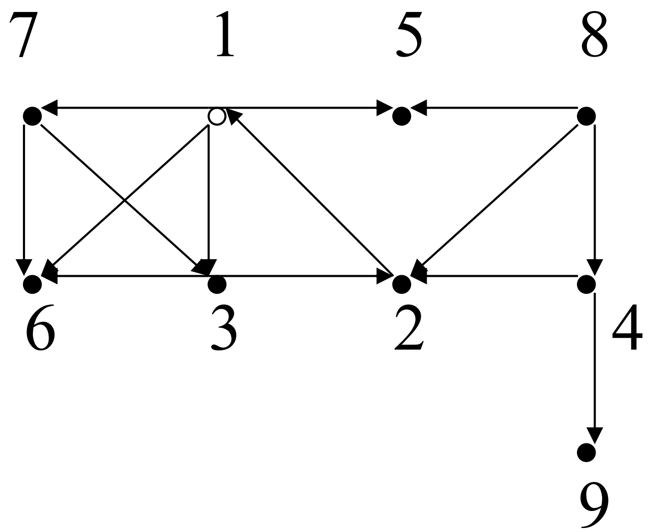
```
procédure DFS(donnée-résultat G : graphe)
variable
    u : sommet
début
  pour tout noeud u de G faire
  |  colorie(u,blanc); set_père(u, nil)
  finpour
  temps ← 0

  pour tout noeud u de G faire
  |  si couleur(u) = blanc alors
  |  |   set_tps_découverte(u, temps);  temps ++
  |  |   colorie(u,gris)
  |  |   DFS_visite(G,u)
  |  finsi
  finpour
fin
```

Procédure interne récursive

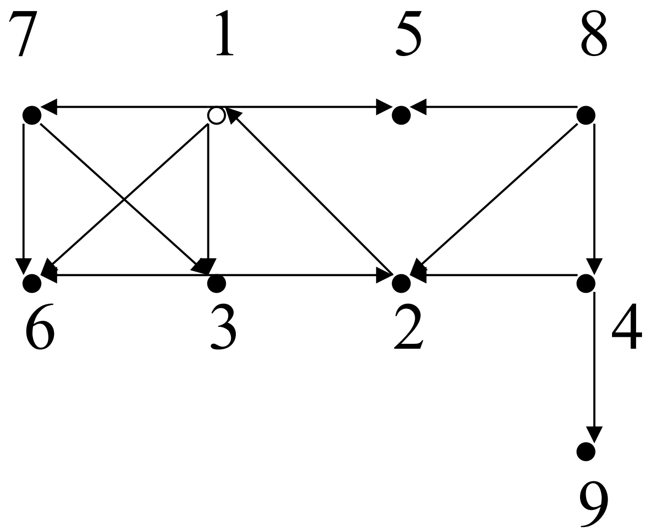
Parcours en profondeur (DFS)

```
Procédure DFS_visite(donnée-résultat G : graphe, u : sommet)
variable
    v : Nœud
début
  pour tout  nœud v successeur de u faire
    si couleur(v) = blanc alors
      set_tps_découverte(v, temps);  temps ++
      colorie(v,gris)
      set_père(v,u)
      DFS_visite(G, v)
    finsi
  finpour
  colorie(u,noir) (facultatif, 2 couleurs suffisent)
  set_tps_fin(u,temps); temps++
fin
```

- Effectuer le parcours en profondeur sur le graphe suivant



- On obtient l’ordre de parcours suivant des sommets en partant de 1 : 1, 3, 2, 6, 5, 7, 4, 9, 8



Parcours en profondeur (DFS)

- *Complexité en temps* :
  O(N+M) où N est le nombre de sommets et M le nombre d’arcs
  – en effet, un sommet n’est empilé qu’une seule fois (passage de blanc à gris) et chaque arc n’est traité qu’une seule fois (à partir de son extrémité initiale)
- *Complexité en espace* : O(M)
  – un nœud est empilé au plus 1 fois,
  – la pile d’appel a une profondeur max O(M) (longueur max d’un chemin entre deux nœuds)

Parcours en largeur (BFS)

- Pour un sommet de départ s on commence par visiter tous les successeurs de s **avant** de visiter les autres descendants de s
- Le parcours en largeur consiste à visiter d'abord
    - tous les sommets à distance 1 de s,
    - puis ceux à distance 2 qui n'ont pas été visités,
    - et ainsi de suite …
- Le parcours en largeur permet donc de résoudre les problèmes de plus court chemin dans un graphe non valué

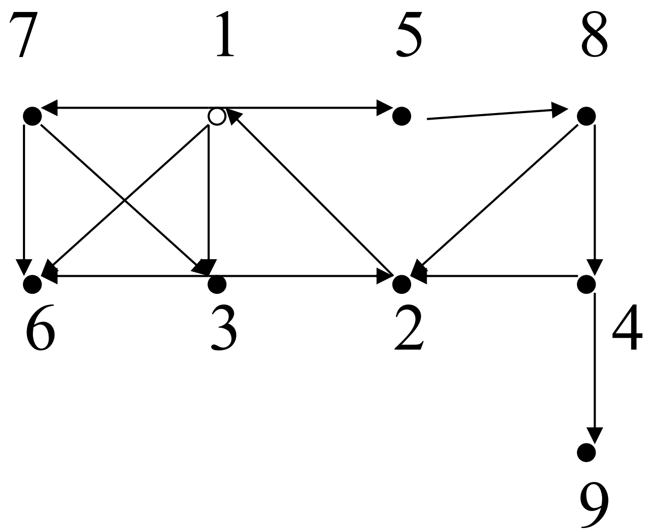
Parcours en largeur (BFS)

- Pour programmer l'algorithme, on utilise une structure de **file**:
  - lorsque à partir de s, on s'apprête à visiter ses successeurs non marqués, il est nécessaire de les ranger successivement dans une file
  - la recherche repartira ainsi de chacun des successeurs de s, à partir du premier.

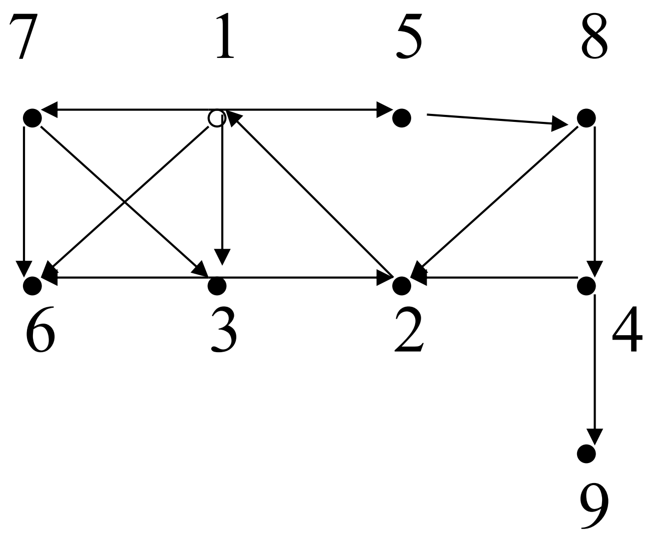
Parcours en largeur (BFS)

```
procédure BFS(donnée-résultat G: graphe, s : noeud)
variables
    u,v : nœud ; F : file
début
 pour tout noeud u ≠ s faire
     colorie(u,blanc); set_père(u, nil); set_dist(u,∞)
 finpour
 colorie(s,gris); set_père(s,nil); set_dist(s,0)
 initialise_file_vide(F); enfile(F,s)
 tant que non est-vide(F) faire
     u ← tête(F);
     pour tout noeud v successeur de u faire
         si couleur(v)=blanc alors
             colorie(v,gris); set_père(v,u); set_dist(v, u.dist +1)
             enfile(F,v)
         finsi
     finpour
     défile(F); colorie(u,noir) (facultatif : 2 couleurs suffisent))
 fintantque
fin
```

- Effectuer le parcours en largeur sur le graphe suivant



- Les sommets sont visités dans l’ordre
  1, 3, 5, 6, 7, 2, 8, 4, 9



- *Complexité en temps* : O(N+M)
  où N est le nombre de nœuds et M le nombre d'arcs
  – en effet, un sommet n'est mis dans la file qu'une seule fois (passage de blanc à gris) et les arêtes sont toutes parcourues 1 fois (découverte des voisins)

- *Complexité en espace* : O(N)
  la file a une longueur au plus en O(N) (si s est connecté à tous les autres sommets)

- En fait, parcours en largeur et en profondeur s’inscrivent dans une même stratégie générale d’exploration des nœuds du graphe
- Diffèrent suivant que les successeurs d’un nœud seront rangés dans une pile ou une file

```
procédure ExplorerGraphe (donnée-résultat G: Graphe, x : noeud)
variable
  E : Salle d'attente de Noeud
début
  pour tout nœud n de G faire
    colorie(n,blanc) …
  finpour
  initialiser(E); colorie(x,gris); ajouter(E,x)
  répéter
    y← sommet(E); retirer_sommet(E)
    pour tout nœud z successeur de y faire
      si couleur(z) = blanc alors
        colorie(z, gris)
        ajouter(z,E)
      finsi
    finpour
    colorie(y, noir) (facultatif))
  jusque estvide(E)
fin
```

Si E correspond à une Pile : parcours en profondeur
Si E correspond à une File : parcours en largeur