# Théorie des Graphes

Université Libanaise
Faculté des Sciences
License Informatique
2ème année – S3

# Syllabus

1. Concepts introductifs
2. Introduction aux graphes et à leurs utilisations
3. Arbres et graphes bipartis
4. Distance et connexité
5. Matrices
6. Algorithmes sur les graphes
7. Graphes Eulériens et Hamiltoniens
8. Coloration des graphes
9. Graphes planaires
10. Digraphes et réseaux

# Algorithmes sur les graphes

Semaine 6

# Plan

- Recherche dans les graphes
  - BFS
  - DFS
- Codage des arbres

# Recherche dans les graphes

- De nombreux problèmes de théorie des graphes nous obligent à explorer un graphe à la recherche d'une structure particulière telle qu'un arbre couvrant, un cycle hamiltonien, un circuit eulérien, un sommet coupé, un appariement, etc.
- Nous discutons de deux des techniques de recherche les plus importantes:
  - Recherche en largeur d’abord ou Breadth-First Search [BFS]
  - Recherche en profondeur d'abord ou Depth-First Search [DFS]

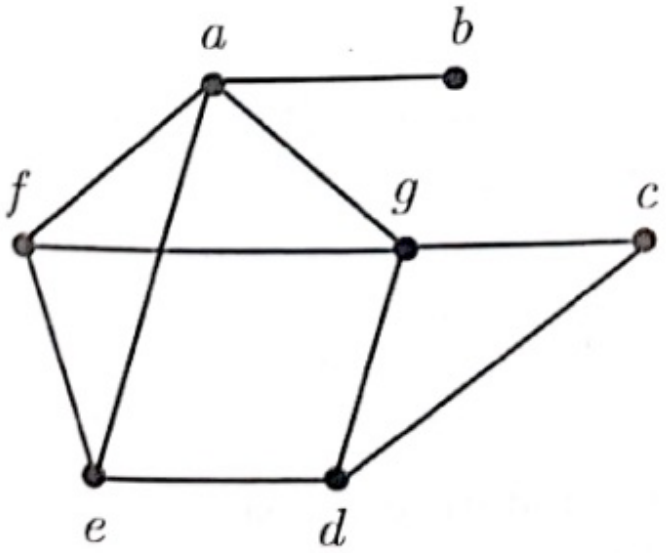
# Recherche dans les graphes

- Nous supposerons que le graphe est stocké en utilisant
  - matrice d'adjacence, ou
  - listes d'adjacence [une méthode généralement plus efficace]: pour chaque sommet, ses voisins sont listés dans un ordre donné.

# Recherche dans les graphes
# Exemple

- Matrice d'adjacence

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$




- Liste d'adjacence

| Sommet $v$ | Liste d'adjacence $N(v)$ |
|---|---|
| $a$ | $b, e, f, g$ |
| $b$ | $a$ |
| $c$ | $d, g$ |
| $d$ | $c, e, g$ |
| $e$ | $a, d, f$ |
| $f$ | $a, e, g$ |
| $g$ | $a, c, d, f$ |

# Plan

- Recherche dans les graphes
  - BFS
  - DFS
- Codage des arbres

# Recherche en largeur d'abord [BFS]

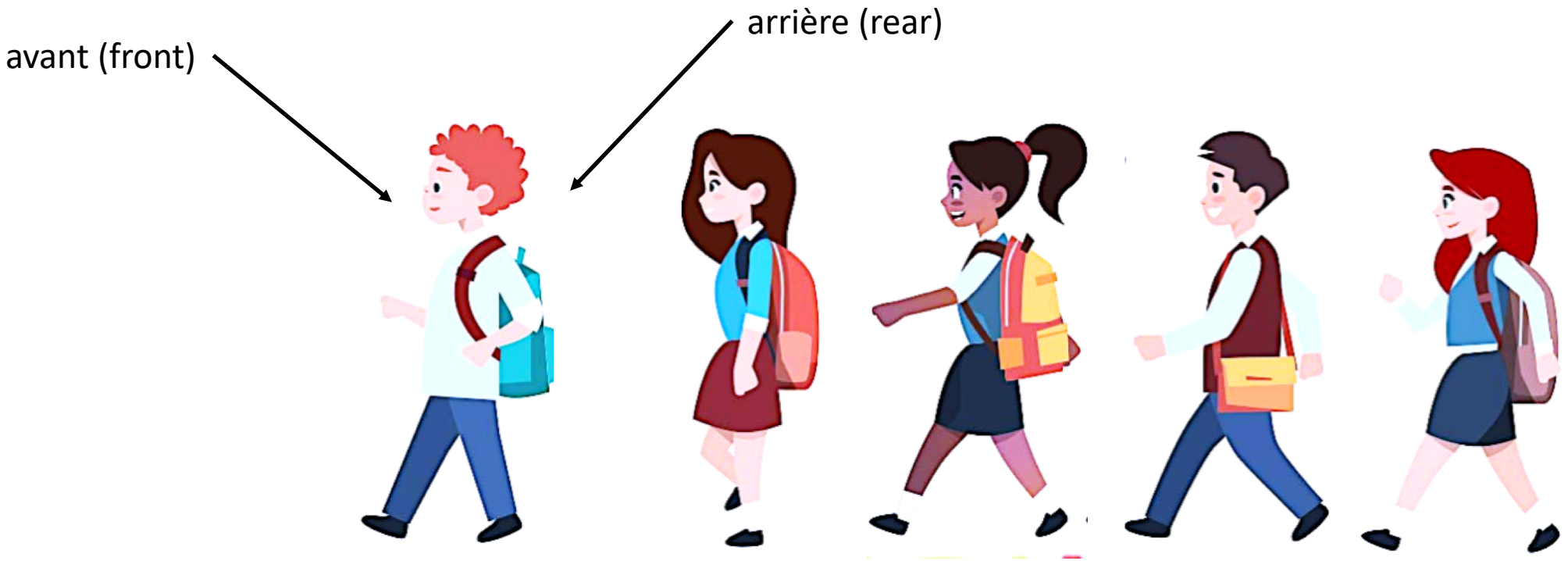
- Étant donné un graphe $G$ et un sommet $v \in V(G)$, nous allons maintenant présenter un algorithme pour construire un arbre couvrant $T$ tel que $d_T(v,x) = d_G(v,x)$ pour tout x$\in V(G)$.
- En d'autres termes, le sommet $v$, appelé racine, a toutes ses distances préservées.
- Un tel sommet est appelé **préservant de la portée** .
- L'algorithme que nous allons maintenant présenter s'appelle une recherche en **largeur d'abord** et est très important en informatique . Nous l'abrévions **BFS** .

# Qu'est-ce qu'une File d'attente (Queue)?

- L'ordre dans lequel les données arrivent est important.
- Exemple: une file d'attente est une ligne de sujets en attente d'être servis dans un ordre séquentiel commençant au début de la ligne ou de la séquence.
- **Définition** :
  - Une file d'attente est une liste ordonnée
  - les insertions se font à une extrémité (arrière)
  - les suppressions se font à l’autre extrémité (avant)
  - Le premier élément inséré est le premier à être supprimé
  - Premier arrivé, premier servi ou First In First Out (FIFO) ou dernier entré dernier sorti ou Last in Last Out (LILO).

# Qu'est-ce qu'une file d'attente (Queue)?

- **`EnFiler`** ou **`EnQueue`** : un élément est inséré dans une file d'attente
- **`DeFiler`** ou **`DeQueue`** : un élément est supprimé de la file d'attente



# Recherche en largeur d’abord [BFS]

- BFS construira un arbre couvrant par étapes, en conservant deux listes croissantes, une liste des sommets visités et une liste d'arêtes.
  - La liste initiale des sommets visités est assez courte; elle ne contient que la racine.
  - La liste initiale des arêtes est encore plus courte. C'est vide.

# Recherche en largeur d’abord [BFS] Algorithme

1. Créez une file d'attente vide.
2. Créez la liste des sommets visités, définissez la racine comme visitée et ajoutez la racine à la file d'attente.
3. Boucle sur la file d'attente tant qu'il y a des éléments dans la file d'attente
    1. Si aucun élément dans la file d'attente, l'algorithme se termine
    2. S'il y a un élément en avant:
        - supprimez-le de la file d'attente;
        - Traitez ce sommet (imprimer,….);
        - Marquez ses sommets adjacents comme visités et ajoutez-les dans la file d'attente;
        - Continuez à l'étape 3.

# Recherche en largeur d’abord [BFS]

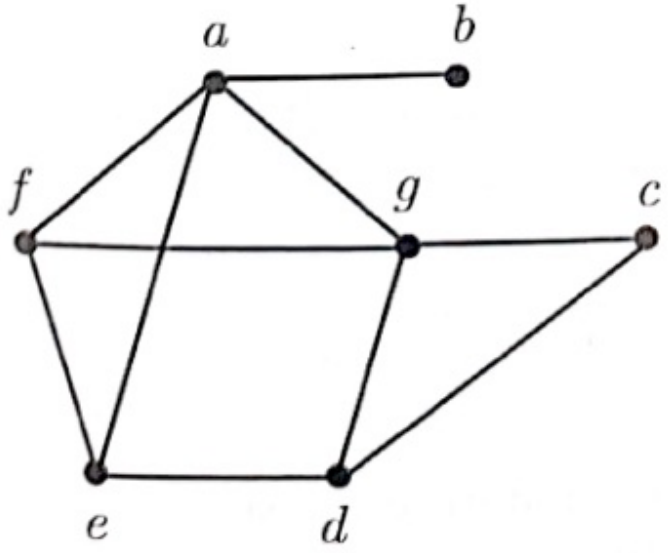
Applications de BFS

- Recherche de tous les composantes connexes dans un graphe
- Recherche de tous les sommets dans une composante connexe
- Trouver le chemin le plus court entre deux sommets
- Tester un graphe pour la biparticité

# Recherche en largeur d'abord [BFS]

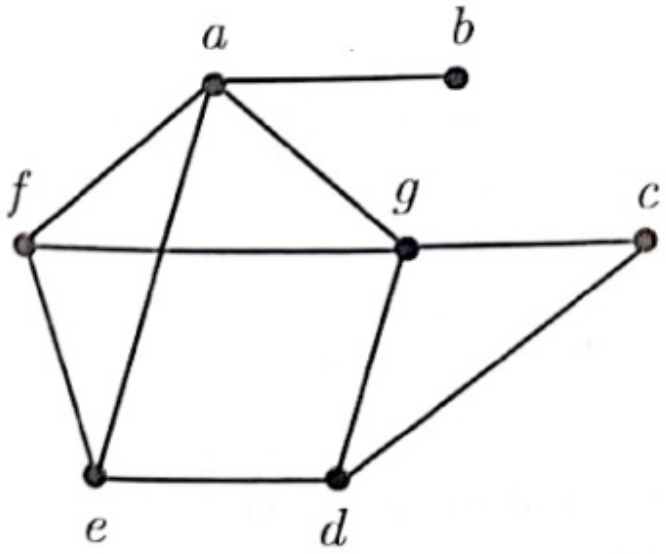
**Exemple**

Illustrez BFS pour le graphe suivant. D'abord en utilisant le sommet $a$ comme racine, puis en utilisant le sommet $d$ comme racine. Dessinez les arbres couvrants résultants.




| $v$ | $N(v)$ |
|---|---|
| $a$ | $b, e, f, g$ |
| $b$ | $a$ |
| $c$ | $d, g$ |
| $d$ | $c, e, g$ |
| $e$ | $a, d, f$ |
| $f$ | $a, e, g$ |
| $g$ | $a, c, d, f$ |

# Recherche en largeur d'abord [BF




**Exemple**

Liste des visités

| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|
| $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ |

| $v$ | $N(v)$ |
|---|---|
| $a$ | $b, e, f, g$ |
| $b$ | $a$ |
| $c$ | $d, g$ |
| $d$ | $c, e, g$ |
| $e$ | $a, d, f$ |
| $f$ | $a, e, g$ |
| $g$ | $a, c, d, f$ |

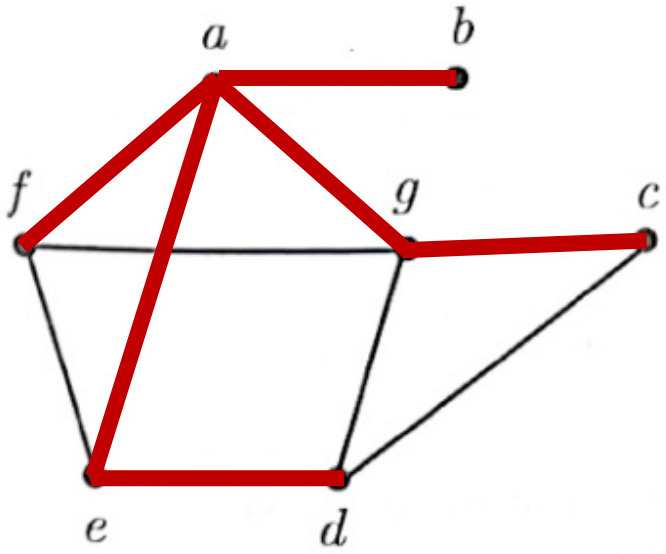
$a$ est adjacent à $b$
$a$ est déjà visité
nous ne l'enfilons pas !

File

| $a$ | $b$ | $e$ | $f$ | $g$ | $d$ | $c$ |
|---|---|---|---|---|---|---|

ordre de BFS

# Recherche en largeur d'abord [BF




**Question rapide**

Comment pouvez-vous modifier cet algorithme pour trouver un arbre couvrant?

**Exemple**

Liste des visités

| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|
| $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ |

Liste des antécédents

| $-1$ | $a$ | $g$ | $e$ | $a$ | $a$ | $a$ |
|---|---|---|---|---|---|---|
| $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ |

arêtes de l'arbre couvrant

File

| $a$ | $b$ | $e$ | $f$ | $g$ | $d$ | $c$ |
|---|---|---|---|---|---|---|

ordre de BFS

| $v$ | $N(v)$ |
|---|---|
| $a$ | $b, e, f, g$ |
| $b$ | $a$ |
| $c$ | $d, g$ |
| $d$ | $c, e, g$ |
| $e$ | $a, d, f$ |
| $f$ | $a, e, g$ |
| $g$ | $a, c, d, f$ |

**Question rapide**

Que faire si le graphe est pondéré? Comment pouvons-nous utiliser BFS pour implémenter Prim?

# Recherche en largeur d’abord [BFS]

- Il convient de noter qu'il existe souvent de nombreux arbres couvrants BFS d'un graphe non étiqueté donné.
- Pour un graphe étiqueté, l'arbre couvrant BFS est déterminé par l'ordre donné aux étiquettes. Nous avons utilisé l'ordre alphabétique en général.

# Recherche en largeur d’abord [BFS]

- Un arbre couvrant $T$ d'un graphe $G$ satisfait l'inégalité $rad(T) \geq rad(G)$.
- **Théorème**. Tout graphe connexe $G$ a un arbre couvrant $T$ tel que $rad(T) = rad(G)$.

# Recherche en largeur d’abord [BFS]

- Un arbre couvrant $T$ d'un graphe $G$ satisfaisant $rad(T) = rad(G)$ est appelé un **arbre couvrant à préservation de rayon ou radius-preserving spanning tree(RPST)**.
- Malheureusement, il n'est pas vrai que chaque graphe connexe ait un **arbre couvrant préservant le diamètre (DPST)**. Le cycle $C_n$ n'a que le chemin $P_n$ comme arbre couvrant et le chemin double environ le diamètre du cycle.
- **Théorème** . Si $T$ est un arbre couvrant d'un graphe $G$, préservant le diamètre, alors $rad(T) = rad(G)$; c'est-à-dire $T$ est également un arbre couvrant préservant le rayon.

# Plan

- Recherche dans les graphes
  - BFS
  - DFS
- Codage des arbres

# Recherche en profondeur d'abord [DFS]

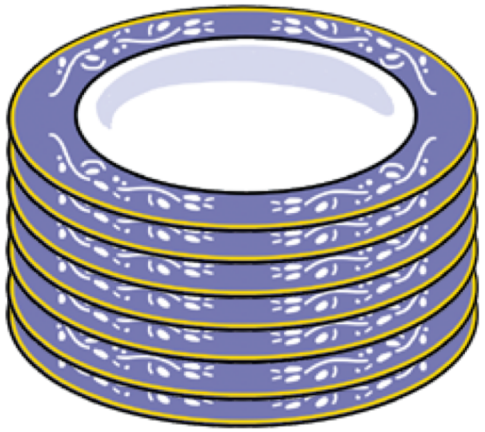
- La recherche en profondeur d'abord ( **DFS** ) est un algorithme de recherche important.
- La façon dont cela fonctionne sur un graphe est de commencer à un sommet, la **racine** , et de voyager aussi loin que possible le long d'un chemin $P$ émanant de $v$. Il revient ensuite vers $v$ jusqu'à ce qu'il trouve un premier sommet $u$ à partir duquel il y a une arête vers un sommet $w$ non encore atteint . Il procède ensuite à la génération d'un chemin $P'$ en se déplaçant le long de l'arête $uw$ vers des sommets supplémentaires non encore atteints. Le chemin $P'$ avec $P$ est un sous-arbre de $G$. L'algorithme continue à nouveau de visiter de nouveaux sommets sur un chemin jusqu'à ce qu'il ne puisse pas aller plus loin, puis de revenir en arrière, en recherchant un autre sommet à partir duquel il peut se dériver.
- De cette façon, DFS générera un arbre couvrant $T$ de $G$ et aura traversé chaque arête de $T$ exactement deux fois, une fois en avant le long d'un chemin de nouveaux sommets et une fois en arrière à la recherche d'un nouvel emplacement de branche

# Recherche en profondeur d'abord [DFS]

- Dans l'algorithme DFS, nous partons du point de départ et plongeons dans le le graphe jusqu'à ce que nous atteignions une impasse, puis nous nous déplaçons vers le nœud parent (Backtrack).
- Dans DFS, nous utilisons une **pile** ou **stack** pour obtenir le prochain sommet pour lancer une recherche.
- Alternativement, nous pouvons utiliser la récursivité (système de pile) pour faire de même.

# Qu'est-ce qu'une Pile (Stack)?

- L'ordre dans lequel les données arrivent est important



- **Définition** : Une *pile* est une liste ordonnée dans laquelle l'insertion et la suppression sont effectuées à une extrémité (appelée en haut). Le dernier élément inséré est le premier à être supprimé. Last In First Out (**LIFO**) ou First In Last Out (**FILO**).

# Qu'est-ce qu'une Pile (Stack)?

- ***empiler*** ou ***push***: insérer un élément dans une pile
- ***depiler*** ou ***pop***: supprimer l'élément supérieur d'une pile

Push ***A***
Push ***B***
Push ***C***
Push ***D***
Pop
Push ***E***

# Recherche en profondeur d'abord [DFS]

- Dans l'algorithme DFS, nous rencontrons les types d'arêtes suivants.

| |
|---|
| **Arête de l'arbre** ou **Tree edge**: rencontrer un nouveau sommet |
| **Arête arrière** ou **Back edge**: de descendant à ancêtre |
| **Arête avant** ou **Forward edge**: de l'ancêtre au descendant |
| **Arête transversale** ou **Cross edge**: entre un arbre ou des sous-arbres |

- Pour la plupart des algorithmes de classification booléenne, non visité / visité suffit:

```
faux ou false  ⟶  sommet non visité

vrai ou true   ⟶  sommet visité
```

# Recherche en profondeur d'abord [DFS] Algorithme

1. Créez une pile vide.
2. Créez la liste des sommets visités, définissez la racine comme visitée et ajoutez la racine à la pile.
3. Boucle sur la pile tant qu'il y a des éléments dans la pile
    1. Si aucun élément dans la pile, l'algorithme se termine
    2. S'il y a un élément supérieur:
        - retirez-le de la pile;
        - Traitez ce sommet (imprimer,….);
        - Marquez ses sommets adjacents comme visités et ajoutez-les à la pile;
        - Continuez à l'étape 3.

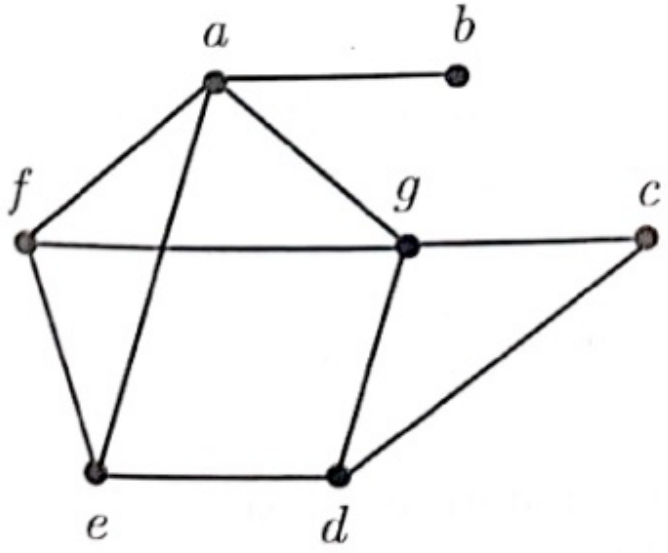
# Recherche en profondeur d'abord [DFS]

- Au départ, tous les sommets sont marqués comme non visités (faux). L'algorithme DFS commence à un sommet *u* dans le graphe. En partant du sommet *u,* il considère les arêtes de *u* vers les autres sommets.
- Si l'arête mène à un sommet déjà visité, revenez au sommet actuel *u* . Si une arête mène à un sommet non visité, accédez à ce sommet et commencez le traitement à partir de ce sommet. Cela signifie que le nouveau sommet devient le sommet actuel. Suivez ce processus jusqu'à ce que nous atteignions l'impasse. À ce stade, commencez à *revenir en arrière.*
- Le processus se termine lorsque le retour en arrière ramène au sommet de départ.

# Recherche en profondeur d'abord [DFS]

**Exemple**

Illustrez DFS pour le graphe suivant. Utilisez le sommet $a$ comme racine. Dessinez les arbres couvrants résultants.



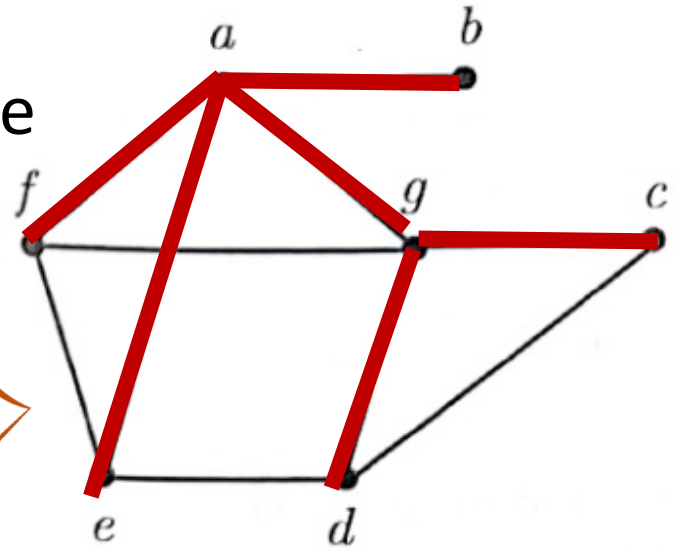

| $v$ | $N(v)$ |
|---|---|
| $a$ | $b, e, f, g$ |
| $b$ | $a$ |
| $c$ | $d, g$ |
| $d$ | $c, e, g$ |
| $e$ | $a, d, f$ |
| $f$ | $a, e, g$ |
| $g$ | $a, c, d, f$ |

# DFS

**Exemple**

| |
|---|
| $d$ |
| $g$ |
| $f$ |
| $e$ |
| $b$ |

$ab$ est une arête d'arbre
$ag$ est une arête bidirectionelle
$gf$ est une arête arrière:
produit un cycle !




ordre DFS

les arêtes de l'arbre couvrant: arête d'arbre

Visited list

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ |

| $v$ | $N(v)$ |
|---|---|
| $a$ | $b, e, f, g$ |
| $b$ | $a$ |
| $c$ | $d, g$ |
| $d$ | $c, e, g$ |
| $e$ | $a, d, f$ |
| $f$ | $a, e, g$ |
| $g$ | $a, c, d, f$ |

# Recherche en profondeur d'abord [DFS]

- On voit que parfois une arête mène à un sommet déjà découvert.
- Ces arêtes sont appelées **arêtes arrière** , et les autres arêtes sont appelées **arêtes d'arbre** car la suppression des arêtes arrière du graphe génère un arbre.

# Applications de DFS

- Tri topologique
- Recherche de composantes connexes
- Recherche de points d'articulation du graphe
- Trouver les composantes fortement connexes
- Résoudre des énigmes telles que des labyrinthes

# BFS et DFS

- BFS et DFS sont utiles dans de nombreux problèmes de théorie des graphes.
- En général, BFS produit des arbres couvrant plus compacts et de plus petit diamètre que ceux produits par DFS.
- Ainsi, DFS est souvent plus utile dans les problèmes où l'on essaie de trouver de longs chemins émanant d'un sommet donné.
- BFS peut facilement être adapté pour aider à trouver les excentricités, le rayon, le diamètre, le centre et la périphérie d'un graphe, entre autres éléments.