Graphes I

CHAPITRE 9(WEISS)

NOTES DE COURS, É. BAUDRY

NOTES DE COURS, S. HAMEL

Graphes

- Structure de données non linéaire.
- Représentation de relations entre des paires d'objets
- Applications
  - Cartes

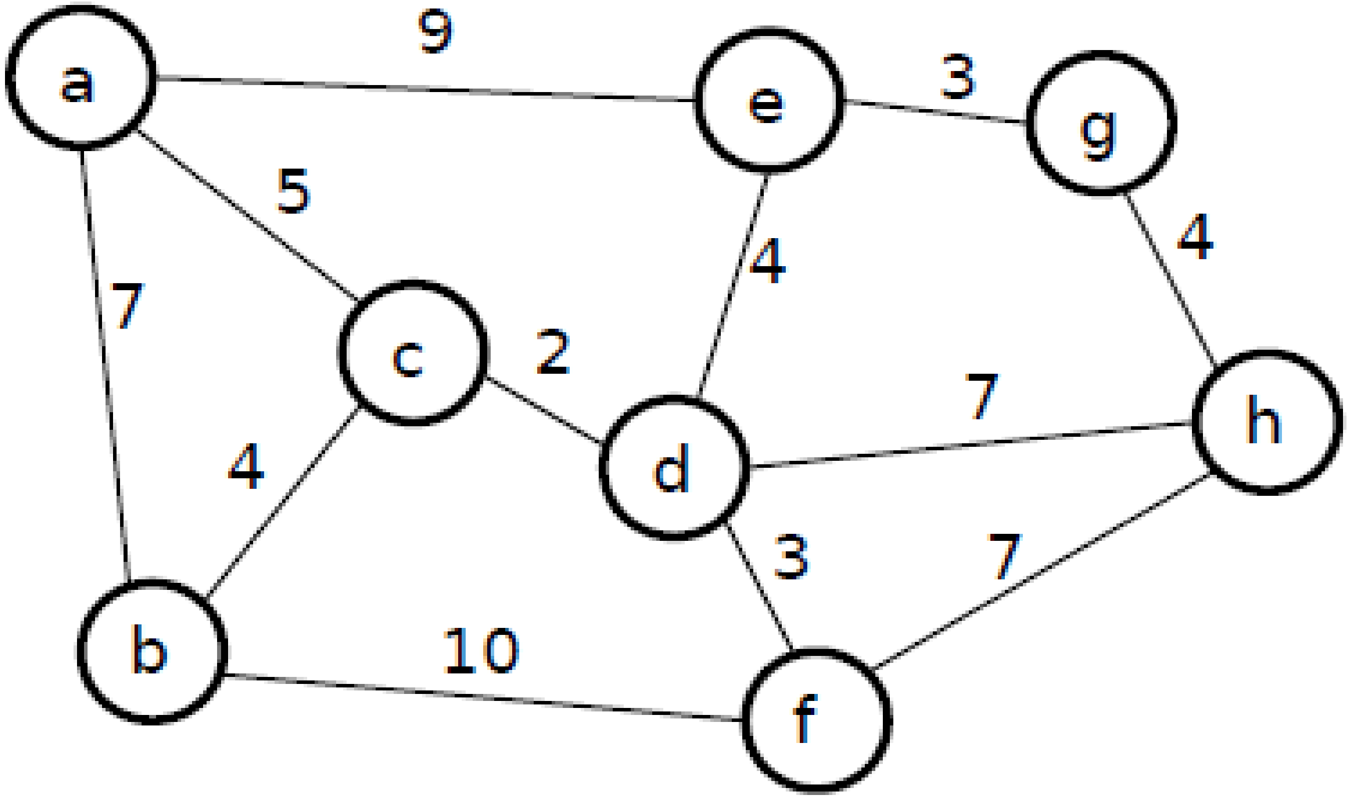
Définition formelle d’un graphe

- Un **graphe** est représenté formellement par G = (V;E) où V est un ensemble de **sommets** **(noeuds)** et E un ensemble d’**arêtes** **(arcs)**
- Un **sommet** est une représentation abstraite d’un objet
- Une **arête** représente une **relation** binaire entre deux objets

Définition formelle d’un graphe

G = (V;E)
V = {a, b, c, d, e, f, g, h}

E = { (a,b), (a,c), (a,e), (c,b), (c,d), (e,d), (b,f), (f,h), (e,g), (d,h), (d,f), (g,h) }

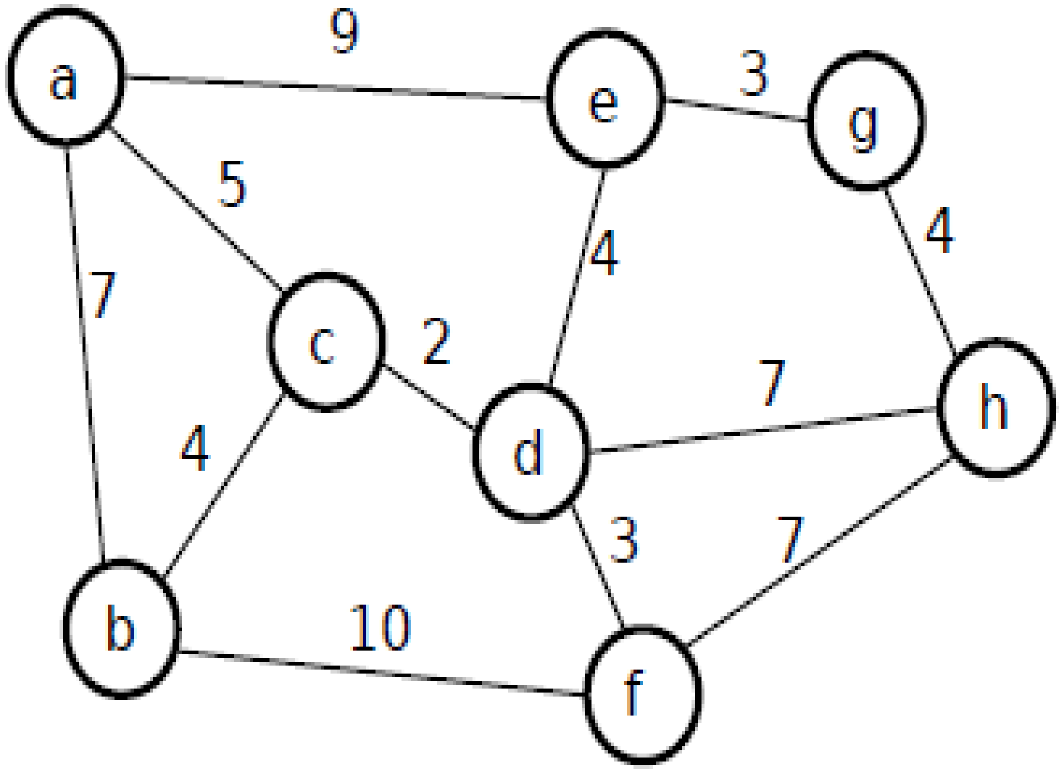


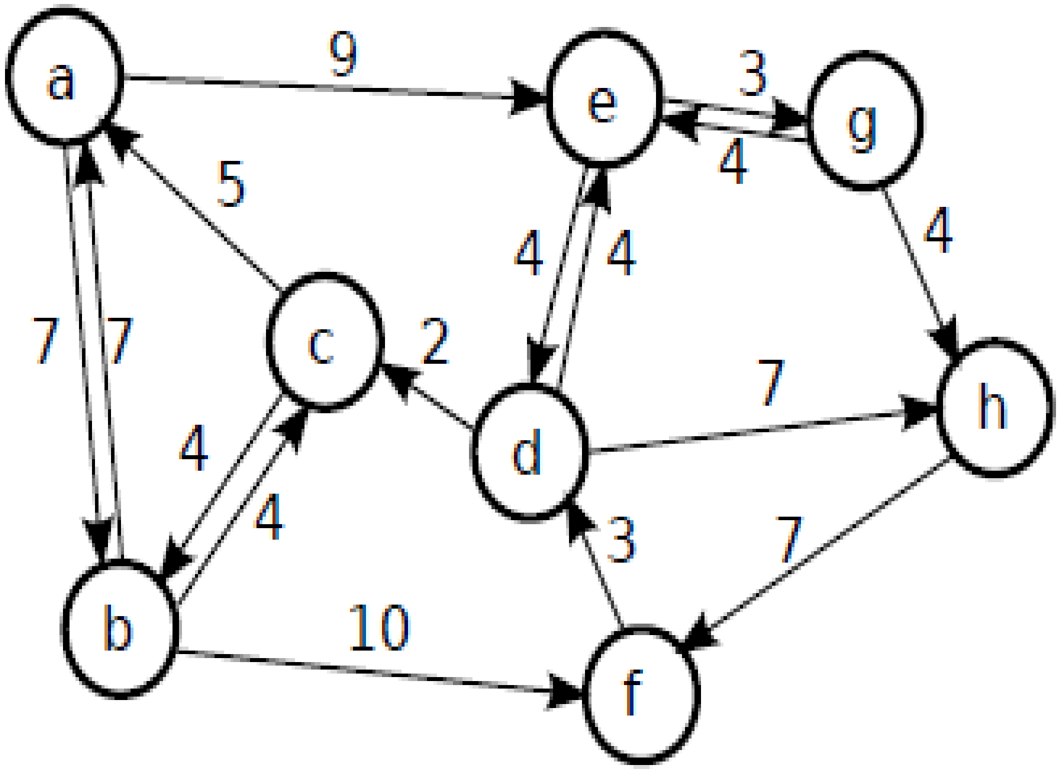
Sous graphe

Le graphe $G' = (V',E')$ est un **sous-graphe** de $G = (V,E)$ ssi $V' \subset V$, $E' \subset E$ et $\forall e = (x,y) \in E'$ $x \in V'$, $y \in V'$

Graphes

Orienté versus non orienté



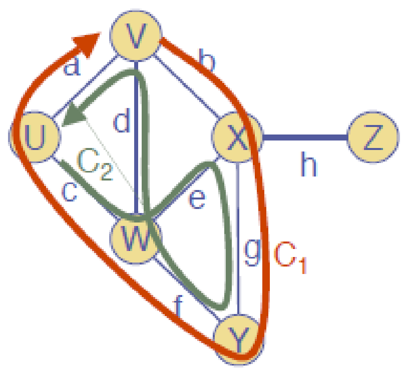


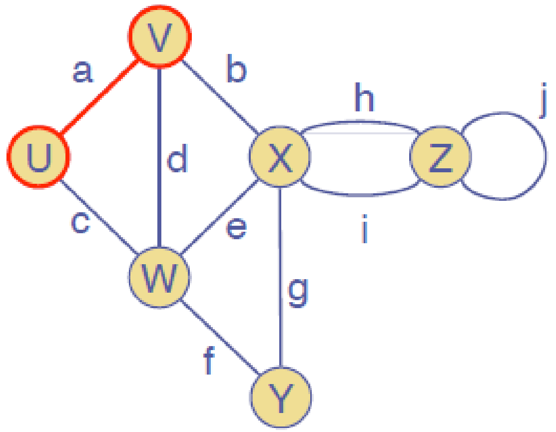
Chemin

- Un **chemin** est une séquence de sommets / arêtes dans un graphe
- La **longueur** d’un chemin peut être le nombre de sommets ou le nombre d’arêtes

Cycle

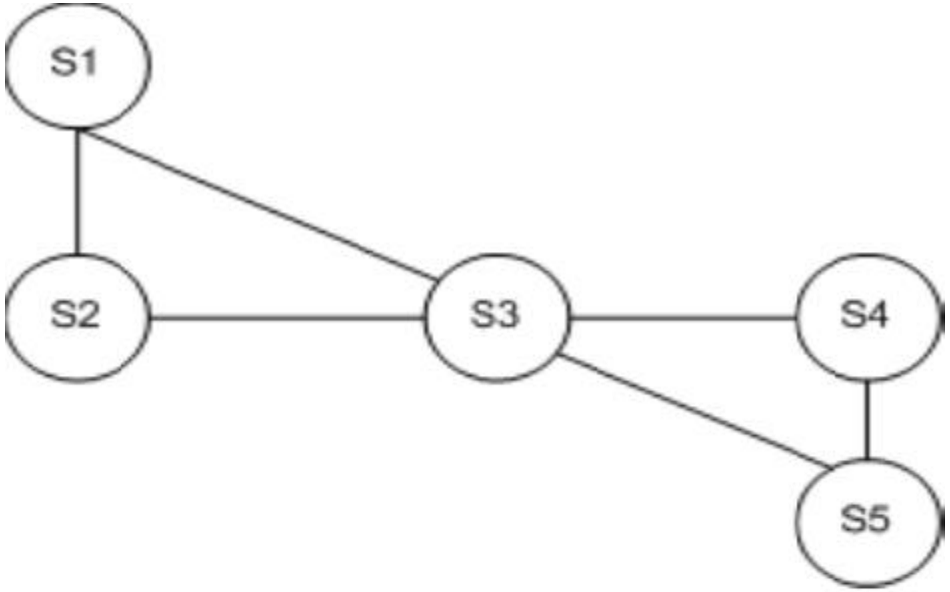
- Un **cycle** est un chemin dans un graphe dont le sommet de départ est le même que le sommet d'arrivé
- Une boucle est un cycle de longueur 1. Une **boucle** est une arête dont les sommets de départ et d'arrivée sont les mêmes.
- Un graphe est dit **acyclique** si et seulement s'il ne contient aucun cycle





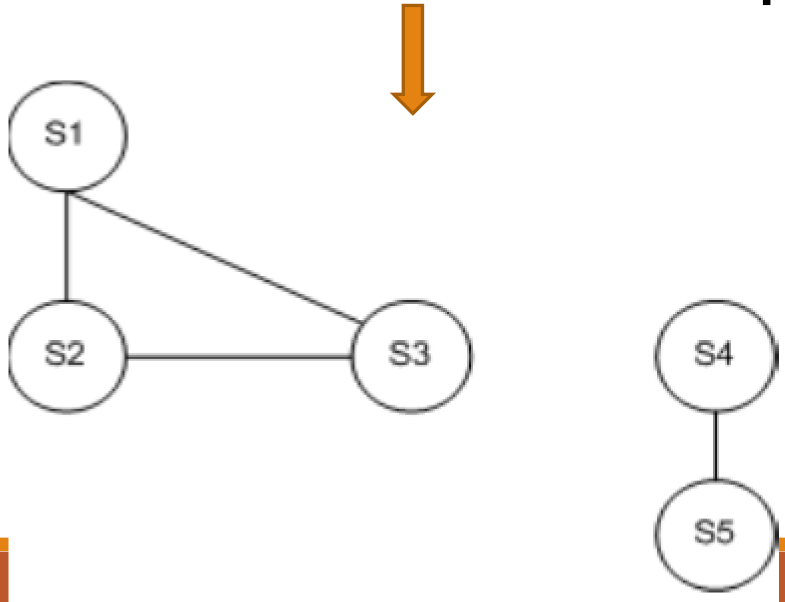
Graphes connexes

- Un graphe non orienté est **connexe** (ou **connecté**) s'il existe au moins un chemin entre toutes les paires de sommets
- Un graphe non orienté où on peut aller de tout sommet vers tous les autres sommets



Graphes connexes

- Une **composante connexe** est un sous-graphe connecté et maximal
- Un graphe G = (V,E) non connecté est composé de plusieurs composantes connexes
  - Graphe **non connexe** avec deux composantes connexes

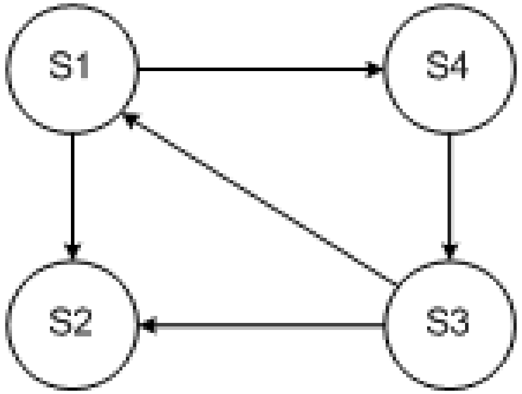


Graphes fortement connexes

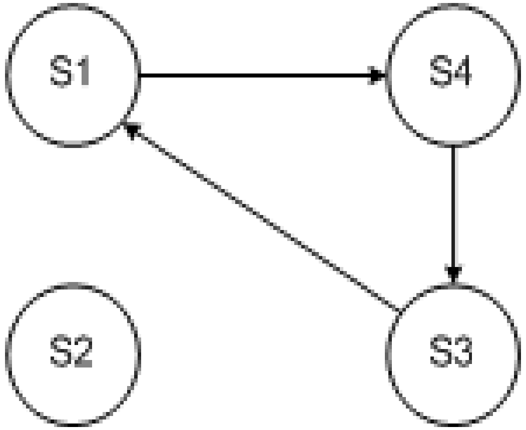
- Un graphe orienté est **fortement connexe** (ou **fortement connecté**) si et seulement si pour toute paire de sommets (a, b) il existe un chemin de a à b, et de b à a
- Un graphe orienté où on peut aller de tout sommet vers tous les autres sommets en passant éventuellement par un ou plusieurs sommets intermédiaires

Graphes fortement connexes

Graphe G non fortement connexe
S2 –>S3 ; S2 ->S1 ?

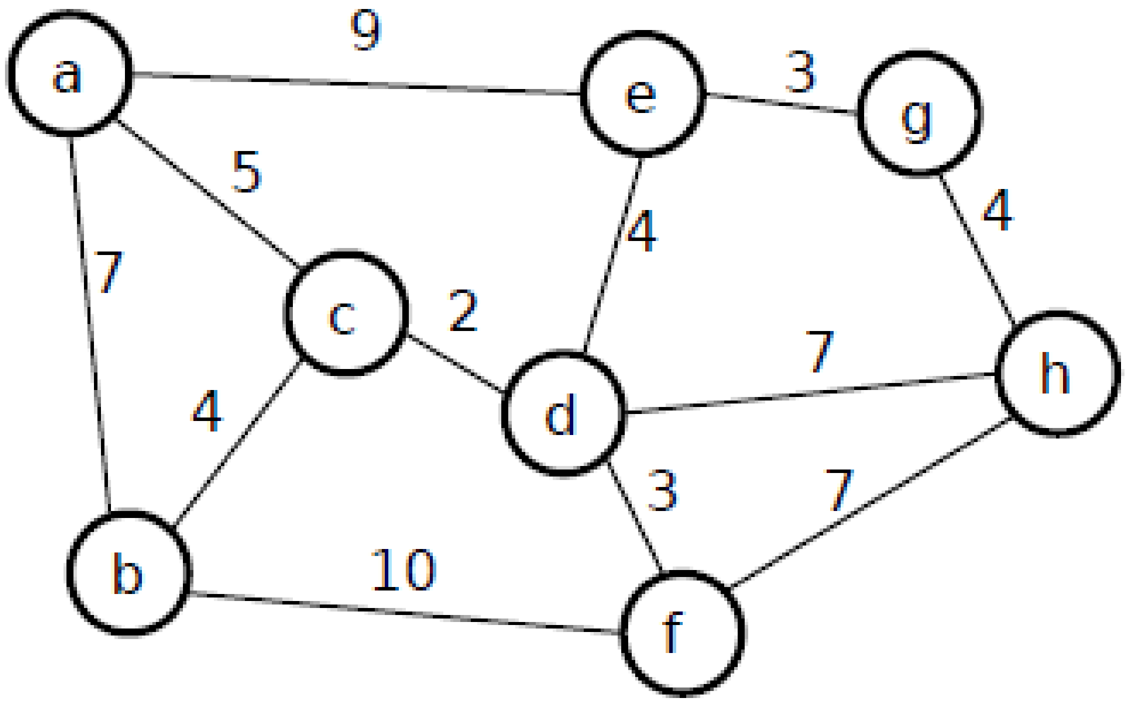


Composantes fortement connexes du graphe G



Étiquette

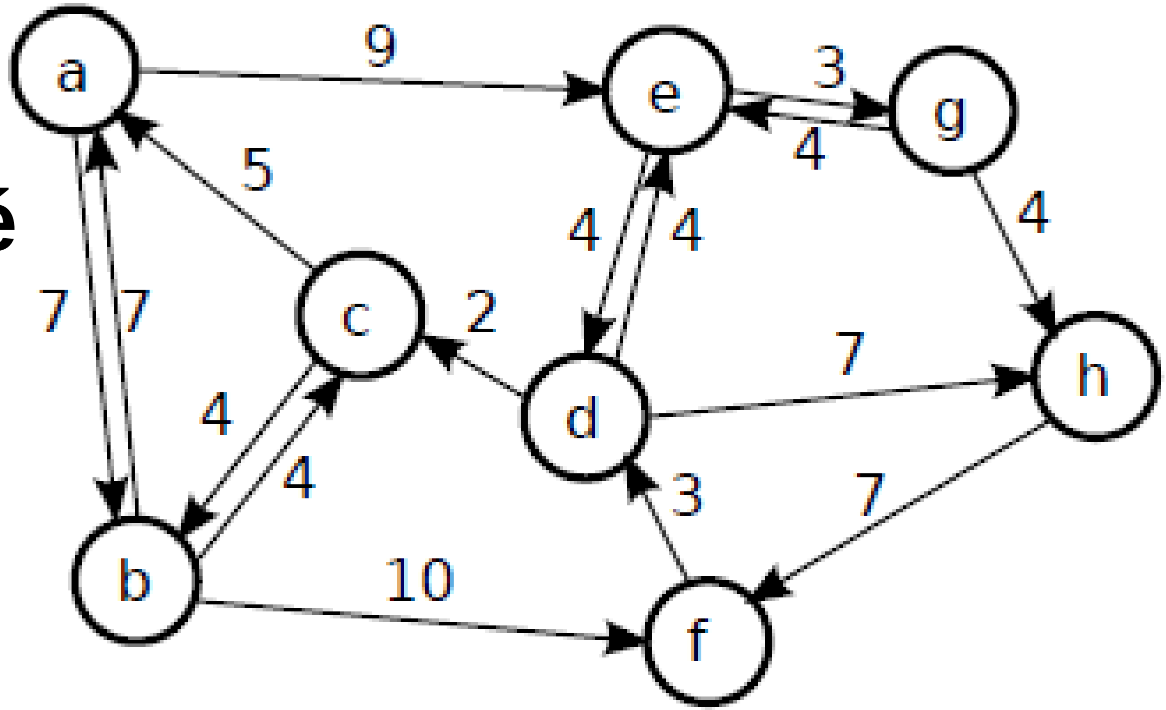
- Une **étiquette** indique une propriété d’une arête
- Poids d’une arête



Étiquette

- Dans un graphe non orienté, le **degré** d'un sommet v ∈ V, noté *deg(v)*, est le nombre d'arêtes qui y sont reliées

- Dans un graphe orienté, on fait la distinction entre le **degré sortant** et le **degré entrant**, qui sont respectivement notés *deg_out (v)* et *deg_in(v)*



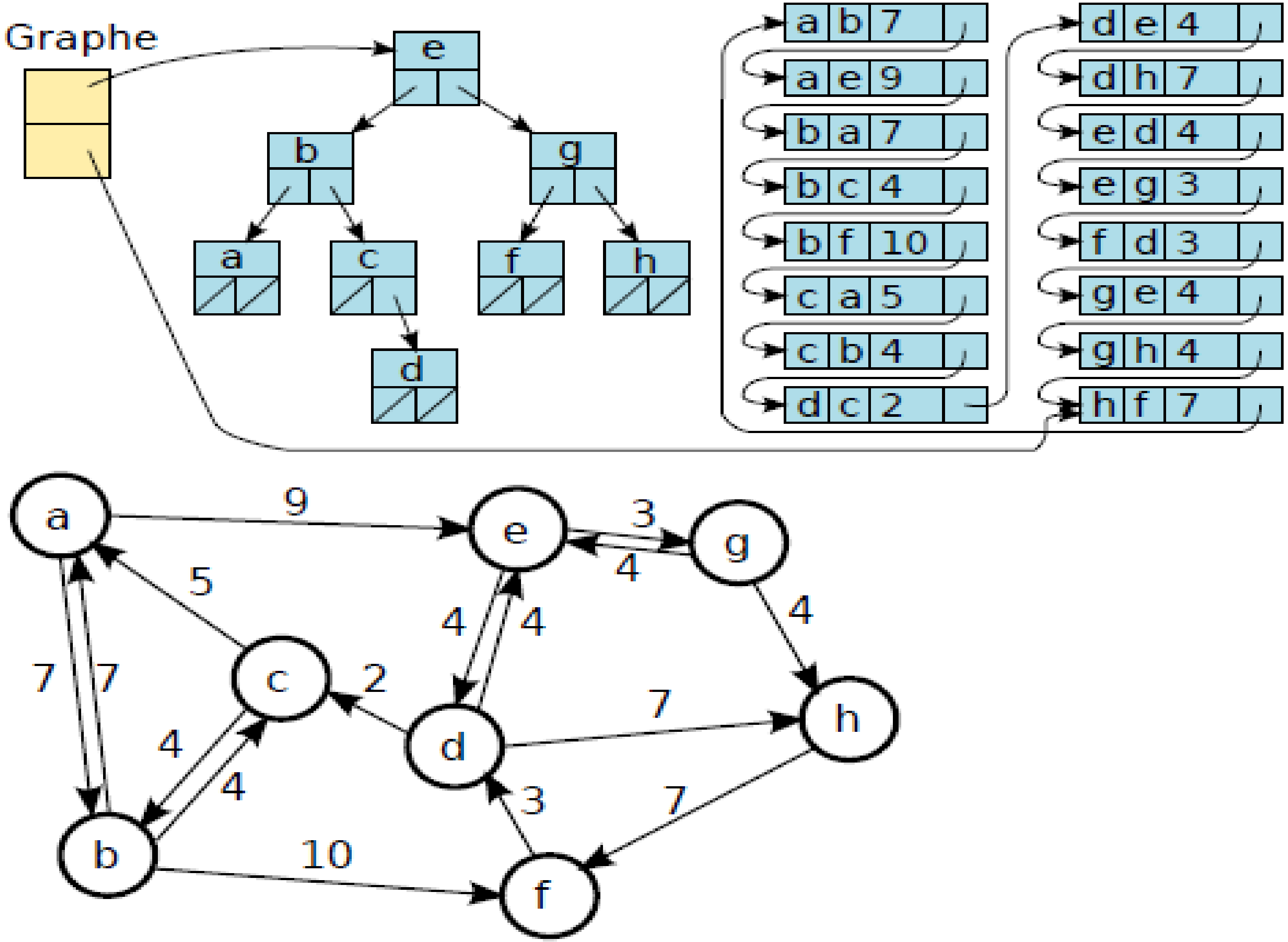
Arbre, Forêt

- Un **arbre** est un cas particulier de graphe non orienté
- Un arbre est un graphe ayant une seule composante connexe tel que le nombre de sommets est égal au nombre d’arêtes plus un (|V| = |E|+1), et où tous les sommets sont accessibles à partir d’un sommet qualifié de **racine**
- Un arbre est un graphe qui est forcément acyclique
- Un graphe qui est composé d’un ensemble d’arbres est appelé une **forêt**

Représentations

Ensemble de sommets et collection d'arêtes

```
class Graphe {
class Arete{
  S depart, arrivee;
  A etiquette;
};
Ensemble<Sommet> sommets;
Collection<Arete> aretes;
//...
};
```



Représentations

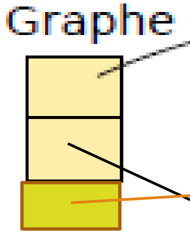
- Deux façons classiques de représenter un graphe $G = (V, E)$
  - Ensemble de listes d’adjacences
    - Graphes peu denses -> $|E| << |V|^2$
  - Matrice d’adjacences
    - Graphe est dense -> $|E| \approx |V|^2$

Représentations

Ensemble de sommets et collection d’arêtes

Matrice d’adjacence (1)

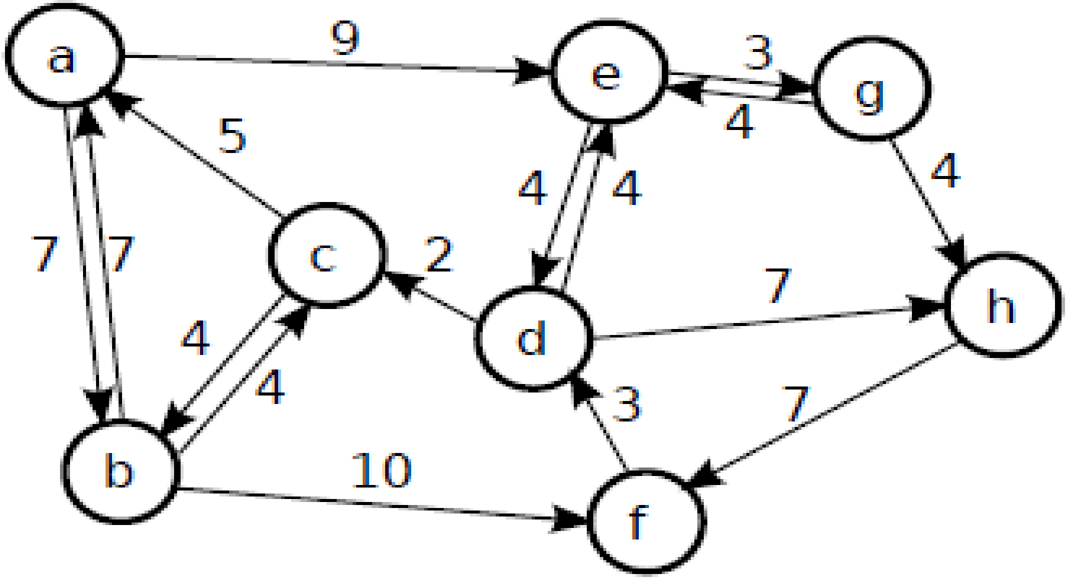
```
class Graphe {
Tableau<S> sommets;
Tableau2D<bool> relations;
Tableau2D<A> etiquettes;
};
```



a
b
c
d
e
f
g
h

f	t	f	f	t	f	f	f
t	f	t	f	f	t	f	f
t	t	f	f	f	f	f	f
f	f	t	f	t	f	f	t
f	f	f	t	f	f	t	f
f	f	f	t	f	f	f	f
f	f	f	f	t	f	f	t
f	f	f	f	f	f	t	f

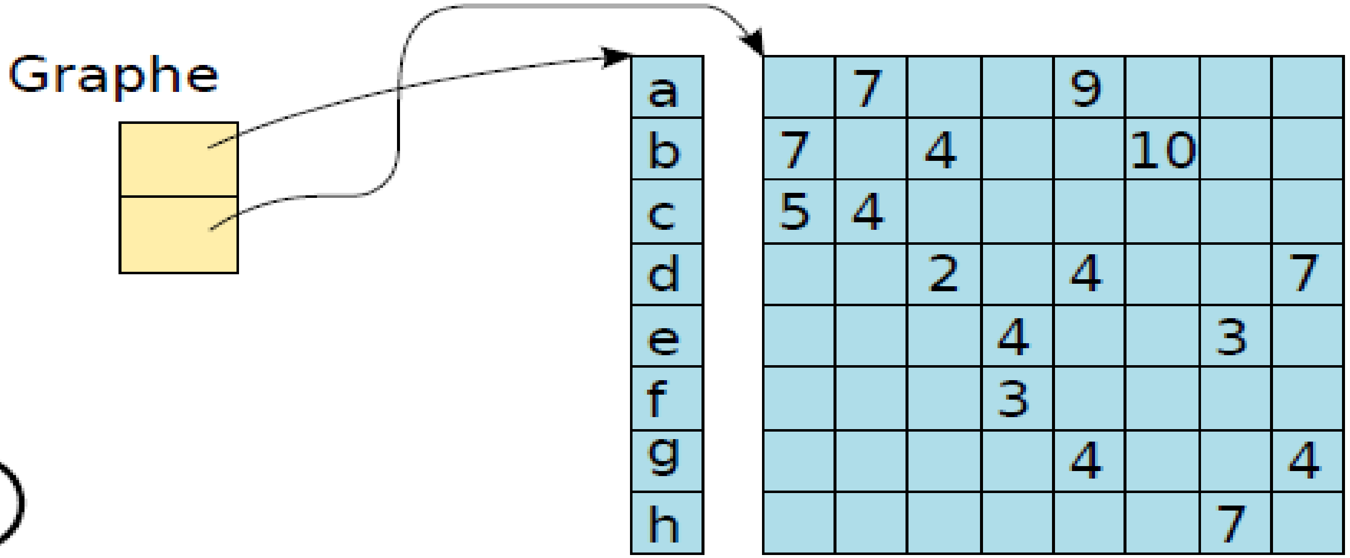
	7			9			
7		4			10		
5	4						
		2		4			7
			4			3	
			3				
				4			4
						7	



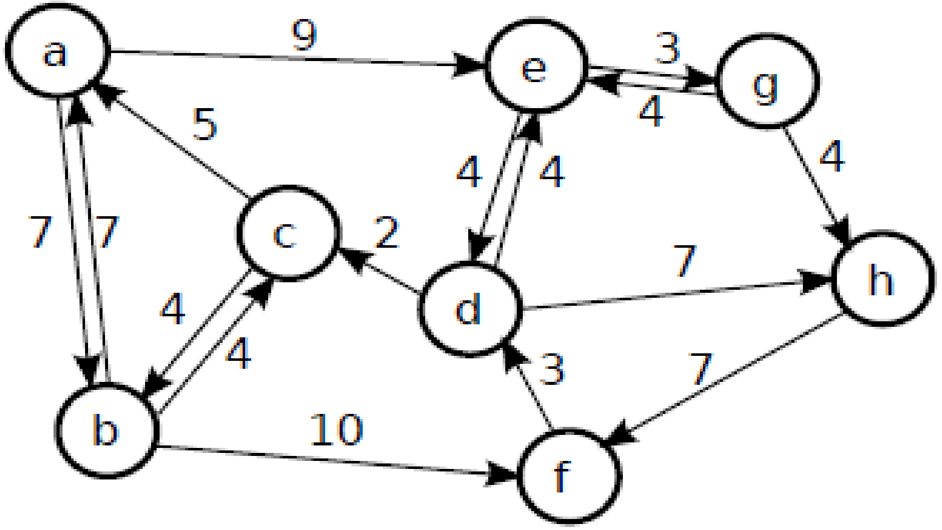
Représentations

Ensemble de sommets et collection d’arêtes

Matrice d’adjacence (2)
Mémoire – $\Theta(|V|^2)$



a		7			9			
b	7		4			10		
c	5	4						
d			2		4			7
e				4			3	
f				3				
g					4			4
h							7	



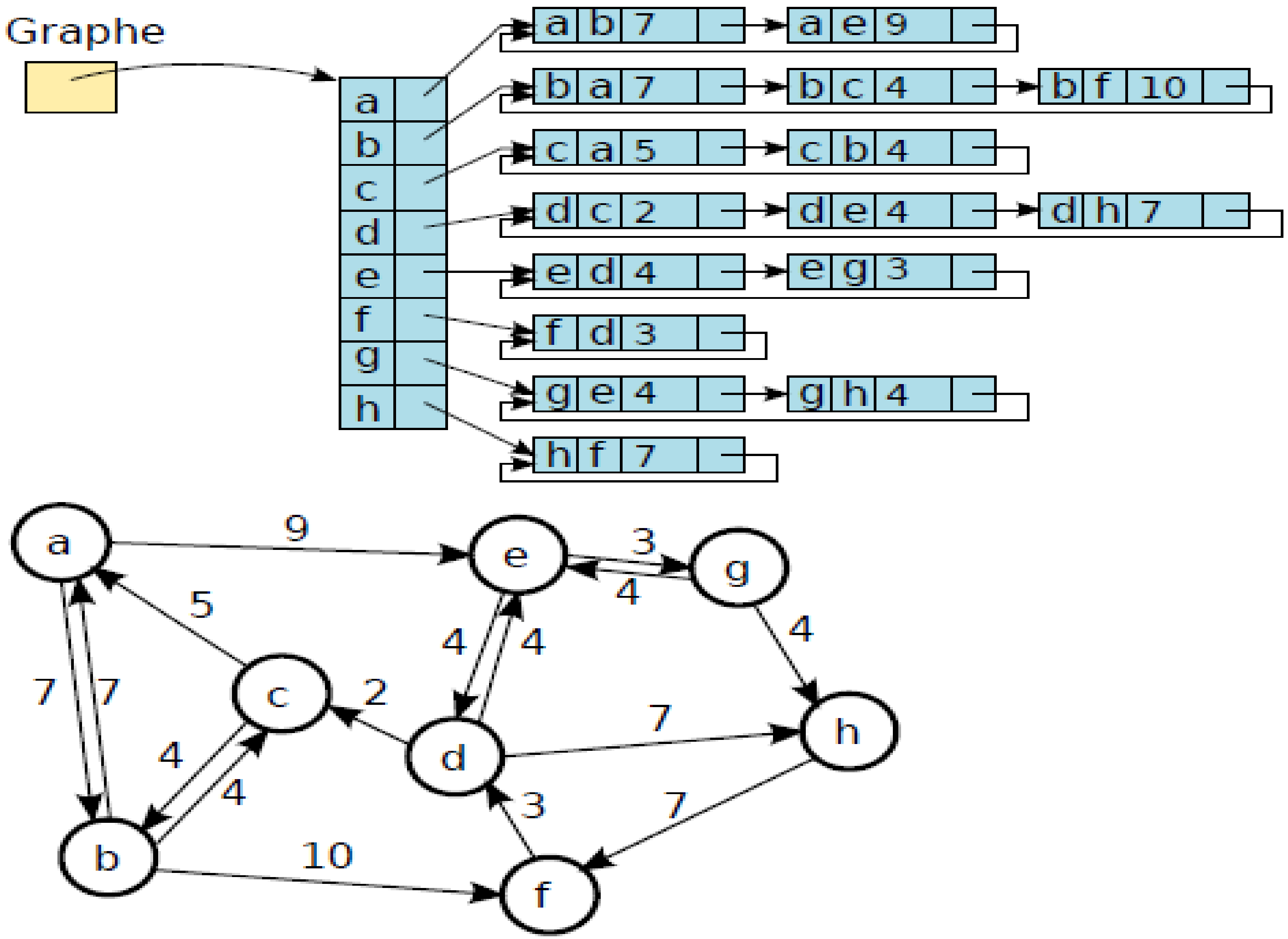

```
class Graphe {
   Tableau<S> sommets;
   Tableau2D<A> etiquettes;
   static A  AUCUNE_RELATION;
}
```

Représentations

Listes d’adjacence

Ensemble de sommets et collection d’arêtes

```
class Graphe {
class Arete{
    S depart, arrivee;
   A valeur;
};
class Sommet {
    S valeur;
    Liste<Arete>
    aretesSortantes;
}
Tableau<Sommet> sommets;
```



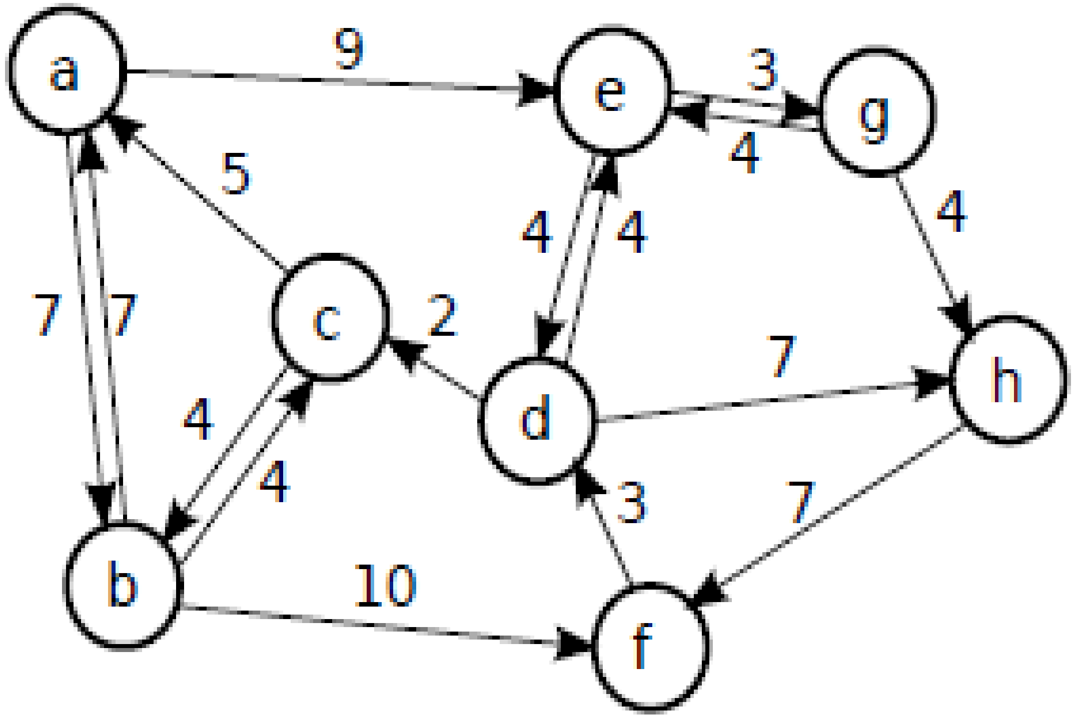
Représentations

Ensemble de sommets et collection d’arêtes

Dictionnaire avec liste d’adjacences

Exercice : représentation mémoire

```
class Graphe {
class Arete{
   S depart, arrivee;
   A valeur;
};
class Sommet {
   Liste<Arete> aretesSortantes;
};
   TreeMap<S, Sommet> sommets;
//...
}
```



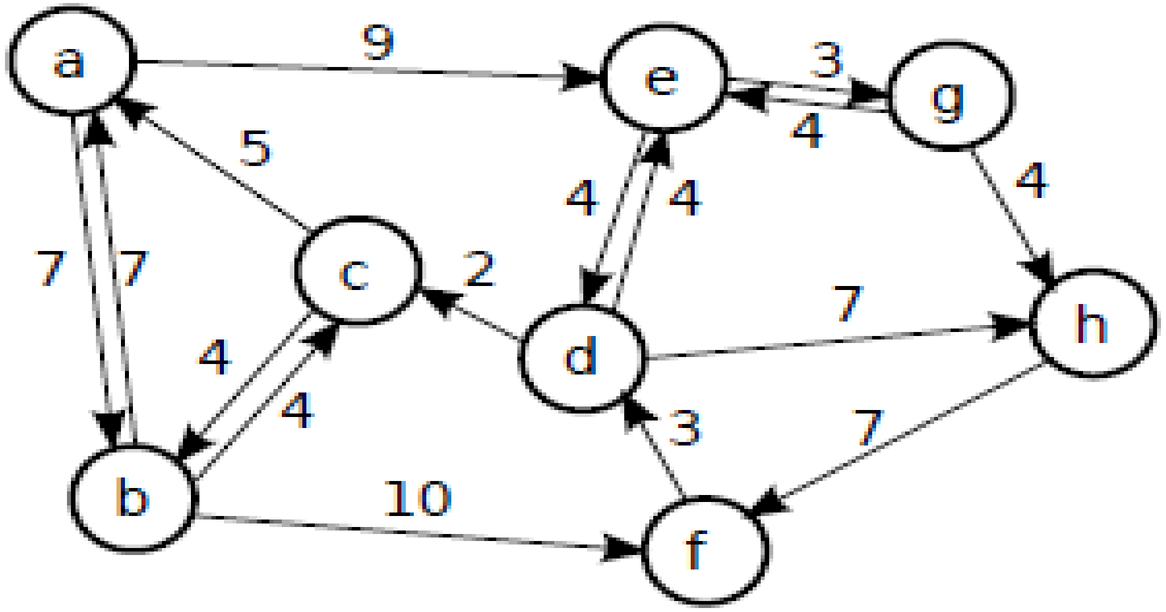
Représentations

Ensemble de sommets et collection d'arêtes

Exercice : représentation mémoire

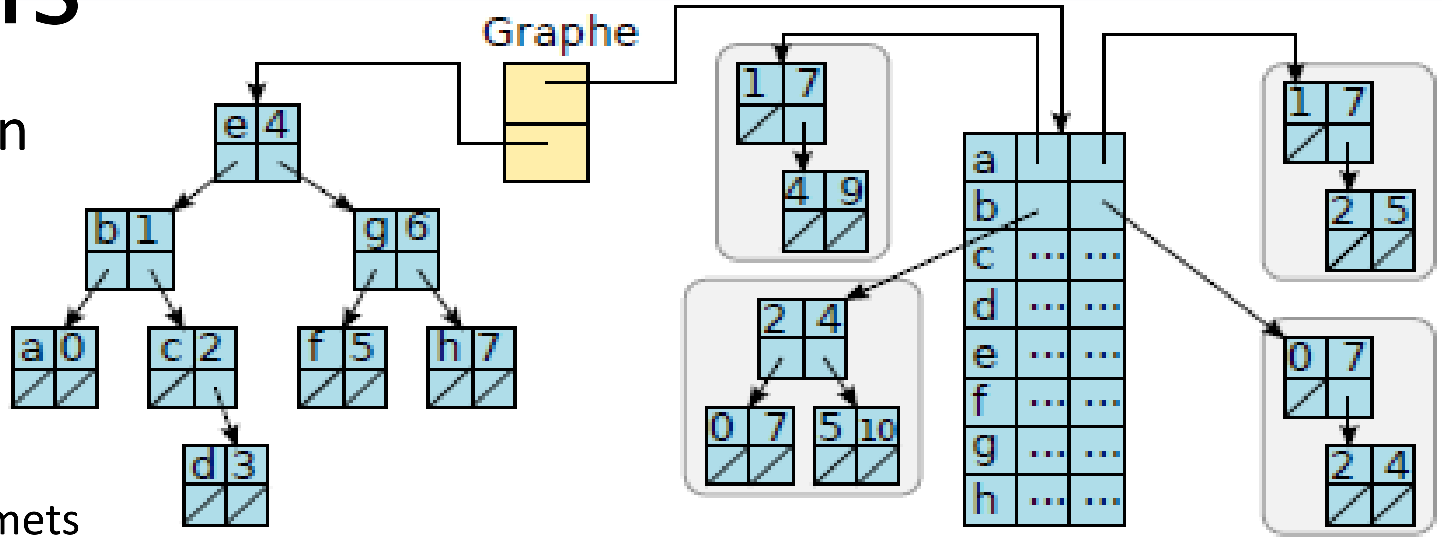
Dictionnaire de dictionnaires d'adjacences

```
class Graphe {
class Sommet {
   TreeMap<S, A> aretesSortantes;
};
   TreeMap<S, Sommet> sommets;
//...
```



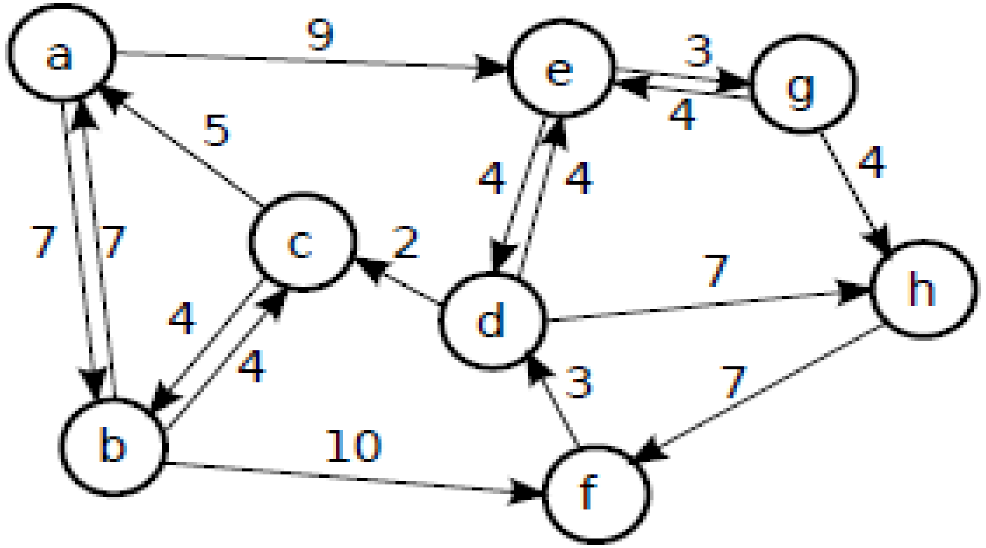
Représentations

Ensembles d'adjacence avec des indices



Ensemble de sommets et collection d'arêtes

```
class Graphe {
class Sommet{
        S s; // optionnel
      // e(v,w) sortante. s = v
      // clé – indice de w dans une table des sommets
      // valeur – poids de e sortante
      // Exemple: (1,7) = 1 => b; e = (a,b) poids 7
     TreeMap aretesSortantes;
     TreeMap aretesEntrantes; //optionel
};
//clé – sommet, valeur – indice de sommet
      TreeMap indices;
      Tableau<Sommet> sommets;
//...
}
```



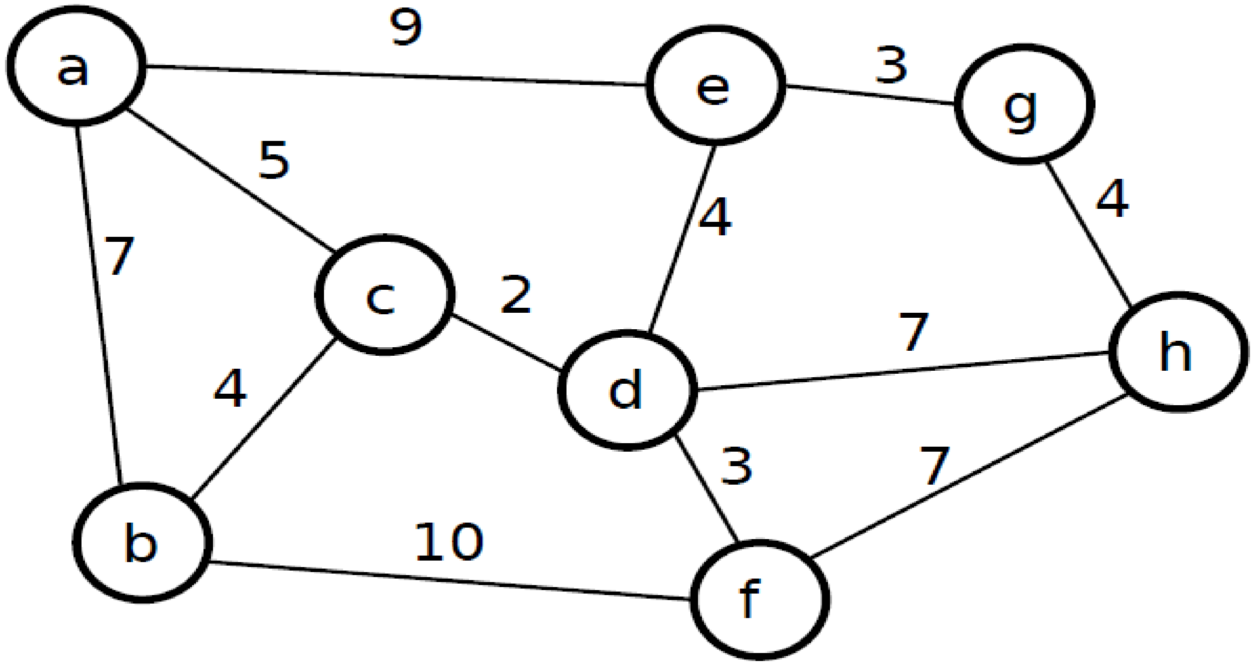
Parcours

- Parcourir un graphe revient à emprunter de façon systématique les arcs de graphe, pour en visiter les sommets
- Un algorithme de parcours permet de mettre en évidence plusieurs caractéristiques de la structure du graphe
- Types de parcours
    - Recherche en profondeur
    - Recherche en largeur

Parcours

Recherche en profondeur

1. RECHERCHEPROFONDEUR(G = (V,E), v ∈ V)
2. v.visité ← vrai
3. pour toute arête e ∈ v.aretesSortantes()
4. w ← e.arrivee
5. si !w.visité
6. RechercheProfondeur(G, w)



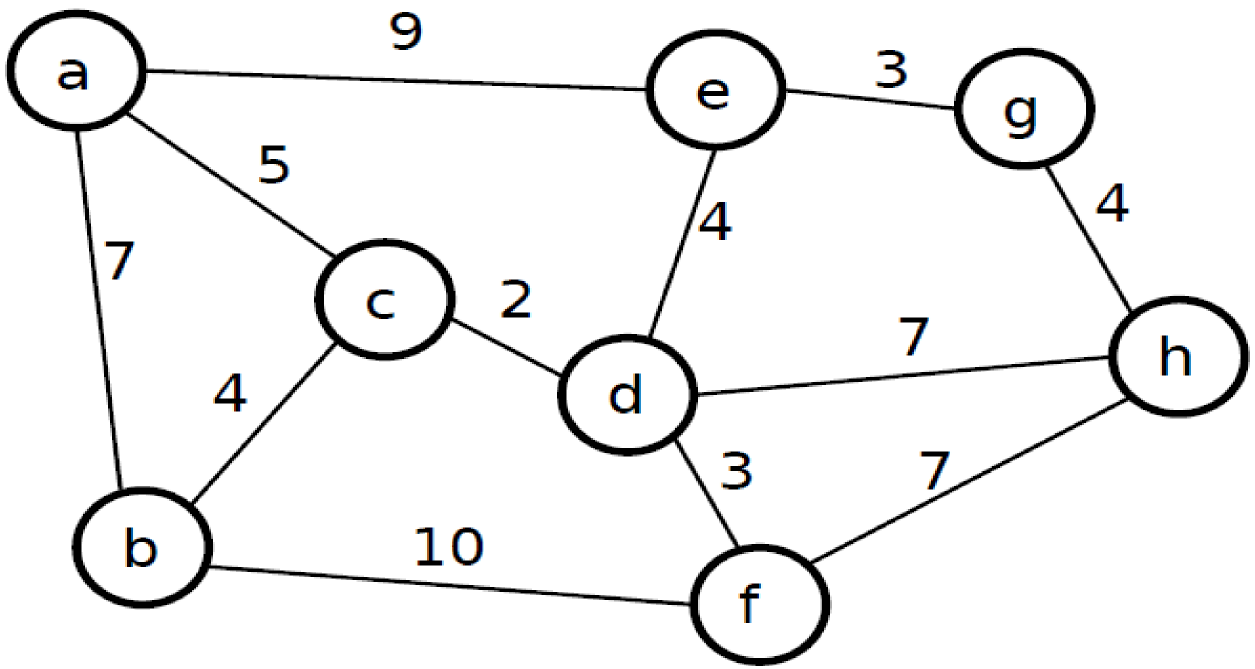
Parcours en profondeur

- Prends un temps en $O(n + m)$ pour un graphe avec n sommets et m arêtes
- Peut être modifié pour résoudre d'autres problèmes sur les graphes
  - Trouver et retourner un chemin entre deux sommets donnés
  - Trouver un cycle dans le graphe

Parcours

Recherche en largeur

1. RECHERCHELARGEUR(G = (V,E), v ∈ V)
2. file CRÉER_FILE
3. v.visité ← vrai
4. file.ENFILER(v)
5. tant que !file.vide()
6.   s ← file.defiler()
7.   pour tout arête a = (s,s′) ∈ E
8.   si !s′.visité
9.   s′.visité ← vrai
10.   file.ENFILER(s′)



Parcours en largeur

- Prends temps $O(n + m)$ sur un graphe avec n sommets et m arêtes

- Peut être modifié pour résoudre d'autres problèmes sur les graphes
  - Découvrir et retourner un chemin entre deux sommets donnés avec le minimum d'arêtes
  - Trouve un cycle