Graphes

INF3105 – Structures de données et algorithmes

Éric Beaudry

Université du Québec à Montréal (UQAM)

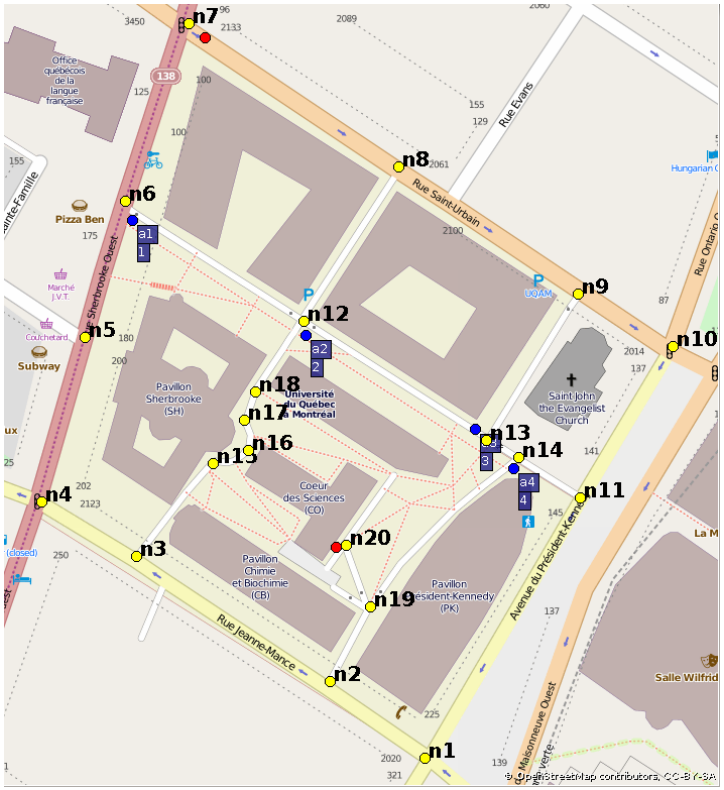
2019

Sommaire

Les Graphes

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

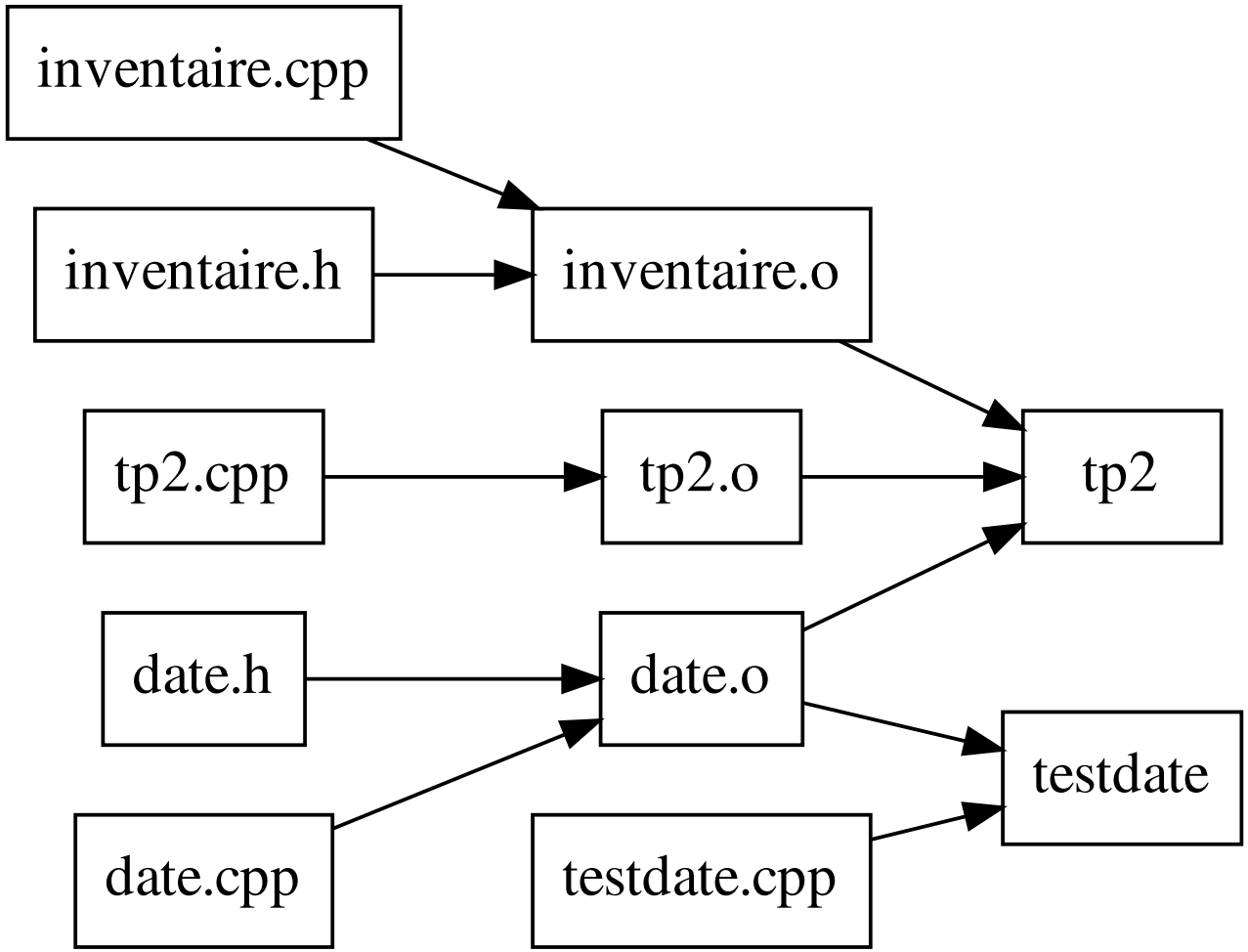
Cartes


n1
n2
n3
n4
n5
n6
n7
n8
n9
n10
n11
n12
n13
n14
n15
n16
n17
n18
n19
n20
a1
a2
a4
Rue Evans
Rue Saint-Urbain
Rue Jeanne-Mance
Avenue du Président-Kennedy
Pavillon Sherbrooke (SH)
Coeur des Sciences (CO)
Pavillon Chimie et Biochimie (CB)
Pavillon (PK)
Saint John the Evangelist Church
Pizza Ben
Subway
© OpenStreetMap contributors, CC-BY-SA

Makefile

```
# Makefile pour TP2.
tp2: tp2.o inventaire.o date.o
    g++ -o tp2 tp2.o date.o epicerie.o

tp2.o: tp2.cpp
    g++ -c -o tp2.o tp2.cpp

date.o: date.h date.cpp
    g++ -c -o date.o date.cpp

inventaire.o: inventaire.h inventaire.cpp date.h
    g++ -c -o inventaire.o inventaire.cpp

testdate : testdate.cpp date.o
    g++ -o testdate testdate.cpp date.o
```

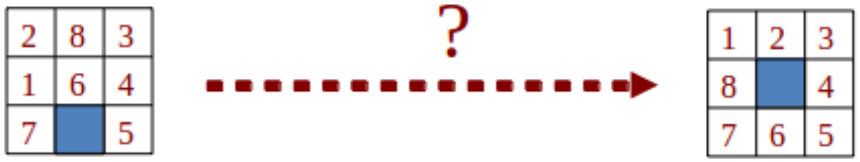
Makefile

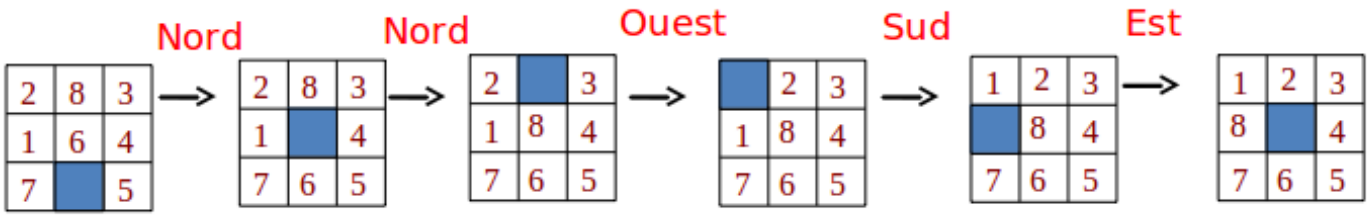


Certaines dépendances .h ne sont pas illustrées

Jeu de taquin – Espace d'états

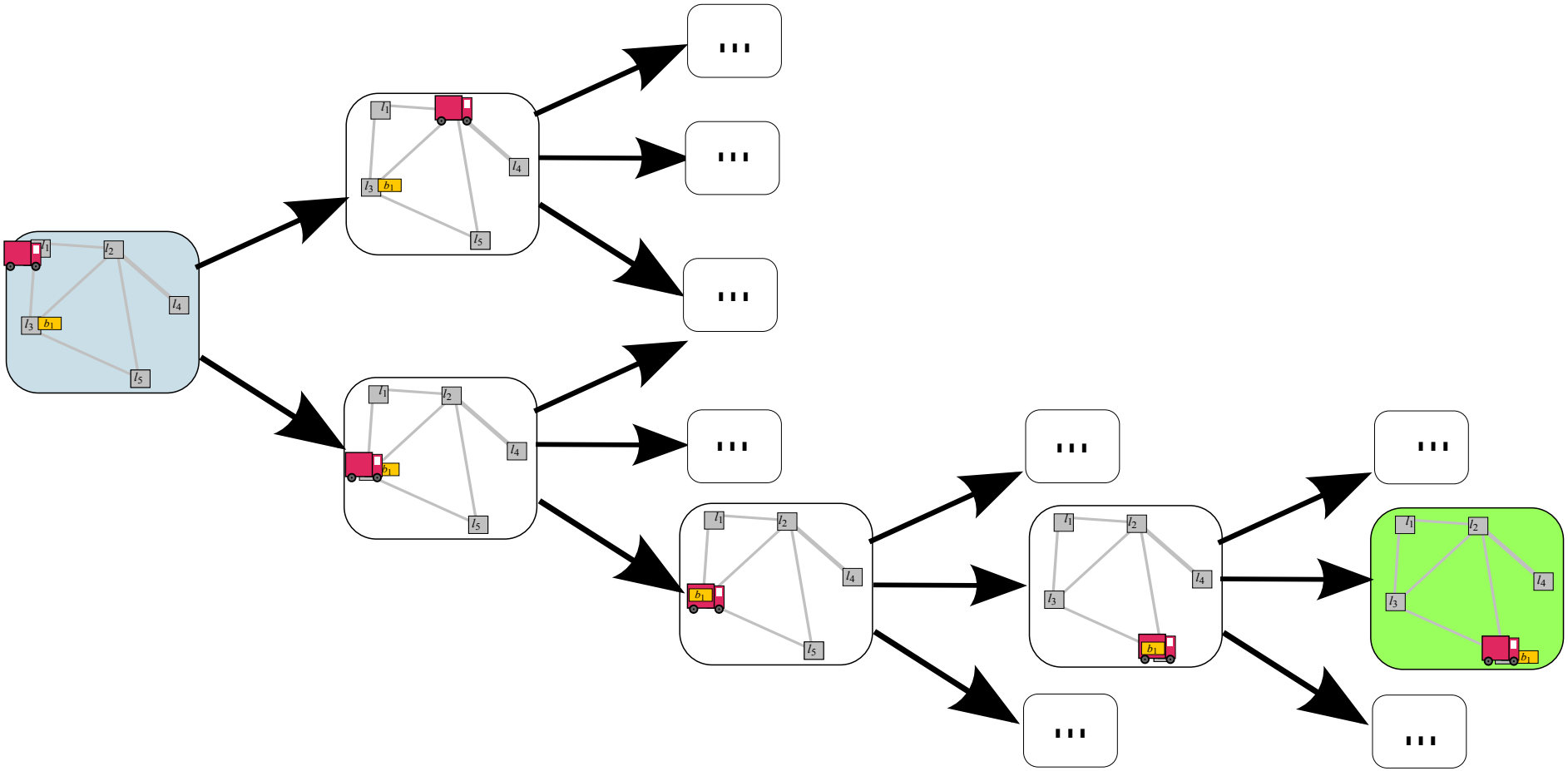
http://cpt.toutimages.com/jeu_de_taquin/





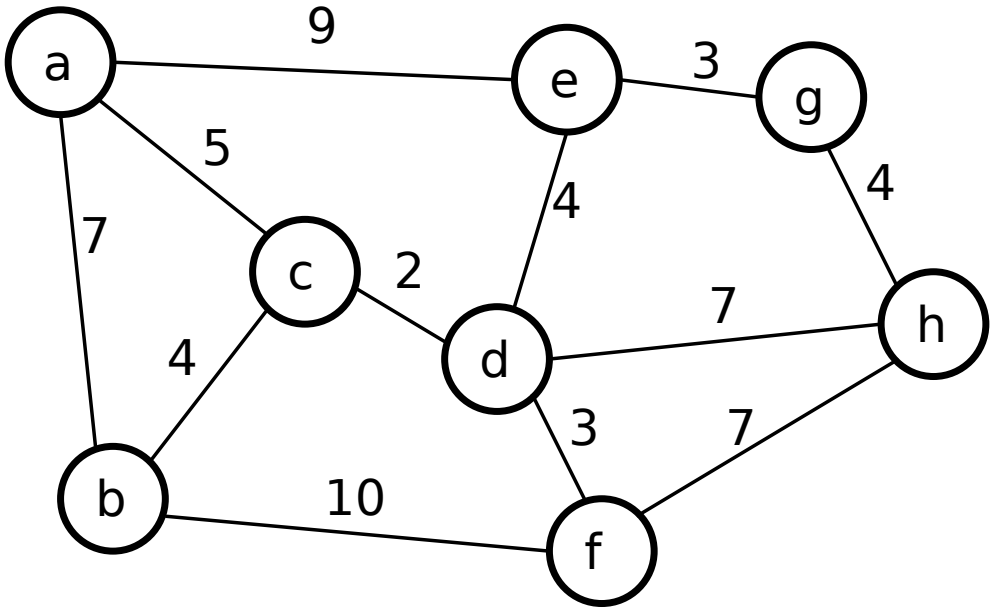
Abordé dans le cours INF4230 (Intelligence artificielle).

Recherche dans un espace d'états

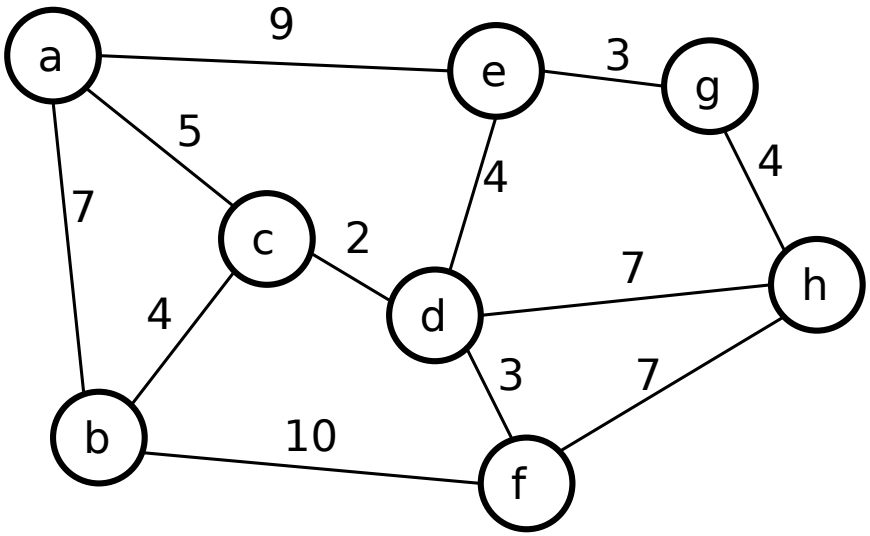


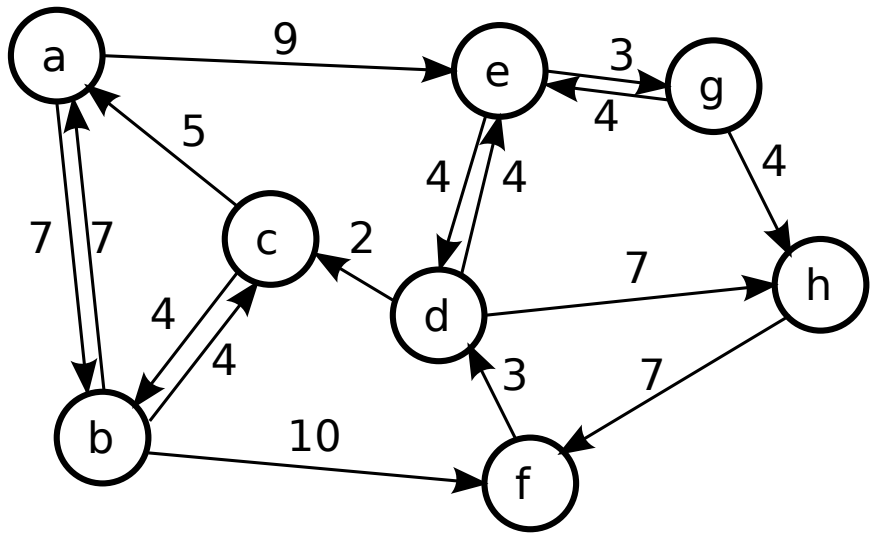
Définition formelle d'un graphe

- Un **graphe** est représenté formellement par $G = (V, E)$ où V est un ensemble de **sommets (*vertex*)** et E un ensemble d'**arêtes (*edges*)**. Les sommets et arêtes peuvent aussi être appelés respectivement des noeuds et des arcs.
- Un **sommet** est une représentation abstraite d'un objet.
- Une **arête** représente une **relation** binaire entre deux objets.



Orienté *vs* Non orienté





- Les arêtes peuvent être **orientées** ou **non orientées**.
- Une arête **orientée** indique une relation directionnelle.
- Un **graphe orienté** (*directed graph* ou *digraph* en anglais) est composé d'arêtes orientés.
- Un **graphe non orienté** (*undirected graph* en anglais) est composé d'arêtes non orientés.

Sous-Graphe

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

Chemin, Longueur

- Un **chemin** est une séquence de sommets / arêtes dans un graphe.
- La **longueur d'un chemin** peut être :
  - le nombre d'arêtes ;
  - nombre de sommets ;
  - la somme des poids des arêtes formant le chemin.

Cycle, Boucle, Graphe acyclique

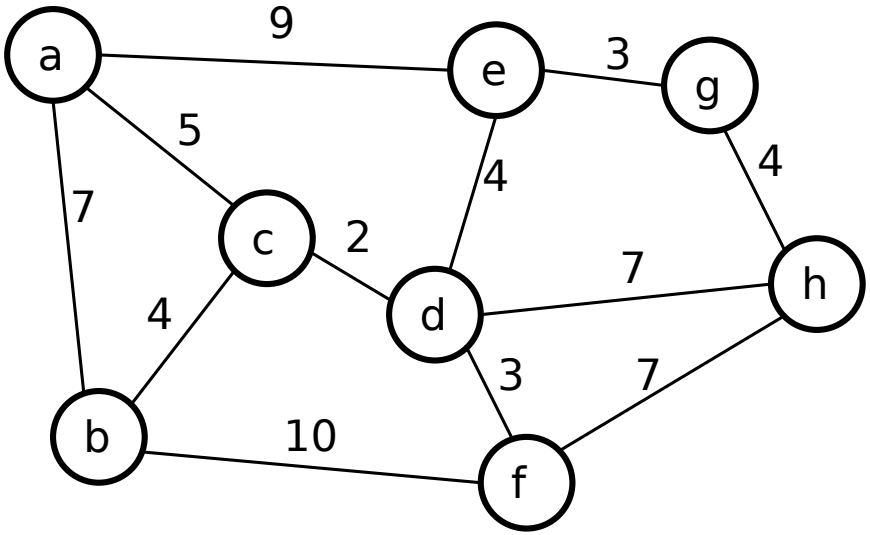
- Un **cycle** est chemin dans un graphe dont le sommet de départ est le même que le sommet d'arrivé.
- Une **boucle** est une arête dont les sommets de départ et d'arrivée sont les mêmes. Une boucle est un cycle de longueur d'une (1) arête.
- Un graphe est dit **acyclique** si et seulement si il ne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

Connexe, Compostante connexe

- Un graphe non orienté est **connexe** (ou **connecté**) s'il existe au moins un chemin entre toutes les paires de sommets.
- Un graphe $G = (V, E)$ non connecté est composé de plusieurs composantes connexes.
- Une **composante connexe** est un sous-graphe connecté et maximal.
- 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 non fortement connexe est composée de plusieurs composantes fortement connexes.

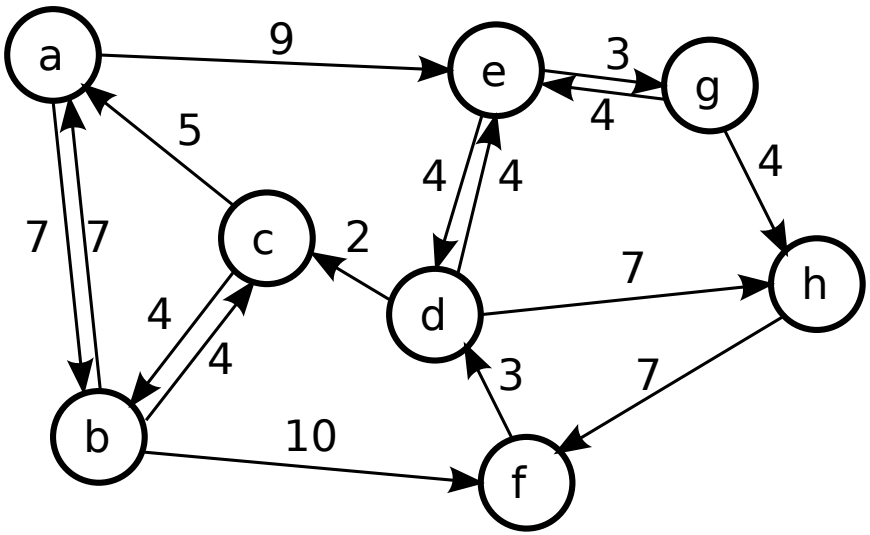
Étiquette

- Une **étiquette** sur une arête indique une propriété.
- Exemple : Le **poids** d'une arête peut indiquer le coût d'une arête.



Degré

- Dans un graphe non orienté, le **degré** d'un sommet $v \in V$, noté $deg(v)$, est le nombre d'arêtes qui y sont reliées.
- Dans un graphe orienté, on peut faire la distinction entre le **degré sortant** et le **degré entrant**, qui sont respectivement notés $deg_{out}(v)$ et $deg_{in}(v)$.



Arbre, Fôret

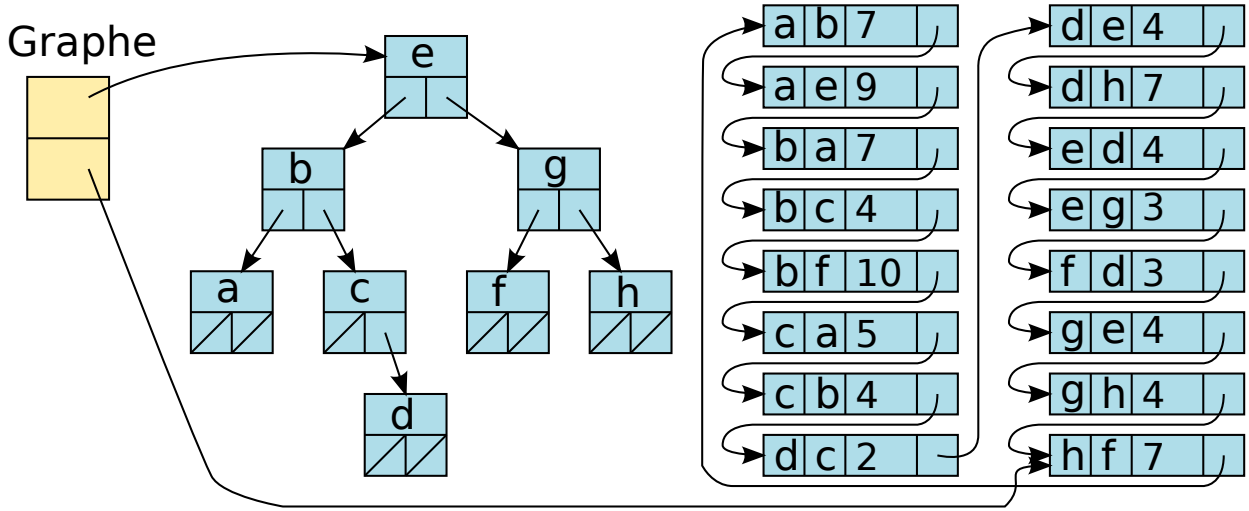
- Un **arbre** est un cas particulier de graphe.
- Un arbre est un graphe ayant une seule composante connexe tel que le nombre de sommets est égal au nombre d'arêtes plus un ($|V| = |E| + 1$), et où tous les sommets sont accessibles à partir d'un sommet qualifié de **racine**.
- Un arbre est forcément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

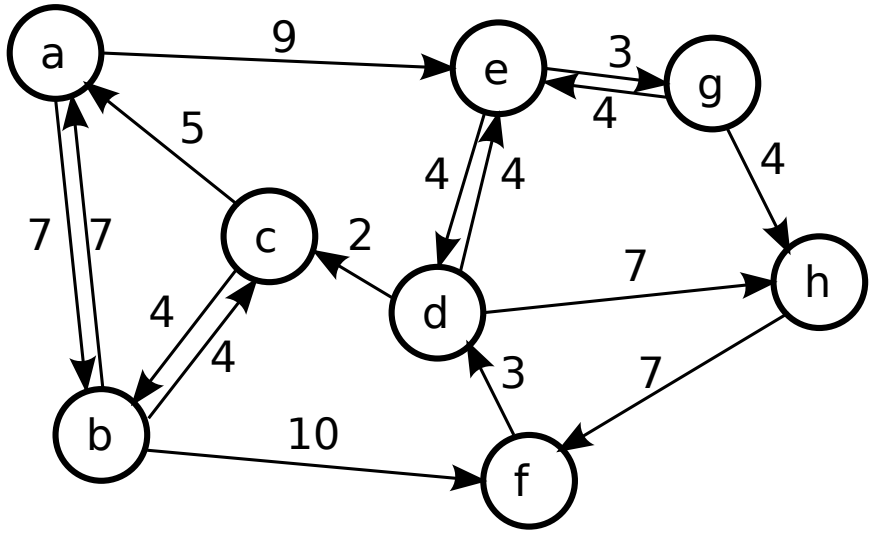
Considérations

- Complexité :
  - Complexité spatiale (mémoire).
  - Complexité temporelle des opérations :
    - Interroger l'existence d'une arête, c-à-d une relation entre une paires de sommets (et obtenir le poids).
    - Énumérer les arêtes sortantes à partir d'un (ou entrantes vers) un sommet.
    - Créer / Modifier le graphe (ajout/suppression de sommets/arêtes).
- Taille du graphe :
  - $n = |V|$: nombre de sommets.
  - $m = |E|$: nombre d'arêtes.
  - densité du graphe : m/n.
- Généralement : choix de compromis en fonction de l'application.

Ensemble de sommets et collection d'arêtes

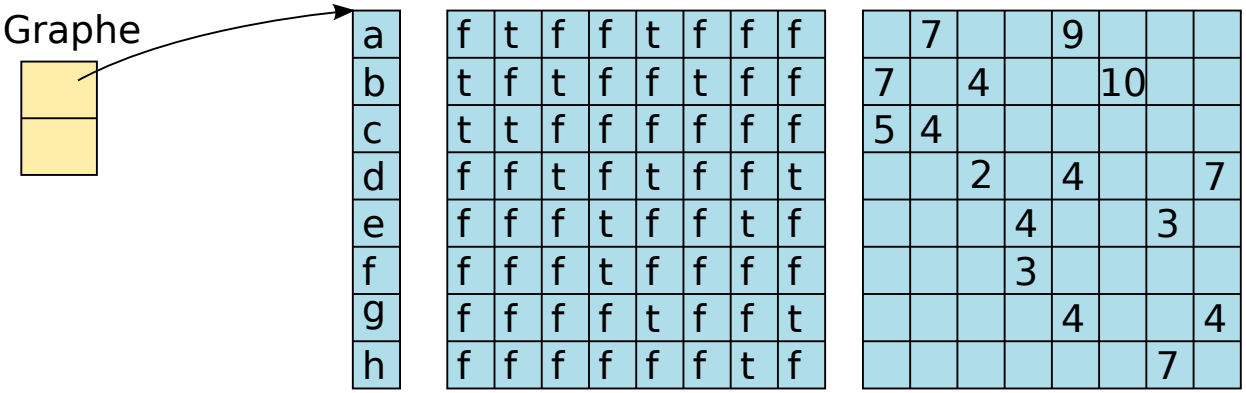
```
template <class S, class A>
class Graphe {
    struct Arete{
        S depart, arrivee;
        A etiquette;
    };
    Ensemble<S> sommets;
    Collection<Arete> aretes;
  //...
};

//Exemples d'instances
Graphe<char, int> g1;
Graphe<string, int> g2;
```

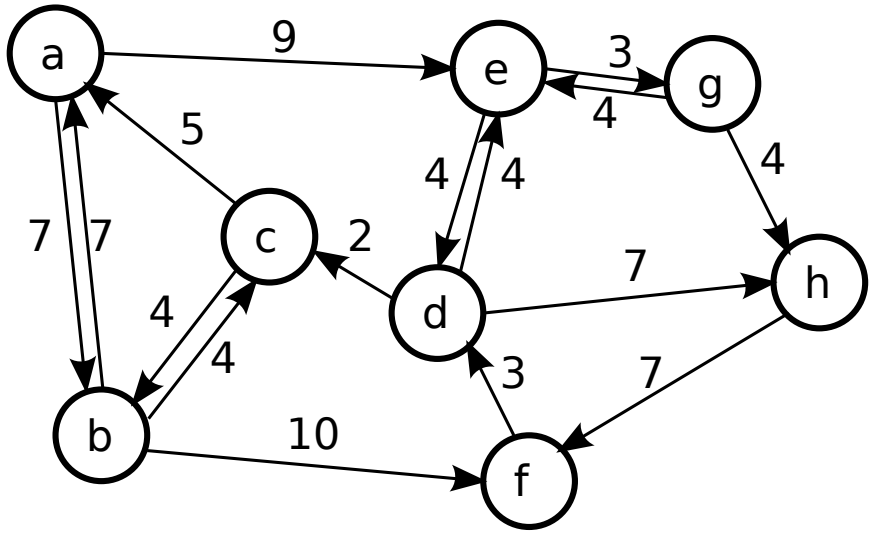




Matrice d'adjacence (1)

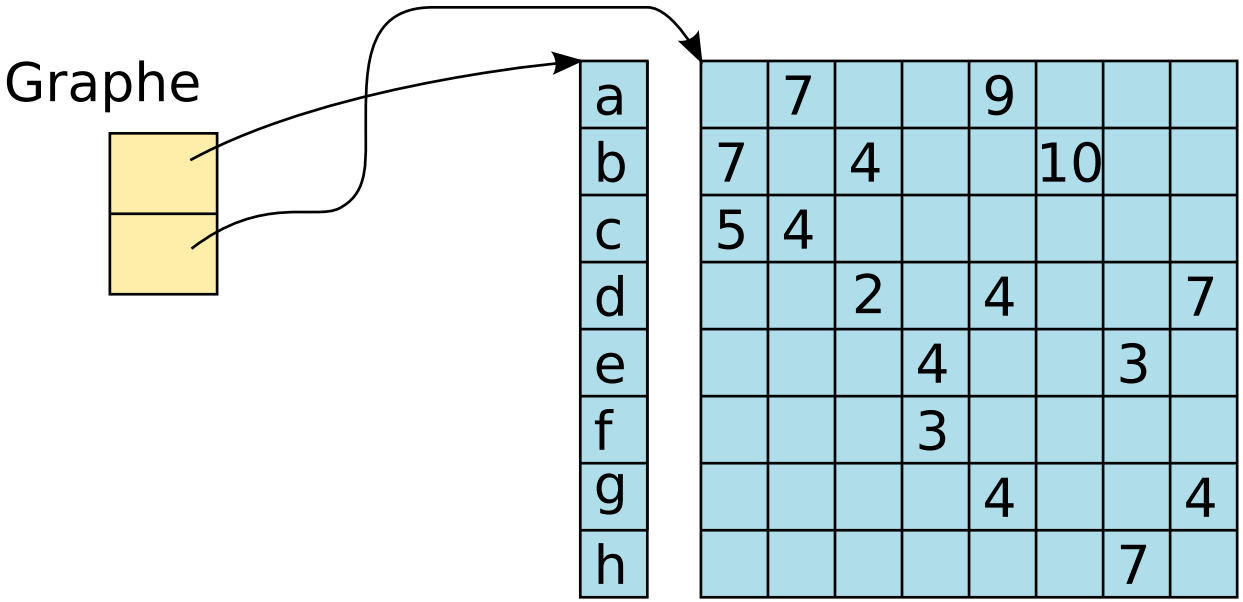
```
template <class S, class A>
class Graphe {
    Tableau<S> sommets;
    Tableau2D<bool>
        relations;
    Tableau2D<A> etiquettes;
};

//Exemples d'instances
Graphe<char, int> g1;
Graphe<string, int> g2;
```

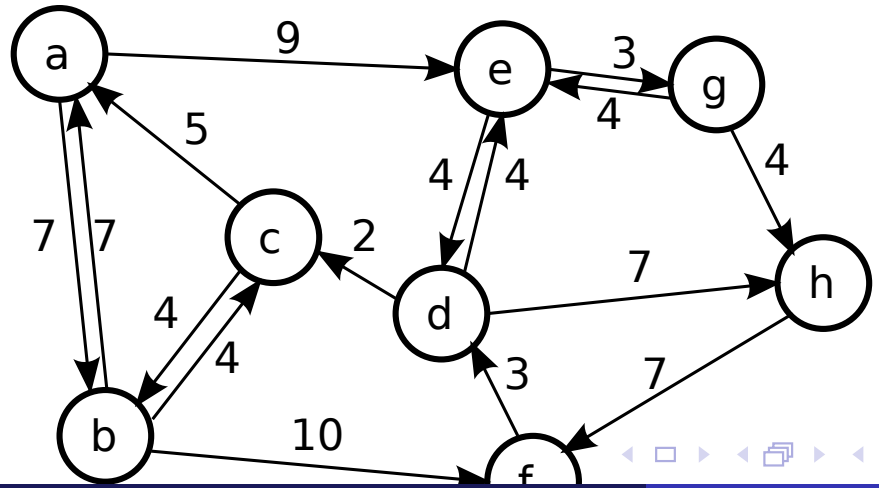




Matrice d'adjacence (2)

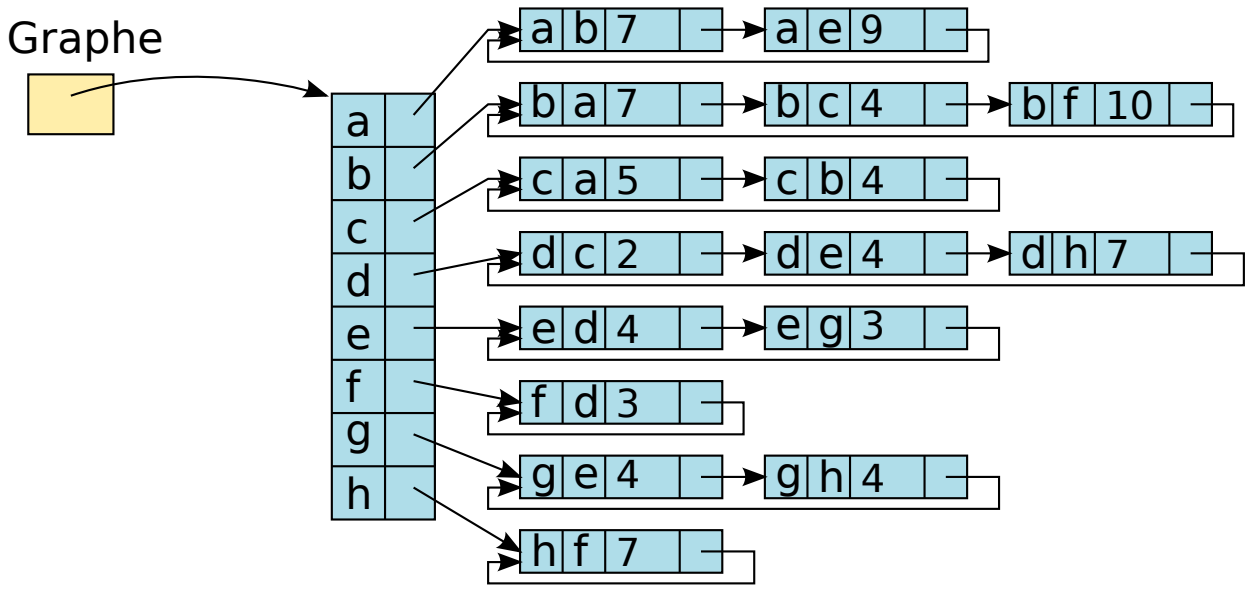
```
template <class S, class A>
class Graphe {
    Tableau<S> sommets;
    Tableau2D<A> etiquettes;
    static A
        AUCUNE_RELATION;
};
//...
double Graphe<string,
        double>::
  AUCUNE_RELATION =
        NaN; // 0/0
//...
Graphe<string, double>
        graphe.
```

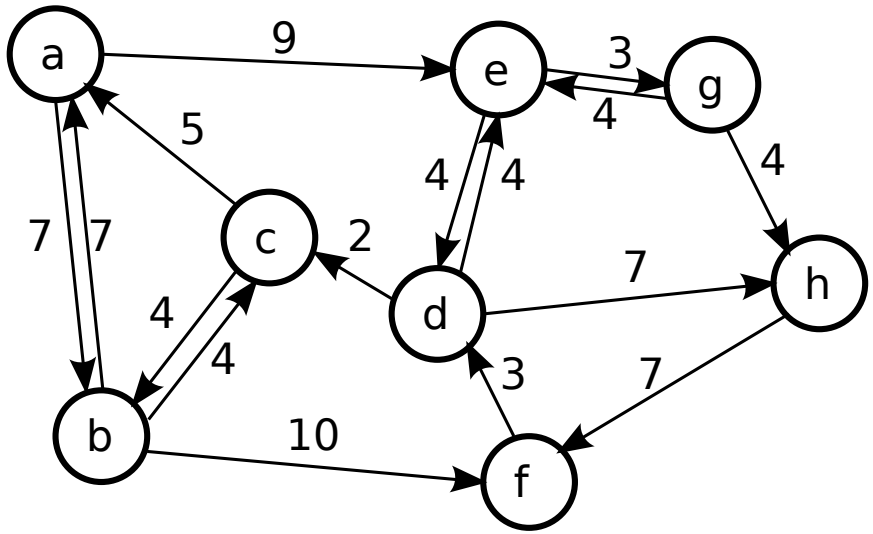




Listes d'adjacence

```
template <class S, class A>
class Graphe {
    struct Arete{
        S depart, arrivee;
        A valeur;
    };
    struct Sommet {
        S valeur;
        Liste<Arete>
        aretesSortantes;
        Liste<Arete>
        aretesEntrantes; //
        optionnel
    };
    Tableau<Sommet>
        sommets;
  //...
};
```

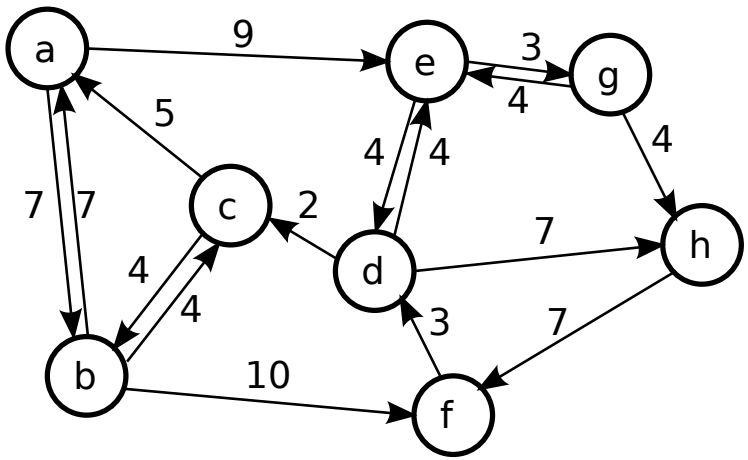




Dictionnaire avec listes d'adjacences

```
template <class S, class A>
class Graphe {
    struct Arete{
        S depart, arrivee;
        A valeur;
    };
    struct Sommet {
        Liste<Arete> aretesSortantes;
        Liste<Arete> aretesEntrantes;
        // optionnel
    };
    map<S, Sommet> sommets;
  //...
};
```

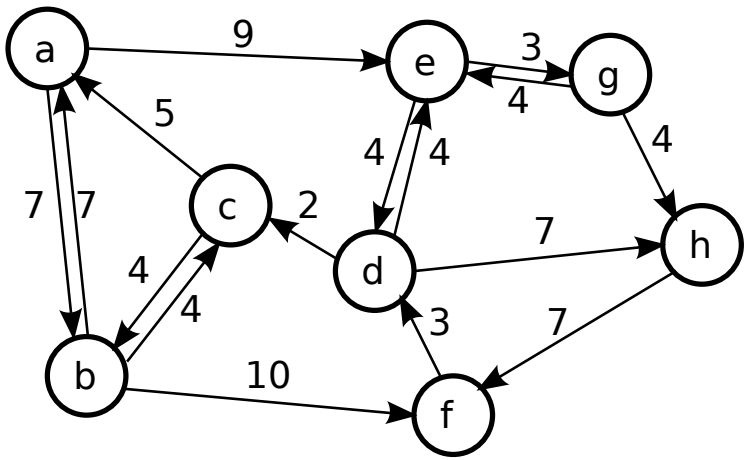
Exercice : représentation mémoire :



Dictionnaire de dictionnaires d'adjacences

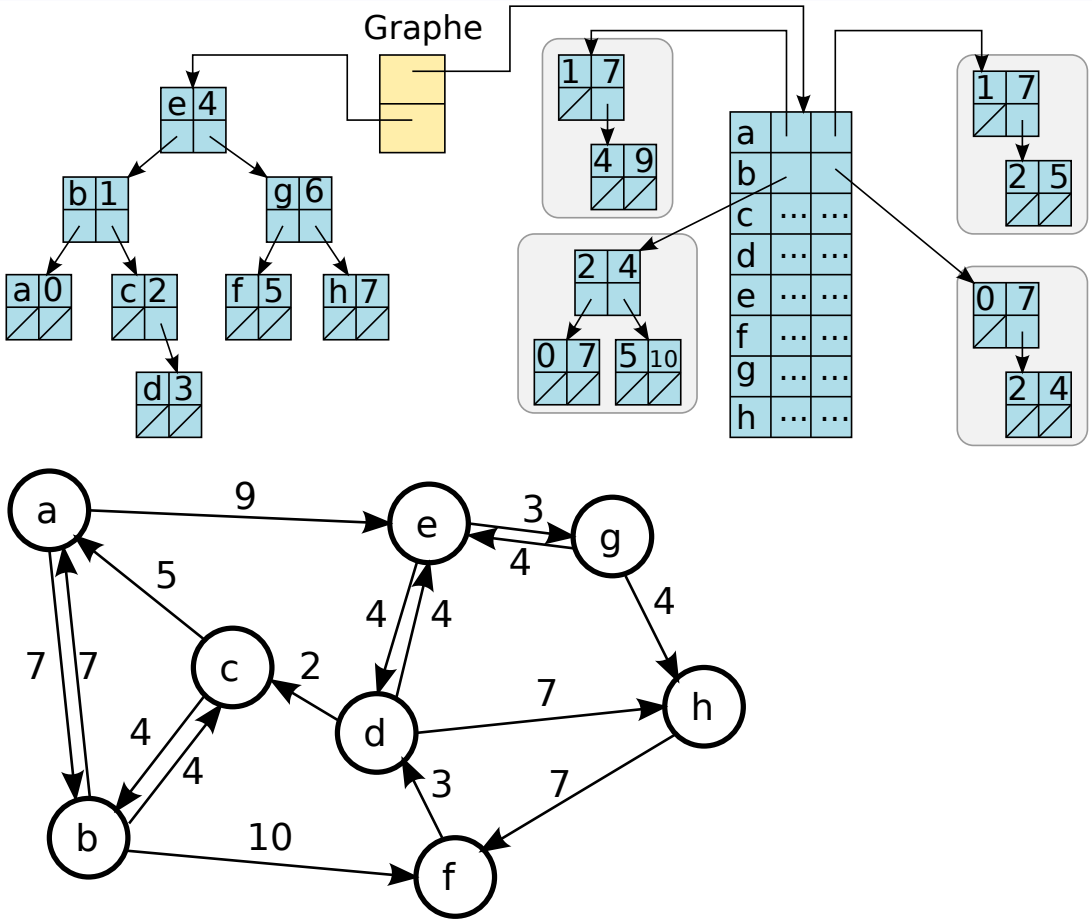
```
template <class S, class A>
class Graphe {
    struct Sommet {
        map<S, A> aretesSortantes;
        //map<S, A> aretesEntrantes;
        // optionnel
    };
    map<S, Sommet> sommets;
  //...
};
```

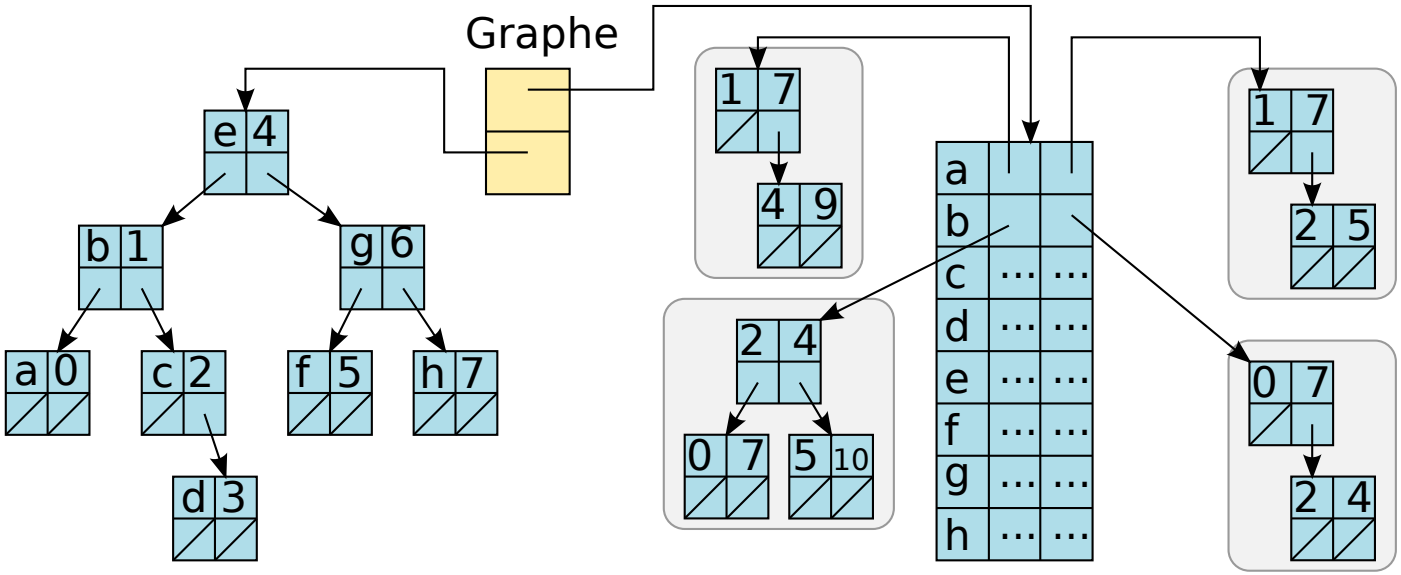
Exercice : représentation mémoire :

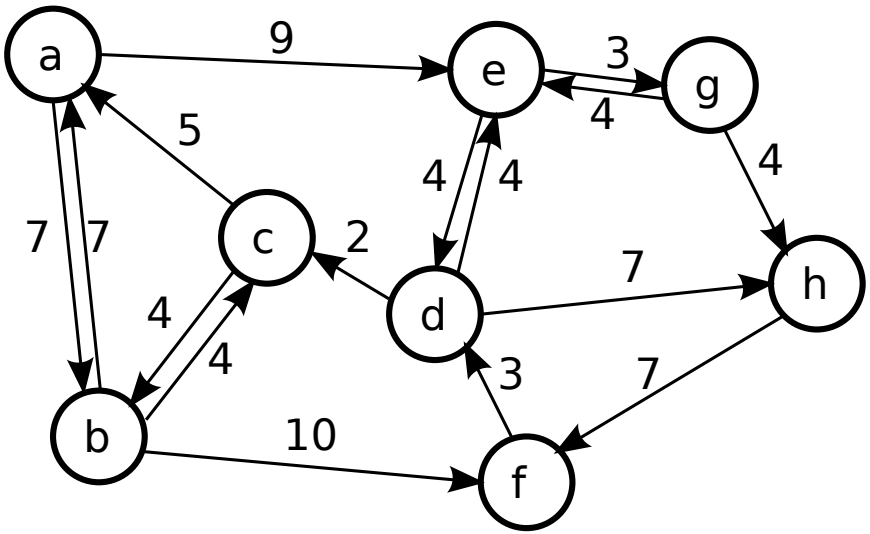


Ensembles d'adjacences avec des indices

```
template <class S, class A>
class Graphe {
  struct Sommet{
    Sommet(const S& s_) : s(s_){}
    S s; // optionnel
    ArbreMap<int, A>
        aretesSortantes;
    ArbreMap<int, A>
        aretesEntrantes; //optionel
  };
  ArbreMap<S, int> indices;
  Tableau<Sommet> sommets;
 //...
}
```




Graphe
e 4
b 1
g 6
a 0
c 2
f 5
h 7
d 3
1 7
4 9
2 4
0 7
5 10
a
b
c
d
e
f
g
h
1 7
2 5
0 7
2 4


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

Parcours

- Recherche en profondeur.
- Recherche en largeur.

Parcours recherche en profondeur

1. RECHERCHEPROFONDEUR($G = (V, E)$, $v \in V$)
2. v.visité $\leftarrow$ vrai
3. pour toute arête $e \in v$.aretesSortantes()
4. $w \leftarrow e$.arrivee
5. si $\neg w$.visité
6. RechercheProfondeur(G, w)

Parcours recherche en largeur

1. RECHERCHELARGEUR($G = (V, E)$, $v \in V$)
2. $\quad$ *file* $\leftarrow$ CRÉERFILE
3. $\quad$ s.visité $\leftarrow$ vrai
4. $\quad$ *file*.ENFILER(v)
5. $\quad$ tant que $\neg$*file*.vide()
6. $\quad\quad$ $s \leftarrow$ *file*.defiler()
7. $\quad\quad$ pour tout arête $a = (s, s') \in E$
8. $\quad\quad\quad$ si $\neg s'$.visité
9. $\quad\quad\quad\quad$ s'.visité $\leftarrow$ vrai
10. $\quad\quad\quad\quad$ *file*.ENFILER(s')