INF3105 – Graphes

Jaël Champagne Gareau

Université du Québec à Montréal (UQAM)

Été 2024

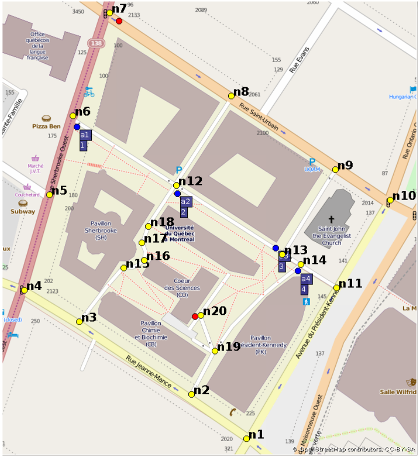
http://cria2.uqam.ca/INF3105/

UQÀM

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 Saint-Urbain
Rue Jeanne-Mance
Rue Evans
Avenue du Président-Kennedy
Pizza Ben
Subway
Office québécois de la langue française
Pavillon Sherbrooke (SH)
Université du Québec à Montréal
Coeur des Sciences (CO)
Pavillon Chimie et Biochimie (CB)
Pavillon Président-Kennedy (PK)
Saint John the Evangelist Church
Salle Wilfrid
© OpenStreetMap contributors, CC-BY-SA

Makefile

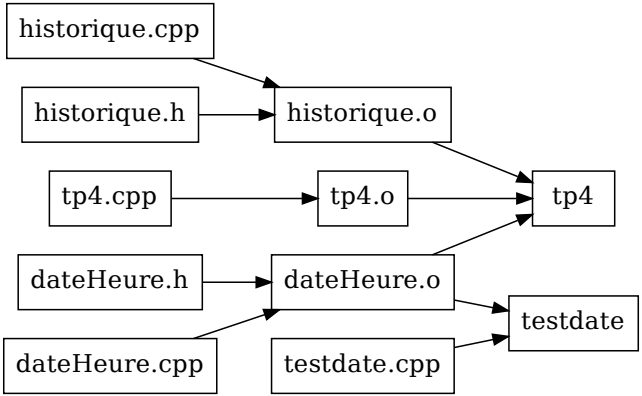
```
# Makefile pour TP4.
tp4: tp4.cpp historique.o dateheure.o
    g++ ${OPTIONS} -o tp4 tp4.cpp historique.o dateheure.o

dateheure.o: dateheure.cpp dateheure.h
    g++ ${OPTIONS} -c -o dateheure.o dateheure.cpp

historique.o: historique.cpp historique.h dateheure.h arbreavl.h
    g++ ${OPTIONS} -c -o historique.o historique.cpp

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

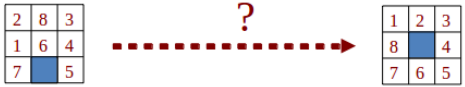
Makefile

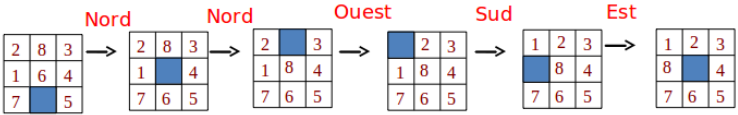


Certaines dépendances .h ne sont pas illustrées.

Jeu de taquin – Espace d'états

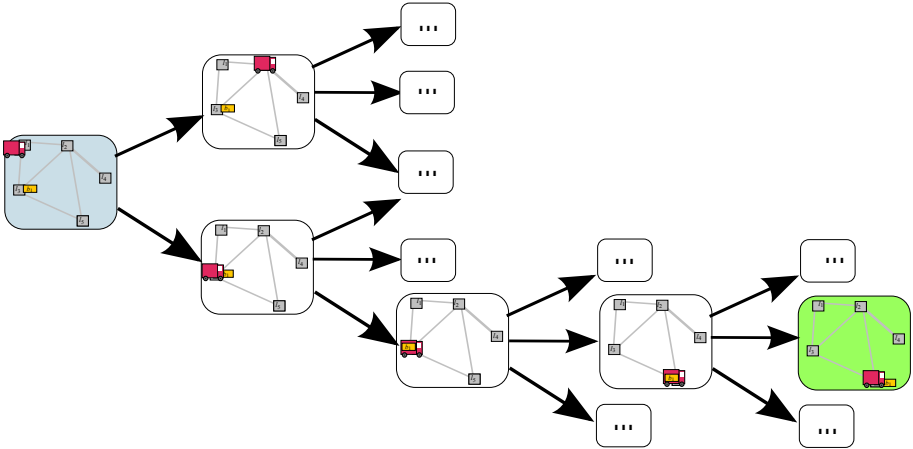
http://cpt.toutimages.com/jeu_de_taquin/





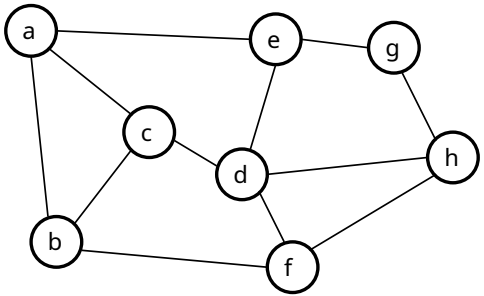
Abordé dans le cours INF4230 (Intelligence artificielle).

Recherche dans un espace d'états

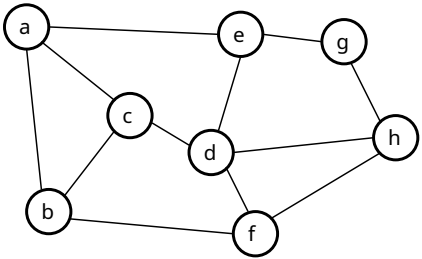


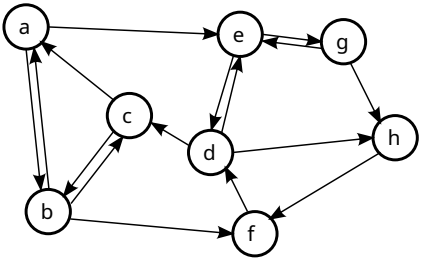
Définition formelle d'un graphe

- Un **graphe** est un tuple $G = (S, A)$ où
  - S est un ensemble de **sommets** / **nœuds** (*vertex / vertices*) ;
  - $A \subseteq S \times S$ est un ensemble d'**arêtes** / **arcs** (*edges*).
- 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 orientées.
- Un **graphe non orienté** (*undirected graph* en anglais) est composé d'arêtes non orientées.

Sous-Graphe, Chemin, Longueur

- Un **sous-graphe** d'un graphe $G = (S, A)$ est un graphe $G' = (S', A')$ où :
  - $S' \subseteq S$;
  - $A' \subseteq A$;
  - $\forall a = (x, y) \in A', x \in S', y \in S'$.
- Un **sous-graphe maximal** est un sous-graphe (S', A') où :
  $\forall a = (x, y) \in A$, si $x \in S'$ et $y \in S'$, alors $a \in A'$.
- Un **chemin** $\langle s_1, s_2, \ldots, s_k \rangle$ est une séquence de sommets dans un graphe.
- La **longueur d'un chemin** peut être :
  - le nombre d'arêtes ;
  - le nombre de sommets ;
  - la somme des poids des arêtes formant le chemin.

Cycle, Boucle, Graphe acyclique

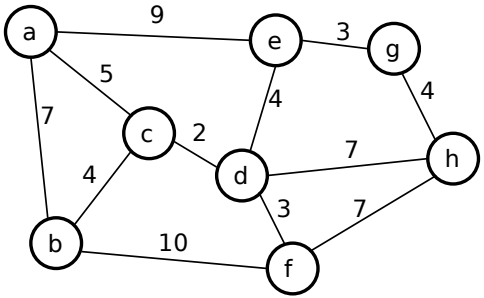
- Un **cycle** est un chemin dans un graphe dont le sommet de départ est le même que le sommet d'arrivée et où chaque arête utilisée sur le chemin est distincte.
- Une **boucle** est une arête dont les sommets de départ et d'arrivée sont les mêmes.
- Un graphe est dit **acyclique** s'il ne contient aucun cycle.

Connexité, Composantes connexes

- Un graphe non orienté est **connexe** (ou **connecté**) s'il existe au moins un chemin entre toutes les paires de sommets.
- Un graphe $G = (S, A)$ non connecté est composé de plusieurs composantes connexes.
- Une **composante connexe** est un sous-graphe connecté et maximal.
- Un graphe orienté est **fortement connexe** si et seulement si pour toute paire de sommets (s_1, s_2) il existe un chemin de s_1 à s_2, et de s_2 à s_1.
- Un graphe orienté non fortement connexe est composée de plusieurs composantes fortement connexes.

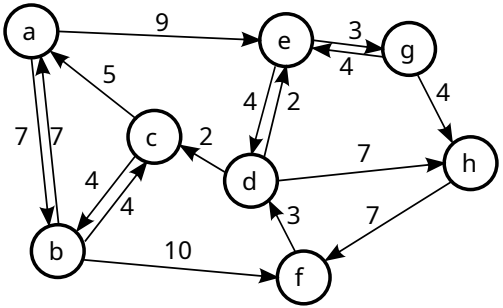
Étiquette

- Une **étiquette** (*label*) sur une arête indique une propriété.
- Exemple : Le **poids** d'une arête peut indiquer son coût.



Degré

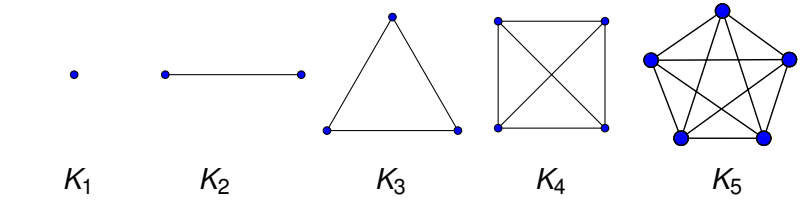
- Dans un graphe non orienté, le **degré** d'un sommet $v \in V$, noté $deg(v)$, est le nombre d'arêtes qui y sont reliées.
- Dans un graphe orienté, on peut faire la distinction entre le **degré sortant** et le **degré entrant**, qui sont respectivement notés $deg_{out}(v)$ et $deg_{in}(v)$.



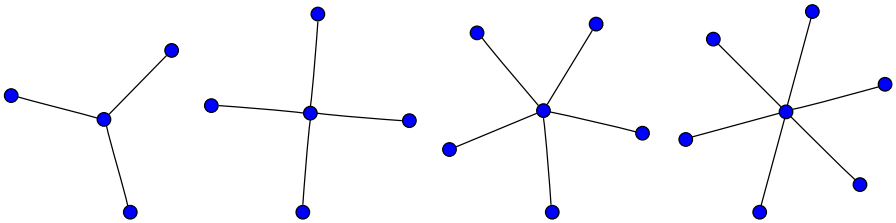
Arbre, Forêt

- Un **arbre** est un graphe connexe acyclique non orienté ($|S| = |A| + 1$).
- Un graphe qui est composé d'un ensemble d'arbres est appelé une **forêt**.

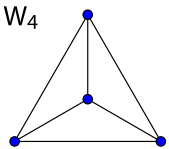
Quelques familles de graphes – Complet, Cycle

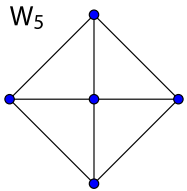


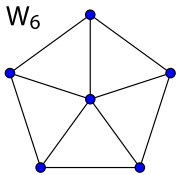
Quelques familles de graphes – Étoile

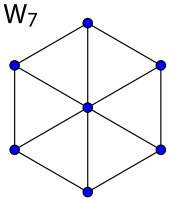


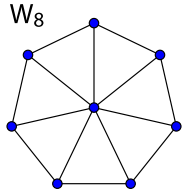
Quelques familles de graphes – Roue

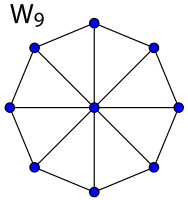










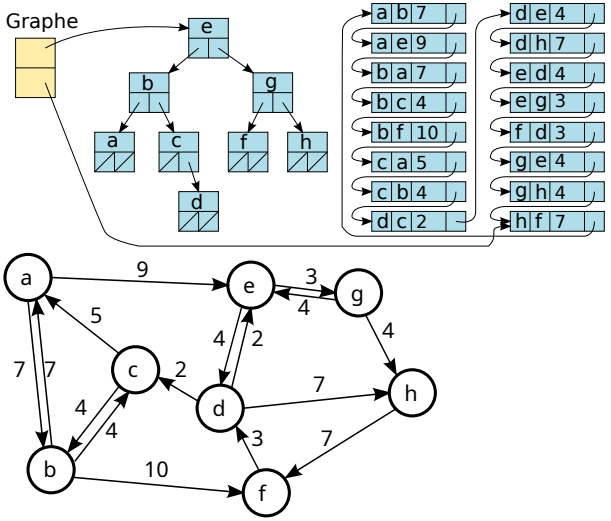


Considérations

- Complexité :
  - Complexité spatiale (mémoire).
  - Complexité temporelle des opérations :
    - Interroger l'existence d'une arête et obtenir son 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 :
  - $n = |S|$: nombre de sommets.
  - $m = |A|$: nombre d'arêtes.
  - densité du graphe : m/n.
- Généralement : compromis en fonction de l'application.

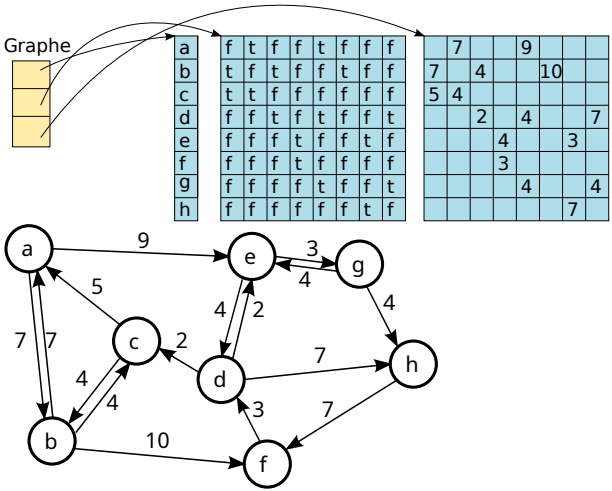
Ensemble de sommets et collection d'arêtes

```
1  template <class S, class A>
2  class Graphe {
3      struct Arete {
4          S depart, arrivee;
5          A etiquette;
6      };
7      Ensemble<S> sommets;
8      Collection<Arete> aretes;
9  };
10
11 // Exemple d'instance
12 Graphe<char, int> g1;
13 Graphe<string, int> g2;
```



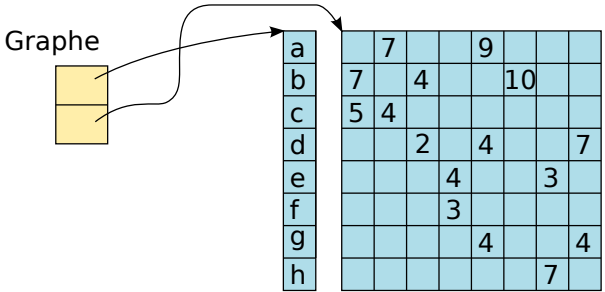
Matrice d'adjacence (1)

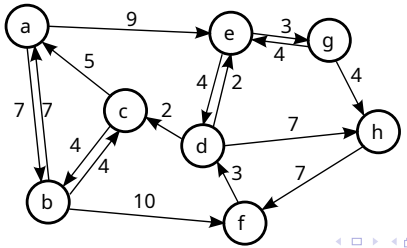
```
1  template <class S, class A>
2  class Graphe {
3      Tableau<S> sommets;
4      Matrice<bool> relations;
5      Matrice<A> etiquettes;
6  };
7
8  //Exemples d'instances
9  Graphe<char, int> g1;
10 Graphe<string, int> g2;
```



Matrice d'adjacence (2)

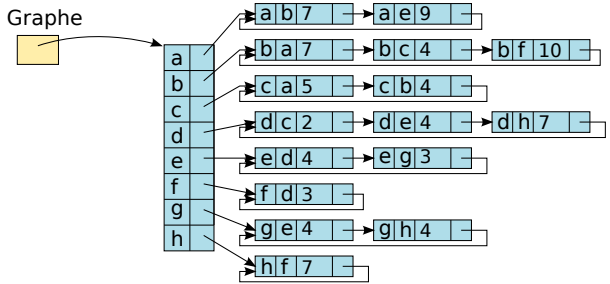
```
1  template <class S, class A>
2  class Graphe {
3      Tableau<S> sommets;
4      Matrice<A> etiquettes;
5      static A AucuneRelation;
6  };
7  //...
8  double Graphe<string, double>::
9    AucuneRelation = NaN; // 0/0
10 //...
11 Graphe<string, double> graphe.
```





Listes d'adjacence

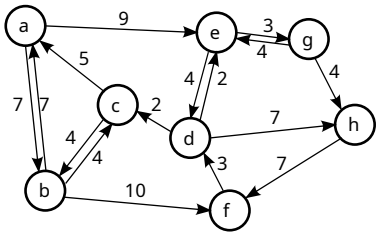
```
1  template <class S, class A>
2  class Graphe {
3      struct Arete {
4          S depart, arrivee;
5          A valeur;
6      };
7      struct Sommet {
8          S valeur;
9          Liste<Arete> arcsSortants;
10         // optionnel
11         Liste<Arete> arcsEntrants;
12     };
13     Tableau<Sommet> sommets;
14   //...
15 };
```



Dictionnaire avec listes d'adjacences

```
1  template <class S, class A>
2  class Graphe {
3      struct Arete {
4          S depart, arrivee;
5          A valeur;
6      };
7      struct Sommet {
8          Liste<Arete> arcsSortants;
9          // optionnel
10         Liste<Arete> arcsEntrants;
11     };
12     map<S, Sommet> sommets;
13   //...
14 };
```

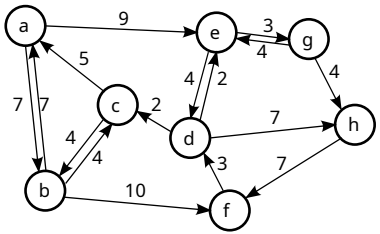
Exercice : représentation mémoire :



Dictionnaire de dictionnaires d'adjacences

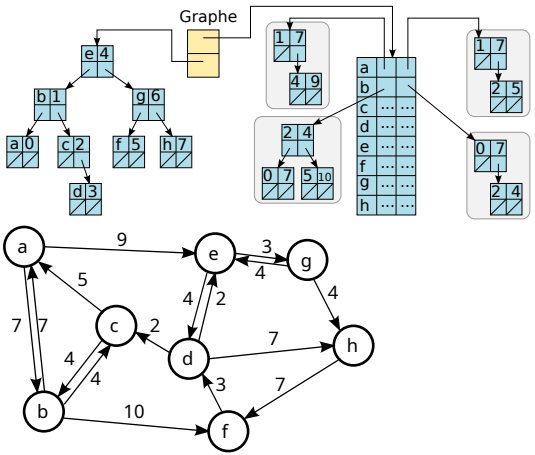
```
1  template <class S, class A>
2  class Graphe {
3      struct Sommet {
4          map<S, A> arcsSortants;
5          // optionnel
6          map<S, A> arcsEntrants;
7      };
8      map<S, Sommet> sommets;
9    //...
10 };
```

Exercice : représentation mémoire :



Ensembles d'adjacences avec des indices

```
1  template <class S, class A>
2  class Graphe {
3    struct Sommet {
4      Sommet(const S& s_) : s(s_){}
5      S s; // optionnel
6      map<int, A> arcsSortants;
7      // optionnel
8      map<int, A> arcsEntrants;
9    };
10   map<S, int> indices;
11   Tableau<Sommet> sommets;
12  //...
13 }
```

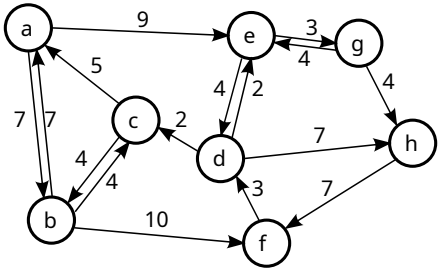


Parcours

- Recherche en profondeur.
- Recherche en largeur.

Parcours recherche en profondeur (DFS)

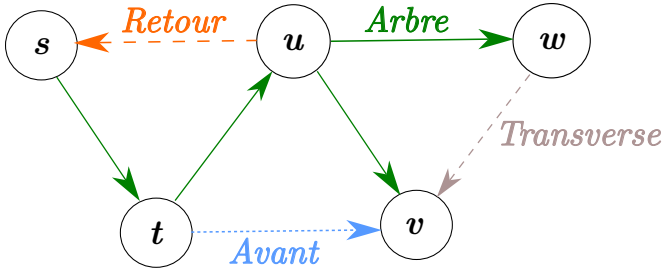
1. RECHERCHEPROFONDEUR($G = (S, A)$, $u \in S$)
2. u.visité $\leftarrow$ vrai
3. pour tout arc $(u, v) \in u$.arcsSortants()
5. si $\neg v$.visité
6. RechercheProfondeur(G, v)



Types d'arcs lors d'une recherche en profondeur

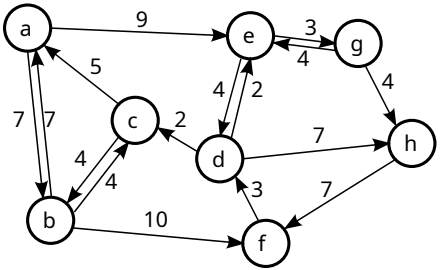
Les arcs (u, v) d'un graphe peuvent être classifiés en quatre types :

- **Arc d'arbre** (*tree edge*) : arc faisant parti de l'arbre de recherche.
- **Arc avant** (*forward edge*) : u est un ancêtre de v dans l'arbre de recherche.
- **Arc de retour** (*back edge*) : v est un ancêtre de u dans l'arbre de recherche.
- **Arc transverse** (*cross edge*) : u et v ne sont pas ancêtres l'un de l'autre.



Parcours recherche en largeur (BFS)

```
1.  RECHERCHELARGEUR(G = (S, A), s ∈ S)
2.      file ← CRÉERFILE
3.      s.visité ← vrai
4.      file.ENFILER(s)
5.      tant que ¬file.vide()
6.          u ← file.defiler()
7.          pour tout arc (u, v) ∈ u.arcsSortants()
8.              si ¬v.visité
9.                  v.visité ← vrai
10.                 file.ENFILER(v)
```



Exemple Dijkstra