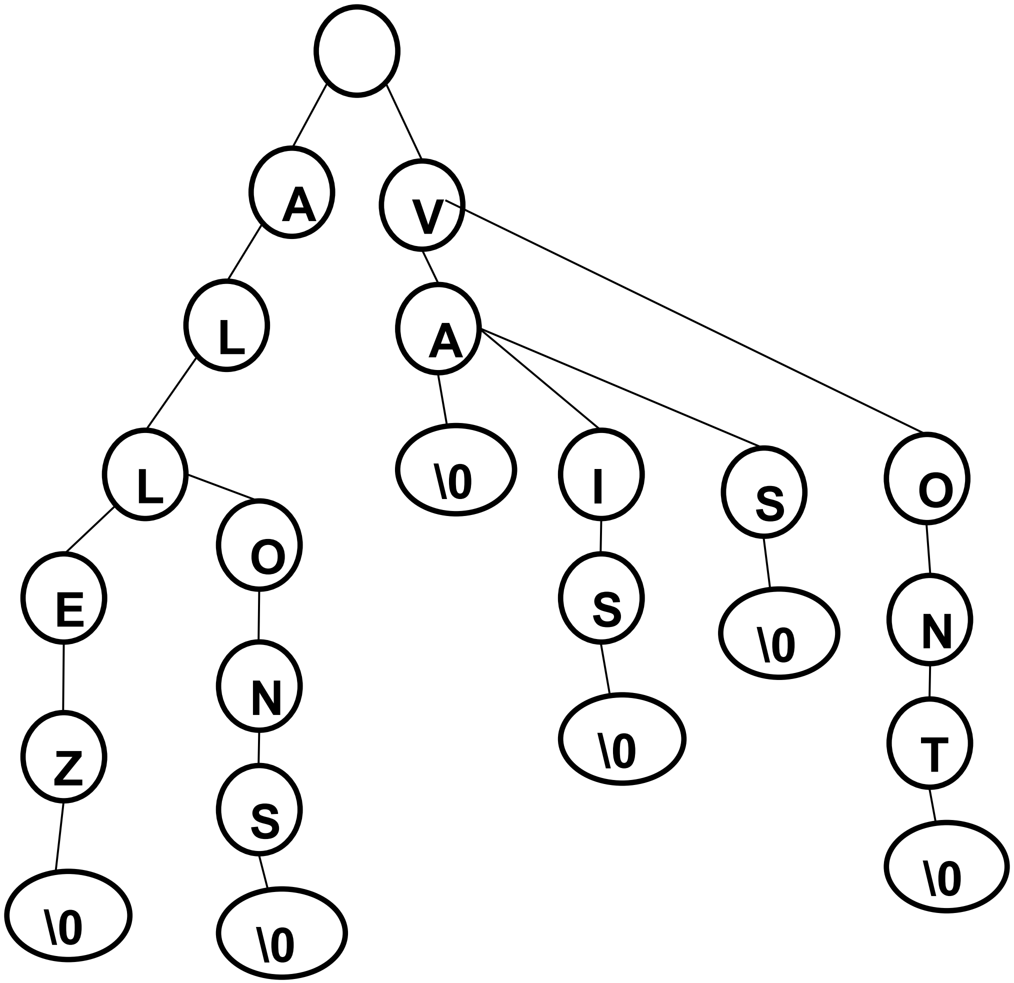
Graphes

Graphe

- Structure de données qui permet de représenter un réseau de données
  – Réseaux de télécommunication
  – Réseaux sociaux
  – Réseaux routier
  – Carte
  – …
- Objectifs
  – Représentation d'un graphe
  – Parcours de graphe
  – Quelques problèmes

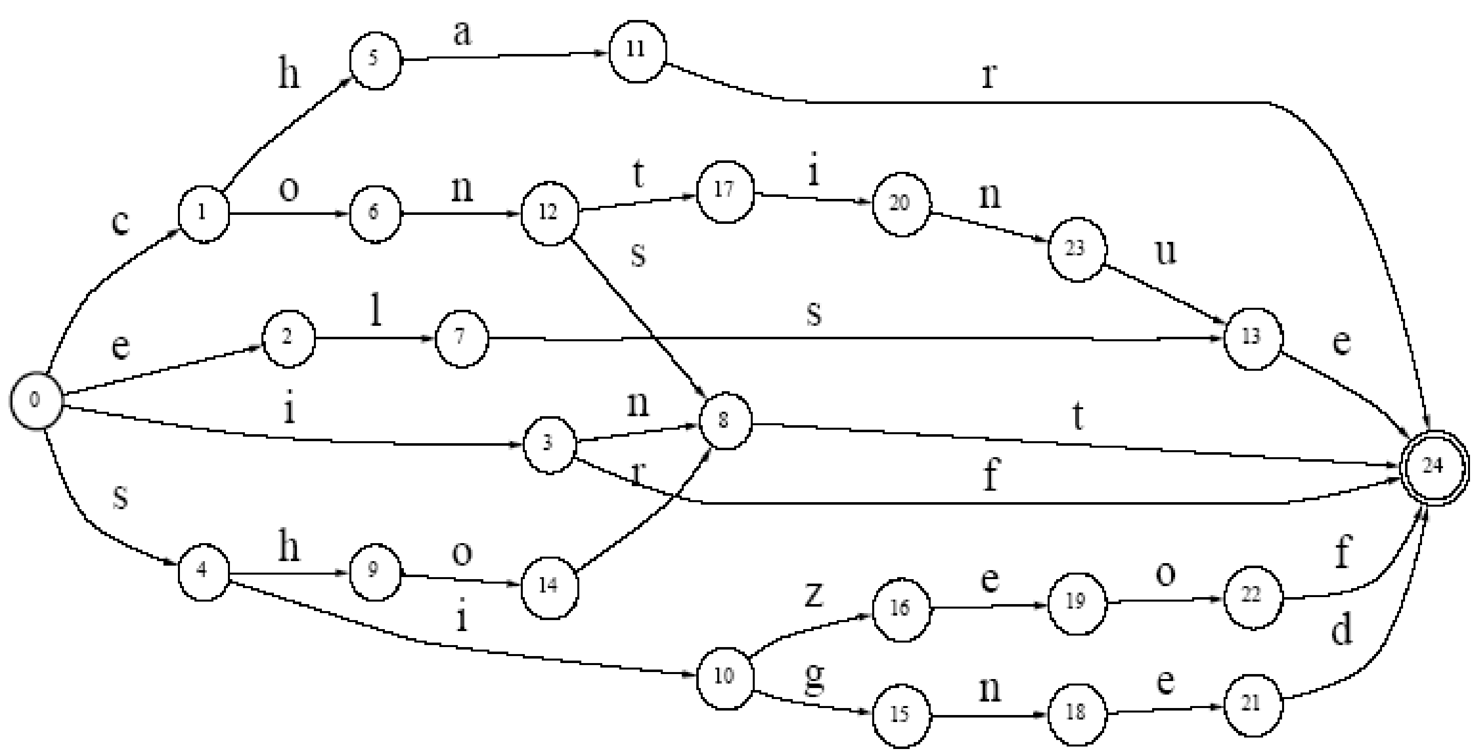
Arbres

- Chaque chemin (de la racine à une feuille) représente un mot
- Optimisation en terme de taille
- Besoin de marquer la fin d’un mot
  - **allez, allons, va, vais, vas, vont**

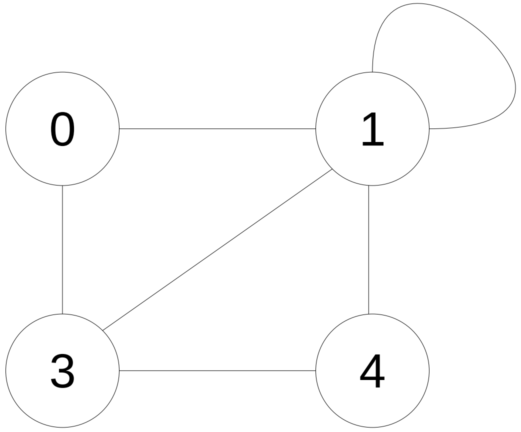


Graphe

- Meilleure représentation pour coder un lexique
  - Optimisation en terme de taille de stockage
- Quels mots sont représentés ci-dessous ?



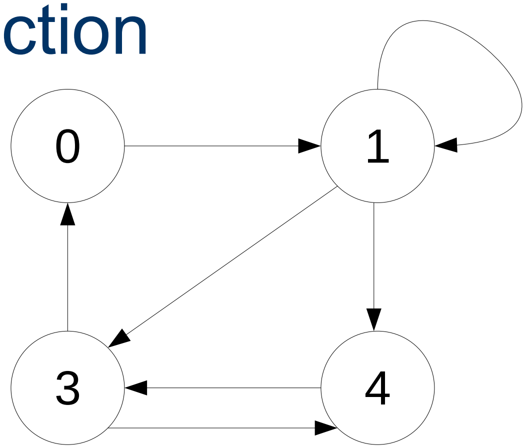
Définitions



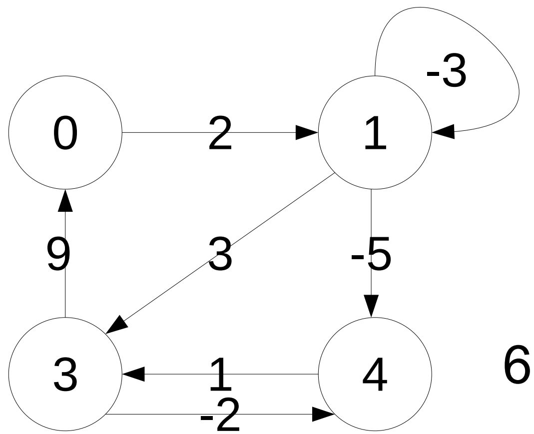
- Graphe : couple G[V,E] où
  - V est un ensemble de nœuds ou sommets et
  - E est l'ensemble des paires de sommets reliés entre eux (arêtes du graphe ou «arc»)
- Chemin = séquence d'arêtes menant d'un sommet i à un sommet j
- Circuit = chemin dont les sommets de départ et d'arrivée sont identiques
- degré d'un sommet = nombre d'arêtes ayant ce sommet pour extrémité
- Voisins : les voisins des sommets sont ceux qui sont reliés à ce sommet par une arête

Graphes orientés

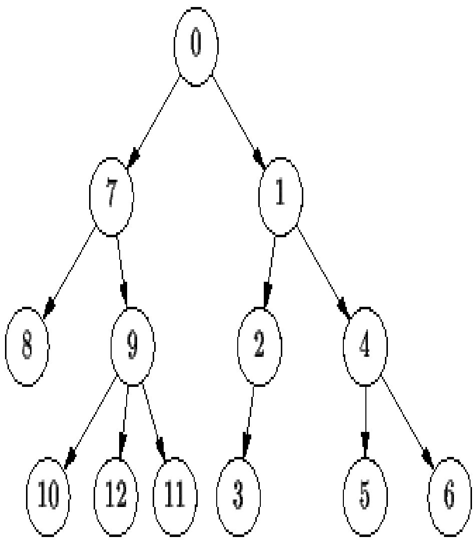
- Graphe dans lequel chaque arête a une direction associée
- Arc = arête orientée



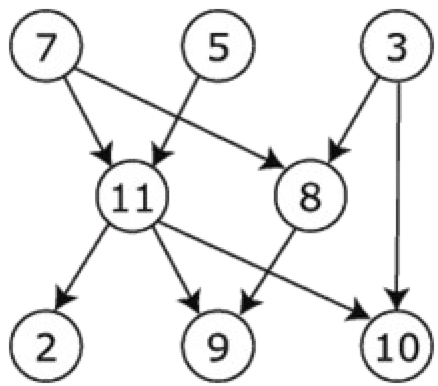
- Graphe orienté pondéré:
  - Graphe étiqueté ou chaque arc a un coût associé
- valuation, coût = valeur numérique associée à un arc ou à un sommet



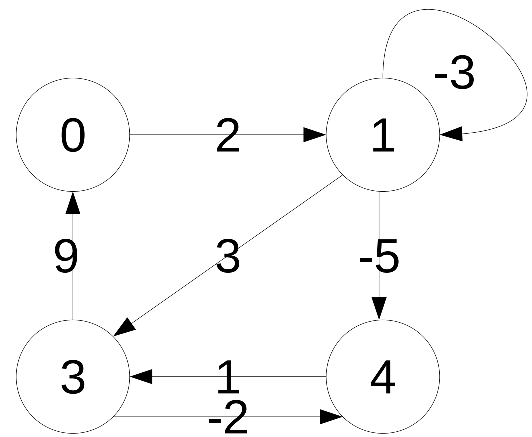
Types de graphe



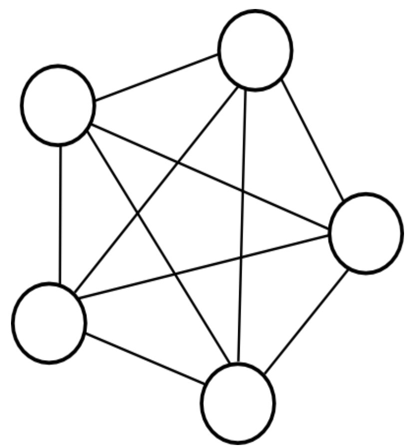
arbre



DAG



Graphe orienté



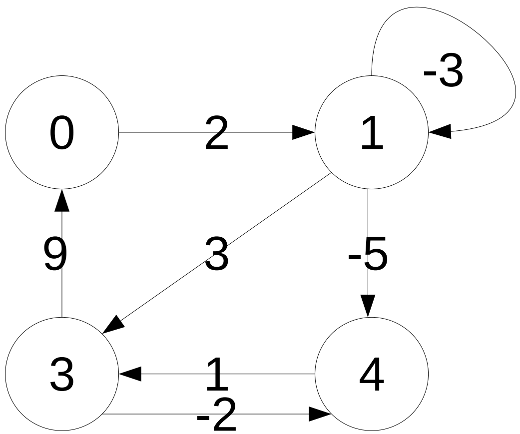
Graphe complet

Opérations

- graph_order : G → Integer
  - Retourne le nombre total de sommets de G
- graph_size : G → Integer
  - Retourne le nombre total d’arêtes de G
- graph_add_vertex : G v → G
  - Ajoute le nœud v si v n'est pas déjà présent
- graph_del_vertex : G v → G
  - Supprime v de G si v présent
- graph_add_edge : G u v→ G
  - Ajoute un arc entre u et v si absent
- graph_del_edge : G u v→ G
  - Supprimer l'arc entre u et v si présent
- graph_get_edges : G v → list
  - Retourne l’ensemble des arêtes du nœud v

Graphe : représentation matricielle

- Tout graphe G = (V,E) peut être représenté par une matrice d'adjacence
- $m_{i,j} = val(v_{i,}, v_j)$ si $(v_{i,}, v_j)$ € E, ∞ sinon



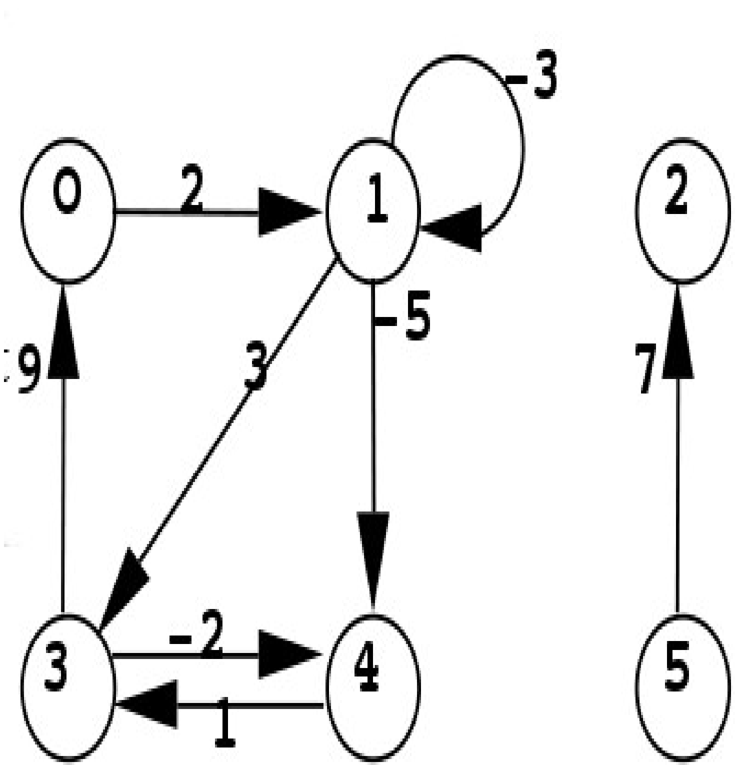
Arrivée →	0	1	3	4
0	∞	2	∞	∞
1	∞	-3	3	-5
3	9	∞	∞	-2
4	∞	∞	1	∞

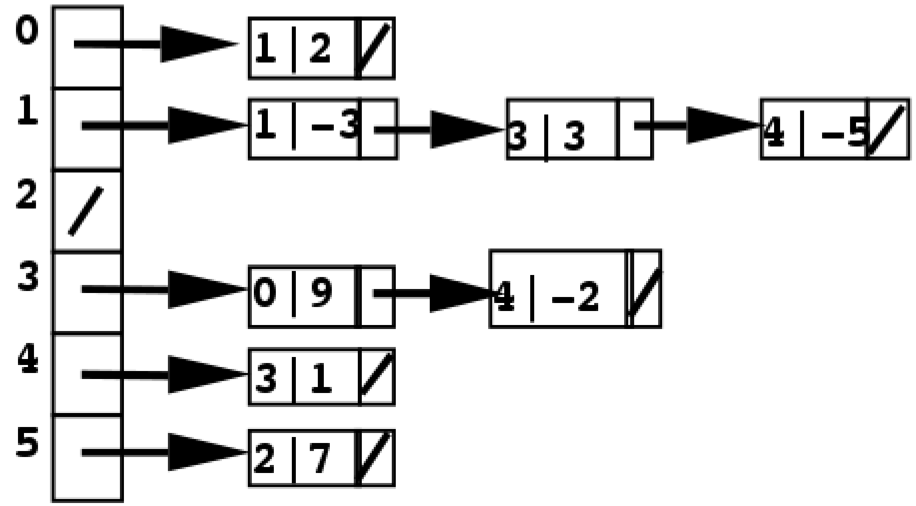
Graphe : représentation matricielle

- Complexité spatiale → $O(|X|^2)$
- Coût d'ajout/suppression d'un nœud → $O(|X|^2)$
- Coût d'ajout/suppression d'un arc → $O(1)$

- Fortement déconseillé pour les graphes creux !
- Meilleur choix si la rapidité d’accès est cruciale, si le graphe est très dense (→ complet) et/ou non mutable

Graphe : représentation par liste d'adjacence





- Sommets
  - numérotés et stockés dans un tableau
  - contiennent une liste des arcs qui partent de ce sommet
- Arc
  - triplet <départ,arrivée,coût >

Graphe : représentation par liste

- Complexité spatiale → O(|X|+|E|)
- Coût d'ajout d'un nœud → O(1)
- Coût d'ajout d'un arc → O(1)
- Coût suppression d'un nœud → O(|E|)
- Coût suppression d'un arc → O(|E|/|X|)

- Accès moins rapide que la représentation matricielle
- Parfait pour représenter les graphes creux !
- Meilleur compromis de complexité

Représentation informatique

- typedef struct {int start, end; double cost; } **edge_t**;
- typedef struct { element_t value; list edges;} **vertex_t**;
- typedef struct { int order_vertex; int size_egdes;
  vertex_t* data; } **graph_t**;
- Pourquoi la définition suivante est-elle dangereuse ?
  - ~~typedef struct {vertex_t start, end; double cost; } edge_t;~~
  - Amènera plein de copies de nœuds possédant eux-mêmes des listes d'arcs… → utilisation d'indice ou de pointeurs vers les nœuds

Parcours d’un graphe

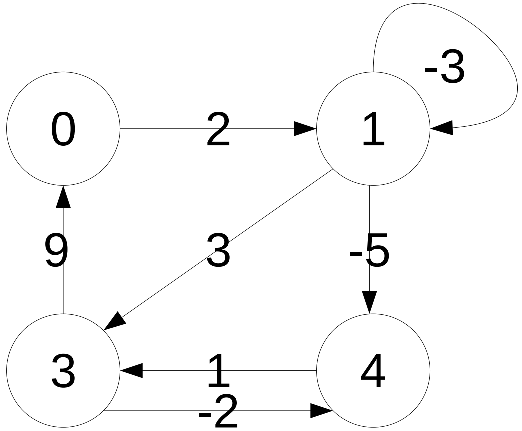
Parcours de graphe

- Parcours non ordonnés :
  – Parcours du tableau de nœuds (p.ex., affichage des valeurs des nœuds)
  – Parcours des listes d'arêtes (p.ex., initialisation de poids)

- Parcours ordonnés selon les arêtes :
  – Problème de cycle (comment être sur de ne pas passer plusieurs fois au même endroit ?)
  – Problème d’initialisation (début) et de terminaison (fin)
  – Problème de sens de parcours (tous les fils ou tous les frères d'abord ?)

Parcours en profondeur d'abord

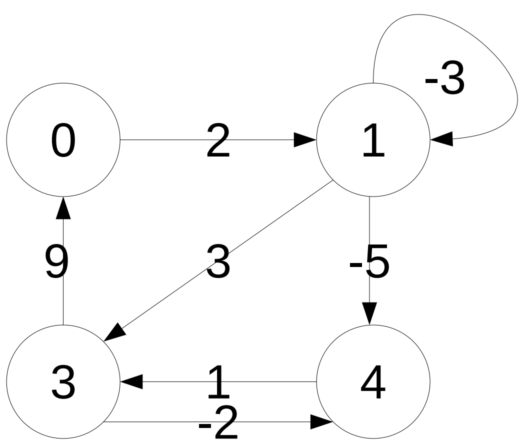
- parcours_profondeur(graphe G, sommet s, ensemble O)
  - Si s ∉ O (s n'a pas encore été ouvert) alors
    - O ← O U s
    - Pour tout voisin v de s :
      - parcours_profondeur(G,v,O)
- Exemple:



- Même algorithme que pour les arbres

Parcours en largeur d'abord

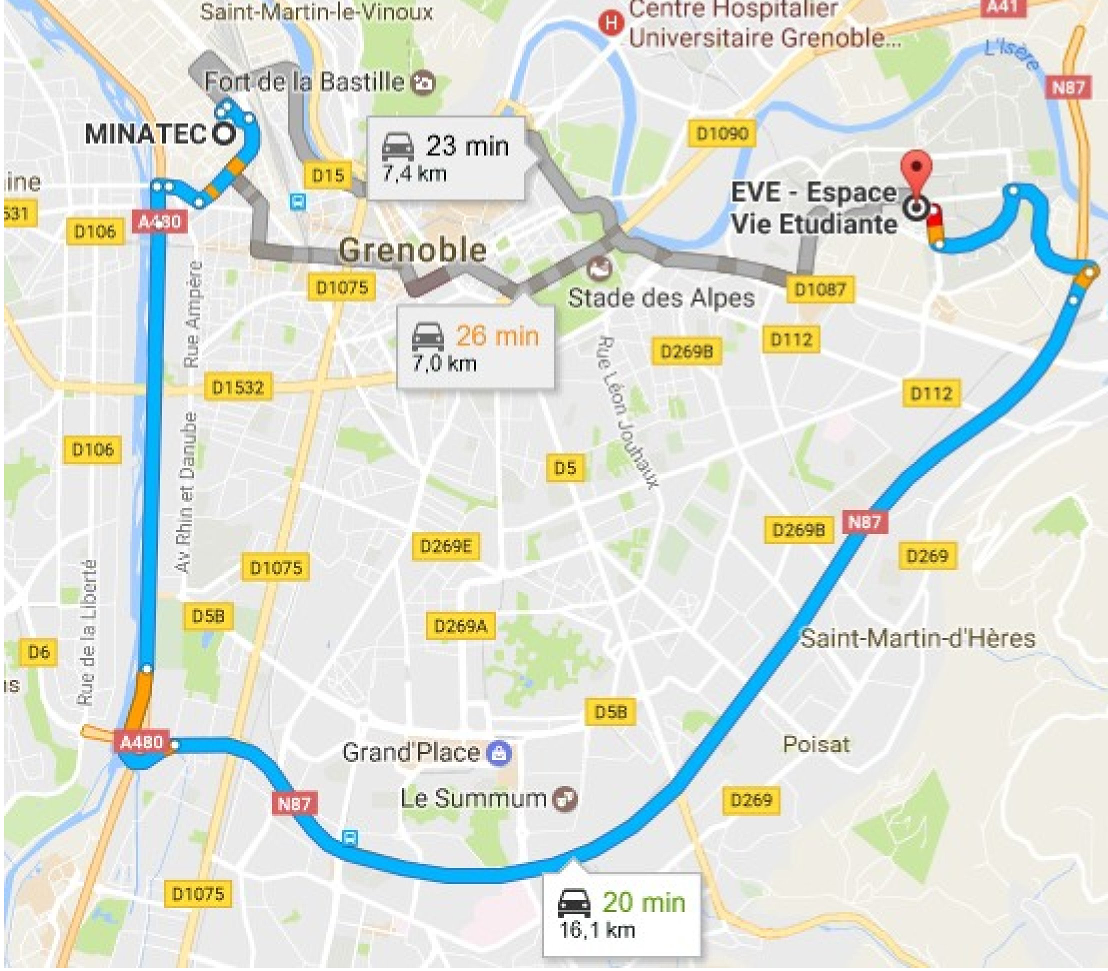
- Stockage des frères par file d'attente
- parcours_largeur(graphe G, sommet s)
  - File F ← {s}
  - **Tant que** F != ∅
    - s ← défiler (F)
    - O ← O U s
    - Action (s) // exemple Afficher s.
    - Pour tout voisin v de s :
      - Si v ∉ O
        - » F ← enfiler (F,v)
- Exemple:



- Même algorithme que pour les arbres

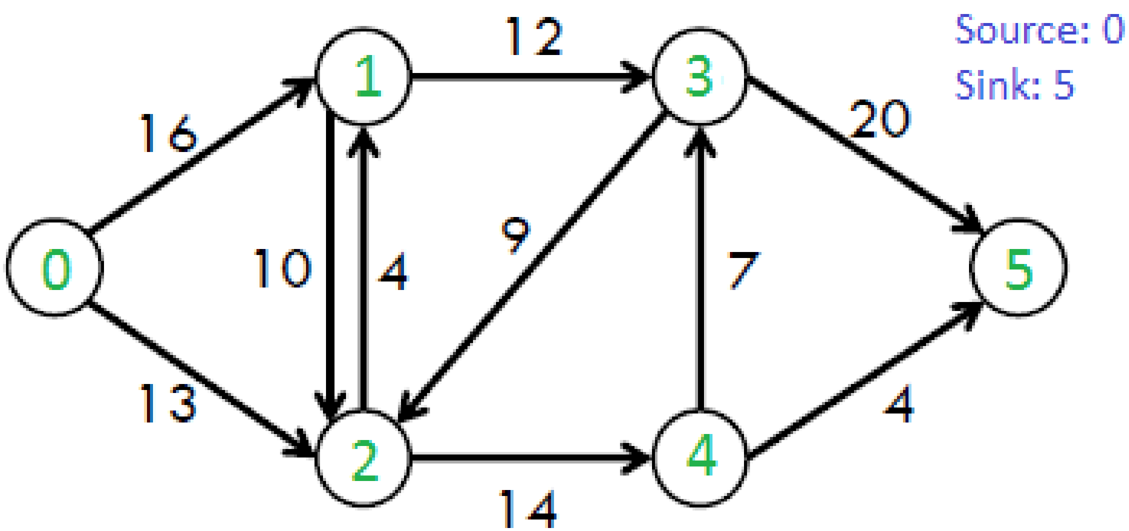
Problèmes classiques représentables par un graphe

Problème : plus court chemin



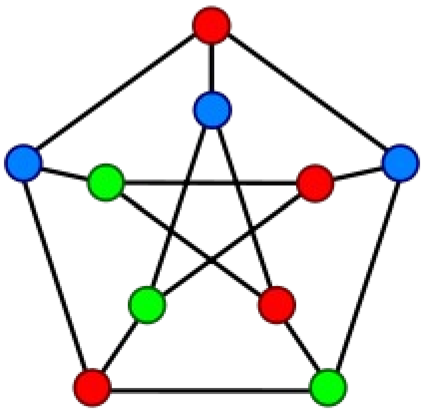
- Les nœuds sont les intersections
- Les arcs sont valués par la distance ou le temps
- Un plus court chemin entre un nœud A et B ou à partir de A ou entre toutes paires
  - Dijkstra $O(V^2)$ une seule source
  - Bellman-Ford O(VE) une seule source
  - A* (e^n pire cas ; n log n meilleur cas) une seule paire
  - Roy-Floyd-Warshall $O(V^3)$ toutes les paires

Exemple de problème : Max flow



- Trouver le flux maximal entre une source et une cible (sink).
- Résolution par l'algorithme de Ford-Fulkerson $O(E^2V)$
- Exemples
  - Flux de voitures entre deux villes selon un réseau
  - EDF et les coupures possibles :)
  - Minimiser le temps d'attente aux remontées mécaniques

Problème : coloration de graphe



- Attribuer une couleur à chaque nœud de façon à ce qu'aucun nœud adjacent n'ait la même couleur
- Si on cherche le minimum alors on cherche le nombre chromatique du graphe
  – Utilisation en allocation de ressource minimale
  – Incompatibilité : des produits co-combustibles sont placés dans des wagons non voisins ==> de couleurs différentes

Et encore

- Test de planéité : un graphe est il planaire ?
  – Routage et circuits imprimés
- Arbre couvrant de poids minimal d’un graphe : sous-ensemble qui connecte tous les sommets dont la somme des coûts des arêtes est minimale
  – Réseaux informatiques (Spanning Tree Protocol)
- Ordonnancement et méthode PERT, MPM
  – Ordonnancement de tâches

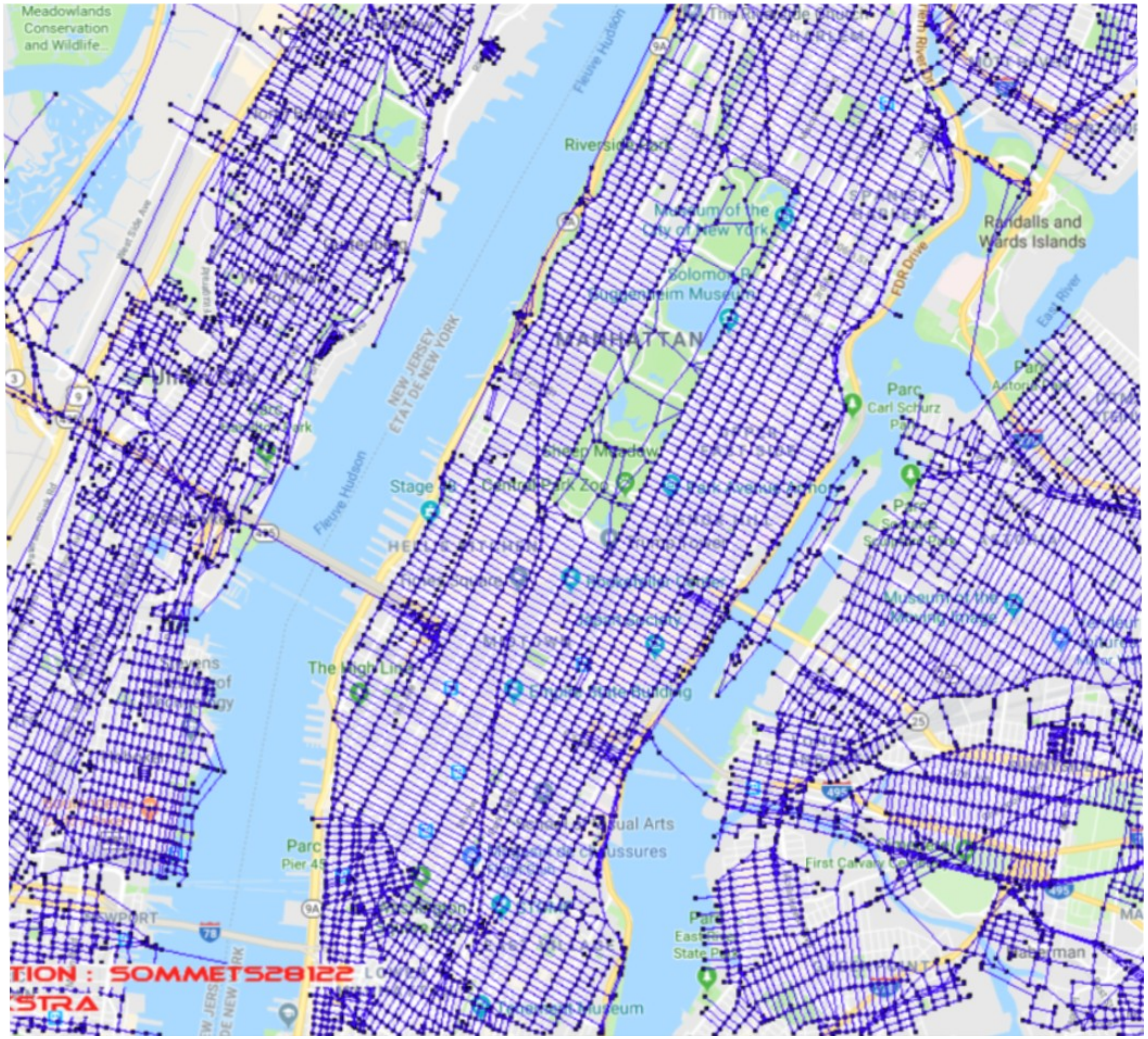
Mini projet 2019-2020

Projet 2019-2020 : PPC

- Plus court chemin dans
  - Un réseau routier
  - Le métro/RER parisien
  -
- Projet sur 4 séances
  - En binôme : bien s'organiser
  - Travail personnel entre séances indispensable
  - Pas de plagiat : ni interne, ni externe :-)
  - Des données réelles:16 Millions de sommets

Projet 2018 : PPC

- Réseau routier USA
  - Tous les arcs = toutes les routes
  - Pas de noms de sommets réels
  - Coordonnées GPS des sommets



Représentation du graphe

- Les sommets

  typedef struct {

  char* nom;

  double x,y ;

  listedge_t voisins;}

  vertex_t ;

- Les arcs

  typedef struct {

  int arrivee;

  • double cout } edge_t ;

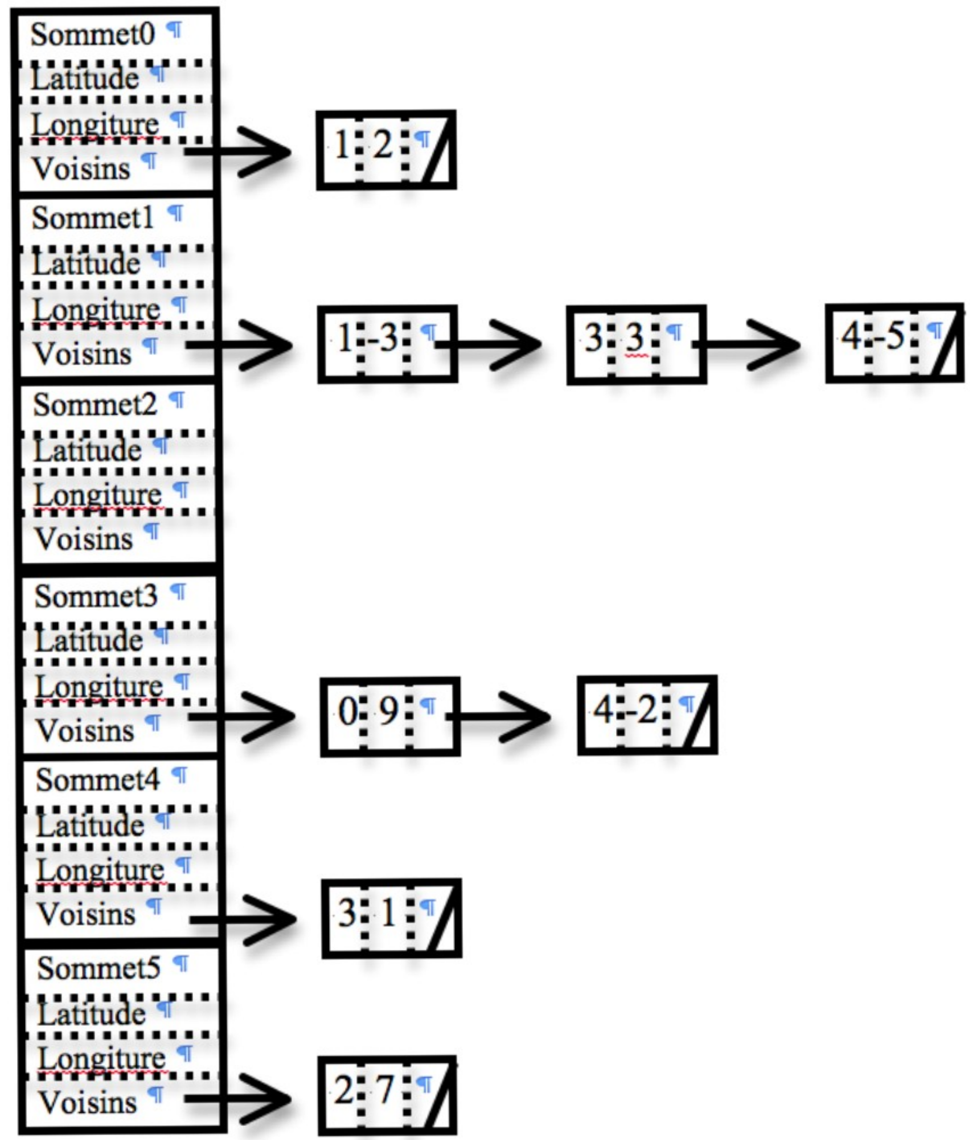
- Le graph :

  typedef struct{

  int size_vertex;

  int size_egdes;

  vertex_t* data; } graph_t;



Format du fichier

- Nb sommets, Nb Arcs
- Une ligne inutile
- Tous les sommets

  Numéro Latitude Longitude Ligne Nom
- Une ligne inutile
- Les arcs

  Départ Arrivée coût

src2 — bash — 58×25

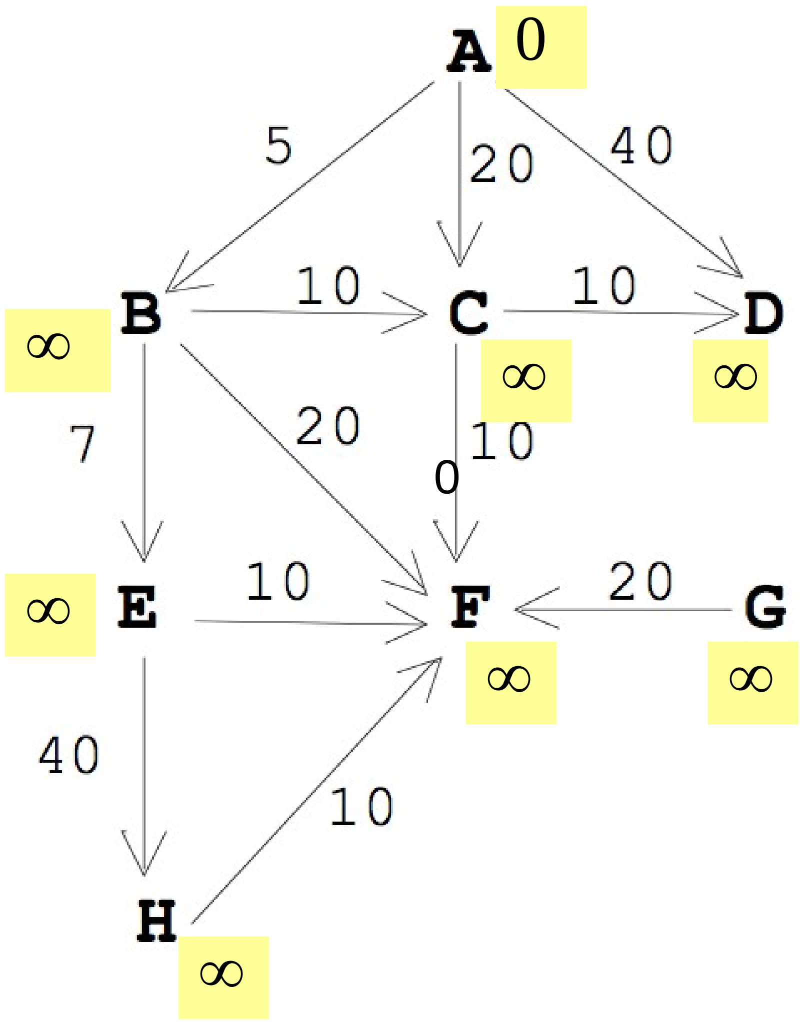
```
8 12
Sommets du graphe
0 0.5   0.95    M1      Aaa
1 0.1   0.7     M1      Baa
2 0.5   0.7     M1      Caa
3 0.9   0.7     M1      Daa
4 0.1   0.35    M1      Eaa
5 0.5   0.35    M1      Faa
6 0.9   0.35    M1      Gaa
7 0.1   0.05    M1      Haa
Arêtes du graphe : noeud1 noeud2 valeur
0 1 5
0 2 20
0 3 40
1 2 10
1 4 7
1 5 20
2 3 10
2 5 10
4 5 10
4 7 40
6 5 20
7 5 10
laptop-235:src2 desvignm$
laptop-235:src2 desvignm$
```

Dijkstra

- F = ensemble fermé des sommets qui restent à visiter ; initialement F={s} (s=start)
- O = ensemble ouvert des sommets dont on connaît leur plus court chemin du point de départ; initialement, O={}

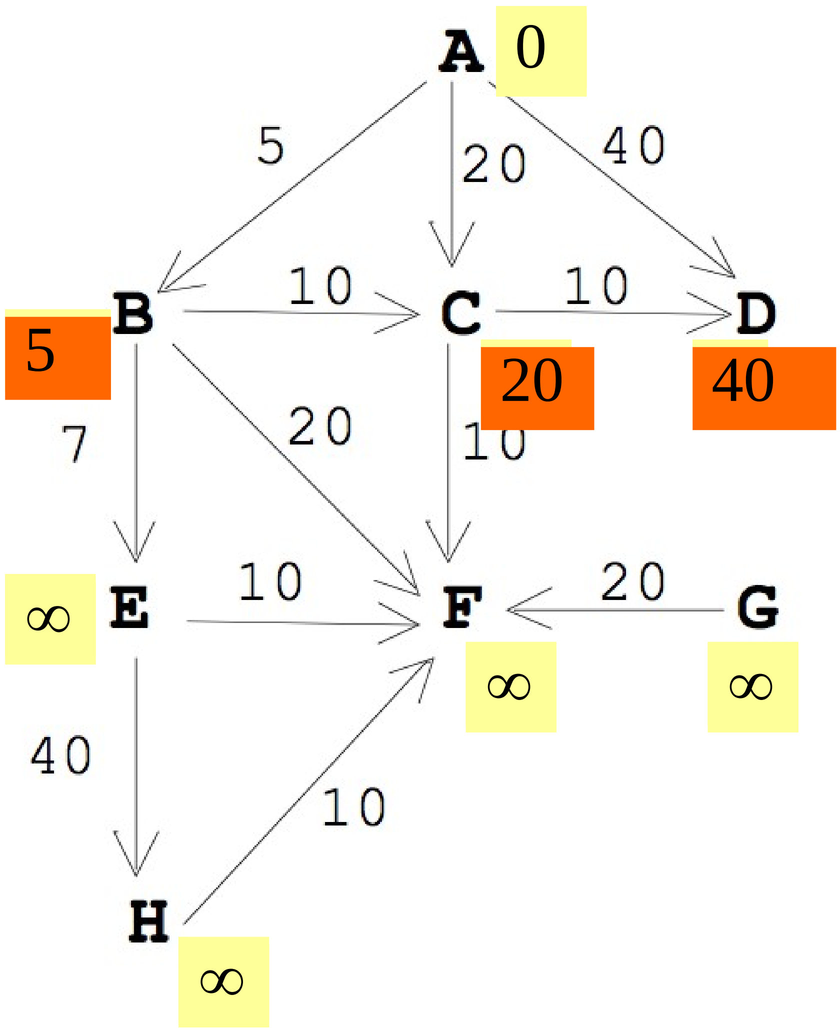
```
1.   début
2.      pour tous les sommets i de G[V,E] faire pcc[i]= INFINI ;
3.      pcc[s]= 0, // s est le nœud start, e est le nœud end
4.      O = {}  // sommet ouverts
5.      F = V // ensemble des sommets
6.      faire
7.            Sélectionner le sommet j de F de plus petite valeur pcc[j]
8.            F = F \ j                          // supprimer j de l'ensemble F
9.            O = O U  j                          // ajouter j à l'ensemble O
10.           pour tous les sommets k adjacents à j faire
                   // tous les successeurs de j
                   // c(j,k) est le coût pour aller de j à k
11.                 si  pcc[k] > pcc[j] + c[j][k]
12.                 alors // Passer par j est plus court pour aller de e en k
13.                        pcc[k] = pcc[j] + c[j][k];
14.                        pere[k]=j ;             /
15.                 fin si
16.           fin pour
17.     tant que e n'est pas dans O et que F n'est pas vide
18.  fin
```

Dijkstra



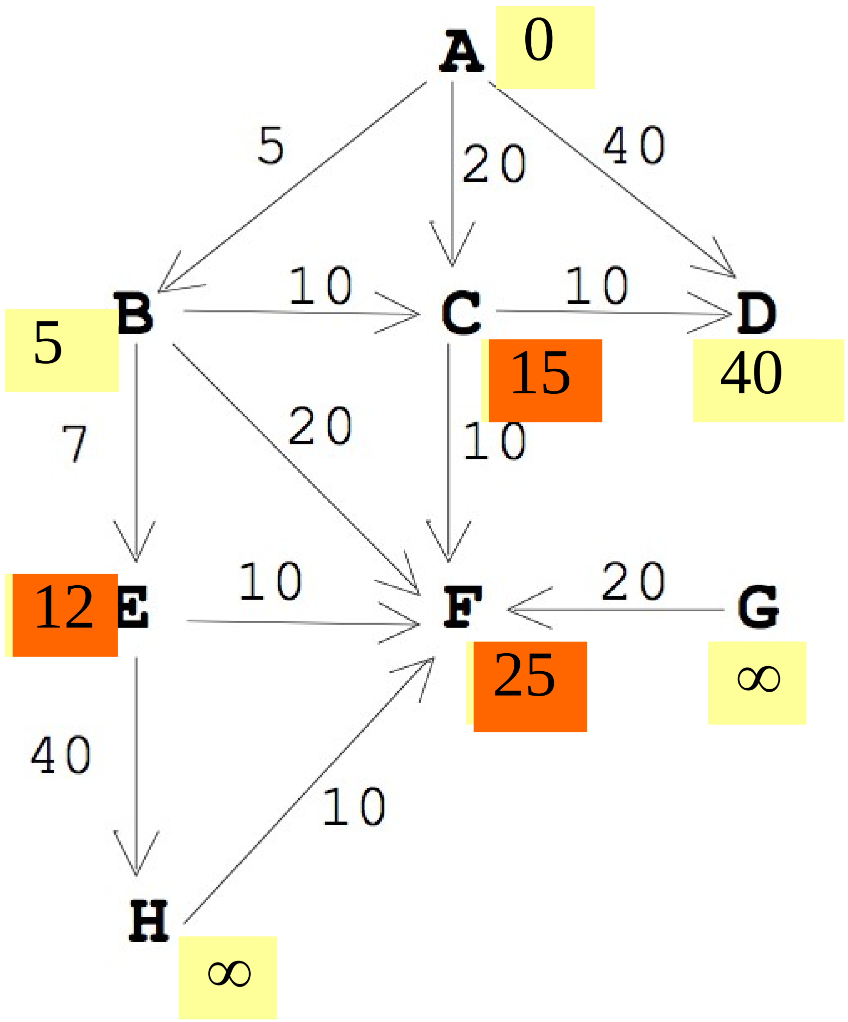
- Chemin de A à F
- O = {}
- F={A,B,C,D,E,F,G,H}
- PCC={0,∞,∞,∞,∞,∞,∞,∞}

Dijkstra Étape 1



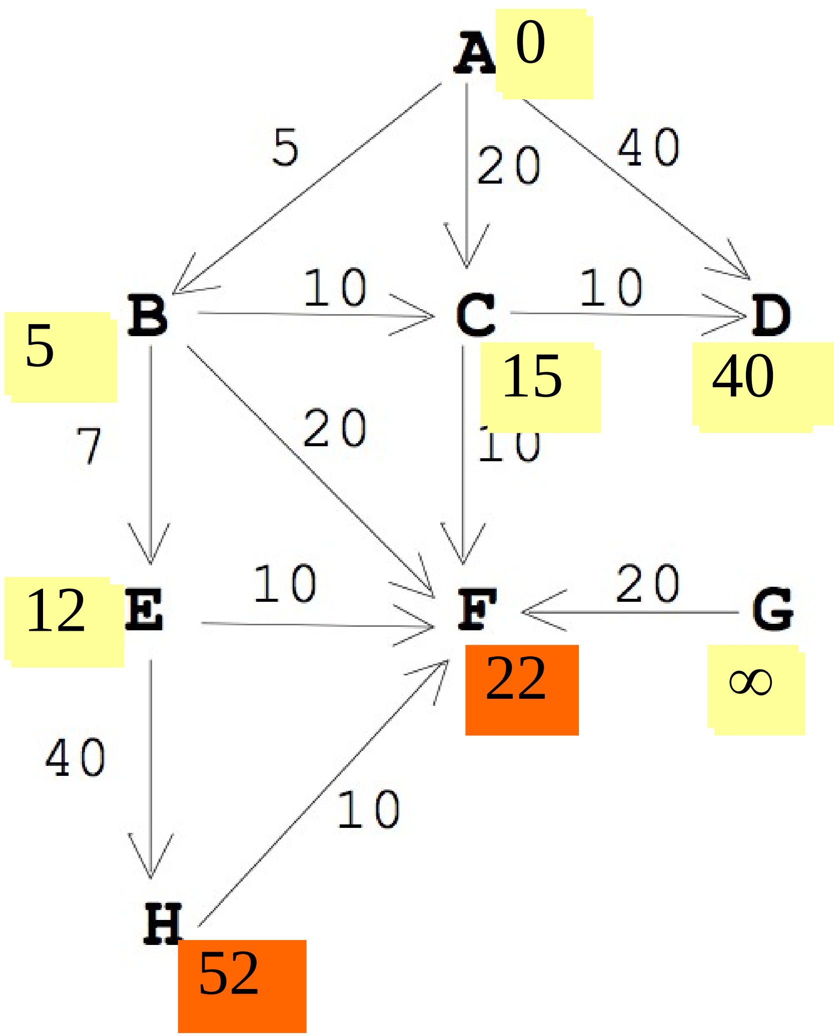
- PCC={0,∞,∞,∞,∞,∞,∞,∞}
- O = {}
- F={A,B,C,D,F,G,H}
- Recherche du meilleur sommet
  - non atteint de plus petit pcc
- ⇒ J =0 //Sommet A
- O = {A}
- F={B,C,D,E,F,G,H} // A est atteint avec son PCC
- Mise à jour des Voisins de A : B,C,D
  - Plus court chemin de A : pcc[0]=0
  - Pour B : pcc[0]+5 (5) < pcc[1] (∞)
  - Pour C : pcc[0]+20 (20) < pcc[2] (∞)
  - Pour D : pcc[0]+40 (40) < pcc[3] (∞)
- PCC={0,5,20,40,∞,∞,∞,∞}

Dijkstra Étape 2



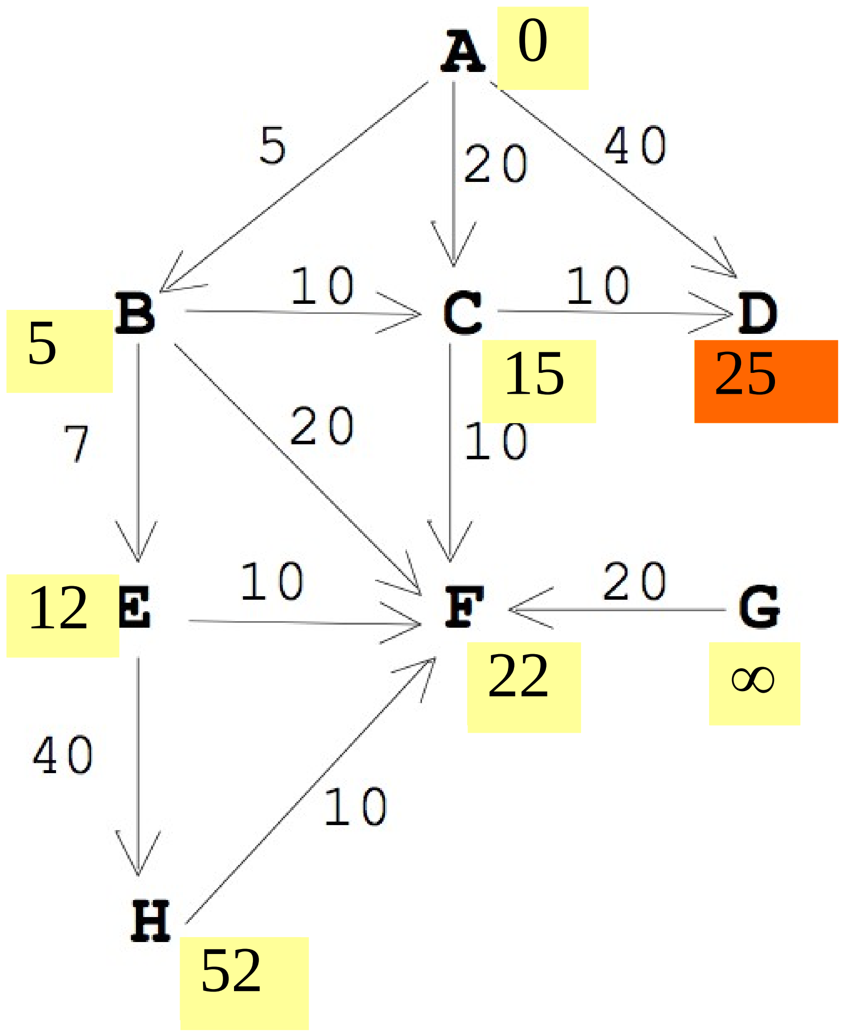
- PCC={0,5,20,40,∞,∞,∞}
- O = {A}
- F={B,C,D,E,F,G,H}
- Recherche du meilleur sommet
  - non atteint de plus petit pcc
- ⇒ J=1 //Sommet B
- O = {A,B}
- F={C,D,E, F,G,H}// B est aussi atteint
- Mise à jour des Voisins de B : C,F,E
  - Plus court chemin de B : pcc[1] = 5
  - Pour C : pcc[1] +10 (15) < pcc[2] (40)
  - Pour F : pcc[1] + 20 (25) < pcc[5] (∞)
  - Pour E : pcc[1] + 7 (12) < pcc[4] (∞)
- PCC={0,5,15,40,12,25,∞,∞}

Dijkstra Étape 3



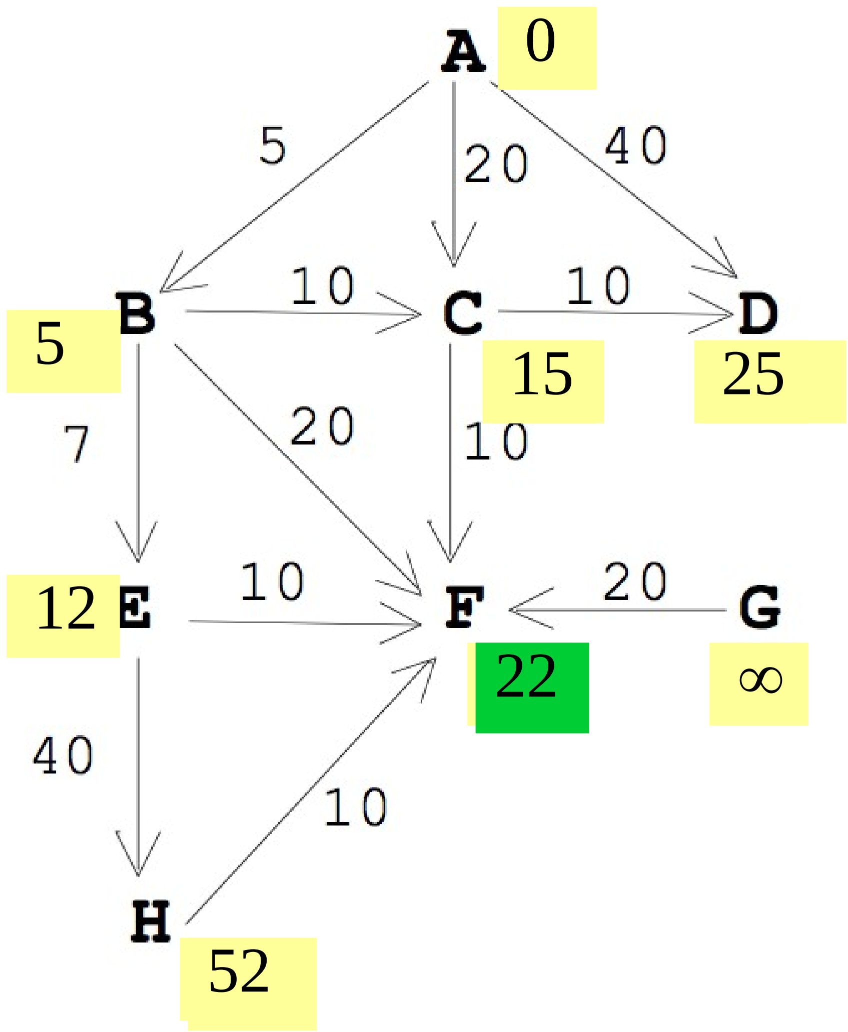
- PCC={0,5,15,40,12,25,∞,∞}
- O = {A,B}
- F={C,D,E, F,G,H}
- Recherche du meilleur sommet
  - – non atteint de plus petit pcc
- ⇒ J=4 //Sommet E
- O = {A,B,E}
- F={C,D,F,G,H} // E est aussi atteint
- Mise à jour des Voisins de E : F,H
  - – Plus court chemin de E : pcc[4] = 12
  - – Pour F : pcc[4] +10 (22) < pcc[5] (25)
  - – Pour H : pcc[4] + 40 (52) < pcc[7] (∞)
- PCC={0,5,15,40,12,22,∞,52}

Dijkstra Étape 4



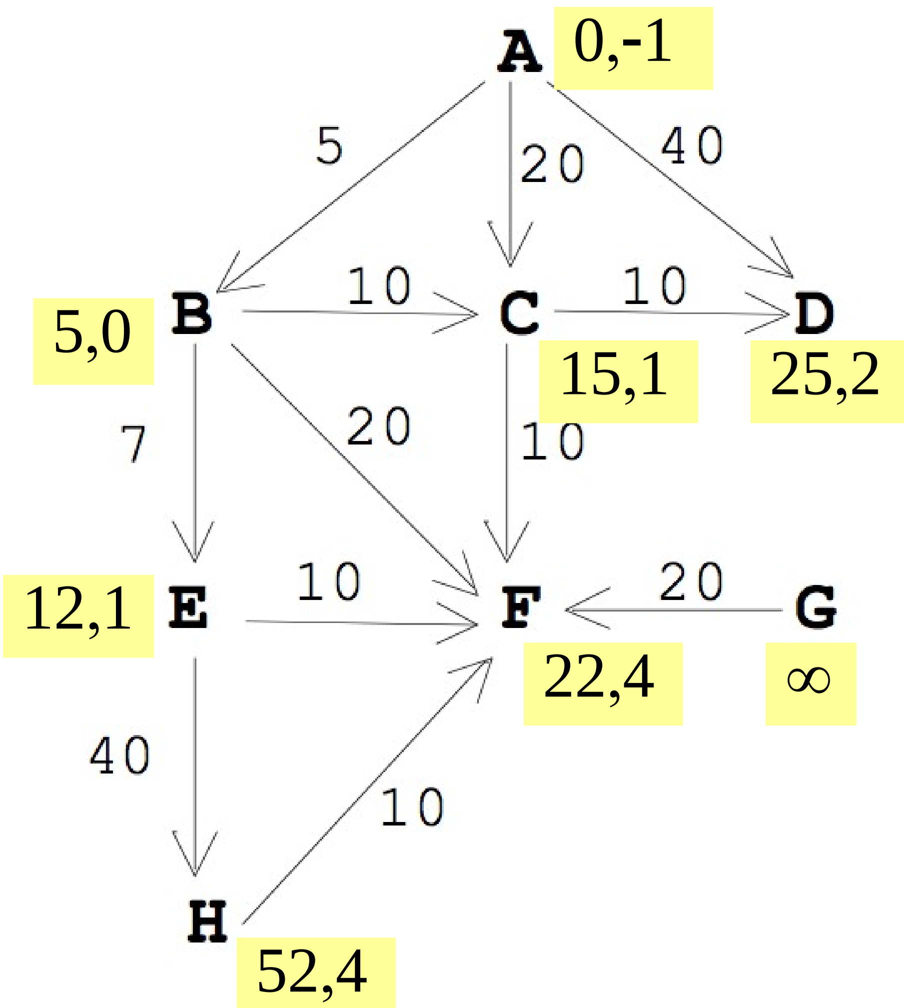
- PCC={0,5,15,40,12,22,∞,52}
- O = {A,B,E}
- A={C,D,F,G, H}
- Recherche du meilleur sommet
  - non atteint de plus petit pcc
- ⇒ J=4 //Sommet C
- O = {A,B,E,C}
- F={D,F,G,H} // C est aussi atteint
- Mise à jour des Voisins de C : D,F
  - Plus court chemin de C : pcc[2] = 15
  - Pour F : pcc[2] +10 (25) > pcc[5] (22)
  - Pas de changement pour F
  - Pour D : pcc[2] + 10 (25) < pcc[3] (40)
- PCC={0,5,15,25,12,22,∞,52}

Dijkstra Étape 5



- PCC={0,5,15,25,12,22,∞,52}
- O = {A,B,E,C}
- F= {D,F,G, H}
- Recherche du meilleur sommet
  – non atteint de plus petit pcc
- ⇒ J=5 //Sommet F
- O={A,B,E,C,F} // F est aussi atteint
- F = {D,G,H}
- Mise à jour des Voisins de F : Aucun
  – Plus court chemin de F : pcc[2] = 15
- FIN : F est atteint : le plus cours chemin de A à F est 22

Dijkstra : le chemin



- PCC={0,5,15,25,12,22,∞,52}
- Etape de mise à jour : ajout de l'information du sommet dont on vient quand on change la valeur du PCC d'un voisin d'un sommet atteint
- Exemple : attributs de B:5 et 0 :
  - 5 est la valeur du PCC
  - 0 est l'indice du sommet par lequel on arrive sur 5 pour obtenir le PCC
- Retrouver le chemin de A à F
  - Plus court chemin de F
    - Pcc[5] = 22
    - Pere[5] = E (4)
  - Chercher le père du sommet tant qu'il existe
    - Pere de F : E (4)
    - Pere de E : B (1)
    - Pere de B : A (0)
    - C'est le depart !!
  - Donc le chemin est A,B,E,F

Projet 2019-2020

- Définir les représentations
  - Comment représenter les coûts des pcc?
  - Comment représenter les pères ?
- Définir les fonctions utiles
  - Commencer par les fonctions les plus basiques
    - Listes, lecture du graphe, affichage du graphe
  - Tester au fur et à mesure
  - Tester d'abord sur des graphes simples
- Cas particulier du métro
  - Correspondance incluse dans le fichier
  - Gestion des sommets grâce à leur nom
    - Attention ; plusieurs sommets ont le même nom
    - Si on part de la gare du Nord, il y a 4 sommets de départ possibles
- Structure de données utiles pour optimiser
  - Tas de sommets : trouver le min de C
  - Table de hachage : trouver le numéro d'un sommet à partir d'une chaîne de caractères