

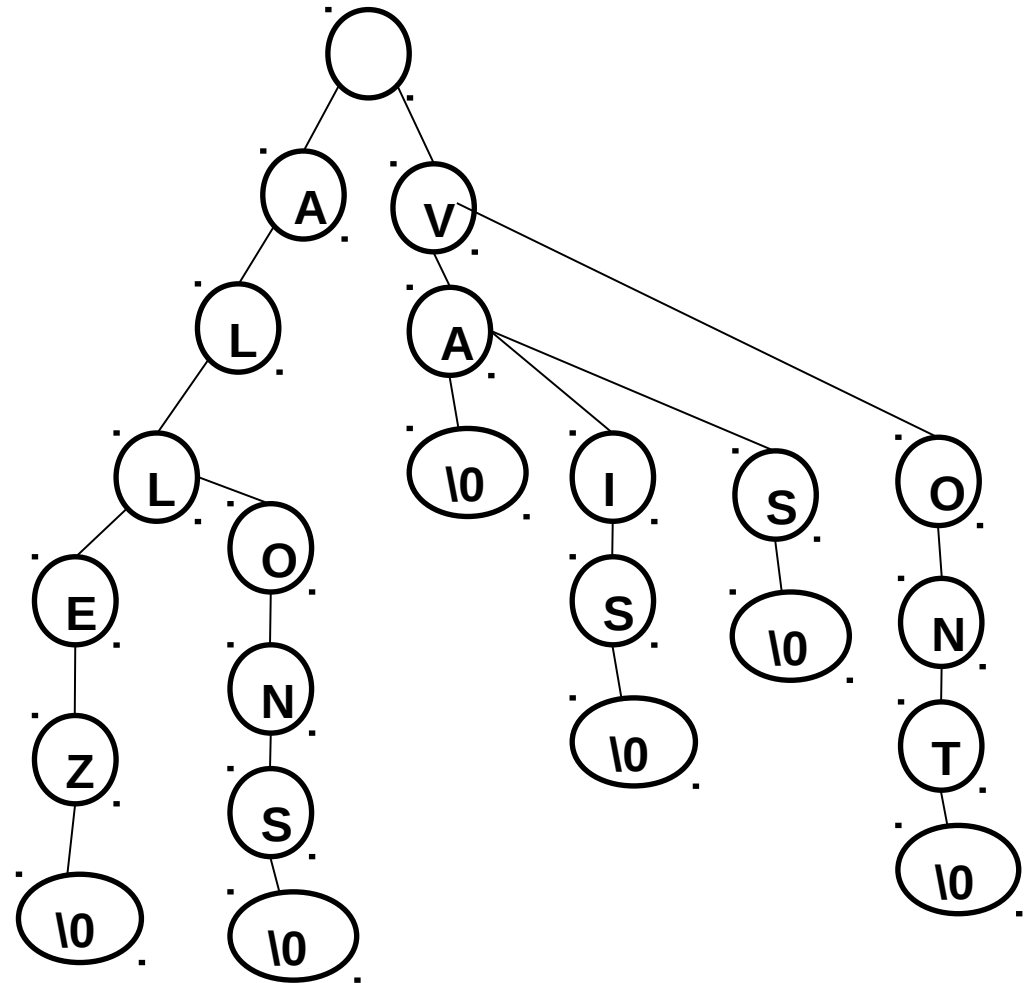
Graphes

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

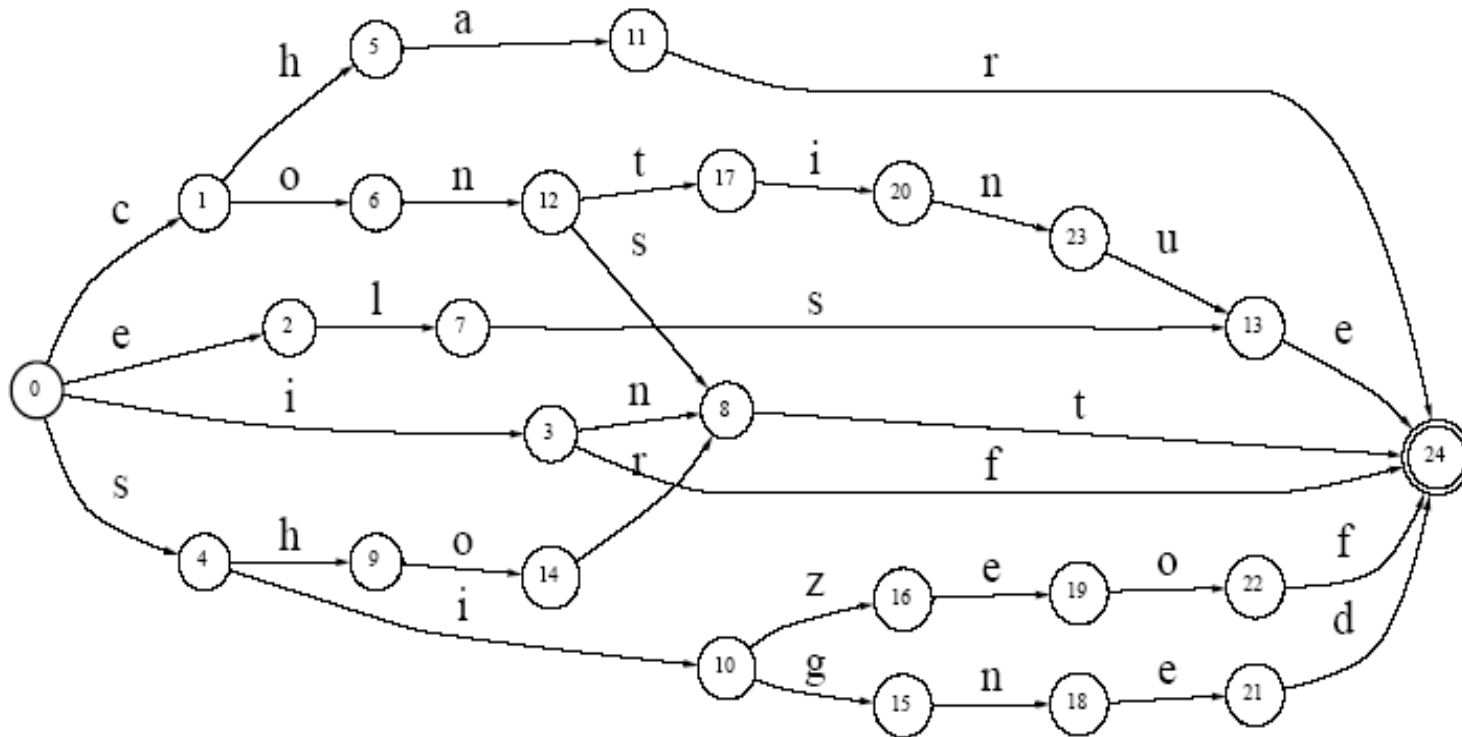
Arbres

- 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

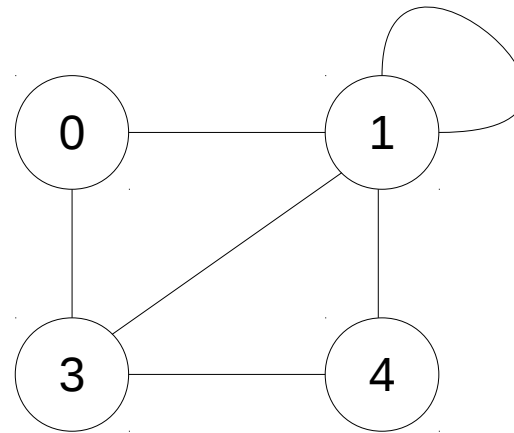


Graphe

- Théoriquement la meilleure implémentation pour coder un lexique
 - Optimisation en terme de taille de stockage
- Quels mots sont représentés ci-dessous ?



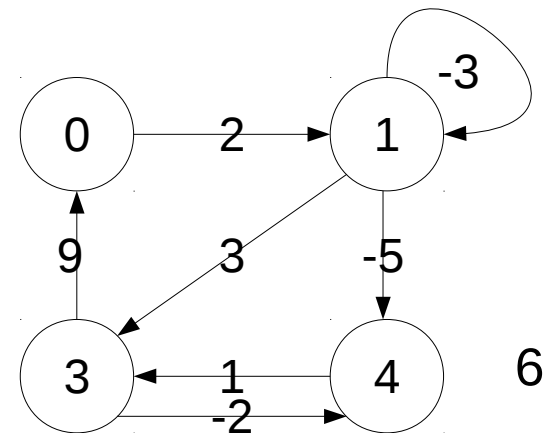
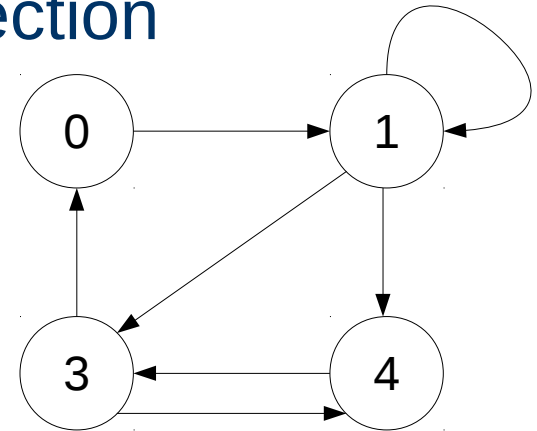
Définitions



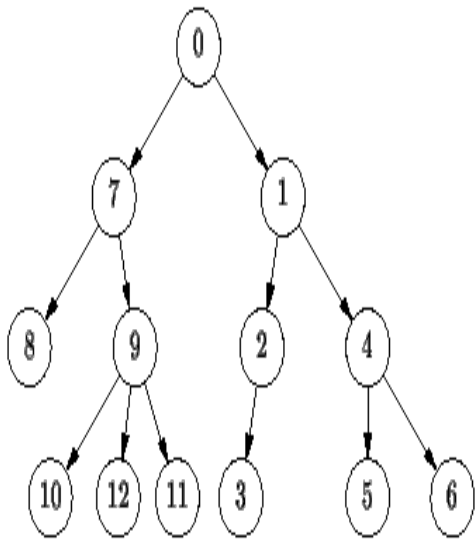
- Graphe : couple $G[X,A]$ où
 - X est un ensemble de nœuds ou sommets et
 - A est l'ensemble des paires de sommets reliés entre eux (arêtes du graphe ou «arc»)
- Chemin = séquence d'arêtes menant d'un sommet i à un sommet j
- Circuit = chemin dont les sommets de départ et d'arrivée sont identiques
- 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

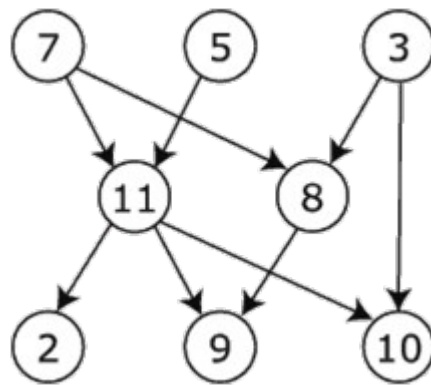
- Graphe dans lequel chaque arete a une direction associée
- Arc = arête orientée
- Graphe orienté pondéré:
 - Graphe étiqueté ou chaque arc a un coût associé
- valuation, coût = valeur numérique associée à un arc ou à un sommet



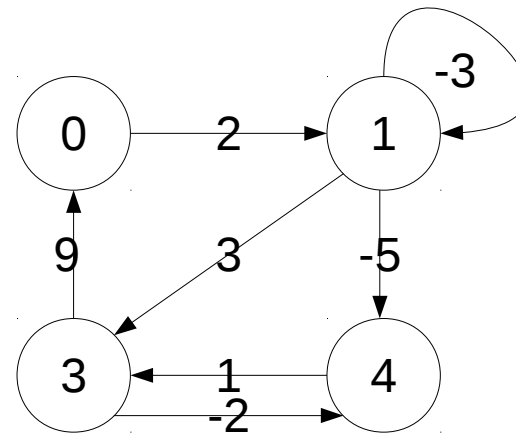
Types de graphe



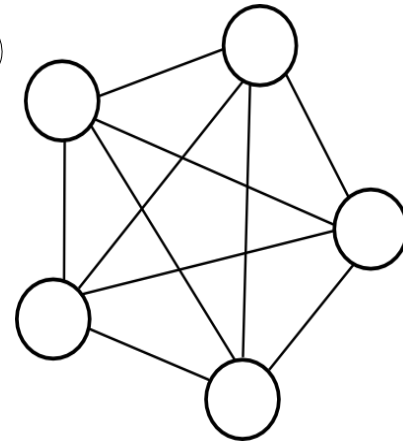
arbre



DAG



Graphe orienté



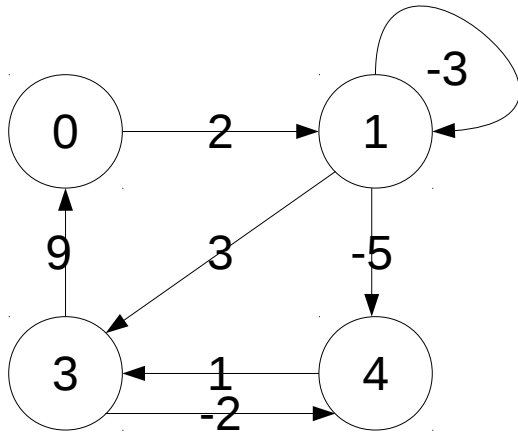
Graphe complet

Opérations

- $\text{voisin} : G \times y \rightarrow \text{Booléen}$
 - Teste si x et y sont voisins
- $\text{voisins} : G \times x \rightarrow \text{liste}$
 - Retourne la liste des voisins de x
- $\text{ajoute_noeud} : G \times x \rightarrow G$
 - Ajoute le nœud x si x n'est pas déjà présent
- $\text{Supprime_noeud} : G \times x \rightarrow G$
 - Supprime x de G si x présent
- $\text{ajoute_arc} : G \times x \times y \rightarrow G$
 - Ajoute un arc entre x et y si absent
- $\text{Supprime_arc} : G \times x \times y \rightarrow G$
 - Supprimer l'arc entre x et y si présent
- $\text{valeur_arc} : G \times x \times y \rightarrow \mathbb{N}$
 - Retourne la valeur de l'étiquette de l'arc entre x et y ; $+\infty$ sinon

Graphe : représentation matricielle

- Tout graphe $G = (X, A)$ peut être représenté par une matrice d'adjacence
- $m_{i,j} = \text{val}(a_i, a_j)$ si $(a_i, a_j) \in A$, ∞ sinon



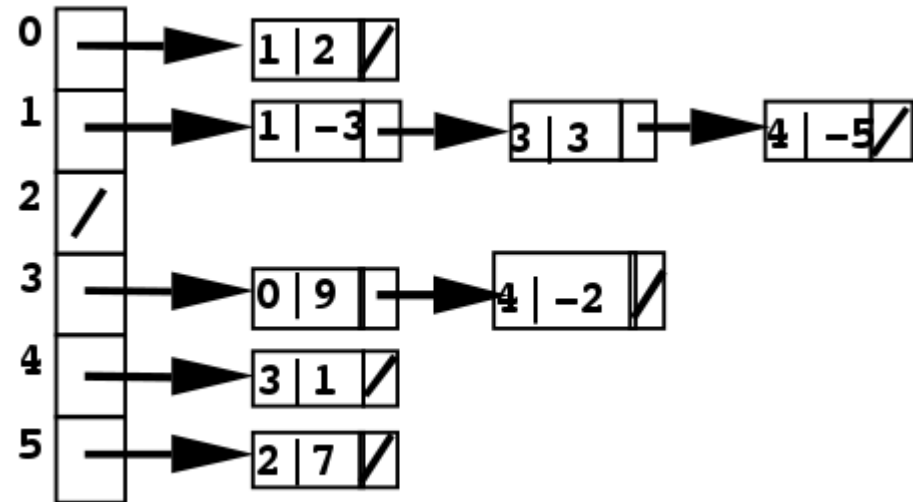
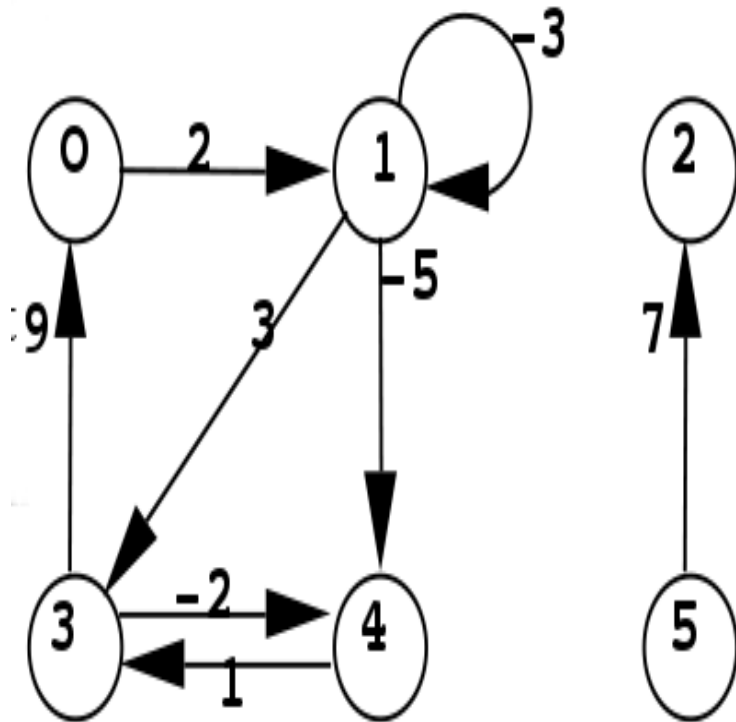
Arrivée →	0	1	3	4
0	∞	2	∞	∞
1	∞	-3	3	-5
3	9	∞	∞	-2
4	∞	∞	1	∞

Graphe : représentation matricielle

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

- Fortement déconseillé pour les graphes creux !
- Meilleur choix si la rapidité d'accès est cruciale ou si le graphe est très dense (\rightarrow complet)

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 $\rightarrow O(|X|+|A|)$
 - Coût d'ajout d'un nœud $\rightarrow O(1)$
 - Coût d'ajout d'un arc $\rightarrow O(1)$
 - Coût suppression d'un nœud $\rightarrow O(|E|)$
 - Coût suppression d'un arc $\rightarrow O(|E|)$
-
- Parfait pour représenter les graphes creux !
 - Meilleur compromis de complexité

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)

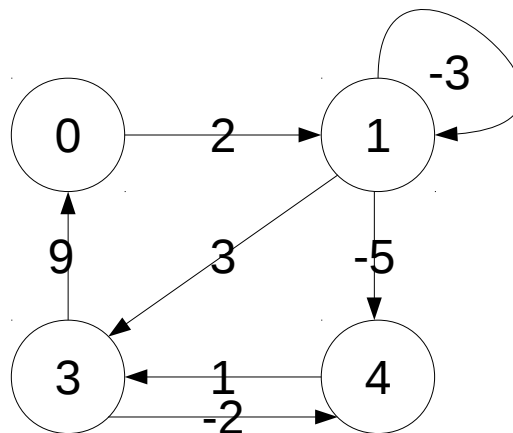
■ Parcours ordonnés selon les arêtes :

- Problème de cycle (comment être sûr 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 X)`
 - Si X n'a pas encore été parcouru alors
 - Noter X parcouru
 - Pour tout voisin V de X :
 - `parcours_profondeur(G,V)`

- Exemple:

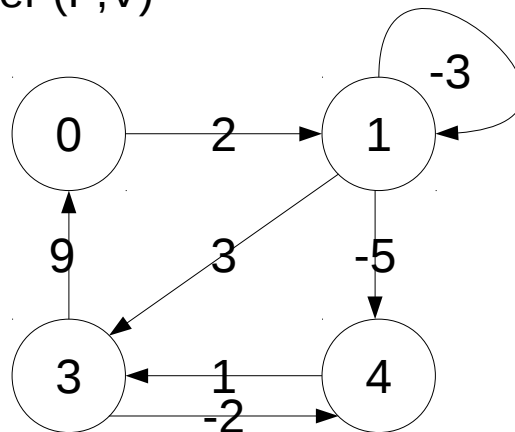


- Même algorithme que pour les arbres

Parcours en largeur d'abord

- Stockage des frères par file d'attente
- `parcours_largeur(graphe G, sommet X)`
 - Noter X parcouru
 - File $F \leftarrow \{X\}$
 - Tant $F \neq \emptyset$
 - $N = \text{défiler}(F)$
 - Action (N) // exemple Afficher N.
 - Pour tout voisin V de N :
 - Si V non parcouru alors
 - » Noter V parcouru
 - » $F = \text{enfiler}(F, V)$

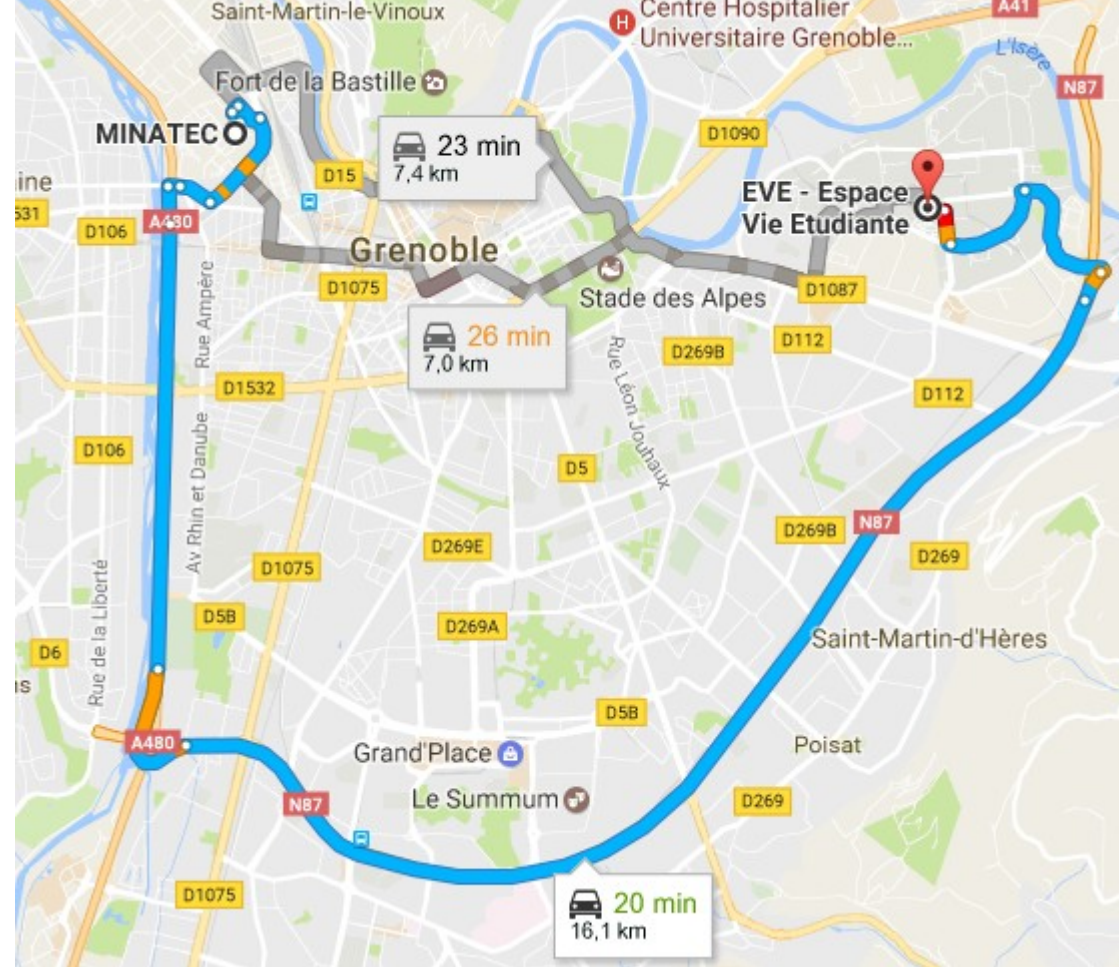
■ Exemple:



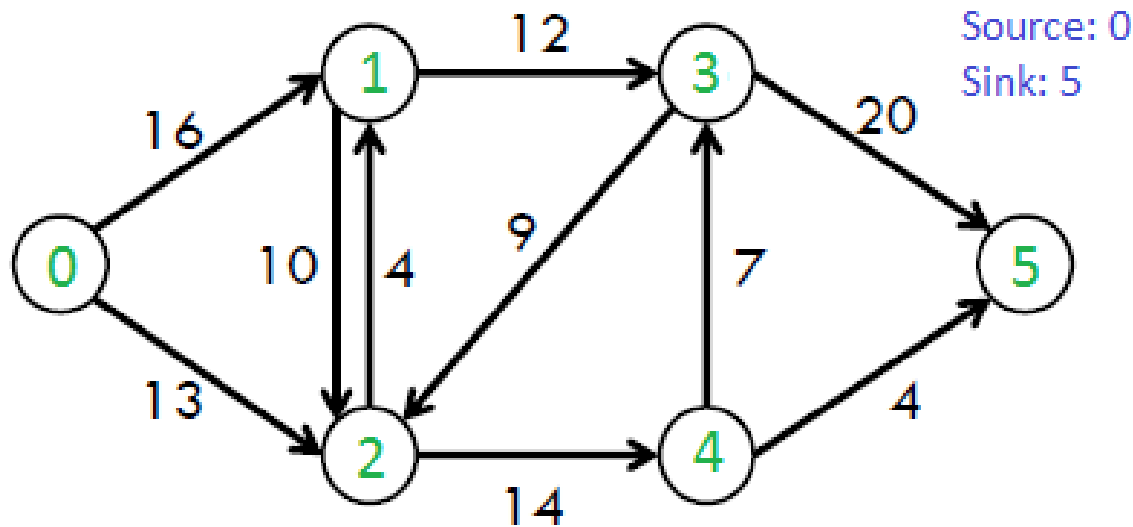
- Même algorithme que pour les arbres

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 $O(VE)$ une seule source
 - A^* (e^n pire cas ; $n \log n$ meilleur cas) une seule paire
 - Floyd–Warshall $O(V^3)$ toutes les paires



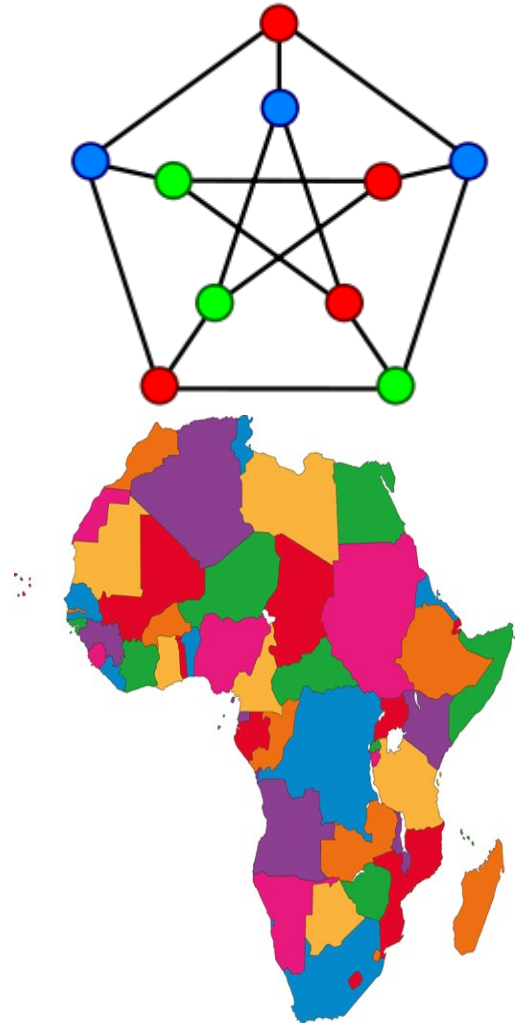
Exemple de problème : Max flow



- Trouver le flux maximal entre une source et une cible (sink).
- Résolution par l'algorithme de Ford-Fulkerson
- Exemples
 - Flux de voitures entre deux villes selon un réseau
 - EDF et les coupures possibles :)
 - Optimiser le temps d'attente des 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 poids des arêtes est minimale
 - Réseaux informatiques
- Ordonnancement et méthode PER, MPM
 - Ordonnancement des tâches

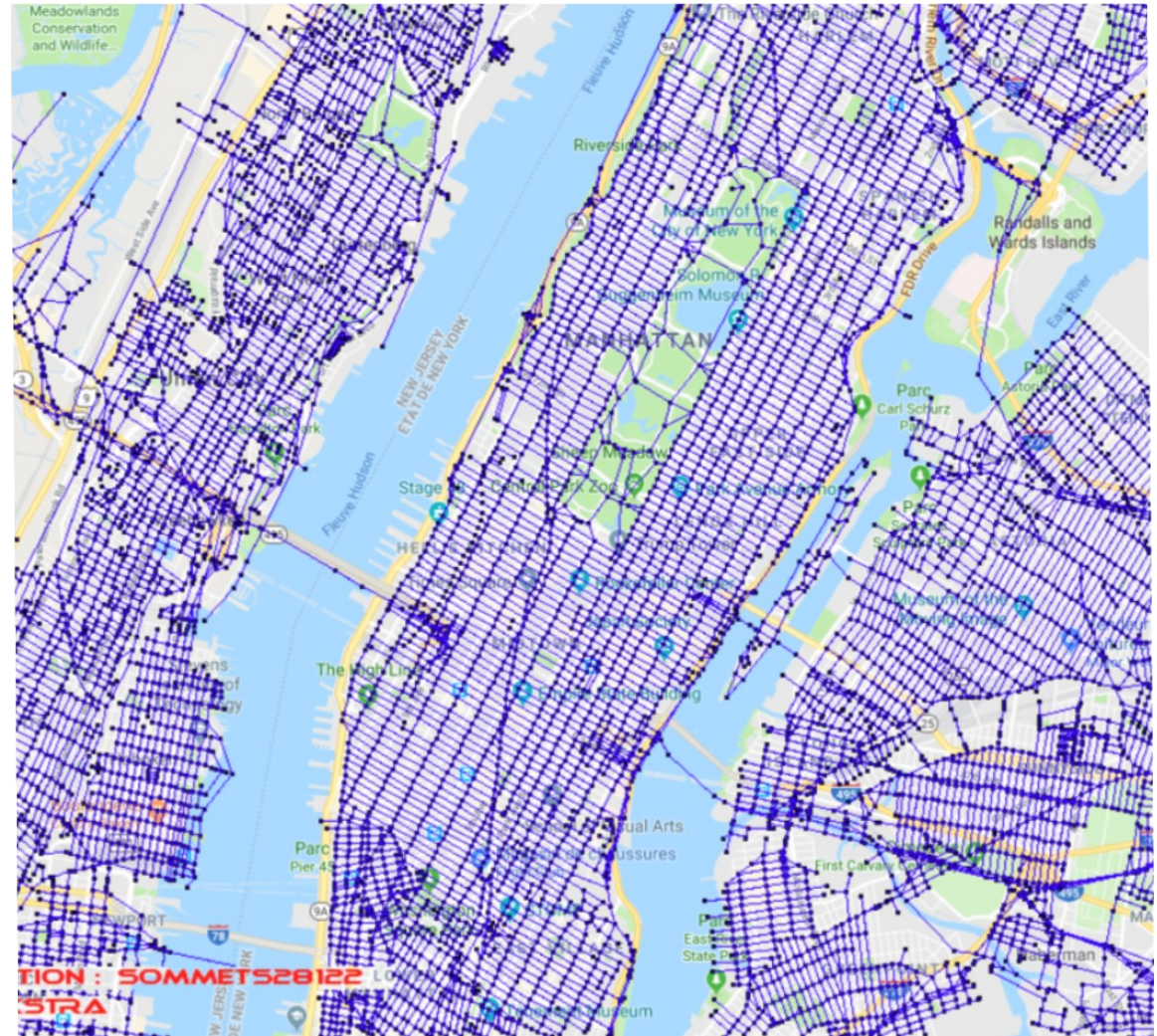
Projet 2018 : PPC

- Plus court chemin dans
 - Un réseau routier
 - Le métro/Rer parisien
 -
- Projet sur 4 seances
 - En binome : 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

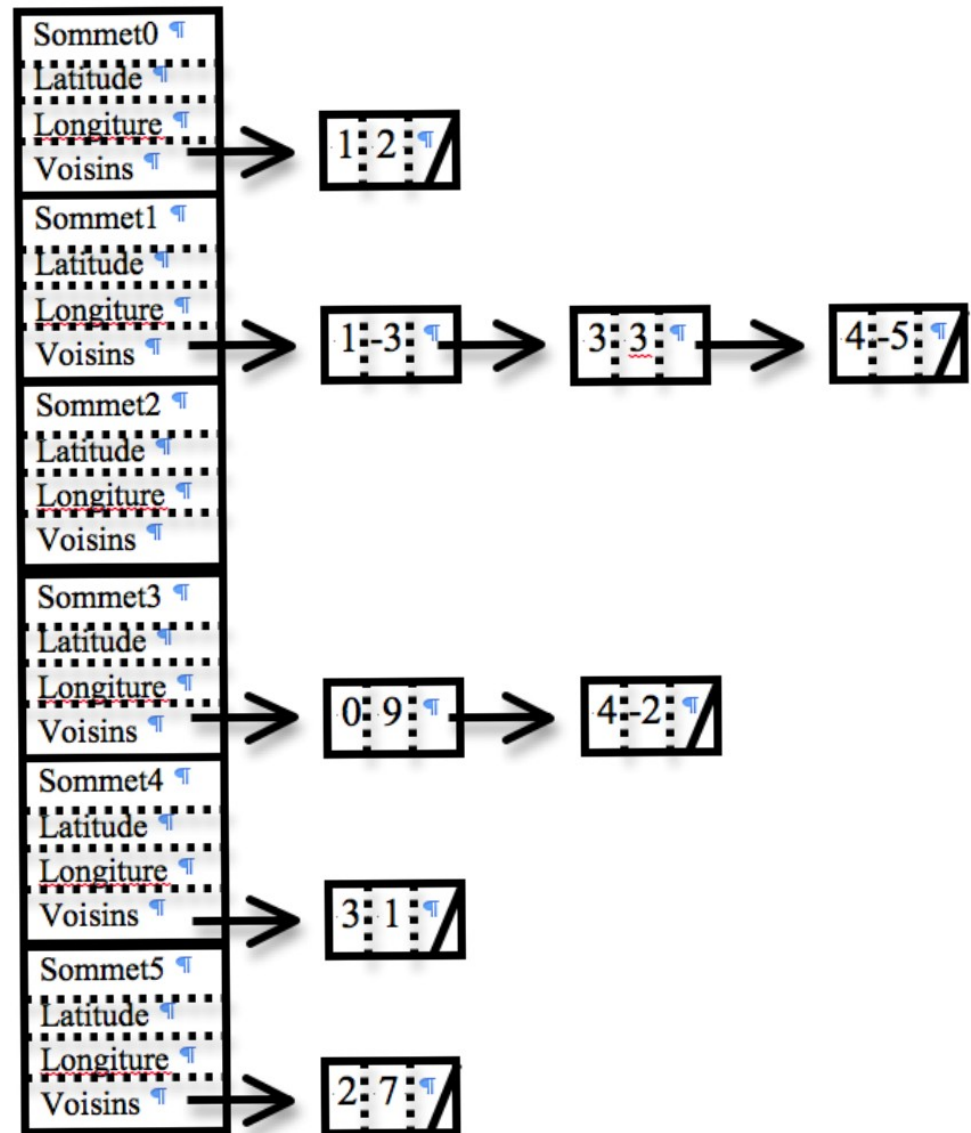
```
typedef struct {  
    char* nom;  
    double x,y ;  
    L_ARC voisins;}  
T_SOMMET ;
```

■ Les arcs

```
typedef struct {  
    int arrivee;  
    double cout } T_ARC ;
```

■ Les listes de successeurs :

```
typedef struct lsucc {  
    T_ARC val;  
    struct lsucc* suiv ; }*  
L_ARC;
```



Format du fichier

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

Numero Latitude Longitude
Ligne Nom

- Une ligne inutile
- Les arcs

Depart Arrivée cout

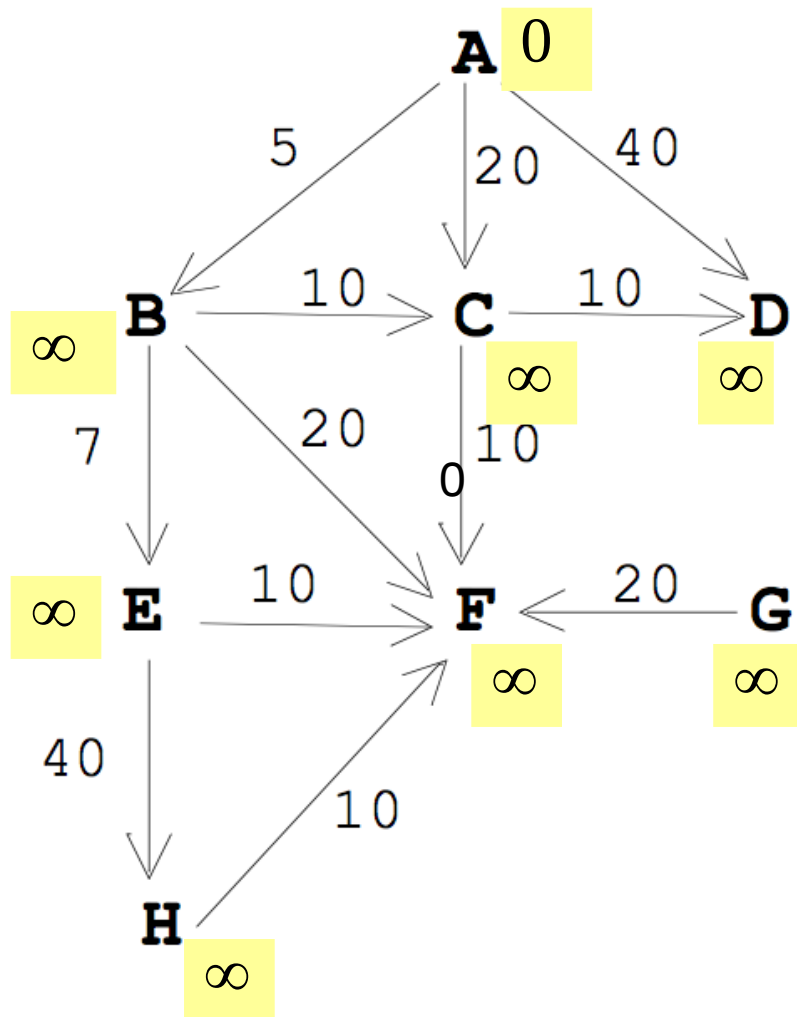
```
src2 — bash — 58x25
8 12
Sommets du graphe
0 0.5 0.95 M1 Aaa
1 0.1 0.7 M1 Baa
2 0.5 0.7 M1 Caa
3 0.9 0.7 M1 Daa
4 0.1 0.35 M1 Eaa
5 0.5 0.35 M1 Faa
6 0.9 0.35 M1 Gaa
7 0.1 0.05 M1 Haa
Arêtes du graphe : noeud1 noeud2 valeur
0 1 5
0 2 20
0 3 40
1 2 10
1 4 7
1 5 20
2 3 10
2 5 10
4 5 10
4 7 40
6 5 20
7 5 10
laptop-235:src2 desvignm$
laptop-235:src2 desvignm$
```

Dijkstra

- C = sommets d qui restent à visiter ; initialement $C=X$
- S = sommets dont on connaît leur plus court chemin au point de départ; initialement, $S=\{\}$

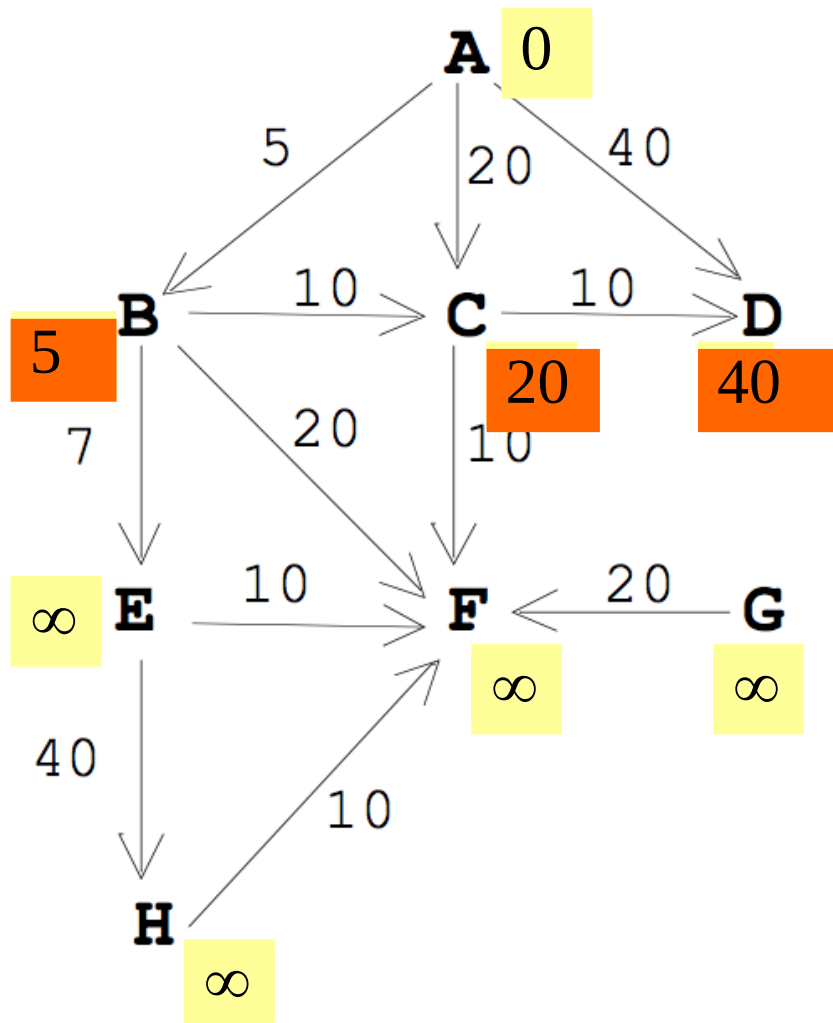
```
1.  début
2.    pour tous les sommets i de G[X,A] faire pcc[i]= INFINI ;
3.    pcc[d]= 0
4.    S = {}
5.    C = X
6.    faire
7.        Sélectionner le sommet j de C de plus petite valeur pcc[j]
8.        C = C \ j                // supprimer j de l'ensemble C
9.        S = S U j                // ajouter j à l'ensemble S
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 a en k
13.           pcc[k] = pcc[j] + c[j][k];
14.           pere[k]=j ;           /
15.       fin si
16.   fin pour
17.   tant que a n'est pas dans S et pcc[j]!=INFINI
18. fin
```


Dijkstra



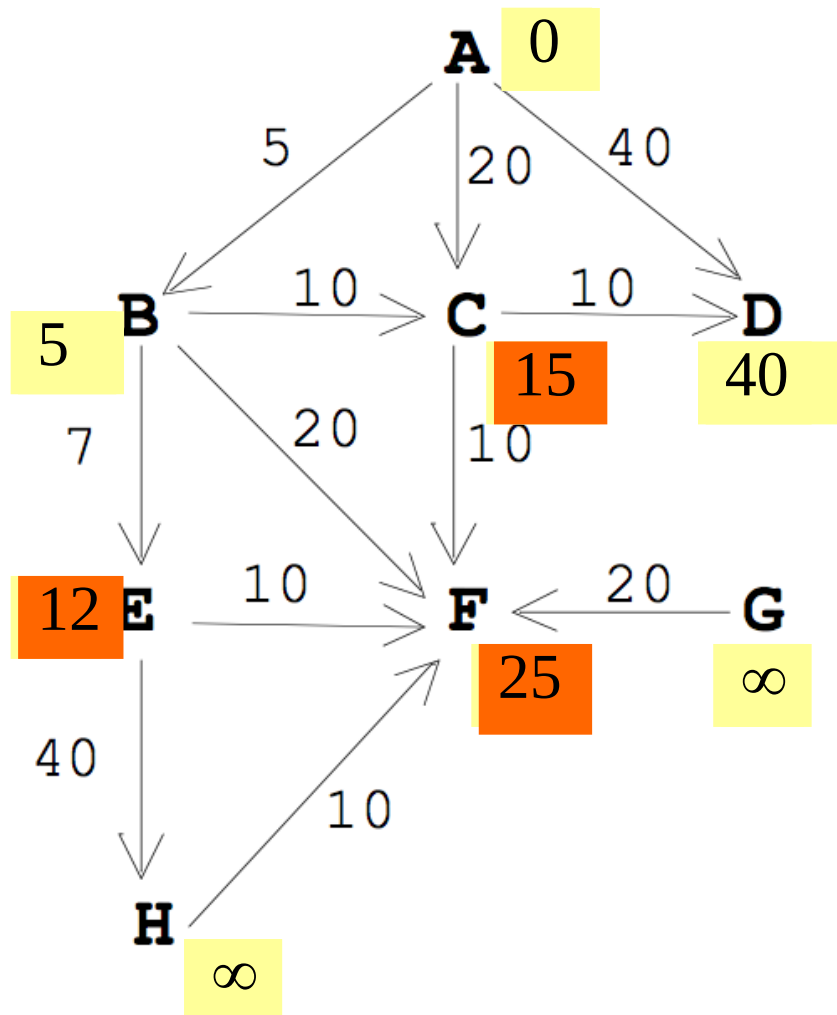
- Chemin de A à F
- $C = \{A, B, C, D, E, F, G\}$
- $S = \{\}$
- $PCC = \{0, \infty, \infty, \infty, \infty, \infty, \infty\}$

Dijkstra Etape 1



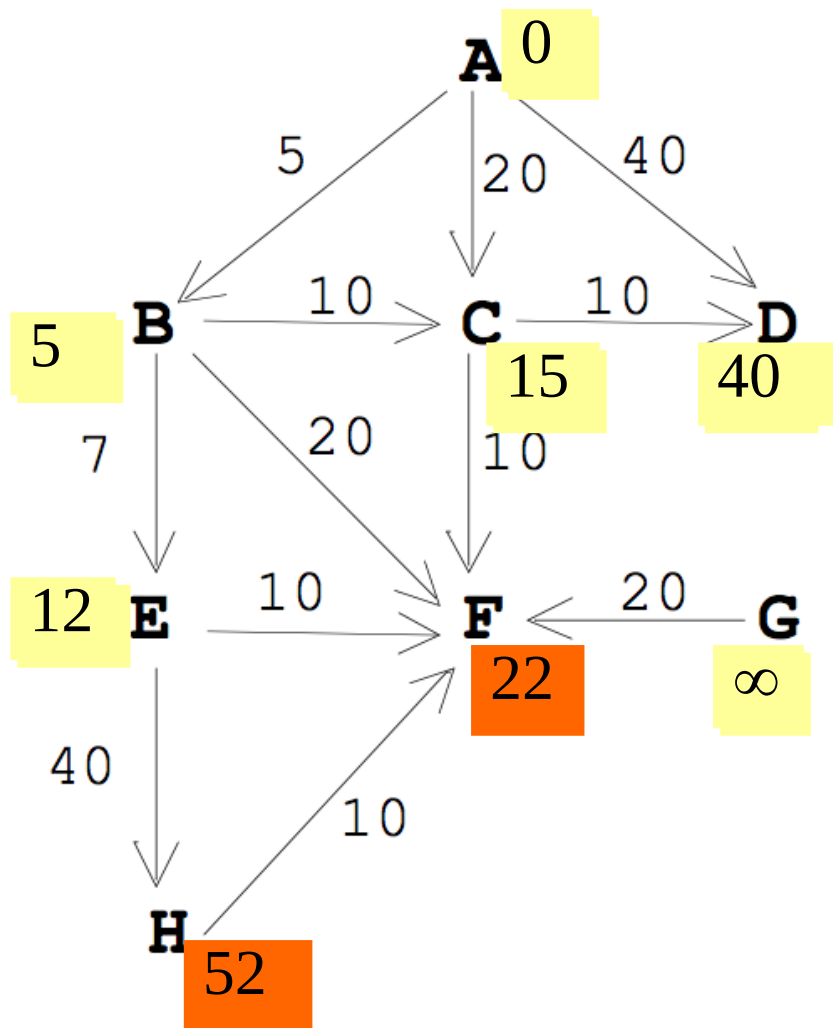
- $PCC = \{0, \infty, \infty, \infty, \infty, \infty, \infty, \infty\}$
- $C = \{A, B, C, D, E, F, G, H\}$
- $A = \{\}$
- Recherche du meilleur sommet
 - non atteint de plus petit pcc
- $\Rightarrow J=0$ // Sommet A
- $C = \{B, C, D, E, F, G, H\}$
- $S = \{A\}$ // A est atteint avec son PCC
- Mise à 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) < $pcc[3]$ (∞)
- $PCC = \{0, 5, 20, 40, \infty, \infty, \infty, \infty\}$

Dijkstra Etape 2



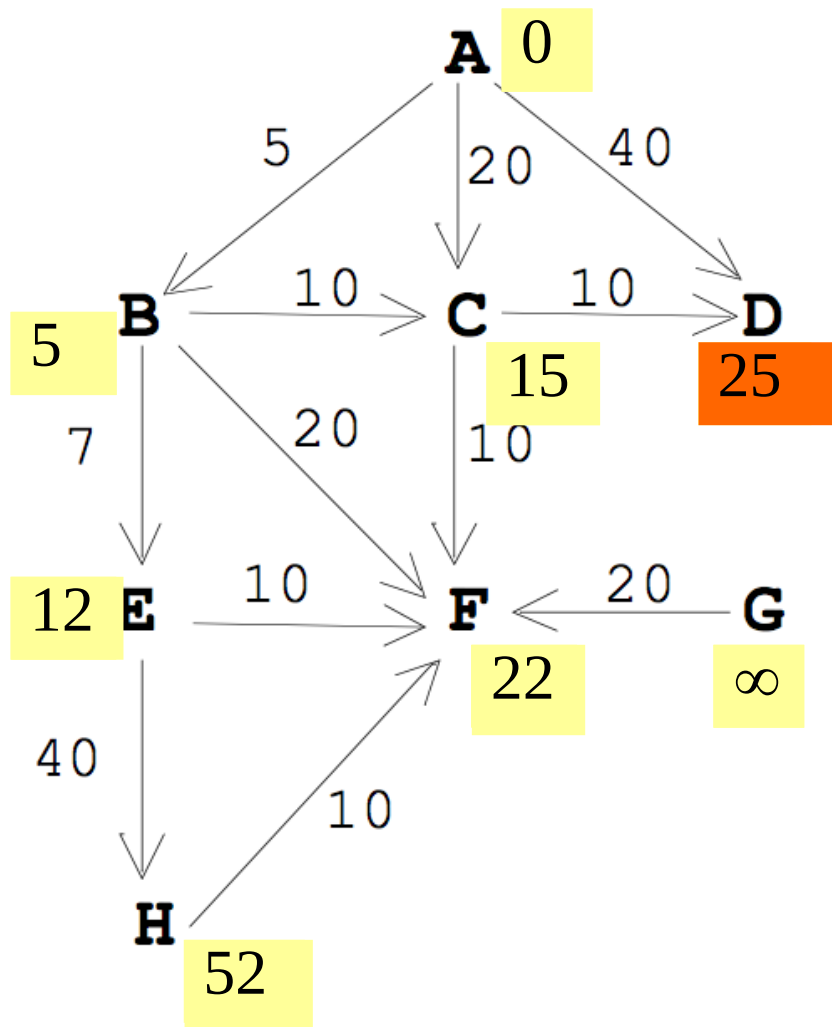
- $PCC = \{0, 5, 20, 40, \infty, \infty, \infty\}$
- $C = \{B, C, D, E, F, G, H\}$
- $A = \{A\}$
- Recherche du meilleur sommet
 - non atteint de plus petit pcc
- $\Rightarrow J=1$ // Sommet B
- $C = \{C, D, E, F, G, H\}$
- $S = \{A, B\}$ // B est aussi atteint
- Mise à jour des Voisins de B : C, F,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

Dijkstra Etape 3



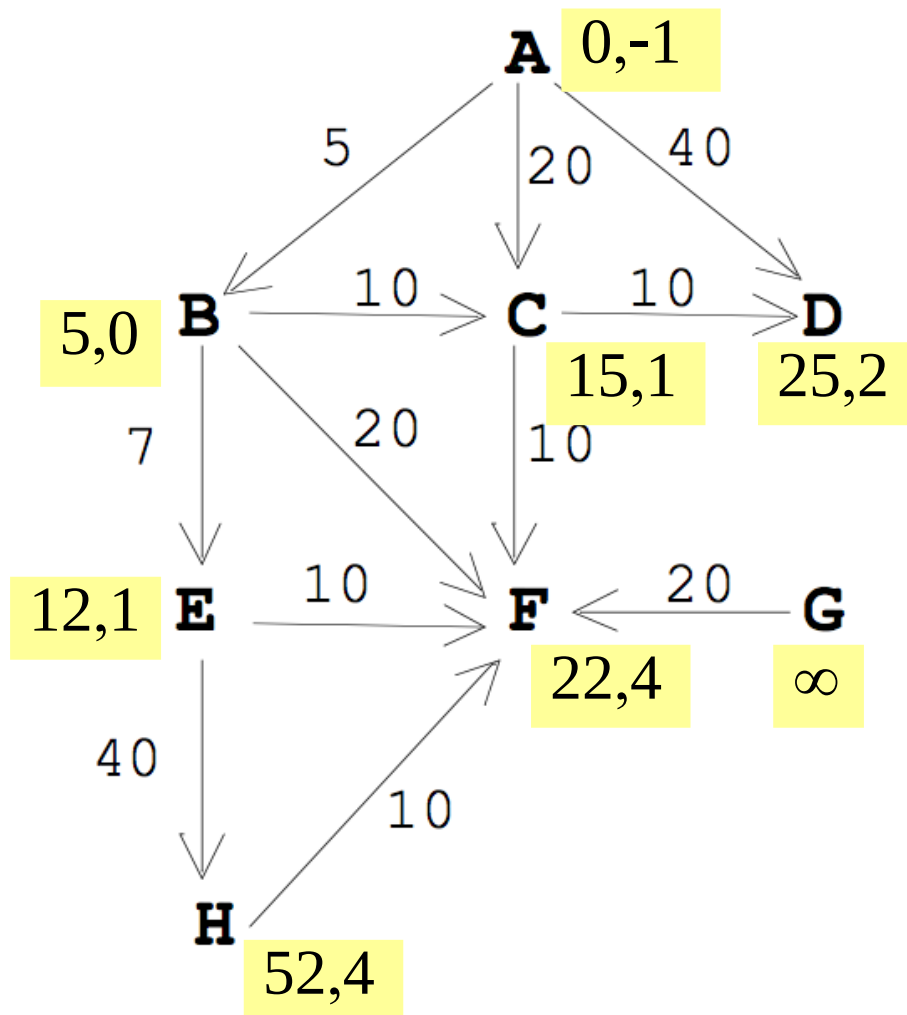
- $PCC = \{0, 5, 15, 40, 12, 25, \infty, \infty\}$
- $C = \{C, D, E, F, G, H\}$
- $A = \{A, B\}$
- Recherche du meilleur sommet
 - non atteint de plus petit pcc
- $\Rightarrow J=4$ // Sommet E
- $C = \{C, D, F, G, H\}$
- $S = \{A, B, E\}$ // E est aussi atteint
- Mise à jour des Voisins de E : F, H
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

Dijkstra Etape 4



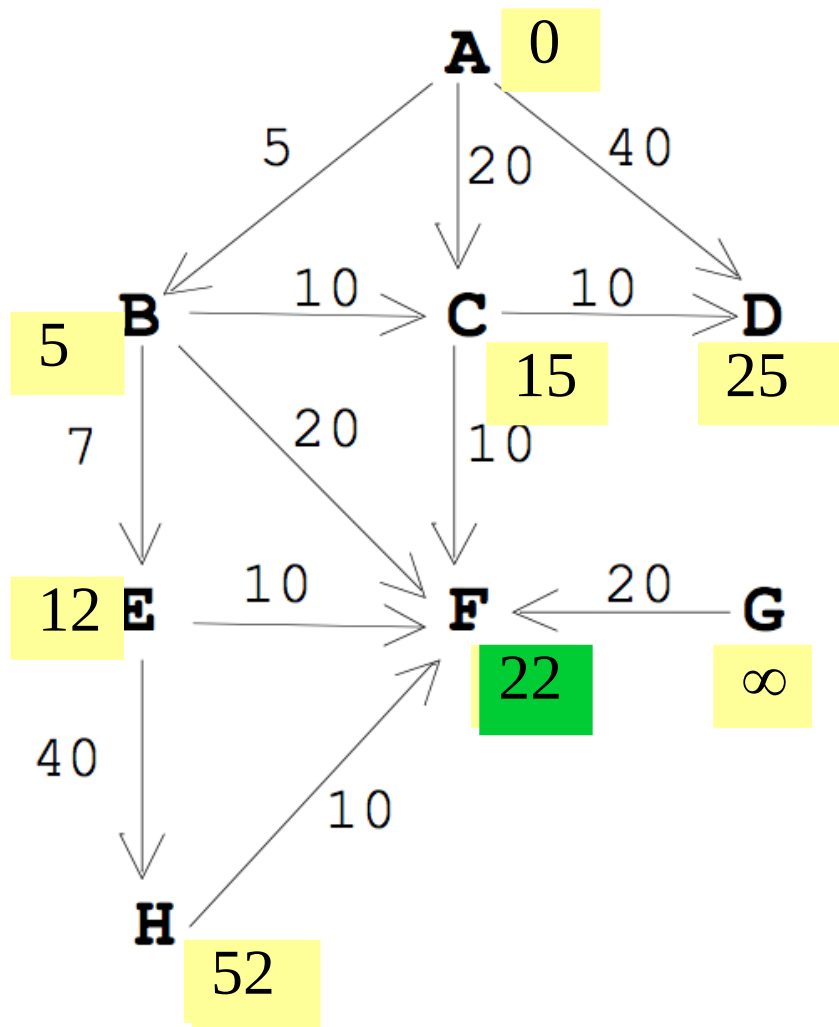
- $PCC = \{0, 5, 15, 40, 12, 22, \infty, 52\}$
- $C = \{C, D, F, G, H\}$
- $A = \{A, B, E\}$
- Recherche du meilleur sommet
 - non atteint de plus petit pcc
- $\Rightarrow J=4$ // Sommet C
- $C = \{D, F, G, H\}$
- $S = \{A, B, E, C\}$ // C est aussi atteint
- Mise à jour des Voisins de C : D, F
 - Plus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

Dijkstra : le chemin



- $PCC = \{0, 5, 15, 25, 12, 22, \infty, 52\}$
- 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
 - $Pcc[5] = 22$
 - $Pere[5] = 4$
 - Chercher le pere 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

Dijkstra Etape 5



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

Dijkstra

- Définir les représentations
 - Comment représenter les couts ?
 - Comment représenter les peres ?
- 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 metro
 - 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 numero d'un sommet à partir d'une chaine de caractères