

Cours 2

# Graphes valués

- Question :
    - ▶ Adopteriez vous la même stratégie quand vous prenez le métro ?
    - ▶ Quand vous voyagez en train ?
    - ▶ Quand vous prenez votre voiture ?
    - Nous allons mettre des poids sur les arêtes
      - ▶ des coûts de transit (ex. payage)
      - ▶ des distances kilométriques
      - ▶ le temps
  - On parle de **graphes valués (pondérés)**

# Plus courts chemins (1/2)

- Définitions :
  - Dans un graphe valué, le **poid**  $c(p)$  d'un chemin  $p$  est la somme des poids des arêtes le long du chemin
  - Comment calculer le chemin de poids minimum (problème d'optimisation) ?
- Propriétés :
  - Un plus court chemin entre 2 sommets  $s$  et  $t$  est élémentaire
  - Les plus courts chemins vérifient le principe de sous-optimalité
    - ✓  $D(y) = 0$  si  $y=s$
    - ✓  $D(y) = \min \{ D(x) + c(x,y) \mid x \text{ voisin de } y \}$  sinon

# Plus courts chemins (2/2)

- Algorithmes classiques :

- Dijkstra :

- ✓ Adaptation de l'algorithme de recherche !
    - ✓ Poids positifs
    - ✓ Plus court chemin entre un sommet et une destination

- Bellman :

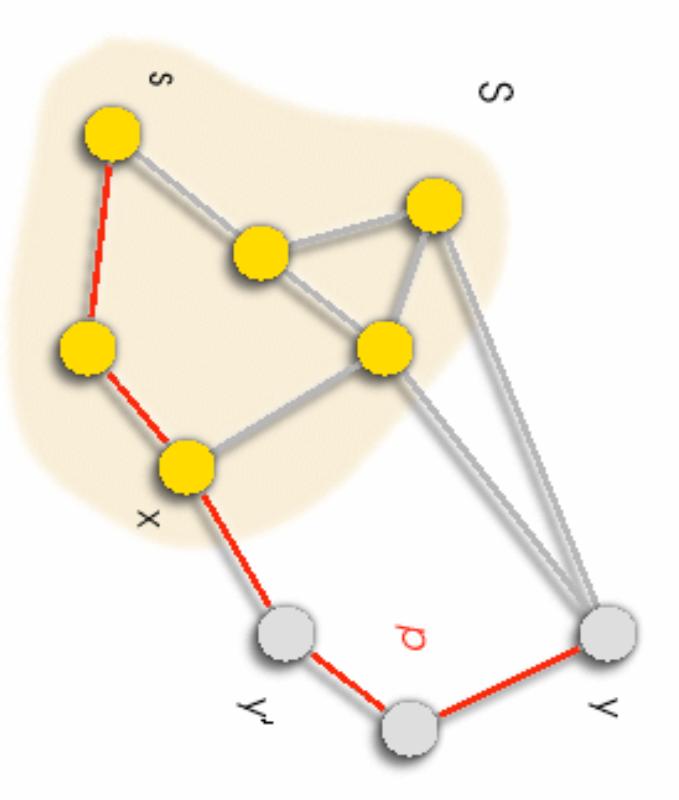
- ✓ Plus puissant
    - ✓ Poids positifs ou négatifs
    - ✓ Plus courts chemins entre toute paire de sommets
    - ✓ Programmation dynamique

# Algorithme de Dijkstra (1/6)

- Idée pour calculer la longueur des plus courts chemins :
  - ▶ Explorer les sommets des plus proches aux plus éloignés
  - ▶ Trouver le bon ordre d'exploration comme pour l'algo BFS !
- ✓ Ayant un ensemble  $S$  de sommets pour lesquels nous connaissons leurs plus courts chemins à  $s$
- ✓ Sélectionner le prochain sommet  $y$  pour lequel on peut calculer sa distance  $D(y)$  à  $s$  uniquement à partir des distances  $D(x)$  avec  $x$  dans  $S$
- **Distance partielle :**
  - ▶ Pour tout sommet  $z \notin S$  nous définissons sa distance partielle à  $S$  comme suit :
- ✓  $DP(S, z) = \min\{ D(x) + c(x, z) \mid x \in S \text{ et } x \text{ voisin de } z \}$

# Algorithme de Dijkstra (2/6)

- Propriété :
  - si  $y$  est un sommet avec la plus petite distance partielle, sa distance partielle est égale à sa distance à  $s$  :
    - ✓  $DP(S,y) = D(y)$
- Preuve par l'absurde :
  - Supposons que  $D(y) \neq DP(S,y)$
  - On a de fait  $D(y) < DP(S,y)$
  - Sous-optimalité  $\Rightarrow DP(S,y') \leq D(y)$
  - Par choix de  $y$ ,  $DP(S,y) \leq DP(S,y')$
  - Donc  $D(y') < DP(S,y')$
  - Sous-optimalité  $\Rightarrow DP(S,y') = D(x) + c(x,y')$
  - Or  $D(S,y') \leq D(x) + c(x,y')$
  - Donc  $D(x) + c(x,y') < D(x) + c(x,y')$  !!!!



# Algorithm de Dijkstra (3/6)

- Idée de l'algorithme :
  - Marquer les noeuds du graphe en sélectionnant à chaque étape le sommet non marqué de plus petite distance partielle
  - Un label  $L(x)$  est maintenu pour chaque sommet  $x$  : ✓ à la fin de l'algorithme le label  $L(x)$  est égal à sa distance à  $s$

**Input** :  $G=(V,E)$  valué positivement avec  $c$ ,  $s$  un sommet de  $V$   
Initialiser tous les sommets à non marqué ; Marquer  $s$  ;  
 $L(s) := 0$  ;

**Tant Que** il existe un sommet non marqué **Faire**

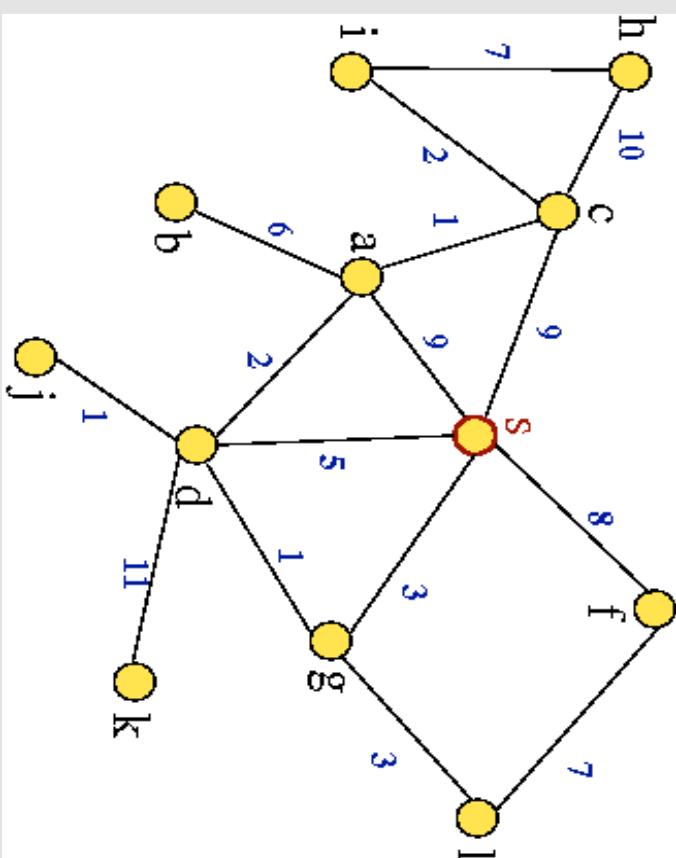
**Pour** chaque sommet  $y$  non marqué **Faire**

    Calculer  $L(y) := \min\{ L(x) + c(x,y) \mid x \text{ voisin marqué de } y \}$ ;

**Fin Pour**

    Choisir le sommet  $y$  non marqué de plus petit label  $L$  ;  
    Marquer  $y$  ;

**Fin TantQue**



# Algorithme de Dijkstra (4/6)

- Théorème :
  - ▶ L'algorithme de recherche calcule en temps  $O(|V| \cdot |E|)$  la longueur des plus courts chemins du sommet  $s$  à tous les autres sommets du graphe
  - ▶ Pour tout  $x$ ,  $L(x) = D(x)$
- Pourquoi ? Preuve :
  - ▶ La preuve de correction est triviale par la propriété des distances partielles (par induction sur les étapes de l'algo et les labels)
  - ▶ La complexité :
    - ✓ L'algorithme comporte  $|V|$  étapes correspondant au marquage de chacun des sommets
    - ✓ Chaque étape, on scanne les voisins de chaque sommet

# Algorithme de Dijkstra (5/6)

- Observation : seuls les labels (distances partielles) des sommets voisins à un sommet nouvellement marqué changent !

# Algorithme de Dijkstra (6/6)

- Théorème :
  - L'algorithme de recherche calcule en temps  $O(|V|^2)$  la longueur des plus courts chemins du sommet  $s$  à tous les autres sommets du graphe
- Preuve :
  - Mise à jour des labels  $O(d(y)) \Rightarrow$  Au total :  $O(|E|)$
  - Choix du sommet avec la distance partielle la plus petite peut se faire en  $O(|V|) \Rightarrow$  Au total :  $O(|V| \cdot |V|)$

# Cours 2 (Suite) :

## Arbres et structures couvrantes

- Définitions
- Minimum Spanning Tree
  - ✓ Algorithme de Prim
  - ✓ Algorithme de Kruskal

# Définitions

- Définitions :
  - ▶ Un arbre est un graphe connexe sans cycle
  - ▶ Une forêt est un graphe sans cycle (un ensemble d'arbres)
  - ▶ On parle de sommet racine (cela revient à orienter l'arbre)
  - ▶ Le père d'un sommet  $v$  est l'unique voisin de  $v$  sur le chemin vers la racine
  - ▶ Les fils de  $v$  sont les voisins de  $v$  autres que son père
  - ▶ Une feuille est un sommet sans père (de degré 1)
  - ▶ La hauteur de  $v$  est la longueur du plus court chemin de  $v$  vers la racine
  - ▶ Habituellement, on se représente un arbre par niveaux successifs

# Caractérisation

- Propriétés (caractérisation) : Il y a équivalence entre les propriétés suivantes pour tout graphe  $T$  (d'ordre  $n$ ) :

- $T$  est un arbre
  - $T$  est connexe avec  $n-1$  arêtes
  - $T$  est connexe et la suppression de toute arête le déconnecte
  - $T$  n'a pas de cycle et possède  $n-1$  arêtes
  - $T$  n'a pas de cycle et l'ajout de toute arête crée un cycle
- Arbre recouvrant
    - $T$  est un arbre recouvrant de  $G$  si
      - ✓  $T$  est un arbre
      - ✓  $T$  est un graphe partiel de  $G$

# Où/Pourquoi des arbres ?

- Systèmes distribués

- On veut acheminer de l'information d'un sommet vers tous les autres sommets du graphe

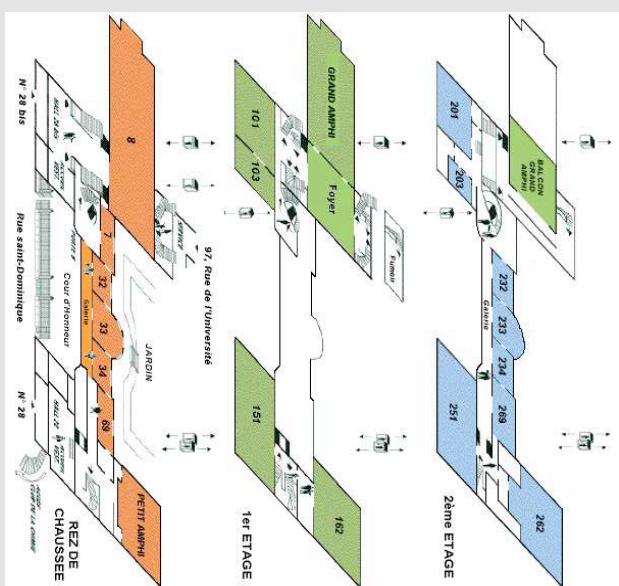
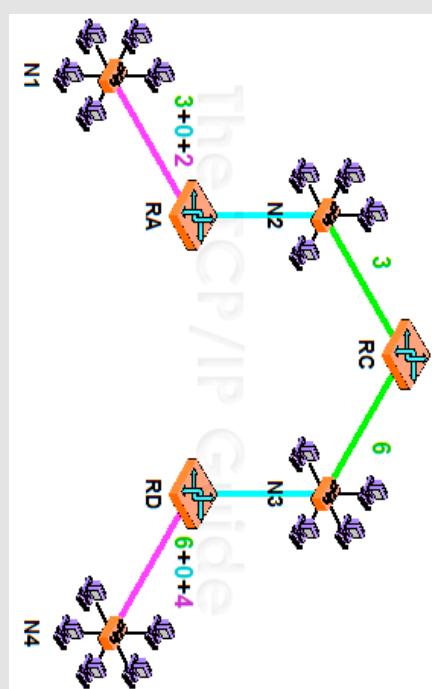
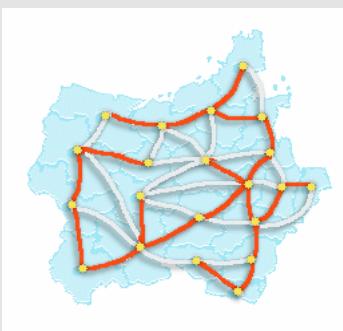
- On veut faire du routage

- Réseaux d'accès

- On veut connecter des points entre eux avec un coût minimum

- Plusieurs applications

- Plusieurs types d'arbres



# Arbre recouvrant de poids minimum (MST)

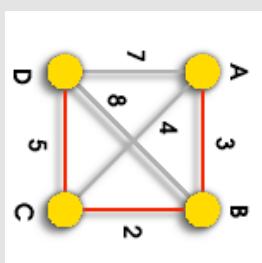
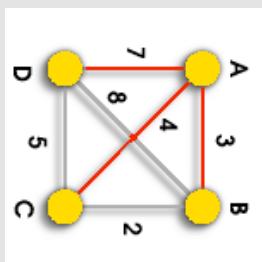
- Problème :

- Soit  $G=(V,E,c)$  un graphe valué positivement
- ✓  $c(e)$  le coût de l'arête  $e$  est unique (pour simplifier)
- On cherche à déterminer un graphe partiel connexe  $H=(V,E')$  de poids minimum

✓ Le poids de  $H$  est la somme des poids de ses arêtes

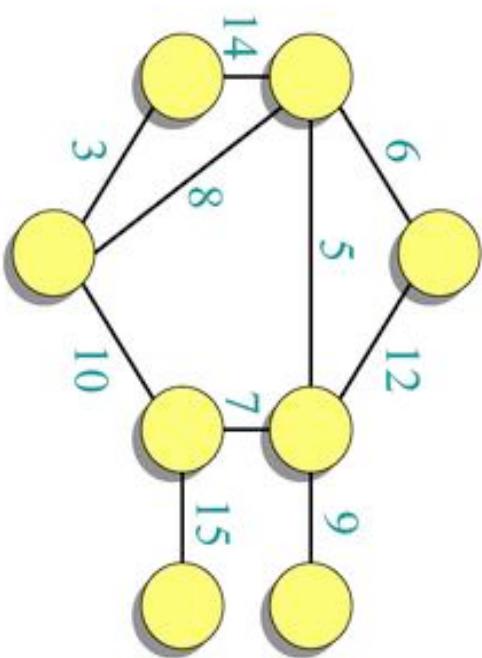
- Observations :

- $H$  est forcément un arbre
- $H$  est en général différent d'un arbre de plus courts chemins !?



# Essayons intuitivement sur un exemple !

- Comment trouver le MST du graphe suivant ?
  - ▶ Quelles arêtes allons nous choisir ?



# Go Greedy !?

- Propriété :
  - Un choix localement optimal est globalement optimal !
- Théorème :
  - Soit  $T$  un MST de  $G=(V,E)$
  - Soit  $A \subseteq V$
  - Supposons que l'arête  $e=(u,v)$  est l'arête de poids minimum connectant  $A$  à  $V \setminus A$
  - Alors,  $e=(u,v) \in T$

# Go Greedy !?

- Pourquoi / Preuve

▼ ???

# Algorithme de Prim 1/2

**Input** :  $G=(V,E)$  valué positivement

**Output** :  $T$  un arbre recouvrant de poids minimum

**F** : L'ensemble des arêtes de l'arbre

Initialiser  $F$  à vide ;

Marquer arbitrairement un sommet initial;

**Tant Que** il existe un sommet non marqué adjacent à un sommet marqué **Faire**

Sélectionner un sommet  $y$  non marqué adjacent à un sommet marqué  $x$   
tel que  $(x,y)$  est l'arête sortante de plus faible poids;

$F := F \cup \{(x,y)\}$ ;

Marquer  $y$ ;

**Fin TantQue**

**Retourner**  $T=(V,F)$

- Faire tourner l'algorithme sur l'exemple précédent
- L'algorithme de Prim est correct ! Pourquoi ?

# Algorithme de Prim 2/2

**Input** :  $G=(V,E)$  valué positivement  
**Output** :  $T$  un arbre recouvrant de poids minimum

```
Q := V;  
key[v] :=  $\infty$  pour tout  $v \in V$ ;  
key[s] := 0 pour un sommet arbitraire s;  
  
Tant Que  $Q \neq \emptyset$  Faire  
     $u := \text{Extract-MIN}(Q)$ ;  
    Pour chaque  $v \in \text{Adj}[u]$  Faire  
        Si  $v \in Q$  et  $c(u,v) < \text{key}[v]$  Alors  
             $\text{key}[v] := c(u,v)$   
            pere(v) := u;  
        Fin Si  
    Fin Pour  
Fin TantQue  
Retourner  $T = (V, \{(v, \text{p}(v))\})$ 
```

- Une implémentation un peu plus détaillée

- Le meilleur algo connu pour le MST s'execute en moyenne en temps  $O(n + m)$ 
  - ▶ Algo. Probabiliste
  - ▶ Karger, Klein, Tarjan (1993)

# Exercice 1

• Faire tourner l'algo sur l'exemple précédent

• Cet algo calcule :

- ???

• Sa complexité est :

- ???

e : Tableau des arêtes du graphe G  
F : Ensemble d'arêtes

Initialiser F à vide;  
Trier les arêtes de e par poids  $c(e[i])$  croissant;

**Pour**  $i = 1$  à  $|E|$  **Faire**  
**Si**  $F \cup \{e[i]\}$  est acyclique **Alors**  
     $F := F \cup \{e[i]\}$   
**Fin Si**  
**Fin Pour**  
**Retourner**  $(V, F)$