# Algorithmes gloutons, programmation dynamique

- Méthodes: algorithmes gloutons et matroïdes, programmation dynamique
- Problèmes: Arbre couvrant maximum, Plus courts chemins à partir d'un sommet, Plus courts chemins entre tous les sommets, Nombres de Fibonacci
- Algorithmes: Kruskal, Dijkstra, Roy-Warshall





## Optimisation Combinatoire : problème abstrait

**Entrée :** $E$ ensemble fini, valuation $v : E \to \mathcal{R}^+$, famille $\mathcal{F} \subseteq 2^E$

**Sortie :** $F \in \mathcal{F}$ maximisant

$$\sum_{f \in F} v(f).$$

**Remarque :** Le plus souvent, $\mathcal{F}$ est close par passage aux sous-ensembles.





## L'algorithme glouton général

1. Trier les éléments de $E$ par valuations décroissantes :

$$v(e_1) \geq v(e_2) \ldots \geq v(e_n)$$

2. Initialisation : $F \leftarrow \emptyset$
3. Pour $i$ allant de 1 à $n$ faire Si $F \cup \{e_i\} \in \mathcal{F}$ Alors $F \leftarrow F \cup \{e_i\}$

**Remarque :** L'algorithme glouton ne marche pas toujours !





## Graphes : définitions

Graphe $G = (V, E)$ : $V$ sommets, $E$ arêtes = paires de sommets.

chemin = suite d'arêtes

$$e_1 = \{x_1, x_2\}, e_2 = \{x_2, x_3\}, \ldots e_p = \{x_p, x_{p+1}\}$$

cycle : chemin tel que $x_1 = x_{p+1}$

chemin élémentaire : ne passe pas deux fois par le même sommet

Graphe connexe : toute paire de sommets est reliée par un chemin.

Arbre : graphe connexe sans cycle.

## Arbres : lemme fondamental

Pour un graphe $G$ ayant $n$ sommets, les propriétés suivantes sont équivalentes :

1. $G$ est un arbre
2. $G$ est connexe minimal
3. $G$ est sans cycle maximal
4. $G$ est connexe et a $n-1$ arêtes
5. $G$ est sans cycle et a $n-1$ arêtes
6. Toute paire de sommets de $G$ est reliée par un unique chemin





## Arbres couvrants

**Notations**

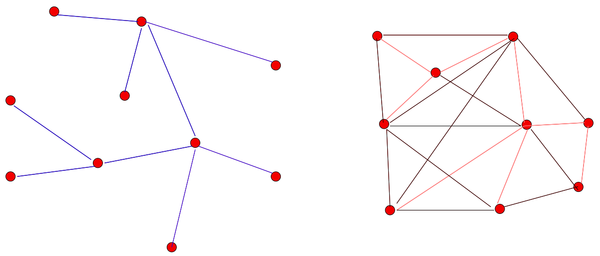
- $G = (V, E)$, $|V| = n$, $|E| = m$
- $(V, T)$ arbre couvrant de $G$

**Propriétés**

1. $|T| = n - 1$
2. Chaque arête $e$ qui n'est pas dans $T$ crée un cycle $C_e$ unique avec $T$.











## Arbre couvrant de poids maximum

$G$ graphe connexe où chaque arête $e$ a un poids $p(e)$.
But: arbre couvrant de poids total maximum

Problème d'optimisation combinatoire avec :

$\mathcal{F}$ = parties de $E$ sans cycle

### Algorithme de Kruskal

**1.** Classer les arêtes par poids décroissants : $p(e_1) \geq p(e_2) \ldots \geq p(e_n)$

**2.** Initialisation: $T \leftarrow \emptyset$

**3.** Pour $i$ allant de 1 à $n$ faire Si $T \cup \{e_i\}$ n'a pas de cycle Alors $T \leftarrow T \cup \{e_i\}$

## Preuve de validité

**Lemme**

Soient $T$ et $U$ deux arbres couvrants de $G$, soit $a \in U,\ a \notin T$, alors il existe $b \in T$ tel que $U - a + b$ soit un arbre couvrant.

De plus $b$ peut être choisi dans le cycle formé par des arêtes de $T$ et $a$.

**Théorème**

L'algorithme de Kruskal construit l'arbre couvrant de poids maximum





## Preuve du théorème

- Soient $T$ l'arbre obtenu par l'algo glouton $U$ un arbre de poids maximal tel que $|T \cap U|$ soit maximum
- Par l'absurde. Si $T \neq U$: soit $e \in T \setminus U$...





## Théorème général

L'algorithme glouton général donne une solution optimale quelle que soit la fonction $v$ si et seulement si :

Si $F$ et $G$ sont deux ensembles de $\mathcal{F}$ tels que $G$ a plus d'éléments que $F$, alors il existe un élément $x$ de $G \setminus F$ tel que $F \cup \{x\} \in \mathcal{F}$.





$\Longrightarrow$

- Soient $F$ et $G$ donnés, $|G| > |F|$. On définit $v$ par:
- On vérifie: $v(G) \geq v(F)$. Glouton $\Longrightarrow F' \supseteq F,\ F' \neq F$.

$\Longleftarrow$

- Soit $F$ obtenu par l'algo glouton, $G$ appartenant à $\mathcal{F}$, optimal et tel que $|F \cap G|$ maximum ; si $F \neq G$, soit $e \in G \setminus F$, $e$ premier élément (par ordre de $v$ décroissant)...





## Matroïdes

$\mathcal{F} \subseteq 2^E$ forme un matroïde si les trois conditions suivantes sont satisfaites :

1. $\mathcal{F} \neq \emptyset$.
2. Si $F \in \mathcal{F}$ et $G \subset F$ alors $G \in \mathcal{F}$
3. Si $F, G \in \mathcal{F}$ et $|G| > |F|$ alors $\exists x \in G \setminus F$ tel que $F \cup \{x\} \in \mathcal{F}$

Les éléments de $\mathcal{F}$ sont appelés les *ensembles indépendants* du matroïde.

**Exemples :** Arêtes sans cycle d'un graphe, vecteurs libres d'un espace vectoriel





## Plus court chemin à partir d'un sommet

**Entrée :** graphe orienté aux arcs valués et un sommet $x_0$
**Sortie :** pour chaque sommet $x$ : la longueur du plus court chemin de $x_0$ à $x$, et la suite des sommets constituant ce plus court chemin

**Structure de données :**

- `d[x]` longueur du chemin de $x_0$ à $x$
- `pere[x]` sommet qui précède $x$ sur le chemin de $x_0$ à $x$

**Calcul du chemin à partir de la structure de données :**

chemin($x_0, x$)=chemin($x_0$,père($x$))@ (père($x$),x)





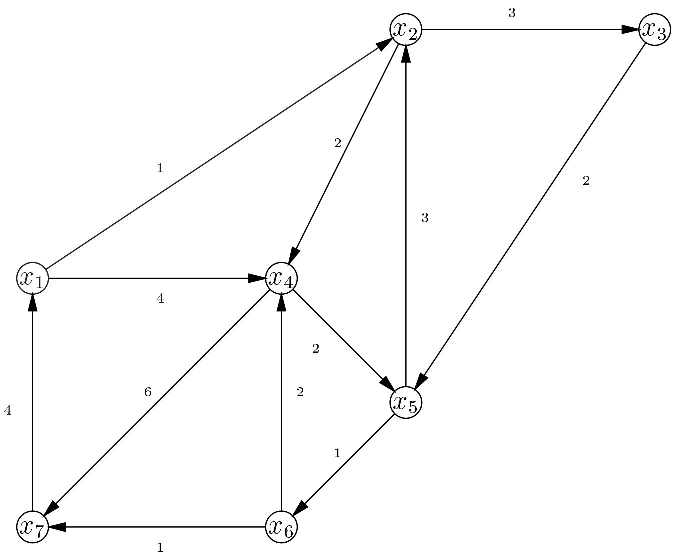



Figure 1: Calcul de plus courts chemins

## Algorithme de Dijkstra

**Initialisation :**

$A \leftarrow \emptyset$

$d[x_0] \leftarrow 0$

$d[x] \leftarrow \infty$ pour $x \neq x_0$

**Tant que** $A \neq V$ **:**

  Soit $y$ l'élément de $V \setminus A$ avec $d[y]$ minimum

  $A \leftarrow A \cup \{y\}$

  Pour tout $z$ voisin de $y$, relaxer l'arc $(y, z)$.





## Relaxer l'arc $(y, z)$

Si $d[y] > d[x] + \ell(x, y)$

Alors

  $d[y] \leftarrow d[x] + \ell(x, y)$

  père$[y] \leftarrow x$





## Algorithme de Dijkstra : analyse

**Invariant :** $d[y]$ représente la longueur d'un chemin de $x_0$ à $x$.

**Invariant :** Si $y \in A$, alors $d[y]$ est la longueur du plus court chemin de $x_0$ à $y$, et pour tout voisin $z$ de $y$, on a considéré le plus court chemin de $x_0$ à $z$ qui se termine par l'arc $(y, z)$.





## Algorithme de Dijkstra : complexité

Structure de données pour $V \setminus A$: Ensemble avec valeurs $d[x]$. Opérations:

1. Trouver et supprimer l'élément de plus petite valeur
2. diminuer la valeur d'un élément

**Tas :** ces opérations on coût $O(\log n)$.

**Complexité :** $O(n + m \log n)$.

## Plus courts chemins entre tous les sommets

Soit $\delta_k(x, y)$ la longueur du plus court chemin entre $x$ et $y$ qui soit composé d'au plus $k$ arcs.

**But :** calculer $\delta_n(x, y)$ pour tout $x$, $y$.

**Définition récursive :**

$$\begin{aligned}\delta_0(x, y) &= \begin{cases} 0 & \text{si } x = y \\ \infty & \text{sinon} \end{cases} \\ \delta_k(x, y) &= \min_{z:(x,z)\in E}(\ell(x, z) + \delta_{k-1}(z, y)).\end{aligned}$$

**Implantation :** Tableau $\delta[k, x, y]$.





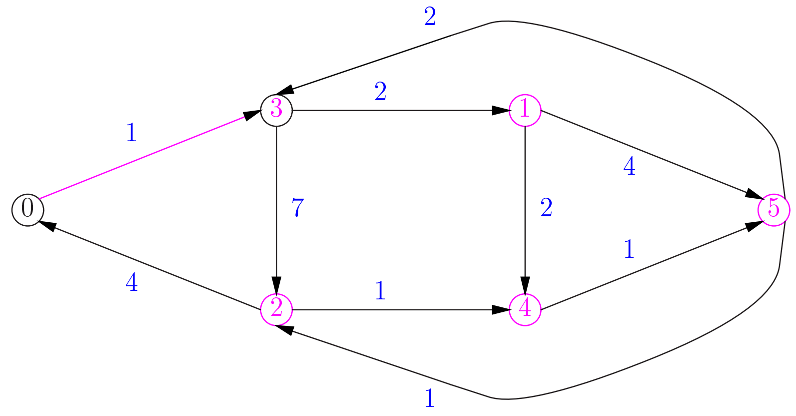



Figure 2: Tous les chemins





## Algorithme de Roy-Warshall

Pour $k$ allant de 1 à $n - 1$<br>
  Pour $i$ allant de 1 à $n$<br>
    Pour $j$ allant de 1 à $n$<br>
      Si $\delta[i, j] > \delta[i, k] + \delta[k, j]$<br>
        Alors $\delta[i, j] \leftarrow \delta[i, k] + \delta[k, i]$





## Suite de Fibonacci

Définition récursive : $f_0 = f_1 = 1, f_n = f_{n-1} + f_{n-2}$.

Implantation en récursif : complexité exponentielle.

Implantation en itératif : tableau $f[0 \ldots n]$, complexité polynomiale.

## Quand peut-on utiliser la programmation dynamique ?

1. $\exists$ un ordre partiel sur les problèmes tel que :
   - on sait résoudre le problèmes minimaux
   - tout problème $P$ non minimal se résoud au moyen de solutions de certains problèmes $P'$, avec $P' < P$
2. Le calcul pour $P$ à partir des solutions pour les $P'$ se fait en temps polynomial en $|P|$.
3. Le nombre total de sous-problèmes, $|\{P'|P' < P\}|$, est **polynomial** en $|P|$.