Généralités sur les graphes

Premières définitions

[Définition] Graphe simple non orienté : On appelle graphe simple non orienté la donnée d'un ensemble fini S, appelé ensemble des sommets, et d'un ensemble A contenant des paires d'éléments de S, appelé ensembledes arêtes. On appelle ordre du graphe le nombre de sommets du graphe.

[Définition] Graphe orienté : On appelle graphe orienté la donnée d' un ensemble fini S, appelé ensemble des sommets, et d' un ensemble A contenant des couples ordonnés d' éléments de S, appelé ensemble des arêtes.

[Définition] voisins de s dans G : Soit $G = (S, A)$ un graphe simple non orienté et $s' \in S$ un sommet de G. On dit que $s' \in S$ est un voisin des dans G si $s - s'$ est une arête de G, c' est-à-dire si $s, s' \in A$. On dit aussi que s et s' sont adjacents et sont les extrémités de l' arête $s - s'$.On appelle degré de s dans G, noté $deg(s)$, le nombre de voisins de s dans G, c' est-à-dire

$$deg(s) = card\{\ s' \in S, s - s'\ arête\ de\ G\ \}$$

[Proposition] : Soit $G = (S, A)$ un graphe, alors $\sum_{s \in S} deg(s) = 2\ card(A)$.

[Définition] sous-graphe de G : Soit $G = (S, A)$ un graphe. On dit que le graphe $G' = (S', A')$ est un sous-graphe de G si $S' \subset S$ et $A' \subset A$. On écrit alors $G' \subset G$. Si A' contient TOUTES les arêtes de G qui relient deux éléments de S', on dit que G' est le sous-graphe de G induit par S'.

[Définition] marche dans G : On appelle marche dans G une liste de la forme $x_0 - x_1 - \cdots - x_l$ telle que

1. x_i est un sommet de G pour tout $i \in [0, l],\ i \in \mathbb{Z}$.
2. $x_i - x_{i+1}$ est une arête de G pour tout $i \in [0, l+1],\ i \in \mathbb{Z}$.

l est appelée la longueur de la marche. Les sommets x_0 et x_l sont les extrémités de la marche : on parle alors de marche de x_0 à x_l.On dit que la marche est:

- un **parcours** si les arêtes $x_i - x_{i+1}$ sont distinctes deux à deux ;
- un **chemin** si les sommets x_i sont tous distincts deux à deux ;
- un **circuit** si les arêtes $x_i - x_{i+1}$ sont distinctes deux à deux, et $x_0 = x_l$ (parcours fermé) ;
- un **cycle** si $x_0 = x_l$, et les sommets $x_0, \dots, x_{l-1}$ sont distincts deux à deux (chemin fermé).

[Proposition] : Soit $G = (S, A)$ un graphe et $(x, y) \in S^2$.

1. S' il existe une marche de x à y, alors il existe un chemin de x à y.
2. S' il existe une marche fermée de x à x, alors il existe un cycle contenant x .

[Définition] distance de x à y : notée $d(x, y)$, la longueur minimale d' un chemin de x à y dans G. S' il n' y a pas de chemin de x à y dans G, on note $d(x, y) = +\infty$. On dit qu' un graphe est connexe s' il existe un chemin de x à y dans G pour tout $(x, y) \in S^2$. Pour $x \in S$, on appelle composante connexe de x dans G le plus grand sous-graphe connexe de G qui contient x.On dit qu' un graphe est acyclique s' il ne contient pas de cycle.

Graphes particuliers

[Définition] les graphes G et G' sont isomorphe : Soit $G = (S, A)$ et $G' = (S', A')$ deux graphes. On dit que les graphes G et G' sont isomorphes si il existe une application $\varphi : S \to S'$ bijective telle que

$$\forall (s_1, s_2) \in S,\ s_1 - s_2 \in A \iff \varphi(s_1) - \varphi(s_2) \in A'$$

L' application φ est alors appelé isomorphisme entre les graphes G et G'.

Quand on dit que deux graphes sont **égaux (ou identiques)**, cela veut dire qu' ils sont en fait **isomorphes**.

[Définition] graphe complet : On dit que $G = (S, A)$ est un graphe complet si : pour tout $(x, y) \in S^2$ avec $x \neq y$, on a $x - y \in A$. Cela veut dire : il y a une arête entre chaque paire de sommets. Pour $n \in \mathbb{N}^*$, il existe un unique graphe complet d' ordre n: on le note K_n.

[Définition] graphe biparti : On dit qu' un graphe $G = (S, A)$ est biparti si on peut trouver une partition $S = S_1 \cup S_2$ avec $S_1 \cap S_2 = \emptyset$, tel que toutes les arêtes de G relient un point de S_1 à un point de S_2. On dit qu' il est biparti complet si, en plus, chaque point de S_1 est relié à tous les points de S_2. Pour $(m, n) \in \mathbb{N}^2$, on note $K_{m,n}$ le graphe biparti complet tel que $card(S_1) = m$ et $card(S_2) = n$.

Recherche de chemins et graphes pondérés

Recherche de chemins

[Définition] fonction prédécesseur : Soit $G = (S, A)$ un graphe et $x \in S$. On appelle fonction prédécesseur de racine x sur l' ensemble S' la donnée d' une fonction $P : S' \to S' \cup \{racine\}$ tel que :

- $P(x) = racine$ (x est appelé la racine) ;
- Pour tout $s \in S' \setminus \{x\}$, il existe $n \in N$ tel que $s - P(s) - P(P(s)) - P(P(P(s)) - \dots - P^n(s)$ est un chemin de s à x.

TP2 : pred_donne_chemin

Graphes pondérés

[Définition] fonction de poids : Soit $G = (S, A)$ un graphe. On appelle fonction de poids pour G toute fonction $w : A \to R$. Pour $x{-}y \in A$, on dit que $w(x{-}y)$ est le poids de l'arête $x{-}y$. On dit alors que G est un graphe pondéré. On appelle poids total de G, noté $w(G)$, la somme des poids de toutes les arêtes de G:

$$w(G) = \sum_{x-y \in A} w(x{-}y).$$

TP3 : flot_nul

Algorithme de Dijkstra

- **Données :** un graphe pondéré G avec poids positifs, un sommet x de G.
- **Initialisation :**
  - Ensemble des sommets visités : $L_{visites} = \emptyset$;
  - Ensemble des sommets non visités : $L_{non-v} = S$;
  - Fonction "prédécesseur" $P : S \to S \cup \{racine\}$, initialisée par
$$\forall s \in S, P(s) = \begin{cases} racine & si\ s = x \\ non\ défini & sinon \end{cases}$$
  - Fonction "poids du chemin le plus court partant de x" $W : S \to \mathbb{R}^+$, initialisée par :
$$\forall s \in S, W(s) = \begin{cases} 0 & si\ s = x \\ +\infty & sinon \end{cases}$$
- **Boucle :** Tant que L_{non-v} est non-vide, faire :
  - Choisir $s \in L_{non-v}$ avec $W(s)$ est minimal ;
  - Changer $L_{visites}$ en $L_{visites} \cup s$ et L_{non-v} en $L_{non-v} \setminus \{s\}$;
  - Pour tous les voisins $z \in L_{non-v}$ de s, faire :
$$si\ W(z) > W(s) + w(s-z), alors \begin{cases} changer\ W(z)\ en\ W(s) + w(s-z) \\ changer\ P(z)\ en\ s \end{cases}$$
- **Résultat :** Deux fonctions $W : S \to \mathbb{R}^+$ et $P : S \to S \cup \{racine\}$ telle que, pour tout $s \in S$,
  - $W(s)$ est le poids du plus court chemin entre x et s dans G;
  - $P(s)$ est le prédécesseur de s dans un chemin de x à s de poids minimal.

L'algorithme de Dijkstra termine toujours et produit le résultat annoncé. Sa complexité est $O(n^2)$, où n est le nombre de sommets du graphe G.

TP2 : dijkstra

Arbres

[Proposition] : Soit $G = (S, A)$ un graphe d'ordre n.

1. Si G est connexe, alors G a au moins $n - 1$ arêtes.
2. Si G est acyclique, alors G a au plus $n - 1$ arêtes.

Démonstration : par hérédité.

[Définition] Arbre : On dit qu'un graphe G est un arbre si G est connexe et acyclique.

[Proposition] : Soit G un graphe. Il y a équivalence entre :

1. G est un arbre ;
2. G est un graphe connexe minimal *ou* G est connexe et a $n-1$ arêtes ;
3. G est un graphe acyclique maximal *ou* G est acyclique et a $n-1$ arêtes ;

Arbre couvrant minimal

[Proposition] : Soit $G=(S,A)$ un graphe connexe. Alors il existe un sous-graphe T de G tel que :

- T contient tous les sommets de G;
- T est un arbre.

On dit alors que T est un arbre couvrant de G .

Démonstration : $C=\{G' \subset G | G' \ contient\ tous\ les\ sommets\ de\ G\ et\ est\ connexe\}$

Recherche d' arbre couvrant minimal

[Définition] arbre couvrant minimal : un arbre couvrant T de G tel que $w(T)=min\{w(T')|T'\ arbre\ couvrant\ de\ G\}$. Soit G un graphe pondéré connexe. Alors il existe un arbre couvrant minimal de G.

Algorithme 3 :

- **Données :** un graphe pondéré connexe $G=(S,A)$, de fonction de poids w.
- **Initialisation :** On choisit un sommet quelconque x de G
  - Ensemble des sommets visités : $L_{visites}=\{x\}$;
  - Ensemble des sommets non visités : $L_{non-v}=S\setminus\{x\}$;
  - Fonction "prédécesseur" $P : S \to S\cup\{racine\}$, initialisée par
$$\forall s\in S, P(s)=\begin{cases} racine & si\ s=x \\ non\ défini & sinon \end{cases}$$
- **Boucle :** Tant que L_{non-v} est non-vide, faire :
  - Trouver l' arête $s-s'$ avec $s\in L_{visites}$ et $s'\in L_{non-v}$ de poids minimal ;
  - Changer $L_{visites}$ en $L_{visites}\cup s'$ et L_{non-v} en $'L_{non-v}\setminus\{s\}'$, et définir $P(s')=s$.
- **Résultat :** une fonction P telle que P est une fonction précédesseur qui décrit un arbre couvrant minimal de G.

Algorithme de Prim-Dijkstra

- **Données :** un graphe pondéré $G=(S,A)$ avec poids positifs, un sommet x de G.
- **Initialisation :**
  - Ensemble des sommets visités : $L_{visites}=\emptyset$;
  - Ensemble des sommets non visités : $L_{non-v}=S$;
  - Fonction "prédécesseur" $P : S \to S\cup\{racine\}$, initialisée par
$$\forall s\in S, P(s)=\begin{cases} racine & si\ s=x \\ non\ défini & sinon \end{cases}$$
  - Fonction "poids minimal des arêtes déjà regardées" $W : S \to \mathbb{R}^+$, initialisée par :

$$\forall s \in S, W(s) = \begin{cases} 0 & si\ s = x \\ +\infty & sinon \end{cases}$$

- **Boucle :** Tant que L_{non-v} est non-vide, faire :
  - Choisir $s \in L_{non-v}$ avec $W(s)$ est minimal ;
  - Changer $L_{visites}$ en $L_{visites} \cup s$ et L_{non-v} en $L_{non-v} \setminus \{s\}$;
  - Pour tous les voisins $z \in L_{non-v}$ de s, faire :

$$si\ W(z) > w(s-z), alors \begin{cases} changer\ W(z)\ en\ w(s-z) \\ changer\ P(z)\ en\ s \end{cases}$$

- **Résultat :** Deux fonctions $W : S \to \mathbb{R}^+$ et $P : S \to S \cup \{racine\}$ telle que
  - P est une fonction prédécesseur qui décrit un arbre couvrant minimal de G ;
  - Pour tout $s \in S$, $W(s)$ est le poids de l'arête précédant s dans cet arbre couvrant.

Graphes orientés et flots

[Définition] : Soit $G = (S, A)$ un graphe **orienté**. Pour $x \in S$, on note

$$\Gamma(x) = \{y \in S \mid (x, y) \in A\ ou\ (y, x) \in A\ \}$$

$$\Gamma^+(x) = \{y \in S \mid (x, y) \in A\ \}$$

$$\Gamma^-(x) = \{y \in S \mid (y, x) \in A\ \}$$

On va définir, pour un graphe orienté, les « flots ».

[Définition] source s et puits p : Soit $G = (S, A)$ un graphe orienté. On fixe deux sommets distincts s et p de G. On appelle flot dans G de source s et de puits p toute fonction $f : A \to \mathbb{R}^+$ qui vérifie la loides noeuds de Kirchhoff pour les sommets de $S \setminus \{s, p\}$:

$$\forall x \in S \setminus \{s, p\}, \sum_{y \in \Gamma^+(x)} f(x, y) = \sum_{z \in \Gamma^-(x)} f(z, x)$$

[Définition] Le flot total sortant de s : Soit f un flot de source s et de puits p. Alors Le flot total sortant de s est égalau flot total entrant de p. Cette valeur est appelée la valeur du flot f, ou la quantité de flot de s à p. On la note $v(f)$.

[Définition] la capacité : Soit $c : A \to \mathbb{R}^+$ une fonction. On dit que le flot f est adapté à la fonction de capacité c si : $\forall (x, y) \in A,\ f(x, y) \leq c(x, y)$. Pour $(x, y) \in A$, le réel $c(x, y)$ est appelé la capacité de l'arête (x, y).

Le théorème flot-max/coupe-min

[Proposition] : Il existe un flot f dont la valeur du flot est maximale. On dit alors que f est un flot maximal.

[Définition] coupe séparant s de p , capacité de la coupe $A(S_1, \bar{S_1})$: Soit S_1 et S_2 deux sous-ensembles de S. On note

$$A(S_1, S_2) = \{(x, y) \in A \mid x \in S_1\ et\ y \in S_2\ \}$$

Si S_1 est un ensemble de sommets qui contient s et pas p, on dit que $A(S_1, \bar{S_1}) = A(S_1, S \setminus S_1)$ est une coupe séparant s de p, ou plus simplement une coupe. On appelle capacité de la coupe $A(S_1, \bar{S_1})$ le réel

$$c(S_1, \bar{S_1}) = \sum_{(x,y)\in A(S_1,\bar{S_1})} c(x,y) = \sum_{\substack{(x,y)\in A \\ x\in S_1, y\in \bar{S_1}}} c(x,y)$$

[Proposition] : soit f un flot et $A(S_1, \bar{S_1})$ une coupe. Alors on a $v(f) \leq c(S_1, \bar{S_1})$ (la valeurdu flot est plus petite que la capacité de la coupe). En particulier,

$$\max_{f\ flot} v(f) \leq \min_{A(S_1,\bar{S_1})\ coupe} c(S_1, \bar{S_1})$$

[Théorème] Théorème flot-max coupe-min // max-flow min-cut Theorem : La valeur maximale de flot entre s et p est égale au minimum des capacités des coupes séparant s de p.

$$\max_{f\ flot} v(f) = \min_{A(S_1,\bar{S_1})\ coupe} c(S_1, \bar{S_1})$$

On construit l'ensemble S_1 étape par étape de la manière suivante :

- Initialisation : On met s dans S_1.
- Boucle : S'il existe $x \in S_1$ et $y \notin S_1$ tel que

  $$c(x,y) > f_{max}(x,y)\ ou\ f_{max}(y,x) > 0$$

  on ajoute y à S_1. On continue cette boucle tant que cela est possible.

Un algorithme pour trouver un flot maximal quand la fonction de capacité est à valeurs dans $\mathbb{N}$:

- Initialisation : on initialise le flot f par $f(x,y) = 0$ pour tout $(x,y) \in A$.
- Boucle : On construit l'ensemble S_1 correspondant à f comme pécédent.
  - Si $p \notin S_1$, alors la démonstration précédente montre que $v(f) \geq c(S_1, \bar{S_1})$. Cela n'est possible que si f est un flot maximal (et $A(S_1, \bar{S_1})$ une coupe minimale). On renvoie alors f.
  - Si $p \in S_1$, alors comme dans la démonstration on pêut changer f en un flot f^* avec $v(f^*) = v(f) + \varepsilon$. Comme les capacités sont entières, on a $\varepsilon \geq 1$. On reprend alors la boucle avec f^*.

[Théorème] Integrality theorem : Si la fonction de capacité est à valeurs entières, alors il existe un flot maximal à valeurs entières.

la présence de p dans S_1 est équivalente à l'existence d'un « chemin » $s = x_0, x_1, \cdots, x_l = p$ tel que

$$\forall i \in [0, l-1],\ i \in \mathbb{Z},\ \ c(x_i, x_{i+1}) > f(x_i, x_{i+1})\ ou\ f(x_{i+1,}, x_i) > 0.$$

Un tel chemin est appelé chemin augmentant, et sa valeur est

$$\varepsilon = \min_{1\leq i\leq l-1} \varepsilon_i = \min_{1\leq i\leq l-1} (\max(c(x_i, x_{i+1}) - f(x_i, x_{i+1}), f(x_{i+1}, x_i)))$$

Algorithme de Ford-Fulkerson

- **Données :** un graphe orienté $G = (S, A)$, une fonction de capacité $c : A \to \mathbb{N}$.
- **Initialisation :** on initialise le flot f par $f(x,y) = 0$ pour tout $(x,y) \in A$.
- **Boucle :** Tant qu'il existe un chemin augmentant $x0 = s – x1 – \cdots – xl = p$ (de valeur ε), faire pour tout $i \in [0, l-1],\ i \in \mathbb{Z}$:
  - Si $c(x_i, x_{i+1}) - f(x_i, x_{i+1}) > f(x_{i+1}, x_i)$, changer $f(x_i, x_{i+1})$ en $f(x_i, x_{i+1}) + \varepsilon$;
  - Sinon, changer $f(x_{i+1}, x_i)$ en $f(x_{i+1}, x_i) - \varepsilon$.
- **Résultat :** Une fonction $f : A \to \mathbb{N}$ qui est un flot maximal.

La complexité de l'algorithme de Ford-Fulkerson est $O(m \times v_{max})$, où m est le nombre d'arêtes et v_{max} est la valeur maximale du flot.

TP4 : trouve_chemin_augmentant3

TP4 : flot_maximal

TP4 : flot_maximal2

[Théorème] Théorème flot-max coupe-min pour sources et puits multiples : La valeur maximale des flots des sources vers les puits est égale à la capacité minimale descoupes séparant les sources des puits.

Démonstration : ajouter nouvlle source et puits

[Définition] coupe de sommets séparant : On va donner des capacités aux sommets. On appelle coupe de sommets séparant s de p tout ensemble de sommets $S_1 \subset S \setminus \{s, p\}$ tel que : si on enlève les sommets de S_1, il n'y a pas de chemin (orienté) de s à p. Cela est équivalent à dire : si on enlève les sommets de S_1, il n'y a de flot de s à p avec une valeur $v(f) > 0$. On appelle capacité de la coupe S_1 le réel $c(S_1) = \sum_{x \in S_1} c(x)$.

[Théorème] Théorème flot-max coupe-min pour capacités aux sommets : Pour un graphe avec capacités sur les sommets, la valeur maximale du flot de s à p est égale au minimum des capacités des coupes de sommets séparant s de p.

Démonstration : séparer un sommet s avec s^+ et s^- .

Couplages

[Définition] couplage : Soit $G = (S, A)$ un graphe. On dit qu'un ensemble $K \subset A$ est un couplage si tout sommet $s \in S$ est dans au plus une arête de K.

[Définition] : Soit $G = (S, A)$ un graphe et K un couplage. On dit que K est :

- un couplage **maximal** si on ne peut pas ajouter d'arête dans l'ensemble K et toujours avoir un couplage ;
- un couplage **maximum** si c'est un couplage contenant le plus grand nombre possible d'arêtes ;
- un couplage **parfait** si chaque sommet de S est dans une (unique) arête de K .

$$parfait \subset maximun \subset maximal$$

[Définition] sommet est exposé : Soit $G = (S, A)$ un graphe et K un couplage.

- On dit qu'un sommet est exposé s'il n'appartient pas à une arête de l'ensemble K.
- On appelle chemin K-augmentant un chemin $x_0 - x_1 - \cdots - x_{2l} - x_{2l+1}$ tel que :
  - x_0 et x_{2l+1} sont des sommets exposés ;
  - Pour tout $i \in [1, l], i \in \mathbb{Z}$, l'arête $x_{2i-1} - x_{2i}$ est dans K.
- Si $x_0 - x_1 - \cdots - x_{2l} - x_{2l+1}$ est un chemin K-augmentant, on peut construire un couplage K' avec plus d'arêtes que K, en posant

$$K' = (K \setminus \{x_{2i-1} - x_{2i} \mid i \in [1, l], i \in \mathbb{Z}\}) \cup \{x_{2i} - x_{2i+1} \mid i \in [0, l], i \in \mathbb{Z}\}$$

[Théorème] Lemme de Berge : Soit $G = (S, A)$ un graphe et K un couplage. Le couplage K est maximum si et seulement si il n'existe pas de chemin K-augmentant.

Couplage maximum dans un graphe biparti : problème des mariages

Soit G un graphe biparti de partition de sommet s associée (S_1, S_2). On construit un graphe orienté G' à partir de G ainsi : on oriente toutes les arêtes de S_1 vers S_2, on ajoute un sommet s avec des arêtes qui vont vers les sommets de S_1, et un sommet p avec des arêtes qui viennent des sommets de S_2. On met une capacité 1 sur toutes les arêtes.

[Proposition] : Un chemin K-augmentant dans G correspondant à un chemin augmentant pour f dans G', et réciproquement. Un couplage maximum de G correspond à un flot maximal (à valeurs entières) de G', et réciproquement.

[Définition] l'ensemble des voisins des sommets de V_1 dans G : Soit G un graphe biparti de partition de sommets associée $(S1, S2)$, et $V_1 \subset S_1$. On note $\Gamma(V_1)$ l'ensemble des voisins des sommets de V_1 dans G, c'est-à-dire

$$\Gamma(V_1) = \{y \in S \mid \exists x \in V_1 : x{-}y \in A\}.$$

[Théorème] Théorème de Hall : Soit $G = (S, A)$ un graphe biparti,de partition de sommets associée (S_1, S_2). Le graphe G admet un couplage qui contient tous les sommets de S_1 si et seulement si

$$\forall V_1 \subset S_1, card(\Gamma(V_1)) \geq card(V_1)$$

[Proposition] : Lorsqu'on trouve un flot maximal pour G' avec l'algorithme de Ford-Fulkerson :

- On obtient un couplage maximum pour G ;
- La dernière recherche de chemin augmentant (qui n'arrive donc pas jusqu'à p) construit l'ensemble B tel que $A(B, \bar{B})$ est de capacité minimale. Ainsi, quand le couplage ne contient pas tous les sommets de S_1, cela permet de trouver un ensemble $V_1 = B \cap S_1$ tel que $card(V_1) > card(\Gamma(V_1))$.

Remarque : Quand $card(S_1) = card(S_2)$, le théorème de Hall devient : Le graphe biparti G admet un couplage parfait si et seulement si $\forall V_1 \subset S_1,\ card(\Gamma(V_1)) \geq card(V_1)$.

TP5 : couplage_vers_flot

TP5 : flot_vers_couplage

TP5 : couplage_maximum_biparti

Couplage de poids minimal dans un graphe biparti : problème d'affection

On suppose que G vérifie les deux conditions suivantes :

1. $card(S_1) = card(S_2)$ (il y a autant d'agents que de tâches à réaliser) ;
2. le graphe est biparti complet (il y a une arête entre chaque élément de S_1 et S_2).

[Définition] poids de K : Soit $G = (S, A)$ est un graphe pondéré avec fonction de poids w, et K un couplage de G. On appelle poids de K le réel $w(K) = \sum_{x-y \in K} w(x{-}y)$.

On dit que K est un couplage parfait de poids minimal de G si K est un couplage parfait et $w(K) = \min\{w(K') \mid K' \ couplage \ pafait \ de \ G\}$.

[Définition] fonction de potentiel : Soit $G = (S, A)$ un graphe pondéré (avec fonction de poids w) et $p : S \to \mathbb{R}$. On dit que p est une fonction de potentiel pour G si

$$\forall x-y \in A, p(x) + p(y) \leq w(x-y)$$

Dans les graphes bipartis, on préfère bien faire la différence entre les sommets de S_1 et les sommets de S_2. On utilisera plutôt deux fonctions, une fonction $p : S_1 \to \mathbb{R}$ et $q : S_2 \to \mathbb{R}$. Ainsi, fonctions p et q forment une fonction de potentiel pour le graphe biparti G si :

$$\forall x-y \in A \; avec \; x \in S_1 \; et \; y \in S_2, \;\; p(x) + q(y) \leq w(x-y)$$

[Proposition] : Si p et q sont des potentiels pour un graphe biparti G, alors pour tout couplage parfait K de G, on a

$$w(K) = \sum_{\substack{x-y \in K \\ x \in S_1, y \in S_2}} w(x-y) \geq \sum_{\substack{x-y \in K \\ x \in S_1, y \in S_2}} p(x) + q(y)$$

[Théorème] : Soit $G = (S, A)$ un graphe biparti, de partition associée (S_1, S_2) et p et q des potentiels pour G. On note $G_{p,q} = (S, A_{p,q})$ le graphe qui a les mêmes sommets que G et pour ensemble d'arêtes

$$A_{p,q} = \{x-y \in A \mid x \in S_1, \; y \in S_2 \; et \; p(x) + q(y) = w(x, y)\}$$

Si K est un couplage parfait de $G_{p,q}$, alors K est un couplage de poids minimal de G.

Algorithme hongrois

- **Données :** un graphe biparti G vérifiant les propriétés 1 et 2 , avec une fonction de poids entiers w.
- **Initialisation :** Pour tout $x \in S_1$, on pose $p(x) = \min_{y \in S_2} w(x-y)$. Pour tout $y \in S_2$, on pose $q(y) = 0$.
- **Boucle:**
  - On cherche un couplage maximum (pour le nombre d'arêtes) du graphe $G_{p,q}$.
    - Si K est un couplage parfait, alors c'est un couplage de poids minimal de G .
    - Sinon, la recherche du couplage maximum fournit un ensemble $V_1 \subset S_1$ tel que $card(V_1) > card(\Gamma_{G_{p,q}}(V_1))$.
  - On pose $\varepsilon = \min_{x \in V_1, y \in S_2 \setminus \Gamma_{G_{p,q}}(V_1)} w(x-y) - p(x) - q(y)$. On change alors p et q ainsi :

$$\forall x \in S_1, \; p(x) = \begin{cases} p(x) + \epsilon & si \; x \in V_1 \\ p(x) & sinon \end{cases}$$

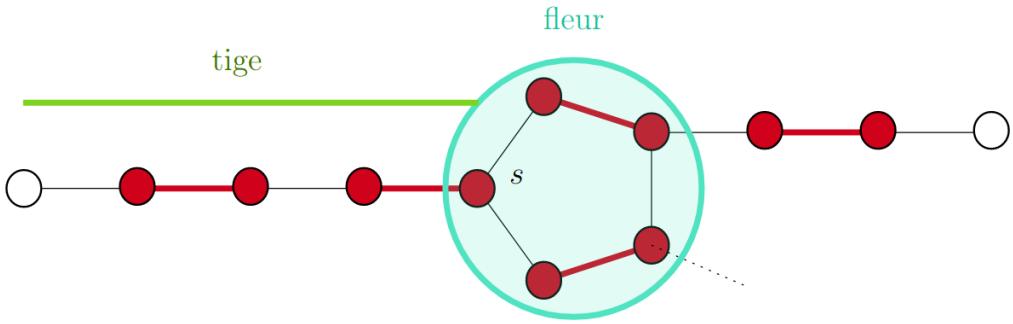
$$\forall y \in S_2, \; q(y) = \begin{cases} q(y) - \varepsilon & si \; y \in \Gamma_{G_{p,q}}(V_1) \\ q(y) & sinon \end{cases}$$

- **Résultat :** un couplage parfait K de poids minimal.

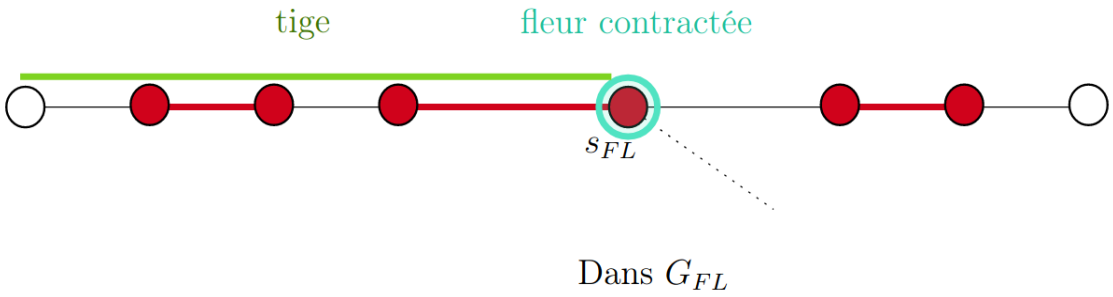
La complexité de l'algorithme hongrois est $O(n^4)$.

Couplage maximum dans un graphe quelconque

[Définition] fleur : Soit G un graphe et K un couplage de G. Une fleur FL est un cycle dans G à $2k + 1$ arêtes, avec exactement k arêtes dans le couplage K, et qui possède un sommet s tel qu'il existe un chemin alterné de longueur paire (la "tige") de s à un sommet exposé.



[Définition] le graphe contracté : Soit G un graphe, K un couplage de G et FL une fleur. On définit le graphe contracté G_{FL} obtenu en réunissant tous les sommets de FL en un seul sommet s_{FL}. Toute arête de G entre un sommet x hors de FL et un sommet y de FL devient dans G_{FL} une arête entre x et s_{FL}. À partir du couplage K défini sur G, on obtient un couplage K' défini sur G_{FL}.



Dans G_{FL}

[Définition] forêt : On appelle forêt tout graphe dont les composantes connexes sont des arbres.

Algorithme d'Edmonds pour trouver un chemin K-augmentant

- **Données :** Un graphe G, un couplage K de G.
- **Initialisation :** L'ensemble des sommets visités est vide ; l'ensemble des arêtes visitées contient les arêtes du couplage. La forêt F contient un arbre pour chaque sommet exposé, et chacun de ces sommets est la racine de son arbre.
- **Boucle:** Tant qu'il existe un sommet non visité x dans la forêt qui est à distance paire de sa racine ; et tant qu'il existe une arête non visitée $a = x\text{–}y$ contenant x, faire :
  - Si y n'est pas dans la forêt :
    - On va agrandir la forêt. Comme y n'est pas un sommet exposé, il est dans une arête $y\text{–}z$ du couplage K. Le sommet z n'est pas non plus dans la forêt. On ajoute alors les sommets y et z à la forêt (avec les arêtes $x\text{–}y$ et $y\text{–}z$) en mettant à jour les fonctions "précédesseur", "racine" et "distance".
  - Sinon (si y est dans la forêt) :
    - Si la distance de y à sa racine $racine(y)$ est paire:
      - Si x et y ne sont pas dans le même arbre de la forêt : On obtient un chemin augmentant $racine(x)\text{–}\cdots\text{–}x\text{–}y\text{–}\cdots\text{–}racine(y)$: renvoyer ce chemin augmentant.
      - Sinon (si x et y sont dans le même arbre de la forêt) : on obtient une fleur FL, formée par l' arête $x\text{–}y$ et le chemin de y à x dans la forêt. on contracte la fleur pour obtenir

un graphe G_{FL} et un couplage K', on recherche un chemin K'-augmentant dans le graphe G_{FL} (en faisant $trouve\_chemin\_augmentant(G_{FL}, K')$) , puis on relève ce chemin pourobtenir un chemin K-augmentant de G. On renvoie alors ce chemin K-augmentant.

- Sinon (si la distance de y à sa racine $racine(y)$ est impaire) : ne rien faire.

🔹 Enfin, on ajoute l'arête x–y dans l'ensemble des arêtes visitées. Quand on a visité toutes les arêtes contenant x, on ajoute x dans l'ensemble des sommets visités.

▪ **Résultat :** Si la boucle se termine sans renvoyer de chemin K-augmentant, c'est qu'il n'en existe pas (admis). On renvoie alors un chemin vide.

Parcours d'un graphe

Parcours d'un arbre

Soit T un arbre, et x_0 un sommet. On peut voir T comme un arbre de racine x_0.

[Définition] parant et fils : Si x est un sommet de l'étage i, on appelle **parent de** x l'unique sommet y de l'étage $i-1$ tel que y–x est une arête de l'arbre. Si x n'a pas de parent, x est la racine x_0. Si x est un sommet de l'étage i, on appelle **fils de** x les sommets z de l'étage $i+1$ tel que x–z est une arête de l'arbre. Si x n'a pas de fils, on dit que x est une feuille de l'arbre.

[Définition] Parcours en profondeur d'un arbre : On appelle parcours en profondeur de T une exploration de l'arbre où on explore chaque branche jusqu'au bout (jusquèà une feuille) avant de revenir en arrière. Précisement, cela correspond à l'utilisation l'algorithme suivant :

Algorithme "explorer en profondeur à partir du sommetx", noté **explorer(G,x)** :

- Visiter le sommet x;
- Pour tout sommet z fils de x: faire explorer(G,z)

[Définition] Parcours en largeur d'un arbre : On appelle parcours en largeur de T une exploration de l'arbre où on explore chaque étage de l'arbre avant de passer au suivant. Un parcours en largeur d'un arbre peut être codé à l'aide d'une « file FIFO » (First In → First Out) contenant les sommets à explorer. Initialement, on met x_0 dans la file. À chaque étape, on retire (et visite) le sommet x du début de la file, et on ajoute ses fils à la fin dela file.

Parcours d'un graphe connexe

[Définition] Parcours en profondeur d'un graphe : On appelle parcours en profondeur d'un graphe G un parcours de G obtenu à l'aide d'une « pile LIFO » (Last In→First Out) de la manière suivante : Initialement, on met x_0 dans la pile. À chaque étape, on désempile (et visite) le sommet x en haut de la pile, et on empile les voisins de x pas visités et pas dans la pile. Exemple: **bon_exite_chemin**

[Proposition] : Lorsqu'on fait un parcours en profondeur d'un graphe connexe G en utilisant une pile, on peut stocker les arêtes qu'on visite, par exemple en utilisant une fonction prédécesseur. On obtient alors un arbre couvrant de G, et le parcours en profondeur de G est un parcours en profondeur de cet arbre.

[Définition] Parcours en largeur d'un graphe : On appelle parcours en largeur de G une exploration de G où on commence par un sommet x_0, puis on visite les sommets de sorte que : pour tout $i \geq 0$, on visite tous les sommets z tel que $d(x_0, z) = i$ avant de visiter les sommets à distance $i+1$ de x_0. Un parcours en profondeur d'un graphe peut être codé à l'aide d'une «

file FIFO » (First In→First Out) contenant les sommets à explorer. Initialement, on met x_0 dans la file. À chaque étape, on retire (et visite) le sommet x du début de la file et on le visite, et on ajoute ses voisins non visités à la fin de la file.

Application : amélioration de l'algorithme de Ford-Fulkerson

en utilisant un arcoursen largeur : ainsi, à chaque étape, on trouve **le plus court chemin augmentant** de s à p. On obtient l'amélioration de l'algorithme de Ford-Fulkerson suivante :

Algorithme de Edmonds–Karp :

- **Données :** un graphe orienté $G = (S, A)$, une fonction de capacité $c : A \to \mathbb{N}$.
- **Initialisation :** on initialise le flot f par $f(x, y) = 0$ pour tout $(x, y) \in A$.
- **Boucle :**
  - Chercher un chemin augmentant avec un parcours en largeur : Initialement, on met la source s dans la file. À chaque étape, on retire le sommet x du début de la file et on le visite. On ajoute à la fin de la file les sommets z non visités voisins de x tels que $c(x, z) > f(x, z)$ ou $f(z, x) > 0$. Pour ces sommets, le précédesseur de z est alors x.
  - Si p n'est pas dans les sommets visités, il n'y a pas de chemin augmentant de s à p et on aobtenu un flot maximal ;
  - Sinon, la fonction prédécesseur nous donne un chemin agmentant $x_0 = s - x_1 - \cdots - x_l = p$ de s à p. On calcule sa valeur ε et on fait pour tout $i \in [0, l-1], i \in \mathbb{Z}$:
  - ◦ Si $c(x_i, x_{i+1}) - f(x_i, x_{i+1}) > f(x_{i+1}, x_i)$, changer $f(x_i, x_{i+1})$ en $f(x_i, x_{i+1}) + \varepsilon$;
  - ◦ Sinon, changer $f(x_{i+1}, x_i)$ en $f(x_{i+1}, x_i) - \varepsilon$.

La complexité de l'algorithme de Edmonds–Karp est $O(n \times m^2)$.

Chemins et cycles hamiltoniens

[Définition] chemin hamiltonien et cycle hamiltonien : Soit G un graphe. On appelle **chemin hamiltonien** dans G tout chemin (avec sommets distincts) qui passe par tous les sommets de G. On appelle **cycle hamiltonien** dans G tout cycle (avec sommets distincts) qui passe par tous les sommets de G.

[Proposition] : Soit G un graphe connexe à n sommets avec $n \geq 3$. On suppose que pour tous sommets x et y non reliés par une arête,

$$deg(x) + deg(x) \geq n$$

Alors G possède un cycle hamiltonien.

[Proposition] : Soit G un graphe connexe à n sommets avec $n \geq 3$, et notons $d_1 \leq d_2 \leq \cdots \leq d_n$ la liste des degrés des sommets de G.

- Si on a

$$\forall k \in [1, \frac{n}{2}[, k \in \mathbb{Z},\ d_k \leq k \Rightarrow d_{n-k} \geq n - k$$

  alors G possède un cycle hamiltonien.
- Si on a

$$\forall k \in [1, \frac{n-1}{2}[, k \in \mathbb{Z},\ d_k \leq k - 1 \Rightarrow d_{n-k+1} \geq n - k$$

alors G possède un chemin hamiltonien.