L3 Informatique

Algorithmique avancée
Démonstrations

Etienne Birmelé
etienne.birmele@parisdescartes.fr

automne 2016

Chapitre 1

Notion de graphe

Propriété 1.1. *Un graphe non orienté vérifie*

$$\sum_{v \in V(G)} d(v) = 2m.$$

Un graphe orienté vérifie

$$\sum_{v \in V(G)} d^+(v) = \sum_{v \in V(G)} d^-(v) = m.$$

Démonstration. Il suffit pour cela de compter le nombre de couples (noeud,arête) qui sont incidents. Si on compte ces couples en regroupant ceux contenant le même noeud, on obtient que leur nombre est de $\sum_{v \in V(G)} d(v)$. Si on les compte en les regroupant par arête commune, ils sont au nombre de $2m$.

Le raisonnement est similaire dans le cas orienté. □

Chapitre 2

Parcours de graphe

2.1 Arbres

Théorème 2.1. *Tout arbre enraciné est un arbre. De plus, pour tout arbre T et tout sommet de $r \in V(T)$, T peut être construit comme un arbre enraciné de racine r.*

Démonstration.

1. Soit T un arbre enraciné. T est acyclique et connexe par construction. En effet, pour toute paire de sommets u et v a un plus ancêtre commun le plus récent a (r étant un ancêtre commun à tous). En passant par a, on peut construire un chemin reliant u et v.

   Soit vw une arête de T, où v est le père de w. Alors, de par l'algorithme de construction de T, aucune autre arête ne relie l'ensemble formé par w et ses descendants au reste du graphe. La suppression de vw rompt donc l'existence d'un chemin entre v et w. Ceci étant vrai pour toute arête de T, T est bien un graphe connexe minimal.

2. Soit r un sommet de T. On lui donne le niveau 0. On peut alors reproduire l'algorithme de construction d'un arbre enraciné en sélectionnant comme descendant de v tout sommet voisin de v dans T et non encore considéré. On obtient un arbre enraciné G.

   G est un sous-graphe de T (puisque toute arête de G est une arête de T). Supposons que $V(G) \neq V(T)$. T étant connexe, il existe une arête reliant reliant un sommet $v \in V(G)$ à un sommet $w \in V(T) \setminus V(G)$. Mais alors l'algorithme de construction de G aurait considéré cette arête et inclus w dans $V(G)$. On a donc bien $V(G) = V(T)$.

   G étant connexe par construction, on a alors $G = T$ par minimalité de T.

□

Théorème 2.2. *Un graphe est connexe si et seulement si il admet un arbre couvrant.*

Démonstration.
1. si G admet un arbre couvrant, G est connexe de façon évidente.
2. si G est connexe, soit $\mathcal{T}$ l'ensemble des sous-graphes de G qui sont des arbres. Cet ensemble est non-vide car il contient les sous-graphes réduits à un sommet. Soit T un élément maximal de $\mathcal{T}$.

   Supposons que $V(T) \neq V(G)$. Soit $u \notin V(T)$ et $v \in V(T)$. Il existe un chemin $(u = w_0, \dots, v = w_p)$ dans G reliant u et v. Soit k le plus petit indice tel que $u_k \in V(T)$. Alors $T + (u_{k-1}u_k)$ est un arbre et un sous-graphe de G, ce qui contredit la maximalité de T.

□

Propriété 2.1. *Un graphe connexe est un arbre si et seulement si il a $n-1$ arêtes.*

Démonstration.
1. Considérons un arbre T, vu comme un arbre enraciné. La construction d'un arbre enraciné entraîne la création d'exactement $n-1$ arêtes puisque l'ajout de tout sommet, hormis la racine, engendre la création d'une arête.
2. Considérons un graphe connexe G à $n-1$ sommets. Sa connexité entraîne qu'il admet un arbre couvrant T. Or T a exacgtement $n-1$ arêtes d'après le point précédent. Toutes les arêtes de G sont donc des arêtes de T, d'où $G = T$.

□

2.2 Parcours en largeur

Propriété 2.2. *BFS construit un arbre enraciné en v contenant tous les sommets de la composante connexe de v.*

Démonstration. Supposons qu'il existe un contre-exemple, soit w le sommet non-atteint dont la distance à v est minimale. Soit k cette distance : il existe $u_1 \dots u_{k-1}$ tels que $v, u_1 \dots u_{k-1}, w$ est un chemin. Par minimalité de w, u_{k-1} est visité. Au moment où u_{k-1} est le sommet courant, et w est visité. □

Propriété 2.3. *BFS est de complexité $\mathcal{O}(m)$.*

Démonstration. Chaque sommet est une seule fois considéré comme le sommet courant et un nombre constant d'opérations est faite pour chacun de ses voisins. Le nombre total d'opérations est donc en $\mathcal{O}(\sum_v d(v))$. Or $\sum_v d(v) = 2m$. □

On considère un graphe non valué G. Soit T un arbre BFS enraciné en v. Pour tout sommet u, on note $niv(u)$ le niveau de u dans T.

Propriété 2.4. *Pour tout $(u, v) \in E(G)$, $|niv(u) - niv(v)| \leq 1$.*

Démonstration. Supposons qu'il existe un contre-exemple, c'est-à-dire u et v tels que (u, v) est une arête de G et u et v diffèrent d'au moins deux niveaux dans T. Quitte à échanger leurs noms, on peut supposer que u est visité en premier, c'est à dire $niv(v) \geq niv(u) + 2$.

Au moment où u est le sommet courant, v est considéré dans la boucle des voisin de u. Comme $niv(v) \geq niv(u) + 2$, v n'a pas encore été visité. Il est donc ajouté dans l'arbre comme fils de u, ce qui contredit $niv(v) \geq niv(u) + 2$. □

Propriété 2.5. *Pour tout $u \in V(G)$, $niv(u) = d(u, v)$.*

Démonstration. Le chemin entre v et u est de longueur $niv(u)$ donc $d(u, v) \leq niv(u)$.

Supposons qu'il existe un chemin plus court dans G. Alors ce chemin doit forcément passer par une arête qui saute un niveau, c'est-à-dire relie deux sommets dont la différence de niveaux est d'au moins deux. Ceci est impossible d'après la proposition précédente. □

2.3 Parcours en profondeur

Propriété 2.6. *DFS construit un arbre enraciné en v contenant tous les sommets de la composante connexe de v.*

Démonstration. Soit w un sommet dans la même composante que v. Il existe un chemin $u_0 = v, u_1 \dots u_{k-1}, u_k = w$ reliant v à w.

Le somme u_i n'est dépilé que si tous ses voisins ont été visités. Par conséquent, si u_i est visité, u_{i+1} a forcément été visité aussi quand l'algorithme se termine. De proche en proche, on en déduit que w est forcément visité. □

Propriété 2.7. *DFS est de complexité $\mathcal{O}(m)$.*

Démonstration. Le pseudo-code tel qu'il est écris dans le cours n'est pas linéaire en le nombre d'arêtes de façon évidente puisqu'il y a un *For* et un *If* imbriqués donnant un algorithme quadratique s'il est mal implémenté.

Cependant, en codant intelligemment, on peut éviter que la boucle *If* reteste des sommets w déjà testés, et chaque arête du graphe n'est ainsi parcourue qu'une fois. Chaque ajout d'une arête s'accompagne d'un nombre constant d'opérations. □

Propriété 2.8. *Soit (u, v) une arête de G, u étant le premier sommet découvert par le DFS.*

Alors u est un ancêtre de v.

Démonstration. Les ancêtres de u dans l'arbre rendu par le DFS sont les sommets qui sont mis sur la pile tant que u lui appartient. Or, par hypothèse, v n'appartient pas encore à la pile quand u y est ajouté. De plus, u n'est dépilé que lorsque tous ses voisins ont été visités, c'est-à-dire ont été empilés. □

2.4 Arbre couvrant de poids minimal

2.4.1 Algorithme de Kruskal

Propriété 2.9. *L'algorithme de Kruskal renvoie bien, pour un graphe connexe, un arbre couvrant de poids minimal.*

Démonstration. **le résultat est un arbre :** F est acyclique par construction. En effet, une arête n'est ajoutée que si elle relie deux sommets de couleur différente, c'est-à-dire appartenant à des arbres différents avant l'ajout de l'arête.

Si c'est une forêt à la fin de l'éxécution, soit T_1 et T_2 deux de ses arbres tels qu'il existe une arête entre T_1 et T_2 dans G. Au moment où cette arête a été examinée, elle aurait du être ajoutée à F. Donc F est un arbre couvrant.

il est de poids minimal : Montrons par récurrence qu'à toute étape il existe un arbre couvrant de poids minimal qui contient F. Le fait que c'est encore vrai quand l'algorithme se termine assure que l'arbre final est de poids minimal.

L'étape d'initialisation est évident puisque tout arbre couvrant contient la forêt couvrante sans arêtes.

Supposons que la proposition est vraie avant l'ajout d'une arête e et soit T l'arbre couvrant de poids minimal qui contient F à ce moment-là. Si $e \in E(T)$, $F + e$ est encore inclus dans T et la propriété reste vraie après l'ajout de e.

Si $e \notin E(T)$, $T+e$ contient un cycle. Il y a donc une arête f qui appartient à T mais pas à $F + e$ puisque celui-ci est acyclique. Alors $T - e + f$ est connexe, couvrant et contient $n - 1$ arêtes. En d'autres termes, c'est un arbre couvrant. Pa minimalité de T, $T - e + f$ est de poids au moins égal à T. Ceci n'est possible que si le poids de e est inférieur au poids de f.

Or, f n'appartient pas à $F + e$, donc est considéré par l'algorithme après e, ce qui implique que e et f ont en fait le même poids. Par conséquent, $T - e + f$ est aussi un arbre de poids minimal et il contient $F + e$. La propriété est donc bien vérifiée après l'ajout de e.

□

Propriété 2.10. *Sa complexité est de* $\mathcal{O}(m \log m)$.

Démonstration. Le tri initial des arêtes se fait en $\mathcal{O}(m \log m)$.

Le reste est également en $\mathcal{O}(m \log m)$, en raison de la propriété suivante (et des slides). □

Propriété 2.11. *La hauteur* $h(T)$ *d'un arbre* T *avec* s *sommets, construit par une suite d'applications de* $UNION$*, est majoré par* $1 + \log_2 s$.

Démonstration. Par récurrence.

— vrai pour $s = 1$

— Supposons que la propriété est vraie pour toutes les valeurs inférieures à s. Soit T de taille s obtenu à partir de deux arbres T_1 et T_2 de taille x et $s-x$. Quitte à les inverser, on peut supposer que $x \leq \frac{n}{2}$.
Alors

$$\begin{aligned} h(T) \leq & \quad max(h(T_1), 1 + h(T_2)) \\ \leq & \quad max(1 + \log_2(n-x), 2 + \log_2(x)) \end{aligned}$$

Or $\log_2(n-x) \leq \log_2(n)$ et $\log_2(x) \leq \log_2(\frac{n}{2}) \leq \log_2(n) - 1$.
On obtient donc $h(T) \leq 1 + \log_2(n)$. □

2.4.2 Algorithme de Prim

Propriété 2.12. *L'algorithme de Prim renvoie bien, pour un graphe connexe, un arbre couvrant de poids minimal.*

Démonstration. Le preuve est essetiellement la même que pour l'algorithme de Kruskal.

le résultat est un arbre couvrant : T est un arbre par construction puisque toute arête relie l'arbre existant à un sommet non encore découvert.

Supposons qu'il n'est pas couvrant. Alors $V(T)$ et $V(G) \setminus V(T)$ sont non vides. Comme G est connexe, il existe au moins une arête entre T et $G \setminus T$. L'algorithme ne renvoie donc pas T en raison de la condition du *While*.

il est de poids minimal : Montrons par récurrence qu'à toute étape il existe un arbre couvrant de poids minimal qui contient T. Le fait que c'est encore vrai quand l'algorithme se termine assure que l'arbre final est de poids minimal.

L'étape d'initialisation est évidente puisque tout arbre couvrant contient le sommet de départ.

Supposons que la proposition est vraie avant l'ajout d'une arête e et soit S l'arbre couvrant de poids minimal qui contient T à ce moment-là. Si $e \in E(S)$, $T + e$ est encore inclus dans S et la propriété reste vraie après l'ajout de e.

Si $e \notin E(S)$, $S + e$ contient un cycle. Soit f l'autre arête de ce cycle qui relie $V(T)$ et $V(G) \setminus V(T)$. f appartient à S mais pas à $T + e$ puisque celui-ci est acyclique. Alors $S - e + f$ est connexe, couvrant et contient $n-1$ arêtes. En d'autres termes, c'est un arbre couvrant. Pa minimalité de S, $S - e + f$ est de poids au moins égal à S. Ceci n'est possible que si le poids de e est inférieur au poids de f.

Or, f relie T et $G \setminus T$ et n'appartient pas à $T + e$, donc est considéré par l'algorithme après e. Ceci implique que e et f ont en fait le même poids. Par conséquent, $S - e + f$ est aussi un arbre de poids minimal et il contient $T + e$. La propriété est donc bien vérifiée après l'ajout de e.

□

Propriété 2.13. *Sa complexité est de $\mathcal{O}(mn)$ si le graphe est codé en liste ou matrice d'adjacence.*

Elle peut être abaissée en $\mathcal{O}(m + n\log n)$ en le codant à l'aide de tes de Fibonacci.

Démonstration. Admis □

2.5 Algorithme de Dijkstra

Propriété 2.14. *L'algorithme de Dijkstra renvoie bien la distance de tout sommet au sommet d'origine.*

Démonstration. On montre par récurrence qu'à tout moment, le sommet ajouté à T vérifie bien que $dist\_finale(v) = d(u, v)$.

- C'est évident au moment de l'initialisation car $dist\_finale(u) = 0 = d(u, u)$.
- Supposons que c'est le cas pour tous les sommets ajoutés à T avant v. Soit P formé de $w_0 = u, \ldots, w_p = v$ un chemin de poids minimal entre u et v et w_l le dernier sommet de T sur ce chemin. Alors,
  Comme $dist\_finale(v)$ est la longueur du chemin entre u et v dans T, il est évident que $d(u, v) \leq dist\_finale(v)$. De plus,

$$\begin{aligned} dist\_finale(v) &= dist\_prov(v) \text{ suite au choix de } v \\ &\leq dist\_prov(w_{l+1}) \text{ car } v \text{ est choisi et non } w_{l+1} \\ &\leq dist\_finale(w_l) + \omega(w_l, w_{l+1}) \text{ en raison du procédé de calcul de } dist\_prov \\ &\leq \omega(P) \\ &\leq d(u, v) \end{aligned}$$

  v vérifie donc bien la propriété.

□

Propriété 2.15. *La complexité est polynomiale. Elle est en $\mathcal{O}(n^2)$ pour un code naïf, et peut être réduite à $\mathcal{O}(m + n\log n)$ à l'aide de tas binaire et même $\mathcal{O}(m + n\log n)$ à l'aide de tas de Fibonacci.*

2.6 Cycles couvrants

Propriété 2.16. *De chaque côté d'un pont se trouve au moins un sommet de degré impair.*

Démonstration. Soit $e = (u, v)$ un pont et G_u et G_v les deux composantes connexes de $G \setminus e$.

Si u est de degré impair dans G, $V(G_u)$ contient bien un sommet de degré impair. Si u est de degré pair dans G, il est de degré impair dans G_u. Or, la somme des degrés dans G_u est paire (c'est le cas dans tout graphe car elle vaut deux fois la somme des arêtes). Il existe donc un autre sommet de degré impair dans G_u.

Le même raisonnement s'applique pour G_v. □

Propriété 2.17. *Si tous les sommets ont un degré pair, l'algorithme de Fleury renvoie un cycle eulérien.*

Démonstration. Cet algorithme renvoie par définition une suite d'arêtes sans répétition. De plus, par un argument de parité des degrés des sommets, il ne peut s'arrêter que quand $v_m = v_0$ (sinon, on arrive sur un sommet de degré pair, donc en supprimant l'arête d'arrivée, il en reste une pour repartir, et en supprimant celle-ci, le sommet est de nouveau de degré pair).

Il reste donc à démontrer que toutes les arêtes sont parcourues. Si ce n'est pas le cas, soit S l'ensemble des arêtes non parcourues et H le graphe induit par ces arêtes. Comme G est connexe, H a des sommets en commun avec le parcours W_m final. Soit k le plus grand entier tel que $v_k \in H$ et $v_{k+1} \notin H$.

Alors à l'étape k, toutes les arêtes sortant de v_k étaient des ponts dans $G_k = G - \{e_1, \ldots, e_k\}$, et ce nombre de ponts est au moins égal à 2 (e_{m+1} et au moins un autre). Il doit donc y avoir au moins trois sommets de degré impair dans G_k, ce qui est impossible car seuls v_0 et v_k peuvent être de degrés impairs. □

Propriété 2.18. *Un graphe contient un parcours eulérien si et seulement si au plus deux de ses sommets ont un degré impair.*

Démonstration. Si G admet un parcours eulérien, seuls son point de départ et son point d'arrivée peuvent être de degré impair, en développant le même argument que dans la preuve du théorème précédent.

Si G ne contient aucun sommée de degré impair, il admet un circuit eulérien, qui est un cas particulier de parcours eulérien. Comme $\sum_v d(v) = 2e(G)$, il n'est pas possible d'avoir un seul sommet de degré impair (la somme totale devant être paire). Si G contient deux sommets u et v de degrés impairs, soit H obtenu en ajoutant une arête (supplémentaire) entre u et v. H n'a plus de sommet de degré impair et contient donc un circuit eulérien C. En retirant (u, v) de C, on obtient un parcours eulérien. □

Théorème 2.3. *Le problème de décision du cycle hamiltonien est NP-complet, c'est-à-dire qu'il n'existe pas d'algorithme polynomial permettant de le résoudre (à moins que tout problème puisse être résolu en temps polynomial, le fameux P=NP).*

Démonstration. 1. Le problème du chemin hamiltonien se réduit au problème du cycle hamiltonien. En effet, soit G' obtenu en ajoutant un sommet u relié à tous les sommets de G. Alors G' admet un cycle hamiltonien si et seulement si G admet un chemin hamiltonien.

2. Le problème du chemin hamiltonien orienté se réduit au problème du chemin hamiltonien. En effet, soit G un graphe orienté. On peut créer G' non orientéen en remplacant chaque sommet v par un chemin non orienté (v_{in}, v, v_{out}) et en reliant u_{out} et v_{in} si et seulement si $(u, v) \in E(G)$. G admet un chemin hamiltonien orienté si et seulement si G' admet un chemin hamiltonien et G' et de taille polynomial en la taille de G.

3. On va réduire le problème 3-SAT au problème du chemin hamiltonien orienté. Pour cela soit $x_1, \ldots, x_n$ et $C_1, \ldots, C_m$ une instance de 3-SAT. On construit un graphe orienté G de la façon suivante :
   — pour toute variable x_i, on met en ligne $2m+2$ sommets $v_{i1}, \ldots, v_{i,2m+2}$ et on ajoute une arête dans chaque sens entre cahque paire de voisins.
   — pour tout i, on met des arêtes arête de v_{i1} et $v_{i,2m+2}$ vers $v_{i+1,1}$ et $v_{i+1,2m+2}$.
   — on ajoute un sommet v_{start} pointant vers $v_{1,1}$ et $v_{1,2m+2}$
   — on ajoute un sommet v_{end} vu par $v_{n,1}$ et $v_{n,2m+2}$

   A ce stade, le graphe admet 2^n chemins hamiltoniens de v_{start} à v_{end} consistant à parcourir chaque ligne de gauche à droite $x_i = 1$ ou de droite à gauche $(x_i = 0)$. Il y a donc bijection entre les chemins hamiltoniens et les assignations des x_i.

   On ajoute alors, pour chaque clause C_j un sommet u_j et les arêtes suivantes :
   — si x_i apparait dans C_j, on ajoute les arêtes $(x_{i,2j}, u_j)$ et $(u_j, x_{i,2j+1})$. Si on assigne $x_i = 1$, on pourra donc inclure u_j dans le chemin hamiltonien correspondant.
   — si $\overline{x_i}$ apparait dans C_j, on ajoute les arêtes $(x_{i,2j+1}, u_j)$ et $(u_j, x_{i,2j})$. Si on assigne $x_i = 0$, on pourra donc inclure u_j dans le chemin hamiltonien correspondant.

   On voit donc qu'une assignation telle que C est vérifié donne lieu à un chemin hamiltonien entre v_{start} et v_{end} et inversement. La taille de graphe étant polynomiale en n et m, le problème du chemin hamiltonien orienté est donc NP-complet.

□

Chapitre 3

Flots dans les graphes

Propriété 3.1. *Pour tout flot f dans G, $f^+(S) - f^-(S) = f^-(P) - f^+(P)$. Cette valeur est notée $val(f)$.*

Démonstration. $A = V(G) \setminus (S \cup P)$. On a $f^+(A) = f^-(A)$. Or $f^+(A) = f^-(S) + f^-(P)$ et $f^-(A) = f^+(S) + f^+(P)$ □

Propriété 3.2. Problème 1 : *Soit (G, c) un graphe dirigé valué, $S \subset V(G)$, $P \subset V(G)$. Déterminer un flot maximal de S vers P.*

Problème 2 : *Soit (H, c) un graphe dirigé valué, $s \in V(H)$, $p \in V(H)$. Déterminer un flot maximal de s vers p.*

Les problèmes 1 et 2 sont équivalents.

Démonstration. Le problème 2 est un cas particulier du problème 1. On peut donc le résoudre si on sait résoudre le problème 1.

Soit (G, c, S, P) une instance du problème 1. On construit un graphe H en ajoutant à G un nouveau sommet s et un nouveau sommet p. Pour tout sommet v de S, on ajoute une arête (s, v) de capacité infinie. De même, pour tout sommet v de P, on ajoute une arête (v, p) de capacité infinie.

DESSIN

Tout flux de G entre S et P peut alors être étendu en un flux f' de H en posant, pour tout $e = (s, v)$, $f'(e) = f^+(v) - f^-(v)$ et, pour tout $e = (v, p)$, $f'(e) = f^-(v) - f^+(v)$. On a alors $val(f') = \sum_{v \in S} f'(e, v) = \sum_{v \in S}(f^+(v) - f^-(v)) = val(f)$.

Inversement, tout flux de H peut être vu comme un flux de G et s'il est maximal pour H, il est maximal pour G puisque les arêtes de $G \setminus H$ ne peuvent pas être les arêtes limitantes. □

Propriété 3.3. *Pour toute coupe $K = (A, \overline{A})$ et tout flot f, $val(f) = f^+(A) - f^-(A)$.*

Démonstration. Pour tout sommet de A différent de s, $f^+(v) - f^-(v) = 0$. De plus, $f^+(s) - f^-(s) = val(f)$. En sommant sur tous les sommets de A, on obtient le résultat. □

Propriété 3.4. *Pour toute coupe K et tout flot f, $val(f) \leq cap(K)$. En particulier,* $\max_f val(f) \leq \min_K cap(K)$.

Démonstration. Soit $K = (A, \overline{A})$ et f un flot.
$val(f) = f^+(A) - f^-(A)$. Or, $f^+(A) \leq cap(K)$ et $f^-(A) \geq 0$. □

Propriété 3.5. *Un flot f est maximal si et seulement si il n'y a pas de chemin non-saturé entre s et p.*

Démonstration. S'il existe un chemin non-saturé, on peut construire un flot de valeur supérieure comme évoqué plus haut.

Supposons maintenant qu'il n'existe pas de chemin non-saturé entre s et p. Soit A l'ensemble des sommets v tels qu'il existe un chemin non-saturé de s vers v. Alors $s \in A$ et $p \in \overline{A}$, ce qui implique que $K = (A, \overline{A})$ est une coupe. Alors :

- toute arête e de $v \in A$ vers $\overline{v} \in \overline{A}$ est saturée, c'est-à-dire que $f(e) = c(e)$. En effet, si $c(e) - f(e) > 0$, le chemin non-saturé de s vers v peut être prolongé en un chemin non-saturé de s vers $\overline{v}$. Ceci contredit $\overline{v} \in \overline{A}$. Par conséquent, $f^+(A) = cap(K)$.
- toute arête e de $\overline{v} \in \overline{A}$ vers $v \in A$ est telle que $f(e) = 0$. En effet, si $f(e) > 0$, le chemin non-saturé de s vers v peut être prolongé en un chemin non-saturé de s vers $\overline{v}$. Ceci contredit $\overline{v} \in \overline{A}$. Par conséquent, $f^-(A) = 0$.

Finalement, $val(f) = f^+(A) - f^-(A) = cap(K)$. On en conclut que f est un flot maximal et K une coupe minimale. □

Théorème 3.1. *Menger Soit S et P deux ensembles de sommets d'un graphe G. Pour tout entier k, soit il existe $k+1$ chemins arêtes-disjoints reliant S à P, soit il existe auplus k arêtes dont la suppression déconnecte S de P.*

Démonstration. Dans le cas orienté, il s'agit de l'application de max-flox, min-cut : un flot de valeur supérieure à k implique l'existence de $k+1$ chemins arêtes-disjoints, une coupe de capacité au plus k correspond à un ensemble d'au plus k arêtes déconnectant S et P.

Le cas non-orienté se déduit du graphe où toutes les arêtes sont remplacées par deux arêtes orientées de direction opposées. □