Master 1 d'Informatique — Recherche Opérationnelle
2010 / 2011

TD n° 1

Généralités sur les graphes

Exercice 1 Soit $G = (V, E)$ un graphe.

(a) Si G est orienté : $|E| = \sum_{x \in V} \delta^-(x) = \sum_{x \in V} \delta^+(x)$.

(b) Si G est non orienté : $\sum_{x \in V} \delta(x) = 2|E|$. En particulier, les sommets de G de degré impair sont en nombre pair.

Exercice 2 Dans un graphe non orienté, il y a toujours deux sommets de même degré.

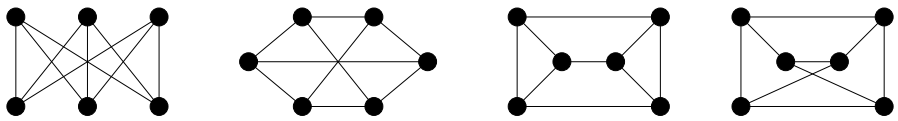
Exercice 3 Le **complémentaire** d'un graphe non orienté G est le graphe $\overline{G}$ défini par : $V(\overline{G}) = V(G)$ et $\forall x, y \in V(\overline{G})$, $(x, y) \in E(\overline{G})$ ssi $(x, y) \notin E(G)$. Prouver ou infirmer les assertions suivantes :

(a) Deux graphes sont isomorphes ssi leurs complémentaires sont isomorphes.

(b) Si G est non connexe alors $\overline{G}$ est connexe.

(c) Si G est connexe alors $\overline{G}$ est non connexe.

Exercice 4 (a) Quels sont ceux des graphes suivants qui sont isomorphes ?



(b) Combien il y a-t-il de graphes non orientés sur un ensemble fixé de 4 sommets ? Combien il y en a-t-il à isomorphisme près ? Lesquels ?

Exercice 5 Prouver le Lemme 4.

Exercice 6 G est connexe ssi pour toute bipartition (X, Y) de $V(G)$, il existe une arête ayant une extrémité dans X et l'autre dans Y.

Exercice 7 Vrai ou faux ? Si chaque sommet d'un graphe non orienté G est de degré 2 alors G est un cycle.

Exercice 8 Un sommet x d'un graphe non orienté connexe G est dit **point d'articulation** de G si $G - x$ est non connexe. Montrer que tout graphe connexe contient au moins deux sommets qui ne sont pas points d'articulation.

Exercice 9 On définit inductivement une classe de graphes non orientés $\mathscr{C}$ par :

1. tout sommet isolé est dans $\mathscr{C}$;
2. si $G \in \mathscr{C}$, tout graphe H obtenu en ajoutant à G un sommet x et une ou plusieurs arêtes adjacentes à x est dans $\mathscr{C}$.

Montrer que $\mathscr{C}$ est exactement la classe des graphes connexes. En déduire une nouvelle preuve de la Proposition 5 (G connexe $\Rightarrow |E(G)| \geq |V(G)| - 1$).

Exercice 10 Un graphe non orienté G est dit **biparti** s'il existe une partition de $V(G)$ en deux ensembles X et Y telle que tout arête de G joint un sommet de X à un sommet de Y. Montrer que G est biparti ssi G ne contient aucun cycle de longueur impaire.

Exercice 11 Tout graphe non orienté connexe admet un sous-graphe couvrant connexe et acyclique (appelé *arbre couvrant* du graphe). Par conséquent, tout graphe non orienté admet un sous-graphe couvrant acyclique (appelé *forêt couvrante* du graphe).

Exercice 12 Combien existe-t-il de graphes orientés (resp. non orientés) à n sommets ?

Exercice 13 Montrer que le nombre d'arbres sur un ensemble fixé de n sommets est n^{n-2}.

Exercice 14 *(Connectivité de sous-graphes et d'extensions de graphes).* On désigne par $\omega(G)$ la *connectivité* d'un graphe non orienté G, c'est à dire le nombre de ses composantes connexes. En particulier, G est connexe ssi $\omega(G) = 1$. Soient $G = (V,E)$ un GNO et I un ensemble de paires de sommets de G. Prouver les assertions suivantes :

1. $\omega(G) \geq \omega(G+I) \geq \omega(G) - |I|$.
2. $\omega(G) \leq \omega(G-I) \leq \omega(G) + |I|$.
3. Si $\omega(G-e) > \omega(G)$ pour tout $e \in I$, alors $\omega(G-I) = \omega(G) + |I|$.
4. $\omega(G) \geq |V| - |E|$.
5. G est acyclique ssi $\omega(G) = |V| - |E|$.

Exercice 15 Prouver le Lemme 8.

Exercice 16 Prouver la Proposition 10.

Exercice 17 Quand on utilise la représentation par matrice d'adjacence, la plupart des algorithmes sur les graphes à n sommets s'exécutent en $O(n^2)$, mais il y a des exceptions. Montrer qu'il est possible de déterminer en $O(n)$ si un graphe orienté contient un **trou noir** – un sommet de degré entrant $n-1$ et de degré sortant 0 –, même si on utilise une représentation par matrice d'adjacence.

Exercice 18 Soit G un graphe non orienté. Montrer que G est un cycle ssi G est non acyclique minimal.

Corrigé du TD n° 1

Généralités sur les graphes

Correction exercice 1.

(*a*) Clairement, $E = \bigcup_{x\in V} Entrants(x) = \bigcup_{x\in V} Sortants(x)$. Comme les ensembles d'arcs $Entrants(x)$ (resp. $Sortants(x)$), $x \in V$, sont deux à deux disjoints : $|E| = \sum_{x\in V} |Entrants(x)| = \sum_{x\in V} |Sortants(x)|$. C'est le résultat cherché.

(*b*) Si H est une orientation du graphe non orienté G, on a clairement : $|E(G)| = |E(H)|$, soit, d'après ce qui précède :

$$|E(G)| = \frac{1}{2}(\sum_{x\in V(H)} \delta_H^-(x) + \sum_{x\in V(H)} \delta_H^+(x)) = \frac{1}{2}\sum_{x\in V(H)} \delta_H(x) = \frac{1}{2}\sum_{x\in V(G)} \delta_G(x).$$

Correction exercice 2. Soit G un graphe non orienté à n sommets. Soit D l'ensemble des degrés des sommets de G. Puisque chaque sommet est relié à au plus $n-1$ autres sommets, $D \subseteq \{0,\ldots,n-1\}$. Si par ailleurs les degrés sont deux à deux distincts, D est de cardinal n et donc, nécessairement, $D = \{0,\ldots,n-1\}$. Mais G ne peut contenir simultanément un sommet de degré 0 et un sommet de degré $n-1$, puisque un sommet de degré $n-1$ est relié à tous les autres sommets du graphe. Donc $|D| < n$ et deux sommets au moins ont même degré.

Correction exercice 3.

(*a*) Clair : tout isomorphisme h de G sur H est aussi un isomorphisme de $\overline{G}$ sur $\overline{H}$ puisque : $(x,y) \in V(\overline{G})$ ssi $(x,y) \notin V(G)$ ssi $(x,y) \notin V(H)$ ssi $(x,y) \in V(\overline{H})$.

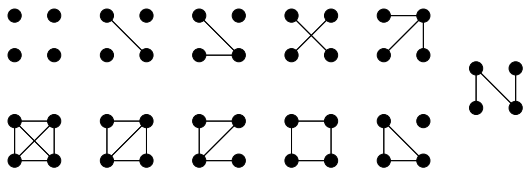
(*b*) Supposons donc G non connexe. Soient x, y deux sommets de $V(\overline{G}) = V(G)$. Si x, y sont dans deux composantes connexes distinctes de G, alors $(x,y) \notin E(G)$ donc $(x,y) \in E(\overline{G})$ et ces sommets sont connectés dans $\overline{G}$. Si x, y sont dans une même composante connexe de G, soit z un sommet appartenant à une autre composante connexe de G (un tel sommet existe puisque G est non connexe). Alors, par le même raisonnement que précédemment, (x,z) et (y,z) sont deux arêtes de $\overline{G}$, donc x et y sont connectés dans $\overline{G}$ par le chemin xzy.

(*c*) Cette assertion est fausse : le graphe $G = (\{1,2,3,4\}, \{\{1,2\},\{2,3\},\{3,4\}\})$ et son complémentaire sont tous les deux connexes.

Correction exercice 4.

(a) Désignons par G_1, G_2, G_3, G_4 les quatre graphes représentés de gauche à droite. Il est facile de voir que G_1, G_2 et G_4 ont tous pour complémentaire une union de deux triangles. Ces trois graphes sont donc isomorphes, en vertu du (a) de l'exercice 3. Par contre G_3 contient un triangle, ce qui n'est pas le cas de G_1. Donc ces G_3 n'est pas isomorphe à G_1 (ni à G_2 ou G_4).

(b) Dans un graphe non orienté à n sommets, il y a $C_n^2 = \frac{n!}{2!(n-2)!}$ paires de sommets. Chacune de ces paires peut constituer ou ne pas constituer une arête du graphe. Ce qui donne $2^{C_n^2}$ graphes possibles sur un ensemble fixé de n sommets. Pour $n = 4$ on obtient $2^6 = 64$ graphes possibles. Ces graphes se répartissent dans les 11 classes d'isomorphismes suivantes :



Correction exercice 5. Récurrence sur la longueur k du chemin : L'hypothèse est claire pour $k = 1$: tout chemin de longueur 1 est élémentaire. Supposons l'hypothèse de récurrence vérifiée par tout chemin de longueur inférieure à un k fixé. Soit $p = x_0a_1x_1 \dots a_{k+1}x_{k+1}$ un chemin de longueur $k+1$. Si p n'est pas élémentaire, soit i l'indice du premier sommet interne de p qui admet plusieurs occurences dans p. Soit j le plus petit indice tel que $x_j = x_i$. Alors $p' = x_0a_1x_1 \dots a_ix_ia_{j+1}x_{j+1} \dots a_{k+1}x_{k+1}$ est un chemin entre x_0 et x_{k+1} de longueur inférieure à k. Par hypothèse de récurrence, il existe un chemin élémentaire entre x_0 et x_{k+1}.

Correction exercice 6. $\Rightarrow$: Soient (U,V) la bipartition de $V(G)$, $u \in U$, $v \in V$. Soit p : $x_0x_1 \dots x_{k-1}x_k$ un chemin de $u = x_0$ à $v = x_k$ (dont l'existence est garantie par la connexité de G). On note i le plus grand indice des sommets de p appartenant à U. Alors $i < k$ (puisque $x_k \in V$ et $V \cap U = \varnothing$) et $x_{i+1} \notin U$ (par maximalité de i). Donc $x_{i+1} \in V$ et (x_i, x_{i+1}) est l'arête cherchée.

$\Leftarrow$: Soit U une composante connexe de G. Si U est unique, le graphe est connexe. Sinon, $(U, \neg U)$ constitue une bipartition non triviale de $V(G)$ et donc il existe une arête qui traverse $(U, \neg U)$: une contradiction.

Correction exercice 7. Faux : prendre un graphe constitué de deux cycles disjoints. Montrons que la bonne assertion est : *si chaque sommet d'un graphe non orienté G est de degré 2 alors G est une union de cycles élémentaires.* Il revient au même de prouver que tout graphe connexe dont tous les sommets sont de degré 2 est un cycle élémentaire. Soit donc G un tel graphe. Par le Lemme 2, G contient un cycle et donc, un cycle élémentaire p (Lemme 4). Ce cycle couvre nécessairement tous les sommets de G : sinon, il existerait une arête a entre $V(p)$ et $V(G) - V(p)$ (par connexité de G et grâce à l'exercice 6), et l'extrémité x de a qui est sur p serait adjacente à 3 sommets (ses 2 voisins dans p et l'autre extrémité de a), ce qui contredirait l'hypothèse $\delta(x) = 2$. Enfin, G ne contient aucune arête autre que celles intervenant dans p, toujours grâce

à l'hypothèse sur les degrés. Par conséquent, $G = p$. (Notons qu'on a même : *tout sommet de G est de degré 2* **ssi** *G est une union de cycles élémentaires*.)

Correction exercice 8. Commençons par introduire quelques définitions : si $p : x \rightsquigarrow y$ et $q : y \rightsquigarrow z$ sont deux chemins partageant une même extrémité y, on note pq le chemin $x \overset{p}{\rightsquigarrow} y \overset{q}{\rightsquigarrow} z$. Un chemin p est un **sous-chemin** d'un chemin q s'il existe deux chemins p_1, p_2 tels que $q = p_1 p p_2$. Un chemin est maximal s'il n'est sous-chemin propre d'aucun chemin. Si p est un chemin *élémentaire* et si x, y sont deux sommets apparaissant dans cet ordre dans p, on note $p[x,y]$ l'unique sous-chemin de p d'extrémités x et y.

Considérons l'extrémité u d'un chemin élémentaire maximal p_0 de G. Tout sommet v adjacent à u est sur p_0 (sinon on pourrait prolonger p_0). Soient $x, y \in V(G)$ et $p : x \rightsquigarrow y$ un chemin élémentaire de x à y dans G. Si p ne contient pas u, alors p est aussi un chemin élémentaire de x à y dans $G-u$. Si p contient u, notons α et β le prédécesseur et le successeur de u dans p. D'après ce qui précède, α et β sont sur p_0. De plus, $u \notin p_0[\alpha,\beta]$, puisque u est une extrémité de p_0. Donc le chemin obtenu à partir de p en remplaçant le sous-chemin $p[\alpha,\beta] = \alpha u \beta$ par le chemin $p_0[\alpha,\beta]$ est un chemin de x à y qui ne contient pas u. Autrement dit, c'est un chemin de x à y dans $G-u$. On a ainsi montré que deux sommets distincts de u connectés dans G sont aussi connectés dans $G-u$. Comme G est connexe, on en déduit que $G-u$ est connexe : u n'est pas un point d'articulation de G. Par symmétrie, il en va de même pour l'autre extrémité de p_0, ce qui finit de prouver le résultat annoncé.

Correction exercice 9. Il est clair que $\mathscr{C}$ est inclu dans la classe des graphes connexes : tout sommet isolé est connexe et, si G est connexe, les graphes obtenus à partir de G par application de la règle de construction décrite en 2 sont aussi connexes. Montrons qu'inversement, tout graphe connexe G est dans $\mathscr{C}$: c'est clair si G est un sommet isolé. Supposons le résultat vérifié pour tout graphe à au plus n sommets, pour un entier n fixé. Soit G un graphe connexe à $n+1$ sommets. D'après l'exercice 8, il existe un sommet u de G qui n'est pas point d'articulation. Par conséquent, $G-u$ est un graphe connexe à n sommets et donc, par hypothèse de récurrence, $G-u \in \mathscr{C}$. Comme G est obtenu à partir de $G-u$ par adjonction du sommet u et des arêtes de $Incid(u)$, G est lui-même dans $\mathscr{C}$.

Application : si G est connexe, alors $|E(G)| \geq |V(G)| - 1$. Le résultat est clair pour les sommets isolés. S'il est vrai pour un graphe connexe fixé G, il l'est pour tout graphe H obtenu à partir de G par application de la règle 2 puisque, selon cette règle, $|V(H)| = |V(G)| + 1$ et $|E(H)| \geq |E(G)| + 1$.

Correction exercice 10. L'implication $\Rightarrow$ est claire : tout chemin de longueur impaire dans le graphe (X,Y)-biparti a nécessairement une extrémité dans X et l'autre dans Y. Ça ne peut donc pas être un cycle. Réciproquement, supposons que G ne contienne aucun cycle de longueur impaire et montrons que G est biparti. Notons tout d'abord qu'un graphe est biparti ssi chacune de ses composantes connexes non triviales est biparti. Soit donc H une composante connexe non triviale de G et u un sommet de H. Pour tout $x \in V(H)$, on note $d(x)$ la longueur d'un plus court chemin entre x et u. (Comme H est connexe, d est bien définie sur tout $x \in V(H)$.) Notons alors

$X = \{x \in V(H) : d(x) \text{ est pair}\}$ et $Y = \{x \in V(H) : d(x) \text{ est impair}\}$. Soient $a = \{x,y\} \in E(H)$ et p (resp. q) un plus court chemin de x (resp. y) à u. Alors $x \overset{p}{\rightsquigarrow} u \overset{q^{-1}}{\rightsquigarrow} y \overset{a}{\to} x$ est un cycle de longueur $|p| + |q| + 1 = d(x) + d(y) + 1$. Ce cycle étant de longueur paire (par hypothèse), $d(x)$ et $d(y)$ sont de parités opposées. Autrement dit, les extrémités de a ne sont pas simultanément dans X ou dans Y : H est (X,Y)-biparti.

Correction exercice 11.

1ère méthode. Soient G un graphe connexe L'ensemble des sous-graphes couvrant et connexes de G est non vide : il contient G. Soit T un sous-graphe couvrant connexe de G, minimal pour le nombre d'arêtes. Pour toute $a \in E(T)$, $T - a$ est un sous-graphe couvrant de G comportant moins d'arête que T. Il est donc non connexe, par hypothèse de minimalité sur T. Ainsi, la suppression d'un arête quelconque du graphe connexe T le déconnecte : T est un arbre (Théorème 7-(**??**)) et donc un arbre couvrant de G.

2ème méthode. Soient G un graphe connexe. L'ensemble des sous-graphes couvrant et acycliques de G est non vide : il contient $(V(G), \varnothing)$. Soit T un sous-graphe couvrant acyclique maximal de G. Montrons par l'absurde que T est connexe : si tel n'est pas le cas, T comporte une composante connexe H telle que $V(H) \neq V(G)$ et par connexité de G, il existe une arête $a = \{x,y\}$ de G qui joint $V(H)$ et $V(G) - V(H)$ (Exercice 6). Or $T + a$ est acyclique, sinon il existerait un cycle élémentaire de $T + a$ contenant a (puisque T est acyclique) : $x \overset{a}{\to} y \overset{p}{\rightsquigarrow} x$, et donc un chemin p entre x et y dans T. Par conséquent, $T + a$ est un sous-graphe acyclique de G qui contient T :contradiction avec l'hypothèse de maximalité de T. Finalement, T est un sous-graphe couvrant acyclique et connexe de G : c'est un arbre couvrant de G.

3ème méthode. Méthode algorithmique. Tant qu'il existe des arêtes a sur un cycle de G, choisir une telle a, faire $G = G - a$. Le graphe obtenu est tjs couvrant, connexec et contient au moins une arête "cyclique" de moins. Le graphe obtenu à la sortie de la boucle TQ est l'arbre cherché.

Correction exercice 12. Il y a $n(n-1)$ couples de sommets distincts et donc $2^{n(n-1)}$ graphes orientés possibles. Il y a $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ paires de sommets et donc $2^{\frac{n(n-1)}{2}}$ graphes non orientés possibles.

Correction exercice 13. Le problème posé revient à calculer le nombre d'arbres possibles sur l'ensemble à n sommets $V = \{1,\ldots,n\}$. Nous allons montrer que les arbres sur V sont en bijection avec les $(n-2)$-uplets d'éléments de V. A tout arbre T sur V on associe les deux suites de sommets $(f_i)_{0<i\leq n-2}$ et $(s_i)_{0<i\leq n-2}$ définies comme suit :

- f_1 est la plus petite feuille de T, s_1 est le voisin de f_1 dans T, et $\forall i < n-2$:
- f_{i+1} est la plus petite feuille de $T - \{f_1,\ldots,f_i\}$;
- s_{i+1} est le voisin de f_i dans $T - \{f_1,\ldots,f_i\}$.

La Figure 1, donne l'exemple d'un arbre sur $\{0,\dots,8\}$ et des suites associées.

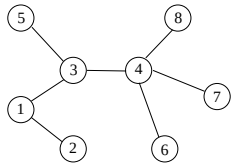


FIGURE 1 – $(f_i) = (2,1,5,3,6,7)$; $(s_i) = (1,3,3,4,4,4)$

On s'intéresse en fait uniquement au $(n-2)$-uplet $U = (s_1,\dots,s_{n-2})$ qui identifie de manière unique l'arbre T. En effet, à tout $(n-2)$-uplet d'éléments de V on peut associer l'arbre T sur V construit de la manière suivante : on part du graphe on repère le plus petit $i_1 \in V-U$. Alors l'arête (i_1,s_1) est dans T. On retire i_1 de V et $s-1$ de U, et on réitère l'opération avec les nouveaux V et U. Lorsqu'il ne reste plus d'élément dans U, on termine en ajoutant à T l'arête constituée des deux derniers éléments de V.

Ces deux constructions prouvent l'équipotence de l'ensemble des arbres et de l'ensemble des $(n-2)$-uplets sur $\{1,\dots,n\}$. Comme ce dernier ensemble est de cardinal n^{n-2}, on a le résultat cherché.

Correction exercice 14.

1 L'inégalité $\omega(G) \geq \omega(G+I)$ est claire : l'ajoût d'arêtes à G ne peut pas faire croître le nombre de composantes connexes. Dans le même temps, une arête ajoutée à un graphe relie au plus deux composantes connexes de ce graphe et donc réduit d'au plus un sa connectivité. Par conséquent, si $I = \{e_1,\dots,e_m\}$, la connectivité de $G+I = (\dots((G+e_1)+e_2)+\dots+e_m)$ est minorée par $\omega(G)-m$.

2 Les arguments sont similaires à ceux de la question précédente : le retrait d'arêtes ne peut pas faire décroître la connectivité, d'où l'inégalité $\omega(G) \leq \omega(G-I)$. De plus, le retrait d'une arête provoque la scission d'au plus une composante connexe. La majoration $\omega(G-I) \leq \omega(G)+|I|$ s'en déduit facilement.

3 Notons $I = \{e_1,\dots,e_m\}$. Le fait que $\omega(G-e) > \omega(G)$ pour tout $e \in I$ signifie qu'aucune de ces arête n'est sur un cycle de G. Puisque le retrait d'arêtes diminue le nombre de cycles, on sait a fortiori que $\omega(G-e_1) < \omega((G-e_1)-e_2) < \omega(((G-e_1)-e_2)-e_3) < \dots$ Par ailleurs, chaque inégalité de la forme $\omega(G-e) > \omega(G)$ équivaut à $\omega(G-e) = \omega(G)+1$, par la question 2. Il s'ensuit que $\omega(G-I) = \omega((\dots((G-e_1)-e_2)-e_3)-\dots-e_m) = \omega(G)+m$.

4 Le graphe $G-E$ est sans arête et donc, clairement, $\omega(G-E) = |V|$. Par ailleurs, par 2, $\omega(G-E) \leq \omega(G)+|E|$. D'où finalement : $|V| \leq \omega(G)+|E|$.

5 $\Rightarrow$: si G est acyclique, alors aucune arête de G n'est sur un cycle et donc, la suppression de toute $e \in E$ incrémente le nombre de composantes connexes de G. Autrement dit : $\omega(G-e) > \omega(G)$ pour tout $e \in E$. Il s'ensuit, par 3, que $\omega(G-E) = \omega(G)+|E|$ et la conclusion vient de $\omega(G-E) = |V|$.

$\Leftarrow$: supposons $\omega(G) = |V| - |E|$. Pour tout $e \in E$, $\omega(G-e) \geq |V| - (|E|-1)$ par 4. D'où $\omega(G-e) \geq |V| - |E| + 1 = \omega(G) + 1 > \omega(G)$ pour tout $e \in E$. Ceci signifie qu'aucun $e \in E$ n'est sur un cycle de G : G est acyclique.

Correction exercice 15.

- Si G est une arborescence de racine r, alors le graphe non orienté sous-jacent à G est connexe, puisque le chemin orienté de r à deux sommets quelconques x et y donne lieu à un chemin non orienté entre ces deux sommets. Il est aussi acyclique. Sinon, soit C un cycle. Une arborescence étant clairement sans circuit, il existe un sommet x de C comportant deux prédécesseurs u et v dans C. Soient p (resp. q) l'unique chemin orienté de r à u (resp. v). Alors $p.ux$ et $q.vx$ sont deux chemins orientés distincts de r à x : une contradiction.
- Soient G un arbre et $r \in V(G)$. On sait qu'il existe un chemin unique p_x de r à n'importe quel sommet x de G (voir Theorem 7, Item 6). On oriente alors tous les arcs de p_x de r vers x. Cette orientation peut être renouvellée sans conflit sur tout autre chemin p_y issu de r. En effet, si l'arc uv apparaît dans p_x, vu n'apparaît pas dans p_y. Si cela se produisait on aurait : $r \rightsquigarrow u \to v \rightsquigarrow x$ et $r \rightsquigarrow v \to u \rightsquigarrow y$, et ceci entraînerait l'existence de deux chemins distincts de r vers u.

Correction exercice 16. Récurrence sur k. Le résultat est immédiat pour $k = 1$ puisqu'un chemin de longueur k est un arc du graphe. Le calcul de M^k pour $k > 1$ donne :

$$M^k_{ij} = \sum_{\ell=1}^{n} M^{k-1}_{i\ell} M_{\ell j}.$$

Or tout chemin de longueur k entre x_i et x_j se décompose en un chemin de longueur $k-1$ entre x_i et un certain x_ℓ, suivi d'un arc reliant x_ℓ et x_j. Le résultat découle alors de l'hypothèse de récurrence selon laquelle $M^{k-1}_{i\ell}$ est le nombre de chemins de longueur $k-1$ joignant x_i à x_k.

Correction exercice 18. $\Rightarrow$ est clair. Pour la réciproque, considérons un cycle élémentaire C de G. S'il existe une arête $a \in E \setminus C$, alors $G - a$ est non acyclique : une contradiction. Donc $G = C$.

Master 1 d'Informatique — Recherche Opérationnelle
2010 / 2011

TD n° 2

Arbres couvrants minimaux

RAPPELS : Soit un graphe non orienté $G = (V, E)$. Si P est une partie non triviale de V (*i.e.*, $P \neq \varnothing$ et $P \neq V$), le couple $(P, V - P)$ est appelé **coupure** du graphe et noté $(P, \neg P)$. On dit d'un chemin $p : x \rightsquigarrow y$ de G qu'il **traverse** la coupure $(P, \neg P)$ si $x \in P$ et $y \notin P$. En particulier, une arête a traverse $(P, \neg P)$ si l'une de ses extrémités est dans P et l'autre non.

Exercice 1 (a) Montrer que tout chemin p traversant une coupure $(P, \neg P)$ contient une arête qui traverse cette même coupure.

(b) Montrer que si une arête est sur un cycle C et traverse une coupure $(P, \neg P)$, alors il existe une autre arête de C traversant cette même coupure.

Exercice 2 Dans un graphe pondéré, une arête a est dite minimale pour la traversée d'une coupure $(P, \neg P)$ si a est de poids minimal parmi les arêtes qui traversent $(P, \neg P)$. Soit a une arête d'un graphe non orienté connexe pondéré. Montrer l'équivalence des assertions suivantes :

(a) il existe un ACM T de G qui contient a ;

(b) il existe une coupure $(P, \neg P)$ de G telle que a est minimale pour la traversée de cette coupure.

Exercice 3 Soit G un graphe non orienté pondéré, connexe et non acyclique. Soient T un ACM de G et a une arête de G n'apparaissant pas dans T. On sait qu'alors $T + a$ contient un unique cycle C passant par a.

(a) Montrer que toute arête de C est de poids inférieur ou égal à celui de a.

(b) Montrer que si de plus a appartient à un arbre couvrant minimal T', alors C contient une arête $b \neq a$ de même poids que a.

Pour tout sous-graphe H de G, on note $\ell(H)$ la suite des poids des arêtes de H triés en ordre croissant.

(c) Montrer que pour deux ACM T et T' quelconques de G, $\ell(T) = \ell(T')$.

Exercice 4 Soit a une arête de poids minimal dans un graphe pondéré. Montrer que a appartient à un arbre couvrant minimal du graphe.

Exercice 5 Soit (G, c) un graphe non orienté connexe valué par une fonction injective. Prouver ou infirmer :

(a) Tout ACM de G contient l'arête de poids minimum ;

(b) Aucun ACM ne contient l'arête de poids maximum ;

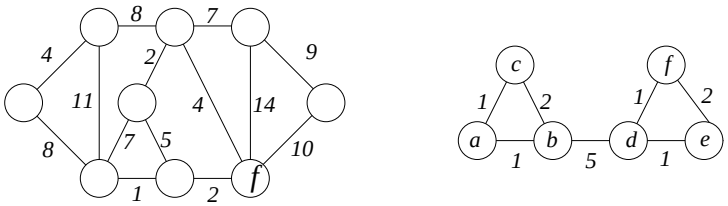
(c) Il n'y a qu'un ACM.

Exercice 6 Dans un graphe non orienté connexe valué (G,c), on construit un sous-graphe T de la manière suivante :

> On considère une partition $V_1 \sqcup V_2$ de $V(G)$. On note G_1, G_2 les sous-graphes de G induits, respectivement, par V_1 et V_2. Soit T_1 (resp. T_2) un ACM de G_1 (resp. G_2). On choisit une arête a de coût minimum parmi les arêtes joignant V_1 à V_2 et on appelle T le sous-graphe $T = (T_1 \cup T_2) + a$.

Le graphe T est-il un arbre couvrant minimal de G ?

Exercice 7 Faire tourner l'algorithme de `Kruskal` sur les graphes pondérés suivants :



Exercice 8 Montrer que la variante suivante de l'algorithme de `Kruskal` retourne un arbre couvrant minimal de G :

Algorithme 1: Variante de l'algorithme de `Kruskal`

Données : Un graphe pondéré connexe (G,c)
Résultat : Un ACM F de (G,c)

```
1 F = ∅ ;
2 numéroter les arêtes de G dans un ordre quelconque a_1 ... a_m ;
3 pour i = 1 à m faire
4 |   F = F + a_i ;
5 |   si F contient un cycle alors
6 |   |   choisir une arête b de poids maximal sur ce cycle ;
7 |   |   F = F − b ;
  |   fin
  fin
8 retourner F
```

Exercice 9 Pour A un ensemble d'arêtes de G, on note $m(A)$ le poids maximal d'une arête de A. Modifier l'algorithme de Prim pour déterminer un arbre couvrant A ayant le plus petit $m(A)$.

Exercice 10 L'algorithme de `Kruskal` peut retourner des ACM différents suivants le tri des arêtes par poids croissant à partir duquel il effectue son calcul. Montrer que pour tout ACM T de G, il existe un tri des arêtes par poids croissant à partir duquel `Kruskal` retourne T.

Exercice 11 Montrer que si H est un sous-graphe d'un ACM de G et si de plus a est une arête de G minimale pour la traversée d'une coupure qui contient H, alors $H + a$ est lui aussi un sous-graphe d'un ACM de G. En déduire une nouvelle preuve de la correction de l'algorithme de `Kruskal`.

Exercice 12 Soient $G = (V, E, w)$ un graphe non orienté connexe et pondéré. On considère l'algorithme suivant.

Données : Un graphe non orienté connexe pondéré $G = (V, E, w)$

1 $F = G$;
2 trier les arêtes par poids décroissants ;
3 **pour** *chaque arête a prise dans cet ordre* **faire**
4 | **si** $F - a$ *est connexe* **alors** $F = F - a$;
fin
5 **retourner** F

(a) Montrer que cet algorithme termine sur toute entrée.

(b) Montrer qu'il retourne un arbre couvrant.

(c) Cet arbre couvrant est-il toujours de poids minimal ?

Corrigé du TD n° 2

Arbres couvrants minimaux

Correction exercice 1.

(a) On considère le premier sommet u de p appartenant à $\neg P$. Alors l'arête qui précède u dans p fait l'affaire.

(b)

Correction exercice 2. $(a) \Rightarrow (b)$: le graphe $T-a$ admet deux composantes connexes P_1 et P_2. Montrons que le couple (P_1,P_2) est la coupure cherchée. Notons tout d'abord que a traverse (P_1,P_2) (clair). Par ailleurs, aucune arête de $T-a$ ne traverse (P_1,P_2) (sinon ces deux parties seraient connectées dans $T-a$). Donc s'il existe une arête $b=\{x,y\}$ distincte de a et traversant (P_1,P_2), $b \notin T$. Par conséquent, $T+b$ comporte un unique cycle de la forme $x \xrightarrow{b} y \overset{p}{\rightsquigarrow} x$, où p est un chemin dans T. Comme p traverse (P_1,P_2), il contient une arête qui traverse cette coupure (d'après la question précédente), et cette arête ne peut être que a. Ainsi, a est sur l'unique cycle de $T+b$ et par conséquent, le graphe $T'=(T+b)-a$ est un arbre couvrant de G. Il doit donc être de poids $\geq$ à celui de T, ce qui entraîne que b est de poids $\geq$ à celui de a : l'arête a est bien minimale pour la traversée de (P_1,P_2).

$(b) \Rightarrow (a)$: Soit T_0 un ACM de G. Si T_0 contient a, c'est terminé. Sinon, T_0+a contient un unique cycle de la forme $x \xrightarrow{a} y \overset{p}{\rightsquigarrow} x$, où p est un chemin de T_0. Comme p traverse la coupure $(P,\neg P)$, il contient une arête b traversant cette coupure. Par minimalité de a pour la traversée de $(P,\neg P)$, b est de poids $\geq$ à celui de a. Par conséquent, $T=(T_0+a)-b$ est un arbre couvrant de poids $\leq$ à celui de T_0 : c'est un ACM de G qui contient a.

Correction exercice 3.

(a) Soit $b \neq a$ appartenant à C. Alors $(T+a)-b$ est un arbre couvrant de (G,c). Par conséquent, $c((T+a)-b) \geq c(T)$. Autrement dit, $c(T)+c(a)-c(b) \geq c(T)$ et donc $c(b) \leq c(a)$.

(b) Si a appartient à un ACM, alors a est minimale pour la traversée d'une coupure $(P,\neg P)$. Mais alors, le sous-chemin de C obtenu en retirant l'arête a de C est un chemin dans T qui traverse la coupure $(P,\neg P)$. Ce chemin contient donc nécessairement une arête b de T qui traverse $(P,\neg P)$. Comme b est sur le cycle, $c(b) \leq c(a)$, d'après la question précédente. Comme b traverse $(P,\neg P)$, $c(a) \leq c(b)$, puisque a est minimale pour la traversée. Finalement, $c(b)=c(a)$.

(c) On note $a_1,a_2,\ldots,a_k$ la suite des arêtes de T'. Soit $(T_i)_{0\leq i\leq k}$ la suite de sous-graphes de G définie récursivement par : $T_0=T$ et pour tout $i<k$,

– si $a_{i+1} \in T_i$ alors $T_{i+1} = T_i$;
– sinon $T_{i+1} = (T_i + a_i) - b_i$, où b_i est une arête de même poids que a_i sur l'unique cycle de $T_i + a_i$. (Voir (b).)

Alors pour tout $i > 0$:

(1) T_i est un arbre couvrant qui contient $(a_1, \ldots, a_i)$;

(2) $\ell(T_i) = \ell(T_{i-1})$.

Par conséquent, $T_k = T'$ et $\ell(T') = \ell(T_0) = \ell(T)$.

Correction exercice 4.

1ère méthode. Il suffit de faire tourner l'algorithme de `Kruskal` à partir d'un tri des arêtes par poids croissant dans lequel a apparaît en première position (un tel tri existe puisque a est de poids minimal). Alors a sera sélectionnée lors du premier passage dans la boucle "pour" et placée dans le graphe F en construction. Le résultat se déduit alors de la preuve de la correction de cet algorithme, qui établit que le graphe F retourné est un ACM de G.

2ème méthode. Soit T un ACM de G. Si T contient a, c'est fini. Sinon, $T + a$ contient un unique cycle passant par a. Donc le graphe T' obtenu à partir de $T + a$ en retirant une arête $b \neq a$ de ce cycle est un arbre couvrant (connexe parce qu'on a enlevé une arête d'un cycle d'un graphe connexe, acyclique parce que ce cycle était unique, couvrant parce que $V(T') = V(T)$). De plus, $c(T') = c(T) + c(a) - c(b) \leq c(T)$ puisque $c(a) \leq c(b)$. Comme T est de poids minimal, T' est un ACM (qui contient a).

Correction exercice 5.

(a) est vraie. Soit T un ACM de G qui ne contient pas l'arête de poids minimum a. Alors $T + a$ contient un unique cycle qui passe par a et, comme dans l'Exercice 4, pour toute arête $b \neq a$ de ce cycle, $(T + a) - b$ est un arbre couvrant de G. De plus, $c((T + a) - b) = c(T) + c(a) - c(b)$. Comme a est de poids minimum et comme c est injective, on a $c(a) < c(b)$ et donc $c((T + a) - b) < c(T)$: T n'est donc pas minimal.

(b) est fausse : prendre un chemin de longueur 2 dont les arêtes sont valuées, respectivement, par 1 et 2.

(c) est vraie : Supposons que G admette deux ACM distincts, T et T'. Soit a_1, a_2,..., a_k (resp. $b_1, b_2, \ldots, b_k$) un tri des arêtes de T (resp. de T') par pondérations croissantes. Soit ℓ le plus petit des indices i tels que $a_i \notin E(T')$. Alors $T' + a_\ell$ comporte un cycle, et ce cycle contient au moins une arête qui n'est pas dans T (sinon ce serait un cycle dans T). Soit b une telle arête. Comme $b \in E(T') - E(T)$, $b \in \{b_i, i > \ell\}$. De plus $(T' + a_\ell) - b$ est un arbre couvrant de G. Enfin, $c((T' + a_\ell) - b) = c(T') + c(a_\ell) - c(b) < c(T')$: contradiction.

Correction exercice 6. Non. Contre exemple, un carré $x - y - z - t - x$ avec $c(xy) = c(zt) = 10$, $c(yz) = c(xt) = 1$ et $V_1 = \{x, y\}$.

Correction exercice 8. On remarque d'abord que l'algorithme termine bien sur toute entrée, puisque son unique boucle consiste en un parcours d'une liste finie d'arêtes. On note H le graphe engendré par toutes les arêtes de G envisagées à un instant donné de l'exécution de l'algorithme, que ces arêtes aient été retenues ou non dans F. En d'autres termes, H est le graphe vide avant le premier passage dans la boucle ; c'est le graphe engendré par $(a_1, \ldots, a_i)$ après i passages dans la boucle. Montrons que l'assertion suivante est un invariant de la boucle "pour" :

$$F \text{ est un sous-graphe acyclique couvrant de poids minimal de } H. \qquad (*)$$

Supposons que l'assertion $(*)$ soit vrai avant un passage dans la boucle et prouvons que tel est encore le cas après ce passage. Soit donc a l'arête ajoutée à F ligne 4.

Si $F+a$ est acyclique, les instructions 6 et 7 ne sont pas exécutées et après passage dans la boucle, F est remplacé par $F+a$, et H par $H+a$.

Sinon, $F+a$ est remplacé par $(F+a)-b$,

Correction exercice 12.

(a) L'algorithme termine sur toute entrée puisque la seule boucle qu'il contient consiste en une boucle **pour** qui parcourt une liste finie.

(b) Le graphe F, obtenu par suppression d'arêtes de G, est clairement un *sous-graphe couvrant* de G. De plus, initialisé à G, qui est connexe, et modifié dans le seul cas où cette modification de compromet pas la connexité, F reste *connexe* tout au long de l'algorithme. Enfin, lorsque l'algorithme termine, toute arête de F a été laissée dans F parce que sa suppression aurait déconnecté F. Donc le graphe F récupéré en fin d'algorithme est un sous-graphe couvrant connexe *minimal* de G, c'est à dire un arbre couvrant de G.

(c) Cet arbre couvrant est *minimal*. Pour le prouver, montrons que l'assertion suivante : $(*)$ « *F contient un ACM de G comme sous-graphe* », est un invariant de l'algorithme. Elle est évidemment vraie à l'initialisation $F = G$. Supposons-la vérifiée avant une exécution de la boucle **pour**. Soit a l'arête considérée lors de cette exécution. Si $F-a$ est non connexe, alors F n'est pas modifié et la validité de l'assertion $(*)$ est préservée. Sinon, F est remplacé par $F-a$. Soit T l'ACM contenu dans F. Si $a \notin T$, alors T est encore un sous-graphe de $F-a$ et $(*)$ est préservée. Sinon, $T-a$ comporte deux composantes connexes. On sait que l'hypothèse « $F-a$ connexe » signifie que a est sur un cycle de F. Ce cycle comporte nécessairement une arête b distincte de a qui connecte les deux composantes connexes de $T-a$. De ce fait, $T-a+b$ est un arbre couvrant de G. De plus, b n'a pas encore été considérée par l'algorithme au moment où s'exécute ce passage dans la boucle **pour** (sinon, puisqu'elle est sur un cycle de F, elle en aurait été retirée). Ceci garantit que $w(b) \leq w(a)$. Par conséquent, $w(T-a+b) \leq w(T)$ et $T-a+b$ est un ACM de G qui contient $F-a$.

Ainsi, l'assertion $(*)$ est vraie tout au long de l'algorithme. En particulier, le graphe F retourné ligne 5 contient un ACM de G. Comme F est lui-même un arbre couvrant, il s'identifie à cet ACM. C'est le résultat cherché.

Master 1 d'Informatique — Recherche Opérationnelle
2010 / 2011

TD n° 3

Algorithmes gloutons - Matroïdes

Exercice 1 *(Divers matroïdes).* Rappelons qu'une base d'un matroïde est un ensemble indépendant maximal pour l'inclusion. Par ailleurs, on appelle **circuit** d'un matroïde tout ensemble dépendant minimal. Autrement dit, C est un circuit de $M = (X, \mathscr{I})$ si $C \notin \mathscr{I}$ et $C - x \in \mathscr{I}$ pour tout $x \in C$. Pour chacun des exemples suivants, vous montrerez que $(X, \mathscr{I})$ est un matroïde et vous décrirez ses bases et ses circuits.

1. *Matroïde graphique $M(G)$:* Étant donné un GNO $G = (V, E)$, on pose $X = E$ et un ensemble $I \subseteq X$ est indépendant si le sous-graphe (V, I) de G est acyclique.
2. *Matroïde cographique $M^*(G)$:* Étant donné un GNO $G = (V, E)$ *connexe*, on pose $X = E$ et un ensemble $I \subseteq X$ est indépendant si le sous-graphe $(V, E \setminus I)$ de G est connexe.
3. *Matroïde uniforme $U_{k,n}$:* On pose $X = \{1, \ldots, n\}$ et un ensemble $I \subseteq X$ est indépendant si $|I| \leq k$.

Exercice 2 *(Min/Max graphique/cographique).* Relire les définitions des matroïdes graphique et cographique associés à un graphe G (les structures $M(G)$ et $M^*(G)$ de l'Exercice 1). Soit $G = (V, E, w)$ un graphe pondéré.

1. Montrer que I est une base de poids maximal de $M(G)$ ssi $E \setminus I$ est une base de poids minimal de $M^*(G)$.
2. En déduire un nouvel algorithme pour le calcul d'un arbre couvrant minimal.

Exercice 3 *(Kruskal et matroïde).* Montrer que Kruskal est un cas particulier de recherche d'indépendant maximal dans un matroïde.

Exercice 4 *(Calculer dans un matroïde).* On se donne un matroïde $M = (E, \mathscr{I})$ et un algorithme qui détermine si un sous-ensemble de E est indépendant en temps polynomial en la taille du sous-ensemble.

1. Le rang d'un sous-ensemble X de E est le cardinal maximum d'un indépendant inclus dans X. Donner un algorithme qui calcule le rang d'un sous-ensemble X en temps polynomial en la taille de X,
2. Donner un algorithme qui construit une base contenant un ensemble indépendant X donné, en temps polynomial en la taille de E.

Exercice 5 *(Un problème d'ordonnancement)*. De nombreux problèmes rencontrés dans l'affectation de tâches sur un processeur s'apparentent à ceux concernant l'organisation du travail dans les ateliers. En voici un exemple : on suppose que l'on doit réaliser des taches $T_1, T_2, \ldots, T_n$ sur une seule machine. Chaque tâche T_i a une durée d_i et une priorité p_i. Une réalisation de ces tâches est donnéee par une permutation $T_{i_1}, T_{i_2}, \ldots, T_{i_n}$ de celles-ci, représentant l'ordre dans lequel elles sont effectuées sur la machine. La date de fin F_i d'une tâche T_i est donnée par la somme des durées des tâches qui la précèdent dans la permutation augmentée de sa durée propre d_i. On mesure la pénalité d'une réalisation par la quantité $\sum_{i=1}^{n} p_i F_i$ qui devra être rendue minimum, ainsi les tâches de priorités plus grandes devraient être réalisées avant celles de priorités plus petites.

1. On suppose qu'il y a 4 tâches ayant pour durées 3, 5, 7, 4, et de priorités 6, 11, 9, 5. Laquelle de ces deux réalisations est-elle la meilleure : T_1, T_4, T_2, T_3 ou T_4, T_1, T_3, T_2 ?
2. Soient deux tâches T_i et T_j qui se suivent dans une réalisation et telles que $\frac{d_i}{p_i} < \frac{d_j}{p_j}$. Laquelle de ces deux tâches doit on mettre avant l'autre pour minimiser la pénalité ?
3. En déduire un algorithme glouton permettant d'obtenir la réalisation de plus faible pénalité.

Exercice 6 *(Processeur séquentiel)*. On considère un ensemble de tâches T_i, $1 \leq i \leq n$, ayant toutes une durée unitaire. Pour chaque tâche sont fixés un délai d_i et une pénalité p_i qui est à acquitter quand la tâche n'est pas remplie avant le temps d_i. On veut trouver un ordre d'exécution des tâches sur une machine unique qui minimise la somme des pénalités. Soit $\mathscr{F}$ la famille des sous-ensembles F de $\{T_1, \ldots, T_n\}$ telle que $F \in \mathscr{F}$ ssi les tâches de F peuvent être toutes réalisées à temps. On appellera ces ensembles réalisables. Montrer que $\mathscr{F}$ est une famille d'ensembles formant un matroïde. (On pourra chercher, étant donnés deux ensembles réalisables X et Y , chacun ordonné par délai croissant, et tels que $|X| < |Y|$, quel élément de Y on peut ajouter à X tout en conservant la réalisabilité.) On conclura donc qu'il suffit de classer les tâches par ordre de pénalité décroissante pour obtenir la réalisation optimale.

Exercice 7 *(Des limites de la gloutonnerie)*. Cet exercice a pour but de montrer que de nombreux problèmes ne sont pas résolubles par un algorithme glouton. Considérons par exemple celui dit du sac à dos. On doit emporter en voyage un certain nombre d'objets parmi n, l'objet i a un poids p_i et un intérêt a_i. On est limité par le poids total P des objets à emporter et on cherche donc un sous-ensemble d'objets dont la somme des intérêts est maximale parmi ceux dont la somme des poids est inférieure ou égale à P.

1. On applique l'algorithme consistant à classer les objets par intérêt décroissant et à considérer les objets itérativement dans cette liste ; on emporte ceux dont le poids ne fait pas dépasser le poids total P acceptable. Montrer en donnant un exemple que cet algorithme ne donne pas toujours l'optimum.
2. On applique ensuite l'algorithme consistant à classer les objets par poids croissant et à procéder ensuite comme pour l'algorithme précédent. Donner un nouvel exemple montrant que cet algorithme ne donne pas l'optimum.
3. On applique enfin l'algorithme consistant à classer les objets suivant le rapport (intérêt/poids) et à poursuivre comme plus haut. Montrer à nouveau que cet algorithme ne donne pas l'optimum.

Exercice 8 *(Axiomatisation des bases d'un matroïde).* Étant donnés un ensemble fini X et un ensemble de parties de X, $\mathscr{B} \subseteq \mathscr{P}(X)$, on considère les propriétés suivantes :

(B_1) $\varnothing \notin \mathscr{B}$.

(B_2) si $X, Y \in \mathscr{B}$ et $x \in X \setminus Y$, alors $\exists y \in Y \setminus X$ t.q. $X - x + y \in \mathscr{B}$.

Montrer que $\mathscr{B}$ satisfait (B_1) et $(B_2) \Leftrightarrow \mathscr{B}$ est l'ensemble des bases d'un matroïde.

Exercice 9 *(Un nouveau matroïde).* On considère un ensemble fini E et une partition $E_1, \ldots, E_k$ de E en sous-ensembles disjoints non vides. On définit la structure $(E, \mathscr{I})$ par la condition :

$$\mathscr{I} = \{I \subseteq E : |I \cap E_i| \leq 1 \text{ pour } i = 1, \ldots, k\}.$$

Montrer que $(E, \mathscr{I})$ est un matroïde.

Exercice 10 Soient $G = (V, E)$ un graphe non orienté. Un ensemble de sommets $C \subseteq V$ est une *couverture des sommets* (*vertex-cover* en anglais) de G si toute arête de G est incidente à un sommet $x \in C$. On considère le problème de minimisation suivant :

Problème :	VERTEX-COVER
Input :	Un graphe non orienté $G = (V, E)$
Output :	Une couverture des sommets minimale de G

Une approche gloutonne pour le traitement de ce problème consiste à construire C itérativement (à partir de $\varnothing$) en ajoutant à chaque étape un sommet v de degré aussi grand que possible, de façon à maximiser le nombre d'arête "couvertes" par v. On espère ainsi minimiser le nombre de sommets à ajouter pour couvrir l'ensemble des arêtes. Ceci donne lieu à l'algorithme suivant :

Données : Un graphe non orienté G
1 $C = \varnothing$;
2 **repeat**
3 | choisir dans G un sommet v de degré maximal ;
4 | $C = C \cup \{v\}$;
5 | $G = G - v$
until $E(G) = \varnothing$;
7 **retourner** C

(a) Montrer que cet algorithme termine sur toute entrée.

(b) Montrer qu'il retourne bien une couverture des sommets.

(c) Cette couverture des sommets est-elle toujours minimale ?

Corrigé du TD n° 3

Matroïdes

Correction exercice 1. Pour chaque exemple $(X, \mathscr{I})$ envisagé on montre que :

$(a)\ \varnothing \in \mathscr{I}$; $(b)\ I \subseteq J \in \mathscr{I} \Rightarrow I \in \mathscr{I}$; $(c)\ (I,J \in \mathscr{I} \wedge |I| < |J|) \Rightarrow \exists x \in J \setminus I : I \cup \{x\} \in \mathscr{I}$.

Puis on décrit : (d) les bases de M ; (e) les circuits de M.

Matroïde graphique $M(G)$: $(a)\ \varnothing \in \mathscr{I}$ parce qu'un graphe sans arête est acyclique.

$(b)\ I \subseteq J$ et $J \in \mathscr{I}$ signifie que I est un sous-ensemble de l'ensemble d'arêtes acyclique J, donc I est lui même acyclique (donc dans $\mathscr{I}$).

(c) Si $I,J \in \mathscr{I}$ et $|I| < |J|$: Soit $\mathscr{C}$ l'ensemble des composantes connexes de (V,I). Pour chaque $C \in \mathscr{C}$, on note I_C (resp. J_C) l'ensemble des arêtes de I (resp. de J) dont les deux extrémités sont dans C. Alors chaque J_C a au plus $|C|-1 = |I_C|$ éléments (sinon (V,J) comporterait un cycle. Donc $\sum_{C\in\mathscr{C}} |J_C| \leq \sum_{C\in\mathscr{C}} |I_C| = |I| < |J|$. Par conséquent, il existe une arête $e \in J$ dont les deux extrémités ne sont pas dans une même composante I-connexe. Et l'on a $(V, I \cup \{e\})$ acyclique, *i.e.* $I \cup \{e\} \in \mathscr{I}$.

(d) On sait que les ensemble d'arêtes acycliques maximaux - *i.e.* les bases de $M(G)$ - sont les arbres couvrants de G.

(e) Un circuit de $M(G)$ est un ensemble d'arêtes non acyclique mais que la suppression d'une arête quelconque rend acyclique. Les circuit de $M(G)$ sont donc exactement les cycles de G.

Matroïde cographique $M^*(G)$: Par définition de $M^*(G)$, un ensemble $I \subseteq E$ est indépendant ssi $G-I$ est connexe.

$(a)\ \varnothing \in \mathscr{I}$, puisque $G - \varnothing = G$ est connexe.

(b) Si $I \subseteq J$ et $J \in \mathscr{I}$, alors $G-J$ est connexe et est un sous-graphe de $G-I$. Par conséquent, $G-I$ est aussi connexe, *i.e.* $I \in \mathscr{I}$.

(c) Supposons que $I,J \in \mathscr{I}$ et $|I| < |J|$. Ceci signifie que $G-I$ et $G-J$ sont tous deux connexes. Nous voudrions prouver qu'il existe $e \in J \setminus I$ tel que $I \cup \{e\} \in \mathscr{I}$, c'est à dire tel que $(G-I)-e$ soit connexe. Supposons, en vue d'une contradiction, qu'un tel e n'existe pas. Autrement dit, pour tout $e \in J \setminus I$, $\omega((G-I)-e) > 1 = \omega(G-I)$. Cela entraîne, grâce à la question 3 de l'exercice 14 :

$$\omega((G-I)-J\setminus I) = \omega(G-I) + |J\setminus I|. \tag{1}$$

Par ailleurs, le graphe $G-J$ peut être obtenu à partir de $(G-I)-J\setminus I$ en rajoutant toutes les arêtes de I qui ne sont pas dans J. En d'autres termes : $G-J = ((G-I)-J\setminus I)+I\setminus J$. Mais alors

$$\omega(G-J) = \omega(((G-I)-J\setminus I)+I\setminus J) \geq \omega((G-I)-J\setminus I) - |I\setminus J|$$

et donc, par (1) :

$$\omega(G-J) \geq \omega(G-I) + |J \setminus I| - |I \setminus J|.$$

Or, l'inégalité $|J| > |I|$ posée en hypothèse équivaut à $|J \setminus I| > |I \setminus J|$. De plus, $\omega(G-I) = 1$ puisque $G-I$ est connexe. On obtient finalement :

$$\omega(G-J) > 1,$$

ce qui contredit l'hypothèse « $G-J$ connexe ».

(d) Un ensemble d'arêtes I est une base de $M^*(G)$ si $G-I$ est connexe et si $G-I-x$ est non connexe pour tout $x \in E \setminus I$. Autrement dit, I est une base si $G-I$ est un sous-graphe connexe minimal de G, donc si $G-I$ est un arbre couvrant de G. Les bases de $M^*(G)$ sont donc les complémentaires d'arbres couvrants de G.

(e) I est un circuit de $M^*(G)$ ssi $G-I$ est non connexe mais pour tout $J \subsetneq I$, $G-J$ est connexe. Autrement dit, les circuits de G sont les ensembles minimaux d'arêtes dont la suppression déconnecte G. C'est ce qu'on appelle les **cocycles** de G.

Matroïde uniforme $U_{k,n}$: (a) $|\varnothing| \leq k$.
(b) $I \subseteq J$ et $|J| \leq k$ entraîne $|I| \leq k$.
(c) $|I|, |J| \leq k$ et $|I| < |J|$ entraîne $|I| < k$ et donc, pour tout $x \in J \setminus I$: $|I \cup \{x\}| \leq k$.
(d) Clairement, les bases de $U_{k,n}$ sont les parties de cardinal k de $\{1, \ldots, n\}$.
(e) Les circuits de $U_{k,n}$ sont les parties de cardinal $k+1$ de $\{1, \ldots, n\}$.

Correction exercice 3. Il suffit de constater qu'un arbre couvrant minimal pour la pondération w est un arbre couvrant maximal (donc un indépendant maximal du matroïde graphique) pour la pondération $C - w$, où C est une constante suffisament grande ($C = w(G)$ par exemple). On vérifie alors facilement que `Kruskal` est la transcription exacte de l'algorithme glouton pour les indépendant maximaux.

Correction exercice 4.

1. L'ensemble des indépendants inclus dans X forme un matroïde (le vérifier). Pour trouver un indépendants maximum il suffit donc d'appliquer l'algorithme glouton en donnant 1 comme valuation à chaque élément.
2. Si X n'est pas un indépendant il n'existe pas de base le contenant. Sinon l'ensemble des parties Y de $M \setminus X$ telles que $X \cup Y$ soit un indépendant forme un matroïde (le vérifier). De nouveau il suffit donc d'appliquer l'algorithme glouton.

Correction exercice 5.

1. La suite T_1, T_4, T_2, T_3 donne $F_1 = 3$, $F_4 = 7$, $F_2 = 12$, $F_3 = 19$, soit une pénalité totale de $3.6 + 12.11 + 9.19 + 5.7 = 356$. La suite T_4, T_1, T_3, T_2 donne $F_1 = 7$, $F_2 = 19$, $F_3 = 14$, $F_4 = 4$, soit une pénalité de $7.6 + 19.11 + 9.14 + 5.4 = 397$. Le premier choix est donc le meilleur.

2. Si ces deux tâches se suivent, soit T le temps écoulé avant la première des deux. Alors si i précède j, la pénalité sera $(T+d_i)p_i+(T+d_i+d_j)p_j$; dans le cas contraire la pénalité sera $(T+d_j)p_j+(T+d_j+d_i)p_i$. La différence de ces deux valeurs vaut $d_ip_j-d_jp_i$. Il s'ensuit que i doit précéder j si $d_ip_j-d_jp_i<0$, soit encore $d_i/p_i<d_j/p_j$.
3. Supposons que dans l'affectation optimale, les tâches ne soient pas effectuées par d/p croissant. Il existe alors deux tâches $t=(d,p)$ et $t'=(d',p')$ avec $d/p>d'/p'$ telle que t est effectuée juste avant t'. La question précédente montre alors qu'inverser t et t' améliore l'affectation, contradiction. Il suffit donc d'utiliser l'algorithme glouton en ordonnant les tâches par ordre d/p croissant. Sur l'exemple de l'énoncé on obtient $T2,T1,T3,T4$ pour une pénalité de 333.

Correction exercice 6. On vérifie les axiomes de matroïde. La partie $\mathscr{F}$ est non vide (elle contient $\varnothing$), et manifestement si $F\in\mathscr{F}$, alors pour tout $G\subset F$, $G\in\mathscr{F}$. Vérifions l'axiome d'extension. Soit un ensemble de tâches $X=\{T_{i_1},T_{i_1},\ldots,T_{i_k}\}$ réalisable dans cet ordre et un autre ensemble $Y=\{T_{j_1},T_{i_2},\ldots,T_{j_\ell}\}$ réalisable dans cet ordre avec $l>k$. Soit T_{j_p} la tâche la plus tardive de Y qui n'est pas dans X. L'ensemble de tâche obtenu en ajoutant T_{j_p} à X est réalisable, dans l'ordre X", T_{j_p}, Y' où $Y'=T_{j_p+1},\ldots,T_{j_\ell}$ est l'ensemble des tâches qui se trouvent après T_{j_p} dans Y (et qui sont aussi des tâches de X), et où X" est la suite des tâches restantes de X. Pour vérifier que cet ordre est bien réalisable, on remarque que :

- les tâches de X'' sont effectuées en un temps qui est inférieur ou égal à celui où elles l'étaient dans X,
- les tâches T_{j_p} et dans Y' sont effectuées en un temps qui est inférieur ou égal à celui où elles l'étaient dans Y.

On a donc bien prouvé que $\mathscr{F}$ définit un matroïde. Il suffit donc d'appliquer l'algorithme glouton pour les valuations auxquelles on s'intéresse, ici les pénalités : la fonction d'intérêt à maximiser est la somme des pénalités des tâches effectuées.

Correction exercice 7. Il suffit de considérer la suite d'objets (poids, intérêt) suivante : $(2,3)$, $(1,2)$, $(1,2)$. Dans un sac de capacité 2, l'algorithme glouton reposant sur l'intérêt choisira d'emporter l'objet 1, alors qu'il est préférable d'emporter les objets 2 et 3. Dans un sac de capacité 3, l'algorithme glouton basé sur le rapport intérêt/poids choisira d'emporter les objets 2 et 3, tandis que 2 et 1 (ou 3 et 1) serait préférable ; il en est de même pour l'algorithme glouton basé sur le poids.

Correction exercice 8. $\boxed{\Leftarrow}$ Soit $\mathscr{B}$ l'ensemble des bases d'un matroïde $(X,\mathscr{I})$, c'est-à-dire, l'ensemble des indépendant maximaux de $(X,\mathscr{I})$. Si X est non trivial (*i.e.*, contient un indépendant non vide), alors aucune base n'est vide, donc $\mathscr{B}$ satisfait (B_1).

Par ailleurs, si X,Y sont deux bases et si $x\in X\setminus Y$, alors $X-x\in\mathscr{I}$ et $|X-x|<|Y|$ (puisque $|X|=|Y|$), donc $\exists y\in Y\setminus X$ t.q. $X-x+y\in\mathscr{I}$ (axiome d'échange des indépendants). Mais puisque $|X-x+y|=|X|$, on a de plus $X-x+y\in\mathscr{B}$.

$\boxed{\Rightarrow}$ Si $\mathscr{B}\subseteq\mathscr{P}(X)$ satisfait (B_1) et (B_2), l'ensemble $\mathscr{I}\subseteq\mathscr{P}(X)$ défini par : « $I\in\mathscr{I}$ ssi $\exists B\in$

$\mathscr{B}$ tel que $I \subseteq B$ » est un ensemble d'indépendants (autrement dit, $(X, \mathscr{I})$ est un matroïde). En effet :

1. $\varnothing \notin \mathscr{B}$: clair.
2. $I \subseteq J \subseteq B$ avec $B \in \mathscr{B}$ entraîne $I \subseteq B$ et donc $I \in \mathscr{I}$.
3. Soient $I, J \in \mathscr{I}$ tels que $|I| < |J|$. Soient de plus $X, Y \in \mathscr{B}$ tels que $I \subseteq X \in \mathscr{B}$ et $J \subseteq Y \in \mathscr{B}$. Considérons un élément a de $J \setminus I$. (En particulier, $a \in Y$.)
   Si $a \in X$ alors $I + a \subseteq X$ et donc $I + a \in \mathscr{I}$.
   Sinon, $a \in Y \setminus X$ donc, par (B_2), $\exists b \in X \setminus Y$ tel que $X + a - b \in \mathscr{B}$. Or, $I + a \subseteq X + a - b$ puisque $b \notin X$ et donc $b \notin I$. Il s'ensuit : $I + a \in \mathscr{I}$.

Correction exercice 9.

(a) $\varnothing \in \mathscr{I}$: On a clairement pour tout $i \leq k$: $|\varnothing \cap E_i| = 0$.

(b) $\mathscr{I}$ *est héréditaire* : si $I \subset J \in \mathscr{I}$ alors $\forall i$, $|I \cap E_i| \leq |J \cap E_i| \leq 1$; d'où $I \in \mathscr{I}$.

(c) *Propriété d'échange* : Considérons $I, J \in \mathscr{I}$ avec $|I| < |J|$. Puisque $(E_i)_i$ est une partition, $J = \bigcup_i (J \cap E_i)$ et $I = \bigcup_i (I \cap E_i)$. L'hypothèse $|I| < |J|$ entraîne donc l'existence d'un $i \leq k$ tel que $|I \cap E_i| < |J \cap E_i| \leq 1$, c'est-à-dire : $|I \cap E_i| = 0$ et $|J \cap E_i| = 1$. Clairement, l'unique élément x de $J \cap E_i$ réalise $I \cup \{x\} \in \mathscr{I}$.

Correction exercice 10.

(a) G est privé d'un sommet de degré maximal à chaque passage de boucle. Or, puisque l'exécution du **repeat** suppose que $E(G) \neq \varnothing$, le degré du sommet v choisi est toujours non nul, de sorte que l'instruction $G = G - v$ s'accompagne nécessairement de la suppression d'une arête. Ainsi, le nombre d'arêtes de G décroît strictement à chaque passage de boucle, et la condition « $G = \varnothing$ » finit par devenir vraie au bout d'un nombre fini d'étapes.

(b) À chaque passage dans la boucle **repeat**, un sommet v est ajouté à C (ligne 4) et des arêtes sont retirées de G, via l'instruction de la ligne 5, qui sont toutes incidentes à v. Ainsi, par construction, chaque arêtes retirée de G est incidente à un sommet de C. Comme à l'issue de l'algorithme, toutes les arêtes de G ont été retirées, on peut conclure que toute arête de G est incidente à un sommet de C, *i.e.* C est une couverture des sommets.

(c) La couverture des sommets obtenue par cette procédure n'es pas toujours minimale. Considérons par exemple le graphe 1—2—3—4—5 qui admet une couverture minimale de taille 2 ($\{2,4\}$). L'algorithme de l'énoncé peut pourtant opérer aux choix suivants : 3 (puisque $\delta_G(3) = 2$ est maximal), puis 1 ($\delta_{G-3}(1) = 1$ est maximal), puis 5 ($\delta_{G-\{3,1\}}(5) = 1$ est maximal). La couverture retournée dans ce cas, $\{1,3,5\}$, n'est pas minimale.

Master 1 d'Informatique — Recherche Opérationnelle
2010 / 2011

TD n° 4

Programmation dynamique

Exercice 1 *(Amélioration Fibonacci).* Soit $(F_n)_{n\geq 0}$ la suite de Fibonacci.

(a) Montrer que pour tout $n > 0$:

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_n \end{pmatrix}$$

(b) En déduire un algorithme en $O(\log n)$ pour le calcul de F_n.

Exercice 2 *(Tri topologique).*

(a) Montrer que tout graphe orienté sans circuit admet un sommet de degré entrant nul.

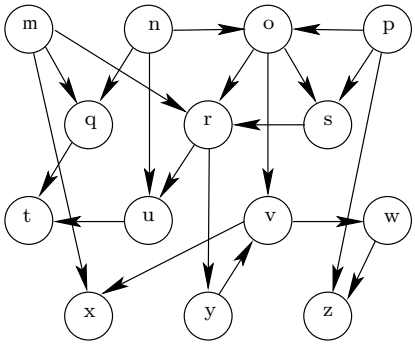
(b) En déduire un algorithme qui produit le tri topologique d'un DAG en temps linéaire.

(c) Prouver sa correction.

(d) Évaluer sa complexité. (On supposera que le graphe est donné par sa matrice d'adjacence.)

(e) Montrer qu'un graphe orienté est sans circuit ssi il admet un tri topologique.

(f) Écrire la liste des sommets obtenue par application de cet algorithme sur le graphe suivant. On supposera qu'à chaque étape nécessitant un choix arbitraire de sommet, ce choix se fait suivant l'ordre alphabétique des étiquettes.



Exercice 3 *(Inversion d'un Graphe).* L'*inverse* d'un graphe orienté $G = (V,E)$ est le graphe $G^r = (V,E^r)$ où E^r est définie par $E^r(x,y)$ ssi $E(y,x)$. Écrire un algorithme de complexité linéaire qui inverse un graphe présenté par listes d'adjacence.

Exercice 4 Déduire des exercices 2 et 3 la complexité exacte de l'algorithme `pccDag`$^{\texttt{iter}}$ vu en cours.

Exercice 5 (a) Améliorer l'algoritme `plsc`$^{\texttt{iter}}$ vu en cours pour qu'il fonctionne en espace $O(\min(m,n))$ plutôt que $O(mn)$.

(b) Comment faire pour calculer, en plus de la longueur des PLSC de X et Y, un exemplaire de PLSC(X,Y) ?

Exercice 6 *(Sous-suites croissantes maximales).* Une *sous-suite* d'une suite d'entiers $\mathbf{a} = a_1,\ldots,a_n$ est une suite de la forme $a_{i_1},\ldots,a_{i_p}$, avec $i_1 < \cdots < i_p$. On parle de sous-suite *croissante* si de plus $a_{i_1} \leq \cdots \leq a_{i_p}$. Par exemple, 2369 est une sous-suite croissante de $5,2,8,6,3,6,9,7$ qui, de surcroît, est de taille maximale parmi les sous-suites croissantes de la suite initiale. On cherche à résoudre le *problème des sous-suites croissantes maximales*, défini comme suit :

SSCM : *Entrée* : une suite d'entiers $\mathbf{a}$;
Sortie : la taille maximale d'une sous-suite croissante de $\mathbf{a}$.

1. Montrer que ce problème se ramène à la recherche d'un plus long chemin dans un DAG.
2. En déduire un algorithme linéaire pour SSCM.

Exercice 7 *(Sous-tableau de somme maximum).* Etant donné un tableau de réels $T[1..n]$, un *sous-tableau* de T est un fragment continu de T. C'est donc un tableau de la forme $T[\ell..h]$, où $1 \leq \ell \leq h \leq n$. La *somme* d'un tableau de nombres est la somme des éléments qui le composent. Le problème du *sous-tableau de somme maximum*, noté STSM, consiste en la recherche de la valeur maximale des sommes des sous-tableau d'un tableau donné. Autrement dit, c'est le problème suivant :

STSM : *Entrée* : un tableau de réels $T[1..n]$;
Sortie : $\max\{\sum_{i=\ell}^{h} T[i],\ 1 \leq \ell \leq h \leq n\}$.

(a) Donner un sous-tableau de somme maximale pour $T = (5,15,-30,10,-5,40,10)$.

(b) Ecrire un algorithme qui résout STSM de manière naïve, en calculant les sommes de tous les sous-tableaux $T[\ell..h]$, pour $1 \leq \ell \leq h \leq n$, et en mettant à jour au fur et à mesure la valeur maximale obtenue.

(c) Donner, en la justifiant, la complexité de cet algorithme.

On se propose maintenant d'aborder ce problème sous l'angle de la programmation dynamique. Commençons par noter $M(j)$ la valeur maximale des sommes des sous-tableaux qui terminent sur la case j. Autrement dit, pour tout $j \in \{1,\ldots,n\}$:

$$M(j) = \max\{\sum_{i=\ell}^{j} T[i],\ 1 \leq \ell \leq j\},$$

(d) Calculer $M(1)$ et établir une relation entre $M(j+1)$ et $M(j)$ pour $j < n$.

(e) Comment s'exprime la somme maximale des sous-tableaux de $T[1..n]$ en fonction de $M(1),\ldots,M(n)$?

(f) En déduire un algorithme en temps linéaire pour STSM.

Exercice 8 *(Rendu de monnaie).* Un commerçant dispose d'une quantité illimitée de pièces de 1, 2, 5 et 10 euros. Il peut obtenir n'importe quelle somme $x \in \mathbb{N}$ à partir de ces pièces, mais cherche à minimiser le nombre total de pièces utilisées.

1. Proposez un algorithme glouton qui, prenant en argument une valeur $x \in \mathbb{N}$, retourne le nombre de pièces minimal $M(x)$ permettant d'obtenir x.
2. Prouvez la correction de votre algorithme.

On appelle *jeu de pièces* tout ensemble d'entiers $J = \{c_1, \ldots, c_n\}$ qui contient 1.

3. Adaptez votre algorithme pour qu'il opère sur un jeu de pièces quelconque $J = \{c_1, \ldots, c_n\}$. On supposera que $c_1 > c_2 > \cdots > c_n = 1$.
4. Montrez, sur un contre-exemple, que votre algorithme ne fonctionne pas pour le jeu $J = \{1,4,6\}$.

On cherche désormais à concevoir un algorithme qui calcule, *pour tout* jeu de pièces J, le nombre de pièces minimal $M_J(x)$ permettant de fabriquer x. On fixe $x \in \mathbb{N}$ et on considère un $c \in J$ tel que $c \leq x$.

5. Montrer que $M_J(x) \leq 1 + M_J(x-c)$.
6. Montrer que $M_J(x) = 1 + M_J(x-c)$ si, et seulement si, c apparaît dans une solution optimale pour x.

Notons a le plus grand entier tel que $a \in J$ et $a \leq x$.

7. Déduire de ce qui précède : $M_J(x) = 1 + \min_{c \in J, c \leq a} M_J(x-c)$.
8. En déduire un algorithme de programmation dynamique qui, prenant en entrée un jeu de pièces J et un montant x, retourne $M_J(x)$.
9. Évaluer sa complexité.
10. Faites tourner votre algorithme sur l'entrée $x = 13$, $J = \{1,4,6\}$.

Exercice 9 *(Les coefficients binomiaux).* Les coefficients binomiaux interviennent dans de nombreuses situations, dans des calculs algébriques, en combinatoire, en probabilité... On veut les calculer de manière efficace. La formule suivante permet un calcul récursif de C_n^p, pour $0 \leq p \leq n$:

$$C_n^n = C_n^0 = 1 \; ; \; C_n^p = C_{n-1}^{p-1} + C_{n-1}^p$$

1. Écrire une fonction récursive $c(n,p)$ qui calcule C_n^p
2. Décrire l'arbre des appels récursifs pour $c(6,3)$ et calculer le nombre d'appels récursifs.
3. Expliquer pourquoi cet algorithme a un coût exponentiel. Indication : décrire l'arbre des appels pour le calcul de $c(2n,n)$ et comparer le nombre d'appels récursifs pour $c(2n,n)$ et $c(2n-2,n-1)$.

4. La fonction précédente a un coût trop élevé, on veut donc maintenant améliorer le calcul. Dessiner le triangle de Pascal pour $n = 5$, et montrer qu'on peut calculer une ligne en se contentant de stocker dans un tableau certains des calculs précédents.
5. Écrire une fonction qui calcule les C_n^p avec un coût polynomial en utilisant la programmation dynamique.

Exercice 10 *(Algorithme de Floyd-Marshall).* On considère le problème suivant : étant donné un graphe valué $G = (V, E, \ell)$ sans cycle négatif, déterminer la longueur d'un plus court chemin de u à v *pour tout couple de sommets* (u, v). On peut résoudre ce problème en applicant l'algorithme de Bellman-Ford à chaque sommet de G. Ceci nécessite un temps $O(V \times VE) = O(V^2E)$. Mais il est possible d'améliorer cette borne par un algorithme issu de la programmation dynamique.

(a) Pour définir une notion pertinente de sous-problèmes, on note $dist(i, j, k)$ le poids d'un plus court chemin de i à j dont tous les sommets intérieurs (*i.e.*, hors extrémités) sont dans $\{1, \ldots, k\}$. Etablissez une relation entre $dist(i, j, k)$ et des valeurs de la fonction $dist$ sur des triplets de "taille" inférieure à celle de (i, j, k). (On utilisera le lien entre les plus courts chemins de i à j passant par $\{1, \ldots, k\}$ et les plus courts chemins de i à k et de k à j passant par $\{1, \ldots, k-1\}$.)

(b) Construire un ordre sur les triplets de V^3 tel que pour chaque triplet (i, j, k), les triplets nécessaires au calcul de $dist(i, j, k)$ (selon la relation trouvée en (a)) soient avant (i, j, k) selon cet ordre.

(c) En déduire l'algorithme attendu.

Exercice 11 *(Distance d'édition).* Etant donnés deux mots $X = x_1 \ldots x_m$ et $Y = y_1 \ldots y_n$ sur un alphabet V, on appelle *alignement* de (X, Y) tout couple de mots (X', Y') sur $V \cup \{-\}$ tel que :

1. $|X'| = |Y'|$;
2. $X' \setminus - = X$ et $Y' \setminus - = Y$;
3. $\forall i \leq |X'|$, on n'a pas simultanément $X'[i] = -$ et $Y'[i] = -$.

Le *coût* d'un tel alignement est le nombre de positions sur lesquelles X' et Y' diffèrent. Autrement dit :

$$\text{coût}(X', Y') = \text{card}\{i \text{ t.q. } X'[i] \neq Y'[i]\}.$$

Exemple :

A	B	B	$-$	B	A	$-$	B	B	R	C	A
A	B	R	A	C	A	D	A	B	R	$-$	A

est un alignement de $(ABBBABBRCA, ABRACADABRA)$ de coût 6. La *distance d'édition* de X à Y est le coût minimal d'un alignement de (X, Y). On la note $edit(X, Y)$.

Écrire un algorithme qui calcule $edit(X, Y)$.

Master 1 d'Informatique — Recherche Opérationnelle
2010 / 2011

Corrigé du TD n° 4

Programmation dynamique

Correction exercice 1.

(a) $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} F_{n-1} \\ F_n \end{pmatrix} = \begin{pmatrix} F_n \\ F_{n-1}+F_n \end{pmatrix} = \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$

(b) D'où l'on déduit : $\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} F_0 \\ F_1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

Le calcul de F_n se ramène donc à celui de $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n$ qui se fait en $O(\log n)$ multiplications scalaires, grâce à l'exponentiation rapide. Si l'on convient que le coût d'une telle multiplication est constant (ce qui est discutable dans ce contexte...), on obtient le résultat annoncé.

Correction exercice 2.

(a) Soit $p = x_1 x_2 \dots x_k$ un chemin élémentaire de longueur maximale dans G. Alors x_1 ne peut avoir de prédécesseur. Car si tel était le cas, un prédécesseur x de x_1 ne pourrait appartenir à p (sinon le graphe comporterait un cycle), et par conséquent, $p' = x x_1 x_2 \dots x_k$ serait un chemin élémentaire de G strictement plus long que p : une contradiction. Il s'ensuit que $\delta^-(x_1) = 0$.

(b) Outre la structure de donnée matricielle qui code le graphe, l'algorithme ci-dessous utilise un tableau d^- pour tenir à jour les degrés entrants et faciliter le choix d'un sommet de degré entrant nul. Un ensemble F est par ailleurs créé, qui contient les sommets non encore traités (*i.e.*, non affichés).

Algorithme 2: Tri topologique

```
Données : G sous forme d'une matrice G[x,y]
Résultat : Tri topologique à l'affichage
pour x = 1 à n faire
    pour y = 1 à n faire  si G[x,y]==1 alors  d⁻[y]++
fin
F = V;
tant que F ≠ ∅ faire
    Choisir x ∈ F tel que d⁻[x] = 0;
    Afficher x;
    F = F − {x};
    pour y = 1 à n faire
        si G[x,y]==1 alors  d⁻[y]−−
    fin
fin
```

(c) L'assertion suivante est un invariant de la boucle **tant que** :

$$\forall y \in F,\ d^-[y] \text{ est le nombre de prédécesseurs de } y \text{ appartenant à } F. \qquad (*)$$

En effet, si elle est vraie avant un passage de boucle, elle l'est encore après puisque les seules instructions modifiant d^- dans la boucle consistent à décrémenter $d^-[y]$ pour les successeurs y d'un sommet qui vient précisément d'être retiré de F, et pour eux seulement. Comme par ailleurs $(*)$ est vraie dès lors que l'initialisation de F a été faite (puisqu'à ce moment-là, $F = V$ et d^- contient les degrés entrants des sommets de G), il s'ensuit qu'elle est vraie à chaque passage de boucle. Par conséquent, le sommet x retiré de F et affiché lors d'un passage étant caractérisé par $d^-[x] = 0$, on est assuré qu'aucun de ses successeurs ne sera affiché après lui. L'ordre d'affichage réalisé par l'algorithme est donc bien topologique.

(d) Les deux boucles **pour** imbriquées qui réalisent l'initialisation de d^- s'exécutent clairement en $O(n^2)$ (où n est le nombre de sommets du graphe). La boucle **tant que** est traversée n fois, puisque F est initialisé à V et que chaque passage dans la boucle décrémente F d'exactement *un* sommet. À l'intérieur de la boucle, la première ligne nécessite, dans le pire des cas, un parcours intégral du tableau d^- ; coût : $O(n)$. Les deux instructions suivantes se font en temps constant et la dernière (boucle **pour**) s'exécute en $O(n)$. Au total, la boucle **tant que** s'exécute en temps $O(n^2)$: c'est aussi la complexité de l'algorithme. Notons que pour une présentation par matrice d'adjacence, cette complexité est qualifiée de *linéaire*, puisqu'elle est proportionnelle à la taille de l'entrée.

(e) L'implication $\Rightarrow$ découle de la preuve de correction établie en (c). La réciproque est immédiate.

(f) *mnpoqsrutyvwxz*

Correction exercice 3. Soit $G = (V, E)$ un graphe orienté. Pour chaque $x \in V$, on note $succ(x)$ l'ensemble des successeurs de x (*i.e.*, l'ensemble des $y \in V$ tels que $xy \in E$). Si G^r désigne le graphe inverse de G et si pour $y \in V$, on note $succ^r(y)$ la liste des successeurs de y dans G^r, on a clairement : $y \in succ(x)$ ssi $x \in succ^r(y)$. L'algorithme qui suit en découle facilement : il déclare un tableau G^r indexé par les sommets $y \in V$, chaque y pointant vers vers une liste de successeurs initialisée à $\varnothing$. Puis on parcourt les G-successeurs y de chaque sommet $x \in V$, et pour chacun d'entre eux, on rajoute x à la liste des G^r-successeurs de y.

Algorithme 3: Inversion d'un graphe

Données : G sous forme de listes de successeurs
Résultat : Le graphe inverse de G
pour $y \in V$ **faire** $succ^r(y) = \varnothing$;
pour $x \in V$ **faire**
  pour $y \in succ(x)$ **faire**
    $succ^r(y) = succ^r(y) + x$
  fin
fin
retourner G^r

Complexité : la première boucle **pour** prend clairement un temps $O(|V|)$. L'instruction $succ^r(y) = succ^r(y) + x$ provoque l'ajoût de x en tête de la liste chaînée des successeurs de y dans G^r : elle s'exécute en temps constant. Les boucles **pour** imbriquées réalisent un parcours de tous les successeurs de chaque sommet de G avec exécution, pour chacun d'entre eux, d'une instruction en temps constant. Le coût total de cette double boucle, et donc de l'algorithme, est en $O(|V|+|E|)$.

Correction exercice 4. Rappelons l'algorithme vu en cours, pour le calcul des distances à un sommet fixé :

Algorithme 4: `pccDag`$^{\texttt{iter}}$

Données : un DAG pondéré $G = (V, E, \ell)$ et un sommet $s \in V$.
Résultat : Les distances de s à tout sommet de G
`initialiser` (d,s) ;
Trier topologiquement V ;
pour chaque $v \in V$ *dans l'ordre croissant* **faire**
  $\mathrm{d}(v) = \min_{u \in pred(v)} \{\mathrm{d}(u) + \ell(u,v)\}$
fin
retourner d

Nous avons vu que l'initialisation de d se fait en $O(|V|)$. Le tri topologique se fait en $O(|V|+|E|)$ (Voir Exercice 2). Reste la dernière instruction, exécutée pour chaque $v \in V$. Il est facile de voir que son temps d'exécution est du même ordre que celui du parcours de la liste des prédécesseurs de v (le calcul du min se faisant au fur et à mesure de ce parcours, il ne modifie le temps de calcul que par une constante multiplicative). Dans le cas d'une représentation du graphe par matrice d'adjacence, le parcours de la liste des prédécesseurs de v se fait en $O(|V|)$: il suffit de scanner les 1 dans la colonne de rang v de la matrice. Pour une représentation par liste de successeurs, c'est plus délicat : repérer tous les prédecesseurs de v nécessite un parcours complet des listes de successeurs, ce qui se fait en $O(|V|+|E|)$. Plutôt que de consommer ce temps pour chaque sommet, il est préférable d'inverser le graphe une fois pour toute, de sorte que la recherche des $u \in pred(v)$ se ramène à celle des $u \in succ^r(v)$, bien plus rapide. L'algorithme s'écrit alors :

Algorithme 5: `pccDag`$^{\texttt{iter}}$ avec inversion du graphe

Données : un DAG pondéré $G = (V, E, \ell)$ et un sommet $s \in V$.
Résultat : Les distances de s à tout sommet de G
`initialiser` (d,s) ;
Trier topologiquement V ;
Calculer G^r ;
pour chaque $v \in V$ *dans l'ordre croissant* **faire**
  $\mathrm{d}(v) = \min_{u \in succ^r(v)} \{\mathrm{d}(u) + \ell(u,v)\}$
fin
retourner d

Aux temps de calcul $O(|V|)$ (initialisation de d) et $O(|V|+|E|)$ (tri topologique) déjà évoqués, s'ajoutent maintenant le temps consommé par le calcul de G^r, soit $O(|V|+|E|)$ d'après l'Exercice 3, et celui correspondant à la boucle **pour** qui consiste essentiellement en un parcours des successeurs de tous les sommets du graphe, et qui s'exécute donc en $O(|V|+|E|)$. La complexité globale de l'algorithme ainsi détaillé est donc bien en $O(|V|+|E|)$, comme annoncé dans le cours.

Correction exercice 5.

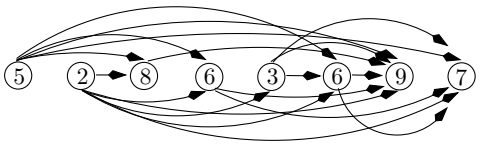
(a) On convient ici que pour une liste $T=[t_0,t_1,\ldots,t_n]$, l'instruction « insérer r en fin de T » remplace la T par la liste $[t_1,t_2,\ldots,t_n,r]$. On améliore alors l'algorithme `plsc`$^{\texttt{iter}}$ en utilisant un tableau de dimension 1 et de taille n, de la manière suivante :

Algorithme 6: Amelioration de `plsc`$^{\texttt{iter}}$

```
Données : Deux suites X et Y
Résultat : La longueur d'une plus longue sous-suite commune de X et Y
m = lg(X) ; n = lg(Y) ;
pour j = 0 à n faire T[j] = 0 ;
pour i = 1 à m faire
    pour j = 0 à n faire
        si [X]_i = [Y]_j alors
            r = 1 + T[0]
        sinon
            r = max{ T[1], T[n] }
        fin
        insérer r en fin de T
    fin
fin
retourner T(n) ;
```

La procédure « insérer r en fin de T » peut être exécutée en temps constant en représentant T par une liste chaînée avec un pointeur en tête et un pointeur en fin de liste. Il s'ensuit facilement que la complexité en temps de cet algorithme est en $O(mn)$, comme l'algorithme initial. Par contre, la complexité en espace devient un $O(n)=O(\min(m,n))$. (On a supposé ici que $n\leq m$. Dans le cas contraire, il suffit d'inverser les rôles de m et n.)

Correction exercice 6. À chaque suite d'entiers $\mathbf{a}=a_1,\ldots,a_n$ on associe le graphe orienté $G_{\mathbf{a}}$ défini comme suit : $G_{\mathbf{a}}$ comporte n sommets étiquetés par a_1, …, a_n ; un arc joint a_i à a_j si et seulement si $i<j$ et $a_i\leq a_j$. Par exemple, pour la suite $\mathbf{a}=5,2,8,6,3,6,9,7$ on obtient pour $G_{\mathbf{a}}$:



Par construction, le tri des sommets de $G_{\mathbf{a}}$ selon l'ordre $a_1,\ldots,a_n$ de leurs étiquettes est topologique. Le graphe est donc sans circuit. Par ailleurs, on se convainc aisément du fait que les sous-suites croissantes de longueur maximale de **a** correspondent aux chemins de longueur maximale dans $G_{\mathbf{a}}$. Le problème posé, SSCM(**a**), se ramène donc au calcul de la longueur maximale d'un chemin dans le DAG $G_{\mathbf{a}}$, pondéré par la fonction qui affecte la même longueur 1 à tous les arcs. D'où l'algorithme, directement adapté de `pccDag`$^{\texttt{iter}}$ (voir exercice 4) :

Algorithme 7: `sscm`

```
initialiser (d,s) ;
max = 0 ;
pour chaque j = 1,...,n faire
    d(j) = 1 + max{d(i), i ∈ pred(j)} ;
    si d(j) > max alors max = d(j) ;
fin
retourner max ;
```

Correction exercice 7.

(a) $(10,-5,40,10)$ est un sous-tableau de somme $10-5+40+10=55$ maximale.

(b) Algorithme très naïf :

Algorithme 8: `stsm-tres-naif`

```
Données : Un tableau de réels T[1..n]
Résultat : La valeur maximale de la somme d'un sous-tableau de T
res = 0 ;
pour ℓ = 1 à n faire
    pour h = ℓ + 1 à n faire
        S = T[ℓ] ;
        pour i = ℓ + 1 à h faire S = S + T[i] ;
        res = max(res, S) ;
    fin
fin
retourner res ;
```

Complexité : $O(n^3)$(grâce aux identités : $\displaystyle\sum_{j=1}^{p} j = \frac{p(p+1)}{2}$ et $\displaystyle\sum_{j=1}^{p} j^2 = \frac{p(p+1)(2p+1)}{6}$).

Un algorithme moins naïf pour ce problème consiste à parcourir chaque sous-tableau $T[\ell..j]$ en sommant ses éléments. Une variable (*res*) est comparée à la somme S du sous-tableau et remplacée par S si $S > res$.

Algorithme 9: `stsm-naif`

Données : Un tableau de réels $T[1..n]$
Résultat : La valeur maximale de la somme d'un sous-tableau de T
$res = 0$;
pour $\ell = 1$ *à* n **faire**
  $S = T[\ell]$;
  pour $h = \ell + 1$ *à* n **faire** $S = S + T[h]$;
  $res = \max(res, S)$;
fin
retourner *res* ;

(c) Toutes les instructions élémentaires s'exécutent en temps constant. La complexité de l'algorithme est donc proportionnelle au nombre de passage dans la boucle imbriquée. Or pour chaque $\ell = 1, \ldots, n$, la boucle indexée par j s'exécute $n - \ell$ fois. Au total, l'exécution des deux boucle prend donc un temps propotionnel à $\sum_{\ell=1}^{n}(n-\ell) = n^2 - \sum_{\ell=1}^{n}\ell = n^2 - \frac{n(n+1)}{2} = \frac{n(n-1)}{2}$. Finalement, la complexité globale de l'algorithme est en $O(n^2)$.

(d) $M(1) = \max\{\sum_{i=\ell}^{1} T[i],\ 1 \leq \ell \leq 1\} = T[1]$.

Pour $1 \leq \ell \leq j \leq n$, notons $S(\ell, j)$ la somme $\sum_{i=\ell}^{j} T[i]$. Ainsi, la fonction $M(j)$ de l'énoncé s'écrit, pour $j \in \{1, \ldots, n\}$: $M(j) = \max\{S(\ell, j),\ 1 \leq \ell \leq j\}$. De plus, pour tous $1 \leq \ell \leq j \leq n$: $S(j, j) = T[j]$ et, si $j < n$, $S(\ell, j+1) = S(\ell, j) + T[j+1]$.

$$\begin{aligned} M(j+1) = &\ \max\{S(\ell, j+1),\ 1 \leq \ell \leq j+1\} \\ = &\ \max\{S(j+1, j+1),\ \max\{S(\ell, j+1),\ 1 \leq \ell \leq j\}\} \\ = &\ \max\{T[j+1],\ \max\{S(\ell, j) + T[j+1],\ 1 \leq \ell \leq j\}\} \\ = &\ \max\{T[j+1],\ T[j+1] + \max\{S(\ell, j),\ 1 \leq \ell \leq j\}\} \\ = &\ \max\{T[j+1],\ T[j+1] + M(j)\} \end{aligned}$$

Finalement, $M(j+1) = T[j+1] + \max\{0,\ M(j)\}$.

(e) La somme maximale des sous-tableaux de $T[1..n]$ s'écrit $\text{STSM}(T) = \max\{S(\ell, j),\ 1 \leq \ell \leq h \leq n\}$, c'est à dire : $\text{STSM}(T) = \max\{M(j),\ 1 \leq j \leq n\}$.

(f) L'algorithme tire parti de l'égalité établie en (d) pour calculer chaque $M(j)$ par simple mise à jour de la valeur de $M(j-1)$ selon que celle-ci est positive ou non. Les valeurs successivement calculées sont stockées dans une variable M. Le maximum des $M(j)$ – qui est le résultat cherché, selon (e) –, est stocké dans une variable *res* et calculé à la volée : *res* est comparé à chaque nouvelle valeur de M et mis à jour si nécessaire.

Algorithme 10: `stsm`

Données : Un tableau de réels $T[1..n]$
Résultat : La valeur maximale de la somme d'un sous-tableau de T
$res = 0$; $M = T[1]$;
pour $j = 2$ *à* n **faire**
    $M = T[j] + \max(0, M)$;
    $res = \max(res, M)$;
fin
retourner res ;

L'algorithme s'exécute clairement en temps $O(n)$.

Correction exercice 8.

1. Algorithme glouton pour le calcul de $M(x)$:

```
   Données : Un entier x ∈ ℕ
   Résultat : M(x)
1  res = 0 ;
2  pour c = 10, 5, 2, 1 pris dans cet ordre faire
3      tant que x − c ≥ 0 faire
4          x = x − c ;
5          res++ ;
       fin
   fin
6  retourner res
```

2. Prouvez la correction de votre algorithme.
3. Adaptation de l'algorithme précédent à un jeu de pièces quelconque $J = \{c_1, \ldots, c_n\}$, avec $c_1 > \cdots > c_n = 1$.

```
   Données : Un entier x ∈ ℕ
   Résultat : M_J(x)
1  res = 0 ;
2  pour i = 1 à n faire
3      tant que x − c_i ≥ 0 faire
4          x = x − c_i ;
5          res++ ;
       fin
   fin
6  retourner res
```

4. Pour le jeu $J = \{1, 4, 6\}$ et l'entrée $x = 8$, l'algo précédent renvoie 3 (puisque $8 = 6+1+1$) alors que l'optimum est 2 ($8 = 4+4$).

On fixe $x \in \mathbb{N}$ et on considère un $c \in J$ tel que $c \leq x$.

5. Si l'on peut construire $x-c$ avec k pièces de J, alors on peut construire x avec $k+1$ pièces de J (en rajoutant une pièce de valeur c aux k pièces précédentes). Ainsi, le nombre minimal de pièces de J nécessaire pour construire x est inférieur à $k+1$. En particulier, en prenant pour k le nombre $M_J(x-c)$, on obtient la relation cherchée : $M_J(x) \leq 1+M_J(x-c)$.
6. Montrer que $M_J(x) = 1+M_J(x-c)$ si, et seulement si, c apparaît dans une solution optimale pour x.

   $\boxed{\Leftarrow}$ Si c apparaît dans une solution optimale pour x, alors l'ensemble de pièces obtenu en retirant de cette solution une pièce de valeur c est une solution optimale pour $x-c$. D'où $M_J(x-c) = M_J(x)-1$.

   $\boxed{\Rightarrow}$ Si $M_J(x) = 1+M_J(x-c)$, notons $a_1,\ldots,a_k$ une solution optimale pour $x-c$. En d'autres termes, supposons que :
   – $k = M_J(x-c)$;
   – les a_i sont des éléments non nécessairement distincts de J ;
   – $x-c = a_1+\cdots+a_k$.

   Alors $a_1,\ldots,a_k,c$ est une solution pour x puisque $x = a_1+\cdots+a_k+c$ et puisque c, comme chaque a_i, est dans J. Cette solution étant de taille $k+1$, il s'ensuit que $M_J(x) \leq k+1$, c'est-à-dire, $M_J(x) \leq 1+M_J(x-c)$. L'inégalité inverse, prouvée dans la question précédente, permet de conclure.
7. Soit $\mathscr{E}_x$ l'ensemble des entiers de la forme $1+M_J(x-c)$ obtenus pour les $c \in J$ inférieurs à x. Autrement dit :

$$\mathscr{E}_x = \{1+M_J(x-c) \text{ pour } c \in J, c \leq x\}.$$

   La question 5 se traduit facilement en : $M_J(x)$ minore $\mathscr{E}_x$ (*i.e.*, est inférieur à chaque élément de $\mathscr{E}_x$). La question 6 affirme pour sa part que tout c apparaissant dans une solution optimale pour x réalise l'égalité $M_J(x) = 1+M_J(x-c)$. Ceci entraîne notamment que l'un des entiers de $\mathscr{E}_x$ au moins a pour valeur $M_J(x)$. Ainsi, $M_J(x)$ minore l'ensemble $\mathscr{E}_x$ et appartient à cet ensemble. Cela s'écrit $M_J(x) = \min \mathscr{E}_x$, ou encore

$$\begin{aligned} M_J(x) &= \min\{1+M_J(x-c) \text{ pour } c \in J, c \leq x\} \\ &= 1+\min\{M_J(x-c) \text{ pour } c \in J, c \leq x\} \end{aligned}$$

   C'est l'égalité cherchée.
8. En déduire un algorithme de programmation dynamique qui, prenant en entrée un jeu de pièces J et un montant x, retourne $M_J(x)$.

Données : Un entier $x \in \mathbb{N}$; un jeu de pièces J
Résultat : $M_J(x)$
$M[0] = 0$;
pour $y = 1$ *à* x **faire**
  $M[y] = 1+\min\limits_{c \in J, c \leq y} M[y-c]$;
fin
retourner $M[x]$

---

9. Complexité : $O(x|J|)$.
10. Simulation de l'algorithme sur l'entrée $x = 13$, $J = \{1,4,6\}$:

0	1	2	3	1	2	1	2	2	3	2	3	2	**3**

D'où : $M_{\{1,4,6\}}(13) = 3$.

Distance d'édition

Soit Σ un alphabet et $-$ un symbole n'apparaissant pas dans Σ. Pour tout mot $\mathbf{u} \in (\Sigma \cup \{-\})^*$, on note $\mathbf{u} \setminus -$ le mot obtenu à partir de $\mathbf{u}$ en supprimant toutes les occurences de $-$. Une *expansion* d'un mot $\mathbf{x} \in \Sigma^*$ est un mot $\mathbf{u} \in (\Sigma \cup \{-\})^*$ tel $\mathbf{u} \setminus - = \mathbf{x}$. Un *alignement* de deux mots $\mathbf{x}, \mathbf{y} \in \Sigma^*$ est un couple de mots $(\mathbf{u} = u_1 \dots u_p, \mathbf{v} = v_1 \dots v_q)$ tel que :

1. $p = q$;
2. $\forall i \leq p : u_i \neq -$ ou $v_i \neq -$;
3. $\mathbf{u}$ est une expansion de $\mathbf{x}$ et $\mathbf{v}$ est une expansion de $\mathbf{y}$.

On appelle *coût* d'un tel alignement le nombre de positions sur lesquelles $\mathbf{u}$ et $\mathbf{v}$ diffèrent, *i.e.* l'entier

$$\text{coût}(\mathbf{u}, \mathbf{v}) = |\{i \leq p : u_i \neq v_i\}|.$$

La *distance d'édition* de deux mots $\mathbf{x}, \mathbf{y}$ est le coût minimal d'un alignement de ces deux mots. On la note $edit(\mathbf{x}, \mathbf{y})$.

Lemme 1 $\forall \mathbf{x}, \mathbf{y} \in \Sigma^*$, $\forall a \in \Sigma$ *:* $edit(\mathbf{x}a, \mathbf{y}a) = edit(\mathbf{x}, \mathbf{y})$.

PREUVE. Soit $(u_1 \dots u_p, v_1 \dots v_p)$ un alignement de $\mathbf{x}a, \mathbf{y}a$. Soient $i, j \leq p$ les entiers maximaux pour lesquels $u_i \neq -$ et $v_j \neq -$. Alors $u_i = v_j = a$, car la dernière lettre d'une expansion d'un mot qui diffère de $-$ est égale à la dernière lettre de ce mot. Par ailleurs, l'un de ces entiers au moins vaut p, sinon on aurait $x_p = y_p = -$. Supposons donc, sans perte de généralité, que $i = p$ (*i.e.* $u_p = a$). Les deux expansions s'écrivent alors :

$$\begin{array}{cccccc} u_1 \dots & u_{j-1} & u_j & u_{j+1} \dots & u_{p-1} & a \\ v_1 \dots & v_{j-1} & a & - \dots & - & - \end{array}$$

avec $u_{j+1}, \dots, u_{p-1} \in \Sigma^*$. Il s'ensuit que le couple

$$\begin{array}{cccccc} u_1 \dots & u_{j-1} & u_j & u_{j+1} \dots & u_{p-1} & a \\ v_1 \dots & v_{j-1} & - & - \dots & - & a \end{array}$$

est un nouvel alignement de $\mathbf{x}a, \mathbf{y}a$ de coût inférieur ou égal au précédent (égal si $u_j = a$, inférieur sinon). En particulier, si le premier alignement est optimal, le second l'est aussi. Tout ceci permet d'affirmer que le couple $(\mathbf{x}a, \mathbf{y}a)$ admet un alignement optimal dont les termes terminent tous deux par la lettre a.

Considérons donc un alignement optimal de la forme $(\mathbf{u}a, \mathbf{v}a)$ pour $\mathbf{x}a, \mathbf{y}a$. Clairement, $(\mathbf{u}, \mathbf{v})$ est un alignement de $\mathbf{x}, \mathbf{y}$. De plus, pour tout autre alignement $(\mathbf{r}, \mathbf{s})$ de $\mathbf{x}, \mathbf{y}$, $(\mathbf{r}a, \mathbf{s}a)$ est un alignement de $\mathbf{x}a, \mathbf{y}a$ et l'on a : $\text{coût}(\mathbf{r}, \mathbf{s}) = \text{coût}(\mathbf{r}a, \mathbf{s}a)$, $\text{coût}(\mathbf{u}, \mathbf{v}) = \text{coût}(\mathbf{u}a, \mathbf{v}a)$, et $\text{coût}(\mathbf{r}a, \mathbf{s}a) \geq \text{coût}(\mathbf{u}a, \mathbf{v}a)$, par optimalité de $(\mathbf{u}a, \mathbf{v}a)$. Par conséquent, $\text{coût}(\mathbf{r}, \mathbf{s}) \geq \text{coût}(\mathbf{u}, \mathbf{v})$. Ainsi, $\text{coût}(\mathbf{u}, \mathbf{v})$ minore le coût de tout alignement de $\mathbf{x}, \mathbf{y}$: c'est un alignement optimal.

On obtient finalement :
- $\mathit{edit}(\mathbf{x}, \mathbf{y}) = \text{coût}(\mathbf{u}, \mathbf{v})$, par optimalité de $(\mathbf{u}, \mathbf{v})$ pour $\mathbf{x}, \mathbf{y}$;
- $\mathit{edit}(\mathbf{x}a, \mathbf{y}a) = \text{coût}(\mathbf{u}a, \mathbf{v}a)$, par optimalité de $(\mathbf{u}a, \mathbf{v}a)$ pour $\mathbf{x}a, \mathbf{y}a$;
- $\text{coût}(\mathbf{u}, \mathbf{v}) = \text{coût}(\mathbf{u}a, \mathbf{v}a)$.

D'où le résultat annoncé. □

Master 1 d'Informatique — Recherche Opérationnelle
2010 / 2011

TD n° 5

Flot maximum

Exercice 1 Trouver le flot maximum du réseau de la figure 2 à l'aide de l'algorithme de Ford-Fulkerson.

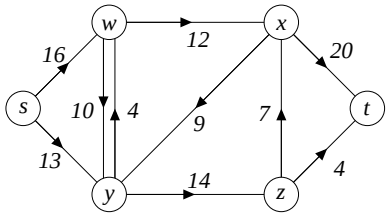


FIGURE 2 – Un réseau

Exercice 2 Donner un exemple d'exécution de l'algorithme de Ford-Fulkerson qui accomplit $\Theta(|E||f^*|)$ améliorations, où f^* est un flot maximal.

Exercice 3 Montrer que le flot de la figure 3 est maximal.

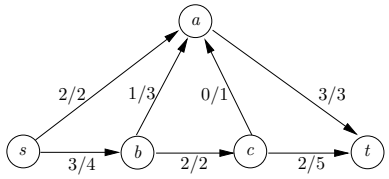


FIGURE 3 – Un flot maximal

Exercice 4 *Réseaux à sources et puits multiples.* Soient $G = (V,E)$ un graphe orienté, c une fonction de capacité sur G, S et T deux sous-ensembles disjoints de V, dont les éléments sont appelés *sources* et *puits* respectivement. Un flot sur un tel graphe est une fonction $f : E \to E$ qui satisfait aux deux contraintes :

1. $\forall (u,v) \in E : \ 0 \leq f(u,v) \leq c(u,v)$;
2. $\forall u \notin S \cup T$, $f(u,V-u) = f(V-u,u)$.

La valeur du flot f est alors défini par : $|f| = f(S,V \setminus S)$. Donner un algorithme pour calculer le flot maximum dans ce contexte, en se ramenant à un problème de flot maximum sur un réseau à source unique et puits unique.

Exercice 5 Chacun des n chercheurs d'un laboratoire doit utiliser un ordinateur vectoriel pour effectuer des calculs pendant une ou plusieurs périodes d'une heure. Chaque chercheur i a donné l'ensemble $c_i = \{h_i^1, \ldots, h_i^{d_i}\}$ des créneaux horaires où il est disponible pour effectuer ses calculs, ainsi que le nombre r_i de créneaux ($r_i \leq d_i$) dont il a effectivement besoin.

On cherche à affecter les créneaux horaires aux chercheurs de manière à satisfaire tout le monde, sachant que l'ordinateur ne peut accueillir plus de deux personnes à la fois.

(a) Modéliser ce problème en terme de problème de flots.

(b) Résoudre le problème pour les entrées suivantes :

$c_1 = \{1,2,4,5\}$	$r_1 = 3$	$c_1 = \{1,2,3,4,5\}$	$r_1 = 3$
$c_2 = \{2,3,4,6\}$	$r_2 = 2$	$c_2 = \{2,3,4\}$	$r_2 = 2$
$c_3 = \{1,3,4\}$	$r_3 = 2$	$c_3 = \{2,3\}$	$r_3 = 2$
$c_4 = \{2,3,4,5\}$	$r_4 = 3$	$c_4 = \{3,4\}$	$r_4 = 2$
$c_5 = \{4,5,6\}$	$r_5 = 1$	$c_5 = \{2,4\}$	$r_5 = 1$
		$c_6 = \{1,3,5\}$	$r_6 = 1$

Exercice 6 Un graphe non orienté $G = (V,E)$ est *biparti* si son domaine V admet une partition $V = L \cup R$ telle que toute arête $(u,v) \in E$ ait une extrémité dans L et l'autre dans R. Un *couplage* dans un tel graphe est un ensemble d'arêtes $M \subseteq E$ tel que chaque sommet $u \in V$ touche au plus une arête de M. Un couplage *maximal* est un couplage de cardinal maximal.

Soit $G = (V,E)$ un graphe biparti pour la partition $V = L \cup R$. On considère le réseau $G' = (V',E')$ défini comme suit : $V' = V \cup \{s,t\}$ où s et t sont deux nouveaux sommets n'appartenant pas à V ; $E' = (E \cap (L \times R)) \cup (\{s\} \times L) \cup (R \times \{t\})$. Chaque arc est étiqueté d'une capacité 1.

(a) Soit M un couplage de G. Montrer qu'il existe un flot f sur G' de valeur $|f| = |M|$.

(b) Soit f un flot sur G'. Montrer qu'il existe un couplage de G de cardinal $|M| = |f|$.

Exercice 7 *Connectivité et flot.* La *connectivité* d'un graphe non orienté est le nombre minimum d'arêtes qu'il faut supprimer pour déconnecter le graphe.

(a) Quelle est la connectivité d'un arbre ? D'un cycle ?

(b) Comment calculer la connectivité d'un graphe en exécutant un algorithme de recherche du flot maximum sur au plus $|V|$ réseaux, chacun comportant $O(V)$ sommets et $O(E)$ arcs ?

Exercice 8 *(Intersection et union de coupures minimales.).* Soient $(S_1, \overline{S}_1)$ et $(S_2, \overline{S}_2)$ deux coupures minimales dans un réseau. Montrer que les coupures $(S_1 \cup S_2, \overline{S_1 \cup S_2})$ et $(S_1 \cap S_2, \overline{S_1 \cap S_2})$ sont minimales.

Exercice 9 *(Chemins intérieurement sommet-disjoints).* À l'aide de l'algorithme de FORD-FULKERSON, trouver le nombre maximum de st-chemins intérieurement sommet-disjoints du graphe de la figure 4.

Exercice 10 La connexité $\kappa(u,v)$ entre deux sommets u et v d'un graphe est le nombre minimal de sommets que l'on doit retirer du graphe pour déconnecter u et v. La connexité de G, notée $\kappa(G)$, est la cardinalité minimum d'un ensemble de sommets S tel que $G - S$ n'est pas connexe. Un graphe est k-connexe si $\kappa(G) \geq k$.

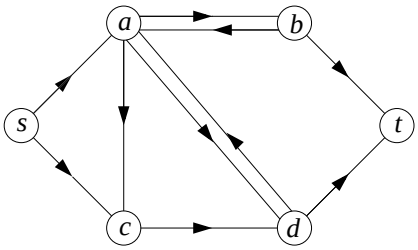


FIGURE 4 – Chemins intérieurement disjoints

(*a*) Montrer que $\kappa(G) = \min\{(u,v) | u,v \in V(G)\}$.

(*b*) Montrer que $\kappa(u,v)$ est égal au nombre maximum de chemins deux à deux sommet-disjoints reliant u et v.

(*c*) En déduire le théorème suivant du à Menger : un graphe est k-connexe si et seulement si pour tout couple de sommets (u,v), il existe k (u,v)-chemins sommet-disjoints.

(*d*) Donner un exemple de graphe régulier de degré k et de connexité 1.

(*e*) Montrer que un graphe est régulier de degré $4k$, alors il contient un sous-graphe k-connexe. (On pourra montrer le résultat par induction sur le nombre de sommets de G).

Exercice 11 *(Un étrange phénomène).* L'algorithme de Ford-Fulkerson commence par le flot nul. Cette contrainte est importante comme nous allons le voir maintenant.

On pose α la racine positive de l'équation $x^3 + x - 1 = 0$, et on considère le réseau (G,c,s,t) décrit Fig. 5 où toutes les capacités sont ∞. On part du flot f_0 donné sur cette même figure. Considérer la suite de chemins augmentants :

- $\forall m$, $p_{4m} = scdabt$, avec $s \leftarrow c \rightarrow d \leftarrow a \rightarrow b \leftarrow t$
- $\forall m$, $p_{4m+1} = scbadt$, avec $s \leftarrow c \rightarrow b \leftarrow a \rightarrow d \leftarrow t$
- $\forall m$, $p_{4m+2} = sabcdt$, avec $s \leftarrow a \rightarrow b \leftarrow c \rightarrow d \leftarrow t$
- $\forall m$, $p_{4m+3} = sadcbt$, avec $s \leftarrow a \rightarrow d \leftarrow c \rightarrow b \leftarrow t$

(*a*) Que vaut le flot max de ce réseau ?

(*b*) Calculer par récurrence le flot f_i $(i > 0)$ obtenu à partir de f_{i-1} par augmentation le long du chemin p_{i-1}.

(*c*) En déduire $\lim_{i \to \infty} val(f_i)$.

Exercice 12 *(Un algorithme de couplage sur les bipartis).* La démonstration du théorème Min-Max de KÖNIG donne immédiatement l'algorithme (très schématique) suivant :

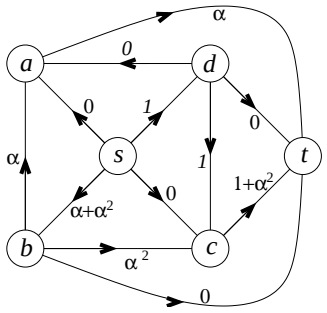


FIGURE 5 – Un flot bizarre

Algorithme 11: Un algorithme de couplage

Données : un graphe biparti G, et $A \cup B$ une bipartition de G
Résultat : la taille d'un plus grand couplage M de G.
Orienter toutes les arêtes $\{a,b\}$ (avec $a \in A$ et $b \in B$) de a vers b;
Rajouter un sommet s et toutes les arêtes $s \to a$, $a \in A$;
Rajouter un sommet t et toutes les arêtes $b \to t$, $b \in B$;
Soit G' ce nouveau graphe ;
Calculer une st-sommet coupure (ou le nombre maximal de st-chemins intérieurement sommet-disjoints) de G;

Le problème, c'est que le calcul d'une sommet-coupure est lourd à implanter.

(a) Montrer que dans G', les st-chemins sommet-disjoints sont les st-chemins arête-disjoints. Donc on peut se contenter de calculer le nombre maximal de st chemins arête-disjoints de G'.

(b) Faire tourner cette version améliorée sur le graphe de la figure 6 :

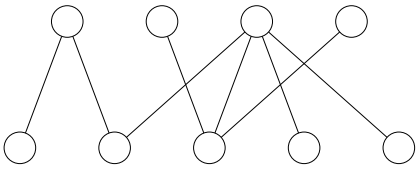


FIGURE 6 – Quel est le couplage maximum ?

Exercice 13 *(L'algorithme d'*EDMONDS-KARP*)*. L'objectif de cet exercice est de prouver la complexité de l'algorithme d'EDMONDS-KARP, qui calcule un flot maximal en temps $O(VE^2)$. Cet algorithme ne se différencie de celui de FORD-FULKERSON que par l'heuristique suivante : dans chaque nouveau graphe résiduel, l'algorithme d'EDMONDS-KARP choisit comme chemin améliorant un *plus court chemin* entre s et t.

(*a*) Considérant un réseau $G = (V, E, c, s, t)$, notons $G_1, \ldots, G_k$ les réseau résiduels successivement obtenus dans l'exécution de l'algorithme d'EDMONDS-KARP sur l'entrée G. Notons de plus $d_i(v)$, pour chaque sommet v, la distance de s à v dans G_i. Montrer que pour tout $i < k$ et tout $v \in V : d_i(v) \leq d_{i+1}(v)$.

(*b*) Au cours de l'algorithme, un arc uv peut apparaître et disparaître un certain nombre de fois dans le graphe résiduel. Montrer que ce nombre est majoré par $V/2$.

(*c*) En déduire que l'algorithme d'EDMONDS-KARP s'exécute en temps $O(VE^2)$.

Master 1 d'Informatique — Recherche Opérationnelle
2010 / 2011

Corrigé du TD n° 5

Flot maximum

Correction exercice 13.

(a) Pour $i < k$ fixé, on prouve $d_i(v) \leq d_{i+1}(v)$ par récurrence sur $d_{i+1}(v)$. Si $d_{i+1}(v) = 0$, alors $v = s$ et le résultat est clair puisque $d_j(s) = 0$ pour tout j. Sinon, soit u le prédécesseur de v dans un plus court chemin de s à v. (S'il n'y a pas de chemin entre s et v, alors $d_{i+1}(v) = \infty$ et le résultat est clair.) Alors $d_{i+1}(u) = d_{i+1}(v) - 1$ et donc, par hypothèse de récurrence : $d_i(u) \leq d_{i+1}(u)$. D'où

$$d_{i+1}(v) \geq d_i(u) + 1. \tag{2}$$

Deux cas de figure se présentent alors :

• Si $uv \in E_i$, alors $d_i(v) \leq d_i(u) + 1$ (puisque un pcc entre s et u suivit de l'arc uv constitue un chemin de longueur $d_i(u) + 1$ entre s et v). Il s'ensuit $d_{i+1}(v) \geq d_i(v)$ par (3).

• Si $uv \notin E_i$, alors nécessairement $vu \in E_i$, et même : vu est sur le chemin améliorant choisi par l'algorithme dans G_i pour augmenter le flot [1]. Ceci signifie, par définition de l'algorithme d'EDMONDS-KARP, que vu est sur un plus court chemin de s à u. Par conséquent, $d_i(v) = d_i(u) - 1$, d'où par (3), $d_{i+1}(v) \geq d_i(v) + 1$ et a fortiori : $d_{i+1}(v) \geq d_i(v)$.

(b) Supposons qu'un arc uv soit dans G_i et G_{j+1} $(i < j)$ mais dans aucun des réseaux résiduels $G_{i+1}, \ldots, G_j$. Alors, uv est dans le chemin augmentant choisi en G_i et donc $d_i(v) = d_i(u) + 1$. D'autre part, vu est dans le chemin augmentant choisi en G_j et donc $d_j(v) = d_j(u) - 1$. L'inégalité établie en (a) entraîne alors : $d_j(v) \geq d_i(v)$ d'où $d_j(u) - 1 \geq d_i(u) + 1$, c'est-à-dire : $d_j(u) \geq d_i(u) + 2$.

Ainsi, la distance de s à u s'accroît d'au moins deux unités à chaque réapparition de uv dans le graphe résiduel. Comme cette distance est majorée par $|V|$ (ou ∞), il s'ensuit que le nombre de disparitions de uv est majoré par $|V|/2$.

(c) Il s'ensuit que le nombre de disparitions d'arc au cours de l'exécution de l'algorihthme d'EDMONDS-KARP est majoré par $EV/2$. Comme il se produit au moins une telle disparition à chaque amélioration, cela signifie que le nombre d'amélioration est lui même majoré par $EV/2$. Or chaque amélioration coûte un temps $O(E)$. La complexité annoncée s'en déduit.

1. Notons que pour tout arc uv de G, un réseau résiduel de G contient au moins l'un des couples uv et vu. De plus, si uv n'est pas dans un G_i, cela implique que le flux est saturé entre u et v. Mais si cet arc apparaît dans G_{i+1}, c'est qu'entre temps le flux entre u et v a été modifié. Ceci impose que uv est sur le chemin améliorant issu de G_i.

Master 1 d'Informatique — Recherche Opérationnelle
2010 / 2011

TD n° 6

Révisions

Exercice 1 *(L'algorithme d'*EDMONDS-KARP*).* L'objectif de cet exercice est de prouver la complexité de l'algorithme d'EDMONDS-KARP, qui calcule un flot maximal en temps $O(VE^2)$. Cet algorithme ne se différencie de celui de FORD-FULKERSON que par l'heuristique suivante : dans chaque nouveau graphe résiduel, l'algorithme d'EDMONDS-KARP choisit comme chemin améliorant un *plus court chemin* entre s et t.

(a) Considérant un réseau $G = (V, E, c, s, t)$, notons $G_0, G_1, \ldots, G_k$ les réseau résiduels successivement obtenus dans l'exécution de l'algorithme d'EDMONDS-KARP sur l'entrée G. Notons de plus $d_i(v)$, pour chaque sommet v, la distance de s à v dans G_i. Montrer que pour tout $i < k$ et tout $v \in V$: $d_i(v) \leq d_{i+1}(v)$.

(b) Au cours de l'algorithme, un arc uv peut apparaître et disparaître un certain nombre de fois dans le graphe résiduel. Montrer que ce nombre est majoré par $V/2$.

(c) En déduire que l'algorithme d'EDMONDS-KARP s'exécute en temps $O(VE^2)$.

Exercice 2 *(Rendu de monnaie).* Un commerçant dispose d'une quantité illimitée de pièces de 1, 2, 5 et 10 euros. Il peut obtenir n'importe quelle somme $x \in \mathbb{N}$ à partir de ces pièces, mais cherche à minimiser le nombre total de pièces utilisées.

1. Proposez un algorithme glouton qui, prenant en argument une valeur $x \in \mathbb{N}$, retourne le nombre de pièces minimal $M(x)$ permettant d'obtenir x.
2. Prouvez la correction de votre algorithme.

On appelle *jeu de pièces* tout ensemble d'entiers $J = \{c_1, \ldots, c_n\}$ qui contient 1.

3. Adaptez votre algorithme pour qu'il opère sur un jeu de pièces quelconque $J = \{c_1, \ldots, c_n\}$. On supposera que $c_1 > c_2 > \cdots > c_n = 1$.
4. Montrez, sur un contre-exemple, que votre algorithme ne fonctionne pas pour le jeu $J = \{1, 4, 6\}$.

On cherche désormais à concevoir un algorithme qui calcule, *pour tout* jeu de pièces J, le nombre de pièces minimal $M_J(x)$ permettant de fabriquer x. On fixe $x \in \mathbb{N}$ et on considère un $c \in J$ tel que $c \leq x$.

5. Montrer que $M_J(x) \leq 1 + M_J(x - c)$.
6. Montrer que $M_J(x) = 1 + M_J(x - c)$ si, et seulement si, c apparaît dans une solution optimale pour x.

Notons a le plus grand entier tel que $a \in J$ et $a \leq x$.

7. Déduire de ce qui précède : $M_J(x) = 1 + \min_{c \in J, c \leq a} M_J(x-c)$.
8. En déduire un algorithme de programmation dynamique qui, prenant en entrée un jeu de pièces J et un montant x, retourne $M_J(x)$.
9. Évaluer sa complexité.
10. Faites tourner votre algorithme sur l'entrée $x = 13$, $J = \{1,4,6\}$.

Master 1 d'Informatique — Recherche Opérationnelle
2010 / 2011

Corrigé du TD n° 6

Révisions

Correction exercice 1.

(a) Pour $i < k$ fixé, on prouve $d_i(v) \leq d_{i+1}(v)$ par récurrence sur $d_{i+1}(v)$. Si $d_{i+1}(v) = 0$, alors $v = s$ et le résultat est clair puisque $d_j(s) = 0$ pour tout j. Sinon, soit u le prédécesseur de v dans un plus court chemin de s à v. (S'il n'y a pas de chemin entre s et v, alors $d_{i+1}(v) = \infty$ et le résultat est clair.) Alors $d_{i+1}(u) = d_{i+1}(v) - 1$ et donc, par hypothèse de récurrence : $d_i(u) \leq d_{i+1}(u)$. D'où

$$d_{i+1}(v) \geq d_i(u) + 1. \tag{3}$$

Deux cas de figure se présentent alors :

• Si $uv \in E_i$, alors $d_i(v) \leq d_i(u) + 1$ (puisque un pcc entre s et u suivit de l'arc uv constitue un chemin de longueur $d_i(u) + 1$ entre s et v). Il s'ensuit $d_{i+1}(v) \geq d_i(v)$ par (3).

• Si $uv \notin E_i$, alors nécessairement $vu \in E_i$, et même : vu est sur le chemin améliorant choisi par l'algorithme dans G_i pour augmenter le flot [2]. Ceci signifie, par définition de l'algorithme d'EDMONDS-KARP, que vu est sur un plus court chemin de s à u. Par conséquent, $d_i(v) = d_i(u) - 1$, d'où par (3), $d_{i+1}(v) \geq d_i(v) + 1$ et a fortiori : $d_{i+1}(v) \geq d_i(v)$.

(b) Supposons qu'un arc uv soit dans G_i et G_{j+1} $(i < j)$ mais dans aucun des réseaux résiduels $G_{i+1}, \ldots, G_j$. Alors, uv est dans le chemin augmentant choisi en G_i et donc $d_i(v) = d_i(u) + 1$. D'autre part, vu est dans le chemin augmentant choisi en G_j et donc $d_j(v) = d_j(u) - 1$. L'inégalité établie en (a) entraîne alors : $d_j(v) \geq d_i(v)$ d'où $d_j(u) - 1 \geq d_i(u) + 1$, c'est-à-dire : $d_j(u) \geq d_i(u) + 2$.

Ainsi, la distance de s à u s'accroît d'au moins deux unités à chaque réapparition de uv dans le graphe résiduel. Comme cette distance est majorée par $|V|$ (ou ∞), il s'ensuit que le nombre de disparitions de uv est majoré par $|V|/2$.

(c) Il s'ensuit que le nombre de disparitions d'arc au cours de l'exécution de l'algorihthme d'EDMONDS-KARP est majoré par $EV/2$. Comme il se produit au moins une telle disparition à chaque amélioration, cela signifie que le nombre d'amélioration est lui même majoré par $EV/2$. Or chaque amélioration coûte un temps $O(E)$. La complexité annoncée s'en déduit.

2. Notons que pour tout arc uv de G, un réseau résiduel de G contient au moins l'un des couples uv et vu. De plus, si uv n'est pas dans un G_i, cela implique que le flux est saturé entre u et v. Mais si cet arc apparaît dans G_{i+1}, c'est qu'entre temps le flux entre u et v a été modifié. Ceci impose que uv est sur le chemin améliorant issu de G_i.

Correction exercice 2.

1. Algorithme glouton pour le calcul de $M(x)$:

```
   Données : Un entier x ∈ ℕ
   Résultat : M(x)
1  res = 0 ;
2  pour c = 10,5,2,1 pris dans cet ordre faire
3  |  tant que x − c ≥ 0 faire
4  |  |  x = x − c ;
5  |  |  res++ ;
   |  fin
   fin
6  retourner res
```

2. Prouvez la correction de votre algorithme.
3. Adaptation de l'algorithme précédent à un jeu de pièces quelconque $J = \{c_1, \ldots, c_n\}$, avec $c_1 > \cdots > c_n = 1$.

```
   Données : Un entier x ∈ ℕ
   Résultat : M_J(x)
1  res = 0 ;
2  pour i = 1 à n faire
3  |  tant que x − c_i ≥ 0 faire
4  |  |  x = x − c_i ;
5  |  |  res++ ;
   |  fin
   fin
6  retourner res
```

4. Pour le jeu $J = \{1,4,6\}$ et l'entrée $x = 8$, l'algo précédent renvoie 3 (puisque $8 = 6+1+1$) alors que l'optimum est 2 ($8 = 4+4$).

On fixe $x \in \mathbb{N}$ et on considère un $c \in J$ tel que $c \leq x$.

5. Si l'on peut construire $x - c$ avec k pièces de J, alors on peut construire x avec $k+1$ pièces de J (en rajoutant une pièce de valeur c aux k pièces précédentes). Ainsi, le nombre minimal de pièces de J nécessaire pour construire x est inférieur à $k+1$. En particulier, en prenant pour k le nombre $M_J(x-c)$, on obtient la relation cherchée : $M_J(x) \leq 1 + M_J(x-c)$.
6. Montrer que $M_J(x) = 1 + M_J(x-c)$ si, et seulement si, c apparaît dans une solution optimale pour x.

   $\boxed{\Leftarrow}$ Si c apparaît dans une solution optimale pour x, alors l'ensemble de pièces obtenu en retirant de cette solution une pièce de valeur c est une solution optimale pour $x-c$. D'où $M_J(x-c) = M_J(x) - 1$.

   $\boxed{\Rightarrow}$ Si $M_J(x) = 1 + M_J(x-c)$, notons $a_1, \ldots, a_k$ une solution optimale pour $x-c$. En d'autres termes, supposons que :

– $k = M_J(x-c)$;
– les a_i sont des éléments non nécessairement distincts de J ;
– $x - c = a_1 + \cdots + a_k$.
Alors $a_1, \ldots, a_k, c$ est une solution pour x puisque $x = a_1 + \cdots + a_k + c$ et puisque c, comme chaque a_i, est dans J. Cette solution étant de taille $k+1$, il s'ensuit que $M_J(x) \leq k+1$, c'est-à-dire, $M_J(x) \leq 1 + M_J(x-c)$. L'inégalité inverse, prouvée dans la question précédente, permet de conclure.

7. Soit $\mathscr{E}_x$ l'ensemble des entiers de la forme $1 + M_J(x-c)$ obtenus pour les $c \in J$ inférieurs à x. Autrement dit :
$$\mathscr{E}_x = \{1 + M_J(x-c) \text{ pour } c \in J, c \leq x\}.$$
La question 5 se traduit facilement en : $M_J(x)$ minore $\mathscr{E}_x$ (*i.e.*, est inférieur à chaque élément de $\mathscr{E}_x$). La question 6 affirme pour sa part que tout c apparaissant dans une solution optimale pour x réalise l'égalité $M_J(x) = 1 + M_J(x-c)$. Ceci entraîne notamment que l'un des entiers de $\mathscr{E}_x$ au moins a pour valeur $M_J(x)$. Ainsi, $M_J(x)$ minore l'ensemble $\mathscr{E}_x$ et appartient à cet ensemble. Cela s'écrit $M_J(x) = \min \mathscr{E}_x$, ou encore
$$\begin{aligned} M_J(x) &= \min\{1 + M_J(x-c) \text{ pour } c \in J, c \leq x\} \\ &= 1 + \min\{M_J(x-c) \text{ pour } c \in J, c \leq x\} \end{aligned}$$
C'est l'égalité cherchée.

8. En déduire un algorithme de programmation dynamique qui, prenant en entrée un jeu de pièces J et un montant x, retourne $M_J(x)$.

Données : Un entier $x \in \mathbb{N}$; un jeu de pièces J
Résultat : $M_J(x)$
$M[0] = 0$;
pour $y = 1$ *à* x **faire**
  $M[y] = 1 + \min_{c \in J, c \leq y} M[y-c]$;
fin
retourner $M[x]$

---

9. Complexité : $O(x|J|)$.
10. Simulation de l'algorithme sur l'entrée $x = 13$, $J = \{1,4,6\}$:

0	1	2	3	1	2	1	2	2	3	2	3	2	**3**

D'où : $M_{\{1,4,6\}}(13) = 3$.