Chapitre 4

Graphes

Ce chapitre commence par la définition des graphes et de certains objets fondamentaux comme les chemins, les chaînes, les circuits et les cycles. Pour les graphes sans circuit, sont ensuite introduites les notions de rang et de liste topologique. La section 2 présente l'algorithme de Roy-Warshall dans le cadre général de la recherche des valeurs optimales des chemins entre tous les couples de sommets d'un graphe dont les arcs sont valués par les éléments d'un semi-anneau. On examine ensuite certains cas particuliers comme le calcul de la relation d'accessibilité, l'algorithme de Floyd et le calcul du langage des chemins d'un graphe étiqueté. La section 3 définit les arbres, les arborescences et leurs principales variantes attribuées. La section 4 introduit la notion de parcours d'un graphe non orienté et présente des algorithmes spécifiques pour les parcours en profondeur et en largeur. Les parcours sont ensuite étendus aux graphes orientés et l'algorithme de Tarjan, qui détermine les composantes fortement connexes d'un graphe, est présenté en tant qu'application des propriétés des parcours en profondeur. On termine par l'étude des parcours spécifiques aux arborescences.

Introduction

Les graphes constituent certainement l'outil théorique le plus utilisé pour la modélisation et la recherche des propriétés d'ensembles structurés. Ils interviennent chaque fois que l'on veut représenter et étudier un ensemble de liaisons (orientées ou non) entre les éléments d'un ensemble fini d'objets. Citons comme cas particuliers de liaisons des applications aussi diverses qu'une connexion entre deux processeurs d'un réseau informatique, une contrainte de précédence entre deux tâches d'un problème d'ordonnancement, la possibilité pour un système de passer d'un état à un autre ou encore une incompatibilité d'horaires. S'agissant d'un outil fondamental, la recherche des algorithmes les plus efficaces pour réaliser les opérations de base sur les graphes constitue un objectif essentiel de l'algorithmique.

4.1 Définitions et propriétés élémentaires

4.1.1 Définitions

Un *graphe orienté* $G = (S, A)$ est composé d'un ensemble *fini* S d'éléments appelés *sommets* et d'une partie A de $S \times S$ dont les éléments sont appelés *arcs*. Les arcs d'un graphe orienté constituent donc une relation sur l'ensemble de ses sommets. Un arc (x, y) représente une liaison *orientée* entre l'origine x et l'extrémité y. Si (x, y) est un arc, y est un *successeur* de x, x est un *prédécesseur* de y et si $x = y$, l'arc (x, x) est appelé *boucle*.

Lorsque l'orientation des liaisons n'est pas une information pertinente, la notion de *graphe non orienté* permet de s'en affranchir. Un *graphe non orienté* $G = (S, A)$ est composé d'un ensemble *fini* S d'éléments appelés *sommets* et d'une famille de *paires* de S dont les éléments sont appelés *arêtes*. Etant donné un graphe orienté, sa *version non orientée* est obtenue en supprimant les boucles et en substituant à chaque arc restant (x, y) la paire $\{x, y\}$.

Deux sommets distincts d'un graphe orienté (respectivement non orienté) G sont *adjacents* s'ils sont les extrémités d'un même arc (respectivement d'une même arête). Soient $G = (S, A)$ un graphe orienté (respectivement non orienté) et T un sous-ensemble de S. L'ensemble noté $\omega(T)$ des arcs (respectivement arêtes) de A dont une extrémité est dans T et l'autre dans $S - T$ est appelé le *cocycle* associé à T. L'ensemble noté $\mathcal{B}(T)$ des sommets de $S - T$ adjacents à au moins un sommet de T est appelé *bordure* de T. Si le sommet u appartient à $\mathcal{B}(T)$, on dit aussi que u est adjacent à T. Le graphe $G = (S, A)$ est dit *biparti* s'il existe un sous-ensemble de sommets T tel que $A = \omega(T)$.

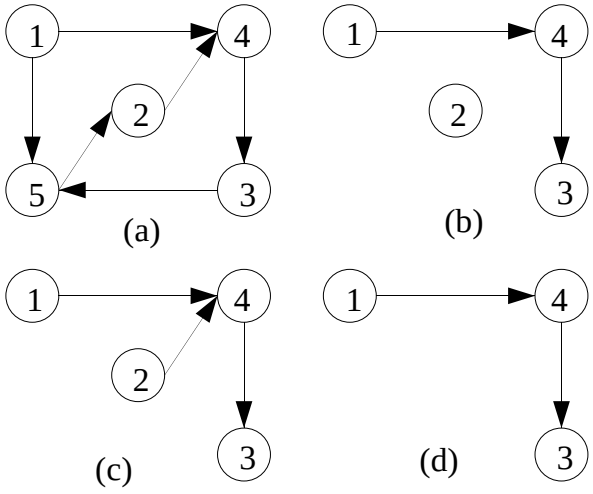

Figure 1.1: *Sous-graphes.*

Un *sous-graphe* du graphe $G = (S, A)$ est un couple $G' = (S', A')$ pour lequel $S' \subset S$ et $A' \subset A$. Le sous-graphe G' est un graphe *partiel* de G si $S' = S$. Si A'

est l'ensemble des arcs de A dont les deux extrémités sont dans S', le sous-graphe G' est dit *induit* par S'. Si S' est l'ensemble des extrémités des arcs de A', le sous-graphe G' est dit *induit* par A'.

Les sommets d'un graphe sont représentés par des points distincts du plan. Un arc (x, y) d'un graphe orienté est représenté par une flèche d'origine x et d'extrémité y, une arête d'un graphe non orienté est représentée par une ligne joignant ses deux extrémités. Sur la figure 1.1 sont représentés en a) un graphe orienté G, en b) un sous-graphe de G, en c) le graphe partiel de G induit par les sommets $\{1, 2, 3, 4\}$ et en d) le sous-graphe induit par les arcs $\{(1, 4), (4, 3)\}$. Le graphe G est biparti car tout arc est incident à $T = \{1, 2, 3\}$.

4.1.2 Implémentations d'un graphe

Parmi les implémentations possibles d'un graphe orienté $G = (S, A)$, on peut en retenir essentiellement trois qui interviennent dans la plupart des problèmes théoriques et pratiques : la matrice d'adjacence, la matrice d'incidence et la liste des successeurs.

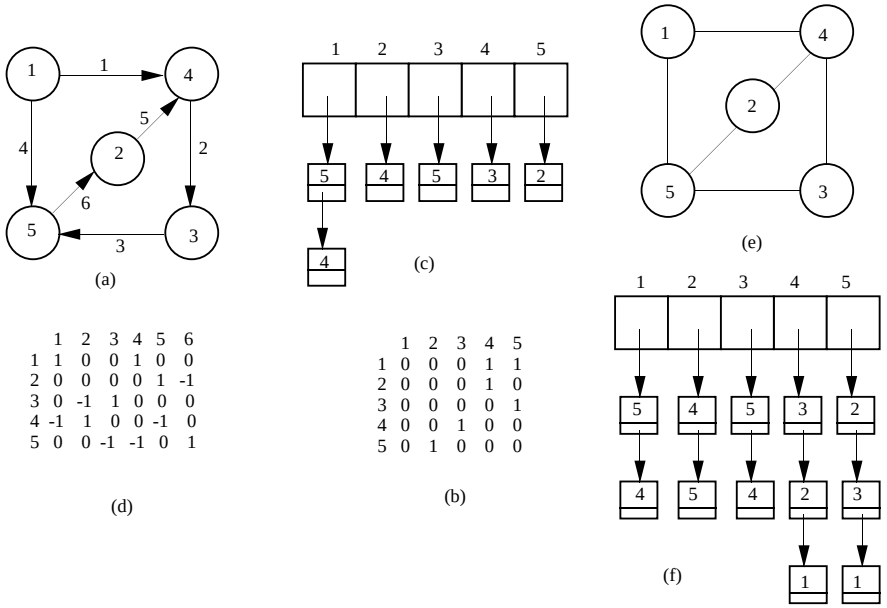

Figure 1.2: *Implémentations d'un graphe.*

La *matrice d'adjacence* $M = M(G)$ est une matrice carrée, indicée par S, définie par :

$$m_{ij} = \begin{cases} 1 & \text{si } (i,j) \in A \\ 0 & \text{sinon} \end{cases}$$

où le 0 et le 1 seront entiers ou booléens selon la convenance.

Si G est un graphe sans boucles, sa *matrice d'incidence* «sommets-arcs» $\Delta(G)$ est définie par :

$$\delta_{xa} = \begin{cases} 1 & \text{si } x \text{ est l'origine de l'arc } a \\ -1 & \text{si } x \text{ est l'extrémité de l'arc } a \\ 0 & \text{sinon} \end{cases}$$

Cette matrice possède une propriété très importante pour certains domaines de l'optimisation combinatoire comme la théorie des flots (voir chapitre 8) : le déterminant de toute sous-matrice carrée extraite vaut 0, 1 ou -1. Une matrice satisfaisant cette propriété est dite *totalement unimodulaire*.

Les matrices d'ajacence et d'incidence du graphe orienté de la figure 1.2 sont les suivantes :

$$M = \begin{pmatrix} 0&0&0&1&1 \\ 0&0&0&1&0 \\ 0&0&0&0&1 \\ 0&0&1&0&0 \\ 0&1&0&0&0 \end{pmatrix} \quad \Delta = \begin{pmatrix} 1&0&0&1&0&0 \\ 0&0&0&0&1&-1 \\ 0&-1&1&0&0&0 \\ -1&1&0&0&-1&0 \\ 0&0&-1&-1&0&1 \end{pmatrix}.$$

La *liste des successeurs* (en anglais : adjacency list) est un tableau $(q_1, \ldots, q_n)$ où q_i est un pointeur sur une liste des successeurs du sommet i.

Dans le cas d'un graphe non orienté, l'implémentation la plus utilisée est la *liste des voisins* qui est un tableau $(q_1, \ldots, q_n)$ où q_i est un pointeur sur une liste des sommets adjacents au sommet i. C'est la version non orientée de la liste des successeurs.

Notons que les tailles de ces diverses représentations sont sensiblement différentes. Si n est le nombre de sommets et si m est le nombre d'arcs (ou d'arêtes pour un graphe non orienté), la taille de la matrice d'adjacence est en $\theta(n^2)$, celle de la matrice d'incidence en $\theta(nm)$ alors que la taille de la liste des successeurs (ou des voisins) est en $\theta(n+m)$.

La figure 1.2 montre la liste des successeurs (c) du graphe orienté (a) et la liste des voisins (d) du graphe non orienté (b).

4.1.3 Chemins, chaînes, circuits, cycles

Soit $G = (S, A)$ un graphe orienté.

Un *chemin* d'origine x et d'extrémité y est une suite finie non vide de sommets $c = (s_0, \ldots, s_p)$ telle que : $s_0 = x$, $s_p = y$ et pour $k = 0, .., p-1$, $(s_k, s_{k+1}) \in A$. La *longueur* du chemin c est p, c'est le nombre d'arcs (non nécessairement distincts) empruntés par ce chemin.

La notion de chemin nous permet d'introduire les *ascendants* et les *descendants* d'un sommet. Soit x un sommet de S, un sommet y est un ascendant (respectivement descendant) de x s'il existe un chemin de y à x (respectivement de x à y)

dans G. Le sommet y est un ascendant (respectivement descendant) *propre* du sommet x s'il existe un chemin de longueur non nulle de y à x (respectivement de x à y). Notons qu'un sommet peut être ascendant propre et descendant propre de lui-même.

Un chemin $c = (s_0, \ldots, s_p)$ est *simple* si les arcs $(s_{i-1}, s_i), i = 1, .., p$ sont deux à deux distincts. Un chemin $c = (s_0, \ldots, s_p)$ est *élémentaire* si ses sommets sont distincts deux à deux. Un chemin $(s_0, \ldots, s_p)$ est un *circuit* si $p \geq 1$ et $s_0 = s_p$. Un circuit $c = (s_0, \ldots, s_p)$ est *élémentaire* si le chemin $(s_0, \ldots, s_{p-1})$ est élémentaire. Soulignons qu'un circuit élémentaire n'est pas un chemin élémentaire et que, de la même façon, un chemin élémentaire n'est pas un circuit élémentaire.

Considérons le graphe orienté de la figure 1.2. La suite $(5, 2, 4, 3, 5)$ est un circuit élémentaire mais n'est pas un chemin élémentaire. Par contre la suite $(1, 4, 3)$ est un chemin élémentaire.

Considérons maintenant un graphe non orienté $G = (S, A)$ et définissons les notions correspondantes de chaîne et cycle. Une *chaîne* d'origine x et d'extrémité y est une suite finie de sommets $(s_0, \ldots, s_p)$ telle que $s_0 = x$, $s_p = y$, deux sommets consécutifs quelconques de la liste sont les extrémités d'une arête, et ces arêtes sont *distinctes* deux à deux. Notons que si $(s_0, \ldots, s_p)$ est une chaîne, il en est de même pour $(s_p, \ldots, s_0)$.

Une chaîne $c = (s_0, \ldots, s_p)$ est *élémentaire* si les sommets de la liste c sont deux à deux distincts. Une chaîne $(s_0, \ldots, s_p)$ est un *cycle* si $s_0 = s_p$. Un cycle $(s_0, \ldots, s_p)$ est *élémentaire* si $p \geq 1$ et si la liste $(s_0, \ldots, s_{p-1})$ est une chaîne élémentaire.

Pour le graphe non orienté représenté sur la figure 1.2, la liste $(5, 1, 4, 1)$ n'est pas une chaîne et $(1, 5, 3, 4)$ est une chaîne élémentaire.

Un graphe non orienté est *connexe* si pour tout couple de sommets, il existe une chaîne ayant ces deux sommets comme extrémités. Par extension, un graphe orienté est connexe si sa version non orientée (c'est-à-dire le graphe non orienté obtenu en supprimant les orientations et les boucles) est connexe. La relation sur l'ensemble des sommets d'un graphe non orienté définie par $x \sim y$ s'il existe une chaîne de x à y est une relation d'équivalence dont les classes sont appelées *composantes connexes* du graphe. Les graphes induits par les composantes connexes sont bien sûr des graphes connexes.

4.1.4 Lemme de König

Les chemins d'un graphe constituent en général un ensemble infini (il faut et il suffit que le graphe possède des circuits). Par contre les chemins élémentaires sont en nombre fini et constituent pour de nombreux problèmes d'optimisation un ensemble *dominant* lorsque l'existence d'un chemin optimal élémentaire est prouvée. Le lemme de König montre que l'existence d'un chemin d'origine x et d'extrémité y entraîne celle d'un chemin élémentaire entre ces mêmes sommets.

Soit $G = (S, A)$ un graphe orienté. Un chemin c est dit *extrait* d'un chemin c' si les deux chemins c et c' ont la même origine et la même extrémité et si la suite des arcs de c est une sous-suite de celle des arcs de c'. La relation «est extrait de» définie sur l'ensemble des chemins de G est une relation d'ordre et le lemme de König ci-dessous montre que ses éléments minimaux sont les chemins élémentaires de G.

Lemme 1.1 (de König)**.** *De tout chemin on peut extraire un chemin élémentaire.*

Preuve. Nous raisonnons par induction sur la longueur p du chemin c. Si $p = 0$, c est élémentaire. Soit donc $c = (x_0, \ldots, x_p)$ un chemin à p sommets avec $p > 0$. Si c n'est pas élémentaire, il existe un couple (r, s) tel que $0 \leq r < s \leq p$ et $x_r = x_s$. Le chemin $c' = (x_0, .., x_r, x_{s+1}, .., x_p)$ est extrait de c et strictement plus petit que c. On peut donc extraire de c', et donc de c, un chemin élémentaire. ∎

Remarque. Le lemme de König s'étend aux circuits d'un graphe orienté et aussi aux chaînes et aux cycles d'un graphe non orienté.

4.1.5 Graphes sans circuit

Les graphes sans circuit interviennent dans de nombreux domaines, en particulier chaque fois que l'on étudie une relation d'ordre partiel. Pour ces graphes, on connaît des algorithmes spécifiques dont l'efficacité est souvent due à l'utilisation d'une liste des sommets qui permet, lors de la visite d'un sommet de la liste, d'être sûr que tous ses ascendants propres ont déjà été visités. Une *liste topologique* des sommets d'un graphe $G = (S, A)$ est une permutation $(s_1, \ldots, s_n)$ des sommets de S telle que pour tout arc (s_i, s_j) on a $i < j$.

Proposition 1.2. *Un graphe orienté $G = (S, A)$ est sans circuit si et seulement s'il existe une liste topologique des sommets de G.*

Preuve. La condition suffisante résulte directement de la définition d'une liste topologique. Démontrons que la condition est nécessaire par induction sur $n = \text{Card}(S)$. La propriété est bien sûr vraie si $n = 1$. Supposons qu'elle le soit pour un graphe à $n - 1$ sommets ($n \geq 2$). Le graphe G ne possédant pas de circuits, il existe au moins un sommet s sans descendants propres car dans le cas contraire, on pourrait construire un chemin infini, donc contenant nécessairement un circuit. Le sous-graphe G' de G induit par $S - \{s\}$ a $n - 1$ sommets et ne possède pas de circuits. Il existe donc une liste topologique L' des sommets de G' ; la liste $L'.(s)$ formée de L' concaténée avec la liste (s) est alors une liste topologique des sommets de G. ∎

La propriété précédente conduit naturellement à un algorithme pour tester si un graphe orienté G est sans circuit. Le principe de cet algorithme est de rechercher

une sortie s de G. S'il n'en n'existe pas alors G possède un circuit sinon la réponse pour G est la même que pour G_s. En utilisant une liste des successeurs pour coder G, on obtient une complexité $O(n^2)$. Nous donnerons dans la section 4.4.4 un algorithme plus efficace, de complexité $O(n+m)$, fondé sur un parcours en profondeur du graphe G.

Il peut exister plusieurs listes topologiques d'un même graphe sans circuit. Le graphe représenté sur la figure 1.3 ne possède qu'une seule liste topologique $(3, 2, 1, 6, 7, 5, 4)$ car il existe un seul chemin élémentaire passant par tous les sommets.

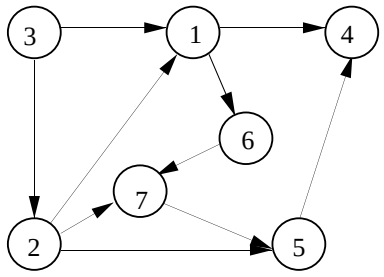

Figure 1.3: *Un graphe sans circuit.*

Plus généralement, soit $\leq$ une relation d'ordre sur un ensemble fini E. Une *extension linéaire* de $\leq$ est un ordre total $\preceq$ sur E qui contient $\leq$, c'est-à-dire tel que :

$$x \leq y \Longrightarrow x \preceq y.$$

Etant donné un ordre total $\preceq$ sur S, on peut constituer la suite $(s_1, \ldots, s_n)$ des éléments de S telle que :

$$s_1 \preceq s_2 \preceq \ldots \preceq s_n.$$

L'opération qui détermine une extension linéaire s'appelle le *tri topologique* et le résultat du tri est une liste topologique. Déterminer une extension linéaire équivaut bien entendu à *numéroter* les sommets du graphe, c'est-à-dire à trouver une bijection $\nu : S \mapsto \{1, \ldots, n\}$ vérifiant :

$$(x, y) \in A \Longrightarrow \nu(x) < \nu(y).$$

Soit $G = (S, A)$ un graphe sans circuit. Le *rang* du sommet s, noté $\rho(s)$, est la longueur maximale d'un chemin d'extrémité s. Notons que $\rho(s)$ est bien défini puisque les chemins de G sont élémentaires et donc en nombre fini. La figure 1.4 montre un graphe sans circuit et à sa droite le même graphe rangé.

Proposition 1.3. *Pour tout sommet x, les ascendants propres de x ont un rang strictement inférieur au rang de x, les descendants propres de x ont un rang strictement supérieur au rang de x, et si x est de rang $k > 0$, il possède un ascendant propre de rang $k-1$.*

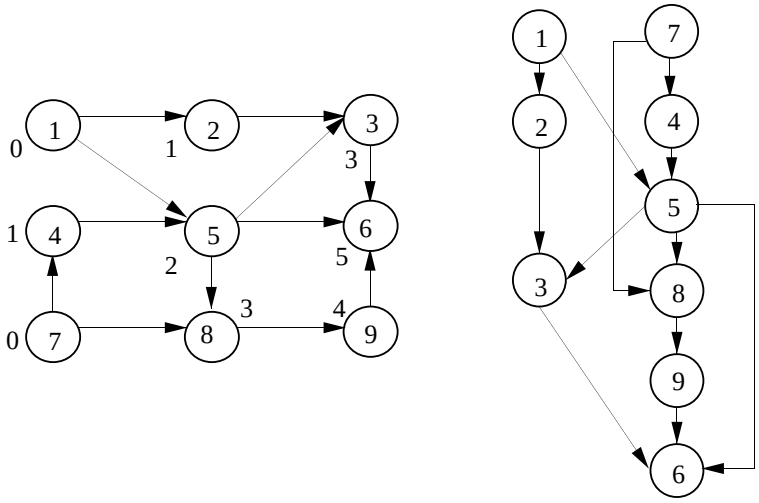

Figure 1.4: *Classement par rang.*

Preuve. Si y est un ascendant propre de x, alors $\rho(x) > \rho(y)$. De même, si y est un descendant propre de x, alors $\rho(x) < \rho(y)$. Enfin si tous les ascendants propres de x étaient de rang inférieur ou égal à $k-2$, on aurait : $\rho(x) \leq k-1$, d'où la contradiction. ∎

Remarque. La liste des sommets d'un graphe sans circuit, ordonnée par rang croissant au sens large, est une liste topologique.

4.2 Accessibilité

Etant donné un graphe orienté $G = (S, A)$, le sommet y est dit *accessible* à partir du sommet x s'il existe un chemin de x à y. L'objet de cette section est le calcul de la relation d'accessibilité. A cet effet, nous décrivons l'algorithme de Roy-Warshall qui fournit la matrice booléenne de la relation d'accessibilité. Cet algorithme qui utilise comme structure de données la matrice booléenne associée au graphe initial a une complexité $O(n^3)$, où n est le nombre de sommets de G.

Nous posons $S = \{1, 2, \cdots, n\}$ et pour $k \in S$, nous notons $E(k)$ l'ensemble $\{1, 2, \cdots, k\}$; par convention l'ensemble $E(0)$ est vide. Si $c = (v_1, \ldots, v_k)$ est un chemin de G, l'intérieur $I(c)$ de ce chemin est l'*ensemble* des sommets du sous-chemin $(v_2, \ldots, v_{k-1})$; si $k \leq 2$, $I(c)$ est l'ensemble vide. Enfin nous notons $G_k = (S, A_k)$ le graphe défini par :

$$(i, j) \in A_k \Longleftrightarrow \exists c : i \to j, \quad I(c) \subset E(k).$$

Remarque. Le graphe G_0 est le graphe initial G complété d'une boucle en chaque sommet; le graphe G_n est le graphe de la relation d'accessibilité.

4.2.1 Algorithme de Roy-Warshall

L'algorithme de Roy-Warshall calcule la suite des graphes G_1, $G_2, \ldots, G_n$ en utilisant le théorème suivant :

Théorème 2.1. *Pour tout $i, j, k \in S$, l'arc (i, j) appartient à A_k si et seulement si $(i, j) \in A_{k-1}$ ou $((i, k) \in A_{k-1}$ et $(k, j) \in A_{k-1})$.*

Preuve. Si $(i, j) \in A_k$, il existe d'après le lemme de König un chemin élémentaire c de i à j dont l'intérieur $I(c)$ est inclus dans $E(k)$. Si $I(c)$ ne contient pas k, alors $I(c)$ est inclus dans $E(k-1)$ et donc $(i, j) \in A_{k-1}$. Si $I(c)$ contient k, nous notons c' (respectivement c'') le sous-chemin de c de i à k (respectivement le sous-chemin de c de k à j). Le chemin c étant élémentaire, $I(c')$ et $I(c'')$ sont inclus dans $E(k-1)$; il en résulte que $(i, k) \in A_{k-1}$ et $(k, j) \in A_{k-1}$.

Réciproquement on a : $(i, j) \in A_{k-1} \Longrightarrow (i, j) \in A_k$, car $E(k-1) \subset E(k)$. Supposons maintenant que $(i, k) \in A_{k-1}$ et $(k, j) \in A_{k-1}$; il existe alors un chemin c' (respectivement c'') de i à k (respectivement de k à j) tel que $I(c')$ (respectivement $I(c'')$) soit inclus dans $E(k-1)$. L'intérieur du chemin c obtenu par la concaténation de c' et c'' est donc inclus dans $E(k)$ et par conséquent $(i, j) \in A_k$. ∎

Une implémentation possible de l'algorithme de Roy-Warshall utilise comme structure de données une seule matrice booléenne, notée a et initialisée avec la matrice d'adjacence de G ; cette implémentation est donnée par la procédure ROY-WARSHALL(G) suivante :

```
procédure ROY-WARSHALL(G);
  {g est la matrice d'adjacence du graphe G}
  {a est la matrice de travail}
  a := g;
  pour i de 1 à n faire a_ii := 1;
  pour k de 1 à n faire
    pour i de 1 à n faire
      pour j de 1 à n faire
        a_ij := a_ij ou (a_ik et a_kj);
  retourner(a).
```

L'utilisation d'une seule matrice de travail rend nécessaire la proposition suivante pour valider la procédure ROY-WARSHALL(G).

Proposition 2.2. *Pour tout $i, j, k \in S$, les deux équivalences suivantes sont vraies :*

(1) $(i, k) \in A_{k-1} \Longleftrightarrow (i, k) \in A_k$;

(2) $(k,j) \in A_{k-1} \Longleftrightarrow (k,j) \in A_k$.

Preuve. (laissée à titre d'exercice). ∎

Théorème 2.3. *La procédure* ROY-WARSHALL(G) *détermine la matrice de la relation d'accessibilité d'un graphe orienté $G = (S, A)$ en $O(n^3)$ opérations élémentaires.*

Preuve. Notons $a^{(k)}$ la valeur de la matrice a calculée par la procédure ROY-WARSHALL(G) lors de l'itération k et supposons que jusqu'au calcul de l'élément (i,j) de a à l'itération k, les valeurs $a_{rs}^{(p)}$ soient correctes. Lors du calcul de $a_{ij}^{(k)}$, l'élément (i,j) de la matrice a contient initialement la valeur $a_{ij}^{(k-1)}$, par contre l'élément (i,k) de la matrice a contient la valeur $a_{ik}^{(k-1)}$ si $k > j$ ou la valeur $a_{ik}^{(k)}$ si $k < j$; de même l'élément (k,j) de la matrice a contient la valeur $a_{kj}^{(k-1)}$ si $k > i$ ou la valeur $a_{ik}^{(k)}$ si $k < i$. Cependant la proposition précédente nous assure que la valeur calculée $a_{ij}^{(k)}$ est correcte. ∎

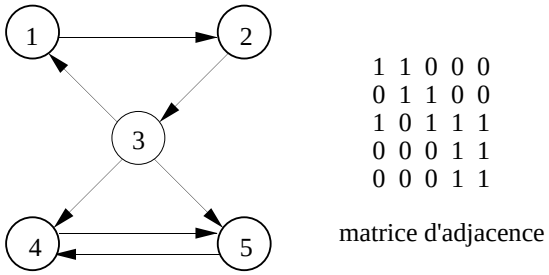

Figure 2.1: *Un graphe orienté et sa matrice d'adjacence.*

Exemple. Considérons le graphe orienté de la figure 2.1. Les matrices d'adjacence des graphes G_1, G_2 et G_3 sont les suivantes :

$$A_1 = \begin{pmatrix} 1&1&0&0&0 \\ 0&1&1&0&0 \\ 1&1&1&1&1 \\ 0&0&0&1&1 \\ 0&0&0&1&1 \end{pmatrix} \quad A_2 = \begin{pmatrix} 1&1&1&0&0 \\ 0&1&1&0&0 \\ 1&1&1&1&1 \\ 0&0&0&1&1 \\ 0&0&0&1&1 \end{pmatrix} \quad A_3 = \begin{pmatrix} 1&1&1&1&1 \\ 1&1&1&1&1 \\ 1&1&1&1&1 \\ 0&0&0&1&1 \\ 0&0&0&1&1 \end{pmatrix}$$

Les graphes G_4 et G_5 sont égaux au graphe G_3, si bien que la matrice de la relation d'accessibilité est celle du graphe G_3.

4.2.2 Autres problèmes d'accessibilité

Nous considérons dans ce paragraphe la notion de graphe étiqueté, c'est-à-dire de graphe dont chaque arc (i,j) porte une étiquette e_{ij} choisie dans un ensemble $\mathcal{E}$.

L'idée centrale de l'algorithme de Roy-Warshall est de résoudre la suite des sous-problèmes obtenus en faisant « grossir » de l'ensemble vide jusqu'à S lui-même, l'ensemble des sommets qui peuvent appartenir à l'intérieur d'un chemin. Une formule de récurrence permet de passer simplement de la solution d'un sous-problème à celle du sous-problème suivant. Cette même idée conduit à la résolution d'une classe de problèmes analogues lorsque l'ensemble des étiquettes possède une structure de *semi-anneau*. Nous illustrons cette méthodologie en traitant deux problèmes classiques sur les graphes étiquetés :

a) Si $\mathcal{E} = \mathbb{R}$, déterminer pour tout couple de sommets la valeur maximale des chemins entre ces deux sommets ;
b) Si $\mathcal{E}$ est l'ensemble des parties non vides d'un alphabet A, déterminer pour tout couple de sommets le langage des étiquettes des chemins entre ces deux sommets.

Chemins de valeur maximale : algorithme de Floyd

Soit $G = (S, A)$ un graphe orienté dont chaque arc est valué par un nombre réel. Nous appelons *valeur* d'un chemin la somme des étiquettes des arcs de ce chemin. Nous supposons de plus que tout circuit du graphe a une valeur négative ou nulle (on dit encore qu'il n'existe pas de *circuits absorbants*). Le problème consiste alors à déterminer, pour tout couple de sommets (i, j) la valeur maximale des chemins de i à j, notée a_{ij}.

Etant donné un couple (i, j), le lemme de König et l'absence de circuits positifs montrent qu'il suffit de calculer la valeur maximale des chemins élémentaires de i à j et donc que a_{ij} est fini. La recherche sera donc conduite dans l'ensemble des chemins élémentaires de G.

Nous notons $a_{ij}^{(k)}$ la valeur maximale d'un chemin de i à j dont l'intérieur est inclus dans $E(k)$ et nous prenons comme valeurs initiales, pour $k = 0$:

$$a_{ij}^{(0)} = \begin{cases} e_{ij}, & \text{si } (i,j) \in A \text{ et } i \neq j\,; \\ 0, & \text{si } i = j\,; \\ -\infty, & \text{sinon.} \end{cases}$$

La proposition suivante permet alors le calcul des $a_{ij}^{(k)}$:

Proposition 2.4. *Pour tout $i, j, k \in S$, on a $a_{ij}^{(k)} = \max\{a_{ij}^{(k-1)}, a_{ik}^{(k-1)} + a_{kj}^{(k-1)}\}$.*

Preuve. La valeur maximale d'un chemin de i à j dont l'intérieur est inclus dans $E(k-1)$ est par définition $a_{ij}^{(k-1)}$.

La valeur maximale d'un chemin de i à j dont l'intérieur est inclus dans $E(k)$ et contient k est $a_{ik}^{(k-1)} + a_{kj}^{(k-1)}$ car tout chemin de cet ensemble est constitué de la concaténation d'un chemin de i à k dont l'intérieur est inclus dans $E(k-1)$ et d'un chemin de k à j dont l'intérieur est inclus dans $E(k-1)$.

Comme les deux ensembles de chemins précédents recouvrent tous les chemins de i à j dont l'intérieur est inclus dans $E(k)$, la valeur maximale cherchée est $\max\{a_{ij}^{(k-1)}, a_{ik}^{(k-1)} + a_{kj}^{(k-1)}\}$. ∎

L'algorithme issu de cette proposition a été découvert par Floyd, la procédure ci-dessous de complexité $O(n^3)$ implémente cet algorithme et n'utilise qu'une seule matrice de travail a.

```
procédure FLOYD(G, e);
  {g est la matrice d'adjacence du graphe G}
  {e est la matrice des étiquettes du graphe G}
  {a est la matrice résultat}
  pour i de 1 à n faire
    pour j de 1 à n faire
      a_ij :=si g_ij = 1 alors e_ij sinon −∞
    finpour;
    a_ii := 0
  finpour;
  pour k de 1 à n faire
    pour i de 1 à n faire
      pour j de 1 à n faire
        a_ij^(k) = max{a_ij^(k−1), a_ik^(k−1) + a_kj^(k−1)}
  retourner(a).
```

La figure 2.2 montre un graphe étiqueté par des réels. Les matrices, où $-\infty$ est

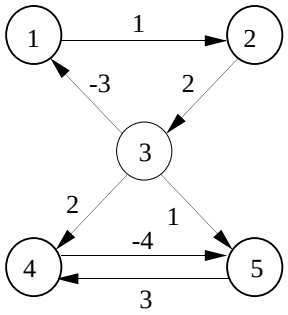

Figure 2.2: *Un graphe étiqueté.*

codé par un tiret, calculées par la procédure FLOYD(G) sont les suivantes :

$$a^{(0)} = \begin{pmatrix} 0 & 1 & - & - & - \\ - & 0 & 2 & - & - \\ -3 & - & 0 & 2 & 1 \\ - & - & - & 0 & -4 \\ - & - & - & 3 & 0 \end{pmatrix} a^{(1)} = \begin{pmatrix} 0 & 1 & - & - & - \\ - & 0 & 2 & - & - \\ -3 & -2 & 0 & 2 & 1 \\ - & - & - & 0 & -4 \\ - & - & - & 3 & 0 \end{pmatrix} a^{(2)} = \begin{pmatrix} 0 & 1 & 3 & - & - \\ - & 0 & 2 & - & - \\ -3 & -2 & 0 & 2 & 4 \\ - & - & - & 0 & -4 \\ - & - & - & 3 & 0 \end{pmatrix}$$

$$a^{(3)} = \begin{pmatrix} 0 & 1 & 3 & 5 & 4 \\ -1 & 0 & 2 & 4 & 3 \\ -3 & -2 & 0 & 2 & 1 \\ - & - & - & 0 & -4 \\ - & - & - & 3 & 0 \end{pmatrix} a^{(4)} = \begin{pmatrix} 0 & 1 & 3 & 5 & 4 \\ -1 & 0 & 2 & 4 & 3 \\ -3 & -2 & 0 & 2 & 1 \\ - & - & - & 0 & -4 \\ - & - & - & 3 & 0 \end{pmatrix} a^{(5)} = \begin{pmatrix} 0 & 1 & 3 & 7 & 4 \\ -1 & 0 & 2 & 6 & 3 \\ -3 & -2 & 0 & 4 & 1 \\ - & - & - & 0 & -4 \\ - & - & - & 3 & 0 \end{pmatrix}$$

Langage des chemins d'un graphe étiqueté

Nous considérons ici un alphabet A et prenons pour $\mathcal{E}$ l'ensemble des parties non vides de A. L'ensemble des étiquettes d'un chemin de i à j est le langage produit des étiquettes associées aux arcs successifs du chemin. Nous appelons alors langage des chemins de i à j, et nous notons L_{ij} l'ensemble des étiquettes des chemins de i à j. Le problème consiste alors à déterminer pour chaque couple (i, j) le langage L_{ij}.

Ce problème est classique dans le cadre de la théorie des automates, on peut en particulier se servir de la construction itérative des L_{ij} réalisée par l'algorithme dans la preuve du théorème de Kleene (voir chapitre 9).

Nous notons $L_{ij}^{(k)}$ le langage des chemins de i à j dont l'intérieur est inclus dans $E(k)$ et nous prenons pour $k = 0$ les valeurs initiales suivantes :

$$L_{ij}^{(0)} = \begin{cases} A_{ij} & \text{si } i \neq j \\ A_{ij} \cup \{1\} & \text{sinon} \end{cases}$$

où 1 représente le mot vide et A_{ij} est l'ensemble des étiquettes des arcs de i à j. La proposition suivante permet alors le calcul des langages $L_{ij}^{(k)}$:

Proposition 2.5. *Pour tout $i, j, k \in S$, on a*

$$L_{ij}^{(k)} = L_{ij}^{(k-1)} \cup L_{ik}^{(k-1)} L_{kk}^{(k-1)*} L_{kj}^{(k-1)}.$$

Preuve. Un mot du second membre est soit dans $L_{ij}^{(k-1)}$, soit dans l'ensemble $L_{ik}^{(k-1)}(L_{kk}^{(k-1)})^* L_{kj}^{(k-1)}$. Dans le premier cas le mot appartient à $L_{ij}^{(k)}$, dans le second cas, il s'écrit uvw où u appartient à $L_{ik}^{(k-1)}$, v à $(L_{kk}^{(k-1)})^*$ et w à $L_{kj}^{(k-1)}$. Par définition u est donc une étiquette d'un chemin de i à k dont l'intérieur est inclus dans $E(k-1)$, v est la concaténation d'un nombre fini d'étiquettes de circuits de k à k dont l'intérieur est inclus dans $E(k-1)$, enfin v est l'étiquette d'un chemin de k à j dont l'intérieur est inclus dans $E(k-1)$. Le mot formé par la concaténation de u, v et w est alors une étiquette du chemin obtenu par concaténation des chemins associés.

Réciproquement, considérons un mot $u \in L_{ik}^{(k)}$. Si l'intérieur de ce chemin ne contient pas k, u appartient par définition à $L_{ij}^{(k-1)}$; si l'intérieur de ce chemin contient k, ce chemin est formé de la concaténation d'un chemin de i à k (premier passage par k en tant que sommet intermédiaire), d'une suite finie (éventuellement

vide) de chemins de k à k (associés à tous les passages successifs par k en tant que sommet intermédiaire) et d'un dernier chemin de k à j. Par construction, l'intérieur de tous ces chemins est inclus dans $E(k-1)$. Le mot u est donc la concaténation d'un mot de $L_{ik}^{(k-1)}$, d'une suite de mots de $L_{kk}^{(k-1)}$ et d'un mot de $L_{kj}^{(k-1)}$. ∎

Nous pouvons donc déduire de la proposition précédente que, quel que soit le couple (i, j), le langage des chemins de i à j est obtenu à partir des langages réduits aux lettres de A par une suite finie d'opérations d'union, concaténation et étoile.

4.2.3 Semi-anneaux et accessibilité

Un ensemble K muni de deux opérations binaires associatives $\oplus$ et $\otimes$ et de deux éléments distingués 0 et 1 est un *semi-anneau* si

1. $(K, \oplus, 0)$ est un monoïde commutatif, c'est-à-dire $a \oplus b = b \oplus a$ pour $a, b \in K$ et $a \oplus 0 = 0 \oplus a = a$ pour tout a dans K ;
2. $(K, \otimes, 1)$ est un monoïde et $a \otimes 1 = 1 \otimes a = a$ pour tout a dans K ;
3. l'opération $\otimes$ est distributive par rapport à $\oplus$, c'est-à-dire : $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ et $(a \oplus b) \otimes c = (a \otimes c) \oplus (a \otimes b)$;
4. l'élément 0 est un zéro pour l'opération $\otimes$: $0 \otimes a = a \otimes 0 = 0$.

On note $(K, \oplus, \otimes, 0, 1)$ un semi-anneau. Par exemple $(\mathbb{N}, +, \times, 0, 1)$ est un semi-anneau. Un semi-anneau $(K, \oplus, \otimes, 0, 1)$ est *complet* si toute famille $\{a_i\}_{i \in I}$ d'éléments de K admet une somme notée $\bigoplus_{i \in I} a_i$ sujette aux conditions suivantes :

a) si I est fini, $I = \{i_1, \ldots, i_n\}$, alors $\bigoplus_{i \in I} a_i = a_{i_1} \oplus a_{i_2} \ldots \oplus a_{i_n}$;

b) la somme est associative : si $I = \bigcup_{j \in J} I_j$, avec $I_j \cap I_k = \emptyset$ pour $j \neq k$, alors :

$$\bigoplus_{i \in I} a_i = \bigoplus_{j \in J} (\bigoplus_{i \in I_j} a_i)$$

c) l'opération $\otimes$ est distributive par rapport à $\oplus$, c'est-à-dire :

$$(\bigoplus_{i \in I} a_i) \otimes (\bigoplus_{j \in J} b_j) = \bigoplus_{i \in I} \bigoplus_{j \in J} (a_i \otimes b_j).$$

Cette définition de la complétude est assez lourde, il suffit de retenir que les sommes infinies sont autorisées et que l'on peut les manipuler comme des sommes finies.

Exemples.

1) Le semi-anneau de Boole $(\mathcal{B}, \vee, \wedge, 0, 1)$, avec $\mathcal{B} = \{0, 1\}$ est complet. On a :

$$\bigvee_{i \in I} a_i = \begin{cases} 1 & \text{s'il existe } i \in I \text{ tel que } a_i = 1\,; \\ 0 & \text{sinon.} \end{cases}$$

2) Soit $\mathcal{N} = \mathbb{N} \cup \{+\infty\}$, avec $a + (+\infty) = (+\infty) + a = +\infty$. Alors $(\mathcal{N}, \min, +, \infty, 0)$ est un semi-anneau complet. La distributivité s'exprime par :

$$a + \min\{b, c\} = \min\{a + b, a + c\}.$$

3) Soit A un alphabet (voir chapitre 9) et A^* l'ensemble des mots sur A. Alors $(\mathcal{P}(A^*), \cup, \cdot, \emptyset, \epsilon)$ est un semi-anneau complet.

4) Soit $\mathcal{R} = \mathbb{R} \cup \{+\infty, -\infty\}$, avec $(+\infty) + (-\infty) = +\infty$. Alors $(\mathcal{R}, \min, +, +\infty, 0)$ est un semi-anneau complet.

5) Soit $\mathcal{N} = \mathbb{N} \cup \{+\infty\}$. Alors $(\mathcal{N}, \max, \min, 0, +\infty)$ est un semi-anneau complet. On a $\max\{a, +\infty\} = +\infty$, $\max\{0, a\} = a$ et

$$\min\{a, \max\{b, c\}\} = \max\{\min\{a, b\}, \min\{a, c\}\}.$$

Dans un semi-anneau complet, on définit l'*étoile* a^* d'un élément a par :

$$a^* = 1 \oplus a \oplus a^2 \oplus \ldots \oplus a^n \oplus \ldots$$

avec $a^0 = 1$, et $a^{n+1} = a \otimes a^n$. Dans le semi-anneau de Boole, on a $a^* = 1$ pour tout a. Dans celui de l'exemple 2), on a $a^* = 0$. Dans celui de l'exemple 4), on a

$$a^* = \min_{n \in \mathbb{N}} na = \begin{cases} 0 & \text{si } a \geq 0, \\ -\infty & \text{sinon.} \end{cases}$$

Les coûts des chemins

Soit maintenant $G = (S, A)$ un graphe et soit K un semi-anneau complet. Soit $c : A \mapsto K$ une application qui à un arc u de A associe son *coût* $c(u)$. On étend c aux chemins comme suit : si $\gamma = (x_0, \ldots, x_n)$ est un chemin, alors

$$c(\gamma) = c(x_0, x_1) \otimes c(x_1, x_2) \otimes \ldots \otimes c(x_{n-1}, x_n).$$

(Notons que pour $n = 0$, $c(\gamma) = 1$). Si Γ est un ensemble de chemins, on pose :

$$c(\Gamma) = \bigoplus_{\gamma \in \Gamma} c(\gamma)$$

(et en particulier $c(\emptyset) = 0$). Le problème général qui nous intéresse est le calcul des éléments

$$c_{ij} = \bigoplus_{\gamma \in \Gamma_{ij}} c(\gamma)$$

où Γ_{ij} est l'ensemble des chemins de i à j dans G. Revenons sur nos exemples. Soit $G = (S, A)$ un graphe.

1) Avec le semi-anneau de Boole, fixons le coût d'un arc égal à 1. On a alors :

$$c_{ij} = \begin{cases} 1 & \text{si } \Gamma_{ij} \neq \emptyset, \\ 0 & \text{sinon.} \end{cases}$$

Ainsi, $c_{ij} = 1$ si et seulement si j est accessible de i.
2) Avec le semi-anneau de Floyd $(\mathcal{N}, \min, +, \infty, 0)$, le coût d'un chemin est la *somme* des coûts des arcs successifs qui le composent, le coût c_{ij} est le minimum des coûts des chemins de i à j. Ce coût est $+\infty$ s'il n'y a pas de chemin de i à j.
3) Avec la terminologie du chapitre 9, c_{ij} est l'ensemble des mots qui sont les étiquettes d'un chemin de i à j.
4) Avec le semi-anneau $(\mathcal{R}, \min, +, +\infty, 0)$, à nouveau c_{ij} est le minimum des coûts des chemins de i à j. Notons que s'il existe un circuit de coût négatif passant par le sommet i, alors $c_{ii} = -\infty$.
5) Appelons *capacité* d'un arc u le nombre $c(u)$. La *capacité d'un chemin* est la capacité minimale des arcs qui le composent et pour deux sommets i et j, c_{ij} est la *capacité maximale* d'un chemin de i à j.

Nous allons voir que l'algorithme de Roy-Warshall (voir section 4.2.1) s'étend au calcul des valeurs c_{ij}.

Soit $G = (S, A)$, avec $S = \{1, \ldots, n\}$. Rappelons que $E(k) = \{1, \ldots, k\}$ et que $I(\gamma)$ est l'intérieur d'un chemin γ. Notons Γ_{ij}^k l'ensemble des chemins γ dont l'intérieur est contenu dans $E(k)$. Soit $c : A \mapsto K$ une fonction de coût dans un semi-anneau complet K. On pose :

$$c_{ij}^k = c(\Gamma_{ij}^k) = \bigoplus_{\gamma \in \Gamma_{ij}^k} c(\gamma).$$

Lemme 2.6. *On a, pour i et j, dans S*

$$c_{ij}^k = c_{ij}^{k-1} \oplus c_{ik}^{k-1} \otimes c_{kk}^{k-1^*} \otimes c_{kj}^{k-1}. \tag{2.1}$$

Preuve. Notons D le membre droit de l'équation 2.1. On a en vertu de la distributivité généralisée

$$c_{kk}^{k-1^*} = \bigoplus_{m \in \mathbb{N}} \bigoplus_{\{\gamma_1, \ldots, \gamma_m\} \in \Gamma_{kk}^{k-1}} c(\gamma_1) \otimes c(\gamma_2) \ldots \otimes c(\gamma_m)$$

et par conséquent

$$D = \bigoplus_{\gamma \in \Gamma_{ij}^{k-1}} c(\gamma) \oplus \bigoplus_{\alpha \in \Gamma_{ik}^{k-1}, \delta \in \Gamma_{kj}^{k-1}} \bigoplus_{m \in \mathbb{N}} \bigoplus_{\gamma_1, \ldots, \gamma_m \in \Gamma_{kk}^{k-1}} c(\alpha) \otimes c(\gamma_1) \otimes c(\gamma_2) \ldots \otimes c(\gamma_m) \otimes c(\delta).$$

Tout chemin $\gamma \in \Gamma_{ij}^k$ est soit chemin de Γ_{ij}^{k-1}, soit se décompose *de manière unique* en $\gamma = \alpha(\gamma_1, \ldots, \gamma_m)\delta$, où $\alpha \in \Gamma_{ik}^{k-1}$, $\delta \in \Gamma_{kj}^{k-1}$, $m \in \mathbb{N}$, $\gamma_1, \ldots, \gamma_m \in \Gamma_{kk}^{k-1}$. D'où le résultat. ∎

Dans les cas particuliers des semi-anneaux K tels que $a^* = 1$ pour tout $a \in K$, la formule 2.1 se simplifie en :

$$c_{ij}^k = c_{ij}^{k-1} \oplus c_{ik}^{k-1} \otimes c_{kj}^{k-1}.$$

Il en est ainsi pour l'accessibilité et pour les plus courts chemins sans circuit de coût négatif. Dans ce cas la proposition 2.2 reste vraie et l'algorithme de Roy-Warshall se transpose directement en remplaçant l'avant-dernière ligne par :

$$a_{ij}^k := a_{ij}^{k-1} \oplus (a_{ik}^{k-1} \otimes a_{kj}^{k-1}).$$

On obtient alors le théorème :

Théorème 2.7. *Le calcul des c_{ij}, pour $i, j \in S$, se fait par l'algorithme de Roy-Warshall en $O(n^3)$ opérations $\oplus$, $\otimes$ et $*$, où n est le nombre de sommets du graphe.*

Preuve. (immédiate d'après ce qui précède.) ∎

4.2.4 Forte connexité

Soit $G = (S, A)$ un graphe orienté. La relation d'accessibilité définie précédemment est un préordre sur S (réflexive et transitive). Si l'on note $\rightarrow$ cette relation, la relation d'équivalence sur S induite par ce préordre, que l'on notera $\leftrightarrow$ est définie par : $i \leftrightarrow j$ si $i \rightarrow j$ et $j \rightarrow i$. Les classes d'équivalence de la relation $\leftrightarrow$ s'appellent les *composantes fortement connexes* de G. Deux sommets distincts appartiennent à une même classe si et seulement s'ils appartiennent à un même circuit. La relation induite par $\rightarrow$ sur l'ensemble quotient des classes d'équivalence définit le *graphe quotient* de G par $\rightarrow$. On appelle alors *graphe réduit* de G le graphe obtenu à partir du graphe quotient en supprimant ses boucles. Ce graphe est *sans circuit* car l'existence d'un circuit passant par deux classes distinctes α et β entraînerait l'équivalence pour $\leftrightarrow$ d'un sommet quelconque de α et d'un sommet quelconque de β, d'où la contradiction. Le graphe de la figure 2.3 possède deux composantes fortement connexes $C' = \{1, 2, 3\}$ et $C'' = \{4, 5\}$. La détermination des composantes fortement connexes d'un graphe est un problème fréquent dans l'analyse des graphes d'état (chaînes de Markov, réseaux de Petri,...) modélisant l'évolution d'un système. On distingue alors souvent les classes «finales» qui correspondent aux sorties du graphe réduit (dès que l'état du système est un sommet d'une classe finale, les états postérieurs appartiendront à cette classe) et les autres dont les sommets sont dits «transitoires». Pour l'exemple de la figure 2.3, C'' est la seule classe finale et les états $\{1, 2, 3\}$ sont transitoires. On verra au paragraphe 4.4.5 un algorithme efficace de calcul des composantes fortement connexes.

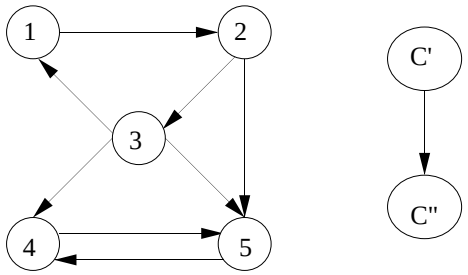

Figure 2.3: *Un graphe orienté et son graphe réduit.*

4.3 Arbres et arborescences

Les arbres représentent une structure *minimale* en nombre de liaisons pour connecter un ensemble de sommets. A ce titre, ils constituent une classe fondamentale de graphes intervenant dans de nombreux problèmes d'optimisation de réseaux. Les arborescences fournissent des structures de données à la base de nombreux algorithmes performants.

4.3.1 Arbres

Nous considérons dans ce paragraphe des graphes non orientés, et nous rappelons que dans un graphe non orienté, la longueur d'un cycle est supérieure ou égale à trois.

Soit G un graphe à n sommets ($n \geq 1$) ; nous allons prouver que les six propriétés ci-dessous sont équivalentes. Un graphe vérifiant l'une de ces propriétés est un arbre.

1) G est un graphe connexe sans cycle ;
2) G est un graphe connexe possédant $n-1$ arêtes ;
3) G est un graphe sans cycle possédant $n-1$ arêtes ;
4) G est un graphe tel que deux sommets quelconques sont liés par une seule chaîne ;
5) G est un graphe connexe qui perd sa connexité par suppression d'une arête quelconque ;
6) G est un graphe sans cycle tel que l'adjonction d'une arête quelconque crée un cycle et un seul.

Pour démontrer l'équivalence de ces définitions, deux lemmes préliminaires seront utiles.

Lemme 3.1. *Un graphe connexe à n sommets possède au moins $n-1$ arêtes.*

Preuve. (Induction sur n.) La propriété est vraie pour $n = 1$. Supposons $n \geq 2$ et soit $G = (S, A)$ un graphe connexe à n sommets. Considérons un sommet u de S. Notons G_u le sous-graphe de G induit par $S - \{u\}$ et soient C_1,C_2,...,C_p

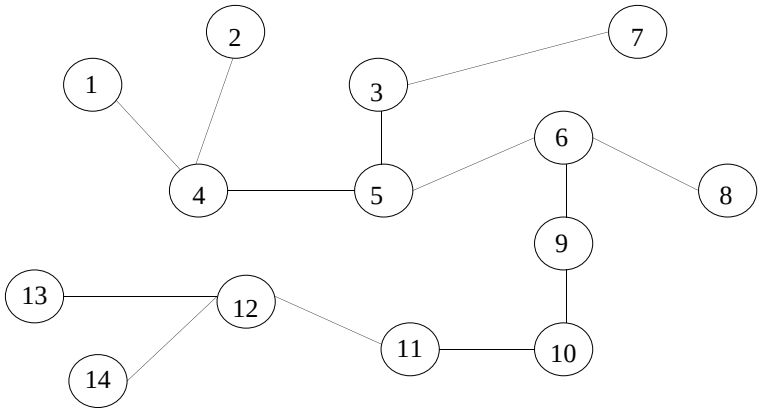

Figure 3.1: *Un arbre à quatorze sommets.*

les composantes connexes de G_u. Sur la figure 3.2 sont représentés à gauche un graphe orienté G et à droite le graphe G_u. Pour tout $k = 1, \ldots, p$, il existe au moins une arête liant u à un sommet de C_k, sinon G ne serait pas connexe. D'autre part les sous-graphes G^k de G induits par les C_k sont connexes. On a donc par induction $m_k \geq n_k - 1$, où n_k (respectivement m_k) désigne le nombre de sommets (respectivement d'arêtes) de G^k. Il en résulte que :

$$m \geq (\sum_{k=1}^{p} m_k) + p \geq \sum_{k=1}^{p} n_k = n - 1.$$ ∎

Lemme 3.2. *Un graphe à n sommets ayant au moins n arêtes possède un cycle.*

Preuve. (Induction sur n.) La propriété est vraie pour $n \leq 3$. Soit donc $n \geq 4$ et $G = (S, A)$ un graphe à n sommets possédant m arêtes ($m \geq n$). Supposons en raisonnant par l'absurde que G soit sans cycles et considérons un sommet u de S. Notons G_u le sous-graphe de G induit par $S - \{u\}$, C_1,C_2,...,C_p les composantes connexes de G_u et G^k le sous-graphe de G induit par C_k (voir figure 3.2). Chaque G^k est sans cycle, donc $m_k \leq n_k - 1$ et le nombre d'arêtes incidentes à u est $m - \sum_{k=1}^{p} m_k \geq m + p - \sum_{k=1}^{p} n_k \geq p + 1$. Il existe donc un k dans $\{1, \ldots, p\}$ tel que u soit adjacent à deux sommets distincts a et b de C_k. Le graphe G^k étant connexe, il existe une chaîne de a à b dans G^k et donc un cycle dans G passant par u, a et b. ∎

Proposition 3.3. *Soit G un graphe à n sommets, les six propriétés suivantes sont équivalentes :*

(1) *G est un graphe connexe sans cycle;*
(2) *G est un graphe connexe possédant $n - 1$ arêtes;*
(3) *G est un graphe sans cycle possédant $n - 1$ arêtes;*
(4) *G est un graphe tel que deux sommets quelconques sont liés par une seule chaîne;*

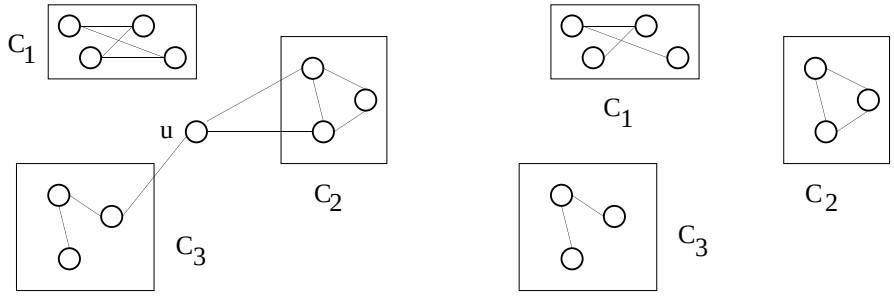

Figure 3.2: *Le graphe G_u.*

(5) *G est un graphe connexe qui perd la connexité par suppression d'une arête quelconque;*
(6) *G est un graphe sans cycle tel que l'adjonction d'une arête quelconque crée un cycle et un seul.*

Preuve. Nous montrons que $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4 \Rightarrow 5 \Rightarrow 6 \Rightarrow 1$.

$1 \Rightarrow 2$. Soit G un graphe à n sommets connexe et sans cycle. D'après le lemme 1, on a $m \geq n-1$ et d'après le lemme 2 on a $m \leq n-1$, d'où il résulte que $m = n-1$.

$2 \Rightarrow 3$. Un graphe G connexe et possédant $n-1$ arêtes est sans cycle car dans le cas contraire, la suppression d'une arête quelconque du cycle entraînerait l'existence d'un graphe connexe à n sommets et $n-2$ arêtes.

$3 \Rightarrow 4$. Supposons qu'un graphe G à $n-1$ arêtes soit sans cycle et ne soit pas connexe. Le graphe induit par chaque composante connexe qui est connexe et sans cycle possède un sommet de plus que d'arêtes. Le nombre de composantes connexes de G étant au moins deux, le graphe G lui même possède au plus $n-2$ arêtes. De plus l'existence de deux chaînes distinctes entre deux sommets entraîne celle d'un cycle dans G.

$4 \Rightarrow 5$. Soit G un graphe tel que pour toute paire de sommets, il existe une chaîne et une seule ayant ces deux sommets pour extrémités. Le graphe G est bien sûr connexe. Si G ne perdait pas la connexité par suppression d'une arête $\{a, b\}$, une chaîne n'utilisant pas cette arête lierait a et b, ce qui contredit l'hypothèse d'unicité de la chaîne liant a et b.

$5 \Rightarrow 6$. Si G est connexe et perd la connexité par suppression d'une arête quelconque, il est sans cycle car dans le cas contraire, la suppression d'une arête du cycle ne lui ferait pas perdre la connexité. De plus l'adjonction d'une arête $\{a, b\}$ crée un cycle puisqu'il existe déjà une chaîne liant a et b dans G. Ce cycle passe par l'arête $\{a, b\}$ puisque G est sans cycle. De plus si l'adjonction de $\{a, b\}$ créait deux cycles distincts, il existerait dans G deux chaînes distinctes de a à b et donc un cycle.

$6 \Rightarrow 1$. Si l'adjonction d'une arête quelconque $\{a, b\}$ crée un cycle, G est connexe car dans le cas contraire, l'adjonction d'une arête entre deux sommets appartenant

à deux composantes connexes distinctes ne créerait pas de cycles. ∎

4.3.2 Arborescences

On obtient une arborescence à partir d'un arbre en distinguant l'un de ses sommets. Une *arborescence* est donc un couple formé d'un arbre H et d'un sommet distingué r appelé *racine*. Une arborescence (H, r) peut aussi être définie naturellement en tant que graphe orienté en substituant l'arc (u, v) à l'arête $\{u, v\}$ si la chaîne de r à v dans H passe par u, l'arc (v, u) sinon. Une arborescence de racine r est alors un graphe orienté tel qu'il existe un chemin *unique* de r à n'importe quel sommet.

L'une des propriétés les plus importantes d'une arborescence est sa structure récursive que nous mettons en évidence par le lemme suivant :

Lemme 3.4. *Soit $A = (H, r)$ une arborescence possédant au moins deux sommets, H_r le graphe induit par $S - \{r\}$ et $r_1, \ldots, r_k$ les sommets adjacents à la racine. Les sous-graphes induits par les composantes connexes de H_r constituent k arbres disjoints contenant chacun un et un seul des sommets r_i.*

Preuve. Chaque composante connexe de H_r contient un et un seul r_i car H ne possède pas de cycle. Les k sous-graphes induits par ces composantes qui sont connexes et sans cycle sont des arbres disjoints. ∎

La figure 3.3 illustre la structure récursive d'une arborescence.

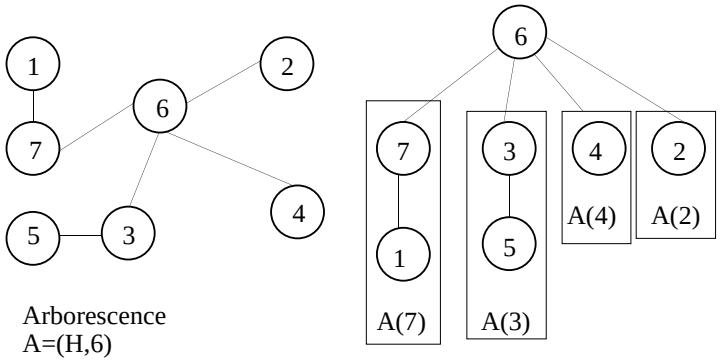

Figure 3.3: *Structure récursive d'une arborescence.*

Etant donnée l'importance des structures arborescentes, certaines notations spécifiques ont été créées. Les *ascendants* et les*descendants* d'un sommet ont leur définition usuelle pour l'arborescence considérée comme un graphe orienté. Tout sommet s (sauf la racine) a un prédécesseur unique noté $p(s)$ et appelé *père* de s. Inversement les successeurs d'un sommet s sont ses *fils*. Si le sommet s n'a pas de fils, il est appelé *feuille* (ou encore *sommet externe*, dans le cas contraire, c'est un

noeud (ou encore *sommet interne*). La *profondeur* d'un sommet est la longueur du chemin de la racine à ce sommet. La *hauteur* d'un sommet u est la longueur maximale d'un chemin d'origine u. Si u et v sont deux sommets, l'*ancêtre* de u et v est l'ascendant commun de u et v de profondeur maximale.

Par conformité avec les six définitions initiales (en particulier : $m = n - 1$), tous les arbres que nous avons définis jusqu'ici possèdent au moins un sommet. Il est cependant commode, surtout dans le cadre des applications informatiques liées aux structures de données et leur programmation (par exemple l'utilisation du pointeur « nil » du langage Pascal), de considérer qu'un arbre vide est aussi un arbre. Sauf mention contraire, nous adopterons désormais cette convention et noterons Λ une arborescence vide.

4.3.3 Arborescences ordonnées

Il est souvent utile de définir pour chaque sommet d'une arborescence un ordre total sur les fils de ce sommet. C'est par exemple le cas pour les arbres 2-3 (voir chapitre 6) ou encore les arbres de dérivation dans une grammaire. Si l'on adjoint à chaque sommet interne d'une arborescence un ordre total sur les fils de ce sommet, on définit une *arborescence ordonnée*. Les relations d'ordre locales permettent de munir l'ensemble des sommets de l'arborescence d'un ordre total.

Soit A une arborescence ordonnée, la liste L représentant l'ordre total sur les sommets de A est construite récursivement comme suit : si A est l'arborescence vide, la liste L est vide; sinon soient $L_1, \ldots, L_k$ les listes associées aux ordres totaux pour les sous-arborescences de racines $r_1, \ldots, r_k$ et $(r_1, \ldots, r_k)$ la liste associéee à l'ordre total sur les fils $r_1, \ldots, r_k$ de la racine, la liste L est égale à $(r) \cdot L_1 \cdot L_2 \cdots L_k$ où l'on note $L \cdot L'$ la concaténation des deux listes L et L'.

Remarque. La liste associée à l'ordre total sur les sommets d'une arborescence ordonnée est une liste préfixe au sens défini dans la section 4.4.5.

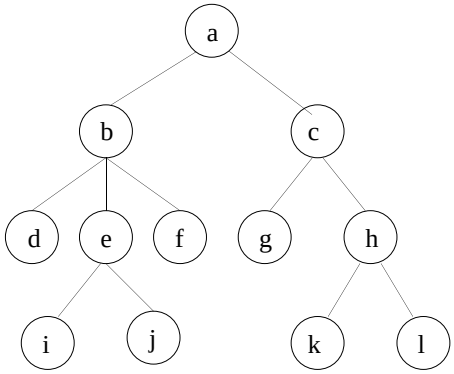

Figure 3.4: *Une arborescence.*

La figure 3.4 représente une arborescence ordonnée dans laquelle, comme il est d'usage, les fils d'un même sommet sont dessinés de la gauche vers la droite dans l'ordre croissant. L'ordre total est donc $(a, b, d, e, i, j, f, c, g, h, k, l)$.

4.3.4 Arbres positionnés et arbres binaires

Un *arbre positionné d'arité p* est une arborescence telle que les arcs liant les frères d'un même sommet à leur père sont étiquetés par des éléments distincts de l'ensemble $\{1, \ldots, p\}$. On parle alors du $k^{ième}$ fils si l'arc a l'étiquette k. Un arbre positionné d'arité p est *complet* si chacun de ses noeuds possède p fils.

Un *arbre binaire* est un arbre positionné d'arité 2. Au lieu de l'étiqueter sur $\{1, 2\}$, il est plus parlant d'utiliser les étiquettes « gauche » et « droit ». Chaque nœud possède donc soit un fils « droit », soit un fils « gauche », soit les deux.

Une définition récursive des arbres binaires, fondamentale dans les applications informatiques, s'énonce comme suit :

(1) l'arbre vide est un arbre binaire ;
(2) soient A_1 et A_2 deux arbres binaires et s un sommet n'appartenant ni à A_1 ni à A_2, alors l'arborescence de racine s, de sous-arbre gauche A_1 et de sous-arbre droit A_2 est un arbre binaire.

La distinction entre fils gauche et fils droit entraîne quelques notations supplémentaires. Soit A un arbre binaire complet et u l'un de ses sommets internes, $A_g(u)$ (respectivement $A_d(u)$) désigne le sous-arbre de A dont la racine est le fils gauche de u (respectivement le fils droit de u). Si u est la racine on note A_g (respectivement A_d) le sous-arbre gauche (respectivement le sous-arbre droit) de A. Le sous-arbre de racine u est noté $A(u)$. La figure 3.5 présente un exemple d'arbre binaire.

4.3.5 Arbre binaire complet

Un *arbre binaire complet* est un arbre binaire non vide et complet, c'est-à-dire tel que chaque noeud possède zéro ou deux fils. La figure 3.5 représente un arbre binaire complet. La définition récursive suivante des arbres binaires complets est également très utile :

(1) une arborescence réduite à sa racine est un arbre binaire complet ;
(2) soient A_1 et A_2 deux arbres binaires complets et s un sommet n'appartenant ni à A_1 ni à A_2, l'arborescence A de racine s, de sous-arbre gauche A_1 et de sous-arbre droit A_2 est un arbre binaire complet.

La figure 3.6 montre les premiers arbres binaires complets. La proposition simple qui suit illustre le raisonnement par induction fréquemment pratiqué sur les arbres binaires complets.

Proposition 3.5. *Un arbre binaire complet à p ($p \geq 1$) sommets externes possède $p - 1$ sommets internes.*

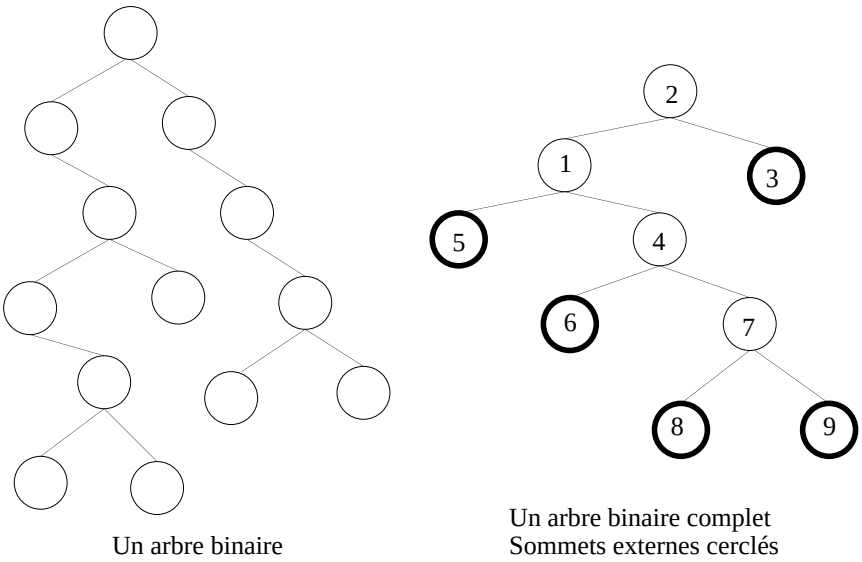

Figure 3.5: *Arbre binaire et arbre binaire complet.*

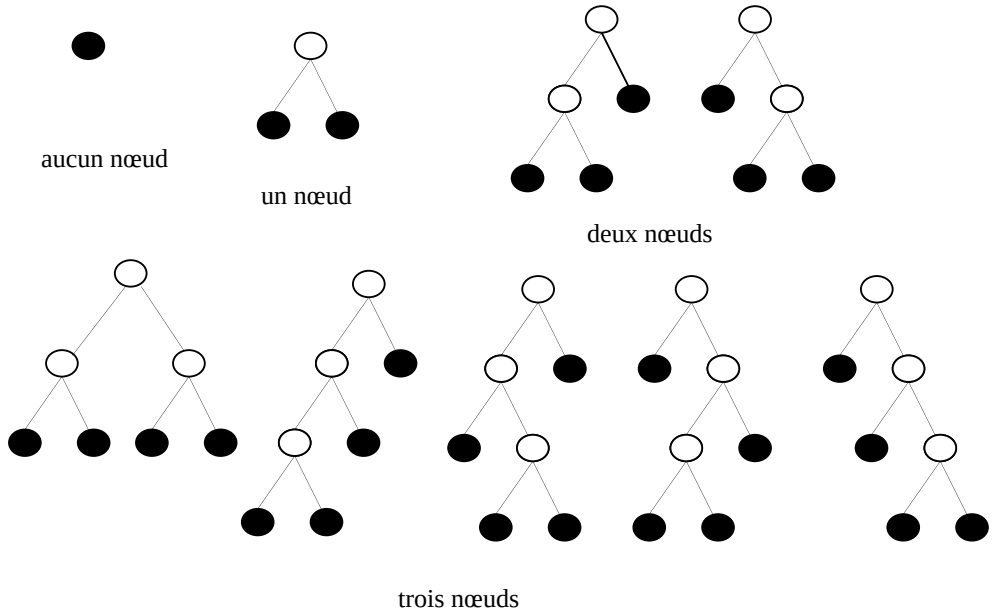

Figure 3.6: *Les premiers arbres binaires complets.*

Preuve. (Induction sur p.) Si $p = 1$, l'arbre binaire complet est réduit à un sommet qui est externe et la propriété est vraie. Soient $p \geq 2$ et A un arbre binaire complet à p sommets externes. L'arbre A est formé d'une racine r, d'un sous-arbre gauche A_1 qui est un arbre binaire complet à p_1 sommets externes ($p_1 \geq 1$) et d'un sous-arbre droit A_2 qui est un arbre binaire complet à p_2 sommets externes ($p_2 \geq 1$). On a $p_1 + p_2 = p$ et par récurrence le nombre de sommets internes de A_1 est donc $p_1 - 1$ et celui de A_2 est $p_2 - 1$. Le nombre de sommets internes de A est donc $1 + (p_1 - 1) + (p_2 - 1) = p - 1$. ∎

Les arbres binaires complets ne constituent en fait qu'une représentation plus structurée des arbres binaires. En effet, l'opération d'*effeuillage* permet d'associer à un arbre binaire complet un arbre binaire (sous-jacent). L'image résultant de l'effeuillage d'un arbre binaire complet A est le sous-graphe de A induit par ses sommets internes. L'effeuillage n'est pas une opération injective mais on peut montrer que tous les arbres binaires complets ayant la même image sont isomorphes. Ils ne diffèrent en fait que par les noms des feuilles supprimées. La figure 3.7 représente un arbre binaire complet et à sa droite l'arbre effeuillé.

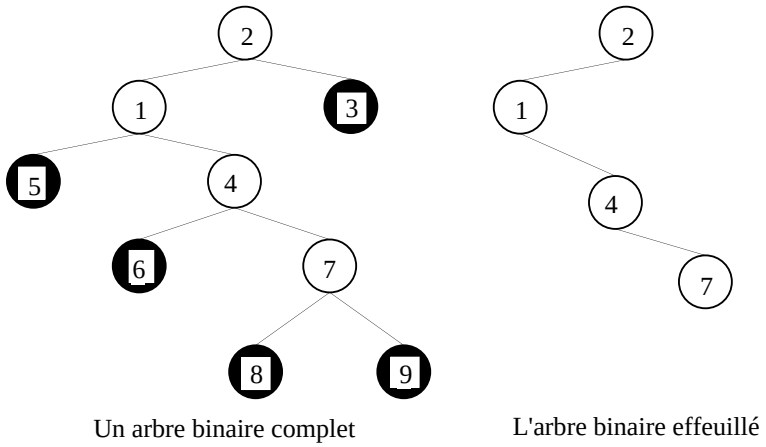

Figure 3.7: *Effeuillage d'un arbre binaire complet.*

L'opération réciproque appelée *complétion* consiste à compléter à deux le nombre de fils de chaque sommet de A. Cette définition ne précise cependant pas la règle de choix des noms des nouveaux sommets. La figure 3.8 montre l'opération de complétion.

4.4 Parcours d'un graphe

4.4.1 Parcours d'un graphe non orienté

Dans cette section nous considérons un graphe non orienté connexe $G = (S, A)$. Un *parcours* de G à partir de l'un de ses sommets s est une liste de sommets L telle que :

- le premier sommet de L est s,
- chaque sommet de S apparaît une fois et une seule dans L,
- tout sommet de la liste (sauf le premier) est adjacent à au moins un sommet placé avant lui dans la liste.

Nous présentons d'abord un algorithme générique de calcul d'un parcours. Nous étudions ensuite les deux types de parcours les plus utilisés : les parcours *en profondeur* et les parcours *en largeur*.

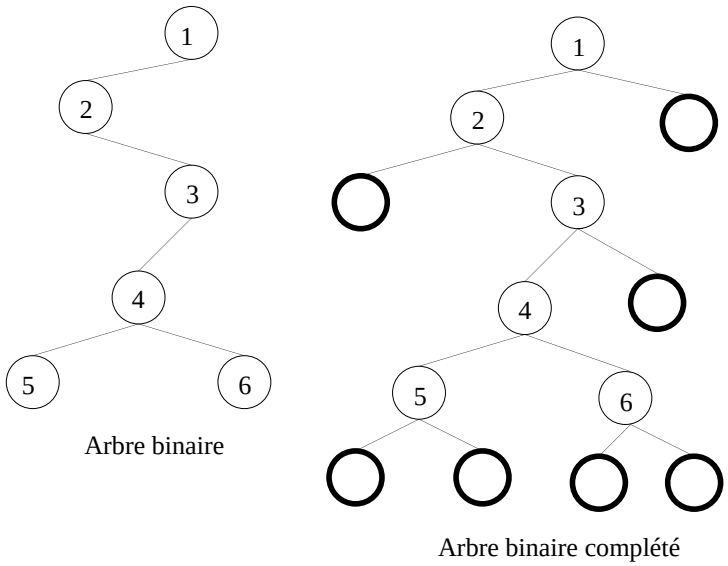

Figure 3.8: *Complétion d'un arbre binaire.*

Définitions et notations

Soit T une partie non vide de S, la *bordure* $\mathcal{B}(T)$ de T est l'ensemble des sommets de $S - T$ adjacents à T. Pour le graphe de la figure 4.1, la bordure de $\{3, 5, 6\}$ est $\{1, 2, 4\}$.

Soit L une liste de sommets de G. Le *support* de L, noté $\sigma(L)$, est l'ensemble des sommets présents dans la liste (par exemple $\sigma(1, 2, 1, 3, 5, 3, 3) = \{1, 2, 3, 5\}$). La liste des k premiers sommets de L est notée L_k. Un sommet u de L est *fermé* si tous ses voisins dans G appartiennent à $\sigma(L)$, dans le cas contraire il est *ouvert*. Comme les listes qui interviennent dans ce chapitre ne contiennent qu'une seule occurrence de chaque élément de leur support, nous utiliserons la notation (abusive mais plus simple) $\mathcal{B}(L)$ à la place de $\mathcal{B}(\sigma(L))$.

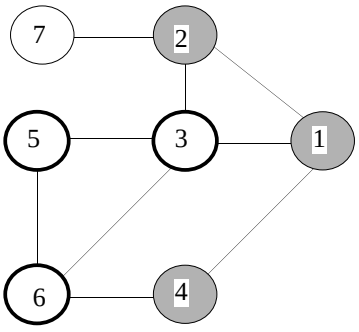

Figure 4.1: *Bordure d'un sous-ensemble de sommets.*

Algorithme générique et propriétés des parcours

Nous donnons dans cette section un algorithme générique de construction d'un parcours. Nous présentons ensuite les propriétés fondamentales des parcours et nous proposons un premier niveau d'implémentation de l'algorithme générique.

Considérons la procédure PARCOURS-GÉNÉRIQUE(G, s) ci-dessous :

```
procédure PARCOURS-GÉNÉRIQUE(G, s);
  L := (s);
  pour k de 1 à n − 1 faire
    choisir un sommet v dans B(L);
    L := L · (v)
  fintantque.
```

La proposition suivante établit l'égalité entre l'ensemble des listes construites par la procédure et l'ensemble des parcours de G.

Proposition 4.1. *Les parcours de G sont exactement les listes construites par* PARCOURS-GÉNÉRIQUE(G, s).

Preuve. Soit L une liste de PARCOURS-GÉNÉRIQUE(G, s). Montrons par induction sur le nombre d'itérations que la liste L_k obtenue après k itérations satisfait l'invariant suivant : « L_k est un parcours du sous-graphe de G induit par $\sigma(L_k)$ ». La propriété est vraie pour $k = 0$. Considérons l'itération k ($0 < k \leq n-1$) et soit u un sommet de l'ensemble non vide $S - \sigma(L_{k-1})$. Comme G est connexe, il existe une chaîne élémentaire du sommet s de $\sigma(L_{k-1})$ au sommet u de $S - \sigma(L_{k-1})$ dont l'une des arêtes $\{x, y\}$ vérifie $x \in S - \sigma(L_{k-1})$ et $y \in \sigma(L_{k-1})$. Le choix du sommet x est donc possible à l'itération k tant que $k < n$. Comme d'une part L_{k-1} est un parcours du sous-graphe de G induit par $\sigma(L_{k-1})$ (hypothèse d'induction) et que d'autre part le sommet x appartient à $\mathcal{B}(L_{k-1})$, la liste $L_k = L_{k-1} \cdot (x)$ est un parcours du sous-graphe de G induit par $\sigma(L_k) = \sigma(L_{k-1}) \cup \{x\}$. La liste L_{n-1} est donc un parcours de G.

Réciproquement soit $L = (x_1, \ldots, x_n)$ un parcours de G à partir de s. Par définition d'un parcours, x_{k+1} appartient à $\mathcal{B}(\{x_1, \ldots, x_k\}) = \mathcal{B}(L_{k-1})$ pour k de 1 à $n - 1$. L est donc aussi la liste obtenue par PARCOURS-GÉNÉRIQUE en choisissant le sommet x_{k+1} à l'itération k. ∎

Soit $L = (x_1, \ldots, x_n)$ un parcours de G. Si nous associons à chaque sommet x_k ($k \in \{2, \ldots, n\}$) un sommet x'_k appartenant à $\mathcal{B}(\{x_k\}) \cap \{x_1, \ldots, x_{k-1}\}$, le sous-graphe de G induit par les arêtes $\{x_k, x'_k\}$, appelées *arêtes de liaison*, est un arbre couvrant de G. Cette propriété est très importante car elle montre qu'un parcours emprunte un nombre minimal d'arêtes de liaison.

Proposition 4.2. *Soit $L = (x_1, \ldots, x_n)$ un parcours de G. Pour tout k, $(2 \leq k \leq n)$, soit x'_k un sommet de $\{x_1, \ldots, x_{k-1}\}$ adjacent à x_k. Le sous-graphe induit par les arêtes $\{x_k, x'_k\}$, $(2 \leq k \leq n)$, est un arbre couvrant de G.*

Preuve. Nous notons G_p le sous-graphe de G induit par les arêtes $\{x_k, x'_k\}$ ($k \in \{1, \ldots, p\}$) et nous raisonnons par récurrence sur p. La propriété est vraie pour $p = 1$ car on a toujours $x'_1 = s$, et G_1 est donc réduit à l'arête $\{s, x_2\}$. Supposons $p \geq 2$; le graphe G_p est connexe car obtenu par adjonction à l'arbre G_{p-1} (hypothèse d'induction) d'un nouveau sommet x_{p+1} et d'une nouvelle arête incidente à x_{p+1} et à un sommet de G_{p-1}. Le graphe G_p qui est connexe et possède $p-1$ arêtes et p sommets (car G_{p-1} possède $p-2$ arêtes et $p-1$ sommets) est donc un arbre. Il en résulte que G_n est un arbre couvrant de G. ∎

La figure 4.2 représente le parcours $(1, 3, 4, 6, 7, 2, 5)$ d'un graphe non orienté et montre l'arbre couvrant associé (arêtes épaisses).

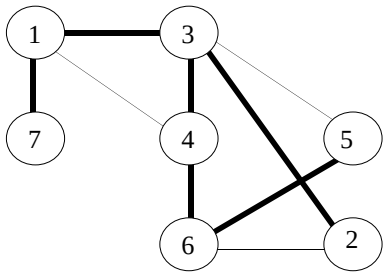

Figure 4.2: *Parcours d'un graphe et arbre couvrant.*

Avant d'étudier des parcours particuliers, nous proposons une implémentation partielle de l'algorithme générique décrivant le calcul des sommets ouverts. Nous appelons *degré résiduel* d'un sommet x, et nous le notons $r(x)$, le nombre de voisins de x qui n'appartiennent pas à la liste en cours. Un sommet de la liste en cours (nous dirons encore *sommet visité*) est donc ouvert si et seulement si son degré résiduel est non nul.

```
procédure PARCOURS(G, s);
  pour tout sommet x dans S faire
    r(x) := d(x); FERMER(x)
  finpour;
  {d(x) est le degré de x dans G}
  OUVRIR(s); EXAMINER-VOISINS(s);
  tantqu'il reste un sommet non visité faire
    choisir une arête de liaison {x, y} où x est ouvert et y non visité;
    L := L · (y);
    si r(y) > 0 alors
      OUVRIR(y);EXAMINER-VOISINS(y)
    finsi
  fintantque.
```

A chaque étape, l'algorithme PARCOURS ci-dessus choisit une arête de liaison $\{x, y\}$ où x est un sommet ouvert et y un sommet non visité, ouvre le sommet y si $r(y) > 0$ et diminue d'une unité le degré résiduel de chacun des voisins de y dans G. L'algorithme utilise un tableau booléen pour gérer l'ensemble des sommets visités, un tableau booléen pour gérer l'ensemble des sommets ouverts, un tableau linéaire pour gérer les degrés résiduels et une liste des voisins pour coder G.

La procédure EXAMINER-VOISINS(y) fait la mise à jour des degrés résiduels et des sommets ouverts à chaque visite d'un nouveau sommet. Sa complexité est $O(d(y))$.

```
procédure EXAMINER-VOISINS(y);
pour tout voisin z de y dans G faire
  r(z) := r(z) − 1;
  si z est visité et r(z) = 0 alors FERMER(z)
finpour.
```

Dans la mesure où l'algorithme de choix de l'arête de liaison n'est pas précisé, il n'est pas possible d'évaluer la complexité globale de l'algorithme PARCOURS. Cependant, comme chaque sommet est visité une fois et une seule, la procédure EXAMINER-VOISINS est également exécutée une fois et une seule pour chaque sommet. Il en résulte la proposition suivante :

Proposition 4.3. *Dans l'algorithme* PARCOURS, *la complexité des opérations autres que le choix de l'arête de liaison est* $O(n + m)$.

Nous allons désormais étudier deux types de parcours en particularisant la règle de choix des arêtes de liaison.

4.4.2 Parcours en profondeur

Un parcours L du graphe G à partir de s est dit *en profondeur* si, à chaque étape, l'arête de liaison $\{x, y\}$ choisie est telle que le sommet x soit le dernier sommet visité ouvert.

Une implémentation possible de l'algorithme de choix consiste alors à utiliser une pile de sommets visités posssédant les propriétés suivantes :

- tous les sommets ouverts sont dans la pile,
- si un sommet ouvert x a été visité avant un sommet ouvert y, alors y est placé plus haut que x dans la pile.

A partir d'une pile contenant initialement le seul sommet s, les opérations sur la pile, à chaque étape du parcours, sont les suivantes :

a) Si le sommet de pile t est ouvert, une arête $\{t, y\}$ est choisie comme arête de liaison et on empile y ;
b) Si le sommet de pile est fermé, on procède à des dépilements tant que le sommet de pile est fermé (et la pile non vide).

La procédure PROFONDEUR(G, s) ci-dessous réalise ces opérations de pile pour choisir les arêtes de liaison.

```
procédure PROFONDEUR(G, s) ;
  CRÉER(Π) ;
  pour tout sommet x dans S faire
    r(x) := d(x) ; FERMER(x)
  finpour ;
  L := (s) ; OUVRIR(s) ; EMPILER(s, Π) ; EXAMINER-VOISINS(s) ;
  tantque Π est non vide faire
    t :=SOMMET(Π) ;
    si t est ouvert alors
      choisir une arête {t, y} telle que y soit non visité ;
      L := L · (y) ; EMPILER(y, Π) ;
      si r(y) > 0 alors OUVRIR(y) ; EXAMINER-VOISINS(y) finsi
    sinon DÉPILER(Π)
    finsi
  fintantque.
```

Nous proposons au lecteur, à titre d'exercice, de démontrer que la pile Π gérée par la procédure PROFONDEUR(G, s) possède les deux propriétés requises. On peut plus précisément montrer, par récurrence sur le nombre de sommets visités, que la liste des sommets contenus de bas en haut dans la pile lors de la visite du sommet x est la liste des sommets du chemin de s à x dans l'arborescence de liaison en cours. Cette propriété est intéressante car elle induit une autre implémentation possible de l'algorithme de choix pour un parcours en profondeur. Il suffit en effet

de construire la fonction père de l'arborescence de liaison au fur et à mesure de la création des arêtes de liaison. Si le sommet visité en cours x est fermé, on utilise alors la fonction père pour remonter au dernier sommet visité ouvert.

Sur le graphe de la figure 4.3, la procédure PROFONDEUR calcule le parcours en profondeur $(1, 3, 4, 6, 5, 2, 7)$. Les contenus de la pile et de la liste sont donnés

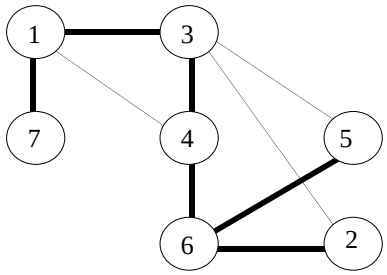

Figure 4.3: *Parcours en profondeur.*

dans le tableau ci-dessous. La première ligne correspond à la pile et la seconde à la liste :

(1)	(1,3)	(1,3,4)	(1,3,4,6)	(1,3,4,6,5)	(1,3,4,6)	(1,3,4,6,2)
(1)	(1,3)	(1,3,4)	(1,3,4,6)	(1,3,4,6,5)	(1,3,4,6,5)	(1,3,4,6,5,2)

(1,3,4,6)	(1,3,4)	(1,3)	(1)
(1,3,4,6,5,2)	(1,3,4,6,5,2)	(1,3,4,6,5,2)	(1,3,4,6,5,2)

(1,7)	(1)	()
(1,3,4,6,5,2,7)	(1,3,4,6,5,2,7)	(1,3,4,6,5,2,7)

Proposition 4.4. *La complexité en temps de la procédure* PROFONDEUR *est* $O(n + m)$.

Preuve. Evaluons d'abord la complexité des opérations sur la pile. Comme un sommet est visité une fois et une seule, il est empilé une fois et une seule. Il est donc dépilé au plus une fois et en fait exactement une fois en raison de la condition de terminaison. La complexité des opérations sur la pile est donc $O(n)$. Comme la complexité des opérations autres que le choix des arêtes de liaison dans l'algorithme PARCOURS est $O(n+m)$ (proposition 4.3), la complexité globale de la procédure PROFONDEUR est $O(n + m)$. ∎

Nous avons décrit jusqu'ici des algorithmes itératifs pour le calcul d'un parcours en profondeur à partir d'un sommet s. Nous présentons maintenant un algorithme récursif qui résout ce problème. Le principe de cet algorithme est de visiter s, puis de réaliser un appel récursif pour chaque voisin de s non encore visité. Un appel pour un sommet x est terminal si tous les voisins de x sont déja visités. La procédure PROFONDEUR-RÉCURSIF ci-dessous implémente cet algorithme.

```
procédure PROFONDEUR-RÉCURSIF(s);
  VISITER(s);
  pour tout voisin x de s faire
    si x est non visité alors
      VISITER(x); PROFONDEUR-RÉCURSIF(x)
    finsi
  finpour.
```

La procédure VISITER(x) met à jour le tableau des sommets visités et la liste des sommets visités. Cet algorithme, dont l'analyse est proposée en exercice, a une complexité $O(\max\{n, m\})$. L'une de ses variantes, utilisée pour le calcul des composantes fortement connexes d'un graphe orienté, est étudiée dans la section 4.4.5.

4.4.3 Parcours en largeur

Un parcours L du graphe G à partir de s est dit *en largeur* si, à chaque étape, l'arête de liaison $\{x, y\}$ choisie est telle que le sommet x soit le premier sommet visité ouvert.

Une implémentation possible de l'algorithme de choix consiste ici à utiliser une file de sommets visités posssédant les propriétés suivantes :

- tous les sommets ouverts sont dans la file,
- si un sommet ouvert x a été visité avant un sommet ouvert y, alors y est placé après x dans la file.

A partir d'une file contenant initialement le seul sommet s, les opérations sur la file, à chaque étape du parcours, sont les suivantes :

a) Si la tête de file t est un sommet ouvert, une arête $\{t, y\}$ est choisie comme arête de liaison et l'on enfile y;
b) Si la tête de file est un sommet fermé, on procède à des défilements tant que la tête de file est un sommet fermé (et la file non vide).

La procédure LARGEUR(G, s) ci-dessous réalise ces opérations de file pour choisir les arêtes de liaison.

```
procédure LARGEUR(G, s);
  CRÉER(Φ);
  pour tout sommet x dans S faire
    r(x) := d(x); FERMER(x)
  finpour;
  L := (s); OUVRIR(s); ENFILER(s, Φ); EXAMINER-VOISINS(s);
  tantque Φ est non vide faire
    t :=TÊTE(Φ);
    si t est ouvert alors
      choisir une arête {t, y} telle que y soit non visité;
      L := L · (y); ENFILER(y, Φ);
      si r(y) > 0 alors OUVRIR(y); EXAMINER-VOISINS(y) finsi
    sinon DÉFILER(Φ)
    finsi
  fintantque.
```

Sur le graphe de la figure 4.4, la procédure LARGEUR calcule le parcours en largeur $(1, 7, 4, 3, 6, 5, 2)$. Par un raisonnement analogue à celui fait pour la procédure PROFONDEUR, la complexité de la procédure LARGEUR est $O(n + m)$.

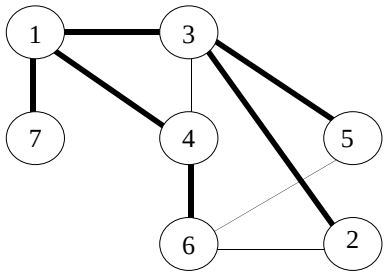

Figure 4.4: *Parcours en largeur.*

Parcours d'un graphe non orienté quelconque

Considérons le cas où le graphe G n'est pas connexe. Un parcours de G est défini comme une liste de sommets telle que :

- chaque sommet de S apparait une fois et une seule dans la liste,
- chaque sommet de la liste (sauf le premier) appartient à la bordure du sous-ensemble des sommets placés avant lui dans la liste, si toutefois cette bordure est non vide.

La condition de connexité n'est alors plus requise si les sommets de la liste en cours constituent une composante connexe de G.

Il résulte de cette définition qu'un parcours de G est une liste $L = L_1 \cdot L_2 \cdots L_p$ où les L_j sont des parcours des sous-graphes (connexes) induits par les p composantes connexes de G. Les arêtes de liaison constituent alors une famille de p arbres où chaque arbre couvre une composante connexe de G. Le parcours $(9, 7, 6, 10, 8, 1, 2, 3, 4, 5)$ est un parcours en profondeur du graphe de la figure 4.5. Les arêtes épaisses correspondent à des arbres couvrant les deux composantes connexes.

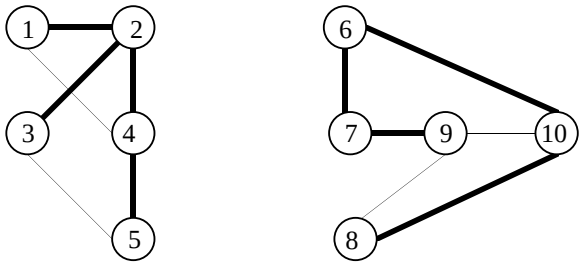

Figure 4.5: *Un graphe non orienté non connexe.*

4.4.4 Parcours d'un graphe orienté

Dans cette section nous considérons un graphe G orienté et *sans boucles*. Pour un graphe orienté, la *bordure* $\Gamma(T)$ d'une partie T de S est le sous-ensemble des sommets de $S - T$ qui sont les extrémités d'un arc dont l'origine est dans T. Si L est une liste de sommets de G, nous noterons encore abusivement $\Gamma(L)$ la bordure du support de L. Un sommet d'une liste L est *fermé* si tous ses successeurs dans G appartiennent à $\sigma(L)$, dans le cas contraire il est *ouvert*. Sur l'exemple de la figure 4.6, la bordure de $\{1, 2\}$ est $\{3, 4, 5\}$.

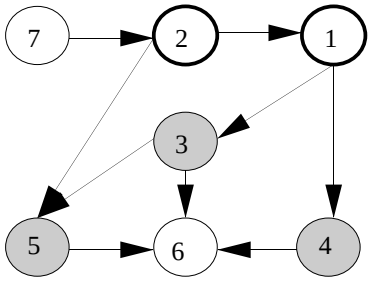

Figure 4.6: *Bordure pour un graphe orienté.*

Un parcours de G est une liste des sommets de G telle que :

- chaque sommet de S apparaît une fois et une seule dans la liste,
- chaque sommet de la liste (sauf le premier) appartient à la bordure du sous-ensemble des sommets placés avant lui dans la liste, si toutefois cette bordure est non vide.

Les propriétés des parcours des graphes non orientés se prolongent au cas orienté. La notion d'arête de liaison est remplacée par celle d'*arc de liaison*. Le graphe partiel des arcs de liaison possède en particulier la propriété suivante que nous énonçons sans démonstration :

Proposition 4.5. *Les arcs de liaison d'un parcours constituent une forêt couvrante du graphe G.* ∎

La liste L=$(4, 5, 9, 8, 7, 6, 10, 11, 1, 3, 2)$ est un parcours du graphe orienté de la figure 4.7. La forêt couvrante contient deux arborescences A_1 et A_2 (en trait plein).

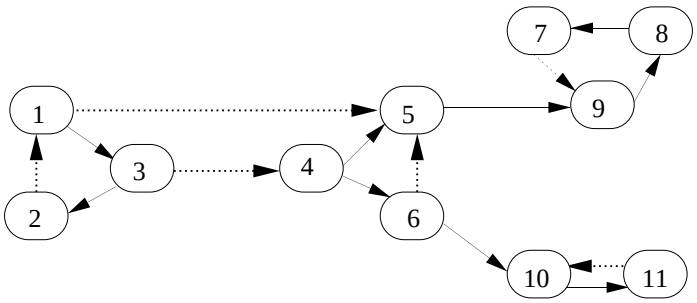

Figure 4.7: *Un graphe orienté.*

Parcours en profondeur d'un graphe orienté

Un parcours de G est dit *en profondeur* si l'origine de chaque arc de liaison est le dernier sommet ouvert déjà visité.

La procédure PARCOURS-PROFONDEUR-GRAPHE-ORIENTÉ ci-dessous calcule un parcours en profondeur en utilisant essentiellement la procédure PROFONDEUR-RÉCURSIF-ORIENTÉ, qui est l'adaptation directe de la procédure PROFONDEUR-RÉCURSIF présentée dans la section 4.4.2 pour le cas non orienté.

```
procédure PARCOURS-PROFONDEUR-GRAPHE-ORIENTÉ(G);
   tant qu'il existe un sommet de G non visité faire
      choisir un sommet non visité s;
      PROFONDEUR-RÉCURSIF-ORIENTÉ(s)
   fintantque.
```

La procédure PROFONDEUR-RÉCURSIF-ORIENTÉ s'écrit alors :

```
procédure PROFONDEUR-RÉCURSIF-ORIENTÉ(s)
  VISITER(s);
  pour tout successeur x de s faire
    si x est non visité alors
      VISITER(x); PROFONDEUR-RÉCURSIF-ORIENTÉ(x)
    finsi
  finpour.
```

Nous présentons maintenant quelques propriétés des parcours en profondeur qui seront particulièrement utiles au calcul des composantes fortement connexes.

Soit L un parcours en profondeur de G. Nous notons $\mathcal{F}$ la forêt couvrante induite par L et $r(x)$ le rang d'un sommet x dans la liste L. En dehors des arcs de $\mathcal{F}$, le parcours L permet de distinguer trois autres classes d'arcs dans G. Un arc (x, y) est *arrière* si y est un ascendant de x dans $\mathcal{F}$. Un arc (x, y) est *avant* si x est un ascendant de y dans $\mathcal{F}$. Un arc (x, y) est *transverse* si ses deux extrémités appartiennent à deux arborescences différentes, ou si x et y ont un ancêtre commun z dans $\mathcal{F}$ distinct de x et de y. Pour le parcours $(4, 5, 9, 8, 7, 6, 10, 11, 1, 3, 2)$ du graphe de la figure 4.7, les arcs $(7, 9)$, $(11, 10)$ et $(2, 1)$ sont arrière, les arcs $(6, 5)$, $(1, 5)$ et $(3, 4)$ sont transverses et il n'y a pas d'arc avant.

Le lemme suivant établit une propriété des arcs transverses.

Lemme 4.6. *Si (x, y) est un arc transverse pour un parcours en profondeur L, alors $r(y) < r(x)$.*

Preuve. Soit $L = L_1 \cdot L_2 \cdots L_q$ un parcours en profondeur et soient $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_q$ les arborescences de la forêt couvrante. Par définition de L, il n'existe pas d'arc dont l'origine soit dans $\mathcal{A}_i$ et l'extrémité dans $\mathcal{A}_j$ avec $i < j$.

Soit (x, y) un arc transverse tel que $x \in \mathcal{A}_i$ et $y \in \mathcal{A}_j$ avec $i \neq j$. On a nécessairement $i > j$ et donc $r(x) > r(y)$.

Soit (x, y) un arc transverse dont les deux extrémités x et y appartiennent à la même arborescence et soit $z \notin \{x, y\}$ l'ancêtre commun de x et y dans cette arborescence. Si $r(x) < r(y)$, lorsque le sommet x est visité, le sommet y ne l'est pas encore. Il existe donc à cette étape du parcours un chemin de sommets non visités de x à y. Donc y sera accessible *dans l'arborescence* à partir de x. D'où la contradiction. ∎

Le classement des arcs de G relativement à un parcours en profondeur L fixé est à la base d'un algorithme performant pour tester si un graphe est sans circuit. En effet, le lemme suivant montre que G possède un circuit si et seulement s'il existe au moins un arc arrière.

Lemme 4.7. *Soient G un graphe orienté et L l'un de ses parcours en profondeur. Le graphe G est sans circuit si et seulement s'il n'existe pas d'arc arrière.*

Preuve. La condition nécessaire est évidente. Soit γ un circuit de G et supposons qu'il n'existe pas d'arc arrière. D'après le lemme 4.6, le circuit γ possède au moins un arc transverse et un arc avant. Considérons alors le sommet z, origine d'un arc avant de γ, dont le rang est minimum et notons y le prédécesseur de z dans γ. Soit x un sommet de γ distinct de z. Si l'arc de γ d'origine x est transverse, alors, d'après le lemme 4.6 son rang $r(x)$ est plus grand que le rang du premier sommet, origine d'un arc avant, rencontré à partir de x sur le circuit; on a donc $r(x) > r(z)$. Si l'arc de γ d'origine x est avant, on a $r(x) > r(z)$ par définition de z. Donc, lorsque le sommet z est visité, aucun autre sommet de γ n'est encore visité. Il en résulte l'existence dans $\mathcal{F}$ d'un chemin de z à y. L'arc (y, z) est donc arrière et transverse (par définition de z). Contradiction. ∎

Pour tester si un graphe orienté est sans circuit, il suffit donc de construire un parcours en profondeur de ce graphe et de tester si les arcs de G qui n'appartiennent pas à la forêt couvrante sont «avant» ou «transverse». La complexité de cet algorithme est en fait celle du parcours, soit $O(n + m)$.

Si le graphe G est fortement connexe, la forêt couvrante du parcours se réduit à une seule arborescence couvrante. Dans le cas contraire, nous montrons que le sous-graphe de $\mathcal{F}$ induit par chaque composante fortement connexe est une arborescente couvrante de cette composante fortement connexe.

Lemme 4.8. *Soient L un parcours en profondeur de G, $\mathcal{F}$ la forêt induite par L et C une composante fortement connexe de G. Le sous-graphe de $\mathcal{F}$ induit par C est une arborescence couvrant C.*

Preuve. Soit s le premier sommet de C visité dans L. Tous les autres sommets de C sont accessibles à partir de s dans le sous-graphe de G induit par C et n'ont pas encore été visités. Ils seront donc visités après s et accessibles dans $\mathcal{F}$ à partir de s. La proposition en résulte. ∎

La proposition précédente met en évidence le premier sommet du parcours L appartenant à une composante fortement connexe donnée C. Nous appellerons ce sommet *point d'entrée* de L dans C. Nous donnons maintenant une caractérisation de ces points d'entrée qui est à la base d'un algorithme efficace pour le calcul des composantes fortement connexes d'un graphe. Nous fixons maintenant un parcours en profondeur L.

Soit x un sommet, le *rang d'attache* de x par rapport à $\mathcal{F}$ est défini par

$$\rho(x) = \min\{r(z) \mid z \in AT(x) \cup \{x\}\},$$

où $AT(x)$ est l'ensemble des sommets extrémités d'un arc arrière ou transverse dont l'origine est un descendant de x dans $\mathcal{F}$ et dont l'extrémité appartient à une composante fortement connexe dont le point d'entrée est un ascendant propre de x dans $\mathcal{F}$. Le sommet de rang $\rho(x)$ est appelé *point d'attache* du sommet x dans $\mathcal{F}$ et est noté $a(x)$.

Soit $B \subset A$ un sous-ensemble des arcs de G. Il sera commode d'écrire respectivement $x \xrightarrow{B} y$, $x \xrightarrow{(*,B)} y$, et $x \xrightarrow{(+,B)} y$ s'il existe de x à y respectivement un arc de B, un chemin dans B et un chemin non nul dans B. Si l'on note R l'ensemble des arcs arrière, T l'ensemble des arcs transverses, $X = R \cup T$ et F l'ensemble des arcs de $\mathcal{F}$, alors un sommet z appartient à $AT(x)$ si et seulement s'il existe deux sommets y et t tels que

$$x \xrightarrow{(*,F)} y \xrightarrow{X} z \xrightarrow{(*,A)} t \xrightarrow{(+,F)} x. \tag{4.1}$$

Il résulte directement de cette définition que les sommets de $AT(x)$ sont dans la même composante fortement connexe que x.

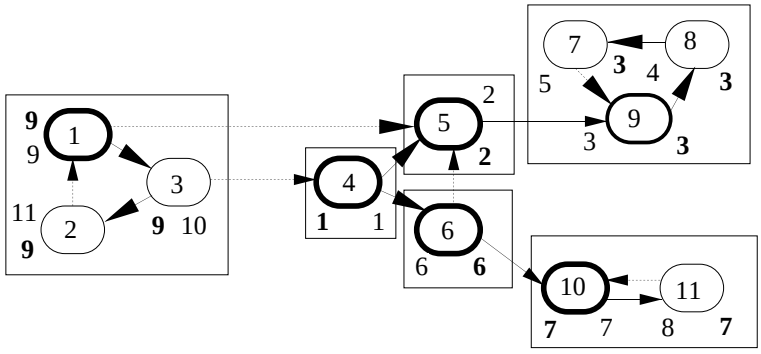

Figure 4.8: *Rangs d'attache et points d'attache.*

La figure 4.8 illustre les définitions du rang d'attache et du point d'attache, L=$(4, 5, 9, 8, 7, 6, 10, 11, 1, 3, 2)$ est un parcours en profondeur d'un graphe G possédant six composantes fortement connexes (encadrées). Les points d'entrée de ces composantes sont cerclés de noir. Le rang d'un sommet est à côté de ce sommet. Le rang d'attache d'un sommet est également inscrit en gras à côté de ce sommet.

Le lemme suivant fournit une caractérisation des points d'entrée.

Lemme 4.9. *Un sommet x est un point d'entrée si et seulement si $x = a(x)$.*

Preuve. Si x est un point d'entrée de la composante fortement connexe C, alors $AT(x) = \emptyset$. En effet, si $z \in AT(x)$ il existe y et t qui vérifient 4.1. On a donc $z \in C$ puisque $z \in AT(x)$ et $t \notin C$ puisqu'aucun ascendant propre de x dans $\mathcal{F}$ n'appartient à C. D'où la contradiction. Il en résulte que $\rho(x) = r(x)$ et que $a(x) = x$.

Réciproquement, si x appartient à la composante fortement connexe C et n'est pas son point d'entrée, nous notons e le point d'entrée de C. On a alors $e \xrightarrow{(+,F)} x$ et $x \xrightarrow{(+,A)} e$. Or comme il n'existe pas de chemin de x à e dans $\mathcal{F}$, il existe un sommet z qui n'est pas un descendant de x dans $\mathcal{F}$ tel que :

$$x \xrightarrow{(*,F)} y \xrightarrow{X} z \xrightarrow{(*,A)} e. \tag{4.2}$$

Le sommet z appartient à $AT(x)$ et nous montrons que $r(z) < r(x)$. Soit t l'ancêtre de z et x dans $\mathcal{F}$. Le sommet t est distinct de x puisque z n'est pas un descendant de x dans $\mathcal{F}$. Si $t = z$ alors z est un ascendant propre de x et $r(z) < r(x)$. Si t est distinct de z et $r(z) > r(x)$, lors de la visite de x seul le sommet x du chemin de x à z a été visité et donc z sera un descendant de x dans $\mathcal{F}$, ce qui contredit l'hypothèse. On a donc $r(z) < r(x)$ et donc $a(x) \neq x$. ∎

4.4.5 Calcul des composantes fortement connexes

Nous présentons dans cette section un algorithme efficace, dû à Tarjan, pour le calcul des composantes fortement connexes d'un graphe orienté quelconque G. Cet algorithme réalise simultanément un parcours en profondeur L du graphe G et le calcul des rangs d'attache de chaque sommet. L'implémentation de cet algorithme par la procédure CFC ci-dessous est une variante de la procédure PROFONDEUR-RÉCURSIF présentée dans la section 4.4.2. Cette variante utilise une pile Π qui empile les sommets au fur et à mesure de leur insertion dans le parcours en profondeur et dépile tous les sommets de chaque nouvelle composante fortement connexe détectée lorsque le parcours en profondeur revient sur un point d'entrée. La procédure CFC se distingue de la procédure PROFONDEUR-RÉCURSIF par les trois points suivants :

a) La notion de voisin est remplacée par celle de successeur ;

b) Lors de l'examen des successeurs du sommet x en cours dans le parcours en profondeur, deux cas sont possibles :

1. Si le successeur y examiné n'est pas déjà visité, alors le rang $r(y)$ est calculé, le rang d'attache $\theta(y)$ est initialisé à $r(y)$, le sommet y est empilé dans Π, un appel récursif est exécuté pour le sommet y et le rang d'attache $\theta(x)$ est actualisé à $\min\{\theta(x), \theta(y)\}$.
2. Si le successeur y examiné est déjà visité, le rang d'attache $\theta(x)$ est actualisé à $\min\{\theta(x), r(y)\}$.

c) Lorsque tous les successeurs d'un sommet x ont été traités, la valeur de $\theta(x)$ est le rang d'attache de x. Si $\theta(x) = r(x)$, le sommet x est un point d'entrée d'une composante fortement connexe C dont les sommets autres que x sont situés au-dessus de x dans Π. Cette pile est alors dépilée jusqu'au sommet x.

La procédure DESC(s) ci-dessous détermine dans le tableau c les composantes fortement connexes qui sont des descendants de la composante fortement connexe contenant s dans le graphe réduit de G. L'élément $c(x)$ est le numéro de la composante fortement connexe contenant x. On suppose également qu'un tableau de booléens est utilisé pour tester l'appartenance d'un sommet à la pile Π.

```
procédure DESC(s);
(1)  EMPILER(s, Π); V := V ∪ {s};
(2)  r := r + 1; r(s) := r; θ(s) := r;
(3)  pour tout successeur x de s dans G faire
(4)     si x ∉ V alors DESC(x); θ(s) := min{θ(s), θ(x)}
(5)     sinon
(6)        si x est dans Π alors θ(s) := min{θ(s), r(x)} finsi
(7)     finsi
(8)  finpour;
(9)  si θ(s) = r(s) alors
(10)    k := k + 1;
(11)    répéter z :=DÉPILER(Π); c(z) := k
(12)    jusqu'à z = s;
(13) finsi.
```

L'algorithme de calcul des composantes fortement connexes de G est implémenté par la procédure CFC(G) ci-dessous.

```
procédure CFC(G)
   V := ∅; {V est l'ensemble des sommets visités}
   r := 0; {r est le compteur pour les rangs}
   k := 0; {k numérote les composantes fortement connexes}
   PILEVIDE(Π);
   tantque V ≠ S faire
      choisir s dans S − V; DESC(s)
   fintantque.
```

Les deux propositions suivantes établissent la complexité et la validité de la procédure CFC.

Proposition 4.10. *Soit G un graphe possédant n sommets et m arcs. La complexité de la procédure* CFC *est $O(\max\{m, n\})$.*

Preuve. Pour évaluer la complexité de la procédure CFC, examinons les opérations *supplémentaires* réalisées par rapport à un simple parcours en profondeur du graphe G. Lors de chaque étape du parcours en profondeur, seules des opérations de mise à jour de complexité $O(1)$ sont introduites. En ce qui concerne la pile Π, chaque sommet est empilé et dépilé exactement une fois. Il en résulte que la complexité globale de la procédure CFC est $O(\max\{m, n\})$. ∎

Proposition 4.11. *Soit s un sommet du graphe G. La procédure* DESC(s) *détermine les composantes fortement connexes de $G^+(s)$.*

Preuve. Au cours d'un parcours en profondeur d'un graphe G, un sommet passe successivement par trois états. Tant qu'il n'a pas été visité, il est *libre*. Lorsqu'il est visité (c'est-à-dire lorsqu'il est l'extrémité d'un mouvement avant du parcours), il devient *examiné*. Lorsqu'il devient l'extrémité d'un mouvement arrière du parcours, il est *exclu*. L'ordre dans lequel les sommets sont examinés est celui du parcours en profondeur. La liste L^- associée à l'ordre d'exclusion des sommets vérifie les propriétés suivantes :

- le dernier sommet exclu d'une composante fortement connexe est son point d'entrée,
- les points d'entrée des composantes fortement connexes apparaissent dans l'ordre d'un parcours *postfixe* du graphe réduit de G.

La liste d'exclusion pour l'exemple de la figure 4.9 est $(7, 8, 9, 5, 11, 10, 6, 4)$.

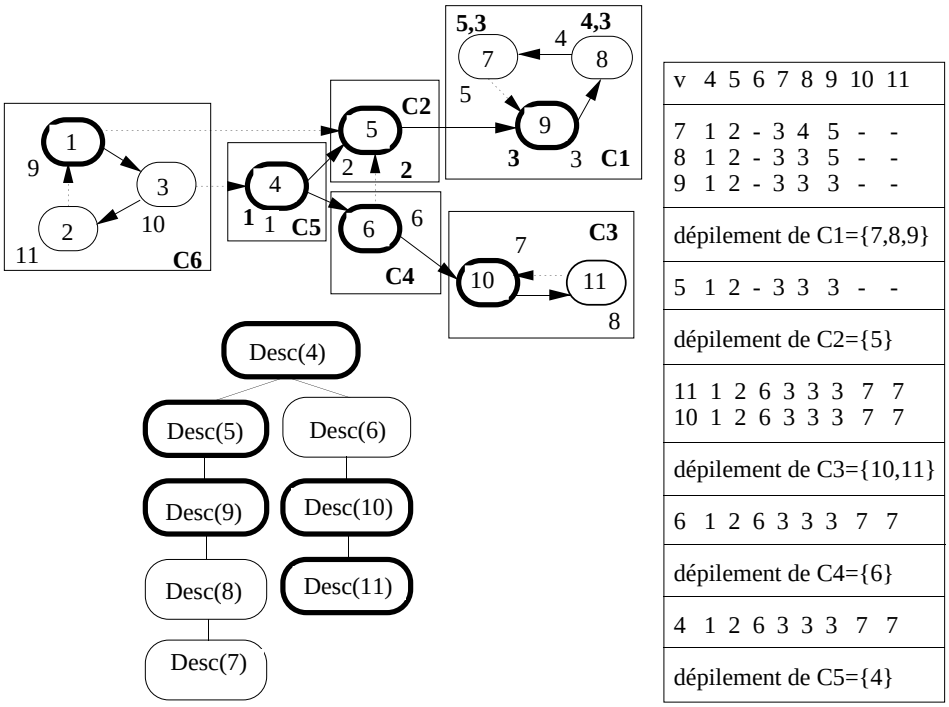

Figure 4.9: *Une exécution de* DESC.

Supposons dans un premier temps que les valeurs $\theta(x)$ calculées par la procédure DESC(s) pour tous les sommets de $G^+(s)$ soient effectivement les rangs d'attache de ces sommets. La liste d'exclusion L^- du parcours en profondeur de $G^+(s)$ correspond à l'ordre dans lequel les appels récursifs de la procédure DESC pour les sommets de $G^+(s)$ se terminent. Il résulte alors du lemme 4.9 que lorsque le nouveau sommet exclu x satisfait $\theta(x) = r(x)$, ce sommet est un point d'entrée. Donc les «macro-dépilements» de la pile sont réalisés par DESC lors des exclusions successives des points d'entrée. Or, lors de l'exclusion d'un point d'entrée x, la pile contient dans sa partie supérieure à partir du sommet x les sommets de la composante fortement connexe C de x. En effet, en raison de l'ordre postfixe des

macro-dépilements, tous les descendants propres de C dans le graphe réduit de G ont déjà été «macro-dépilés».

Nous montrons maintenant que, pour tout sommet x de $G^+(s)$, lors de la terminaison de l'appel DESC(x), la valeur calculée $\theta(x)$ est égale au rang d'attache du sommet x. Nous raisonnons par récurrence sur le nombre d'appels terminés de la procédure DESC.

Considérons la terminaison de l'appel DESC(v). Nous montrons d'abord que si le sommet w est un fils de v dans $\mathcal{F}$ et si $r(v) > \rho(w)$ alors on a nécessairement $\rho(v) \leq \rho(w)$. En effet, soit (y, x) un arc arrière ou transverse tel que $w \xrightarrow{(*,F)} y$ et $x \xrightarrow{(*,A)} e$ où x est le point d'attache de w et e est le point d'entrée de la composante fortement connexe contenant x. On a $\rho(w) = r(x)$ et donc $r(x) < r(v)$. Comme $x \in AT(v)$, on a $\rho(v) \leq \rho(w)$. Nous montrons maintenant que si l'arc (v, w) est un arc arrière ou transverse dont l'extrémité w appartient à une composante fortement connexe C non encore «macro-dépilée» et si $r(v) > r(w)$ alors on a nécessairement $\rho(v) < r(w)$. En effet, soit e le point d'entrée de C. L'appel DESC(e) n'est pas terminé lors de la terminaison de DESC(v). Le sommet e qui est un ascendant de w est aussi un ascendant de v. En effet, dans le cas contraire, $r(e) \leq r(w) < r(v)$ impliquerait que l'appel DESC(e) soit terminé avant DESC(v). Il en résulte que $w \in AT(v)$ et $\rho(v) < r(w)$.

En utilisant l'hypothèse de récurrence pour les successeurs de v qui sont des fils de v dans $\mathcal{F}$, la valeur $\theta(v)$ calculée par l'appel DESC(v) est égale à $\min\{r(v), \alpha, \beta\}$ où, si S désigne les fils de v dans $\mathcal{F}$ et T les autres successeurs de v dans G :

$$\alpha = \min\{\rho(w) \mid w \in S\} \quad \beta = \min\{r(w) \mid w \in T\}.$$

D'après ce qui précède, cette valeur correspond à celle du rang d'un sommet de $AT(v)$. Montrons que lors de la terminaison de DESC(v), on a $\theta(v) = \rho(v)$. Nous considérons à cet effet un arc (x, y) arrière ou transverse dont l'origine est un descendant de v dans $\mathcal{F}$ et l'extrémité y appartient à $AT(v)$ et nous montrons que $\theta(v) \leq r(y)$. Nous notons e le point d'entrée de la composante fortement connexe contenant y et envisageons deux cas :

- Supposons $x = v$. D'après l'hypothèse de récurrence, les composantes fortement connexes ont été correctement calculées jusqu'à la terminaison des appels de DESC antérieurs à DESC(v). Le sommet y est donc dans la pile puisque DESC(e) n'est pas terminé. Or lors de l'examen du successeur y de v dans l'appel DESC(v), la valeur $\theta(v)$ est actualisée (ligne (6)) à une valeur inférieure ou égale à $r(y)$.

- Supposons $x \neq v$. Le sommet x est alors un descendant propre de v dans $\mathcal{F}$. Si z est le fils de v dans $\mathcal{F}$ dont x est le descendant, après la terminaison de l'appel DESC(z), la valeur $\theta(v)$ est actualisée (ligne (4)) et rendue inférieure ou égale à $\rho(z)$ donc à $r(y)$.

Il en résulte que $\theta(v) = \rho(v)$. ∎

La figure 4.9 montre en a) l'arbre des appels récursifs où les sommets grisés correspondent aux points d'entrée (donc aux macro-dépilements), en b) les valeurs des rangs d'attache en cours après la terminaison de DESC(v).

4.4.6 Parcours d'une arborescence

La structure récursive des arborescences permet de définir des parcours spécifiques fondés sur d'autres règles que la propriété d'adjacence commune aux parcours étudiés jusqu'ici. Deux parcours sont très utilisés : le parcours *préfixe* pour lequel les ascendants propres d'un sommet sont placés avant ce sommet dans la liste et le parcours *postfixe* pour lequel les descendants propres d'un sommet sont placés avant ce sommet dans la liste.

Soit A une arborescence de racine r. La structure récursive de A permet d'établir simplement les algorithmes des parcours préfixe et postfixe.

```
fonction LPREF(A) :liste;
  si A = Λ alors LPREF :=()
  sinon
    L := (r);
    pour chaque fils u de r faire
      L :=L·LPREF(A(u));
    LPREF :=L
  finsi.
```

```
fonction LPOST(A) :liste;
  si A = Λ alors LPOST :=()
  sinon
    L := ();
    pour chaque fils u de r faire
      L :=L·LPOST(A(u));
    LPOST :=L · (r)
  finsi.
```

Pour l'arborescence de la figure 4.10, $(6, 7, 1, 3, 5, 4, 2)$ est le parcours préfixe et $(1, 7, 5, 3, 4, 2, 6)$ est le parcours postfixe.

Proposition 4.12. *La liste* LPREF(A) *contient chaque sommet de A une fois et une seule et tous les ascendants propres d'un sommet sont placés avant ce sommet dans la liste.*

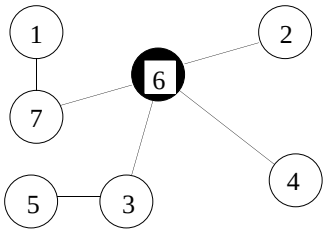

Figure 4.10: *Une arborescence.*

Preuve. (Induction sur le nombre n de sommets de A.) Si $n \leq 1$, la propriété est vraie. Soit une arborescence $A = (H, r)$ à n sommets ($n \geq 2$). Par définition, LPREF(A) est la concaténation des listes (r), LPREF($A(r_1)$), ..., LPREF($A(r_k)$). Par induction, pour tout $j \in \{1, \ldots, k\}$, LPREF($A(r_j)$) vérifie la proposition pour la sous-arborescence $A(r_j)$. La liste LPREF(A) contient donc chaque sommet de A une et une seule fois. D'autre part, soit u un sommet de A et v un ascendant propre de u. Si $v = r$ alors il est placé avant u. Sinon, par induction, v est placé avant u dans la sous-arborescence $A(r_j)$ qui contient u et la proposition est donc vérifiée. ∎

Parcours symétrique d'un arbre binaire

Les arbres binaires sont très utilisés comme structures de données permettant de manipuler efficacement les ensembles ordonnés. Le *parcours symétrique* d'un arbre binaire est une liste des sommets de l'arbre telle que tout sommet est placé dans la liste après les sommets de son sous-arbre gauche et avant les sommets de son sous-arbre droit. Une telle liste est unique puisque la définition impose la place de la racine, puis successivement la place des racines de tous les sous-arbres.

L'algorithme récursif suivant détermine le parcours symétrique noté SYM(A) d'un arbre binaire A de racine r :

```
fonction SYM(A :arbre binaire) :liste;
   si A = Λ alors SYM:= ()
   sinon
      SYM :=SYM(A_g) · (r)· SYM(A_d)
   finsi.
```

Le parcours symétrique de l'arborescence de la figure 4.11 est $(12, 2, 5, 4, 6, 3, 1, 7, 10, 11, 8, 9)$.

Proposition 4.13. *La liste* SYM(A) *est le parcours symétrique de l'arbre binaire A.*

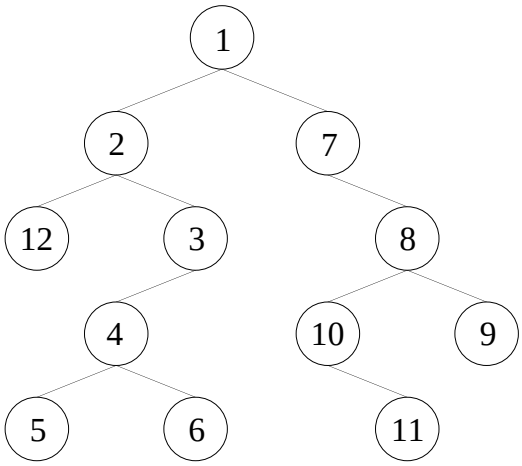

Figure 4.11: *Un arbre binaire.*

Preuve. (Induction sur le nombre n de sommets de A.) La propriété est vraie pour $n = 0$ puisque $\textsc{Sym}(\Lambda)$ est vide. Soit A un arbre binaire à $n \geq 1$ sommets. Par définition, $\textsc{Sym}(A)$ est la concaténation des listes $\textsc{Sym}(A_g)$, (r) et $\textsc{Sym}(A_d)$. Par induction la propriété est vraie pour tout sommet de A_g et de A_d et elle est vraie par construction pour la racine r de A, donc elle est vraie pour tout sommet de A. ∎

Notes

Les ouvrages sur les graphes sont nombreux. Nous avons utilisé les définitions du livre de référence :

C. Berge, *Graphes*, Gauthier-Villars (troisième édition), 1983.

D'autres livres utiles sont :

F. Harrary, *Graph Theory*, Addison-Wesley, Reading, Mass, 1969 ;
M. Gondran et M. Minoux, *Graphes et Algorithmes*, Eyrolles, 1979.

L'algorithme de calcul des composantes fortement connexes est de :

R. Tarjan, Depth First Search and Linear Graph Algorithms, *SIAM J. Computing* **1** (1972), 146–160.

Exercices

4.1. Démontrer que la matrice d'incidence sommets-arcs d'un graphe orienté sans boucles est totalement unimodulaire, c'est-à-dire que le déterminant de toute sous-matrice carrée extraite vaut 0, +1 ou −1.

4.2. Soit $G = (S, A)$ un graphe orienté. On considère la famille $\mathcal{G}$ des graphes partiels de G qui ont même fermeture transitive que G. Un graphe H de $\mathcal{G}$ est *minimal* si le graphe obtenu à partir de H en supprimant un arc quelconque n'est plus dans $\mathcal{G}$. Montrer qu'en général il existe dans $\mathcal{G}$ plusieurs graphes minimaux. Démontrer que si le graphe G est sans circuit, alors $\mathcal{G}$ ne contient qu'un seul graphe minimal (qui est appelé graphe de Hasse de G).

4.3. Ecrire un algorithme de complexité $O(\max\{m, n\})$ pour déterminer si un graphe à n sommets et m arcs est sans circuit.

4.4. Soient $G = (S, A)$ un graphe orienté et $c : A \mapsto \mathbb{N}$. Soit p un entier fixé. Pour tout couple de sommets i, j, on cherche les longueurs des p plus courts chemins (relativement à c) de i à j. On utilise pour cela le semi-anneau $(\mathbb{N}^p, \min, +, 0^p, 1^p)$ des vecteurs à p coordonnées dans $\mathbb{N}$ où les opérations $+$ et min sont définies pour deux vecteurs $a = (a_1, \ldots, a_p)$ et $b = (b_1, \ldots, b_p)$ par $\min\{a, b\} = (c_1, \ldots, c_p)$, où les c_i, $i \in \{1, \ldots, p\}$ sont les p plus petits éléments de $a_1, a_2, \ldots, a_p, b_1, b_2 \ldots, b_p$ et $a+b = c$, où les c_i, $i \in \{1, \ldots, p\}$ sont les p plus petits éléments parmi les p^2 sommes $a_i + b_j$, $i \in \{1, \ldots, p\}$, $j \in \{1, \ldots, p\}$. Montrer que le vecteur c_{ij}^n défini par la formule de récurrence 2.1 fournit les valeurs des p plus courts chemins de i à j dans G.

4.5. Soit K_n l'ensemble des matrices carrées d'ordre n à coefficients dans un semi-anneau complet K. Montrer comment munir K_n d'une structure de semi-anneau complet. On définit pour $M \in K_n$ la matrice M^* par :

$$M^* = I \oplus M \oplus M^2 \ldots \oplus M^h \oplus \ldots .$$

Soit $G = (S, A)$ un graphe à n sommets, soit $c : A \mapsto K$ une fonction coût et soit $M = m_{ij}$ la matrice définie par :

$$m_{ij} = \begin{cases} c((i,j)) & \text{si } (i,j) \in A \\ 0 & \text{sinon.} \end{cases}$$

Démontrer que :

$$m_{ij}^* = c_{ij} \quad \text{où} \quad c_{ij} = \bigoplus_{\gamma \in \Gamma_{ij}} c(\gamma).$$

4.6. Montrer que pour $\mathcal{R} = \mathbb{R} \cup \{+\infty, -\infty\}$, $(\mathcal{R}, \min, \max, +\infty, -\infty)$ est un semi-anneau complet. Soit $G = (S, A)$ un graphe non orienté, $c : A \mapsto \mathbb{R}$ une fonction coût et soit

$$c_{ij} = \min_{\gamma \in \Gamma_{ij}} c(\gamma)$$

où Γ_{ij} est l'ensemble des chaînes élémentaires de i à j. Soit

$$U = \{(i, j) \in A \mid c_{ij} = c((i, j))\}.$$

Montrer que le sous-graphe (S, U) est un arbre couvrant de coût minimal (au sens du chapitre 7).

4.7. Démontrer la proposition 2.2.

4.8. Soit $G = (S, A)$ un graphe orienté connexe à n sommets et m arcs. On appelle *cycle élémentaire* de G une suite d'arcs de G telle que deux arcs consécutifs quelconques de la suite (y compris le dernier et le premier) aient une extrémité commune et que ces extrémités communes successives soient toutes distinctes. On associe à un cycle élémentaire de G le vecteur de $\{0, 1, -1\}^m$ pour lequel la coordonnée associée à l'arc u vaut 0 si le cycle n'emprunte pas l'arc u, 1 si le cycle emprunte l'arc u dans le sens de u et -1 si le cycle emprunte l'arc u dans le sens opposé à u. Démontrer que le sous-espace vectoriel de $\mathbb{R}^m$ engendré par les cycles élémentaires de G est de dimension $m - n + 1$. En déduire que si G possède p composantes connexes, le nombre maximum de cycles « indépendants » de G est $m - n + p$.

4.9. Montrer que l'on peut utiliser un parcours en largeur d'un graphe non orienté à partir d'un sommet s pour déterminer les longueurs des plus courtes chaînes de s à tous les autres sommets du graphe.

4.10. Soit G un graphe non orienté connexe et soit s l'un de ses sommets. Montrer que l'arborescence des appels récursifs de la procédure PROFONDEUR-RÉCURSIF(s) s'identifie avec une arborescence couvrante de G de racine s. Montrer que si $\{x, y\}$ est une arête n'appartenant pas à l'arborescence, l'un des deux sommets x ou y est ascendant de l'autre dans l'arborescence. Montrer que la complexité en temps de la procédure est $O(\max\{n, m\})$, où n est le nombre de sommets et m le nombre d'arêtes du graphe G.