2.1. Parcours en profondeur du graphe et de son inverse

2.1.1. Principe et algorithme

Soit G un graphe orienté. L'algorithme utilise deux fois le parcours en profondeur, la première fois sur G et la deuxième fois sur le graphe G^{-1} obtenu à partir de G en inversant le sens de tous les arcs. Le principe de l'algorithme est le suivant. On effectue un parcours en profondeur de G, en construisant une liste L des sommets en ordre suffixe, à l'aide de la procédure *profsuf*. On effectue ensuite un parcours en profondeur sur G^{-1}, en appelant la procédure *parcours* pour le dernier sommet de la liste L, c'est-à-dire, le sommet ayant le numéro le plus élevé dans la numérotation suffixe de G. Si tous les sommets ne sont pas marqués à la fin de cet appel, on recommence l'exécution de la procédure *parcours* avec le dernier sommet non marqué de L, et ainsi de suite, jusqu'à ce que tous les sommets soient marqués. Les numéros suffixes (ordre pour G) des racines des arborescences de la forêt couvrante de G^{-1} sont donc en ordre décroissant. Les sous-graphes de G engendrés par les sommets des arborescences de la forêt couvrante de G^{-1} ainsi obtenue, constituent les composantes fortement connexes de G.

Dans la procédure qui suit, on associe à chaque sommet le numéro de la composante fortement connexe à laquelle il appartient. On manipule le graphe G^{-1} à l'aide des opérations *$d^{\circ -}$_ de_ dans_* et *_ ème-pred-de_ dans_*, effectuées sur G. On représente la liste L dans un tableau, si bien qu'on peut facilement la parcourir de la fin vers le début.

```
procedure comp-fort-connexe1(G : Graphe; var comp : array [1..n] of integer);
{Après exécution de la procédure comp-fort-connexe1, comp[i] contient
le numéro de la composante fortement connexe du sommet i}
type LISTE = array[1..n] of integer;
var marque : array[1..n] of boolean; i,h,k : integer; L : LISTE;
procedure profsuf (x : integer; G : Graphe; var M : array [1..n] of boolean;
                   var L : LISTE; var h : integer);
{parcours en profondeur du graphe G et construction de la liste L des sommets
de G en ordre suffixe; x est un sommet; h est le nombre de sommets déjà visités}
var j,y : integer;
{y est un sommet}
begin
    M[x] := true;
    for j := 1 to d°+ de x dans G do begin
          y := j ème-succ-de x dans G;
          if not M[y] then profsuf(y,G,M,L,h)
    end;
    {traitement du sommet x à la dernière rencontre :}
    h := h+1; L[h] := x
end profsuf;
```



```
procedure parcours (x : integer; G : Graphe; var M : array[1..n] of boolean;
                    var cp : array [1..n] of integer; k : integer);
{parcours en profondeur du graphe inverse de G et remplissage du tableau cp
par la valeur k, pour tous les sommets de l'arborescence; x est un sommet}
var y,i : integer;
{y est un sommet}
begin
    M[x] := true;
    cp[x] := k;
    for i := 1 to d°- de x dans G do begin
          y := i ème-pred-de x dans G;
          if not M[y] then parcours(y,G,M,cp,k)
    end
end parcours;

begin
    {initialisations :}
    for i := 1 to n do begin comp[i] := 0; marque[i] := false end;
    k := 0;
    {k est le nombre de composantes fortement connexes détectées jusque-là}
    for h := 1 to n do L[h] := 0;
    h := 0;

    {construction de la liste L des sommets en ordre suffixe de G :}
    for i := 1 to n do if not marque[i] then profsuf(i,G,marque,L,h);
    {parcours de G^-1 :}
    for i := 1 to n do marque[i] := false;
    for i := n downto 1 do if not marque[L[i]] then begin
          k := k+1;
          parcours(L[i],G, marque, comp, k)
    end
end comp-fort-connexe1;
```

2.1.2. Exemple

Considérons le graphe G de la figure 16.

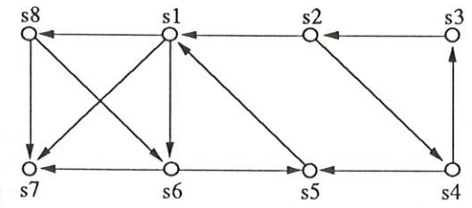

Figure 16. Graphe orienté G.

L'exécution de l'algorithme sur G donne les résultats suivants (on prend toujours la convention que les sommets et les successeurs d'un sommet sont choisis par ordre de numéros croissants).

La forêt couvrante associée au parcours en profondeur de G est dessinée à la figure 17; elle permet d'obtenir facilement la liste L des sommets en ordre suffixe.

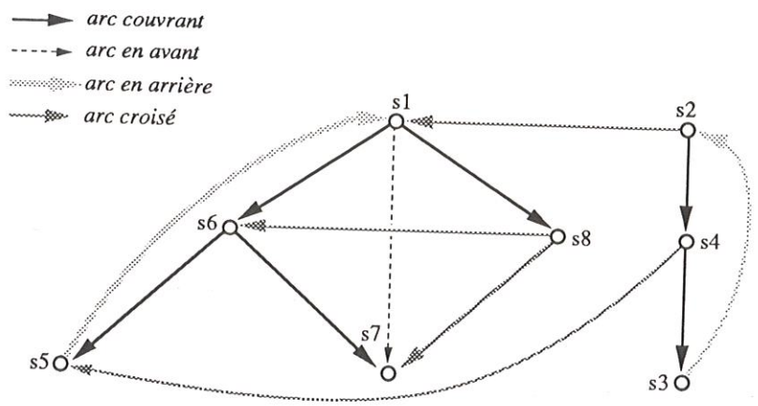

Figure 17. Forêt couvrante de G.

L :

5	7	6	8	1	3	4	2
1	2	3	4	5	6	7	8

Le graphe G^{-1} est donné à la figure 18.

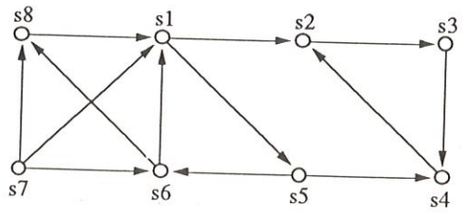

Figure 18. Graphe orienté G^{-1}.

On effectue le parcours en profondeur de G^{-1} en commençant par le sommet $s2$ car c'est le sommet de numéro suffixe le plus élevé dans la liste L. Après avoir obtenu l'arborescence de racine $s2$ (qui contient $s3$ et $s4$), on repart avec $s1$ qui est, parmi les sommets non encore marqués, celui qui a le numéro suffixe le plus élevé.

La forêt couvrante associée au parcours en profondeur de G^{-1}, effectué en lisant la liste L de droite à gauche, est dessinée à la figure 19.

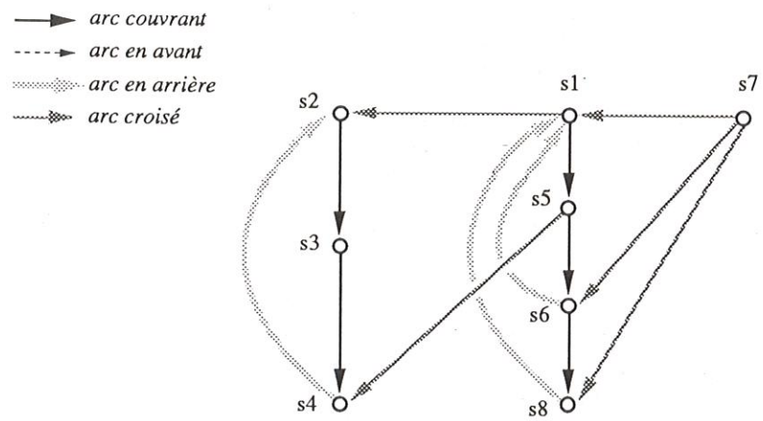

Figure 19. Forêt couvrante de G^{-1}.

Le graphe G a trois composantes fortement connexes.

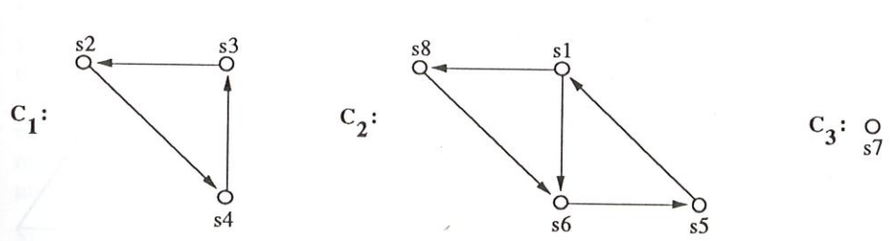

Figure 20. Composantes fortement connexes de G.

On obtient le tableau *comp* suivant :

comp :

2	1	1	1	2	2	3	2
1	2	3	4	5	6	7	8

2.1.3. Justification

On va montrer que deux sommets x et z appartiennent à la même composante fortement connexe de G si, et seulement si, ils appartiennent à une même arborescence de la forêt couvrante de G^{-1}.

• On suppose d'abord que x et z sont dans la même composante fortement connexe de G; ils sont donc dans la même composante fortement connexe de G^{-1}. Or, deux sommets appartenant à deux arborescences distinctes de la forêt couvrante de G^{-1} ne peuvent être dans la même composante fortement connexe de G^{-1}, car les arcs croisés vont de la droite vers la gauche. Par suite, x et z sont dans la même arborescence de la forêt couvrante de G^{-1}.

• Réciproquement, soit x et z deux sommets d'une même arborescence de la forêt couvrante de G^{-1}. Soit y la racine de cette arborescence. On va d'abord montrer que x et y sont dans une même composante fortement connexe de G.

Comme x appartient à l'arborescence de racine y, il existe un chemin de y vers x dans G^{-1}, et par suite, il existe un chemin de x vers y dans G.

D'autre part, si x appartient à l'arborescence de racine y dans la forêt couvrante de G^{-1}, c'est que le sommet x n'est pas marqué lorsque le parcours en profondeur de G^{-1} marque y; mais alors y apparaît après x dans le parcours suffixe de G. Comme il existe un chemin de x vers y dans G, il est exclu que x et y soient dans des arborescences distinctes de la forêt couvrante de G. Il reste deux possibilités pour x et y dans la forêt couvrante de G, dessinées à la figure 21.

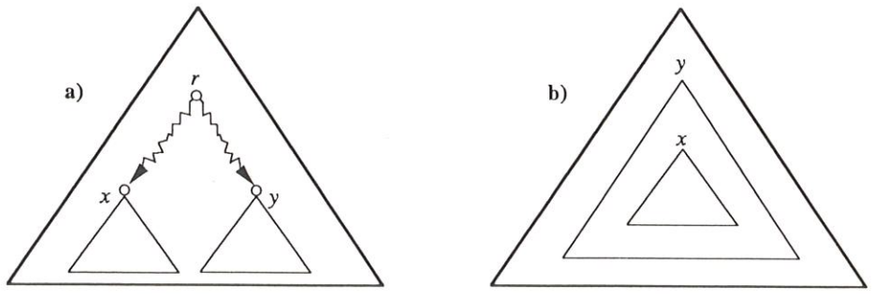

Figure 21. Forêt couvrante de G : y est après x dans l'ordre suffixe de G et il existe un chemin de x vers y dans G.

On va montrer que le premier cas est impossible. Rappelons qu'il existe un chemin de x vers y dans G. Si y n'est pas dans la sous-arborescence de racine x, c'est qu'on a rencontré un sommet du chemin de x vers y, qui était déjà marqué avant le parcours de cette arborescence (cf figure 22). Ce chemin contient donc un sommet v qui est un ancêtre commun à x et y et qui est après x et y dans l'ordre suffixe de la forêt couvrante de G. Si un tel sommet v existe, y ne peut pas être racine de l'arborescence de G^{-1} qui contient x, telle qu'elle est construite par l'algorithme. En effet, si dans cette forêt couvrante de G^{-1}, x est dans l'arborescence de racine y, c'est que x n'est pas accessible dans G^{-1} depuis un sommet v rencontré dans l'ordre suffixe de la forêt couvrante de G, après y, et donc il n'y a pas de chemin de x vers v dans G. Ce premier cas est donc impossible.

Seul reste alors le cas où x appartient à la sous-arborescence de racine y dans la forêt couvrante de G : il existe donc un chemin de y vers x dans G. En conséquence, x et y appartiennent à une même composante fortement connexe de G.

On montre de même que y et z appartiennent à une même composante fortement connexe de G. Il en résulte que x et z sont dans la même composante fortement connexe de G.

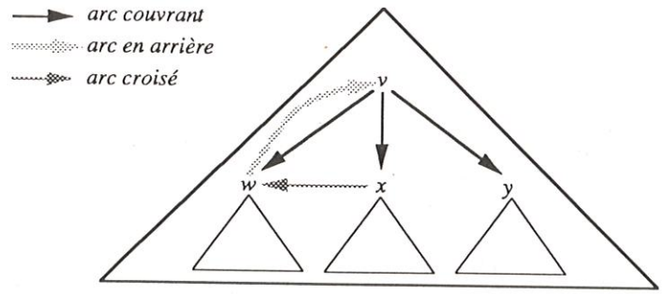

Figure 22. Chemin de x vers y dans G.

2.1.4. Analyse de la complexité

Soit n le nombre de sommets de G (resp. G^{-1}), et soit p le nombre d'arcs de G (resp. G^{-1}). Si on considère une représentation de G et de G^{-1} par des listes d'adjacence, les parcours en profondeur réalisés sur G et sur G^{-1} ont une complexité en $\Theta(\max(n,p))$. Avec cette représentation des graphes, la construction de G^{-1} s'effectue avec la même complexité en temps et demande $\Theta(n+p)$ places. La recherche des sommets des différentes composantes connexes nécessite dans ce cas un temps $\Theta(\max(n,p))$.

Si on choisit pour G (et G^{-1}) une représentation par matrices d'adjacence, la complexité en temps de l'algorithme est en $\Theta(n^2)$, comme celle du parcours en profondeur, et la construction de G^{-1} est inutile, puisqu'on peut utiliser la matrice de G en échangeant les indices. Cette représentation est donc intéressante si on a des graphes avec beaucoup d'arcs.

2.2. Algorithme de Tarjan

2.2.1. Principe

L'algorithme de recherche des composantes fortement connexes présenté maintenant, effectue un seul parcours en profondeur et en a la complexité. Comme l'algorithme précédent, il exploite les propriétés de la forêt couvrante associée au parcours en profondeur, plus précisément ici, les propriétés des arcs en arrière. Cet algorithme, dit de Tarjan, utilise plusieurs propriétés qui sont données ci-dessous.

Propriétés élémentaires

1) Soient x et y deux sommets qui sont dans une même composante fortement connexe. Soit z un sommet qui appartient à un chemin de x vers y, alors z est dans la même composante fortement connexe que x et y.

2) Soient x et y deux sommets tels qu'il y a un chemin de x vers y. Si dans un parcours en profondeur, on marque x alors que y n'est pas encore marqué, y va être dans la même arborescence que x.

Propriété 1 : Soit $G = < S, A >$ un graphe orienté. Soit $C_i =< S_i, A_i >$ une composante fortement connexe de $G, S_i \subseteq S, A_i \subseteq A$. Soit B_i l'ensemble des arcs de A_i qui sont des arcs couvrants de la forêt couvrante associée au parcours en profondeur de G. Alors $G_i =< S_i, B_i >$ est une arborescence.

Preuve : Il est clair que le graphe non orienté sous-jacent à G_i est sans cycle car il provient d'arcs appartenant à une arborescence (cf. la définition d'une arborescence au chapitre 8). Soit r_i le premier sommet de la composante fortement connexe C_i qui est rencontré lors du parcours en profondeur. Tous les sommets de C_i vont être marqués à partir de r_i et sont donc dans l'arborescence A_i de racine r_i. Soit x un sommet de C_i ; tout sommet sur le chemin de r_i vers x dans A_i est dans C_i (car C_i est une composante fortement connexe). Le graphe G_i est donc une arborescence de racine r_i (c'est l'arborescence A_i dans laquelle on a éventuellement supprimé une ou plusieurs sous-arborescences (cf. figure 23)). □

On appelle ***racine de la composante fortement connexe*** C_i, la racine de l'arborescence G_i associée à C_i, comme dans la propriété 1. C'est le premier sommet de la composante fortement connexe rencontré dans l'ordre préfixe.

On numérote en ordre suffixe les racines des m composantes fortement connexes : $r_1, r_2, ..., r_m$. Il est important de remarquer que si i est inférieur à j, alors soit r_i est à gauche de r_j, soit r_i est un descendant de r_j.

Propriété 2 : Pour tout i, les sommets de la composante fortement connexe C_i sont les descendants de r_i dans la forêt couvrante, qui ne sont pas dans $C_1, ..., C_{i-1}$.

Preuve : On a vu que tout sommet de la composante fortement connexe C_i est dans l'arborescence de racine r_i. Considérons l'un des descendants x de la racine r_i dans la forêt couvrante. Si x n'appartient pas à C_i , alors x appartient à un certain C_j dont la racine r_j appartient, elle aussi, à l'arborescence de racine r_i. Comme les racines sont numérotées en ordre suffixe, les sommets,

descendants de r_i, qui ne sont pas dans C_i, sont donc dans C_j, avec j inférieur à i. □

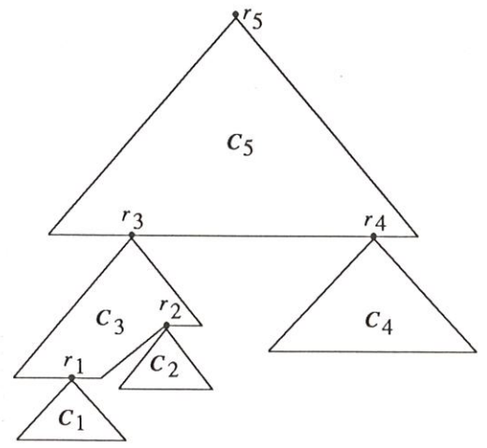

Figure 23. Composantes fortement connexes et leurs racines.

Le principe de l'algorithme de Tarjan est le suivant. Les racines r_i étant considérées en ordre suffixe, on obtient les sommets de la composante fortement connexe C_i en prenant les sommets de l'arborescence de racine r_i qui n'ont pas été pris en compte dans les arborescences précédentes.

Pour décrire l'algorithme, on introduit deux fonctions sur les sommets : la fonction *prefixe* associe à tout sommet sa numérotation dans le parcours préfixe de la forêt couvrante associée à G. La fonction *retour* associe au sommet x le numéro préfixe d'un sommet de la composante fortement connexe de x, rencontré avant x, s'il en existe, ou bien de x lui-même.

Dans quels cas trouve-t-on un sommet z de la composante fortement connexe de x tel que *prefixe*(z) est inférieur à *prefixe*(x) ? Un tel sommet doit être accessible depuis x et situé au-dessus de x ou à gauche de x. Pour accéder à ce sommet z, on doit prendre un chemin qui commence par une suite (éventuellement vide) d'arcs couvrants, puis qui continue soit par un arc en arrière, soit par un arc croisé, et qui se poursuit peut-être par d'autres arcs.

Pour savoir s'il existe un sommet z de la composante fortement connexe de x tel que *prefixe*(z) est inférieur à *prefixe*(x), il suffit donc d'examiner si on peut trouver l'un des cas présentés dans la figure 24.

Dans le cas a), il est clair que *prefixe*(z) est inférieur à *prefixe*(x) et que z et x sont dans la même composante fortement connexe.

Dans le cas b), il faut supposer de plus que *prefixe*(z) est inférieur à *prefixe*(x). Si z et x sont dans la même composante fortement connexe, alors x n'est pas le premier sommet de sa composante fortement connexe. Sinon c'est que z appartient à une composante fortement connexe dont la racine r_z a un numéro suffixe plus petit que le numéro suffixe de la racine r_x de la composante fortement connexe de x (car *prefixe*(z) est inférieur à *prefixe*(x)) et donc z a déjà été considéré dans une composante fortement connexe précédente.

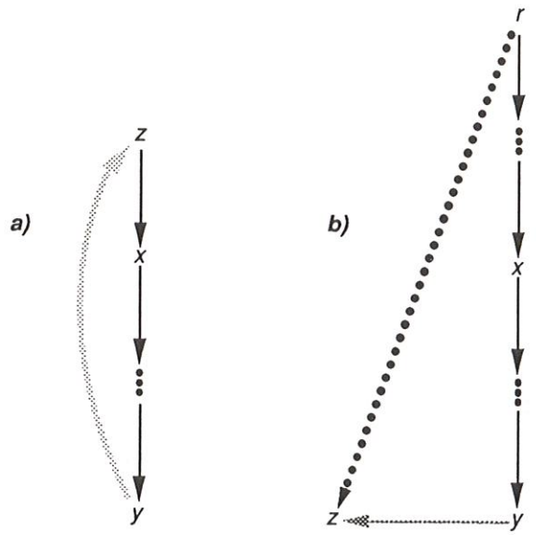

Figure 24. *prefixe*(z) < *prefixe*(x).

On est donc amené à donner la définition suivante de la fonction *retour* :

$$retour(x) = \min\{prefixe(x), prefixe(z), retour(y)\},$$

où le minimum est calculé sur tous les sommets z tels que $x \to z$ est un arc en arrière, ou un arc croisé (si la racine de la composante fortement connexe de z est un ancêtre de x), et sur tous les sommets y descendants de x dans la forêt couvrante.

Notons que la valeur *prefixe*(x) correspond au cas où il n'y a pas de sommet z tel que *prefixe*(z) est inférieur à *prefixe*(x); la valeur *prefixe*(z) avec $x \to z$ arc en arrière, ou arc croisé, correspond au cas où le chemin ne comprend pas d'arcs couvrants; et la valeur *retour*(y) correspond au cas où le chemin comprend au moins un arc couvrant $x \to y$ et où l'on se ramène à la recherche d'un chemin convenable à partir de y.

2.2.2. Algorithme

L'évaluation de *retour*(x) se fait de façon récursive en effectuant un parcours en profondeur du graphe à partir du sommet x. On empile les sommets au fur et à mesure qu'on les rencontre (ils sont donc en ordre préfixe). Lorsqu'on trouve un sommet y dont on peut évaluer la valeur de retour (on a calculé la valeur de retour de tous ses descendants), et tel que *retour*(y) = *prefixe*(y) alors y est la racine r_i d'une composante fortement connexe C_i (voir la justification plus loin). On peut obtenir tous les sommets de C_i en dépilant les sommets empilés jusqu'à r_i, r_i compris. Les sommets dépilés reçoivent alors la valeur $n+1$ en numérotation préfixe, si le graphe a n sommets, afin qu'ils ne soient plus considérés lors de la recherche des autres composantes fortement connexes.

Dans l'algorithme qui suit (voir page suivante), on utilise les opérations sur les piles, et les opérations sur les graphes spécifiées au chapitre 8. On stocke dans le tableau d'entiers *comp*, à l'indice i, le numéro de la composante fortement connexe du sommet i. On stocke dans *pref*[i] le numéro de i dans la numérotation préfixe, et dans *ret*[i] le sommet *retour*(i), où *pref* et *ret* sont des tableaux d'entiers de taille n.

2.2.3. Exemple

On reprend le graphe G de la figure 16, dont la forêt couvrante est dessinée à la figure 17. A côté de chaque sommet x, on a indiqué sur la figure 25 la valeur de *ret*[x] en éclairé, et la valeur de *pref*[x] en italique. (On n'a pas tenu compte des instructions *pref*[x] := $n+1$.)

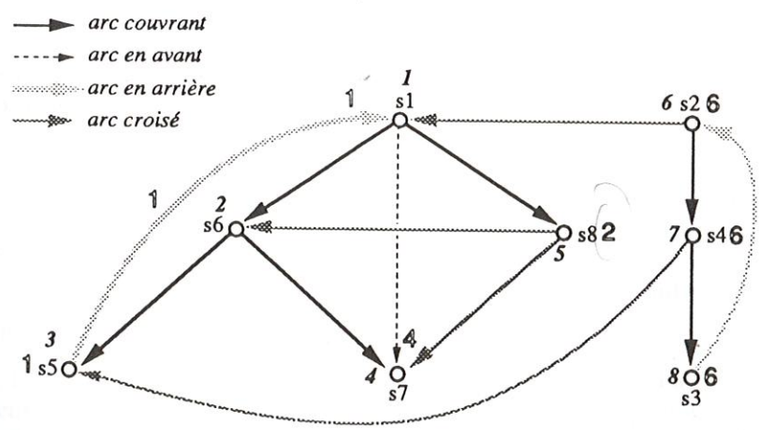

Figure 25. Forêt couvrante de G avec numérotations préfixe et de retour.

```
Programme principal :
{Le graphe G a n sommets ; cpt, m et i sont des variables de type integer ;
P est une variable de type Pile}
cpt := 0 ; P := pile-vide ; m := 0 ;
{m désigne le nombre de composantes fortement connexes déjà trouvées}
for i := 1 to n do pref[i] := 0 ;
for i := 1 to n do
    if pref[i] = 0 then comp-fort-connexe2(i, G, cpt, m, P, pref, ret, comp) ;

procedure comp-fort-connexe2(x : integer ; G : Graphe ; var cpt, m : integer ;
var P : Pile ; var pref, ret, comp : array[1..n] of integer) ;
{Après exécution de la procédure comp-fort-connexe1, comp[i] contient le
numéro de la composante fortement connexe du sommet i ; x est un sommet}
var y, j, min : integer ;
{y est un sommet et min est la valeur provisoire de retour(x)}

begin
    {numérotation préfixe de x :}
    cpt := cpt + 1 ; pref[x] := cpt ;
    min := cpt ; {ret[x] est inférieur ou égal à pref[x]}
    P := empiler(P, x) ;
    for j :=1 to d^o+ de x dans G do begin
            y := j ème-succ-de x dans G ;
            if pref[y] = 0 then begin {première rencontre de y}
                    comp-fort-connexe2(y, G, cpt, m, P, pref, ret, comp) ;
                    if ret[y] < min then min := ret[y]
            end
            else {y a déjà été rencontré : x → y est un arc en arrière
                ou un arc croisé}
                    if pref[y] < min then min := pref[y]
                            {si pref[y] = n + 1, c'est que le sommet y a déjà été
                            mis dans une autre composante fortement connexe}
    end ;
    ret[x] := min ;
    if ret[x] = pref[x] then begin
            {on a obtenu une composante fortement connexe}
            m := m + 1 ;
            while sommet(P) <> x do begin
                    y := sommet(P) ; comp[y] := m ;
                    pref[y] := n + 1 ; P := dépiler (P)
            end ;
            pref[x] := n + 1 ; comp[x] := m ; P := dépiler(P)
    end
end comp-fort-connexe2 ;
```


Dans le parcours en profondeur de $G, s7$ est le premier sommet rencontré dont on connaisse la valeur de retour et qui soit tel que cette valeur de retour soit égale à sa numérotation préfixe. Comme $s7$ n'a pas de descendant, il est l'unique sommet d'une composante fortement connexe dont il est la racine.

Ensuite, on rencontre le sommet $s1$ qui est tel que *ret*[1] = *pref*[1]. Le sommet $s1$ et ceux de ses descendants dans la forêt qui n'ont pas été déjà mis dans une composante fortement connexe, constituent une autre composante fortement connexe de racine $s1 : s1, s6, s5$ et $s8$ sont dans la même composante.

Enfin, on rencontre le sommet $s2$ tel que *ret*[2] = *pref*[2]. Les sommets $s2, s4$ et $s3$ engendrent la troisième composante fortement connexe de racine $s2$. On retrouve les composantes fortement connexes données à la figure 20.

2.2.4. Justification de l'algorithme

On va montrer qu'un sommet x est racine d'une composante fortement connexe si, et seulement si, *retour*(x) = *prefixe*(x).

1) On suppose que x est racine d'une composante fortement connexe C_i. Par définition de la fonction *retour*, *retour*$(x) \leq$ *prefixe*(x). Raisonnons par l'absurde, en supposant que l'inégalité est stricte. Il existe alors des sommets y, z et r tels que :

a) y est descendant de x,
b) z est atteint à partir de y par un arc croisé ou un arc en arrière,
c) r est la racine de la composante fortement connexe de z,
d) r est un ancêtre de x, racine de C_i,
e) *prefixe*$(z) <$ *prefixe*(x).

On en déduit que :

prefixe$(r) \leq$ *prefixe*(z) d'après c).

D'où : *prefixe*$(r) <$ *prefixe*(x) d'après e).

Par suite, r est un ancêtre de x, distinct de x d'après d).

Il existe donc un chemin de r vers x. Mais il y a aussi un chemin de x vers z. Comme r et z sont dans la même composante fortement connexe, il y a aussi un chemin de z vers r. Donc x et r sont tous les deux dans C_i ; or, *prefixe*(r) étant strictement inférieur à *prefixe*$(x), x$ ne peut pas être racine de C_i, d'où la contradiction.

2) On suppose que *retour*(x) = *prefixe*(x). Raisonnons par l'absurde en supposant que x n'est pas la racine de la composante fortement connexe C_i.

Soit r_i la racine de C_i : r_i est un ancêtre de x, distinct de x. Comme x et r_i sont tous deux dans C_i, il existe un chemin CH de x vers r_i. Considérons le premier arc de CH qui aille d'un sommet y descendant de x, vers un sommet z qui ne soit pas un descendant de x : cet arc est soit un arc croisé, soit un arc en arrière. Dans