MA351 : Introduction à la théorie des graphes

Accessibilité dans les graphes orientés

Yann Kieffer

ESISAR

Accessibilité dans les graphes orientés : enjeux

Définition

On dit que y est *accessible* depuis x dans $\overrightarrow{G} = (V, A)$ lorsqu'il existe un chemin dans $\overrightarrow{G}$ de x à y.

Cette définition est très proche de la notion de chemin ; mais c'est bien une définition distincte, qui définit une relation (non symétrique) entre les sommets.

Pourquoi s'y intéresser ? Parce qu'elle permet d'introduire toute une démarche :

- une notion de problème algorithmique ;
- la réponse au problème sera un algorithme ;
- les réponses de l'algorithme pourront être vérifiées de manière autonome à l'aide de certificats ;
- le tout sera prouvé correct, à l'aide de preuves au sens mathématique du terme.

Accessibilité : deux formulations de la problématique

Problème : accessibilité de s à t

Etant donné un graphe orienté $\overrightarrow{G} = (V, A)$, et deux sommets s et t de $\overrightarrow{G}$, t est-il accessible depuis s dans $\overrightarrow{G}$?

Il est apparent que la réponse à cette question sera nécessairement un algorithme. Son entrée sera le graphe $\overrightarrow{G}$, et les deux sommets s et t ; sa sortie sera une réponse OUI ou NON.
Un moyen pour résoudre le problème ci-dessus est de résoudre le problème apparemment plus difficile ci-dessous :

Problème : sommets accessibles depuis s

Etant donné un graphe orienté $\overrightarrow{G} = (V, A)$, et un sommet s de $\overrightarrow{G}$, quels sont les sommets de $\overrightarrow{G}$ accessibles depuis s ?

A nouveau, la réponse attendue à ce problème est un algorithme.
L'entrée est maintenant un graphe et un sommet ; la sortie attendue est un ensemble de sommets.
Nous allons résoudre ces deux problèmes, en commençant par le plus difficile.

Observations concernant l'accessibilité

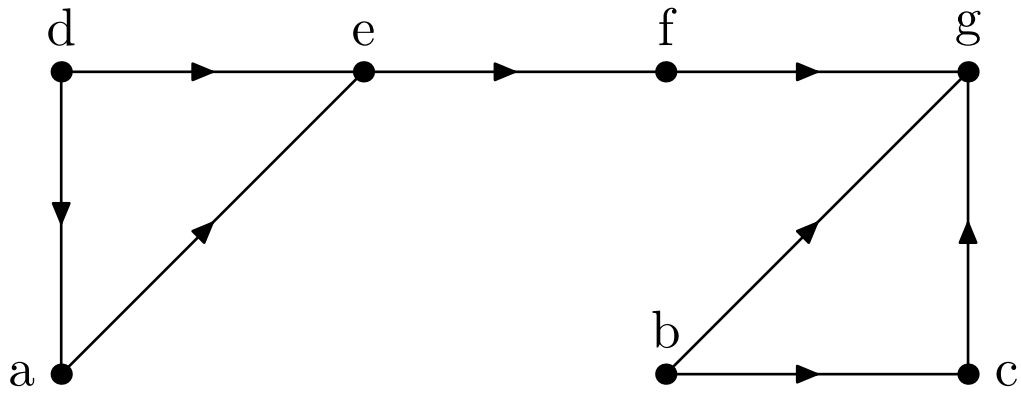
Dans toute la suite, nous supposons donné un graphe $\overrightarrow{G} = (V, A)$. Les notations s, t, x, y représentent des sommets de $\overrightarrow{G}$.

- Selon la définition, tout sommet est accessible depuis lui-même.
- Si x est accessible depuis s, et si $\overrightarrow{xy}$ est un arc, alors y est accessible depuis s.

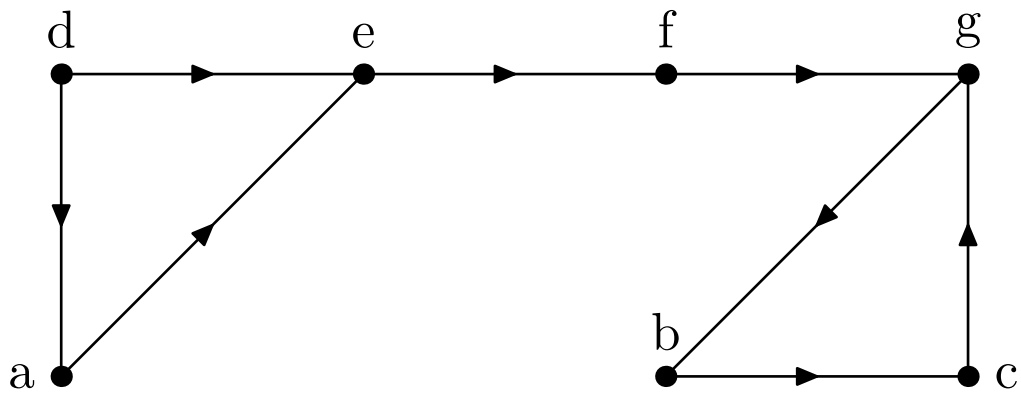
Sur la base de ces deux observations, il est facile d'écrire un algorithme récursif pour déterminer les sommets accessibles depuis s.
Voici un tel algorithme :

*Accessibles*1($\overrightarrow{G} = (V, A)$, $x \in V$) :
- $M \leftarrow \{x\}$
- Pour y dans $N^+(x)$, faire :
  - $R \leftarrow$ *Accessibles*1($\overrightarrow{G}, y$)
  - $M \leftarrow M \cup R$
- **Retourner** M

Appliquons cet algorithme sur quelques exemples



Depuis c ; depuis e ; depuis d.



Depuis f.

Quelques constats

- Un algorithme qui finit, c'est mieux qu'un algorithme qui ne s'arête jamais !
- Inutile de faire le même travail plusieurs fois

Comment savoir qu'un calcul a déjà été fait ?

Version non redondante

Voici une seconde proposition d'algorithme pour le calcul de l'ensemble des sommets accessibles depuis un sommet du graphe.

Accessibles2$(\overrightarrow{G} = (V, A),\ x \in V, M \subseteq V)$:
- Pour y dans $N^+(x)$, faire :
  - Si $y \notin M$, faire :
    - $M \leftarrow M \cup \{y\}$
    - $M \leftarrow$ *Accessibles2*$(\overrightarrow{G}, y, M)$
- **Retourner** M

Quels arguments doit-on donner au moment de l'appel de cet algorithme ?

Où sont les chemins ?

Par définition, un sommet accessible depuis s, est un sommet pour lequel il existe un chemin de s à ce sommet.
Il est bon de connaître l'ensemble des sommets accessibles depuis s. Mais peut-être serait-il mieux encore de connaître les chemins qui mènent de s à chacun de ces sommets !
Pour cela, nous enrichissons notre algorithme :

Accessibles3$(\overrightarrow{G} = (V, A),\ x \in V, M \subseteq V, o \subseteq V \times V)$: 1
– Pour y dans $N^+(x)$, faire : 2
  • Si $y \notin M$, faire : 3
    * $M \leftarrow M \cup \{y\}$ 4
    * $o \leftarrow o \cup \{(y, x)\}$ 5
    * $(M, o) \leftarrow$ *Accessibles3*$(\overrightarrow{G}, y, M, o)$ 6
– **Retourner** (M, o) 7

Quels arguments passer au moment de l'appel de cette routine ?
Quel type d'objet mathématique est stocké dans o ?

Les fonctions et leur représentation

Voici les définitions de fonction (ou application), et fonction partielle.

Fonction

Une *fonction* $f : A \to B$ est une relation $R \subseteq A \times B$ telle que :

$$\forall x \in A, \quad \exists y \in B \ / \ (x, y) \in R$$

et

$$\forall x \in A, \ \forall y_1, y_2 \in B, \ (x, y_1) \in R \text{ et } (x, y_2) \in R \Rightarrow y_1 = y_2.$$

Dit autrement, pour chaque x dans A, il existe un et un seul y dans B tel que (x, y) est dans la relation R.

Fonction partielle

Une *fonction partielle* $f : A \to B$ est une relation $R \subseteq A \times B$ telle que :

$$\forall x \in A, \ \forall y_1, y_2 \in B, \ (x, y_1) \in R \text{ et } (x, y_2) \in R \Rightarrow y_1 = y_2.$$

En comparant ces deux définitions, on voit que *toute fonction est une fonction partielle*.

Récupération des chemins

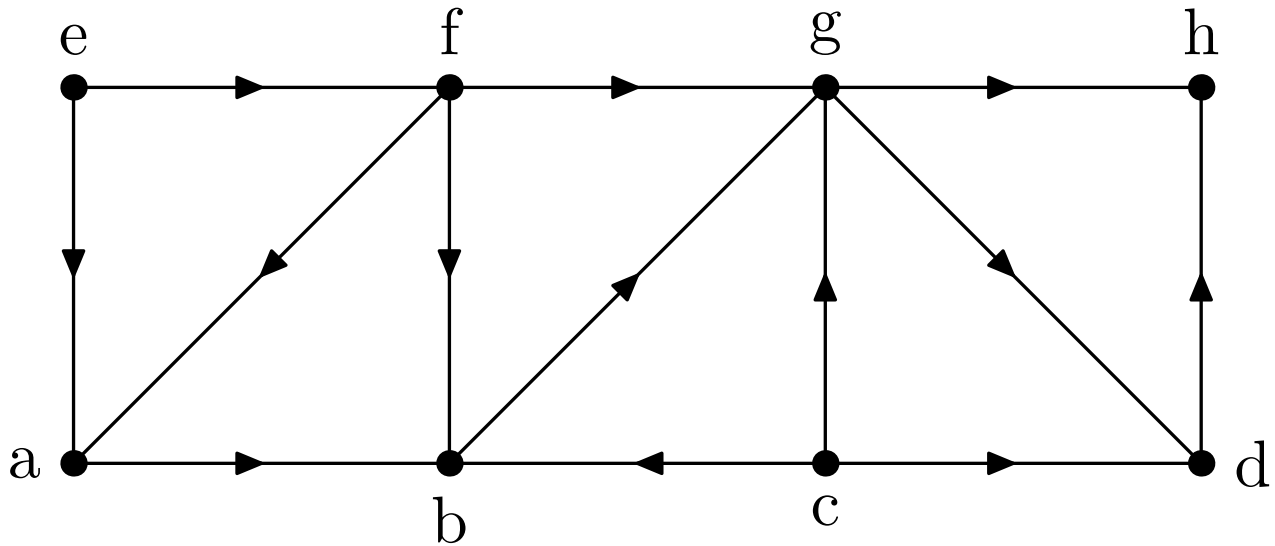
Une fois la fonction partielle o calculée, et pour un sommet t qui fait partie des sommets accessibles depuis s (ensemble de sommets retourné par Accessibles3() dans la variable M), il est possible de recalculer un chemin à l'aide de la routine suivante.

Note : cette routine n'a pas besoin de $\overrightarrow{G}$, ni de M.

```
Reconstruire_Chemin(s ∈ V, z ∈ V, o ⊆ V × V):                    1
– Si z ≠ s, faire :                                              2
   * Retourner Ajout_en_queue(z,                                 3
                        Reconstruire_Chemin(s, o[z], o))         4
– Sinon :                                                        5
   * Retourner (s)                                               6
```

Application pratique

Dérouler `Accessibles3()` sur le graphe ci-dessous à partir du sommet *e* ; et également à partir du sommet *a*.
Puis utilisez `Reconstruire_Chemin()` pour obtenir un chemin de *a* à *h*, un chemin de *a* à *d*, et un chemin de *e* à *b*.



Démarche de preuve ; nécessité des preuves

Vous êtes peut-être convaincus à présent que `Accessibles3()` calcule l'ensemble des sommets accessibles depuis un sommet donné ; et que `Reconstruire_Chemin()` reconstruit des chemins, lorsqu'on lui donne les bons paramètres.
Cette conviction, forgée sur des exemples, ne peut être à toute épreuve. Il est possible qu'un algorithme incorrect donne un résultat correct sur un ou plusieurs jeux de données.
Pour être pleinement convaincu, d'une conviction qui ne puisse être remise en cause, il nous faut prouver mathématiquement la correction de ces deux algorithmes.
Notre démarche de preuve va s'appuyer en très grande partie sur la notion de *certificat*.

Certificats pour un problème de décision

Problème de décision

Un *problème de décision* est un problème dont la réponse attendue, pour une donnée d'entrée D, est OUI ou NON.

Certificats

Un certificat du OUI pour un problème de décision P, avec D comme donnée d'entrée, est une information C telle que la connaissance de C prouve de manière certaine que la réponse OUI est la bonne réponse pour ce problème appliqué à la donnée d'entrée D.
De même, un certificat du NON pour un problème de décision P, avec D comme donnée d'entrée, est une information C telle que la connaissance de C prouve de manière certaine que la réponse NON est la bonne réponse pour ce problème appliqué à la donnée d'entrée D.

Certificats pour le problème de l'accessibilité

La première version du problème d'accessibilité, "Y a-t-il un chemin de s à t dans le graphe $\overrightarrow{G}$?", est un problème de décision. Il est donc possible de lui appliquer la notion de certificat.
Un certificat du OUI est facile à trouver pour ce problème. Comment pouvez-vous prouver à quelqu'un que le sommet t est accessible depuis le sommet s dans un certain graphe $\overrightarrow{G}$?
Il suffit de présenter un chemin de s à t de ce graphe.

Vers un certificat du NON pour accessibilité (1)

Théorème

Si un ensemble de sommets S de $\overrightarrow{G}$ contient s, ne contient pas t, et s'il n'existe pas d'arc $\overrightarrow{xy}$ avec $x \in S$ et $y \notin S$, alors il n'y pas de chemin de s à t dans $\overrightarrow{G}$.

Démonstration : nous allons montrer, par récurrence sur la longueur d'un chemin partant de s : $C = (x_0, \ldots, x_k)$, avec $x_0 = s$, que son extrémité x_k est dans S (propriété P_k). Ce qui est affirmé dans l'énoncé du théorème est une conséquence immédiate de ce fait.

Vers un certificat du NON pour accessibilité (2)

Initialisation : pour $k = 0$, $x_k = x_0 = s \in S$, par hypothèse.

Transmission : nous allons montrer que dès lors que P_k est vérifiée, P_{k+1} l'est également. Supposons donc que P_k est vérifiée, et prenons un chemin quelconque $C = (x_0, \ldots, x_k, x_{k+1})$. Formons le chemin C' en supprimant le dernier sommet de C : $C' = (x_0, \ldots, x_k)$. D'après l'hypothèse de récurrence P_k, on déduit que $x_k \in S$. Or l'hypothèse indique qu'il n'y a pas d'arc $\overrightarrow{xy}$ avec $x \in S$ et $y \notin S$. Donc l'arc $\overrightarrow{x_k x_{k+1}}$ ne peut pas être de ce type. Donc nécessairement, $x_{k+1} \in S$. Fin de la preuve de transmission ; fin de la récurrence ; fin de la preuve du théorème.

Un certificat du NON pour accessibilité

Théorème

L'ensemble S des sommets accessibles depuis s dans $\overrightarrow{G}$ est un certificat de non-accessibilité pour tout sommet t de $\overrightarrow{G}$ qui n'est pas dans S.

Démonstration : Soit $t \notin S$. Comme s est accessible depuis lui-même (avec le chemin $C = (s)$), on sait que $s \in S$. D'autre part, si $x \in S$, et $y \notin S$, l'existence d'un arc $\overrightarrow{xy}$ contredirait le fait que x est accessible depuis s, et que y ne l'est pas : car l'ajout de l'arc $\overrightarrow{xy}$ à un chemin C de s à x formerait un chemin de s à y.
Toutes les hypothèses du théorème précédent sont vérifiées ; donc par application du théorème, on sait qu'il n'y a pas de chemin de s à t.
(*fin de la preuve*)

Remarques sur le théorème précédent

Notez qu'il faut employer des moyens assez fins pour prouver qu'il n'y a pas de chemin de s à t ; simplement examiner des chemins dans le graphe ne nous permettra pas de conclure à la non-existence de chemins.

Ce théorème démontre qu'il existe toujours un moyen de prouver la non-existence de chemin entre deux sommets d'un graphe orienté.

Armés de tous ces concepts et théorèmes, nous pouvons à présent commencer les preuves de nos algorithmes.

Preuve d'algorithmes : la méthode

Pour prouver la correction d'un algorithme, il faut procéder en 3 étapes. Elles sont présentées dans un certain ordre, qui est souvent le plus commode ; mais l'ordre peut être changé si besoin.

Exécutabilité : Le premier point est de s'assurer que chaque instruction de l'algorithme s'exécutera correctement. Cette vérification, comme les autres, peut dépendre des données passées en entrée.

Arrêt : Pour qu'un algorithme soit correct, il faut en particulier qu'il se termine, quelles que soient ses entrées. Il faut donc vérifier ce point en lui-même.

Correction : Enfin, il faut vérifier que la ou les valeurs retournées par l'algorithme, s'il venait à s'arrêter, sont bien conformes à ce qui était attendu.

Il est important de ne pas confondre l'arrêt et la correction. Ce sont deux éléments complémentaires. Le plus souvent, la preuve de l'un ne constitue pas une preuve de l'autre.

Premières étapes de preuve pour Accessibles3()

Exécutabilité : On vérifie ligne à ligne que chaque instruction s'exécute correctement. Aucune ligne ne pose de véritable problème. Le Pour de la ligne 3 ne sera pas exécuté si l'ensemble $N^+(x)$ est vide. L'affectation ligne 6 pourra toujours être effectuée, car la valeur de retour d'un appel à `Accessibles3()` est au bon format (ligne 7).

Arrêt : Nous observons que l'ensemble M est passé à chaque appel récursif, et sa nouvelle valeur récupérée au retour de l'appel. On peut donc raisonner sur l'évolution de M au fil de l'exécution des appels emboîtés.

Par ailleurs, le traîtement effectué lors d'une exécution de `Accessibles3()` est limité en nombre d'instructions. Il suffit donc de montrer qu'il n'y a qu'un nombre fini d'appels récursifs.

Or un appel récursif est lancé sur y seulement si on vient de mettre y dans M. Donc on ne peut pas lancer deux appels récursifs sur le même sommet. Le nombre d'appels récursifs est donc limité par le nombre de sommets du graphe.

Donc toute exécution d'`Accessibles3()` finira bien par s'arrêter.

Exécutabilité de `Reconstruire_Chemin` (1)

La seule instruction qui pose problème est l'évaluation, à la ligne 4, de l'expression $o[z]$. Il faut voir o comme une fonction (fonction partielle, c'est-à-dire pas nécessairement définie partout) ; donc $o[z]$ signifie la valeur de la fonction o pour le paramètre z.
Il nous faut vérifier qu'à chaque fois que $o[z]$ est évaluée, il y a bien une valeur en z dans la fonction o. Pour cela, il faut se pencher sur le code de la routine `Accessibles3()`, puisque c'est là que o est définie.
Il faut noter deux choses :

- pour l'appel initial à `Reconstruire_Chemin()`, le sommet z sera toujours un élément de M ;
- l'expression $o[z]$ n'est évaluée que si $z \neq s$.

Exécutabilité de `Reconstruire_Chemin` (2)

Montrons le fait suivant : pour tout sommet z de $M \setminus \{s\}$, $o[z]$ est bien défini ; et sa valeur est un élément de M.
En effet, lorsqu'un élément est ajouté à M (hormis s, qui est intégré au M passé à l'appel initial), c'est un y (affecté à la ligne 2), et pour ce y, on lui associe dans o un x qui est déjà un élément de M (ligne 5).
En utilisant ce fait, on peut maintenant montrer par récurrence sur le numéro de l'appel récursif de `Reconstruire_Chemin()` que à chaque évaluation de l'expression $o[z]$, z est bien dans M, et $o[z]$ est bien dans M.
En effet, pour l'appel initial, nous avons déjà rappelé que $z \in M$. Ensuite, il suffit de montrer que si pour un appel donné, si $z \in M$, et $z \neq s$, alors $o[z] \in M$. C'est justement ce que dit le fait ci-dessus.
(fin de la preuve d'exécutabilité pour `Reconstruire_Chemins()`)

Arrêt de `Reconstruire_Chemin()`

Pour démontrer que `Reconstruire_Chemin()` s'arrête bien, il suffit de faire l'observation suivante. Si on associe à chaque sommet de $M \setminus \{s\}$ un numéro d'étape correspondant à quand il a été rajouté à M, alors il est facile de voir que les numéros successifs des sommets passés en paramètre des appels successifs de `Reconstruire_Chemin()` sont strictement décroissants. En effet, la valeur $o[z]$ a nécessairement un numéro d'étape inférieur à z, puisque $o[z]$ est un x qui était déjà dans M à l'étape où $z = y$ est mis dans M.
Une série décroissante d'entiers étant nécessairement finie, tout appel à `Reconstruire_Chemin()` (avec les paramètres adéquats) s'arrête.

Correction de `Reconstruire_Chemin()`

Pour démontrer que la valeur de retour de l'appel initial à `Reconstruie_Chemin()` est conforme à ce qui est attendu, il faut vérifier que la liste de sommets retournés forme bien un chemin de s à t dans $\overrightarrow{G}$.
Notez bien que les appels récursifs successifs procèdent en partant de t et en remontant jusqu'à s.
Le chemin commence bien en s : c'est le cas Sinon, ligne 5. Le chemin finit bien en t, puisque t est la valeur du paramètre z dans l'appel initial.
Enfin, deux sommets successifs insérés dans la liste $(o[z], z)$ forment bien un arc du graphe ; puisque justement, c'est à partir d'un arc $\overrightarrow{xy}$ que `Accessibles3()` renseigne o en prenant $o[z] = x$ et $z = y$.
La valeur de retour de `Reconstruire_Chemins()` est donc bien un chemin de s à t dans $\overrightarrow{G}$.

Correction d'Accessibles3()

Démontrer la correction d'Accessibles3(), c'est démontrer qu'à l'issue de l'exécution d'Accessibles3($\overrightarrow{G}, s, \{s\}, \emptyset$), l'ensemble M retourné est exactement l'ensemble des sommets accessibles depuis s dans $\overrightarrow{G}$.

Notons U l'ensemble des sommets accessibles depuis s dans $\overrightarrow{G}$; et M l'ensemble 'M' retourné à l'issue de l'appel d'Accessibles3(). Nous allons décomposer cette preuve en deux étapes :

$$(i) \quad M \subseteq U$$

$$(ii) \quad U \subseteq M$$

Tout sommet de M est accessible depuis s

Pour démontrer que $M \subseteq U$, il suffit de considérer un sommet t de M, et de démontrer qu'il est accessible depuis s dans $\overrightarrow{G}$. Il suffit pour cela de montrer un chemin allant de s à t dans $\overrightarrow{G}$.
Or nous avons fini la preuve de Reconstruire_Chemin(), pour tout sommet t de M. Appelons donc Reconstruire_Chemin(s, t, o). Cela nous fournit un chemin C de s à t. (fin de la preuve)

Tout sommet accessible depuis s est dans M (1)

Avant de montrer que $U \subseteq M$, nous allons démontrer le lemme suivant.

Lemme

Pour le M retourné par Accessibles3(), il n'y a pas de chemin de s vers un sommet hors de M.

Rappelons-nous du premier théorème indiquant un certificat du NON pour le problème de l'accessibilité (diapo 15).
Considérons un sommet t qui ne soit pas dans M. Notez que M vérifie les hypothèses du théorème :

- $s \in M$;
- $t \notin M$;
- il n'y pas d'arc $\overrightarrow{xy}$ avec $x \in M$, $y \notin M$: à chaque fois qu'un y est mis dans M, Accessibles3() sera appelé sur les successeurs de ce y, et seront donc eux-mêmes immanquablement mis dans M.

Donc il ne peut y avoir de chemin de s vers t, par application de ce théorème. (fin de la preuve)

Tout sommet accessible depuis s est dans M (2)

Nous sommes maintenant prêts à montrer que $U \subseteq M$.
Soit t un sommet accessible depuis s dans $\overrightarrow{G}$.
Si t n'était pas dans M, alors d'après le lemme précédent, il ne pourrait y avoir de chemin de s à t. Or t est accessible depuis s : ce cas est donc impossible.
Il ne reste que l'autre possibilité, à savoir que t est dans M.
(fin de la preuve)

La solution définitive au problème d'accessibilité

Voici maintenant un algorithme qui répond de manière totale au problème (de décision) de l'accessibilité d'un sommet depuis un autre sommet dans un graphe orienté :

Accessibilite$(\overrightarrow{G} = (V, A),\ s \in V, t \in V)$: 1

– $(M, o) =$ *Accessibles3*$(\overrightarrow{G}, s, \{s\}, \emptyset)$ 2

– Si $t \in M$, alors : 3

  * **Retourner** $(OUI, Reconstruire\_Chemin(s, t, o)$ 4

– Sinon : 5

  * **Retourner** (NON, M) 6

Il s'agit d'un algorithme répondant à la question par OUI ou par NON, et accompagnant sa réponse d'un certificat associé à cette réponse.

Preuve pour la routine Accessibilite()

L'exécutabilité et l'arrêt découlent de celles d'Accessibles3() et de Reconstruire_Chemin(), puisqu'il n'y a aucune boucle dans le code d'Accessibilite().
Il reste à montrer que le résultat retourné est toujours celui qui est attendu.
Nous faisons simplement observer que l'information retournée comme certficat du OUI - la valeur de retour de Reconstruire_Chemin() - est bien un certicat du OUI, puisque c'est un chemin de s à t dans $\overrightarrow{G}$.
De la même façon, l'information retournée comme certificat du NON - l'ensemble M - est bien un certificat du NON, comme conséquence de deux théorèmes : la preuve d'Accessibles3(), qui montre que l'ensemble M retourné est exactement l'ensemble des sommets accessibles depuis s dans $\overrightarrow{G}$; et le théorème qui affirme que l'ensemble des sommets accessibles depuis s forme un certificat du NON (diapo 17).
Puisque les certificats retournés sont corrects, on en déduit que les réponses retournées sont elles-mêmes correctes. (fin de la preuve)

Accessibilité : conclusion

Nous offrons en guise de conclusion quelques obsevations sur la démarche adoptée.

- Nous avons formulé deux problématiques d'accessibilité ;
- Afin de résoudre la "plus facile", nous avons dû résoudre au passage la "plus difficile" ;
- L'algorithme de résolution proposé pour la version "décision" du problème propose des certificats pour ses réponses ;
- Les algorithmes de résolution ont été prouvés ;
- Les preuves d'algorithmes s'appuient sur la preuve de la validité des certificats.