Chapitre 2

Les bases de la théorie des graphes

Ce chapitre montre comment utiliser les fonctions de Matlab pour résoudre les problèmes de la théorie des graphes. Matlab fourni une boite à outils destinée aux spécialistes en bioinformatique (Bioinformatics Toolbox™). Les fonctions de cette boite appliquent des algorithmes de théorie des graphes de base aux matrices creuses. Dans nos illustrations, nous utiliserons ces fonctions pour résoudre nos problèmes d'optimisation. Comme nous avons mentionné auparavant, vous pouvez utiliser des logiciels open source Octave / Python qui fournissent des outils didactiques tel que GRAPHTHEO (Graph Theory Didactic Toolkit). Nous citons parmi les notions de bases que nous aborderons dans ce chapitre, création de graphe, nœud, arc, orienté, arête, non-orienté, matrice d'adjacence, composants fortement et faiblement connectés, test des cycles, isomorphisme, tri topologique et comment traverser un graphe en suivant les nœuds adjacents.

2.1. Boîte à outils de bioinformatique

La boîte à outils de bioinformatique (Bioinformatics Toolbox™) fournit des algorithmes et des applications pour :

- Le séquencement de nouvelle génération (Next Generation Sequencing).

- L'analyse de microréseaux (microarray analysis).

- La spectrométrie de masse (mass spectrometry)

- L'ontologie des gènes (gene ontology).

À l'aide des fonctions de la boîte à outils, vous pouvez :

- Lire des données génomiques et protéomiques.

- Explorer et visualiser ces données avec des navigateurs de séquences, des cartes thermiques spatiales et des diagrammes de cluster.

La boîte à outils fournit également des techniques statistiques pour :

- Détecter les pics.

- Imputer des valeurs pour les données manquantes.

- Sélectionner des entités.

Vous pouvez combiner les fonctions de la boîte à outils pour prendre en charge les workflows bioinformatiques courants.

Vous pouvez utiliser les données ChIP-Seq pour identifier les facteurs de transcription; analyser les données RNA-Seq pour identifier les gènes différentiellement exprimés; identifier les variantes du nombre de copies et les SNP dans les données de microréseau; et classer les profils protéiques à l'aide de données de spectrométrie de masse.

2.2. Fonctions de théorie des graphes

Les fonctions de la théorie des graphes de Bioinformatics Toolbox ™ appliquent des algorithmes de la théorie des graphes de base aux matrices creuses. Une matrice creuse représente un graphe, toutes les entrées non nulles de la matrice représentent les arcs du graphe et les valeurs de ces entrées représentent le poids associé (coût, distance, longueur ou capacité) de l'arc. Les algorithmes du graphe qui utilisent les informations de poids annulent l'arc si un NaN (non défini) ou un Inf (inféni) est trouvé. Les algorithmes de graphe qui n'utilisent pas les informations de poids considéreront l'arc si un NaN ou un Inf est trouvé, car ces algorithmes ne regardent que la connectivité décrite par la matrice creuse et non les valeurs stockées dans la matrice creuse. Les matrices creuses peuvent représenter quatre types de graphes:

Graphe orienté - Matrice creuse, de type double réel ou logique. L'indice de ligne (colonne) indique la source (destination) de l'arc. Les auto-boucles (valeurs dans la diagonale) sont autorisées, bien que la plupart des algorithmes ignorent ces valeurs.

Graphique non orienté - Une matrice creuse triangulaire inférieur, de type double réel ou logique. Un algorithme qui génère un graphe non orienté ignore les valeurs stockées dans le triangle supérieur de la matrice creuse et les valeurs dans la diagonale.

Graphique acyclique direct (GAD) - Matrice creuse, de type double réel ou logique, avec des valeurs nulles dans la diagonale. Bien qu'une diagonale de valeur zéro soit une exigence dans un GAD,. Un algorithme qui génère un GAD ne testera pas les cycles car cela ajoutera une complexité indésirable.

Arbre d'enjambement - Graphe non orienté sans cycles et avec un composant connecté.

Aucun attribut n'est attaché aux graphes; des matrices creuses représentant les quatre types de graphes peuvent être transmises à n'importe quel algorithme de graphe. Toutes les fonctions retournent une erreur sur les matrices creuses non carrées.

Les algorithmes de graphes ne prétextent pas les propriétés des graphes car de tels tests peuvent introduire une pénalité de temps. Par exemple, il existe un algorithme efficace pour trouver le plus court chemin dans un GAD, mais tester si un graphe est acyclique est coûteux par rapport à l'algorithme. Par conséquent, il est important de sélectionner une fonction et des propriétés de la théorie des graphes appropriées au type de graphe représenté par votre matrice d'entrée. Si l'algorithme reçoit un type de graphe différent de ce qu'il attend, il:

- Retourne une erreur lorsqu'il atteint une incohérence. Par exemple, si vous passez un graphe cyclique à la fonction `graphshortestpath` et spécifiez `Acyclic` comme propriété de la méthode.

- Produire un résultat invalide. Par exemple, si vous passez un graphe orienté à une fonction avec un algorithme qui génère un graphe non orienté, il ignorera les valeurs dans le triangle supérieur de la matrice creuse.

Parmi les fonctions de la théorie des graphes nous étudions les suivants :

`Biograph`,`graphallshortestpaths`, `graphconncomp`, `graphisdag`, `graphisomorphism`, `graphmaxflow`, `graphminspantree`, `graphshortestpath`, `graphtopoorder`, et `graphtraverse`.

2.3. Création de graphe, nœud, arc, orienté et matrice d'adjacence

La fonction Biograph

Syntaxe

```
Gobj =biograph(G,NodeIDs,'ID', IDValue,'Label',LabelValue, 'Description', ...
DescriptionValue,'LayoutType',LayoutTypeValue,'EdgeType',EdgeTypeValue,'Scale',
... ScaleValue,'LayoutScale',LayoutScaleValue,'EdgeTextColor',EdgeTextColorValue,
... 'EdgeFontSize',EdgeFontSizeValue,'ShowArrows',ShowArrowsValue,'ArrowSize', ...
ArrowSizeValue,'ShowWeights',ShowWeightsValue,'ShowTextInNodes',ShowTextInNodesVal
ue, ...
'NodeAutoSize',NodeAutoSizeValue,'NodeCallback',NodeCallbackValue,'EdgeCallback',
... EdgeCallbackValue,'CustomNodeDrawFcn',CustomNodeDrawFcnValue)
```

`Gobj = biograph(CMatrix)` crée un objet graphe, Gobj, à l'aide d'une matrice de connexion, G. Toutes les entrées non diagonales et positives dans la matrice de connexion, G, indiquent les nœuds connectés, les lignes représentent les nœuds sources et les colonnes représentent les nœuds distinations.

Arguments

Arguments d'entrée	
G	Matrice carrée creuse qui sert de matrice de connexion. Autrement dit, une valeur de 1 indique une connexion entre les nœuds tandis qu'un 0 indique aucune connexion. Le nombre de lignes ou de colonnes est égal au nombre de nœuds.Types de données: double
NodeIDs	Étiquettes de noeud. Entrez l'un des éléments suivants: o Tableau de cellules (Cell array) de vecteurs de caractères ou vecteur de chaînes de caractères (string) avec le nombre de vecteurs de caractères (ou chaînes) égal au nombre de lignes ou de colonnes dans la matrice de connexion G. Chaque vecteur de caractères (ou chaîne) doit être unique. o Tableau de caractères avec un nombre de lignes égal au nombre de nœuds. Chaque ligne du tableau doit être unique. o Vecteur de caractères ou chaîne de caractères avec un nombre de caractères égal au nombre de nœuds. Chaque caractère doit être unique. Les valeurs par défaut sont les numéros de ligne ou de colonne. o Vous devez spécifier *NodeIDs* si vous souhaitez spécifier des paires nom / valeur de propriété. o Définissez *NodeIDs* sur [] pour utiliser les valeurs par défaut des numéros de ligne / colonne.
IDValue	Vecteur de caractères ou chaîne de caractères pour identifier l'objet graphe. Par défaut est ''.
LabelValue	Vecteur de caractères ou chaîne de caractères pour identifier l'objet graphe. La valeur par défaut est ''.
DescriptionValue	Vecteur de caractères ou chaîne de caractères pour étiqueter l'objet graphe. La valeur par défaut est ''.
LayoutTypeValue	Vecteur de caractères ou chaîne de caractères qui spécifie l'algorithme du moteur de présentation (layout engine). Les choix sont: o `'hierarchical'` (par défaut) - Utilise un ordre topologique du graphe pour attribuer des niveaux, puis organise les nœuds de haut en bas, tout en minimisant les arcs croisées. o `'radial'` - Utilise un ordre topologique du graphe pour attribuer des niveaux, puis organise les nœuds de l'intérieur vers l'extérieur du cercle, tout en minimisant les arcs croisées. o `'equilibrium'` - Calcule la disposition (layout) en minimisant l'énergie dans un système de ressort dynamique (spring system).
EdgeTypeValue	Vecteur de caractères ou chaîne de caractères qui spécifie comment les arcs s'affichent. Les choix sont : `'straight'` ('tout droit'), `'curved'` ('courbe') par défaut et `'segmented'` ('segmenté'). N.B : Les arcs incurvés ou segmentés ne se produisent que lorsque cela est nécessaire pour éviter toute obstruction par les

	nœuds. Les objets du graphe avec `LayoutType` égal à `'equilibrium'` ou `'radial'` ne peuvent pas produire d'arcs incurvées ou segmentées.
ScaleValue	Nombre positif qui remet à l'échelle les coordonnées du nœud. La valeur par défaut est 1.
LayoutScaleValue	Nombre positif qui met à l'échelle la taille des nœuds avant d'appeler le moteur de présentation. La valeur par défaut est 1.
EdgeTextColorValue	Vecteur numérique à trois éléments des valeurs RGB. La valeur par défaut est [0, 0, 0], qui définit le noir.
EdgeFontSizeValue	Nombre positif qui définit la taille de la police de l'arc en points. La valeur par défaut est 8.
ShowArrowsValue	Contrôle d'affichage des flèches pour les arcs. Les choix sont «on» (par défaut) ou «`off`».
ArrowSizeValue	Nombre positif qui définit la taille des flèches en points. La valeur par défaut est 8.
ShowWeightsValue	Contrôle d'affichage du texte indiquant le poids des arcs. Les choix sont «on» ou «`off`» (par défaut).
ShowTextInNodesValue	Vecteur de caractères ou chaîne de caractères qui spécifie la propriété du nœud utilisée pour étiqueter les nœuds lorsque vous affichez un objet graphe à l'aide de la méthode d'affichage. Les choix sont: o `'Label'` - Utilise la propriété `Label` de l'objet nœud (par défaut). o 'ID' - Utilise la propriété `ID` de l'objet nœud. o 'None'.
NodeAutoSizeValue	Contrôle le précalcul de la taille du nœud avant d'appeler le moteur de présentation. Les choix sont «on» (par défaut) ou «`off`». N.B : Désactivez-le si vous souhaitez appliquer différentes tailles de nœuds en modifiant la propriété Size.
NodeCallbackValue	Rappel l'utilisateur pour tous les nœuds. Entrez le nom d'une fonction, d'un descripteur de fonction ou d'un tableau de cellules avec plusieurs descripteurs de fonction. Après avoir utilisé la fonction `view` pour afficher le graphe dans la visionneuse de graphes, vous pouvez double-cliquer sur un nœud pour activer le premier rappel, ou cliquer avec le bouton droit et sélectionner callback to activate. La valeur par défaut est `@(node) inspect(node)`, qui affiche la boîte de dialogue `Property Inspector`.
EdgeCallbackValue	Rappel l'utilisateur pour tous les arcs. Entrez le nom d'une fonction, d'un descripteur de fonction ou d'un tableau de cellules avec plusieurs descripteurs de fonction. Après avoir utilisé la fonction `view` pour afficher l'objet graphe dans la visionneuse de graphe, vous pouvez cliquer avec le bouton droit et sélectionner un rappel à activer. Par défaut est la fonction anonyme, `@(edge) inspect(edge)`, qui affiche la boîte de dialogue Property Inspector.
CustomNodeDrawFcnValue	Descripteur de fonction vers une fonction personnalisée pour dessiner des nœuds. La valeur par défaut est [].

Tableau 2.1. Arguments d'entrée – biograph –

Illustration 2.1

Cette illustration montre comment créer un objet graphe (objet biographe).

Créez un objet graph avec des identifiants (ID) de nœud par défaut, puis utilisez la fonction `get` pour afficher les ID de nœud.

```
m = [0 1 1 0 0;1 0 0 1 1;1 0 0 1 0;0 0 0 0 1;1 0 1 0 0];
G1 = biograph(m)
```

Un objet graphe à 5 nœuds et 11 arcs est cré.

```
get(G1.nodes,'ID')
ans = 5x1 cell
    {'Node 1'}
    {'Node 2'}
    {'Node 3'}
    {'Node 4'}
    {'Node 5'}
```

Créez un objet graphe, attribuez les ID de nœud, puis utilisez la fonction `get` pour afficher les ID de nœud.

```
m = [0 1 1 0 0;1 0 0 1 1;1 0 0 1 0;0 0 0 0 1;1 0 1 0 0];
ids = {'S831','P025','K454','T233','M190'};
G2 = biograph(m,ids);
get(G2.nodes,'ID')
ans = 5x1 cell
    {'S831'}
    {'P025'}
    {'K454'}
    {'T233'}
    {'M190'}
```

Affichez l'objet graphe.

```
view(G2)
```

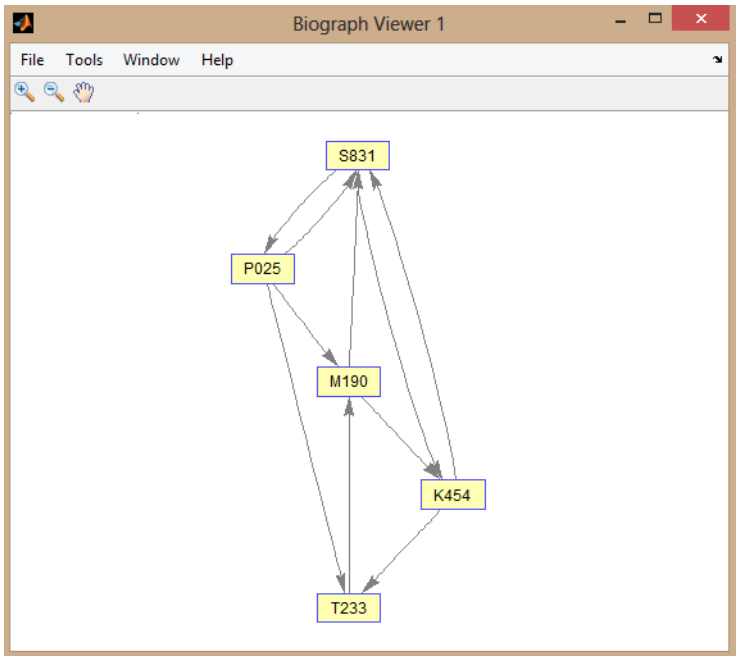

Figure 2.1. Objets graphe à 5 nœuds et 11 arcs.

2.4. Composants fortement et faiblement connectés

La fonction graphconncomp

Trouver les composants fortement ou faiblement connectés dans un graphe.

Syntaxe

```
[S, C] = graphconncomp(G,'Directed', DirectedValue,'Weak', WeakValue)
```

`graphconncomp` trouve les composants fortement connectés du graphe représenté par la matrice G en utilisant l'algorithme de Tarjan. Un composant fortement connecté est un groupe maximal de nœuds qui sont accessibles mutuellement sans violer les directions des arcs. L'entrée `G` est une matrice creuse$\boldsymbol{N \times N}$. Les entrées non nulles dans la matrice représentent les arcs.
L'algorithme de Tarjan a une complexité temporelle de $O(N+E)$, où N et E sont le nombre de nœuds et d'arêtes respectivement.

Arguments

Argument d'entrée	
G	Graphe représenté par son formalisme mathématique sous-djacent qui est une matrice d'adjacence, spécifiée comme une matrice creuse$\boldsymbol{N \times N}$. Les entrées non nulles dans la matrice représentent les poids des arrêtes. Types de données: double
SNode	Nœud dans G.
DNode	Nœud dans G.
DirectedValue	Propriété indiquant si le graphe est orienté ou non. Entrez `false` pour un graphe non orienté. Il en résulte que le triangle supérieur de la matrice creuse est ignoré. La valeur par défaut est `true`. Un algorithme basé sur DFS calcule les composants connectés. La complexité temporelle est $O(N+E)$, où N et E sont respectivement le nombre de nœuds et d'arcs.
WeakValue	Propriété indiquant s'il faut rechercher des composants faiblement connectés ou des composants fortement connectés. Un composant faiblement connecté est un groupe maximal de nœuds qui sont mutuellement accessibles en violant les directions des arêtes. Définissez `WeakValue` à `true` pour rechercher les composants faiblement connectés. La valeur par défaut est `false`, qui trouve des composants fortement connectés. L'état de ce paramètre n'a aucun effet sur les graphes non orientés car les composants faiblement et fortement connectés sont les mêmes dans les graphes non orientés. La complexité temporelle est $O(N+E)$, où N et E sont respectivement le nombre de nœuds et d'arêtes.

Tableau 2.2. Arguments d'entrée – graphconncomp –

Argument de sortie	
S	Retourne le nombre de composants trouvés.
C	Un vecteur indiquant à quel composant appartient chaque nœud.

Tableau 2.3. Arguments de sortie – graphconncomp –

N.B :

- Par définition, un seul nœud peut être un composant fortement connecté.
- Un graphe acyclique orienté ne peut avoir aucun composant fortement connecté supérieur à un.

Illustration 2.2

Créez et affichez un graphe orienté composé de 10 nœuds et 16 arcs.

```
DirG = sparse([1 1 1 2 2 3 3 4 5 6 7 7 8 9 9  9 9], ...
        [2 6 8 3 1 4 2 5 4 7 6 4 9 8 10 5 3],true,10,10)

DirG =


   (2,1)        1
   (1,2)        1
   (3,2)        1
   (9,3)        1
   (3,4)        1
   (5,4)        1
   (7,4)        1
   (4,5)        1
   (9,5)        1
   (1,6)        1
   (7,6)        1
   (6,7)        1
   (1,8)        1
   (9,8)        1
   (8,9)        1
   (9,10)       1

h = view(biograph(DirG));
```

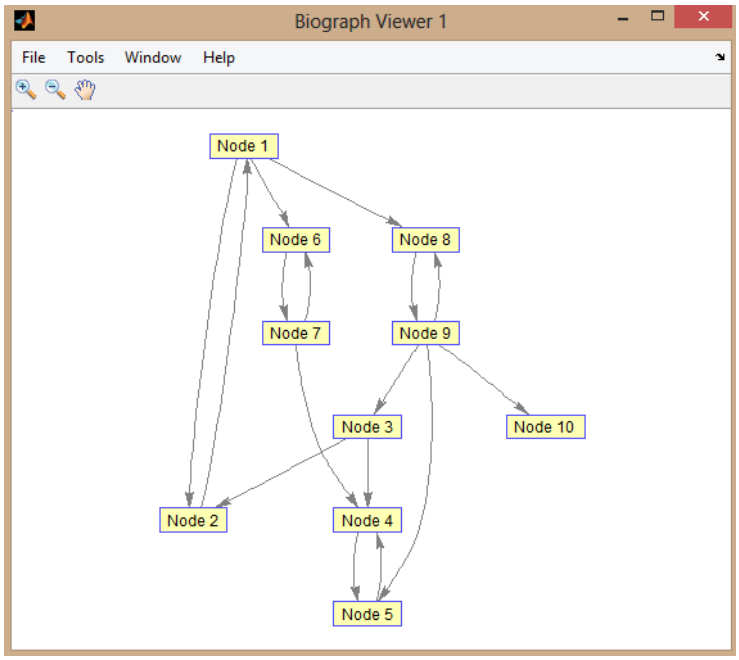

Figure 2.2. Graphe – graphconncomp –

Trouvez le nombre de composants fortement connectés dans un graphe orienté et déterminez à quel composant chacun des 10 nœuds appartient.

```
[nfc,comp] = graphconncomp(DG)

nfc =

     4

comp =

     4     4     4     1     1     2     2     4     4
```

Colorez les nœuds de chaque composant avec une couleur différente.

```
colors = jet(nfc);
for i = 1:numel(h.nodes)
    h.Nodes(i).Color = colors(comp(i),:);
end
```

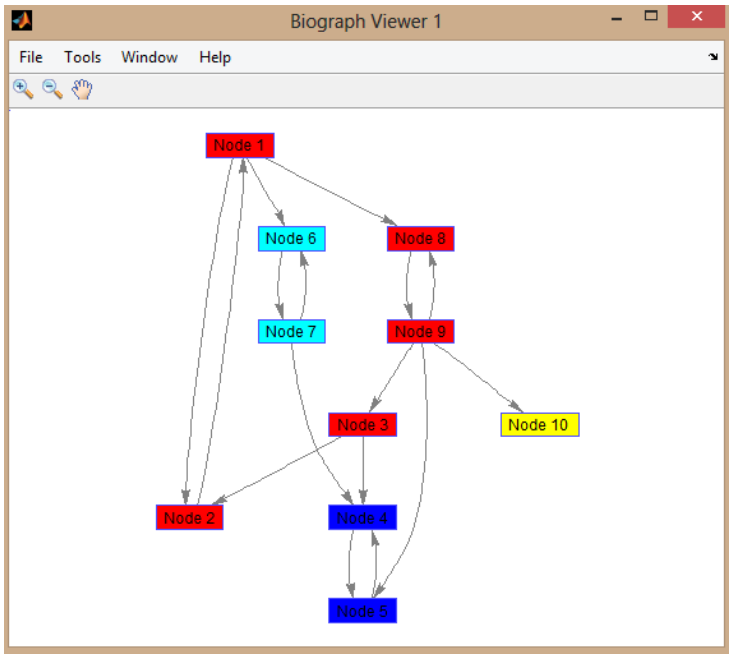

Figure 2.3. Coloration du graphe – graphconncomp –

2.5. Test des cycles

La fonction graphisdag

Test des cycles dans un graphe orienté.

Syntaxe

`graphisdag(G)`

`graphisdag(G)` retourne 1 logique (`true`) si le graphe orienté représenté par la matrice G est un graphe acyclique orienté (GAO) et 0 logique (`false`) sinon. G est une matrice creuse $N \times N$ qui représente un graphe orienté. Les entrées non nulles dans la matrice G indiquent la présence d'un arc.

Arguments

Argument d'entrée	
G	Graphe orienté représenté par son formalisme mathématique sous-djacent qui est une matrice d'adjacence, spécifiée comme une matrice creuse$\boldsymbol{N \times N}$. Les entrées non nulles dans la matrice représentent les poids des arrêtes. Types de données: double

Tableau 2.4. Arguments d'entrée – graphisdag –

Illustration 2.3

Test des cycles dans les graphes orientés.

Créez et affichez un graphe acyclique orienté (GAO) composé de six nœuds et huit arcs.

```
GAO = sparse([1 1 1 2 2 3 4 6],[2 4 6 3 5 4 6 5],true,6,6)
```

```
GAO =
```

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

```
view(biograph(GAO))
```

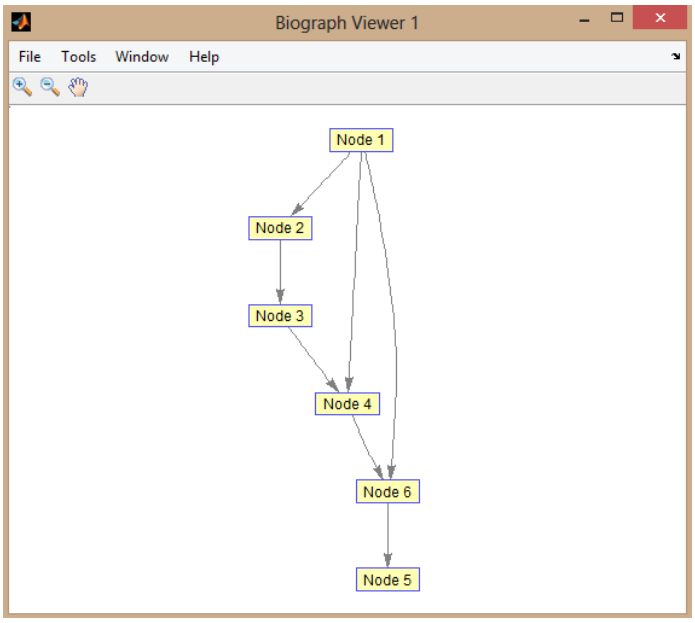

Figure 2.4. Graphe GAO1 – graphisdag –

Testez les cycles dans le GAO.

```
graphisdag(GAO)
ans =
       1
```

Ajoutez un arc au GAO pour le rendre cyclique, puis affichez le graphe orienté.

```
>> GAO(5,1) = true
GAO =
   (5,1)        1
   (1,2)        1
   (2,3)        1
   (1,4)        1
   (3,4)        1
```

```
    (6,5)        1
    (1,6)        1
    (4,6)        1
```

```
view(biograph(GAO))
```

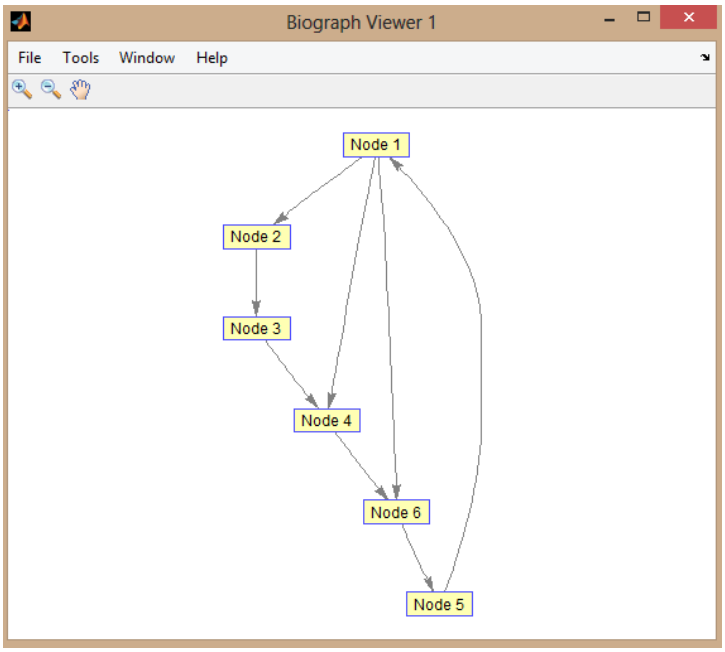

Figure 2.5. Graphe GAO2 – graphisdag –

Testez les cycles dans le nouveau graphe.

```
graphisdag(GAO)
```

```
ans =

     0
```

2.6. Isomorphisme des graphes

La fonction graphisomorphism

Trouver l'isomorphisme entre deux graphes

Syntaxe

```
[Isomorphic,Map] = graphisomorphism(G1,G2)
```

`graphisomorphism(G1, G2)` retourne 1 logique (`true`) en isomorphe si G1 et G2 sont des graphes isomorphes, et 0 logique (`false`) sinon.

```
[Isomorphic,Map] = graphisomorphism(G1,G2,'Directed',DirectedValue)
```

`[Isomorphic, Map] = graphisomorphism(G1, G2,'Directed', DirectedValue)` indique si les graphiques sont orientés ou non. Entrez **false** lorsque G1 et G2 sont des graphes non orientés. Dans

ce cas, les triangles supérieurs des matrices creuses G1 et G2 sont ignorés. La valeur par défaut est **true**, ce qui signifie que les deux graphiques sont orientés.

La complexité temporelle la plus défavorable est $O\ (N!)$, Où N est le nombre de nœuds.

Arguments

Argument d'entrée	
G1	Matrice creuse $\boldsymbol{N \times N}$ qui représente un graphe orienté ou non orienté. Les entrées non nulles dans la matrice G1 indiquent la présence d'une liaison.
G2	Matrice creuse $\boldsymbol{N \times N}$ qui représente un graphe orienté ou non orienté. Les entrées non nulles dans la matrice G2 indiquent la présence d'une liaison. G2 doit être le même (dirigé ou non) que G1.
DirectedValue	Propriété qui indique si les graphiques sont orientés ou non. Entrez **false** lorsque G1 et G2 sont des graphes non orientés. Dans ce cas, les triangles supérieurs des matrices creuses G1 et G2 sont ignorés. La valeur par défaut est **true**, ce qui signifie que les deux graphiques sont orientés.

Tableau 2.5. Arguments d'entrée – graphisomorphism –

<table>
<tr><th colspan="4">Argument de sortie</th></tr>
<tr><td></td><td colspan="3"></td></tr>
<tr><td rowspan="3">Isomorphic</td><td colspan="3">Un isomorphisme de graphe est un mappage 1 à 1 des nœuds du graphe G1 et des nœuds du graphe G2 de telle sorte que les contiguïtés (l'adjacence) sont préservées. G1 et G2 sont toutes deux des matrices creuses $\boldsymbol{N \times N}$ qui représentent des graphes orientés ou non orientés. L'argument de sortie Isomorphic est booléenne.</td></tr>
<tr><td>Lorsque Isomorphic est <code>true</code>.</td><td rowspan="2">Map</td><td>Map est un vecteur ligne contenant les indices de nœuds qui mappent de G2 à G1 pour un isomorphisme possible.</td></tr>
<tr><td>Lorsque Isomorphic est <code>false</code>.</td><td>Map est vide.</td></tr>
</table>

Tableau 2.6. Arguments de sortie – graphisomorphism –

Illustration 2.4

Créez et affichez un graphe orienté composé de 8 nœuds et 11 arcs.

```
M('ABCDEF') = [1 2 3 4 5 6];
G1 = sparse(M('ABDCCE'),M('BCBDEF'),true,8,8)
        G1 =
           (1,2)        1
           (4,2)        1
           (2,3)        1
           (3,4)        1
           (3,5)        1
           (5,6)        1

view(biograph(G1,'ABCDEF'))
```

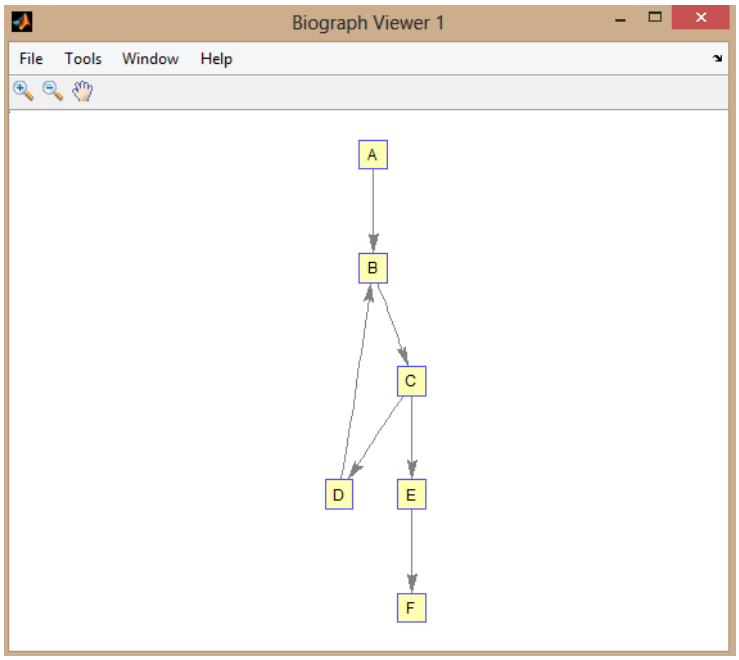

Figure 2.6. Graphe G1 – graphisdag –

Définissez un vecteur de permutation aléatoire, puis créez et affichez un nouveau graphe permuté.

```
per = randperm(6)
```

```
per =

     6     3     5     1     2     4
```

```
G2 = G1(per,per);
view(biograph(G2,'123456'))
```

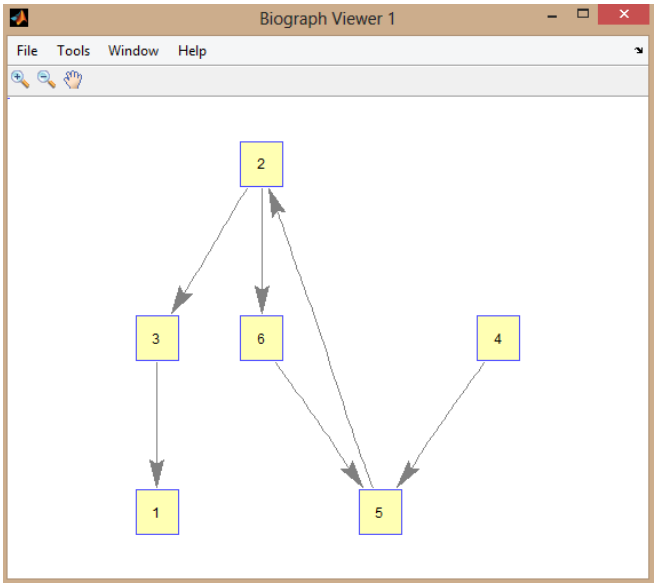

Figure 2.7. Graphe G2 – graphisdag –

Vérifiez si les deux graphes sont isomorphes.

```
[isom,Map] = graphisomorphism(G2,G1)
```

```
isom =

       1


Map =

     6     3     5     1     2     4
```

Notez que le vecteur ligne `Map` contenant les indices de noeud qui mappent de G2 à G1 est le même que le vecteur de permutation que vous avez créé à l'étape 2.

Inversez la direction des arcs B-D dans le premier graphe, puis vérifiez à nouveau l'isomorphisme.

```
G1(M('BD'),M('DB')) = G1(M('DB'),M('BD'));
```

```
view(biograph(G1,'ABCDEF'))
```

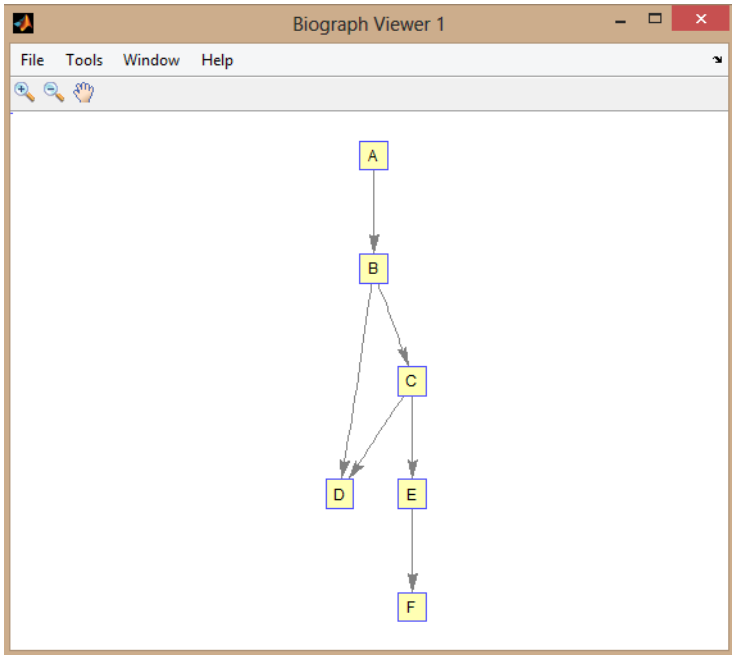

Figure 2.8. Graphe G1 – modifié –

```
[isom,Map] = graphisomorphism(G2,G1)
```

```
isom =

     0


Map =

     []
```

Convertir les graphes en graphes non orientés, puis vérifiez l'isomorphisme

```
[isom,Map] = graphisomorphism(G2+G2',G1+G1','directed',false)
```

```
isom =
```

```
     1

Map =

     6     3     5     1     2     4
```

2.7. Le tri topologique d'un graphe acyclique orienté

La fonction graphtopoorder

Effectuer un tri topologique d'un graphe acyclique orienté.

Syntaxe

`order = graphtopoorder(G)`

`order = graphtopoorder(G)` retourne un vecteur d'indices des nœuds triés avec l'ordre topologique. Dans l'ordre topologique, un arc peut exister entre un nœud source u et un nœud de destination v, si et seulement si u apparaît avant v dans l'ordre vectoriel.

Arguments

Argument d'entrée	
G	Matrice creuse $\boldsymbol{N} \times \boldsymbol{N}$ qui représente un graphe acyclique orienté. Les entrées non nulles dans la matrice G indiquent la présence d'une liaison.

Tableau 2.7. Arguments d'entrée – graphtopoorder –

Argument de sortie	
order	retourne un vecteur d'indices des nœuds triés avec l'ordre topologique. Dans l'ordre topologique, un arc peut exister entre un nœud source u et un nœud de destination v, si et seulement si u apparaît avant v dans l'ordre vectoriel.

Tableau 2.8. Arguments de sortie – graphtopoorder –

Illustration 2.5

Créez et affichez un graphe acyclique orienté (GAO) composé de six nœuds et huit arcs.

```
GAO = sparse([6 6 6 2 2 3 5 1],[2 5 1 3 4 5 1 4],true,6,6)

GAO =

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

```
view(biograph(GAO))
```

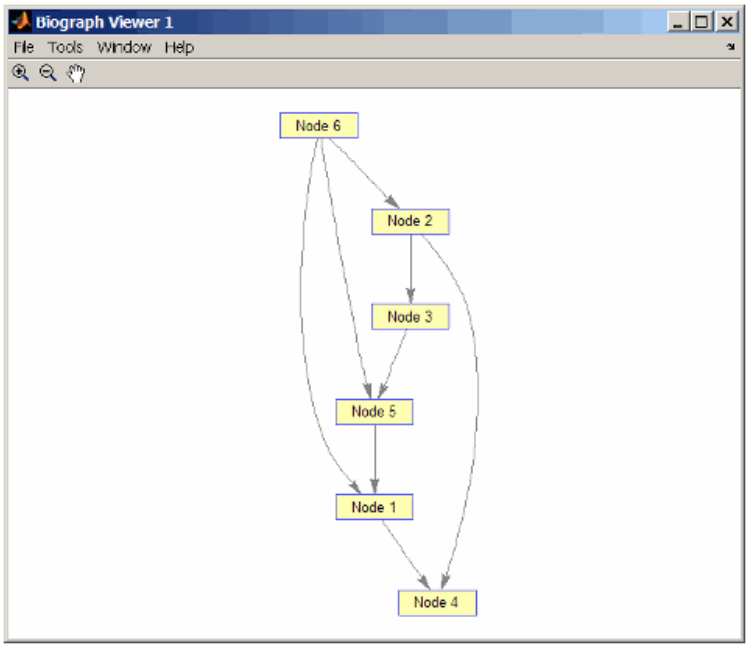

Figure 2.9. Graphe – graphtopoorder –

Trouvez l'ordre topologique du GAO.

```
order = graphtopoorder(GAO)

order =

     6     2     3     5     1     4
```

Permutez les nœuds afin qu'ils apparaissent ordonnés dans l'affichage du graphe.

```
GAO = GAO(order,order)

GAO =

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

view(biograph(GAO))
```

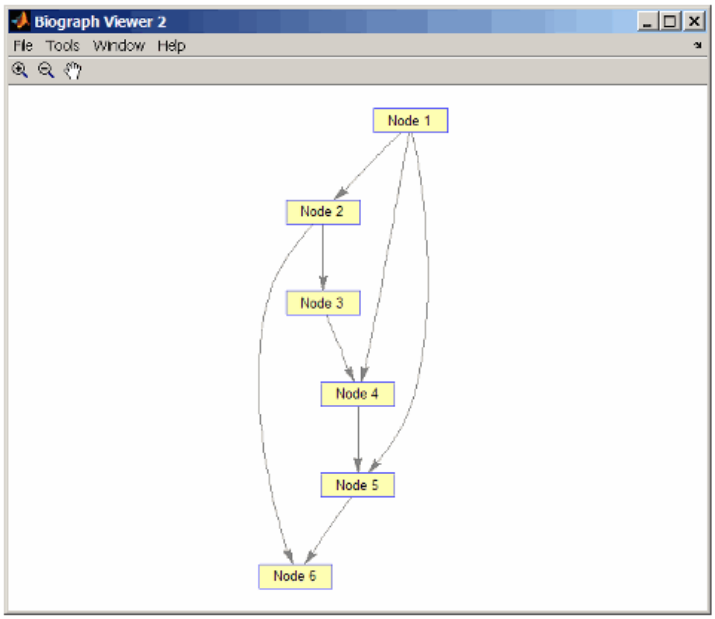

Figure 2.10. Graphe après tri – graphtopoorder –

2.8. Traverser un graphe en suivant les nœuds adjacents

La fonction graphtraverse

Traverser le graphe en suivant les nœuds adjacents.

Syntax

```
[disc, pred, closed] = graphtraverse(G,S,'Depth', DepthValue,'Directed', ...
DirectedValue,'Method', MethodValue)
```

`[disc, pred, closed] = graphtraverse(G, S)` parcourt le graphe G à partir du nœud indiqué par l'entier S. G est une matrice creuse $N \times N$ qui représente un graphe orienté. Les entrées non nulles dans la matrice G indiquent la présence d'une liaison.

Arguments

Argument d'entrée	
G	Matrice creuse $N \times N$ qui représente un graphe orienté. Les entrées non nulles dans la matrice G indiquent la présence d'une liaison.
S	Entier qui indique l'indice du nœud source dans le graphe G.
DepthValue	Spécifie la profondeur de la recherche. DepthValue est un entier indiquant un nœud dans le graphe G. La valeur par défaut est Inf (infini).
DirectedValue	Propriété qui indique si le graphe G est orienté ou non. Entrez `false` pour un graphique non orienté. Il en résulte que le triangle supérieur de la matrice creuse est ignoré. La valeur par défaut est `true`.
MethodValue	Vecteur de chaîne de caractères (char ou string) qui spécifie l'algorithme utilisé pour parcourir le graphique. Les choix sont: o 'BFS' – (Breadth-first search) Recherche en premier. La complexité temporelle est $O\,(N+E)$, où N et E sont respectivement le nombre de nœuds et d'arcs. o 'DFS' (Depth-first search) Default algorithm - Algorithme par défaut. Recherche en profondeur d'abord. La complexité temporelle est $O\,(N+E)$, où N et E sont respectivement le nombre de nœuds et d'arcs.

Tableau 8.10. Arguments d'entrée – graphtraverse –

Argument de sortie	
disc	Vecteur d'indices de nœuds dans l'ordre dans lequel ils sont découverts.
pred	Vecteur d'indices de nœuds prédécesseurs (répertoriés dans l'ordre des indices de nœuds) de l'arbre couvrant résultant.
closed	Vecteur d'indices de nœuds dans l'ordre dans lequel ils sont fermés.

Tableau 2.11. Arguments de sortie – graphtraverse –

Illustration 2.6

Créez un graphe orienté composé de 8 nœuds et 9 arcs.

```
GO = sparse([1 2 3 4 5 5 5 6 7 8 8 9],[2 4 1 5 3 6 7 9 8 1 10 2],true,10,10)

GO =

   (3,1)        1
   (8,1)        1
   (1,2)        1
   (5,3)        1
   (2,4)        1
   (4,5)        1
   (5,6)        1
   (5,7)        1
   (7,8)        1


h = view(biograph(DG))
```

Objet graphe composé de 8 nœuds et 9 arcs.

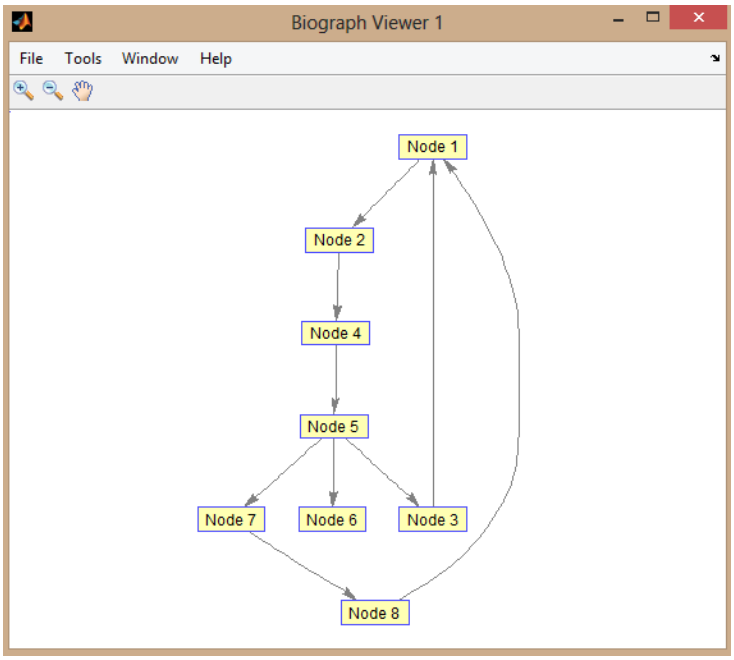

Figure 2.11. Graphe – graphtraverse –

Parcourez le graphe pour trouver l'ordre de découverte de la recherche en profondeur d'abord (DFS) à partir du nœud 4.

```
order = graphtraverse(GO,4)
order =
        4     5     3     1     2     6     7     8
```

Étiquetez les nœuds avec l'ordre de découverte DFS.

```
for i = 1:8
        h.Nodes(order(i)).Label =...
        sprintf('%s:%d',h.Nodes(order(i)).ID,i);
end
h.ShowTextInNodes = 'label'
dolayout(h)
```

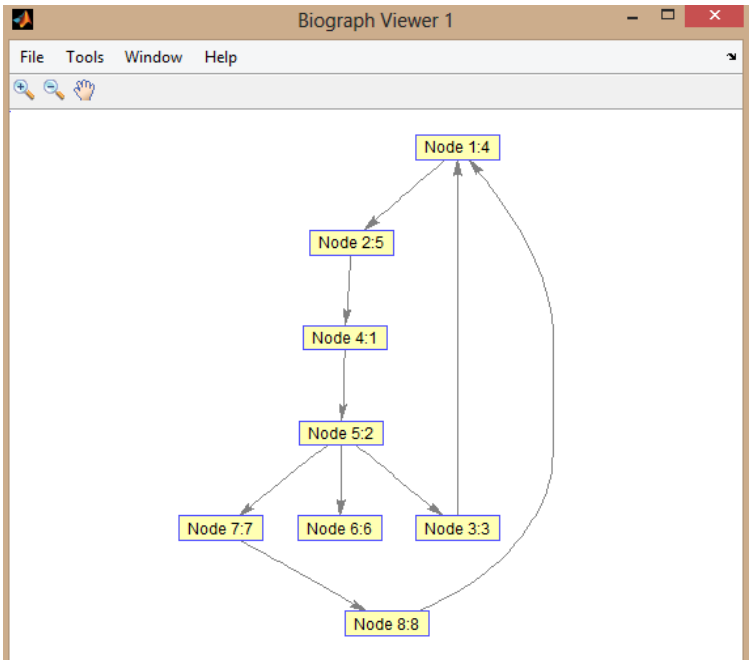

Figure 2.12. Graphe étiquité DFS – graphtraverse –

Parcourez le graphe pour trouver l'ordre de découverte de la recherche en premier (BFS) à partir du nœud 4.

```
order = graphtraverse(GO,4,'Method','BFS')
```

```
order =

     4     5     3     6     7     1     8     2
```

Étiquetez les nœuds avec l'ordre de découverte BFS.

```
for i = 1:8
        h.Nodes(order(i)).Label =...
        sprintf('%s:%d',h.Nodes(order(i)).ID,i);
end
h.ShowTextInNodes = 'label'
dolayout(h)
```

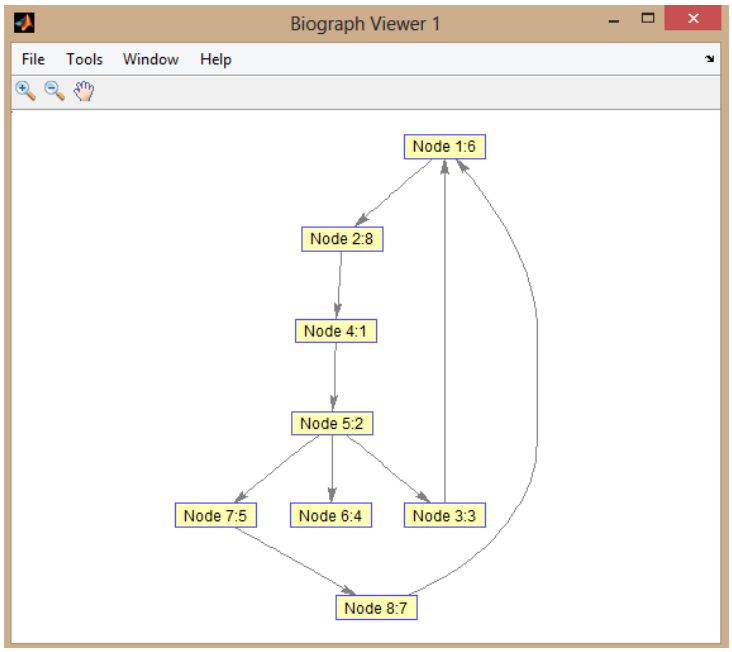

Figure 2.13. Graphe étiquité BFS – graphtraverse –

Recherchez et coloriez les nœuds proches du nœud 4 (à l'intérieur de deux arcs).

```
node_idxs = graphtraverse(GO,4,'depth',2)
```

```
node_idxs =

     4     5     3     6     7
```

```
set(h.nodes(node_idxs),'Color',[0 0 1])
```

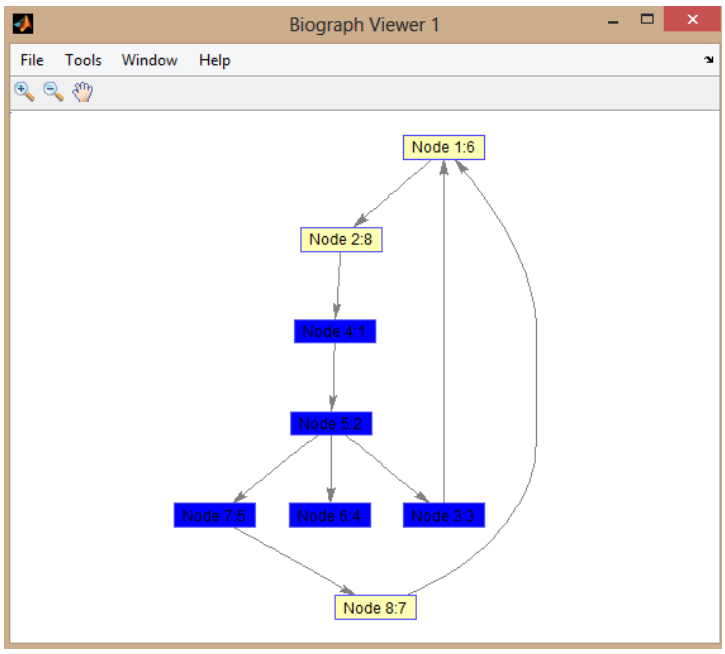

Figure 2.14. Graphe coloré – graphtraverse –