Chapitre 3

Problèmes du plus court chemin

Dans le chapitre 2, nous avons montré avec des illustrations les différentes opérations sur les graphes. Ce chapitre présente des illustrations sur les problèmes du plus court chemin. Les algorithmes adoptés par le logiciel Matlab sont : l'algorithme de Dijkstra, Bellman-Ford, Breadth-First Search et Acyclic.

3.1. La fonction graphshortestpath

Résoudre le problème du plus court chemin dans un graphe.

Syntaxe

```
[dist,path,pred] = graphshortestpath(G,S,D,'Name',NameValue,'Directed',DirectedVal
ue,                                    ...,'Method',MethodValue,'Weights',Weights
Value)
```

```
[dist,path,pred] = graphshortestpath(G,S)
```

Détermine les chemins les plus courts du nœud source S à tous les autres nœuds du graphe G. Ce dernier est représenté par une matrice d'adjacence $N \times N$. Les poids des arcs sont tous des entrées non nulles dans la matrice d'adjacence.

- `dist` retourne les distances du nœud source à tous les autres nœuds.
- `path` retourne les plus courts chemins vers chaque nœud.
- `pred` retourne les nœuds prédécesseurs des plus courts chemins.

```
[dist,path,pred] = graphshortestpath(G,S,D)
```

Détermine le plus court chemin entre le nœud source S et le nœud distination D et retourne l'un des arguments de sortie de la syntaxe précédente.

```
[...] = graphshortestpath(...,'Name',NameValue, ...)
```

Spécifie les options supplémentaires en utilisant un (plusieurs) paire(s) d'argument(s) Nom-valeurNom séparées par des virgules. Le nom doit apparaître entre apostrophes.

Arguments

Arguments d'entrée	
G	Graphe représenté par son formalisme mathématique sous-djacent qui est une matrice d'adjacence, spécifiée comme une matrice creuse $\boldsymbol{N} \times \boldsymbol{N}$. Les entrées non nulles dans la matrice représentent les poids des arrêtes. Types de données: double
S	Nœud source dans le graphe représenté par une matrice d'adjacence $\boldsymbol{N} \times \boldsymbol{N}$, spécifié comme indice de noeud numérique. Types de données : double.
D	Nœud de destination dans le graphe, spécifié comme indice de noeud numérique.Types de données : double.
DirectedValue	Propriété indiquant si le graphe est orienté ou non. Faire entrez `false` pour un graphe non orienté. Cela ignore le triangle supérieur de la matrice creuse. La valeur par défaut est `vraie`. Par défaut, la propriété est `true`. Types de données : logique.
MethodValue	Vecteur de chaîne de caractères (char | string) qui spécifie l'algorithme utilisé pour trouver le plus court chemin. Trois choix sont possibles: `'Bellman-Ford'` : Suppose que les poids des arcs sont des entrées non nulles dans la matrice d'adjacence $N \times N$. La complexité temporelle est $O\ (N\ \times\ E)$, où N et E sont respectivement le nombre de nœuds et d'arrêtes. `'BFS' Breadth-First Search` : Recherche en premier suppose que toutes les pondérations sont égales et que les entrées non nulles de la matrice d'adjacence $N \times N$ représentent les arcs. La complexité temporelle est $O\ (N + E)$. `'Acyclic' :` Suppose que le graphe représenté par la matrice d'adjacence $N \times N$, est un graphe acyclique orienté et que les poids des arcs sont des entrées non nulles dans la matrice d'adjacence. La complexité temporelle est $O\ (N + E)$. `'Dijkstra'` : Algorithme par défaut. Suppose que les poids des arcs sont des valeurs positives dans la matrice d'adjacence $N \times N$. La complexité temporelle est $O\ (log\ (N) \times\ E)$.
WeightsValue	Vecteur colonne qui spécifie des poids personnalisés pour les arcs de la matrice d'adjacence $N \times N$ d'un graphe. Il doit remplir les conditions suivantes : o Il doit avoir une entrée pour chaque valeur non nulle (arc) dans la matrice. o L'ordre des pondérations personnalisées dans le vecteur doit correspondre à l'ordre des valeurs non nulles dans la matrice d'adjacence $N \times N$ lorsqu'il est parcouru en colonne. Cette propriété vous permet d'utiliser des poids à valeur nulle. Par défaut, `graphshortestpath` obtient des informations de poids à partir des entrées non nulles dans la matrice d'adjacence. Types de données : double.

Tabeau 3.1. Arguments d'entrée – graphshortestpath –

Arguments de sortie	
dist	Distances entre le nœud source et tous les autres nœuds du graphe. Elles sont retournées sous forme de scalaire ou de vecteur numérique. o dist est retournée comme un scalaire si vous spécifiez un nœud de destination comme troisième argument d'entrée. o La fonction retourne Inf pour les nœuds non accessibles et 0 pour le nœud source.
path	Chemins les plus courts du nœud source vers tous les autres nœuds, retournés sous forme de tableau ou de vecteur. Il est retourné en tant que vecteur si vous spécifiez un nœud de destination. Chaque nombre représente un indive de nœud dans le graphe.
pred	Retourne les nœuds prédécesseurs des plus courts chemins.

Tabeau 3.2. Arguments de sortie – graphshortestpath –

Illustration 3.1

Rechercher les plus courts chemins dans un graph orienté.

Créez un graph orienté composé de 6 nœuds et 9 arcs.

```
Wei = [41 99 51 32 15 45 38 36 21];
GO = sparse([6 1 2 2 3 4 4 5 1],[2 6 3 5 4 1 6 4 5],Wei)
GO =

   (4,1)       45
   (6,2)       41
   (2,3)       51
   (3,4)       15
   (5,4)       36
   (1,5)       21
   (2,5)       32
   (1,6)       99
   (4,6)       38
```

Afficher le graphe.

```
h = view(biograph(GO,[],'ShowWeights','on'))
```

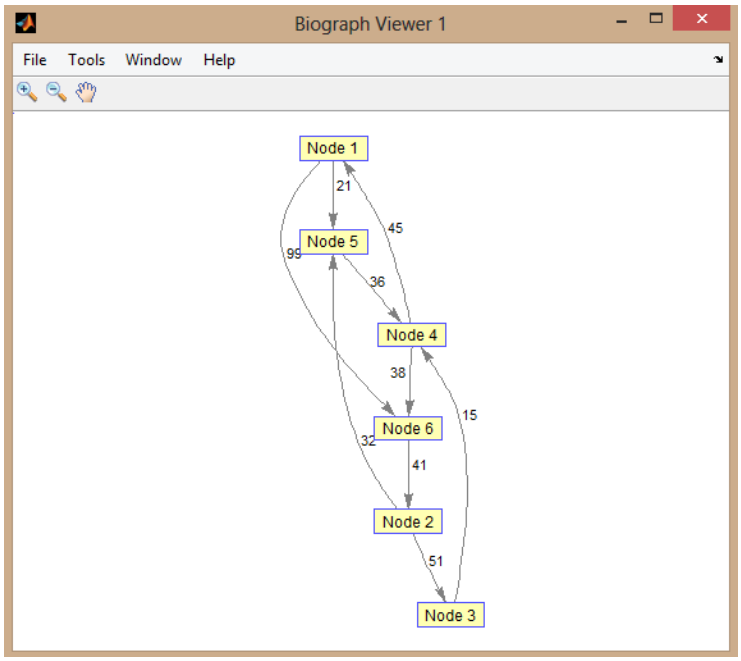

Figure 3.1. Graphe – graphshortestpath –

Trouvez le chemin le plus court du nœud 1 au nœud 6.

```
[dist,path,pred] = graphshortestpath(GO,1,3)

dist =

   187

path =

     1     5     4     6     2     3

pred =

     0     6     2     5     1     4
```

Marquez les nœuds et les arcs du plus court chemin en les colorant en rouge et en augmentant l'épisseur de la ligne.

```
set(h.Nodes(path),'Color',[1 0.7 0.7])
edges = getedgesbynodeid(h,get(h.Nodes(path),'ID'));
set(edges,'LineColor',[0 1 0])
set(edges,'LineWidth',1.75)
```

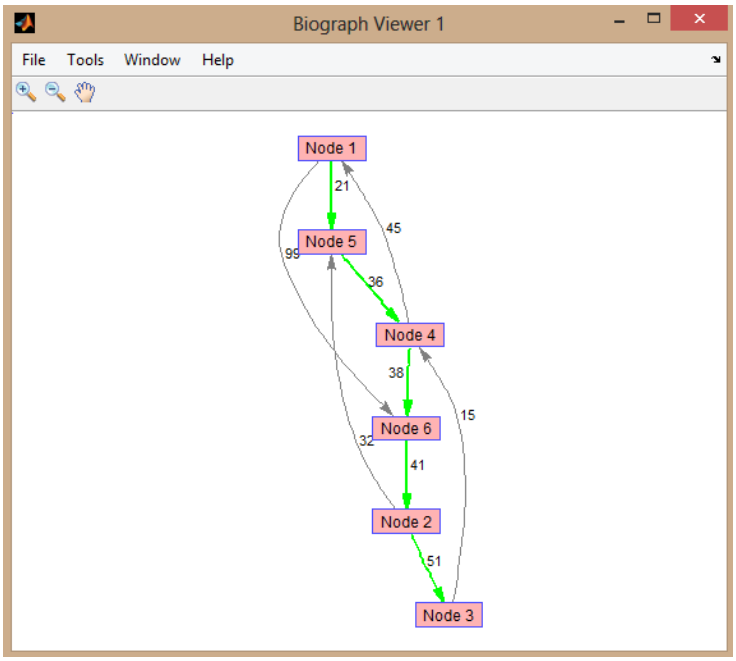

Figure 3.1. Coloration du plus court chemin – graphshortestpath –

`pred` retourne les nœuds prédécesseurs des plus courts chemins du nœud source 1, à tous les autres nœuds, pas seulement le nœud de destination spécifié. Vous pouvez utiliser `pred` pour interroger les plus courts chemins du nœud source vers tout autre nœud du graphe.

Par exemple, pour déterminer le plus court chemin entre le nœud 1 et le nœud 3 en utilisant les informations de `pred` :

- Interrogez `pred` avec le nœud de destination comme première requête.
- Utilisez ensuite la réponse renvoyée pour obtenir le nœud suivant.
- Répétez cette procédure jusqu'à ce que la réponse à la requête soit 0, ce qui indique le nœud source.

```
>> next = pred(3)
next =
     2
>> next = pred(next)
next =
     6
>> next = pred(next)
next =
     4
>> next = pred(next)
next =
     5
>> next = pred(next)
next =
     1
>> next = pred(next)
next =
     0
```

Les résultats indiquent que le plus court chemin entre le nœud 1 et le nœud 3 est `1-> 5-> 4-> 6-> 2-> 3`.

Illustration 3.2

Pour rechercher les plus courts chemins dans un graph non orienté.

Créez un graph non orienté composé de 6 nœuds et 9 arcs.

```
Wei = [41 99 51 32 15 45 38 36 21];
GO = sparse([6 1 2 2 3 4 4 5 1],[2 6 3 5 4 1 6 4 5],Wei)
% tril retourne la matrice creuse triangulaire inférieure.
GNO = tril(GO+GO')
GNO =

   (4,1)       45
   (5,1)       21
   (6,1)       99
   (3,2)       51
   (5,2)       32
   (6,2)       41
   (4,3)       15
   (5,4)       36
   (6,4)       38
```

Afficher le graphe.

```
h = view(biograph(GNO,[],'ShowArrows','off','ShowWeights','on'))
```

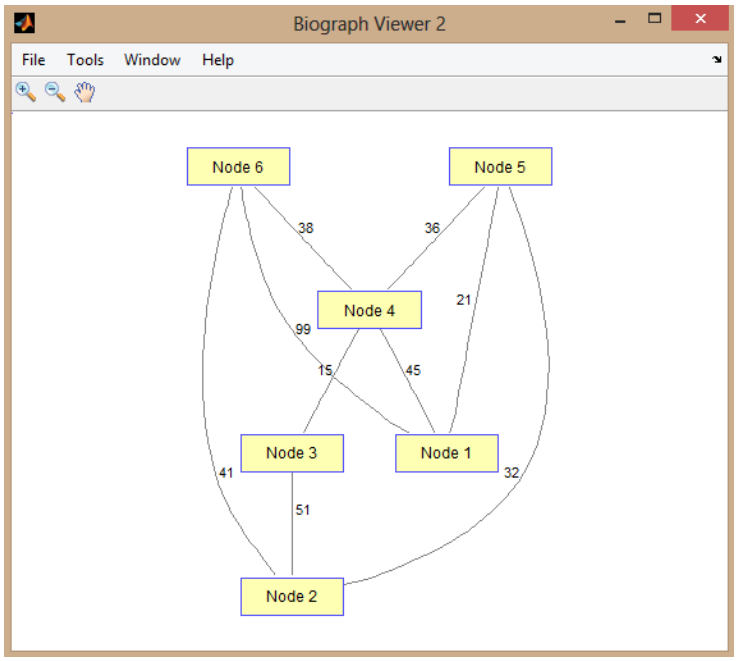

Figure 3.3. Graphe orienté à 6 nœuds – graphshortestpath –

Pour rechercher le plus court chemin entre le nœud 1 et le nœud 6. Définissez «`Directed`» sur `false` pour spécifier que le graphe n'est pas orienté.

```
[dist,path,pred] = graphshortestpath(GNO,1,3,'Directed',false)

dist =
```

```
    60

path =

     1     4     3

pred =

     0     5     4     1     1     4
```

Marquez les nœuds et les arrêtes du plus court chemin en les colorant en rouge et en augmentant l'épisseur de la ligne.

```
set(h.Nodes(path),'Color',[0 0.5 0.5])
fowEdges = getedgesbynodeid(h,get(h.Nodes(path),'ID'));
revEdges = getedgesbynodeid(h,get(h.Nodes(fliplr(path)),'ID'));
edges = [fowEdges;revEdges];
set(edges,'LineColor',[0 0 1])
set(edges,'LineWidth',2.5)
```

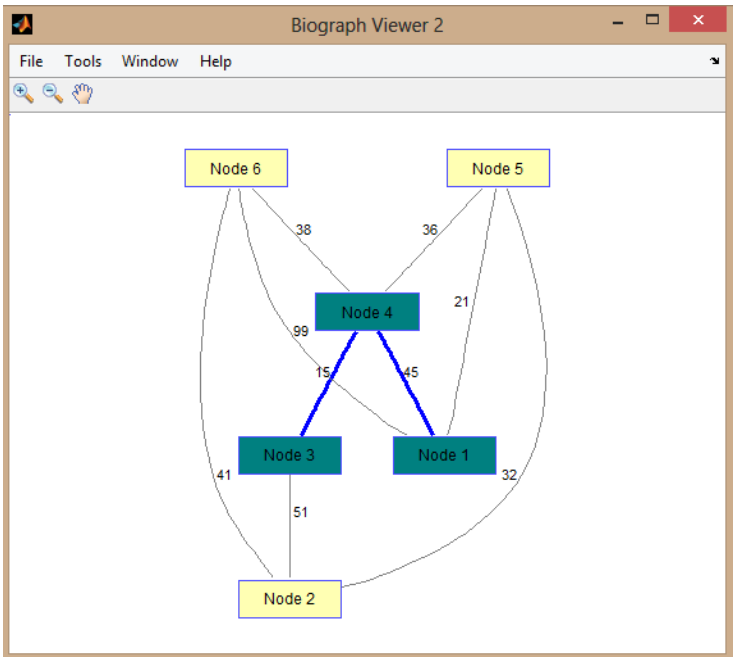

Figure 3.4. Coloration du plus court chemin du graphe orienté à 6 nœuds – graphshortestpath –

Vous pouvez utiliser `pred` pour déterminer les plus courts chemins entre le nœud source et tous les autres nœuds. Supposons que vous ayez un graphe orienté à 6 nœuds.

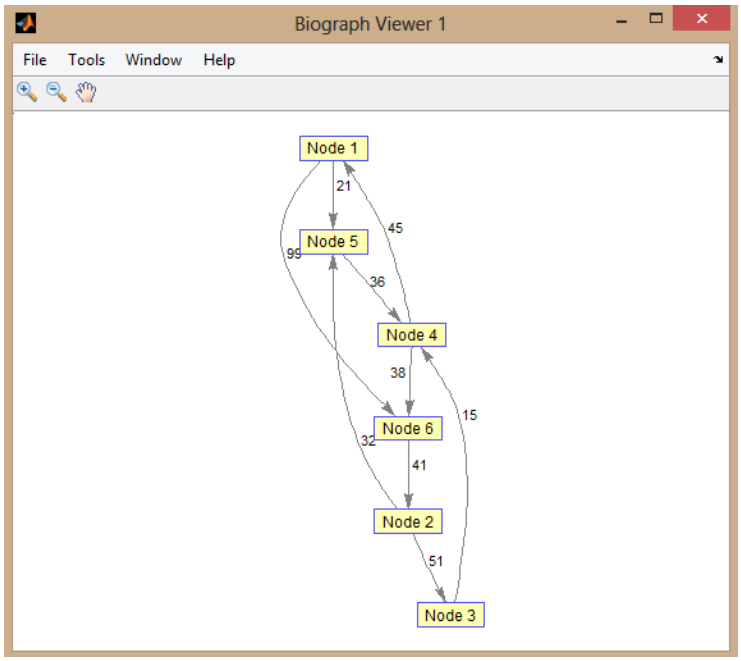

Figure 3.3. Graphe orienté de l'illustration 9.1 – graphshortestpath –

La fonction trouve que le plus court chemin entre le nœud 1 et le nœud 3 est `path = [1 5 4 6 2 3]` et `pred = [0 6 2 5 1 4]`. Vous pouvez maintenant déterminer les plus courts chemins entre le nœud 1 et tout autre nœud du graphe en l'indixant dans `pred`. Par exemple, pour déterminer le plus court chemin entre le nœud 1 et le nœud 6, vous pouvez interroger `pred` avec le nœud de destination comme première requête, puis utiliser la réponse renvoyée pour obtenir le nœud suivant. Répétez cette procédure jusqu'à ce que la réponse à la requête soit 0, ce qui indique le nœud source.

```
>> next=pred(6)

next =

     4

>> next=pred(next)

next =

     5

>> next=pred(next)

next =

     1

>> next=pred(next)

next =

     0
```

Les résultats indiquent que le chemin le plus court du nœud 1 au nœud 6 est `1-> 5-> 4-> 6`.

3.2. La fonction graphallshortestspaths

Trouver tous les plus courts chemins dans le graphe.

Syntaxe

```
[dist] =
graphallshortestpaths(G,'Directed', DirectedValue,'Weights', WeightsValue)
```

Détermine les plus courts chemins entre chaque paire de nœuds dans le graphe représenté par la matrice G, en utilisant l'algorithme de Johnson. L'entrée G est une matrice creuse $N \times N$ qui représente un graphique. Les entrées non nulles dans la matrice G représentent les poids des arcs.

L'algorithme de Johnson a une complexité temporelle de $O\ (N \times log\ (N)\ +\ N \times\ E)$, où N et E sont respectivement le nombre de nœuds et d'arcs.

Arguments

Arguments d'entrée	
G	Graphe représenté par son formalisme mathématique sous-djacent qui est une matrice d'adjacence, spécifiée comme une matrice creuse $\boldsymbol{N} \times \boldsymbol{N}$. Les entrées non nulles dans la matrice représentent les poids des arcs. Types de données: double
S	Nœud source dans le graphe représenté par une matrice d'adjacence $\boldsymbol{N} \times \boldsymbol{N}$, spécifié comme indice de noeud numérique. Types de données : double.
D	Nœud de destination dans le graphe, spécifié comme indice de noeud numérique.Types de données : double.
DirectedValue	Propriété indiquant si le graphe est orienté ou non. Faire entrez `false` pour un graphe non orienté. Cela ignore le triangle supérieur de la matrice creuse. La valeur par défaut est `vraie`. Par défaut, la propriété est `true`. Types de données : logique.
WeightsValue	Vecteur colonne qui spécifie des poids personnalisés pour les arrêtes de la matrice d'adjacence $N \times N$ d'un graphe. Il doit remplir les conditions suivantes : o Il doit avoir une entrée pour chaque valeur non nulle (arc) dans la matrice. o L'ordre des pondérations personnalisées dans le vecteur doit correspondre à l'ordre des valeurs non nulles dans la matrice d'adjacence $N \times N$ lorsqu'il est parcouru en colonne. Cette propriété vous permet d'utiliser des poids à valeur nulle. Par défaut, `graphallshortestspaths` obtient des informations de poids à partir des entrées non nulles dans la matrice d'adjacence. Types de données : double.

Tabeau 3.3. Arguments d'entrée – graphallshortestspaths –

Arguments de sortie	
dist	Matrice `N × N` où `dist(S,D)` est la distance du chemin le plus court entre le nœud source S et le nœud distination D. ○ Les éléments de la diagonale sont toujours 0, indiquant que le nœud source et le nœud distination sont les mêmes (boucle). ○ Un 0 hors la diagonale indique que la distance entre le nœud source et le nœud distination est 0. ○ Un Inf indique qu'il n'y a pas de chemin entre le nœud source et le nœud distination.
path	Les plus courts chemins du nœud source vers tous les autres nœuds, retournés sous forme de tableau ou de vecteur. Il est retourné en tant que vecteur si vous spécifiez un nœud de destination. Chaque nombre représente un indice de nœud dans le graphe.
pred	Retourne les nœuds prédécesseurs des plus courts chemins.

Tabeau 3.4. Arguments de sortie – graphallshortestspaths –

Illustration 3.3

Recherche de tous les plus courts chemins dans un graphe orienté.

Créez et affichez un graphe orienté composé de 6 nœuds et 9 arcs.

```
Wei = [41 99 51 32 15 45 38 36 21];
GO = sparse([6 1 2 2 3 4 4 5 1],[2 6 3 5 4 1 6 4 5],Wei)

GO =       (4,1)       45
           (6,2)       41
           (2,3)       51
           (3,4)       15
           (5,4)       36
           (1,5)       21
           (2,5)       32
           (1,6)       99
           (4,6)       38
view(biograph(GO,[],'ShowWeights','on'))
```

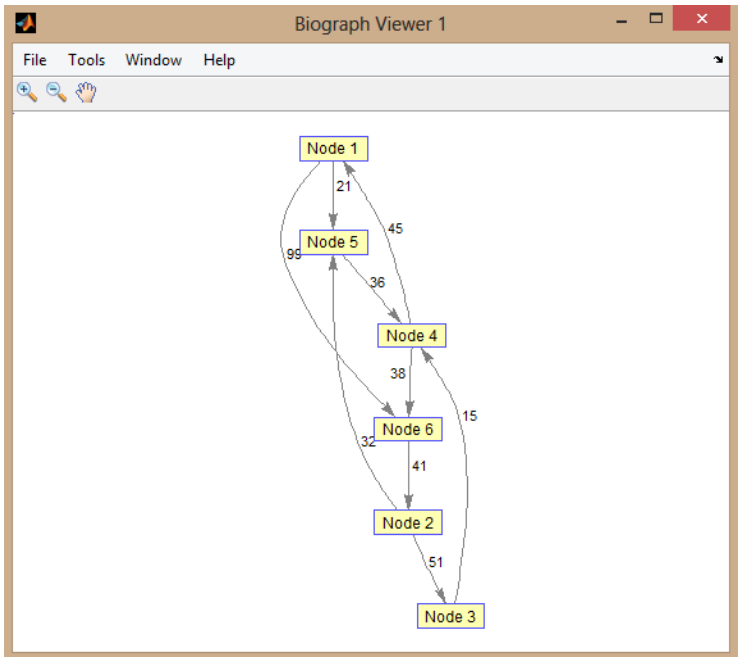

Figure 3.6. Graphe orienté – graphallshortestspaths –

Trouvez tous les plus courts chemins entre chaque paire de nœuds dans le graphe orienté.

```
graphallshortestpaths(GO)
ans =

     0   136   187    57    21    95
   111     0    51    66    32   104
    60    94     0    15    81    53
    45    79   130     0    66    38
    81   115   166    36     0    74
   152    41    92   107    73     0
```

La matrice résultante montre que le plus court chemin entre le nœud 1 (première ligne) et le nœud 6 (sixième colonne) est de 95. Vous pouvez le voir dans le graphe en traçant le chemin du nœud 1 au nœud 5 au nœud 4 au nœud 6 (21 + 36 + 38 = 95).

Illustration 3.4

Recherche de tous les plus courts chemins dans un graphe non orienté.

Créez et affichez un graphe non orienté composé de 6 nœuds et 9 arêtes.

```
GNO = tril(GO + GO')
GNO =
           (4,1)        45
           (5,1)        21
```

```
    (6,1)       99
    (3,2)       51
    (5,2)       32
    (6,2)       41
    (4,3)       15
    (5,4)       36
    (6,4)       38
```

```
view(biograph(GNO,[],'ShowArrows','off','ShowWeights','on'))
```

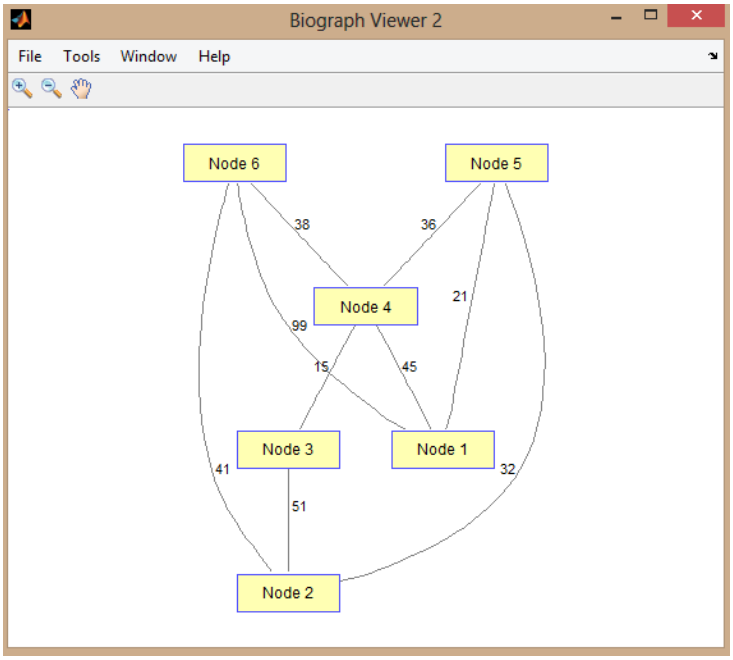

Figure 3.7. Graphe GNO – graphallshortestspaths –

Trouvez tous les plus courts chemins entre chaque paire de nœuds dans le graphe non orienté.

```
graphallshortestpaths(GNO,'directed',false)
```

```
ans =

     0    53    60    45    21    83

    53     0    51    66    32    41

    60    51     0    15    51    53

    45    66    15     0    36    38

    21    32    51    36     0    73

    83    41    53    38    73     0
```

La matrice résultante est symétrique car elle représente un graphe non orienté. Elle montre que le plus court chemin entre le nœud 1 (première ligne) et le nœud 6 (sixième colonne) est de 83. Vous pouvez le voir dans le graphe en traçant le chemin du nœud 1 au nœud 4 au nœud 6 (45 + 38 = 83). Parce que GNO est un graphe non orienté, nous pouvons utiliser l'arrête entre le nœud 1 et le nœud 4, ce que nous ne pouvions pas faire dans le graphe orienté GO.