Social Network Analysis
Lecture 2-Introduction Graph Theory

Donglei Du
(ddu@unb.ca)

Faculty of Business Administration, University of New Brunswick, NB Canada Fredericton E3B 9Y2
What is Graph theory?

Graph theory is the study of graphs, which are mathematical representations of a network used to model pairwise relations between objects. A graph consists of a set of “vertices” or “nodes”, with certain pairs of these nodes connected by “edges” (undirected) or “arcs” (directed). A graph may be undirected, meaning that there is no distinction between the two vertices associated with each edge, or directed, meaning that its arcs may be directed from one vertex to another.
What is Graph theory?

Graph theory is the study of graphs, which are mathematical representation of a network used to model pairwise relations between objects.
Graph theory is the study of graphs, which are mathematical representation of a network used to model pairwise relations between objects.

A graph consists of a set of "vertices" or "nodes", with certain pairs of these nodes connected by "edges" (undirected) or "arcs" (directed).
What is Graph theory?

- Graph theory is the study of graphs, which are mathematical representation of a network used to model pairwise relations between objects.

- A graph consists of a set of "vertices" or "nodes", with certain pairs of these nodes connected by "edges" (undirected) or "arcs" (directed).

- A graph may be undirected, meaning that there is no distinction between the two vertices associated with each edge, or directed, meaning that its arcs may be directed from one vertex to another.
What is Graph theory?

- Graph theory is the study of graphs, which are mathematical representation of a network used to model pairwise relations between objects.

- A graph consists of a set of "vertices" or "nodes", with certain pairs of these nodes connected by "edges" (undirected) or "arcs" (directed).

- A graph may be undirected, meaning that there is no distinction between the two vertices associated with each edge, or directed, meaning that its arcs may be directed from one vertex to another.

- *This is an extremely brief introduction of graph theory.*
The Seven Bridges of Königsberg:

Q: Is there an Eulerian trail through the city that would cross each bridge once and only once? (The second (undirected) graph represents the bridge network!)

A: A (connected) undirected graph has an Eulerian trail if and only if at most two vertices have odd degree. Leonhard Euler (15 April 1707-18 September 1783, Swiss Mathematician) in 1735 laid the foundations of graph theory and prefigured the idea of topology by studying this problem.
The Seven Bridges of Königsberg:

Q: Is there a Eulerian trail through the city that would cross each bridge once and only once? (The second (undirected) graph represents the bridge network!)

A: an (connected) undirected graph has an Eulerian trail if and only if at most two vertices have odd degree.
The Seven Bridges of Königsberg:

Q: Is there a Eulerian trail through the city that would cross each bridge once and only once? (The second (undirected) graph represents the bridge network!)

A: an (connected) undirected graph has an Eulerian trail if and only if at most two vertices have odd degree.

Leonhard Euler (15 April 1707-18 September 1783, Swiss Mathematician) in 1735 laid the foundations of graph theory and prefigured the idea of topology by studying this problem.
The Erdös coauthor network and Erdös number:

Paul Erdös (26 March 1913–20 September 1996, Hungarian mathematician): one of the most prolific publishers of papers in mathematical history, comparable only with Leonhard Euler; Erdös published more papers, mostly in collaboration with other mathematicians, while Euler published more pages, mostly by himself.
The Erdős coauthor network and Erdős number:

- Paul Erdős (26 March 1913–20 September 1996, Hungarian mathematician): one of the most prolific publishers of papers in mathematical history, comparable only with Leonhard Euler; Erdős published more papers, mostly in collaboration with other mathematicians, while Euler published more pages, mostly by himself.
My Erdös Number: 3
My Erdös Number: 3

- Paul Erdös (0) and Prasad Tetali (1), Representations of integers as the sum of \( k \) terms, *Random Structures Algorithms*, 1(3) (1990), 245-261.
My Erdös Number: 3

- Paul Erdös (0) and Prasad Tetali (1), Representations of integers as the sum of $k$ terms, *Random Structures Algorithms*, 1(3) (1990), 245-261.
My Erdös Number: 3

- Paul Erdös (0) and Prasad Tetali (1), Representations of integers as the sum of $k$ terms, *Random Structures Algorithms*, 1(3) (1990), 245-261.
- Qiaming Han, Donglei Du (3), Juan C. Vera (2) and Luis F. Zuluag, Improved bounds for the symmetric rendezvous search problem on the line, *Operations Research*, 56(3) (2008), 772-782.
My Erdős Number: 3

- **Paul Erdős (0) and Prasad Tetali (1)**, Representations of integers as the sum of \( k \) terms, *Random Structures Algorithms*, 1(3) (1990), 245-261.
- Find your own Erdős Number

Find your own Erdős Number

http://www.ams.org/mathscinet/collaborationDistance.html

If you want to know more about Erdős Number, try here:

http://www.oakland.edu/enp/compute/
My Erdős Number: 3

- Paul Erdős (0) and Prasad Tetali (1), Representations of integers as the sum of \( k \) terms, *Random Structures Algorithms*, 1(3) (1990), 245-261.
- Qiaming Han, Donglei Du (3), Juan C. Vera (2) and Luis F. Zuluag, Improved bounds for the symmetric rendezvous search problem on the line, *Operations Research*, 56(3) (2008), 772-782.
- Find your own Erdős Number
  - here: [http://www.ams.org/mathscinet/collaborationDistance.html](http://www.ams.org/mathscinet/collaborationDistance.html)
My Erdős Number: 3

- Paul Erdős (0) and Prasad Tetali (1), "Representations of integers as the sum of \( k \) terms," *Random Structures Algorithms*, 1(3) (1990), 245-261.


- Qiaming Han, Donglei Du (3), Juan C. Vera (2) and Luis F. Zuluag, "Improved bounds for the symmetric rendezvous search problem on the line," *Operations Research*, 56(3) (2008), 772-782.

- Find your own Erdős Number
  - here: [http://www.ams.org/mathscinet/collaborationDistance.html](http://www.ams.org/mathscinet/collaborationDistance.html)

- If you want to know more about Erdős Number, try here: [http://www.oakland.edu/enp/compute/](http://www.oakland.edu/enp/compute/)
Six Degrees of Kevin Bacon:

1995 — Ed Harris stars with Kevin Bacon in Apollo 13.

2003 — Nicole Kidman stars with Kevin Bacon in The Human Stain.

1996 — Viggo stars with Nicole Kidman in Portrait of a Lady.

2009 — Guy Pearce stars with Viggo Mortensen in The Road.

1997 — Simon Baker stars with Guy Pearce in LA Confidential.

1996 — Molly meets (dis has an argument over music) with Simon Baker at a party.

One random Sydney Cut and The Six Degrees of Kevin Bacon!
Six Degrees of Kevin Bacon:

- Kevin Norwood Bacon (July 8, 1958-) is an American actor and musician.
Types of graphs

Undirected vs directed:

- Undirected networks: coauthorship network, actor network, Königsberg Bridges Network, Facebook friendship network
- Directed networks: URLs on the www, phone calls, Retweet network
Types of graphs

- Undirected vs directed:

![Diagram showing undirected and directed graphs](image-url)
Types of graphs

- Undirected vs directed:

- Undirected networks: coauthorship network, actor network, Königsberg Bridges Network, Facebook friendship network
Types of graphs

- Undirected vs directed:

  - Undirected networks: coauthorship network, actor network, Königsberg Bridges Network, Facebook friendship network
  - Directed networks: URLs on the www, phone calls, Retweet network
Types of graphs
Types of graphs

- Simple vs multigraph: loops or multiedges
Types of graphs

Unweighted vs weighted:

- Fredericton
- Hampton
- Moncton
- St. Stephen
- Woodstock
- New Bandon

Mobile phone calls, Collaboration network

Donglei Du (UNB)
Social Network Analysis
11 / 1
Types of graphs

- Unweighted vs weighted:

![Diagram of weighted graph with cities and connection weights]
Types of graphs

- Unweighted vs weighted:

- Mobile phone calls, Collaboration network
Important graphs
Important graphs

- Regular graph
Important graphs

- Regular graph
- Complete graph
Important graphs

- Regular graph
- Complete graph
- Path
Important graphs

- Regular graph
- Complete graph
- Path
- Cycle
Important graphs

- Regular graph
- Complete graph
- Path
- Cycle
- Bipartite graph
Important graphs

- Regular graph
- Complete graph
- Path
- Cycle
- Bipartite graph
- Euler graph
Important graphs

- Regular graph
- Complete graph
- Path
- Cycle
- Bipartition graph
- Euler graph
- Hamilton graph
Important graphs

- Regular graph
- Complete graph
- Path
- Cycle
- Bipartite graph
- Euler graph
- Hamilton graph
- Planar graph
Important graphs

- Regular graph
- Complete graph
- Path
- Cycle
- Bipartitie graph
- Euler graph
- Hamilton graph
- Planar graph
- Tree and forest
Important graphs

- Regular graph
- Complete graph
- Path
- Cycle
- Bipartite graph
- Euler graph
- Hamilton graph
- Planar graph
- Tree and forest
- ...
Layout
Representation of graphs using different data structure (Hi, Computer Science guys)
Graph can be represented in many different ways for different purpose.
Graph can be represented in many different ways for different purpose

- Adjacency matrix
Graph can be represented in many different ways for different purposes:
- Adjacency matrix
- Edge list
Graph can be represented in many different ways for different purpose

- Adjacency matrix
- Edge list
- Adjacency list
Graph can be represented in many different ways for different purpose

- Adjacency matrix
- Edge list
- Adjacency list
- Laplace matrix
Adjacency matrix: undirected graph

The adjacency matrix for this undirected graph is:

\[
\begin{pmatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 \\
2 & 1 & 0 & 1 & 0 & 1 & 0 \\
3 & 0 & 1 & 0 & 1 & 0 & 0 \\
4 & 0 & 0 & 1 & 0 & 1 & 1 \\
5 & 1 & 1 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 1 & 0 & 0 \\
\end{pmatrix}
\]

The adjacency is symmetric for an undirected graph.
Adjacency matrix: undirected graph

Adjacency matrix: $A_{ij} = 1$ iff there is a link between $i$ and $j$.

\[
\begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 \\
2 & 1 & 0 & 1 & 0 & 1 & 0 \\
3 & 0 & 1 & 0 & 1 & 0 & 0 \\
4 & 0 & 0 & 1 & 0 & 1 & 1 \\
5 & 1 & 1 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 1 & 0 & 0 \\
\end{bmatrix}
\]
Adjacency matrix: undirected graph

- Adjacency matrix: \( A_{ij} = 1 \) iff there is a link between \( i \) and \( j \).

\[
\begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 \\
2 & 1 & 0 & 1 & 0 & 1 & 0 \\
3 & 0 & 1 & 0 & 1 & 0 & 0 \\
4 & 0 & 0 & 1 & 0 & 1 & 1 \\
5 & 1 & 1 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 1 & 0 & 0 \\
\end{bmatrix}
\]

- The adjacency is symmetric for undirected graph.
R code based on package igraph: generate graph from adjacency matrix

```r
library(igraph)
#Generate graph object from adjacency matrix
adjm_u<-matrix(
        c(0, 1, 0, 0, 1, 0,
        1, 0, 1, 0, 1, 0,
        0, 1, 0, 1, 0, 0,
        0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 0, 0,
        0, 0, 0, 1, 0, 0), # the data elements
        nrow=6, # number of rows
        ncol=6, # number of columns
        byrow = TRUE) # fill matrix by rows

g_adj_u <- graph.adjacency(adjm_u, mode="undirected")
tkplot(g_adj_u)
```
Adjacency matrix: directed graph

\[
A_{ij} = 1 \quad \text{iff there is a link from } j \text{ to } i
\]

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
\end{bmatrix}
\]

Note the direction of the edge runs from the second index to the first: counter-intuitive, but convenient mathematically!
Adjacency matrix: directed graph

Adjacency matrix: \( A_{ij} = 1 \) iff there is a link from \( j \) to \( i \)

\[
\begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 & 1 \\
3 & 0 & 1 & 0 & 1 \\
4 & 0 & 1 & 0 & 0 \\
5 & 0 & 1 & 0 & 0
\end{bmatrix}
\]

Note the direction of the edge runs from the second index to the first: counter-intuitive, but convenient mathematically!
library(igraph)

# Generate graph object from adjacency matrix
adjm_d <- matrix(
  c(0, 1, 0, 0, 0,
    0, 0, 1, 1, 1,
    0, 0, 0, 0, 0,
    0, 1, 1, 0, 0,
    0, 0, 0, 1, 0), # the data elements
  nrow=5, # number of rows
  ncol=5, # number of columns
  byrow = TRUE) # fill matrix by rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows

# Generate graph object from adjacency matrix

# number of columns

# fill matrix by rows

# number of rows
Edge list: undirected graph

1
2
3
4
5
6

Diagram:

- (1, 2)
- (1, 5)
- (2, 3)
- (2, 5)
- (3, 4)
- (4, 5)
- (4, 6)
Edge list: undirected graph

Edge list:

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

# Generate graph object from edge list
el_u <- matrix(c(1, 2, 1, 5, 2, 3, 2, 5, 3, 4, 4, 5, 4, 6), nc=2, byrow=TRUE)
g_el_u <- graph.edgelist(el_u, directed=FALSE)
tkplot(g_el_u)
Edge list: directed graph
Edge list: directed graph

Edge list:

12
23
24
42
25
43
54
library(igraph)

# Generate graph object from edge list
el_d <- matrix(c(1, 2, 2, 3, 2, 4, 4, 2, 2, 5, 4, 3, 5, 4), nc=2, byrow=TRUE)
g_el_d <- graph.edgelist(el_d, directed=TRUE)
tkplot(g_el_d)
Adjacency list: undirected graph

Easier to work with if network is large and parse, and quick in retrieving all neighbors for a node.
Adjacency list: undirected graph

- Adjacency list:

<table>
<thead>
<tr>
<th>node</th>
<th>neighbor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>2 5</td>
</tr>
<tr>
<td>2:</td>
<td>1 3 5</td>
</tr>
<tr>
<td>3:</td>
<td>2 4</td>
</tr>
<tr>
<td>4:</td>
<td>3 5 6</td>
</tr>
<tr>
<td>5:</td>
<td>1 2 4</td>
</tr>
<tr>
<td>6:</td>
<td>4</td>
</tr>
</tbody>
</table>
Adjacency list: undirected graph

- Adjacency list:

<table>
<thead>
<tr>
<th>node</th>
<th>neighbor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>2 5</td>
</tr>
<tr>
<td>2:</td>
<td>1 3 5</td>
</tr>
<tr>
<td>3:</td>
<td>2 4</td>
</tr>
<tr>
<td>4:</td>
<td>3 5 6</td>
</tr>
<tr>
<td>5:</td>
<td>1 2 4</td>
</tr>
<tr>
<td>6:</td>
<td>4</td>
</tr>
</tbody>
</table>

- Easier to work with if network is large and parse, and quick in retrieving all neighbors for a node
Adjacency list: directed graph

```
{1: [2],
  2: [1, 4, 3, 4, 5],
  3: [2, 4],
  4: [2, 5],
  5: [2, 4]}
```
Adjacency list: directed graph

Adjacency list:

<table>
<thead>
<tr>
<th>node</th>
<th>inneighbor</th>
<th>outneighbor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1, 4</td>
<td>3, 4, 5</td>
</tr>
<tr>
<td>3</td>
<td>2, 4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2, 5</td>
<td>2, 3</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
Graph-theoretic concepts
Graph-theoretic concepts

- Density
Graph-theoretic concepts

- Density
- Degree, indegree and outdegree
Graph-theoretic concepts

- Density
- Degree, indegree and outdegree
- Path and cycles
Graph-theoretic concepts

- Density
- Degree, indegree and outdegree
- Path and cycles
- Distance, diameter
Graph-theoretic concepts

- Density
- Degree, indegree and outdegree
- Path and cycles
- Distance, diameter
- Components
Graph-theoretic concepts

- Density
- Degree, indegree and outdegree
- Path and cycles
- Distance, diameter
- Components
- Clustering coefficient
Degree of any node $i$: the number of nodes adjacent to $i$. It can be calculated from the three different representations discussed earlier. Every loop adds two degrees to a node.
Degree of any node $i$: the number of nodes adjacent to $i$. It can be calculated from the three different representations discussed earlier.
Degree for undirected graph

- Degree of any node \( i \): the number of nodes adjacent to \( i \). It can be calculated from the three different representations discussed earlier.
- Every loop adds two degrees to a node.

<table>
<thead>
<tr>
<th>node</th>
<th>degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>
R code based on package igraph: degree

degree(g_adj_u)
Indegree and outdegree for directed graph

Every loop adds one degree to each of the indegree and outdegree of a node.
Indegree and outdegree for directed graph

- Indegree of any node $i$: the number of nodes destined to $i$.
Indegree and outdegree for directed graph

- Indegree of any node $i$: the number of nodes destined to $i$.
- Outdegree of any node $i$: the number of nodes originated at $i$.

<table>
<thead>
<tr>
<th>Node</th>
<th>Indegree</th>
<th>Outdegree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Indegree and outdegree for directed graph

- Indegree of any node $i$: the number of nodes destined to $i$.
- Outdegree of any node $i$: the number of nodes originated at $i$.
- Every loop adds one degree to each of the indegree and outdegree of a node.

<table>
<thead>
<tr>
<th>node</th>
<th>indegree</th>
<th>outdegree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
R code based on package igraph: degree

degree(g_adj_d, mode="in")
degree(g_adj_d, mode="out")
Walk and Path

A walk is a sequence of nodes in which each node is adjacent to the next one. A walk can intersect itself and pass through the same link repeatedly. Each time a link is crossed, it is counted separately.

One walk on the graph: 1, 2, 5, 1, 2, 3, 4.

In a directed network, the walk can follow only the direction of an arrow.

A path is a walk without passing through the same link more than once (e.g., 1, 2, 5, 4, 3).

Cycle: A path with the same start and end node (e.g., 1, 2, 5).
Walk and Path

- Walk: a walk is a sequence of nodes in which each node is adjacent to the next one.

Each time a link is crossed, it is counted separately.

A walk can intersect itself and pass through the same link repeatedly.

One walk on the graph:

1, 2, 5, 1, 2, 3, 4.

In a directed network, the walk can follow only the direction of an arrow.

A path is a walk without passing through the same link more than once (e.g., 1, 2, 5, 4, 3).

Cycle: A path with the same start and end node (e.g., 1, 2, 5).
Walk and Path

- **Walk**: a walk is a sequence of nodes in which each node is adjacent to the next one.
  - A walk can intersect itself and pass through the same link repeatedly. Each time a link is crossed, it is counted separately.

One walk on the graph: 1, 2, 5, 1, 2, 3, 4.

In a directed network, the walk can follow only the direction of an arrow.

A path is a walk without passing through the same link more than once (e.g., 1, 2, 5, 4, 3).

**Cycle**: A path with the same start and end node (e.g., 1, 2, 5).
Walk and Path

- **Walk**: a walk is a sequence of nodes in which each node is adjacent to the next one.
  - A walk can intersect itself and pass through the same link repeatedly. Each time a link is crossed, it is counted separately.
  - One walk on the graph: 1, 2, 5, 1, 2, 3, 4.

- **Path**: a walk without passing through the same link more than once (e.g., 1, 2, 5, 4, 3).

- **Cycle**: a path with the same start and end node (e.g., 1, 2, 5).
Walk and Path

- **Walk**: a walk is a sequence of nodes in which each node is adjacent to the next one.
  - A walk can intersect itself and pass through the same link repeatedly. Each time a link is crossed, it is counted separately.
  - One walk on the graph: 1, 2, 5, 1, 2, 3, 4.
- In a directed network, the walk can follow only the direction of an arrow.

A path is a walk without passing through the same link more than once (e.g., 1, 2, 5, 4, 3).

A cycle is a path with the same start and end node (e.g., 1, 2, 5).
Walk and Path

Walk: a walk is a sequence of nodes in which each node is adjacent to the next one

- A walk can intersect itself and pass through the same link repeatedly. Each time a link is crossed, it is counted separately.
- One walk on the graph: 1, 2, 5, 1, 2, 3, 4.
- In a directed network, the walk can follow only the direction of an arrow.

A path is a walk without passing through the same link more than once (e.g. 1, 2, 5, 4, 3.).
Walk and Path

- **Walk**: a walk is a sequence of nodes in which each node is adjacent to the next one.
  - A walk can intersect itself and pass through the same link repeatedly. Each time a link is crossed, it is counted separately.
  - One walk on the graph: 1, 2, 5, 1, 2, 3, 4.
  - In a directed network, the walk can follow only the direction of an arrow.
- A path is a walk without passing through the same link more than once (e.g., 1, 2, 5, 4, 3.).
- **Cycle**: A path with the same start and end node (e.g., 1, 2, 5)
Distance

The distance $d_{ij}$ (shortest path, geodesic path) between two nodes $i$ and $j$ is the number of edges along the shortest path connecting them.

For directed graphs, the distance from one node $A$ to another $B$ is generally different from that from $B$ to $A$. 

Donglei Du (UNB)
Social Network Analysis
Distance

- The distance $d_{ij}$ (shortest path, geodesic path) between two nodes $i$ and $j$ is the number of edges along the shortest path connecting them.
The distance $d_{ij}$ (shortest path, geodesic path) between two nodes $i$ and $j$ is the number of edges along the shortest path connecting them.

The Distance between every pair of nodes are

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
The distance $d_{ij}$ (shortest path, geodesic path) between two nodes $i$ and $j$ is the number of edges along the shortest path connecting them.

- The Distance between every pair of nodes are

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

- For directed graphs, the distance from one node $A$ to another $B$ is generally different from that from $B$ to $A$. 
R code based on package igraph: distance

\texttt{shortest.paths(g_adj_u)}
Number of walks of length $k$
Number of walks of length \( k \):

If \( A \) is the adjacency matrix of the directed or undirected graph \( G \), then the matrix \( A^k \) (i.e., the matrix product of \( k \) copies of \( A \)) has an interesting interpretation:

- The entry in row \( i \) and column \( j \) gives the number of (directed or undirected) walks of length \( k \) from vertex \( i \) to vertex \( j \).

This implies, for example, that the number of triangles in an undirected graph \( G \) is exactly the trace of \( A^3 / 3! \).
Number of walks of length $k$

- If $A$ is the adjacency matrix of the directed or undirected graph $G$, then the matrix $A^k$ (i.e., the matrix product of $k$ copies of $A$) has an interesting interpretation:
- the entry in row $i$ and column $j$ gives the number of (directed or undirected) walks of length $k$ from vertex $i$ to vertex $j$. This implies, for example, that the number of triangles in an undirected graph $G$ is exactly the trace of $A^3/3!$. 

Donglei Du (UNB)
Social Network Analysis
Number of walks of length $k$

- If $A$ is the adjacency matrix of the directed or undirected graph $G$, then the matrix $A^k$ (i.e., the matrix product of $k$ copies of $A$) has an interesting interpretation:
  - the entry in row $i$ and column $j$ gives the number of (directed or undirected) walks of length $k$ from vertex $i$ to vertex $j$.
  - This implies, for example, that the number of triangles in an undirected graph $G$ is exactly the trace of $A^3/3!$. 

Connectivity and connected components for undirected graph

A graph is connected if there is a path between every pair of nodes. A connected component is a subgraph in which any two nodes are connected to each other by paths, and which is connected to no additional nodes in the original graph.

Largest Component: Giant Component

Bridge: an edge whose deletion increases the number of connected components.

Figure: A graph with three connected components
Connectivity and connected components for undirected graph

A graph is connected if there is a path between every pair of nodes.

Figure: A graph with three connected components
Connectivity and connected components for undirected graph

- A graph is connected if there is a path between every pair of nodes.
- A connected component is a subgraph in which any two nodes are connected to each other by paths, and which is connected to no additional nodes in the original graph.

Figure: A graph with three connected components
Connectivity and connected components for undirected graph

- A graph is connected if there is a path between every pair of nodes.
- A connected component is a subgraph in which any two nodes are connected to each other by paths, and which is connected to no additional nodes in the original graph.
- Largest Component: Giant Component

Figure: A graph with three connected components
Connectivity and connected components for undirected graph

- A graph is connected if there is a path between every pair of nodes.
- A connected component is a subgraph in which any two nodes are connected to each other by paths, and which is connected to no additional nodes in the original graph.
- Largest Component: Giant Component
- Bridge: an edge whose deletion increases the number of connected components.

Figure: A graph with three connected components
Connectivity and connected components for directed graph

Figure: A graph with three strongly connected components
Connectivity and connected components for directed graph

A directed graph is strongly connected if there is a path from any node to each other node, and vice versa.

Figure: A graph with three strongly connected components
Connectivity and connected components for directed graph

A directed graph is strongly connected if there is a path from any node to each other node, and vice versa.

A directed graph is weakly connected if it is connected by ignoring the edge directions.

Figure: A graph with three strongly connected components
Connectivity and connected components for directed graph

- A directed graph is strongly connected if there is a path from any node to each other node, and vice versa.
- A directed graph is weakly connected if it is connected by ignoring the edge directions.
- A strongly connected component of a directed graph $G$ is a subgraph that is strongly connected, and is maximal with this property: no additional edges or vertices from $G$ can be included in the subgraph without breaking its property of being strongly directed.

Figure: A graph with three strongly connected components
For any given node $A$ and two randomly selected nodes $B$ and $C$:

$$CC(A) = P(B \in N(C) | B, C \in N(A)) = P(two \text{ randomly selected friends of } A \text{ are friends}) = P(fraction \text{ of pairs of } A's \text{ friends that are linked to each other})$$.

For example, in the Figure above, node $A$ has two friends $B$ and $C$, and another friend $D$ is not connected to $B$ and $C$. Therefore, the clustering coefficient for node $A$ is $1/3$. 
Clustering coefficient

For any given node $A$ and two randomly selected nodes $B$ and $C$:

$$CC(A) = \mathbb{P}(B \in N(C)|B, C \in N(A))$$
$$= \mathbb{P}(\text{two randomly selected friends of } A \text{ are friends})$$
$$= \mathbb{P}(\text{fraction of pairs of } A's \text{ friends that are linked to each other}).$$
Clustering coefficient

- For any given node $A$ and two randomly selected nodes $B$ and $C$:

$$CC(A) = \mathbb{P}(B \in N(C) | B, C \in N(A)) = \mathbb{P}(\text{two randomly selected friends of } A \text{ are friends}) = \mathbb{P}(\text{fraction of pairs of } A\text{'s friends that are linked to each other}).$$

- For example, in the Figure above,

<table>
<thead>
<tr>
<th>node</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>$1/3$</td>
</tr>
<tr>
<td>C</td>
<td>$1/3$</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
</tr>
</tbody>
</table>
R code based on package igraph: Clustering coefficient

```r
# First generate the graph from edge list
el_cc <- matrix( c("A", "B", "A","C", "B", "C", "B","E","D","E","C","D"), nc=2, byrow=TRUE)
g_el_cc <- graph.edgelist(el_cc,directed=FALSE)
# Then calculate CC
transitivity(g_el_cc, type="localundirected")
```
Duncan Watts - The Myth of Common Sense

http://www.youtube.com/watch?feature=player_detailpage&v=D9XF0QOzWM0
References I