## CS4423 - Networks

Prof. Götz Pfeiffer
School of Mathematics, Statistics and Applied Mathematics
NUI Galway

# Lecture 4: New Networks from Old

In many situations, simple graphs are preferred over directed graphs. A simple method of turning a directed graph into a simple graph is given by ignoring the directions of the arrows, i.e., by reading each directed edge as an undirected edge. Some information will get lost on the way. Other methods of producing a simple graph from a directed one are based on composition of relations. We will see that the same method can be used to produce simple projections from bipartite graphs.

## Product of Relations

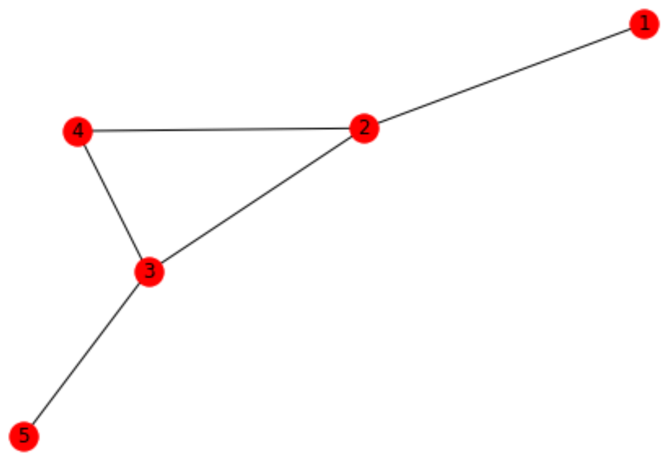
Suppose that $A$ is the (square) matrix adjacency matrix of a (undirected) network, corresponding to a relation $R$.

In [1]:
```
import networkx as nx
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
G = nx.Graph()
```

In [3]:
```
G.add_edges_from([(1,2),(2,3),(2,4), (3,4),(3,5)])
```

In [4]:
```
nx.draw(G, with_labels=True)
```



In [5]:
```
A = nx.adjacency_matrix(G)
```

In [6]:
```
print(A.todense())
```

```
[[0 1 0 0 0]
 [1 0 1 1 0]
 [0 1 0 1 1]
 [0 1 1 0 0]
 [0 0 1 0 0]]
```

In [7]:
```
AA = A * A
```

In [8]:
```
print(AA.todense())
```

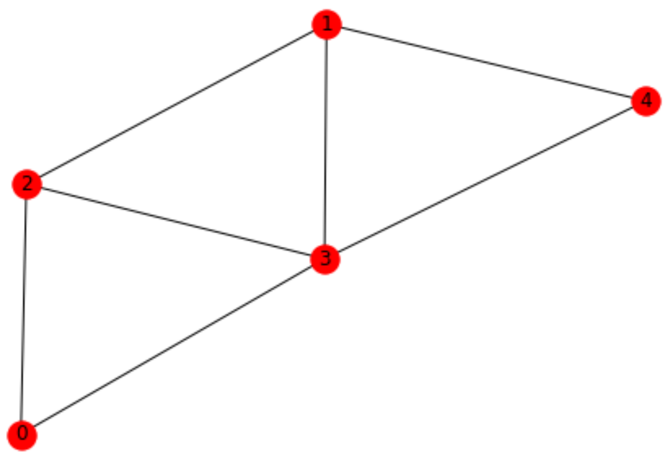
```
[[1 0 1 1 0]
 [0 3 1 1 1]
 [1 1 3 1 0]
 [1 1 1 2 1]
 [0 1 0 1 1]]
```

In [9]:
```
AA[AA>0] = 1
```

In [10]:
```
print(AA.todense())
```

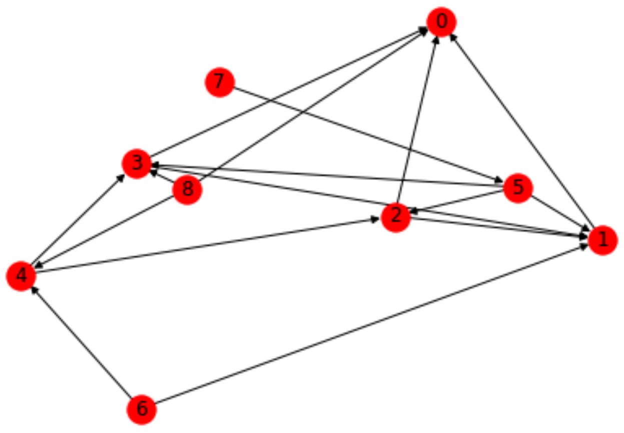
```
[[1 0 1 1 0]
 [0 1 1 1 1]
 [1 1 1 1 0]
 [1 1 1 1 1]
 [0 1 0 1 1]]
```

In [11]:
```
GG = nx.from_scipy_sparse_matrix(AA)
nx.draw(GG, with_labels = True)
```



## Directed Graphs

In [12]:
```
G = nx.DiGraph()
G.add_nodes_from(range(9))
G.add_edges_from([(1,0), (2,0), (2,1), (3,1), (3,0), (4,2), (4,3), (5,1), (5
,2), (5,3),
               (6,1), (6,4), (7,5), (8, 0), (8,3), (8,4)])
nx.draw(G, with_labels=True)
```



In [13]:
```
list(G.successors(2))
```

Out[13]:
```
[0, 1]
```

In [14]:
```
G.nodes()
```

Out[14]:
```
NodeView((0, 1, 2, 3, 4, 5, 6, 7, 8))
```

In [15]:
```
A = nx.adjacency_matrix(G)
print(A.todense())
```

```
[[0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0]
 [1 1 0 0 0 0 0 0 0]
 [1 1 0 0 0 0 0 0 0]
 [0 0 1 1 0 0 0 0 0]
 [0 1 1 1 0 0 0 0 0]
 [0 1 0 0 1 0 0 0 0]
 [0 0 0 0 0 1 0 0 0]
 [1 0 0 1 1 0 0 0 0]]
```

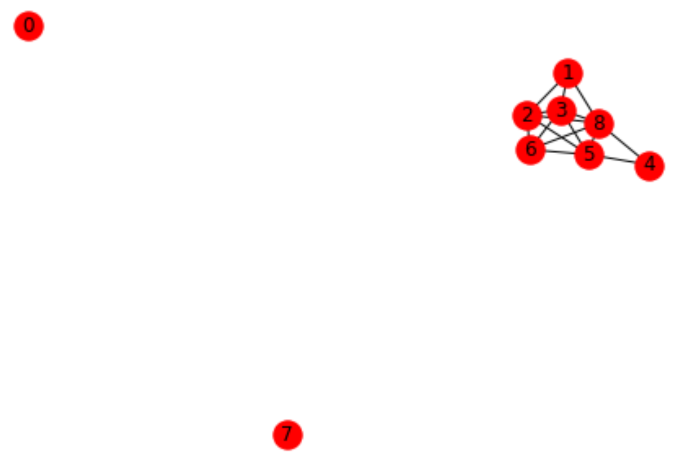
In [16]:
```
At = A.transpose()
print(At.todense())
```

```
[[0 1 1 1 0 0 0 0 1]
 [0 0 1 1 0 1 1 0 0]
 [0 0 0 0 1 1 0 0 0]
 [0 0 0 0 1 1 0 0 1]
 [0 0 0 0 0 0 1 0 1]
 [0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0]]
```

In [17]:
```
AAt = A * At
print(AAt.todense())
```

```
[[0 0 0 0 0 0 0 0 0]
 [0 1 1 1 0 0 0 0 1]
 [0 1 2 2 0 1 1 0 1]
 [0 1 2 2 0 1 1 0 1]
 [0 0 0 0 2 2 0 0 1]
 [0 0 1 1 2 3 1 0 1]
 [0 0 1 1 0 1 2 0 1]
 [0 0 0 0 0 0 0 1 0]
 [0 1 1 1 1 1 1 0 3]]
```

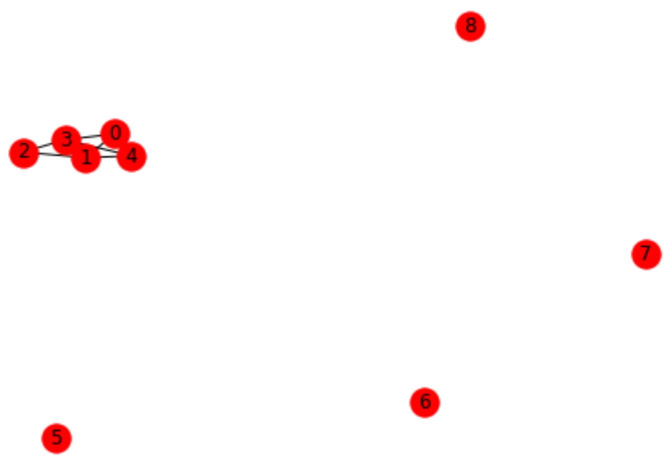
In [18]:
```
GGt = nx.from_scipy_sparse_matrix(AAt)
nx.draw(GGt, with_labels=True)
```



In [19]:
```
AtA = At * A
print(AtA.todense())
```

```
[[4 2 0 1 1 0 0 0 0]
 [2 4 1 1 1 0 0 0 0]
 [0 1 2 2 0 0 0 0 0]
 [1 1 2 3 1 0 0 0 0]
 [1 1 0 1 2 0 0 0 0]
 [0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0]]
```

In [20]:
```
GtG = nx.from_scipy_sparse_matrix(AtA)
nx.draw(GtG, with_labels=True)
```



## Bipartite Graphs

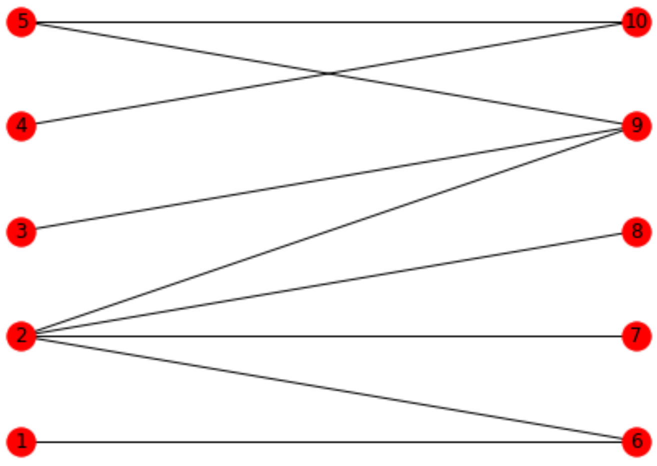
A (simple) graph $G = (X, E)$ is called **bipartite**, if the vertex set $X$ is a disjoint union of two sets $B$ (of black nodes) and $W$ (of white nodes) so that each edge in $E$ links a black vertex with a white vertex.

In [21]:
```
B = nx.Graph([(1,6), (2,6), (2,7), (2,8), (2,9),
              (3,9), (4,10), (5,9), (5,10)])
```

In [22]:
```
pos = [divmod(x, 5) for x in range(10)]
pos
```

Out[22]:
```
[(0, 0),
 (0, 1),
 (0, 2),
 (0, 3),
 (0, 4),
 (1, 0),
 (1, 1),
 (1, 2),
 (1, 3),
 (1, 4)]
```

```
In [23]: nx.draw(B, [""] + pos, with_labels=True)
```


5
4
3
2
1
10
9
8
7
6