CS4423 - Networks

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

Lecture 5: Bipartite Graphs and Projections

We'll look at further properties of graphs and networks, both from a theoretical point of views and from the practical side of handling graphs in the NetworkX environment. Start by importing the necessary python libraries into this jupyter notebook.

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

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.

Here is a sample bipartite graph B, specified to the `Graph` constructor by its edge list.

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

In this graph, the *white* nodes can be taken as the set $\{1, 2, \ldots, 5\}$ and the *black* nodes as $\{6, 7, \ldots, 10\}$. The drawing command `nx.draw` takes as optional argument a dictionary `pos` that specifies for each node a (relative) position in the drawing. Here, the node is the key and the position is a pair of x,y-coordinates. In this example we can use the (integer) quotient and remainder, as returned by the python method `divmod` to quickly compute a dictionary of positions that have the white nodes on the left, and the black nodes on the right.

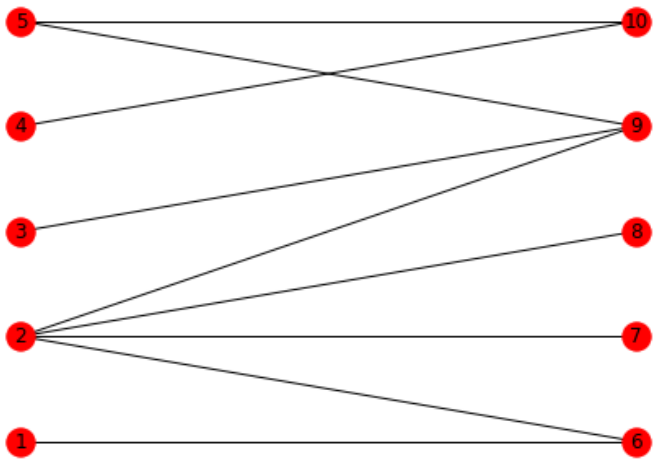
In [3]:
```
divmod(7, 5)
```

Out[3]:
```
(1, 2)
```

In [4]:
```
pos = {x + 1: divmod(x, 5) for x in range(10)}
pos
```

Out[4]:
```
{1: (0, 0),
 2: (0, 1),
 3: (0, 2),
 4: (0, 3),
 5: (0, 4),
 6: (1, 0),
 7: (1, 1),
 8: (1, 2),
 9: (1, 3),
 10: (1, 4)}
```

In [5]:
```
nx.draw(B, pos, with_labels=True)
```

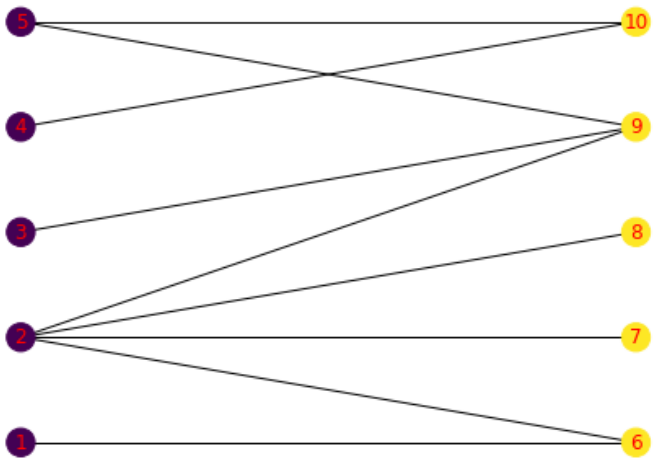


Node colors can be specified as a *list* assigned to the keyword argument `node_color` . We can use the x-coordinates of the node positions for that purpose.

In [6]:
```
color = [pos[x][0] for x in B.nodes()]
color
```

Out[6]:
```
[0, 1, 0, 1, 1, 1, 0, 0, 1, 0]
```

In [7]:
```
nx.draw(B, pos, with_labels=True, node_color=color, font_color='r')
```



A **(vertex)-coloring** of a graph G is an assignment of (finitely many) colors to the nodes of G, so that any two nodes which are connected by an edge have *different* colors.

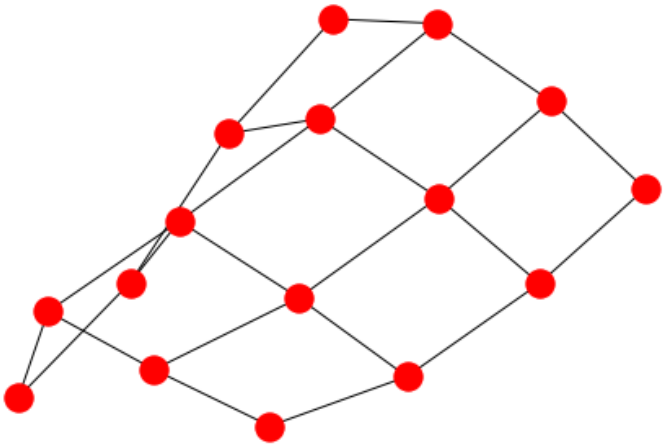
A graph is called N**-colorable**, if it has a vertex coloring with (at most) N colors.

Theorem. Let G be a graph. The following are equivalent:

- G is bipartite;
- G is 2-colorable;
- each cycle in G has even length. (See below for **cycle** and **length**)

2D grids are naturally bipartite:

In [8]:
```
G44 = nx.grid_2d_graph(4, 4)
nx.draw(G44)
```

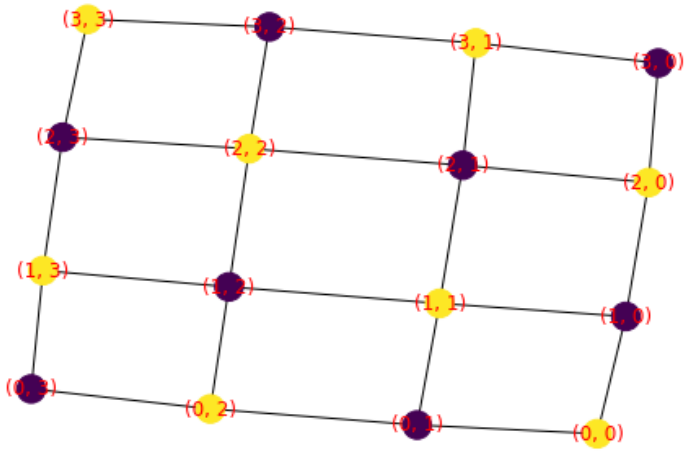


The method `nx.bipartite.color` determines a 2-coloring of a graph G algorithmically, if it exists, i.e. if G is bipartite.

In [9]:
```
color = nx.bipartite.color(G44)
color
```

Out[9]:
```
{(0, 0): 1,
 (1, 0): 0,
 (0, 1): 0,
 (1, 1): 1,
 (0, 2): 1,
 (1, 2): 0,
 (0, 3): 0,
 (1, 3): 1,
 (2, 3): 0,
 (3, 3): 1,
 (2, 2): 1,
 (3, 2): 0,
 (2, 1): 0,
 (3, 1): 1,
 (2, 0): 1,
 (3, 0): 0}
```

In [10]:
```
color = [color[x] for x in G44.nodes()]
nx.draw(G44, with_labels=True, node_color=color, font_color='r')
```



Affiliation Networks and Projections

Bipartite graphs arise in practice as models for **affiliation networks**. In such a network, the *black* nodes are people, and the *white* nodes are attributes of the people, such as common interests (books bought online), workplaces, social events attended ... Edges in such network connect people with their attributes.

A frequently cited example form the sociology literature (Davis, A., Gardner, B., and Gardner, R. 1941. Deep South. Chicago: University of Chicago Press.) is the **Southern Women Network**. This is a data set of 18 women observed over a nine-month period. During that period, various subsets of these women met in a series of 14 informal social events. The data recorded which women met for which events.

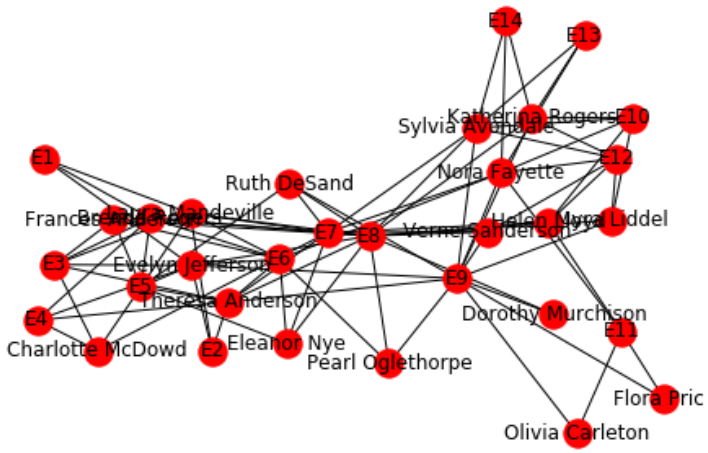
Names of Participants of Group I	Code Numbers and Dates of Social Events Reported in *Old City Herald*													
	(1) 6/27	(2) 3/2	(3) 4/12	(4) 9/26	(5) 2/25	(6) 5/19	(7) 3/15	(8) 9/16	(9) 4/8	(10) 6/10	(11) 2/23	(12) 4/7	(13) 11/21	(14) 8/3
1. Mrs. Evelyn Jefferson	X	X	X	X	X	X	.	X	X	.	.	.	.	.
2. Miss Laura Mandeville	X	X	X	.	X	X	X	X	.	.	.	.	.	.
3. Miss Theresa Anderson	.	X	X	X	X	X	X	X	X	.	.	.	.	.
4. Miss Brenda Rogers	X	.	X	X	X	X	X	X	.	.	.	.	.	.
5. Miss Charlotte McDowd	.	.	X	X	X	.	X	.	.	.	.	.	.	.
6. Miss Frances Anderson	.	.	X	.	X	X	.	X	.	.	.	.	.	.
7. Miss Eleanor Nye	.	.	.	.	X	X	X	X	.	.	.	.	.	.
8. Miss Pearl Oglethorpe	.	.	.	.	.	X	.	X	X	.	.	.	.	.
9. Miss Ruth DeSand	.	.	.	.	X	.	X	X	X	.	.	.	.	.
10. Miss Verne Sanderson	.	.	.	.	.	.	X	X	X	.	.	X	.	.
11. Miss Myra Liddell	.	.	.	.	.	.	.	X	X	X	.	X	.	.
12. Miss Katherine Rogers	.	.	.	.	.	.	.	X	X	X	.	X	X	X
13. Mrs. Sylvia Avondale	.	.	.	.	.	.	X	X	X	X	.	X	X	X
14. Mrs. Nora Fayette	.	.	.	.	.	X	X	.	X	X	X	X	X	X
15. Mrs. Helen Lloyd	.	.	.	.	.	.	X	X	.	X	X	X	.	.
16. Mrs. Dorothy Murchison	.	.	.	.	.	.	.	X	X	.	.	.	.	.
17. Mrs. Olivia Carleton	.	.	.	.	.	.	.	.	X	.	X	.	.	.
18. Mrs. Flora Price	.	.	.	.	.	.	.	.	X	.	X	.	.	.

The resulting bipartite graph on the vertex set consisting of the 18 woman and the 14 events is readily available in NetworkX.

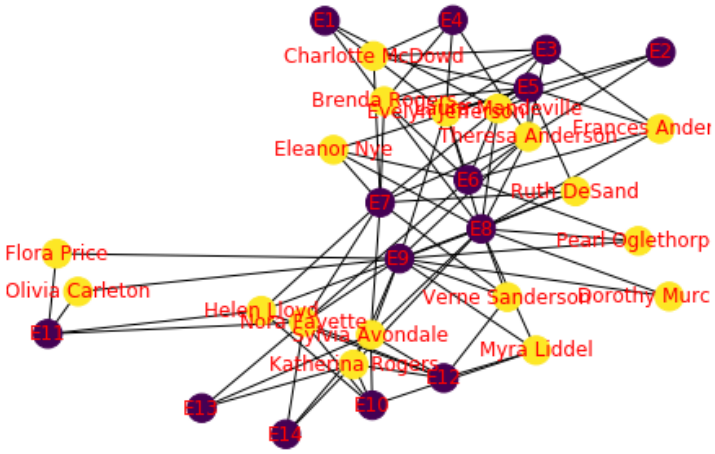
In [11]:
```
G = nx.generators.social.davis_southern_women_graph()
list(G.nodes())
```

Out[11]:
```
['Evelyn Jefferson',
 'Laura Mandeville',
 'Theresa Anderson',
 'Brenda Rogers',
 'Charlotte McDowd',
 'Frances Anderson',
 'Eleanor Nye',
 'Pearl Oglethorpe',
 'Ruth DeSand',
 'Verne Sanderson',
 'Myra Liddel',
 'Katherina Rogers',
 'Sylvia Avondale',
 'Nora Fayette',
 'Helen Lloyd',
 'Dorothy Murchison',
 'Olivia Carleton',
 'Flora Price',
 'E1',
 'E2',
 'E3',
 'E4',
 'E5',
 'E6',
 'E7',
 'E8',
 'E9',
 'E10',
 'E11',
 'E12',
 'E13',
 'E14']
```

In [12]:
```
nx.draw(G, with_labels=True)
```



In [13]:
```
color = nx.bipartite.color(G)
color = [color[x] for x in G.nodes()]
nx.draw(G, with_labels=True, node_color=color, font_color='r')
```



Note. The adjacency matrix A of a bipartite graph G, with respect to a suitable ordering of the vertices (B first, then W), has the form of a 2×2-block matrix,

$$A = \begin{pmatrix} 0 & C \\ C^T & 0 \end{pmatrix}$$

where the blocks on the diagonal consist entirely of zeros, as there are no edges between vertices of the same color, and the lower left block is the *transpose* of the matrix C of entries in the upper right.

In [14]:
```
A = nx.adjacency_matrix(G)
print(A.todense())
```

```
[[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]
 [0 0 0 ... 0 0 0]]
```

In [15]:
```
import numpy as np
with np.printoptions(threshold=9999):
    print(A.todense())
```

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

In NetworkX, all parts of a graph can have **attributes**: the nodes, the edges, and the graph object itself. Graph object attributes of a graph `G` are stored in the field `G.graph` . By convention, the two underlying sets of a bipartite graph are stored here as attributes called `'top'` and `'bottom'` .

In [16]:
```
X, Y = G.graph['top'], G.graph['bottom']
C = nx.bipartite.biadjacency_matrix(G, X, Y)
print(C.todense())
```

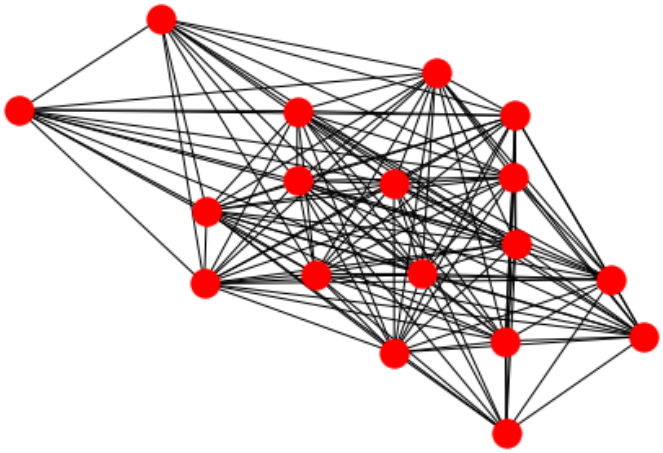
```
[[1 1 1 1 1 1 0 1 1 0 0 0 0 0]
 [1 1 1 0 1 1 1 1 1 0 0 0 0 0]
 [0 1 1 1 1 1 1 1 1 1 0 0 0 0]
 [1 0 1 1 1 1 1 1 1 0 0 0 0 0]
 [0 0 1 1 1 0 1 0 0 0 0 0 0 0]
 [0 0 1 0 1 1 0 1 0 0 0 0 0 0]
 [0 0 0 0 1 1 1 1 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 1 1 0 0 0 0 0]
 [0 0 0 0 1 0 1 1 1 0 0 0 0 0]
 [0 0 0 0 0 0 1 1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

As $A = A^T$, we get

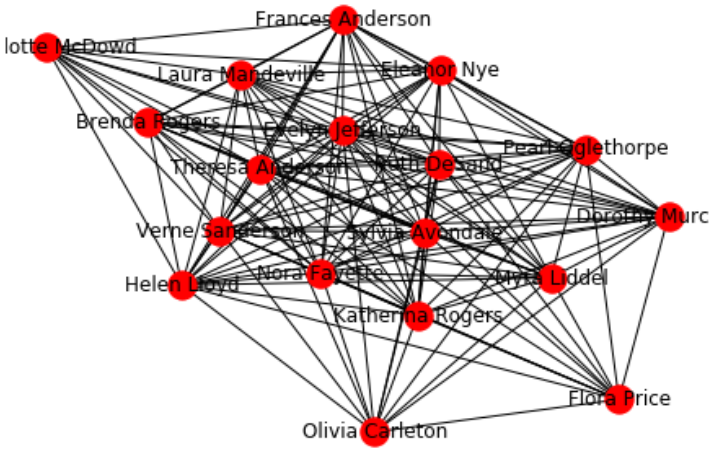
$$A^T \cdot A = A \cdot A^T = A \cdot A = \begin{pmatrix} C \cdot C^T & 0 \\ 0 & C^T \cdot C \end{pmatrix}$$

where $C \cdot C^T$ is the adjacency matrix of the **projection** onto the vertex set B, and $C^T \cdot C$is the adjacency matrix of the **projection** onto the vertex set W.

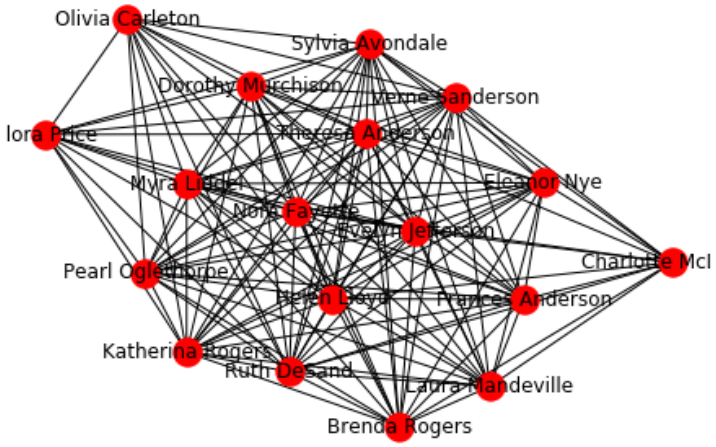
In [17]:
```
BB = nx.from_numpy_matrix((C*C.transpose()).todense())
nx.draw(BB)
```



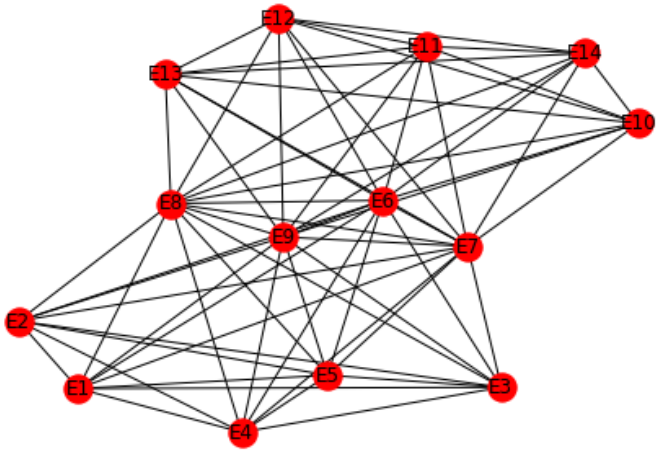
In [18]:
```
nodes = G.graph['top']
mapping = {i: nodes[i] for i in range(len(nodes))}
nx.relabel_nodes(BB, mapping, False)
nx.draw(BB, with_labels=True)
```



In [19]:
```
BBB = nx.bipartite.projected_graph(G, G.graph['top'])
nx.draw(BBB, with_labels=True)
```



In [20]:
```
WWW = nx.bipartite.projected_graph(G, G.graph['bottom'])
nx.draw(WWW, with_labels=True)
```



In [21]:
```
print((C*C.transpose()).todense())
```

```
[[8 6 7 6 3 4 3 3 3 2 2 2 2 2 1 2 1 1]
 [6 7 6 6 3 4 4 2 3 2 1 1 2 2 2 1 0 0]
 [7 6 8 6 4 4 4 3 4 3 2 2 3 3 2 2 1 1]
 [6 6 6 7 4 4 4 2 3 2 1 1 2 2 2 1 0 0]
 [3 3 4 4 4 2 2 0 2 1 0 0 1 1 1 0 0 0]
 [4 4 4 4 2 4 3 2 2 1 1 1 1 1 1 1 0 0]
 [3 4 4 4 2 3 4 2 3 2 1 1 2 2 2 1 0 0]
 [3 2 3 2 0 2 2 3 2 2 2 2 2 2 1 2 1 1]
 [3 3 4 3 2 2 3 2 4 3 2 2 3 2 2 2 1 1]
 [2 2 3 2 1 1 2 2 3 4 3 3 4 3 3 2 1 1]
 [2 1 2 1 0 1 1 2 2 3 4 4 4 3 3 2 1 1]
 [2 1 2 1 0 1 1 2 2 3 4 6 6 5 3 2 1 1]
 [2 2 3 2 1 1 2 2 3 4 4 6 7 6 4 2 1 1]
 [2 2 3 2 1 1 2 2 2 3 3 5 6 8 4 1 2 2]
 [1 2 2 2 1 1 2 1 2 3 3 3 4 4 5 1 1 1]
 [2 1 2 1 0 1 1 2 2 2 2 2 2 1 1 2 1 1]
 [1 0 1 0 0 0 0 1 1 1 1 1 1 2 1 1 2 2]
 [1 0 1 0 0 0 0 1 1 1 1 1 1 2 1 1 2 2]]
```

Exercises

1. Compute the adjacency matrix of the bipartite graph B at the top of this page and verify its block structure.
2. Compute the biadjacency matrix C of the graph B.
3. Compute the two products of C and its transpose, and, using the products as adjacency matrix, construct two graphs from them.
4. Compute the two projections of the bipartite graph B and compare them with the graphs constructed in the previous exercise.

In [ ]: