CS4423 - Networks

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

4. Small Worlds

Lecture 16: Small World Random Graphs

The fact that random networks tend to have low transitivity and clustering necessitates a new kind of (random) network model that is better at mimicking real world networks. One idea, developped by Watts and Strogatz in 1998, is to start with some regular network that naturally has a high clustering, and then to randomly distort its edges.

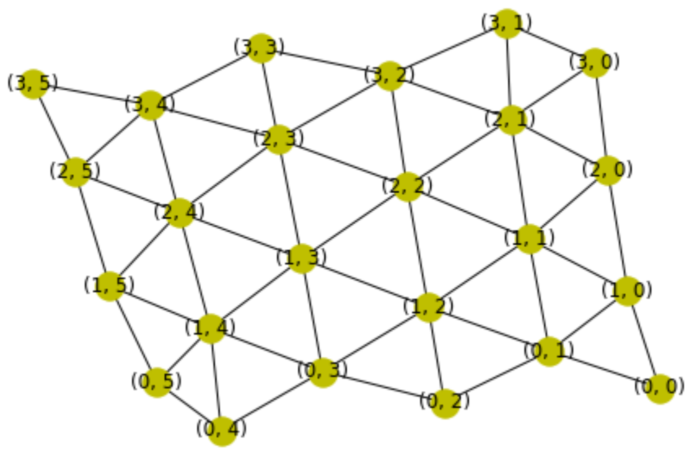
```
In [1]: import networkx as nx
        import matplotlib.pyplot as plt
```

Lattices, Line Graphs, Circle Graphs

A **triangular lattice** is (a finite porition of) a regular tiling of the Euclidean plane by triangles. Here, each (inner) vertex has 6 neighbors, which are linked in a cycle, giving a node clustering coefficient of $6/\binom{6}{2} = 2/5 = 0.4$.

A rectangular finite region of a triangular lattice with m strips of n triangles of constant height can be generated with the command `nx.triangular_lattice_graph(m, n)

```
In [2]: G = nx.triangular_lattice_graph(5, 6)
        nx.draw(G, with_labels=True, node_color='y')
```

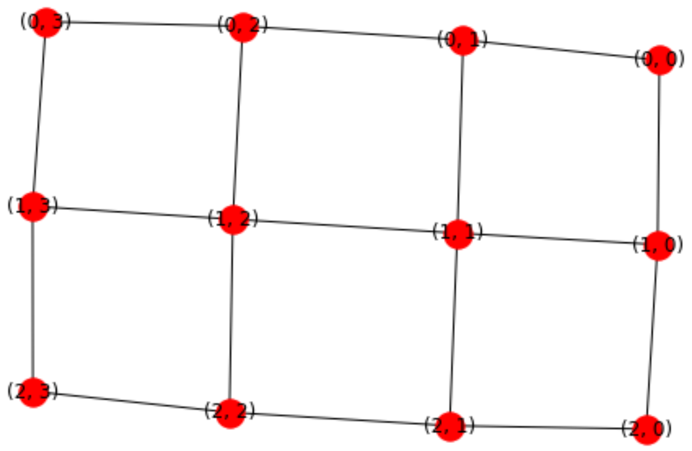


```
In [3]: nx.clustering(G)

Out[3]: {(0, 0): 1.0,
         (1, 0): 0.5,
         (2, 0): 0.5,
         (3, 0): 0.6666666666666666,
         (0, 1): 0.4,
         (1, 1): 0.4,
         (2, 1): 0.4,
         (3, 1): 0.6666666666666666,
         (0, 2): 0.6666666666666666,
         (1, 2): 0.4,
         (2, 2): 0.4,
         (3, 2): 0.4,
         (0, 3): 0.4,
         (1, 3): 0.4,
         (2, 3): 0.4,
         (3, 3): 0.6666666666666666,
         (0, 4): 0.6666666666666666,
         (1, 4): 0.4,
         (2, 4): 0.4,
         (3, 4): 0.4,
         (0, 5): 0.6666666666666666,
         (1, 5): 0.5,
         (2, 5): 0.5,
         (3, 5): 1.0}
```

```
In [4]: nx.average_clustering(G)

Out[4]: 0.5333333333333334
```

However, other kinds of lattice graphs contain no triangles at all!

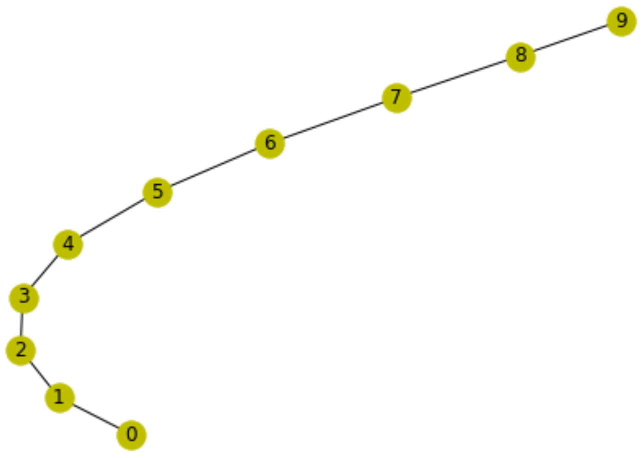
```
In [5]: G = nx.grid_2d_graph(3, 4)
        nx.draw(G, with_labels=True, node_colors='y')
```



In [6]:
```
nx.clustering(G)
```

Out[6]:
```
{(0, 0): 0,
 (0, 1): 0,
 (0, 2): 0,
 (0, 3): 0,
 (1, 0): 0,
 (1, 1): 0,
 (1, 2): 0,
 (1, 3): 0,
 (2, 0): 0,
 (2, 1): 0,
 (2, 2): 0,
 (2, 3): 0}
```

In [7]:
```
n = 10
G = nx.grid_graph([n])
nx.draw(G, with_labels=True, node_color='y')
```



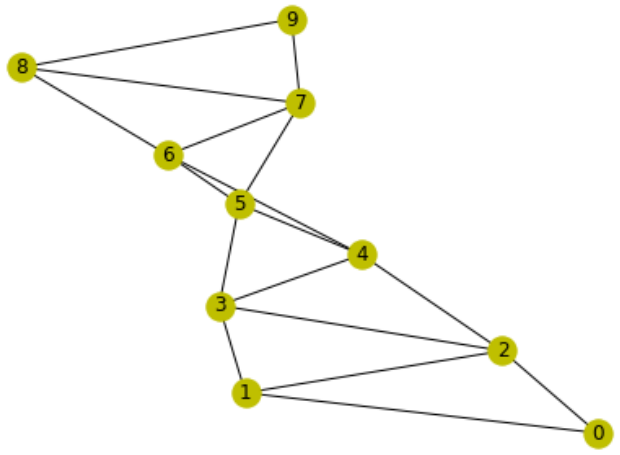
In [8]:
```
nx.clustering(G)
```

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

Idea: in a line graph with n vertices, additionally connect each node to all nodes not further than d steps away ...

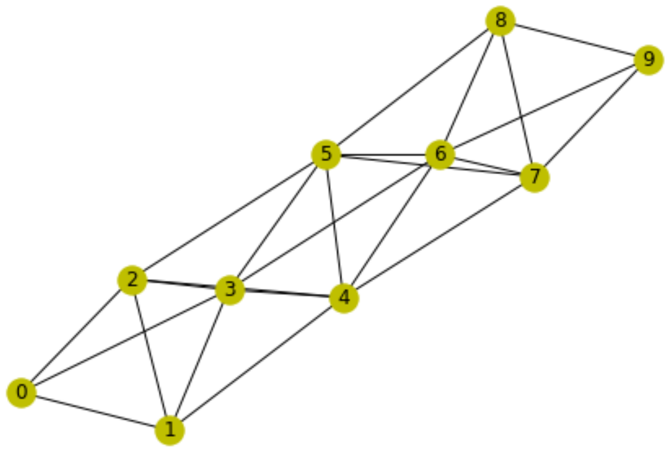
In [9]:
```
for v in range(n-2):
    G.add_edge(v, v+2)
nx.draw(G, with_labels=True, node_color='y')
print(nx.average_clustering(G))
```

```
0.6333333333333333
```



In [10]:
```
for v in range(n-3):
    G.add_edge(v, v+3)
nx.draw(G, with_labels=True, node_color='y')
print(nx.average_clustering(G))
```

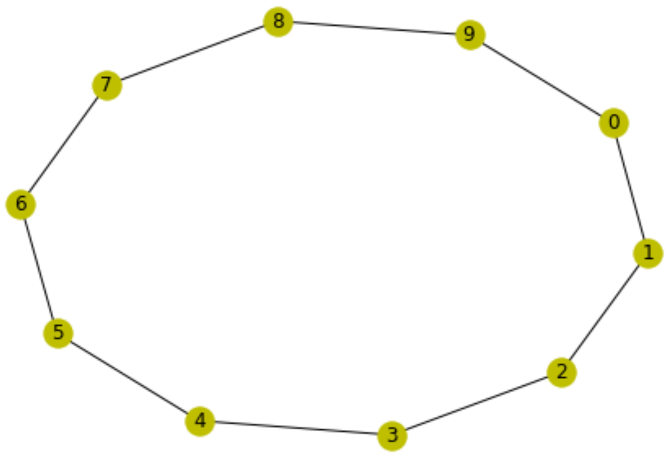
```
0.7466666666666666
```



Or, same thing on a cycle ...

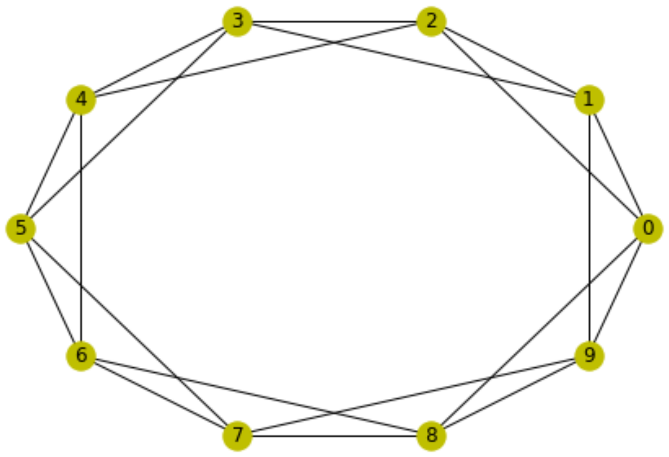
In [11]:
```
n = 10
G = nx.cycle_graph(n)
nx.draw(G, with_labels=True, node_color='y')
print(nx.average_clustering(G))
print(nx.average_shortest_path_length(G))
```

```
0.0
2.7777777777777777
```



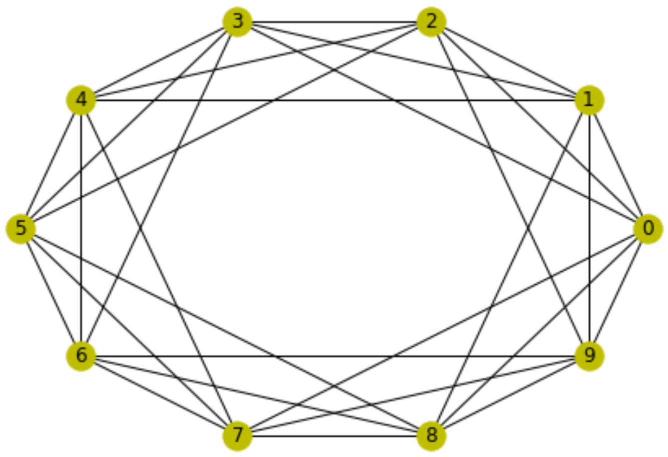
In [12]:
```
for v in range(n):
    G.add_edge(v, (v+2) % n)
nx.draw_circular(G, with_labels=True, node_color='y')
print(nx.average_clustering(G))
print(nx.average_shortest_path_length(G))
```

```
0.5
1.6666666666666667
```



In [13]:
```
for v in range(n):
    G.add_edge(v, (v+3) % n)
nx.draw_circular(G, with_labels=True, node_color='y')
print(nx.average_clustering(G))
print(nx.average_shortest_path_length(G))
```

```
0.5999999999999999
1.3333333333333333
```



Definition (Circle Graph). For $1 < d < n/2$, an (n, d)-**circle graph** is obtained from a cycle on n vertices by additionally linking each node to all nodes that are not more than d steps away on the cycle.

- An (n, d)-circle graph has n nodes and $m = nd$ edges.
- The graph clustering coefficient of an (n, d)-circle graph is
$$C = \frac{3d-3}{4d-2} \to \frac{3}{4}, \text{ as } d \to \infty.$$
In particular:

d	1	2	3	4	5
C	0	0.5	0.6	0.643	0.667

- The characteristic path length of an (n, d)-circle graph is approximately
$$L \approx \frac{n}{4d},$$
growing linearly with n (for fixed d).

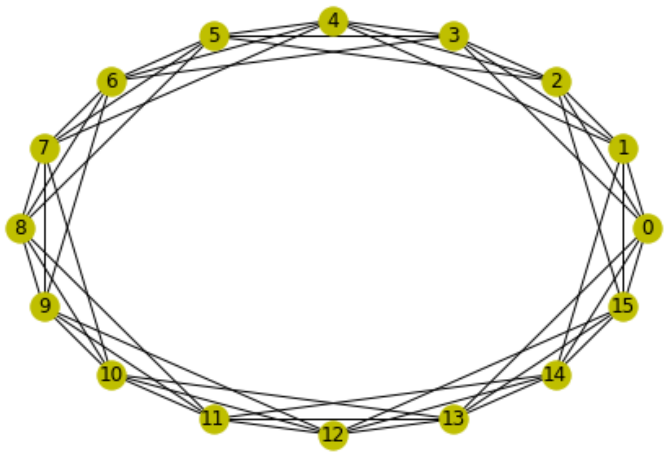
In conclusion, such regular graphs have high clustering but long shortest paths, hence (n, d)-circle graphs do not exhibit the small world behaviour.

Definition (The WS Model). Let $1 < d < n/2$ and $0 \leq p \leq 1$. An (n, d, p)-WS graph $G = (X, E)$ is constructed from an (n, d)-circle graph $G_0 = (X, E_0)$ by rewiring each of the edges in E_0 with probability p, as follows: 1. visit the nodes $X = \{1, \ldots, n\}$ in turn ('clockwise'). 2. for each node $i \in X$ consider the d edges connecting i to j in a clock wise sense ($j = i + 1, \ldots, i + d$). 3. With probability p, in the edge (i, j) replace j by node $k \in X$ chosen uniformly at random, subject to * $k \neq i$, and * (i, k) must not be an edge of G already.

A WS graph with parameters (n, d, p) can be generated with the command `nx.watts_strogatz_graph(n, 2*d, p)` .

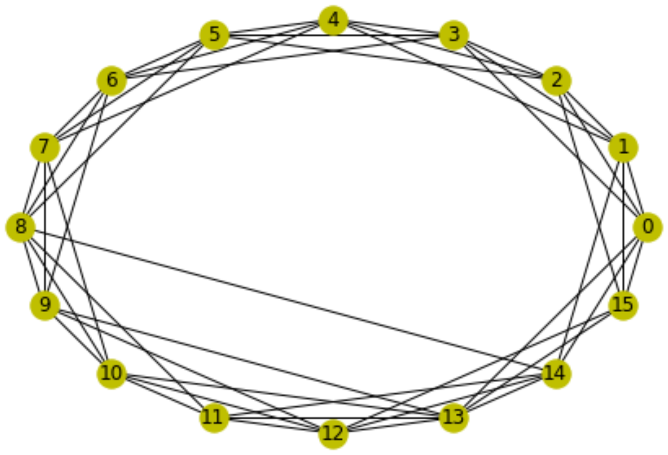
In [14]:
```
n, d = 16, 3
G = nx.watts_strogatz_graph(n, 2*d, 0)
nx.draw_circular(G, with_labels=True, node_color='y')
print(nx.average_clustering(G))
print(nx.average_shortest_path_length(G))
```

```
0.5999999999999999
1.8
```



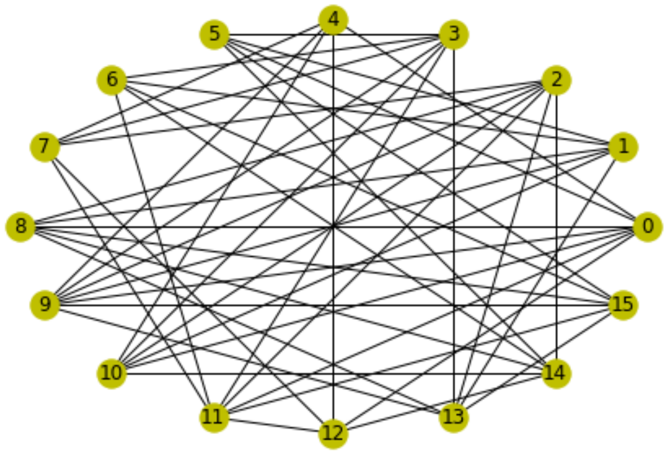
In [15]:
```
n, d = 16, 3
G = nx.watts_strogatz_graph(n, 2*d, 0.06)
nx.draw_circular(G, with_labels=True, node_color='y')
print(nx.average_clustering(G))
print(nx.average_shortest_path_length(G))
```

```
0.5523809523809523
1.7416666666666667
```



In [16]:
```
n, d = 16, 3
G = nx.watts_strogatz_graph(n, 2*d, 1)
nx.draw_circular(G, with_labels=True, node_color='y')
print(nx.average_clustering(G))
print(nx.average_shortest_path_length(G))
```

```
0.2392857142857143
1.6416666666666666
```



Exercises

1. Design an experiment with random graphs to verify the predicted graph clustering coefficient.

In [ ]:
```
```