Practice 6

March 4, 2020

1 Introduction

Mayra Cristina Berrones Reyes.

As seen in previous exercises, in graph theory we know the Hamiltonian path is a path that visits every node in a graph exactly one time. The hamiltonian cycle is a hamiltonian path that connects the first and last visited nodes. To determine if a graph has a hamiltonian path or cycle, we find a $\mathcal{NP}$–complete problem. There are several algorithms that can solve a hamiltonian path. One of them is known as brute force, in which the algorithms searches for all the possible sequences of the graph. This gives us a $n!$ complexity, and it is obvious that the complexity will rise with the number of nodes in a complete graph.

Then there is another algorithm called backtraking. This technique is based on recursively trying to build a solution one pice at the time. It removes all of the solutions which do not match the constraints of the problem. In the case of the hamiltonian path, that it has to visit every node exactly one time. This algorithm is at most $O(N^K)$. On this work, we are going to try and prove this complexity given some experiments made with the tool Networkx.

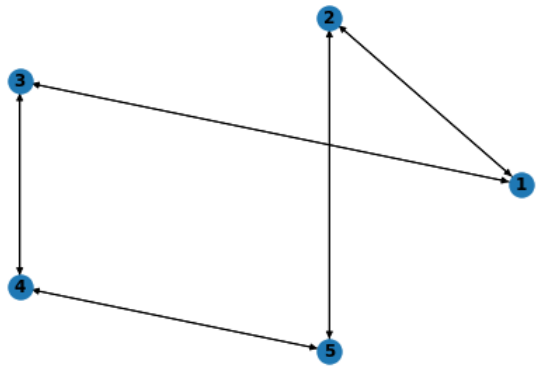
First, we play a little bit with the library, to see how the graphs work, and how we can plot them so we can have a more visually apealing example.

After reading the manual and some of the tutorials provided on their web page, we took some examples of already made algorithms, such as tournament.hamiltonian_path. We were not able to use this code, because it needed to be in a directed graph, and to do that we had to conect the edges manually. So, for the sake os a bigger experimentation, we used other things.

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

        #G = nx.gnm_random_graph(10, 12)
        G=nx.DiGraph()

        G.add_edge(1,2)
        G.add_edge(2,1)
```

```
G.add_edge(1,3)
G.add_edge(3,1)
G.add_edge(3,4)
G.add_edge(4,3)
G.add_edge(4,5)
G.add_edge(5,4)
G.add_edge(2,5)
G.add_edge(5,2)
#nx.draw_circular(G, with_labels=True, font_weight='bold')
#plt.subplot(122)

nx.draw_circular(G, nlist=[range(9, 10), range(5)], with_labels=True, font_weight='bold'
tournament.hamiltonian_path(G)
```

Out[1]: [2, 3, 1, 5, 4]



In order to be able to use the hamiltonian algorithm, we had to find a way to work with an undirected graph, so we searched and found this piece of code in which it takes a graph, and finds a path. Later we installed a library to mesure the time it takes to compute this process.

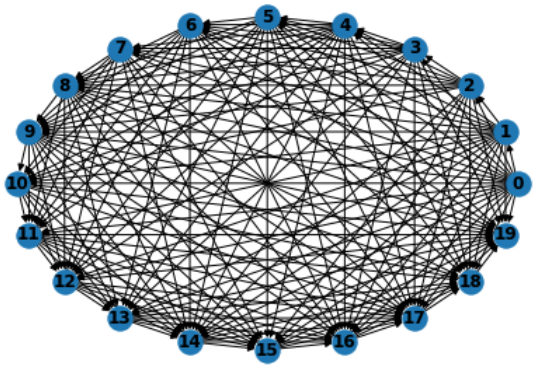
In [5]:
```
#code by https://gist.github.com/mikkelam/ab7966e7ab1c441f947b

import networkx as nx
import time
def hamilton(G):
    F = [(G,[list(G.nodes())[0]])]
    n = G.number_of_nodes()
    while F:
        graph,path = F.pop()
        confs = []
        for node in graph.neighbors(path[-1]):
            conf_p = path[:]
            conf_p.append(node)
```

```
                conf_g = nx.Graph(graph)
                conf_g.remove_node(path[-1])
                confs.append((conf_g,conf_p))
            for g,p in confs:
                if len(p)==n:
                    return "True"
                else:
                    F.append((g,p))
        return "False"
```

Next we have the different graphs. It is important to note that all the experiments and time reported are from experiments with a 100 nodes. The graphics shown bellow each one is a representation of 20 nodes only. This was made like this only to appreciate the number of edges that go away when we change the parameter k.

This parameter k helps us control the number of edges going out of each node. According to the complexity problem, we should have a similar behavior.

```
In [22]: from itertools import combinations
         from random import randint, random
         G2 = nx.DiGraph()
         node = 20
         G2.add_nodes_from(range(node))
         k = 0.9
         for i in range(node):
             for j in range(i, node):
                 if i != j and random() < k:
                     G2.add_edge(i, j)
         nx.draw_circular(G2, with_labels=True, font_weight='bold')
```

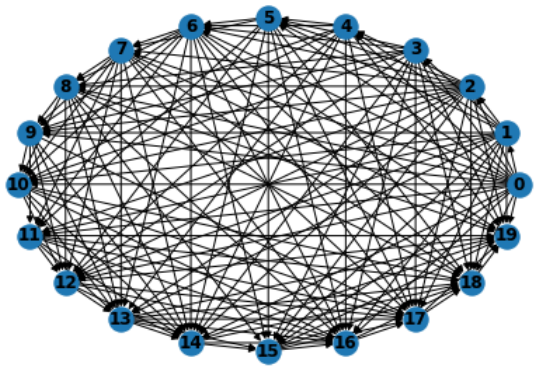



```
In [49]: import time
         start_time = time.time()
```

```
        #k = 0.9
        hamilton(G2)
        print( (time.time() - start_time))
```

```
34.48089385032654
```

```
In [23]: from itertools import combinations
         from random import randint, random
         G2 = nx.DiGraph()
         node = 20
         G2.add_nodes_from(range(node))
         k = 0.8
         for i in range(node):
             for j in range(i, node):
                 if i != j and random() < k:
                     G2.add_edge(i, j)
         nx.draw_circular(G2, with_labels=True, font_weight='bold')
```

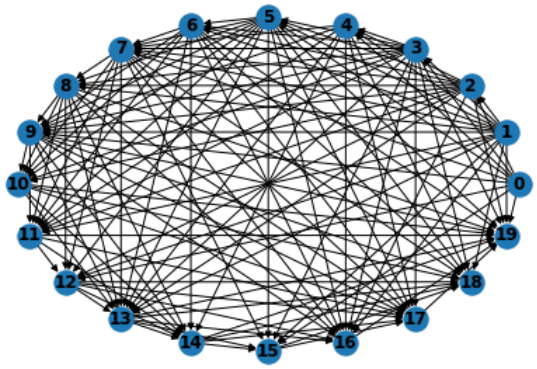



```
In [51]: start_time = time.time()
         #k = 0.8
         hamilton(G2)
         print( (time.time() - start_time))
```

```
28.018199920654297
```

```
In [24]: from itertools import combinations
         from random import randint, random
         G2 = nx.DiGraph()
         node = 20
         G2.add_nodes_from(range(node))
         k = 0.7
```

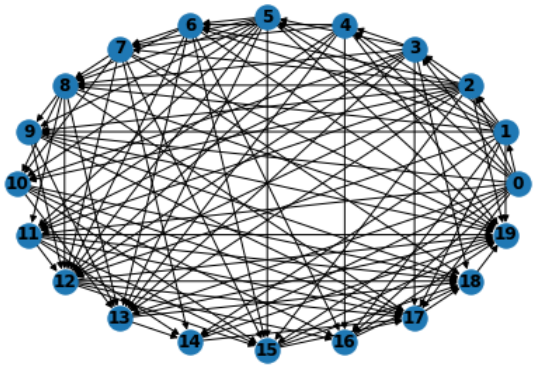
```
        for i in range(node):
            for j in range(i, node):
                if i != j and random() < k:
                    G2.add_edge(i, j)
        nx.draw_circular(G2, with_labels=True, font_weight='bold')
```



In [53]:
```
start_time = time.time()
#k = 0.7
hamilton(G2)
print( (time.time() - start_time))
```

```
19.465803146362305
```

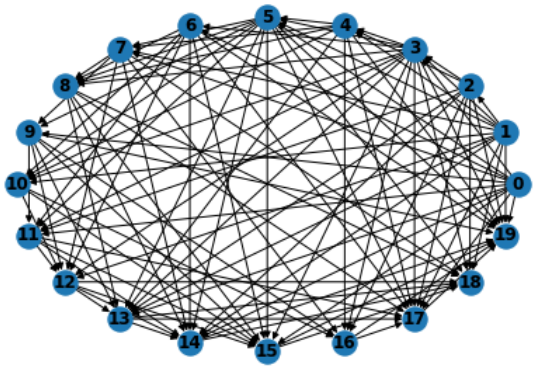
In [25]:
```
from itertools import combinations
from random import randint, random
G2 = nx.DiGraph()
node = 20
G2.add_nodes_from(range(node))
k = 0.6
for i in range(node):
    for j in range(i, node):
        if i != j and random() < k:
            G2.add_edge(i, j)
nx.draw_circular(G2, with_labels=True, font_weight='bold')
```



In [55]:
```
start_time = time.time()
#k = 0.6
hamilton(G2)
print( (time.time() - start_time))
```

```
14.404574871063232
```

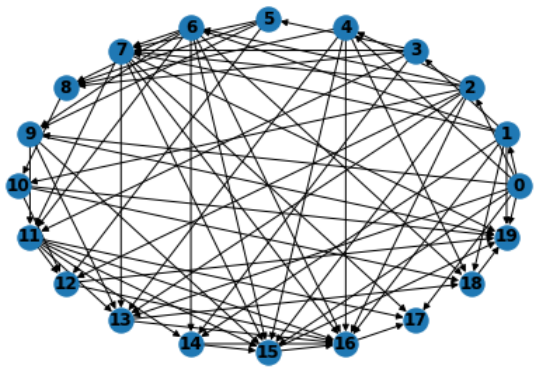
In [26]:
```
from itertools import combinations
from random import randint, random
G2 = nx.DiGraph()
node = 20
G2.add_nodes_from(range(node))
k = 0.5
for i in range(node):
    for j in range(i, node):
        if i != j and random() < k:
            G2.add_edge(i, j)
nx.draw_circular(G2, with_labels=True, font_weight='bold')
```



```
In [58]: start_time = time.time()
         #k = 0.5
         hamilton(G2)
         print( (time.time() - start_time))
```

```
10.570278882980347
```

```
In [27]: from itertools import combinations
         from random import randint, random
         G2 = nx.DiGraph()
         node = 20
         G2.add_nodes_from(range(node))
         k = 0.4
         for i in range(node):
             for j in range(i, node):
                 if i != j and random() < k:
                     G2.add_edge(i, j)
         nx.draw_circular(G2, with_labels=True, font_weight='bold')
```

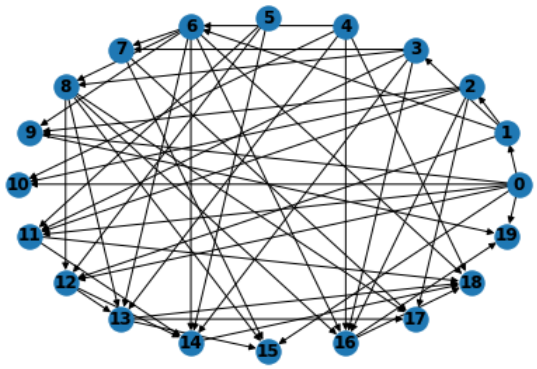



```
In [6]: start_time = time.time()
        #k = 0.4
        hamilton(G2)
        print(hamilton(G2), (time.time() - start_time))
```

```
True 12.908562898635864
```

```
In [29]: from itertools import combinations
         from random import randint, random
```

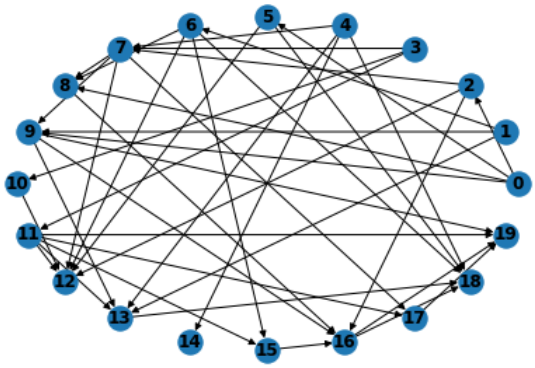
```
G2 = nx.DiGraph()
node = 20
G2.add_nodes_from(range(node))
k = 0.3
for i in range(node):
    for j in range(i, node):
        if i != j and random() < k:
            G2.add_edge(i, j)
nx.draw_circular(G2, with_labels=True, font_weight='bold')
```



In [63]:
```
start_time = time.time()
#k = 0.3
hamilton(G2)
print(hamilton(G2), (time.time() - start_time))
```

```
True 8.270284175872803
```

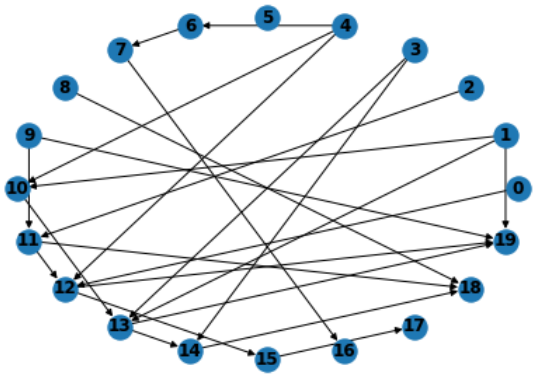
In [32]:
```
from itertools import combinations
from random import randint, random
G2 = nx.DiGraph()
node = 20
G2.add_nodes_from(range(node))
k = 0.2
for i in range(node):
    for j in range(i, node):
        if i != j and random() < k:
            G2.add_edge(i, j)
nx.draw_circular(G2, with_labels=True, font_weight='bold')
```



In [76]:
```
start_time = time.time()
#k = 0.2
hamilton(G2)
print(hamilton(G2))
print( (time.time() - start_time))
```

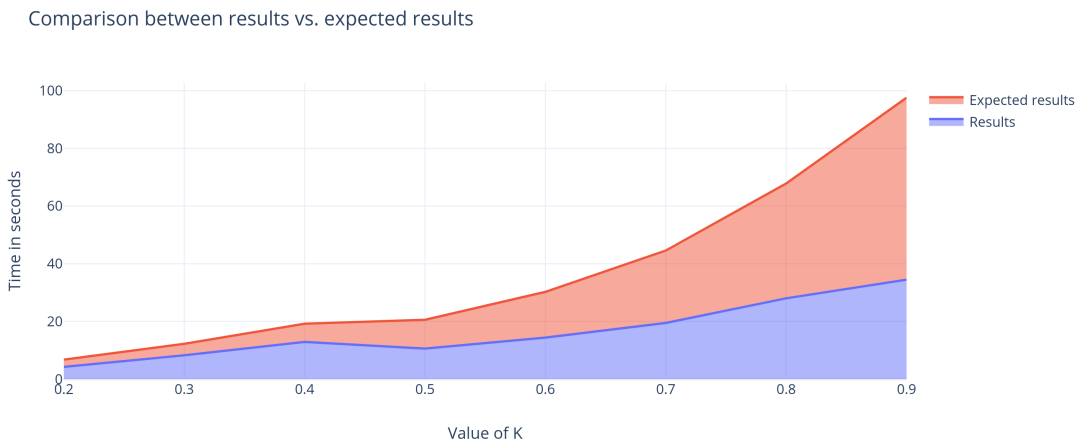
```
True
4.2513508796691895
```

In [39]:
```
from itertools import combinations
from random import randint, random
G2 = nx.DiGraph()
node = 20
G2.add_nodes_from(range(node))
k = 0.1
for i in range(node):
    for j in range(i, node):
        if i != j and random() < k:
            G2.add_edge(i, j)
nx.draw_circular(G2, with_labels=True, font_weight='bold')
```



```
In [40]: start_time = time.time()
         #k = 0.1
         hamilton(G2)
         print(hamilton(G2))
         print( (time.time() - start_time))

False
0.0604097843170166
```

2 Results and conclusions



As we can appreciate in the graphic above, the expected results and the results we had in this experimentation are not so different. We can conclude that, given the algorithm we choose to search for paths in our graph, it is reasonable that the more edges the graph has, the computational time will grow, this can be blamed on the way it searches for all possible paths that match our restrictions. If we shorten the list of edges, the time decreases. But as seen in the experimentation, too low of a value K, and there will not be enough edges to make a connection between all the nodes, making it impossible to form a hamiltonian path.