Task 5. Algorithms on graphs. Introduction to graphs and basic algorithms on graphs

Samarin Anton, C4113

January 23, 2020

In [1]:

```
import warnings
import networkx as nx
import sys
import numpy as np

np.set_printoptions(threshold=sys.maxsize)
warnings.filterwarnings('ignore')
```

In [2]:

```
# for more beauty image
%matplotlib notebook
```

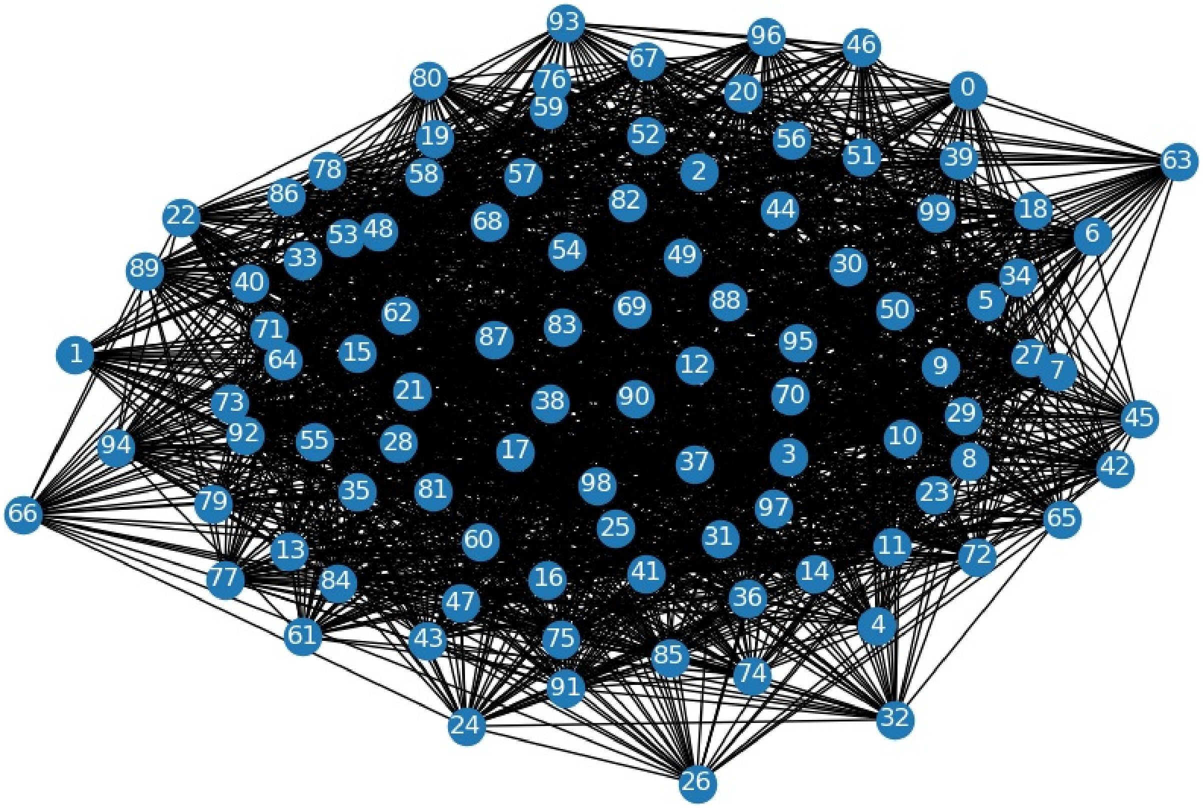
Generate data

In [3]:

```
nodes = 100
edges = 2000
```

In [4]:

```
G = nx.gnm_random_graph(nodes, edges, seed=42)
nx.draw(G, with_labels=True, font_color='w')
```



In [5]:

```
# show first 3 rows
matrix = nx.to_numpy_array(G)
matrix[0:3]
```

Out[5]:

```
array([[0., 0., 1., 0., 0., 1., 0., 1., 0., 1., 0., 0., 0., 1., 0., 0.,
        0., 1., 0., 1., 1., 0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 1.,
        0., 0., 1., 1., 0., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0.,
        0., 1., 1., 0., 0., 0., 1., 1., 0., 0., 1., 0., 0., 0., 0., 1.,
        0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 1., 0.,
        0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 1.,
        1., 1., 0., 1.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 1.,
        1., 0., 0., 1., 1., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 1.,
        0., 1., 0., 1., 0., 1., 1., 0., 0., 0., 0., 1., 0., 0., 1., 1.,
        0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 1., 1., 0., 1., 0.,
        0., 0., 0., 0., 0., 1., 1., 1., 0., 1., 0., 0., 0., 0., 1., 0.,
        0., 1., 0., 0., 0., 1., 0., 1., 1., 0., 1., 0., 1., 0., 0., 0.,
        0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 1., 0., 0., 1., 1., 1., 0., 0., 1., 1.,
        0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
        0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 1., 1.,
        1., 1., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0., 1., 0., 0., 1.,
        1., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 1.,
        1., 0., 1., 1.]])
```

In [6]:

```
# show first 3 nodes
adjacency_list = nx.to_dict_of_lists(G)
for node in range(3):
    print(f'{node}:', adjacency_list[node])
```

```
0: [35, 92, 76, 9, 49, 42, 55, 70, 96, 45, 20, 84, 34, 23, 7, 63, 78, 17, 31, 95, 28, 97, 2, 39
, 50, 99, 58, 27, 5, 19, 93, 54, 13]
1: [87, 69, 58, 85, 71, 43, 90, 13, 70, 92, 59, 35, 33, 16, 4, 23, 73, 38, 19, 52, 81, 37, 12,
62, 78, 46, 20, 31, 47, 88, 15, 57, 60, 28]
2: [14, 75, 89, 60, 79, 11, 37, 6, 22, 55, 51, 46, 10, 68, 49, 95, 63, 56, 92, 96, 98, 18, 0, 6
7, 64, 40, 43, 15, 9, 48, 28, 47, 99]
```

- Adjacency list use O(|E|) memory, allows to know number of neighbors O(1). It is used when |E| << $|V|^2$.
- Adjacency matrix use O($|V|^2$) memory, allows to check existing edge O(1). It is used when graph is dense.

Search of the number connected components

In [7]:

```
nx.number_connected_components(G)
```

Out[7]:

```
1
```

Search of the shortest path

In [8]:

```
nx.shortest_path(G, source=0, target=1)
```

Out[8]:

```
[0, 58, 1]
```

Conclusion

Time of search of the number connected components is O(|E| + |V|) for a simple graph, where DFS is base.

Time of Search of the shortest path is O(|E| + |V|) for a simple graph, where BFS is base.