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

Egor Turukhanov

Group C4134

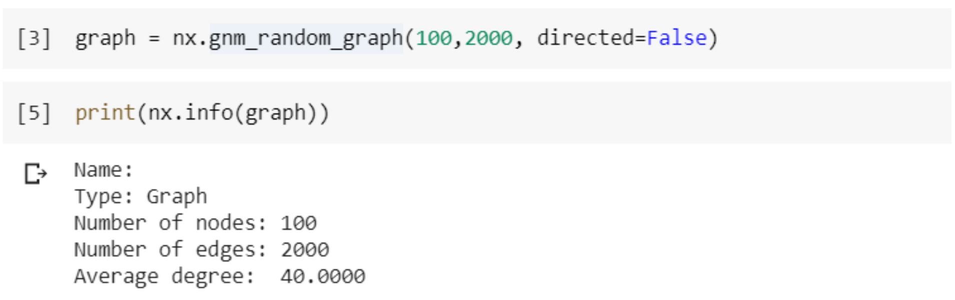
20.11.19

1.

Graph Generation

Firstly, we generated simple indirect graph with 100 vertices and 2000 edges.

You can see code below:

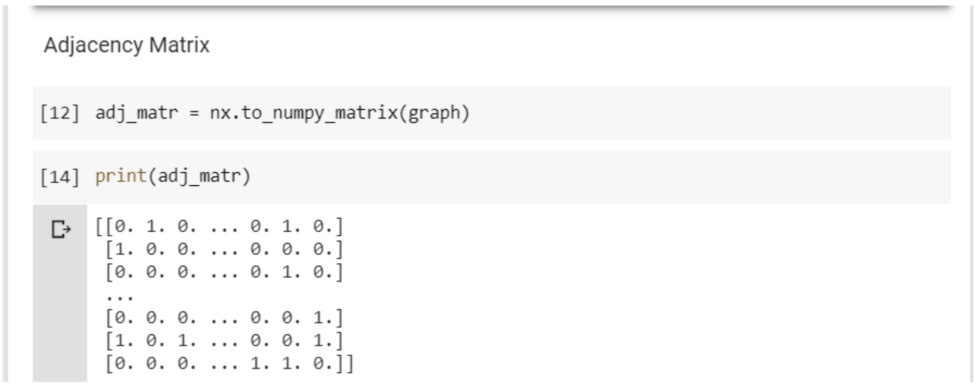


```
[3]  graph = nx.gnm_random_graph(100,2000, directed=False)
```

```
[5]  print(nx.info(graph))
```

```
Name: 
Type: Graph
Number of nodes: 100
Number of edges: 2000
Average degree:  40.0000
```

After that we generated an adjacency matrix for our graph.



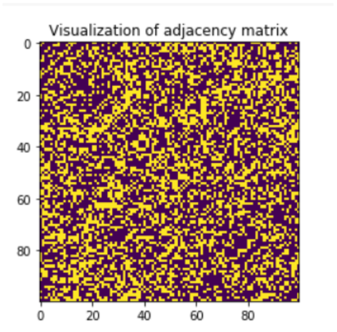
Adjacency Matrix

```
[12] adj_matr = nx.to_numpy_matrix(graph)
```

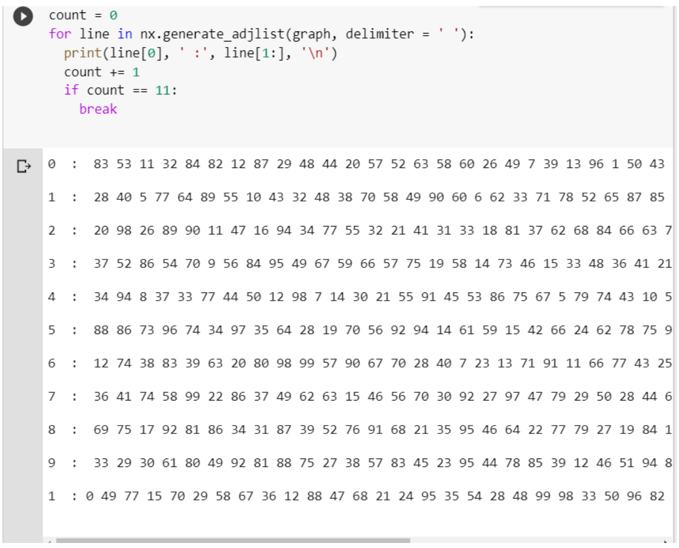
```
[14] print(adj_matr)
```

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

Below you can see plot of generated adjacency matrix.



Then we generated adjacency list from our adjacency matrix.

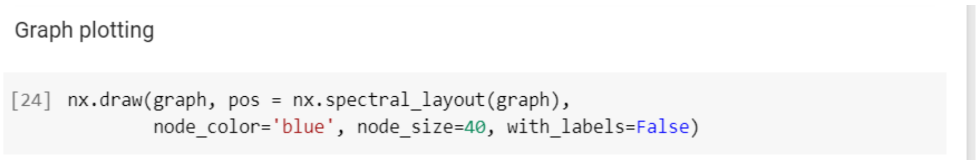


```
count = 0
for line in nx.generate_adjlist(graph, delimiter = ' '):
  print(line[0], ' :', line[1:], '\n')
  count += 1
  if count == 11:
    break
```

```
0  :  83 53 11 32 84 82 12 87 29 48 44 20 57 52 63 58 60 26 49 7 39 13 96 1 50 43

1  :  28 40 5 77 64 89 55 10 43 32 48 38 70 58 49 90 60 6 62 33 71 78 52 65 87 85

2  :  20 98 26 89 90 11 47 16 94 34 77 55 32 21 41 31 33 18 81 37 62 68 84 66 63 7

3  :  37 52 86 54 70 9 56 84 95 49 67 59 66 57 75 19 58 14 73 46 15 33 48 36 41 21

4  :  34 94 8 37 33 77 44 50 12 98 7 14 30 21 55 91 45 53 86 75 67 5 79 74 43 10 5

5  :  88 86 73 96 74 34 97 35 64 28 19 70 56 92 94 14 61 59 15 42 66 24 62 78 75 9

6  :  12 74 38 83 39 63 20 80 98 99 57 90 67 70 28 40 7 23 13 71 91 11 66 77 43 25

7  :  36 41 74 58 99 22 86 37 49 62 63 15 46 56 70 30 92 27 97 47 79 29 50 28 44 6

8  :  69 75 17 92 81 86 34 31 87 39 52 76 91 68 21 35 95 46 64 22 77 79 27 19 84 1

9  :  33 29 30 61 80 49 92 81 88 75 27 38 57 83 45 23 95 44 78 85 39 12 46 51 94 8

1  : 0 49 77 15 70 29 58 67 36 12 88 47 68 21 24 95 35 54 28 48 99 98 33 50 96 82
```

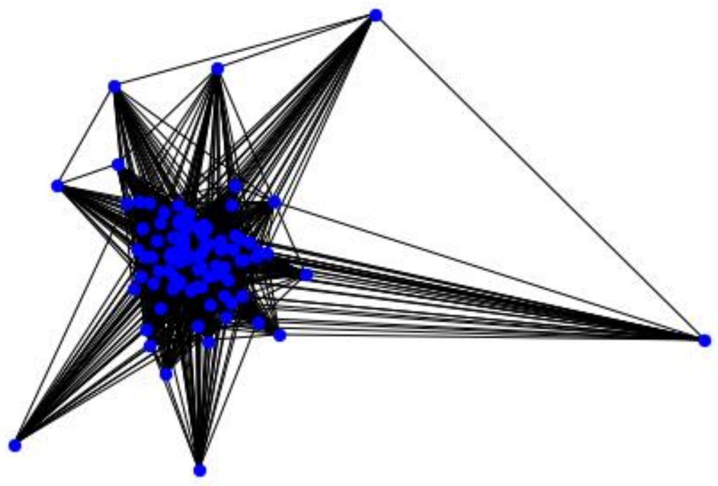
Graph plotting.

Below you can see visualization of our simple undirected graph with 100 vertices(nodes) and 2000 edges.



Graph plotting

```
[24] nx.draw(graph, pos = nx.spectral_layout(graph),
             node_color='blue', node_size=40, with_labels=False)
```



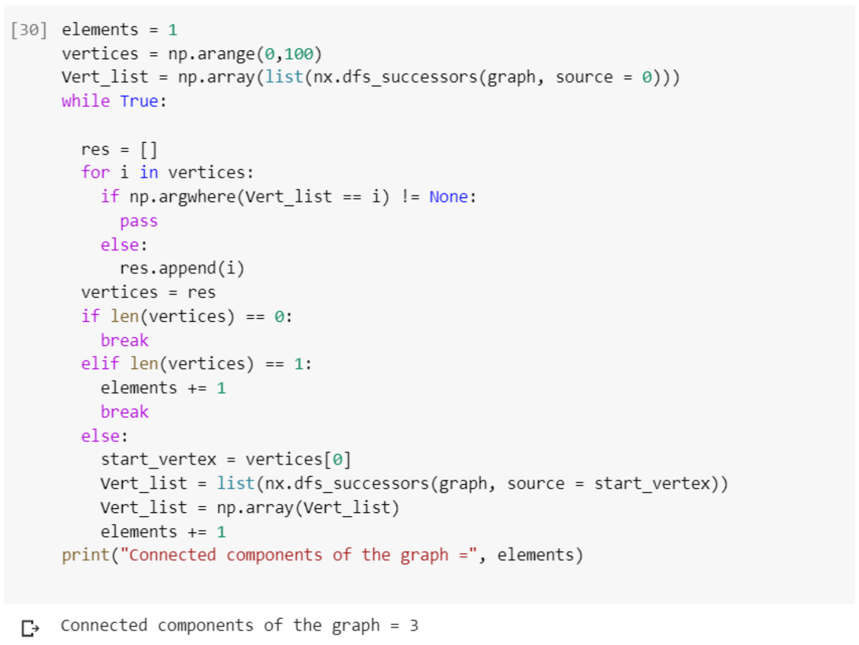
As can be seen above, we have produced adjacency matrix and adjacency list for our graph. In our case graph representation in adjacency matrix is better than adjacency list, because we have 100 vertices and only 2000 edges for them. If number of vertices would be small enough compared to number of nodes, then most element of matrix will be equal 0.

Adjacency list and matrix allow us access graph element quicker and more efficient.

2.Depth-first search.

We used Depth-first search algorithms to find connected elements of the graph.

Code of it and result below:

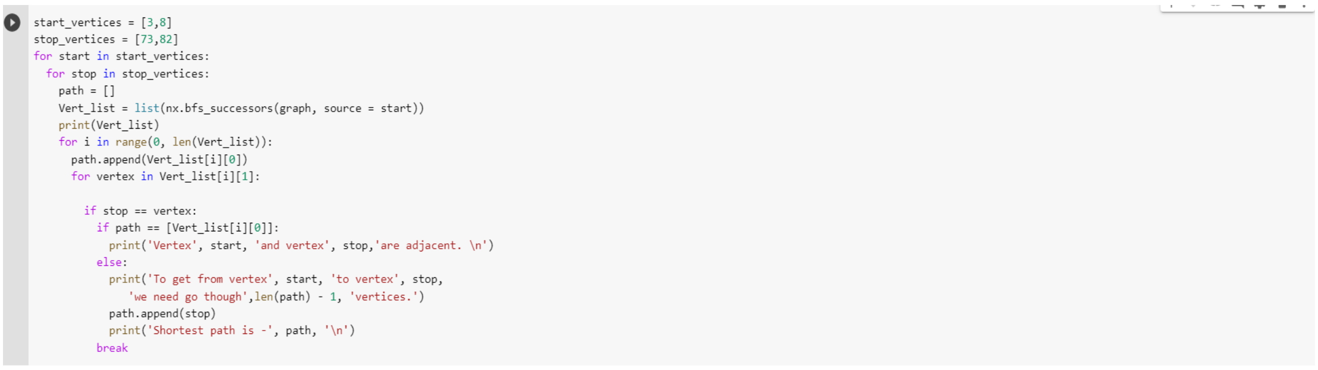


```
[30] elements = 1
     vertices = np.arange(0,100)
     Vert_list = np.array(list(nx.dfs_successors(graph, source = 0)))
     while True:

       res = []
       for i in vertices:
         if np.argwhere(Vert_list == i) != None:
           pass
         else:
           res.append(i)
       vertices = res
       if len(vertices) == 0:
         break
       elif len(vertices) == 1:
         elements += 1
         break
       else:
         start_vertex = vertices[0]
         Vert_list = list(nx.dfs_successors(graph, source = start_vertex))
         Vert_list = np.array(Vert_list)
         elements += 1
     print("Connected components of the graph =", elements)
```

```
Connected components of the graph = 3
```

We found 3 connected components in our graph.

Next, we used Breadth-First Search algorithm to find shortest path between 2 random vertices.



```python
start_vertices = [3,8]
stop_vertices = [73,82]
for start in start_vertices:
  for stop in stop_vertices:
    path = []
    Vert_list = list(nx.bfs_successors(graph, source = start))
    print(Vert_list)
    for i in range(0, len(Vert_list)):
      path.append(Vert_list[i][0])
      for vertex in Vert_list[i][1]:

        if stop == vertex:
          if path == [Vert_list[i][0]]:
            print('Vertex', start, 'and vertex', stop,'are adjacent. \n')
          else:
            print('To get from vertex', start, 'to vertex', stop,
                'we need go though',len(path) - 1, 'vertices.')
            path.append(stop)
            print('Shortest path is -', path, '\n')
          break
```

```
[(3, [37, 52, 86, 54, 70, 9, 56, 84, 95, 49, 67, 59, 66, 57, 75, 19, 58, 14, 73, 46, 15, 33, 48,
36, 41, 21, 94, 2, 55, 47, 24, 77, 5, 51, 80, 97, 63, 31, 10]), (37, [64, 96, 93, 43, 72, 4, 76,
7, 79, 44, 38, 16, 13, 32, 98, 40, 39, 62, 99, 92, 20, 27, 26, 89, 53, 0]), (52, [8, 83, 12, 68,
69, 81, 1, 11, 28, 78, 65, 82, 45, 90]), (86, [74, 60, 91, 25, 17, 34, 87, 85, 71]), (54, [30,
22]), (70, [23, 18, 61, 50, 6, 88, 35]), (9, [29]), (84, [42])]
Vertex 3 and vertex 73 are adjacent.

[(3, [37, 52, 86, 54, 70, 9, 56, 84, 95, 49, 67, 59, 66, 57, 75, 19, 58, 14, 73, 46, 15, 33, 48,
36, 41, 21, 94, 2, 55, 47, 24, 77, 5, 51, 80, 97, 63, 31, 10]), (37, [64, 96, 93, 43, 72, 4, 76,
7, 79, 44, 38, 16, 13, 32, 98, 40, 39, 62, 99, 92, 20, 27, 26, 89, 53, 0]), (52, [8, 83, 12, 68,
69, 81, 1, 11, 28, 78, 65, 82, 45, 90]), (86, [74, 60, 91, 25, 17, 34, 87, 85, 71]), (54, [30,
22]), (70, [23, 18, 61, 50, 6, 88, 35]), (9, [29]), (84, [42])]
To get from vertex 3 to vertex 82 we need go through 2 vertices.
Shortest path is - [3, 37, 52, 82]

[(8, [69, 75, 17, 4, 92, 81, 86, 34, 31, 87, 39, 52, 76, 91, 68, 21, 35, 95, 46, 64, 22, 77, 79,
27, 19, 84, 11, 73, 43, 12, 48, 72, 16, 36, 33, 62, 56, 41, 50, 53, 9, 97, 74, 5, 10, 67]), (69,
[55, 32, 83, 66, 70, 24, 96, 18, 20, 63, 58, 85, 45, 65, 1, 40, 99, 98, 47, 88, 59, 61, 44]),
(75, [37, 93, 38, 13, 25, 3, 54, 42, 89, 2, 60, 90, 51, 7, 14]), (17, [26, 71, 82, 30, 57, 28,
78, 29]), (4, [94]), (92, [23]), (81, [49, 0]), (86, [15]), (39, [6]), (76, [80])]
Vertex 8 and vertex 73 are adjacent.

[(8, [69, 75, 17, 4, 92, 81, 86, 34, 31, 87, 39, 52, 76, 91, 68, 21, 35, 95, 46, 64, 22, 77, 79,
27, 19, 84, 11, 73, 43, 12, 48, 72, 16, 36, 33, 62, 56, 41, 50, 53, 9, 97, 74, 5, 10, 67]), (69,
[55, 32, 83, 66, 70, 24, 96, 18, 20, 63, 58, 85, 45, 65, 1, 40, 99, 98, 47, 88, 59, 61, 44]),
(75, [37, 93, 38, 13, 25, 3, 54, 42, 89, 2, 60, 90, 51, 7, 14]), (17, [26, 71, 82, 30, 57, 28,
78, 29]), (4, [94]), (92, [23]), (81, [49, 0]), (86, [15]), (39, [6]), (76, [80])]
To get from vertex 8 to vertex 82 we need go through 3 vertices.
Shortest path is [8, 69, 75, 17, 82].
```

Conclusion

We have implemented Depth-First Search and Breadth-First Search algorithms for our graph. They were quite efficient for our simple undirected graph.