FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report
on the practical task No. 5
«Algorithms on graphs. Introduction to graphs and basic algorithms on graphs»

Performed by:
Putnikov Semyon
J4132c

Accepted by:
Dr Petr Chunaev

St. Petersburg
2020

1. Goal

The use of different representations of graphs and basic algorithms on graphs (Depth-first search and Breadth-first search).

2. Formulation of the problem

I Generate a random adjacency matrix for a simple undirected unweighted graph with 100 vertices and 200 edges (note that the matrix should be symmetric and contain only 0s and 1s as elements). Transfer the matrix into an adjacency list. Visualize the graph and print several rows of the adjacency matrix and the adjacency list. Which purposes is each representation more convenient for?

II Use Depth-first search to find connected components of the graph and Breadth-first search to find a shortest path between two random vertices. Analyse the results obtained.

III Describe the data structures and design techniques used within the algorithms.

3. Brief theoretical part

3.1 Depth-first search (DFS)

Depth-first search (DFS) is an algorithm for traversing or searching a graph. The algorithm starts at a chosen root vertex and explores as far as possible along each branch before backtracking. Applied for: searching connected components, searching loops in a graph, testing bipartiteness, topological sorting, etc.

The time complexity of DFS is $O(|V| + |E|)$.

A connected component of a graph is a subgraph in which any two vertices are connected by paths, and which is not connected to any vertex in the rest graph.

3.2 Breadth-first search (BFS)

Breadth-first search (BFS) is an algorithm for traversing or searching a graph. The algorithm starts at a chosen root vertex and explores all of the neighbour vertices at the present depth prior to moving on to the vertices at the next depth level. It uses an opposite strategy to DFS, which instead explores the vertex branch as far as possible before being forced to backtrack and expand other vertices.

Applied for: searching shortest path.

The time complexity of BFS is $O(|V| + |E|)$.

4. Results

In theory for sparse graph adjacency matrix fit into more memory than adjacency list, but also exist sparse data structures which erase this drawback. The main drawback of this adjancy list is that there is no quick way to check existence of edge (u, v).

As we can see on graphs visualization which have 100 nodes and 200 edges visualization of graphs not worthless only for very small graphs.

Using DFS we can obtain number of components in graph and connected problems. Using BFS is one of ways which we can get shortest path between 2 nodes.

I used dictionaries and lists for programming visualization of above algorithms. Also I used recursive technique.

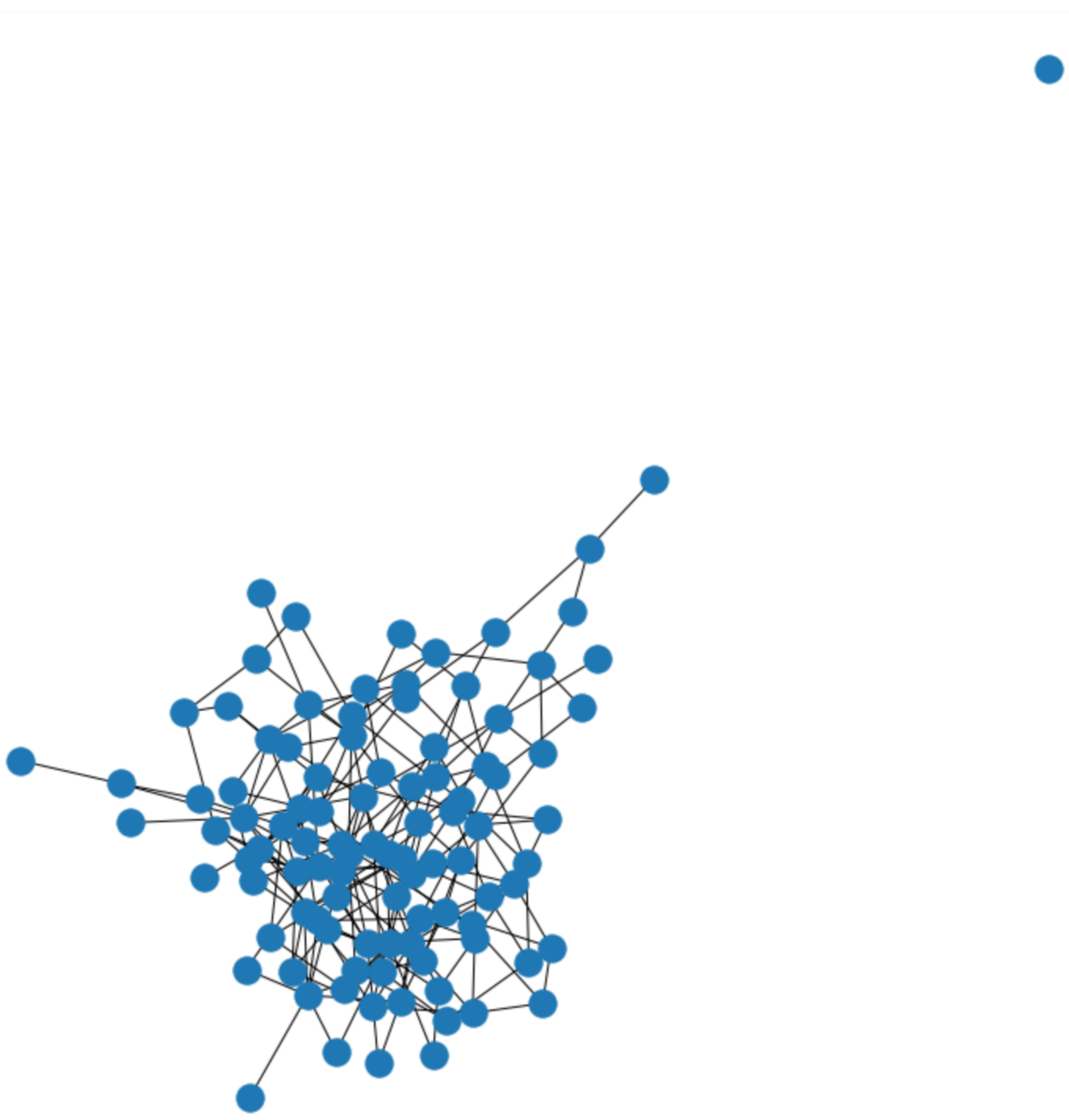
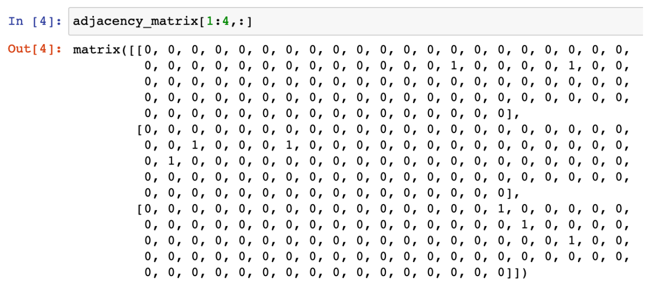
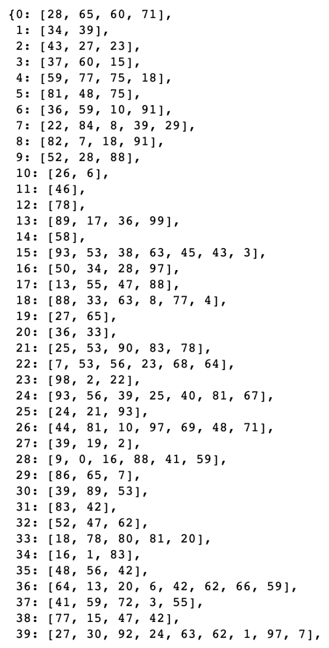


Figure 1: Graph visualization.



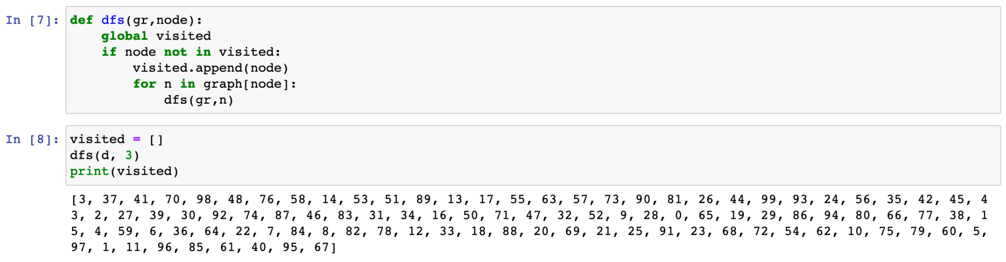
```
In [4]: adjacency_matrix[1:4,:]

Out[4]: matrix([[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, 1, 0, 0, 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, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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,
                 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, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 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, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

Figure 2: Adjacency matrix.



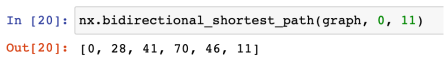
```
{0: [28, 65, 60, 71],
 1: [34, 39],
 2: [43, 27, 23],
 3: [37, 60, 15],
 4: [59, 77, 75, 18],
 5: [81, 48, 75],
 6: [36, 59, 10, 91],
 7: [22, 84, 8, 39, 29],
 8: [82, 7, 18, 91],
 9: [52, 28, 88],
 10: [26, 6],
 11: [46],
 12: [78],
 13: [89, 17, 36, 99],
 14: [58],
 15: [93, 53, 38, 63, 45, 43, 3],
 16: [50, 34, 28, 97],
 17: [13, 55, 47, 88],
 18: [88, 33, 63, 8, 77, 4],
 19: [27, 65],
 20: [36, 33],
 21: [25, 53, 90, 83, 78],
 22: [7, 53, 56, 23, 68, 64],
 23: [98, 2, 22],
 24: [93, 56, 39, 25, 40, 81, 67],
 25: [24, 21, 93],
 26: [44, 81, 10, 97, 69, 48, 71],
 27: [39, 19, 2],
 28: [9, 0, 16, 88, 41, 59],
 29: [86, 65, 7],
 30: [39, 89, 53],
 31: [83, 42],
 32: [52, 47, 62],
 33: [18, 78, 80, 81, 20],
 34: [16, 1, 83],
 35: [48, 56, 42],
 36: [64, 13, 20, 6, 42, 62, 66, 59],
 37: [41, 59, 72, 3, 55],
 38: [77, 15, 47, 42],
 39: [27, 30, 92, 24, 63, 62, 1, 97, 7],
```

Figure 3: Adjacency list.



```
In [7]: def dfs(gr,node):
            global visited
            if node not in visited:
                visited.append(node)
                for n in graph[node]:
                    dfs(gr,n)

In [8]: visited = []
        dfs(d, 3)
        print(visited)

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

Figure 4: Results of dfs calculations.



```
In [20]: nx.bidirectional_shortest_path(graph, 0, 11)
Out[20]: [0, 28, 41, 70, 46, 11]
```

Figure 5: Results of bfs calculations.