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 Pavel Belenko, J4132C

Accepted by Dr Petr Chunaev

St. Petersburg
2021

Goal:

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

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. Analyze the results obtained.

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

Brief theoretical part:

Graph theory is actively used in modeling various complex systems, including social, transport, communication systems etc. Graph theory tools allow for a comprehensive analysis of data represented as graphs. As part of the current laboratory work, it is proposed to study in practice various graph representations, as well as standard graph traversal algorithms.

Let's discuss some definitions that may be needed to solve the tasks. An undirected graph is called a pair $G = (V, E)$, where $V = \{v_i\}$ is a set of vertices (or nodes), and $E = \{e_{ij}\} = \{v_i,\ v_j\}$ is a set of pairs of vertices called edges (or connections). The number of vertices is denoted by $|V|$, and the number of edges is denoted by $|E|$.

A directed graph is a graph in which the edges have directions (orientations). A weighted graph is a graph in which weights are assigned to each edge. A simple graph is a graph in which only one edge between a pair of vertices. A multigraph is a generalization of a simple graph, in which several edges between a pair of vertices are possible in the graph. A complete graph is a graph in which all vertices are connected by a rib. A path (chain) in a graph is a sequence of pairwise distinct edges connecting twoë different vertices. The length of the path (chain) is the number of

edges (or the sum of the weights of the edges) in the path (chain). Vertexes v_1 and v_2 in a graph are called connected if there is a path from v_1 to v_2. Otherwise, these vertices are called disconnected. Connected graph is a graph in which any pair of vertices is connected. Otherwise, the graph is called disconnected. The connectivity component of a graph is the maximal connected subgraph of a graph. In this task we will consider simple undirected unweighted graph.

Let's move on to different types of graph representation. The first of them is an adjacency matrix, i.e. a matrix whose rows and columns are indexed by vertices and whose cells contain a Boolean value (0 or 1) indicating whether the corresponding vertices are connected (for weighted graphs, the corresponding weights are instead of 1). The adjacency matrix (as a 2D array) requires $O(|V|^2)$ memory.

Another type of representation is an adjacency list, i.e. a collection of vertex lists, where a list of adjacent vertices is given for each vertex of the graph. An adjacency list (as in a 1D array of lists) requires $O(|V| + |E|)$ memory.

For a sparse graph, i.e. a graph in which most pairs of vertices are not connected by edges, $|E| \ll |V|^2$, the adjacency list is significantly more efficient for storage than the adjacency matrix.

Let's move on to the classical graph traversal algorithms considered in the framework of this work:

1) Deep-first Search (DFS) is a graph traversal algorithm that starts traversing from the selected root vertex and goes "deep" into the graph, as far as possible, before traversing from the new vertex. DFS has time complexity of $O(|V| + |E|)$.

2) Breadth-first Search (BFS) is a graph traversal algorithm in which the traversal starts from the selected root vertex and all adjacent vertices are examined vertices at the current "depth" before moving to vertices at the next "depth". This traversal algorithm uses a strategy that is in some sense the opposite of DFS. BFS time complexity has also time complexity of $O(|V| + |E|)$.

Both algorithms can be used to find the shortest path between vertices and to search for graph connectivity components.

Results:

All calculations were performed in the Jupiter lab development environment. NumPy, random and networkx modules were used in the process.

```
[[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.]]
```

Figure 1 – fragment of the generated adjacency matrix

```
0: [1, 20, 62]
1: [0, 48, 49, 75]
2: [47, 58, 79]
3: [11, 28, 61, 82, 91]
4: [10, 37, 56, 85, 86]
5: [55, 95, 96]
6: [32, 36, 84]
7: [61, 76, 99]
8: [21, 24, 40, 43, 54, 59, 70, 71]
9: [13, 15, 51, 83, 99]
```

Figure 2 – fragment of the adjacency list generated from the adjacency matrix

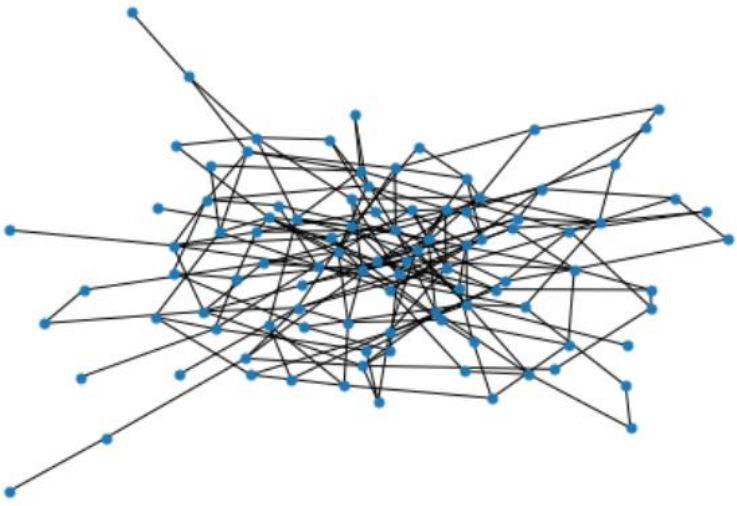

Figure 3 – visualization of the graph

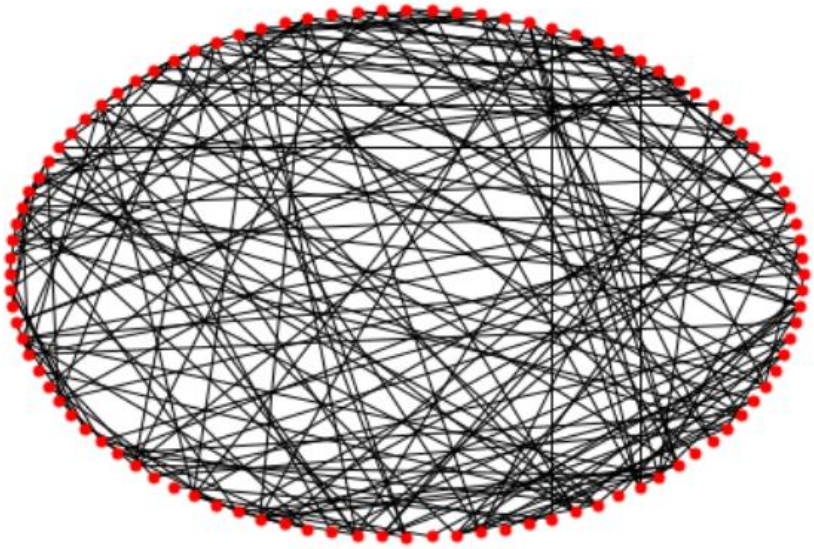

Figure 4 – cyclic visualization of the graph

```
Selected nodes: 73, 79

Fragment of list connected components of the graph finded using DFS:

[[73 62]
 [62  0]
 [ 0  1]
 [ 1 48]
 [48 23]
 [23 12]
 [12 50]
 [50 21]
 [21  8]
 [ 8 24]
 [ 8 40]
 [40 25]
 [25 26]
 [26 86]
 [86  4]]

Path beetween randomly selected nodes finded using BFS:

[73 62 47  2 79]
```

Figure 5 – result of using BFS and DFS, as given in the task

Conclusions:

I. In which cases is it more convenient to use an adjacency matrix / an adjacency list?

An adjacency matrix is preferable when we expect a graph to be dense (that is, when there are many connections between vertices of the graph, and its adjacency list is long and uninformative).

Using an adjacency matrix, you can quickly understand whether there is a connection between two specific vertices in the graph, and you can also quickly create or delete existing connections. The disadvantage of using an adjacency matrix (compared to an adjacency list) is the need to use more space, especially when it comes to dense graphs.

An adjacency list is preferable when we expect a graph to be resolved (that is, when there are few connections between vertices of the graph and its adjacency matrix is unreadable due to its sparsity).

II. Which is better to use for graph traversal: depth-first search (DFS) or breadth-first search (BFS)?

This largely depends on the graph structure: if it is known that the graph is deep and/or the solution is close to the initial search node, BFS may be better. If the graph is very wide, BFS will need a lot of memory to traverse it. So, in this case, DFS is better suited.

Appendix:

Code in GitHub repository: github.com/belpablo/Algorithms-21/tree/main/Lab_5