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
Mikhail Grigoryev (370852)
Semenova Valeria (370061)
Academic group J4133c
Accepted by
Dr Petr Chunaev

St. Petersburg
2022

Goal

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

Formulation of the problem

Task 1. Generate a random adjacency matrix for a simple undirected unweighted graph of 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?

Task 2. 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.

Brief theoretical part

Path search algorithms on graphs are used to find shortest paths between two vertices. In case of unweighted graphs, those paths consist of the least number of edges.

Algorithms used in this practical work (implemented from NetworkX library):

1. Depth-first search (here for finding connected components) was implemented from scratch recursively. DFS utilizes a neighbor stack (unlinke BFS which uses a queue). It is a simple algorithm that looks at all neighbors of the current vertex, puts them on the stack, then takes the first neighbor in the stack and that vertex becomes current. This results in the vertices in one branch being traversed first.
2. Breadth-first search (here for finding the shortest path) was taken from NetworkX. The method *nx.shortest_path* was used, if the graph is unweighted, BFS is used by default. It is a traversal algorithm similar to DFS. However, BFS uses a neighbor queue, thus the vertices closest to the stariting one are being traversed first.

Results

Task 1. A random adjacency matrix of a simple undirected graph $G(|V| = 100, |E| = 200)$ was generated. The matrix is symmetrical (undirected graph) and contains only 1s (edge) and 0s (no edge). The first 3 rows of the adjacency matrix were printed.

First row: [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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 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. 1. 0.]

Second row: [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. 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.]

Third row: [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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 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.]

The adjacency matrix was transferred to an adjacency list which is either a list (or a dictionary) of lists of neighbors. The first 3 rows are presented below:

First row: [17, 51, 63, 71, 98]

Second row: [38, 78]

Third row: [41, 48, 62, 70]

Additionally, the graph was visualized.

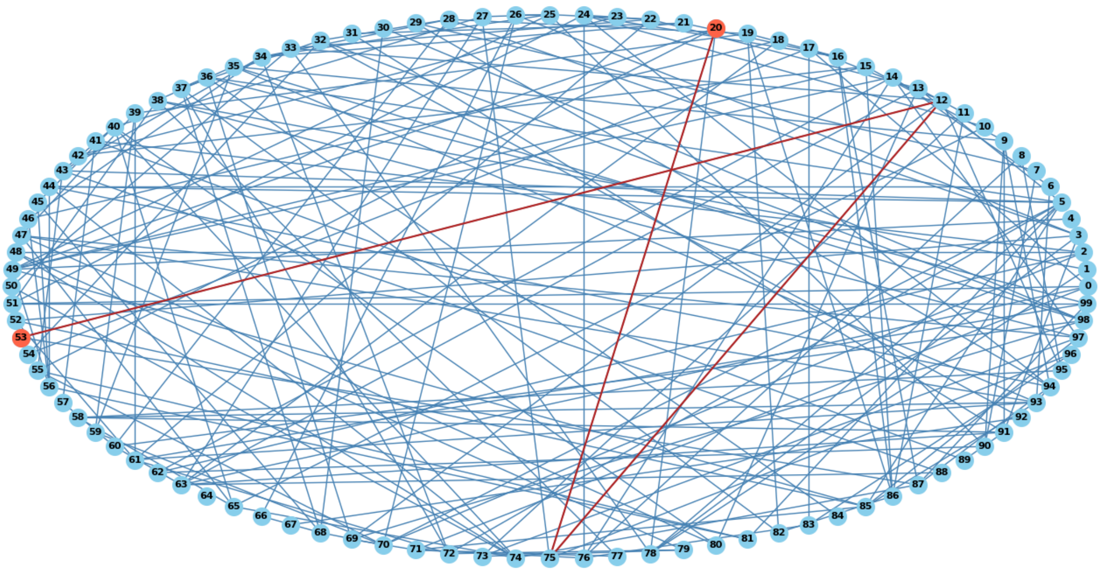

Figure 1: The random undirected graph with 100 vertices and 200 edges visualized (with shortest path between two random vertices).

The printed rows show that adjacency lists are significantly more space-efficient than adjacency matrices. Additionally, they are more legible for humans as we can read the rows of the adjacency list as: *the 0th vertex is adjacent to vertices 17, 51, 63, 71 and 98*. Both of the structures are better representations than plain visuals that get complicated even for small graphs.

Task 2. In the second task, DFS was used to find connected components of the generated graph. For the graph visualized previously, the connected components were: vertex number 64 and all the other vertices (they happened to be connected). The code in the listing (see Appendix) prints both connected components and all connected vertices (in pairs).

BFS was used to compute the shortest path between a pair of random vertices (from 53 to 20). The path for the mentioned vertices was:

$$53 \to 12 \to 75 \to 20$$

The path contains 3 edges and 4 vertices. This path was visualized previously. The algorithm provides a path for any pair of vertices, if they are connected.

Conclusions

Different representations of graphs were compared. Graph traversal algorithms for unweighted graphs such as Depth- and Breadth-first search were applied for searching the shortest path between random vertices in random graphs and for yielding connected components of the graph.

Appendix

GitHub link: https://github.com/Dormant512/itmo_lab_listings/blob/main/lab5.py.