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

*Vdovkina Sophia*

*Syrchenko Arina*

*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**

**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.

**Brief theoretical part**

A path in a graph is a finite sequence of vertices in which each vertex (except the last) is connected to the next in the sequence of vertices by an edge. If any two vertices in a graph are connected by a path, then such a graph is called connected. We can consider such a subset of the vertices of the graph such that every two vertices of this subset are connected by a path, and no other vertex is connected to any vertex of this subset. Each such subset, together with all the edges of the original graph connecting the vertices of this subset, is called a connected component. Graphs in which all edges are links, that is, the order of the two ends of an edge in the graph is not essential, are called undirected. A graph without arcs, loops and multiple edges is called simple. Unweighted graph is such a graph where all the edges have the same weight (or no weight at all).

Graphs can be represented as adjacency matrix – square matrix, where number of rows and columns equals to number of vertices. Each value in an adjacency matrix is equal to the weight of the edge that connects respective vertices.

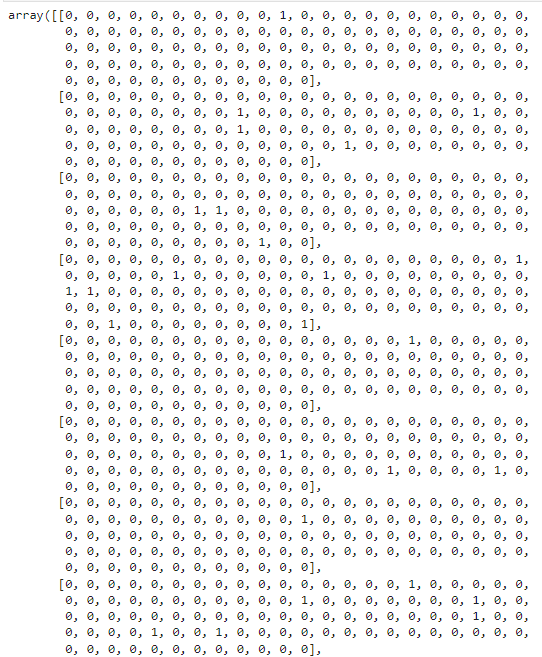
Another common representation is *adjacency list*, usually represented as a dictionary. Each sub-list with index *u* corresponds to a vertex *u* and contains a list of edges *(u, v)* that originate from *u*. For simple graphs such sub-list can contain only indices of vertices *v*, adjacent to vertex *u*.

Depth first Search (DFS) or Depth first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure. Traversal means visiting all the nodes of a graph. The purpose of the algorithm is to mark each vertex as visited while avoiding cycles. A standard DFS implementation puts each vertex of the graph into one of two categories: visited or not visited.

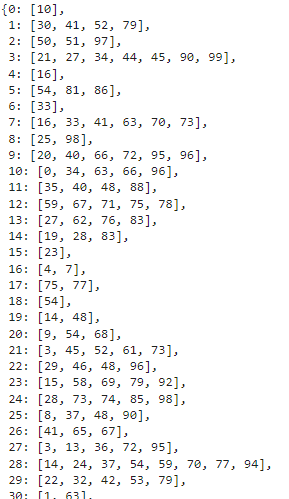
Breadth first search (BFS) is a recursive algorithm for searching all the vertices of a graph or tree data structure, in the same way as DFS each vertex can be visited or not visited. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored.

**Results**

1. The simple random graph with 100 nodes and 200 edges was generated. Part of the adjacency matrix is depicted in the picture 1 and a slice of adjacency list is in the picture 2.



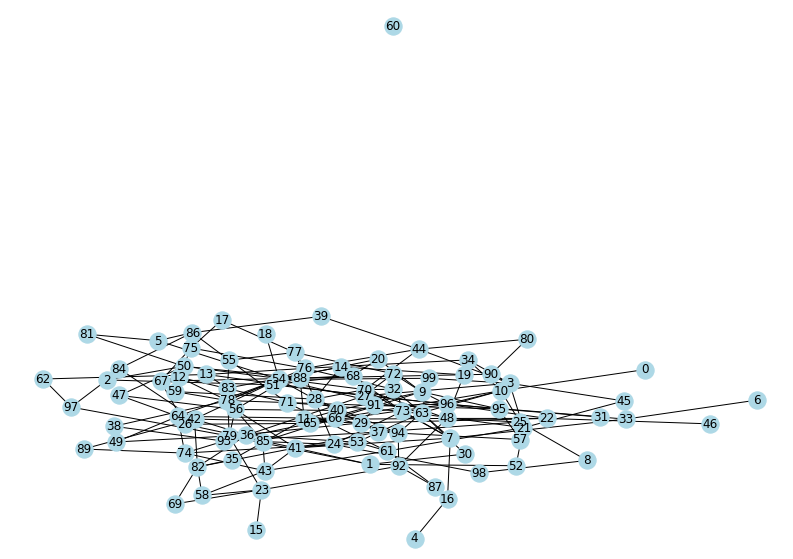
Picture 1 – the adjacency matrix



Picture 2 – the adjacency list

The adjacency matrix of this graph looks less informative because the graph is sparse, so that is why the matrix has lots of zeros. But if it was a matrix of a small weighted directed graph, the matrix representation would be much simpler and more informative than the list representation.

The graph is shown in picture 3.



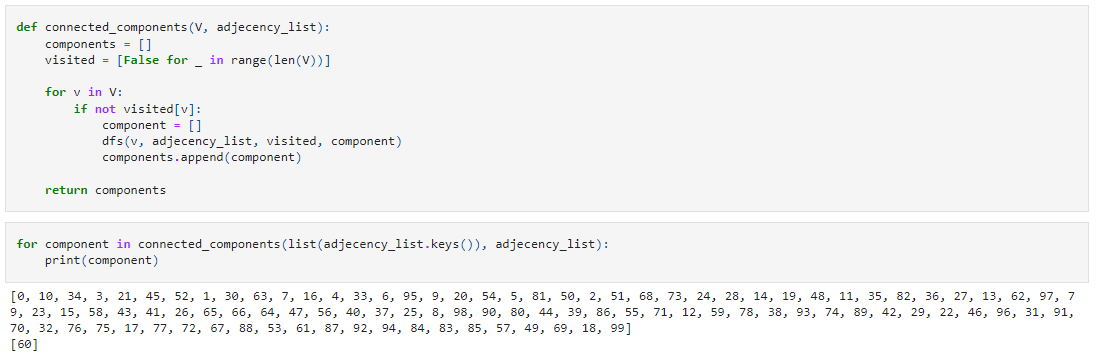
Picture 3 – the graph

The graph has 2 disconnected components: the big one contains 99 vertices and there is one separate vertex.

**II-III.** Depth-first search was used to find connected components of the graph. The algorithm uses recurs design.

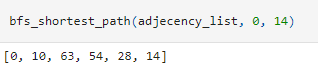
The algorithm has a loop (iteration over all elements in a graph, the algorithm finds an unseen vertex which will be a root for the search of all vertices connected to it).

The result is shown on picture 4.



Picture 4 – finding connected elements

Breadth-first search strategy was used to find shortest paths between all the components. The algorithm was implemented with non-recursive design and uses queue data structure. The search of all possible shortest paths between all vertex pairs was conducted independently – none of the searches used any information from the previous search. The result is shown on picture 5.



Picture 5 – the shortest path

**Conclusion**

Random simple graph with 100 nodes and 200 edges was generated. It has 1 isolated node. The depth-first search had to make O(|V| + 2|E|) operations to visit all the vertices and check all the bidirectional bonds between them.

The breadth-first search can also make up to O(|V| + 2|E|) operations to find the shortest path between the nodes since it also visits all the nodes and checks every bidirectional bond.

**Appendix**

Source code is available on <https://github.com/sophia-vdovkina/Analysis-and-development-of-algorithms/blob/main/Task%205/Task5.ipynb>