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:
Roman Bezaev
J4133c

Accepted by:
Dr Petr Chunaev

St. Petersburg
2021

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

1. 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?
2. Use Depth-first search to find connected components of the graph and Breadthfirst search to find a shortest path between two random vertices. Analyse the results obtained.
3. Describe the data structures and design techniques used within the algorithms.

3 Brief theoretical part

Graph is a structure amounting to a set of objects in which some pairs of the objects are in some sense "related". The objects correspond to mathematical abstractions called vertices (also called nodes or points) and each of the related pairs of vertices is called an edge (also called link or line). In simple graph two vertices can be connected directly only with one edge. Unweighted graph is such a graph where all the edges have the same weight (or no weight at all).

Graphs can be represented as **adjancecy matrix** – square matrix, where number of rows and columns equals to number of vertices $|V|$. Each value in a adjancecy matrix is equal to the weight of the edge that connects respective vertices. An adjacency matrix takes up $O(|V|^2)$ storage.

Another common representation is **adjacency list**, the list of lists. 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. An adjacency list takes up $O(|V| + |E|)$ space. Adjacency lists are quicker for the task of giving the set of adjacent vertices to a given vertex than adjacency matrices – $O(|neighbors|)$ for the former vs $O(|V|)$ for the latter.

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph

data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

It uses the opposite strategy of **depth-first search**, which instead explores the node branch as far as possible before being forced to backtrack and expand other nodes. Both search strategies make $O(|V| + |E|)$ steps to walk through complete graph.

4 Results

Adj matrix 2nd 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, 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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Adj matrix 5th 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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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]
Adj matrix 10th 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, 0, 0, 1, 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]
Adj list: 0: [67, 98], 1: [56, 66, 68, 99], 2: [67, 82], 3: [12, 23, 35, 87], 4: [6, 8, 21, 72, 73], 5: [35, 54, 60], 6: [4, 25, 37, 40, 42, 56, 73, 80, 85], 7: [8, 62, 83, 88], 8: [4, 7, 63, 87, 92, 98], 9: [46, 51], 10: [43, 61], ...

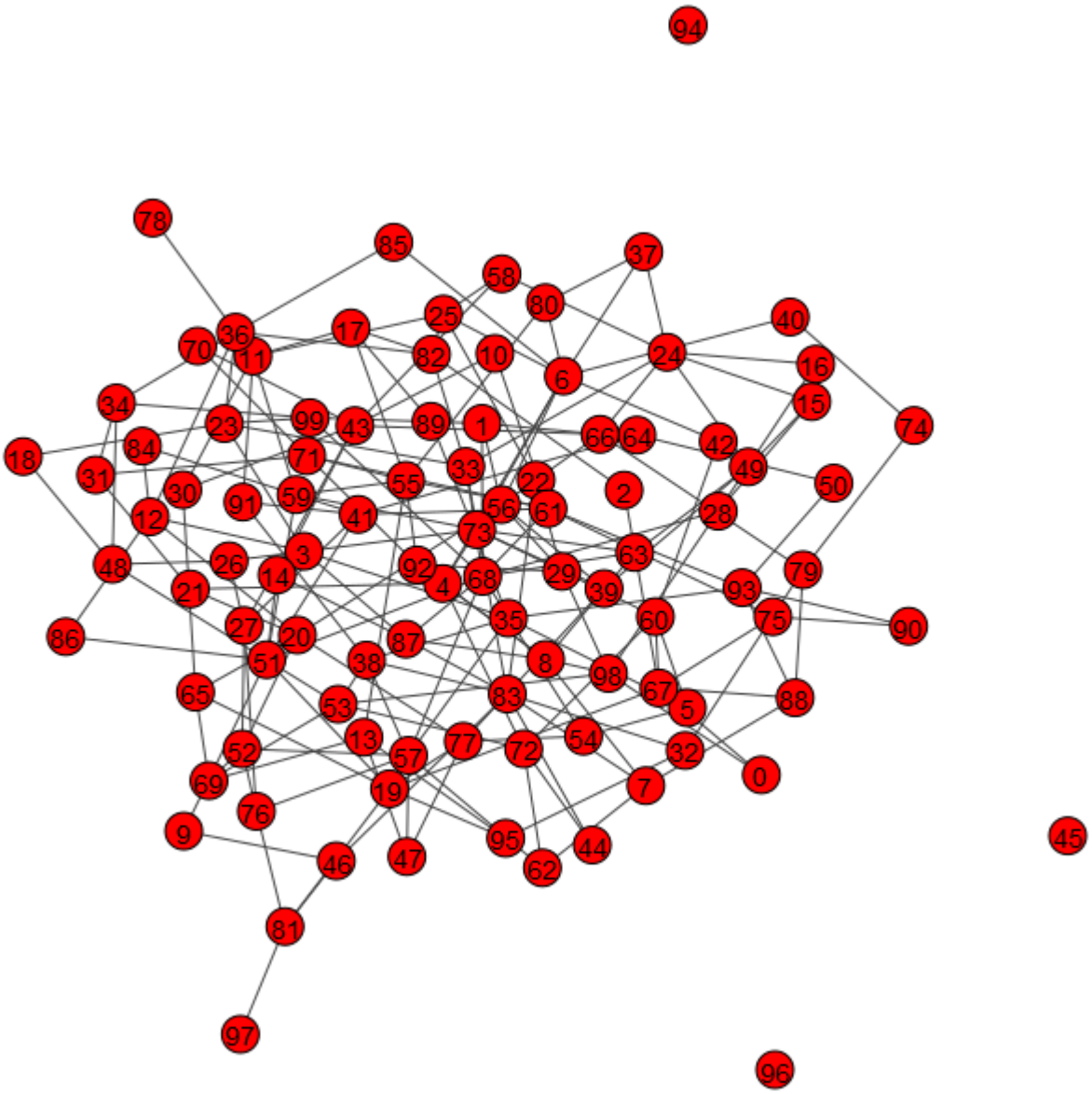

Figure 1: Graph visualisation

Adjacency matrix has a lot of zeros so it's much less informative than adjacency list when there is a big amount of vertices. But if we are working with small, weighted, directed graph, the matrix representation would be much simpler and more informative than list representation.

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

Depth-first search strategy was used to find connected components of the graph. The algorithm was implemented with recursive design and makes use of a stack – the data structure which is able to make operations "add element to the beginning" and "get the last element" in constant time. The algorithm successfully found all 4 connected components.

Algorithm randomly chose vertices with id's 54 and 95. Shortest path from 46 to 56 according to BFS: [46, 83, 22, 29, 56]

Breadth-first search strategy was used to find shortest paths between all the components. The algorithm was implemented in non-recursive design and makes use of deque data structure which is able to make operations "get first element, get last element" and "add first element, add last element" for constant time

5 Conclusions

Random simple graph with 100 nodes and 200 edges was generated. The analysis shown that there happened to be 4 connected components, three of them being isolated nodes. 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.

6 Appendix

https://github.com/trixdade/ITMO_algorithms/blob/master/Lab5/main.py