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
Alexander Yamoldin
j4134c
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. Analyse the results obtained.
III. Describe the data structures and design techniques used within the algorithms.

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 usually is equal 1 (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* $\Theta(|V|^2)$ *storage. But because our matrix is symmetric we could decrease consumption of storage to* $\Theta(0.5 * |V|^2)$ *because we could create half of a matrix by the other half.*

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,adjancent to vertex u. An adjacency list takes up* $\Theta(|V| + |E|)$ *space.*

Adjacency lists *are quicker for the task of giving the set of adjacent vertices to a given vertex than* ***adjancency 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 arbitrarynode 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.

Results

Generating a graph

Generate a symmetric matrix which look likes follow:

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

Check that number of links are 200. In adjacency matrix of not oriented graph number of “1” is 2 * links

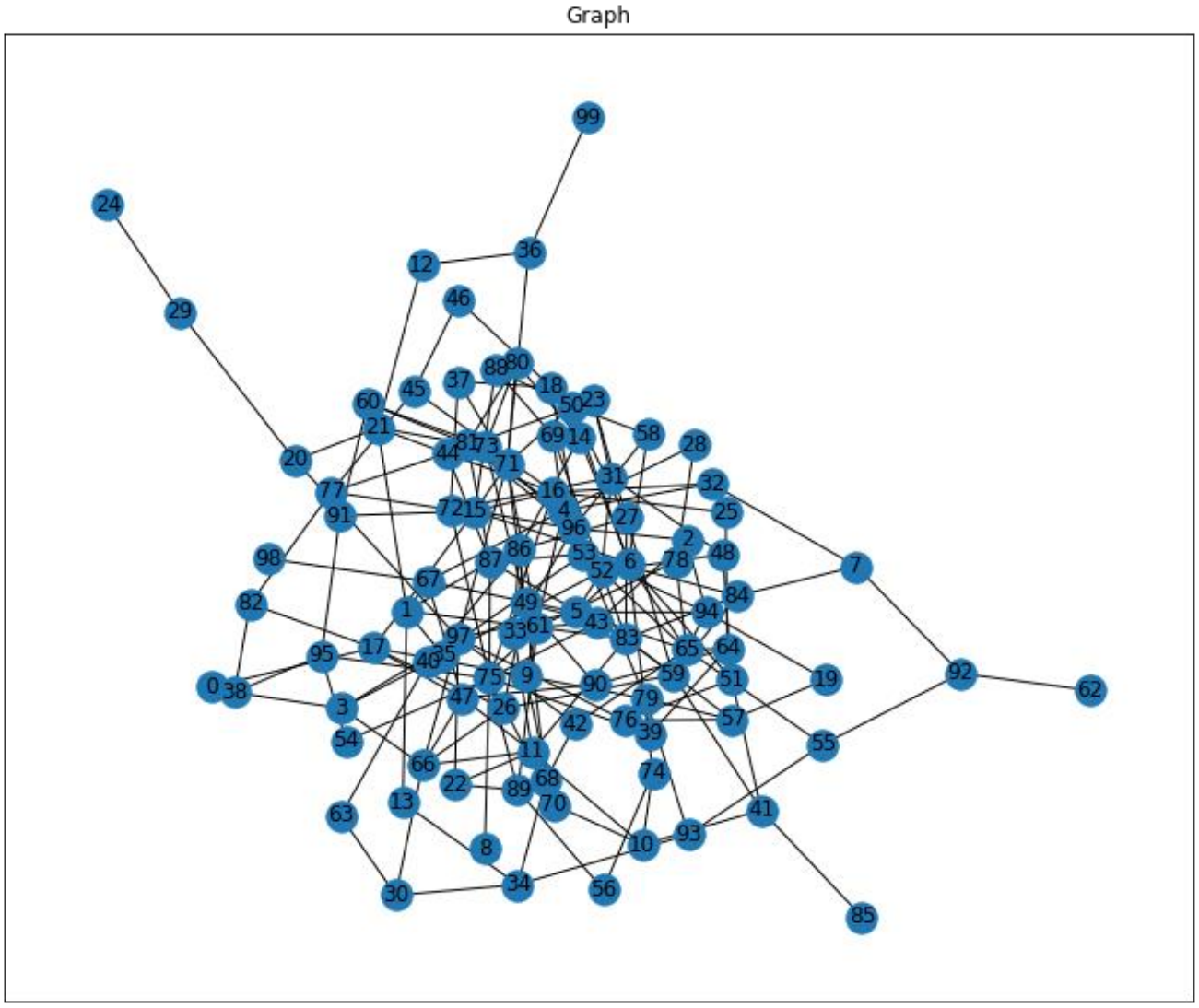
```
1 np.sum(L == 1) / 2

  200.0
```

The matrix was then converted to the adjacency list (I show first 10) :

```
0 17 38
1 13 21 35 61 67 87
2 43 94 96
3 35 38 66 97
4 9 28 32 67 69 81
5 6 52 65 67 94 97
6 14 15 50 51 58 65 74
7 32 84 92
8 75
9 17 70 79
10 11 41 70 74
```

The adjacency matrix of this graph looks less informative because the graph is sparse, i.e., the matrix has lots of zeros. But if we were to work with small weighted directed graph, the matrix representation would be much simpler and more informative than list representation.



Search of connected components

Depth-first search strategy was used to find connected components of the graph.
My start node was node number 1. So algorithm found 3 connected components with node 1.

[13, 34, 30, 63, 40, 53, 18, 14, 6, 5, 52, 31, 23, 27, 50, 6, 15, 16, 25, 64, 39, 75, 8, 16, 32, 4, 9, 17, 0, 38, 3, 35]

[21, 12, 36, 71, 15, 17, 26, 49, 43, 2, 94, 5, 65, 6, 51, 41, 10, 11, 22, 89, 56, 74, 6, 58, 50, 46, 45, 16, 44, 21, 20, 29, 24, 97, 3, 66, 11, 33, 86, 14, 80, 71, 37, 18, 72, 31, 48, 51, 76, 57, 19, 94, 53, 84, 7, 32, 92, 55, 65, 42, 68, 34, 93, 55, 79, 9, 70, 10, 74, 75, 43, 61]

[67, 4, 28, 78, 61, 47, 11, 83, 27, 59, 26, 66, 30, 47, 17, 82, 38, 95, 54, 75, 72, 77, 44, 23, 87]

Search for shortest paths in a graph

Breadth-first search strategy was used to find shortest paths between giving components.

The shortest path between node = 0 and node = 87 is

[0, 38, 95, 75, 87]

Conclusions

Random simple graph with 100 nodes and 200 edges was generated and visualized. We represent graph in 2 notations: as adjacency matrix and as adjacency list.

We found connected 3 components using Depth-first search strategy.

The analysis of the shortest paths between the nodes we used Breadth-first search strategy. We found the shortest path between nodes 0 and 87 that took 4 links.

Appendix

https://github.com/AAYamoldin/TrainingPrograms/blob/master/Python/ITMO_algorithms_lab/Task_5/task5_Algorithms_on_graphs_Introduction_to_graphs_and_basic_algorithms_on_graphs.ipynb