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

Chumakov Mike

J4132c

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

Problems and methods

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 theory is widely used in modeling various complex systems, including social, transport, communication, and other networks. Graph theory tools allow you to perform comprehensive analyzes of data represented as graphs.

An undirected graph is a pair G = (V, E), where V = {v_i} is the set of vertices (or nodes), and E ={e_{ij}} = {(v_i, v_j)} is the set of pairs of vertices called edges (or connections). The number of vertices is denoted by |V|, and the number of edges by |E|. A directed graph is a graph in which the edges have directions (orientation). A weighted graph is a graph in which each edge assigned weights. 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 are possible in the graph between the pair peaks. A complete graph is a graph in which all vertices are connected by an edge. A path (chain) in a graph is a sequence of pairwise distinct edges connecting two different vertices. The length of a path (chain) is the number of edges (or the sum of the weights of the edges) in a path (chain). Tops v_1 and v_2 in a graph are said to be connected if there is a path from v_1 to v_2. Otherwise, these vertices are said to be unconnected. Connected a graph is a graph in which any pair of vertices is connected. Otherwise case, the graph is called disconnected. The graph connectivity component is the maximal connected subgraph of a graph

Adjacency matrix, i.e. matrix whose rows and columns indexed by vertices and whose cells contain a boolean value (0 or 1), indicating whether the corresponding vertices are adjacent (for weighted graphs, the corresponding weights are replaced by 1). Adjacency matrix (as a 2D array) requires O ($|V|^2$) memory.

Another type of presentation is an adjacency list, i.e. collection of lists of vertices, where each vertex of the graph is given a list of vertices adjacent to it. Adjacency list (as 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, $|E| << |V|^2$, the adjacency list is significantly more storage efficient than the adjacency matrix.

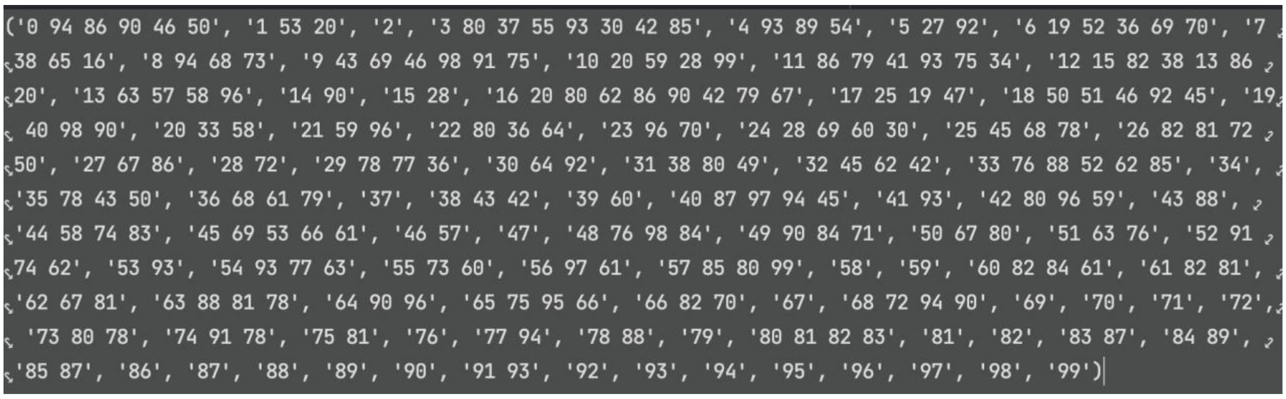
Depth-first search (DFS) is a graph traversal algorithm that starts traversing at a selected root vertex and goes "Deep" of the graph, as far as possible, before traversing from a new vertex. Its time complexity is O (| V | + | E |).

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

Both algorithms can be used to find the shortest path between vertices and to find the connected components of a graph.

Results

Fragment of adjacency list for a graph with 100 vertices and 200 edges (see Figure 1).



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

Figure 1: Fragment of adjacency list

Visualize the graph (see Figure 2)

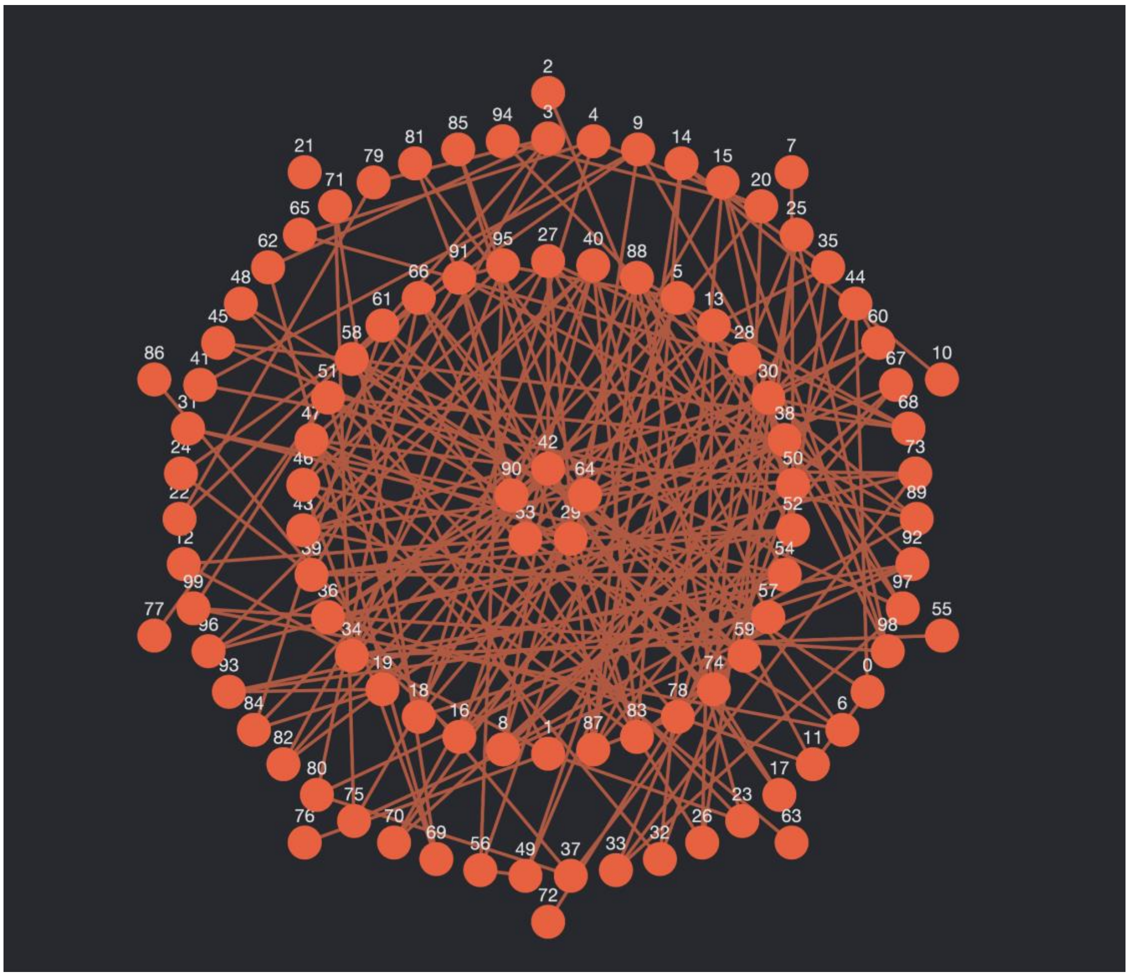


Figure 2: Graph visual representation

Pros and cons of different views

Adjacency matrix

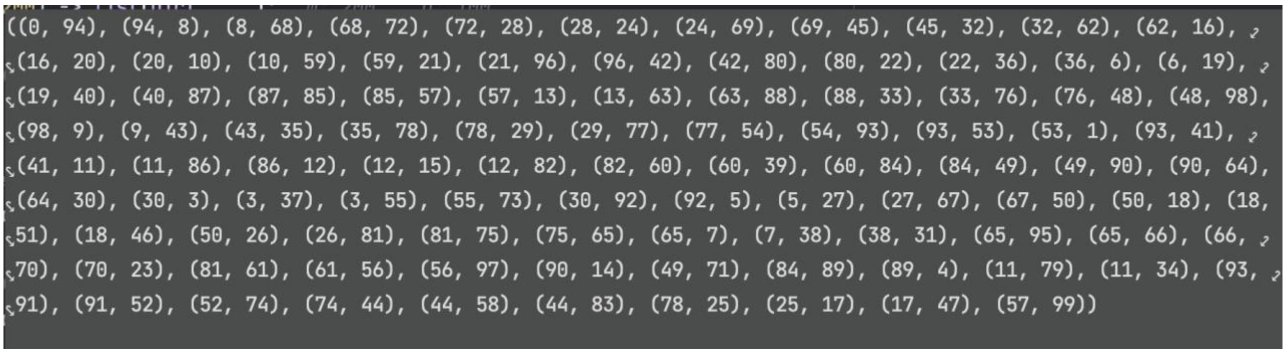
- *Uses O(n^2) memory*
- *It is fast to look up and check for presence or absence of a specific edge between any two nodes O(1)*
- *It is slow to iterate over all edges*
- *It is slow to add/delete a node; a complex operation O(n^2)*
- *It is fast to add a new edge O(1)*

Adjacency List

- *Memory usage depends on the number of edges (not the number of nodes), which can save a lot of memory if the adjacency matrix is sparse*
- *Finding the presence or absence of a specific edge between any two nodes slightly slower than with O (k) matrix; where k is the number of neighboring nodes*
- *It is fast to iterate over all edges because you can access any node neighbors directly*
- *It is fast to add/delete a node; easier than the matrix representation*
- *It fast to add a new edge O(1)*

Depth-first search

Graph traversal from edge zero (see Figure 3)

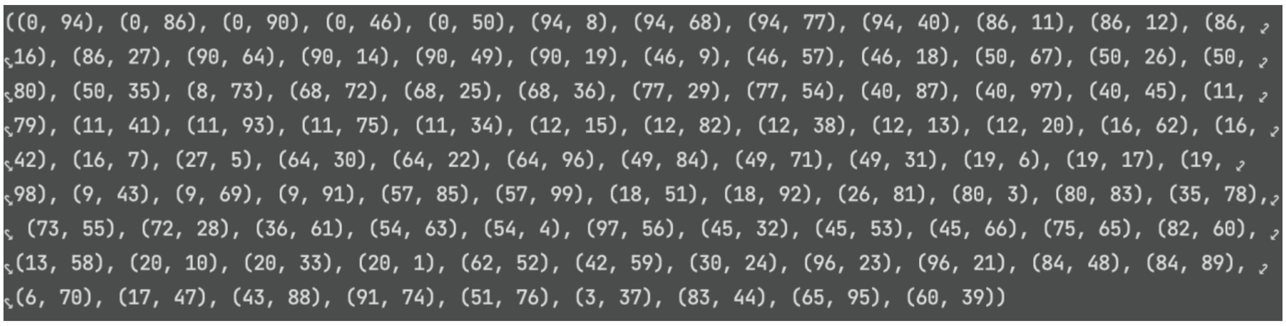


```
((0, 94), (94, 8), (8, 68), (68, 72), (72, 28), (28, 24), (24, 69), (69, 45), (45, 32), (32, 62), (62, 16), 
(16, 20), (20, 10), (10, 59), (59, 21), (21, 96), (96, 42), (42, 80), (80, 22), (22, 36), (36, 6), (6, 19), 
(19, 40), (40, 87), (87, 85), (85, 57), (57, 13), (13, 63), (63, 88), (88, 33),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

Figure 4: Fragment of Depth-first list

Breadth-first search

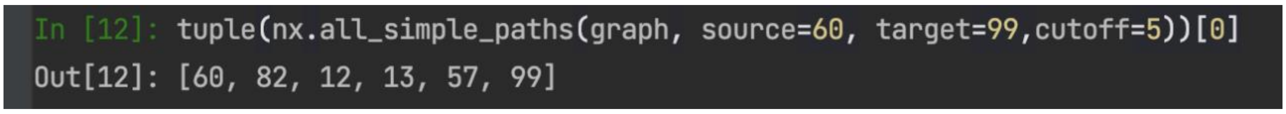
Graph traversal from edge zero (see Figure 4)



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

Figure 4: Fragment of Breadth-first list

Example of finding the shortest path from edge 60 to 99 (see Figure 6)



```
In [12]: tuple(nx.all_simple_paths(graph, source=60, target=99,cutoff=5))[0]
Out[12]: [60, 82, 12, 13, 57, 99]
```

Figure 5: Shortest path for edges 60 to 99 by Breath-first search

Conclusions

In the course of the laboratory work, various representations of graphs were considered and several basic algorithms were implemented on graphs (depth-first search and breadth-first search). The pros and cons of various representations were also considered, and examples of algorithms and representations for a graph with 100 vertices and 200 edges were given. Algorithms and visualization of the graph was done using the Python language and the library networkx.

Appendix

Source Link and Graph Link