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
Alexandra Matveeva
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).

Problems

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 Breadthfirst 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 **graph** is a data structure that is defined by two components:

- a node or a vertex

- an edge E or ordered pair is a connection between two nodes u, v that is identified by unique pair(u, v). The pair (u, v) is ordered because (u, v) is not same as (v, u) in case of directed graph. The edge may have a weight or is set to one in case of unweighted graph.

Types of graph:

- Directed graph:

A graph in which the direction of the edge is defined to a particular node is a directed graph.

Directed Acyclic graph: It is a directed graph with no cycle. For a vertex 'v' in DAG there is no directed edge starting and ending with vertex 'v'.

Tree: A tree is just a restricted form of graph. That is, it is a DAG with a restriction that a child can have only one parent.

- Undirected graph:

A graph in which the direction of the edge is not defined. So if an edge exists between node 'u' and 'v',then there is a path from node 'u' to 'v' and vice versa.

Connected graph: A graph is connected when there is a path between every pair of vertices. In a connected graph there is no unreachable node.

Complete graph: A graph in which each pair of graph vertices is connected by an edge. In other words, every node 'u' is adjacent to every other node 'v' in graph 'G'.A complete graph would have $n(n-1)/2$ edges. See below for proof.

Biconnected graph: A connected graph which cannot be broken down into any further pieces by deletion of any vertex. It is a graph with no articulation point.

Depth First Search (DFS) algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

Algorithm:

Step 1: Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.

Step 2: If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)

Step 3: Repeat Step 1 and Step 2 until the stack is empty.

Breadth-first search (BFS) is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.

Algorithm:

Step 1: Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.

Step 2: If no adjacent vertex is found, remove the first vertex from the queue.

Step 3: Repeat Step 1 and Step 2 until the queue is empty.

Results

I. Adjacency matrix:

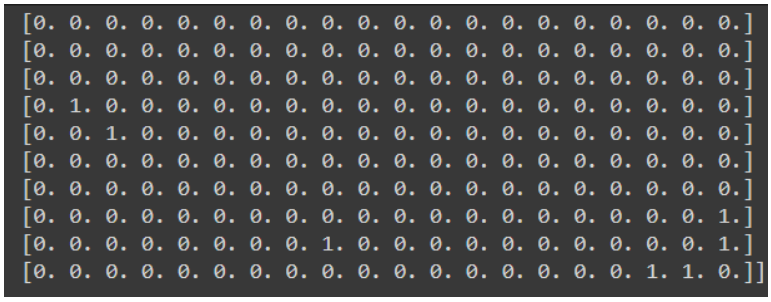

Fig. 1 – Adjacency matrix

Adjacency list:

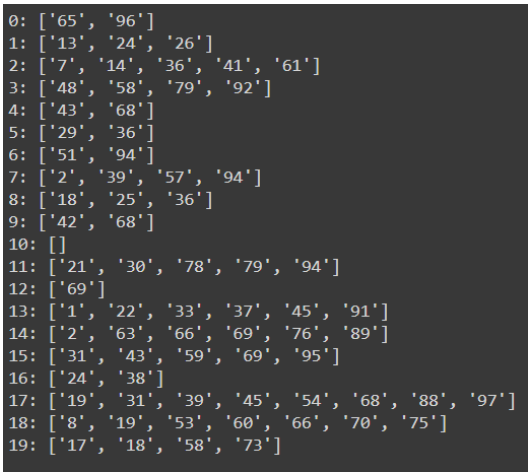

Fig. 2 – Adjacency matrix

Adjacency matrix:

- uses $O(n^2)$ memory

- it quickly searches for and checks for the presence or absence of a certain edge between any two nodes $O(1)$

- iteration over all edges is slow

- adding/removing a node is slow; complex operation $O(n^2)$

- quickly 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 certain edge between any two nodes is slightly slower than with the matrix $O(k)$; where k is the number of neighboring nodes

- iteration over all edges is fast, since you can directly access any neighbors of nodes

- quickly add/remove node; easier than matrix representation

- it's fast to add a new edge $O(1)$

Generated graph:

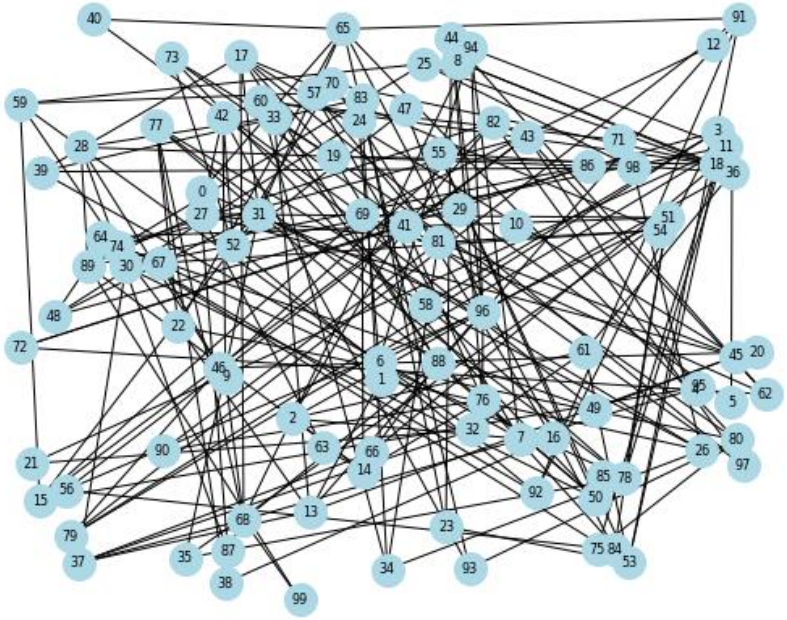

Fig. 3 – Visualization of the generated graph

II.

Depth-first search (DFS):

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

Breadth-first search (BFS):

```
Shortest path =  3 48 89 99
```

Breadth-first search has a running time of $O(V + E)$ since every vertex and every edge will be checked once. Depending on the input to the graph, $O(E)$ could be between $O(1)$ and $O(V^2)$.

Depth-first search visits every vertex once and checks every edge in the graph once. Therefore, DFS complexity is $O(V + E)$. This assumes that the graph is represented as an adjacency list.

III.

The data structure of a **graph** is a set of vertices that have data and are connected to other vertices.

A **linked list** is a linear data structure, in which the elements are not stored at contiguous memory locations

Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO).

Conclusion

During the execution of laboratory work, simple undirected unweighted random graph was generated. Connected components of the graph were found by Depth-first search method. Also shortest path between two random vertices was found by Breadth-first search.

Appendix

GitHub Link: https://github.com/alex-mat-s/Algorithms/blob/main/Lab5.ipynb