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

Anastasiia Pokasova

J4133c+

Accepted by

Dr Petr Chunaev

St. Petersburg
2020

Goal

The use of different representations of graphs and basic algorithms on graphs (Depth-first search and Breadth-first search).

Problems and methods

- 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?
- 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.
- Describe the data structures and design techniques used within the algorithms.

Brief theoretical part

Graph Undirected graph is an ordered pair $G = (V, E)$, where V is a set of vertices and E is a set of edges.

Graph can be represented using any of listed structures:

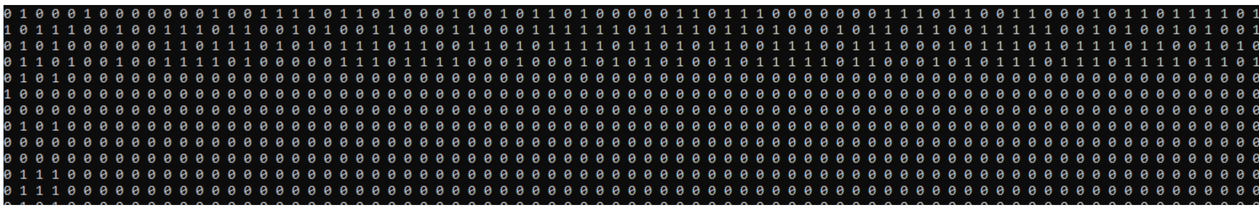
- Adjacency list
- Adjacency matrix
- Incidency list
- Incidency matrix

Adjacency matrix is a $|V|X|V|$ square matrix which elements indicate whether pairs of vertices are adjacent or not in the graph.

Adjacency list consists of V lists one for each vertex v_i, which gives the vertices to which v_i is adjacent.

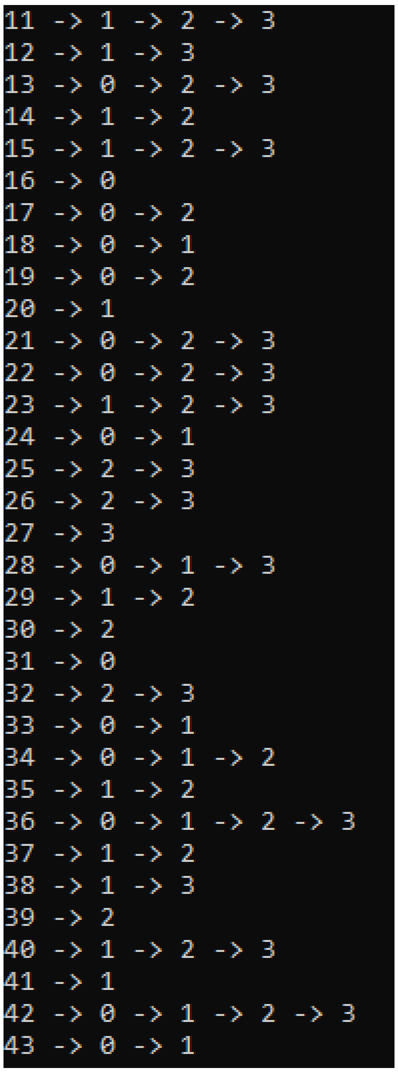
Whether to choose matrix or list representation depends on the task. For example, matrix is symmetric, so it's not necessary to store the whole matrix and reduce space complexity of an algorithm.

Adjacency matrix, list and visualization of example graph of 100 vertices and 200 edges are shown below.



```
0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 1 1 0 1 0 0 0 1 0 0 1 0 1 1 0 1 0 0 0 0 0 1 1 0 1 1 1 0 0 0 0 0 0 0 1 1 1 0 1 1 0 0 1 1 0 0 0 1 0 1 1 0 1 1 1 1 0 1
1 0 1 1 1 0 0 1 0 0 1 1 1 0 1 1 0 0 1 0 1 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 0 1 1 1 1 0 1 1 0 1 0 0 0 1 0 1 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 1 0 0 1 0 1 0 0 1
0 1 0 1 0 0 0 0 0 0 1 1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 0 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 1 0 1 1 1 0 1 0 1 1 1 0 1 1 0 0 1 0 1 0
0 1 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 1 1 1 0 0 0 1 0 0 0 1 0 1 0 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1 0 0 0 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 1 1 0 1
0 1 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 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 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 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 0 0 0 0 0 0
0 1 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 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 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 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 1 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 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 1 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 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
```

Figure 1: Adjacency matrix



```
11 -> 1 -> 2 -> 3
12 -> 1 -> 3
13 -> 0 -> 2 -> 3
14 -> 1 -> 2
15 -> 1 -> 2 -> 3
16 -> 0
17 -> 0 -> 2
18 -> 0 -> 1
19 -> 0 -> 2
20 -> 1
21 -> 0 -> 2 -> 3
22 -> 0 -> 2 -> 3
23 -> 1 -> 2 -> 3
24 -> 0 -> 1
25 -> 2 -> 3
26 -> 2 -> 3
27 -> 3
28 -> 0 -> 1 -> 3
29 -> 1 -> 2
30 -> 2
31 -> 0
32 -> 2 -> 3
33 -> 0 -> 1
34 -> 0 -> 1 -> 2
35 -> 1 -> 2
36 -> 0 -> 1 -> 2 -> 3
37 -> 1 -> 2
38 -> 1 -> 3
39 -> 2
40 -> 1 -> 2 -> 3
41 -> 1
42 -> 0 -> 1 -> 2 -> 3
43 -> 0 -> 1
```

Figure 2: Adjacency matrix

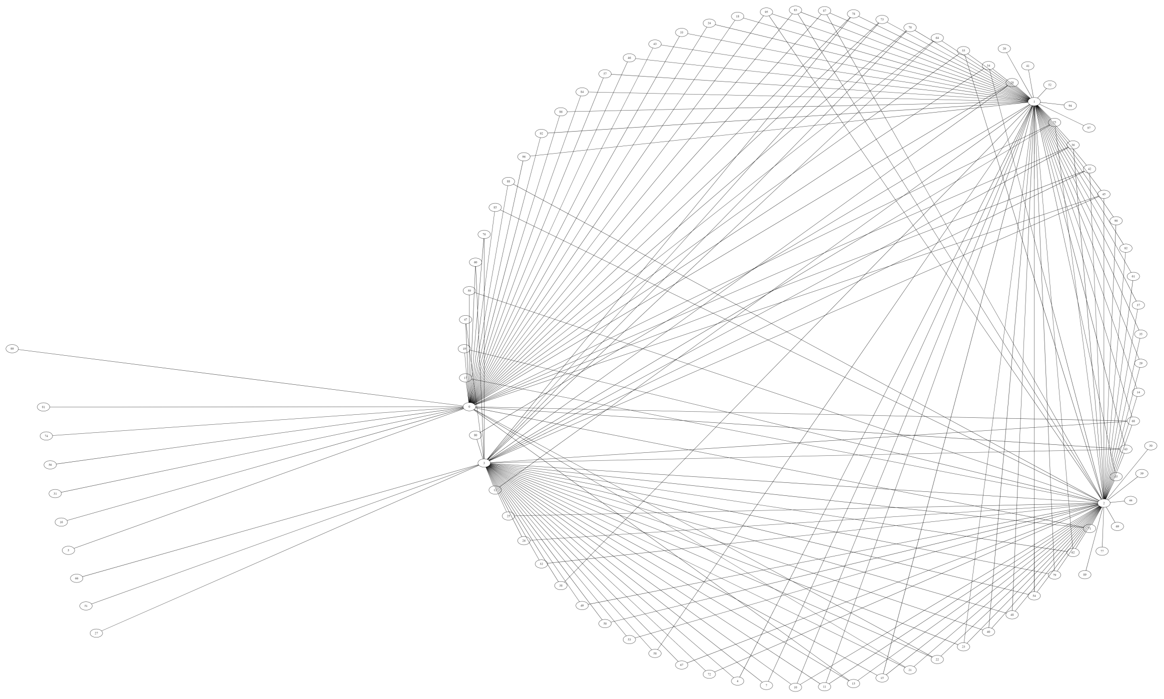


Figure 3: Graph with 100 vertices and 200 edges

Breadth-first search 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 may be used for finding shortest path in graph.

Time Complexity: $O(|V| + |E|)$ Space complexity: $O(V)$

Depth-first search Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

In graph theory, a component, sometimes called a connected component, of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph.

Time Complexity: $O(|V| + |E|)$ Space complexity: $O(V)$

Results

Both algorithms traverse the whole graph in worst case, so it is estimated that they will have almost similar time results.

Measures give reaults of 1969 and 2016 microseconds.

Design and data structures

Data structures used in this laboratory work are listed below:

- matrix
- linked list
- stack
- queue

Matrix is a NXM table of elements.

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. A linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.

A stack is a basic data structure that can be logically thought of as a linear structure represented by a real physical stack or pile, a structure where insertion and deletion of items takes place at one end called top of the stack. The basic concept can be illustrated by thinking of your data set as a stack of plates or books where you can only take the top item off the stack in order to remove things from it. This structure is used all throughout programming.

The order is LIFO (Last In First Out) to demonstrate the way it accesses data, since as we will see there are various variations of stack implementations.

There are basically three operations that can be performed on stacks: inserting an item into a stack (push), deleting an item from the stack (pop), displaying the contents of the stack (peek or top).

A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). Can be implemented via array or linked list.

Conclusion

As a part of a laboratory work, basic graph algorithms were compared. A program was implemented for the task of comparison.

Source code

Source code for the implementation is located here: here: `https://github.com/NeoIsALie/Algorithm_Analysis/tree/lab5`.