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

Rami Al-Naim

J4132c

Accepted by

Dr Petr Chunaev

St. Petersburg
2020

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

A random adjacency matrix for a simple undirected unweighted graph with 100 vertices and 200 edges is generated. The matrix then transferred into an adjacency list. During the next step the graph, adjacency matrix are visualized.

Depth-first search are going to be used for finding connected components of the graph. Breadth-first search is used to find a shortest path between two random vertices. The results will be compared based on computation time.

3 Brief theoretical part

An undirected graph is a pair $G = (V, E)$, where V is a set whose elements are called vertices (or nodes), and E is a set of sets with two distinct vertices,whose elements are called edges.

Most of the time graph is represented in two ways:

- Adjacency List is a collection of linked list of vertices. Each vertex points on it's neighbours. In `Python` the dictionary of lists is widely used. In this case the keys are the vertices and the value is the list on neighbours' keys. With some modification, this method can be used to represent weighted graphs.

- Adjacency Matrix: Adjacency Matrix is a two dimensional array of size V x V, where V is the number of vertices in a graph. The value with indexes i and j shows the presence of edge between the vertex i and vertex j. If this value is 0 then there is no edge. Adjacency matrix for an undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

Those two ways of representing a graph have different complexety for operations on graphs:

- Adding a vertex.
  - Adjacency matrix - In order to add a new vertex to VxV matrix the storage must be increases to $(|V| + 1)^2$. To achieve this we need to copy the whole matrix. Therefore the complexity is $O(|V|^2)$.
  - Adjacency list - There are two pointers in adjacency list first points to the front node and the other one points to the rear node. Thus insertion of a vertex can be done directly in O(1) time.

- Adding an edge.
  - Adjacency matrix - To add an edge from i to j the following operation should be done: `matrix[i][j] = 1`, which requires O(1) time.
  - Adjacency list - Similar to insertion of vertex here also two pointers are used pointing to the rear and front of the list. Thus, an edge can be inserted in $O(1)$ time.
- Removing a vertex.
  - Adjacency matrix - In order to remove a vertex from V*V matrix the storage must be decreased to $|V|^2$ from $(|V|+1)^2$. To achieve this we need to copy the whole matrix. Therefore the complexity is $O(|V|^2)$.
  - Adjacency list - In order to remove a vertex, we need to search for the vertex which will require $O(|V|)$ time in worst case, after this we need to traverse the edges and in worst case it will require $O(|E|)$ time. Hence, total time complexity is $O(|V| + |E|)$.
- Removing an edge.
  - Adjacency matrix - To remove an edge from i to j, the following operation should be done: `matrix[i][j] = 0`, which requires $O(1)$ time.
  - Adjacency list - To remove an edge traversing through the edges is required and in worst case we need to traverse through all the edges.Thus, the time complexity is $O(|E|)$.

Depth-first search (DFS) is an algorithm for traversing or searching a graph. The algorithm starts at a chosen root vertex and explores as far as possible along each branch before backtracking.

Breadth-first search (BFS) is an algorithm for traversing or searching a graph. The algorithm starts at a chosen root vertex and explores all of the neighbour vertices at the present depth prior to moving on to the vertices at the next depth level.

This work was done using Python3.7 and mainly with module `networkx`.

4 Results

The random graph was generated using `networkx`'s random graph generator function (Fig. 1). The adjacency matrix was then fetched and visualized as a binary image, where white pixel is the edge between corresponding modes (Fig. 2). Then the adjacency matrix was transferred to dictionary of lists (analogue of adjacency list in `Python`). First 5 entries in it can be seen in Figure 3.

At the next step the Depth-first search was used find connected components of the graph. Connected component is a set of vertices connected by paths. The following algorithm was used: a list of not-visited nodes were initialized with all

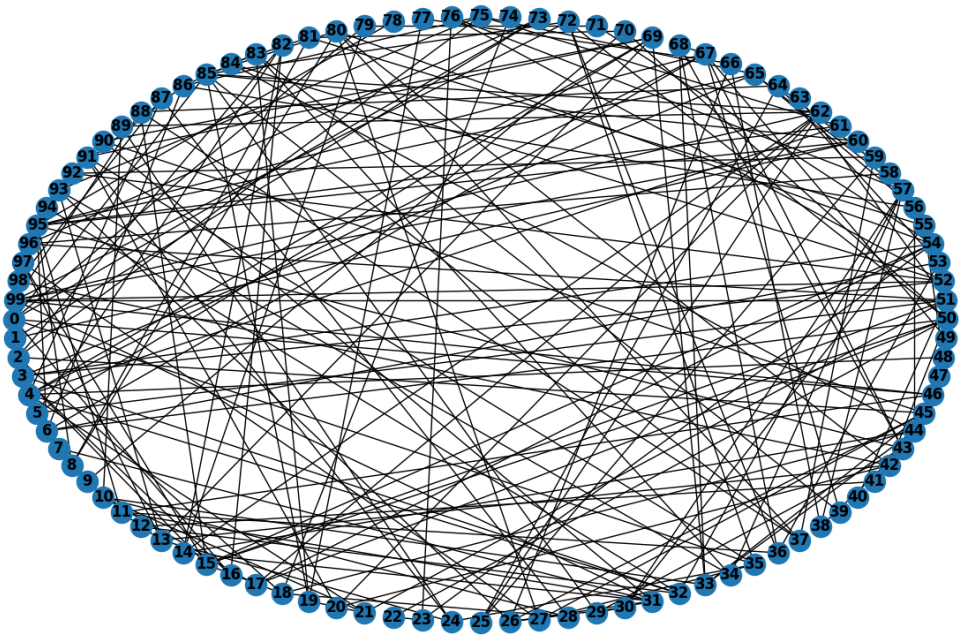

Figure 1: Generated graph with 100 vertices and 200 edges

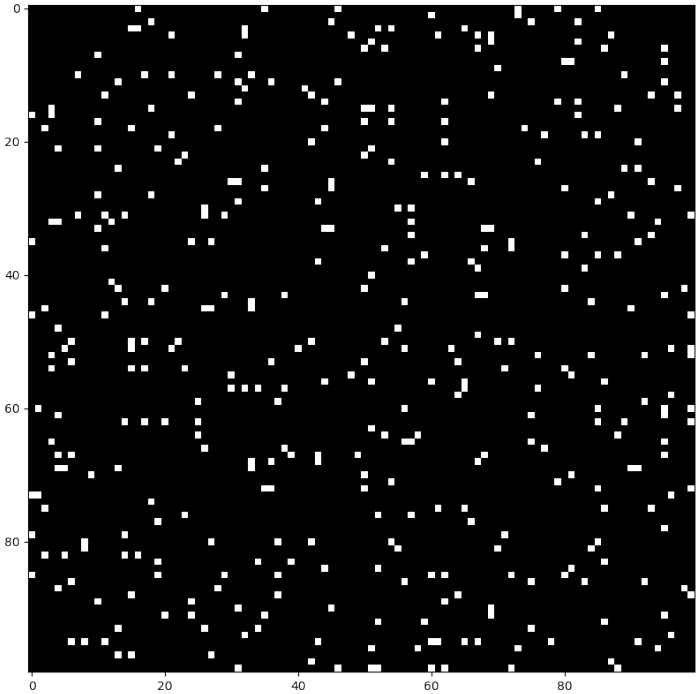

Figure 2: Adjacency matrix

```
0: [73, 46, 16, 79, 85, 35]
1: [73, 60]
2: [75, 18, 45, 82]
3: [52, 32, 15, 16, 54, 65]
4: [69, 32, 48, 67, 61, 87, 21]
```

Figure 3: First 5 collections from adjacency list

the nodes from the graph. DFS was launched from the first node, and all the nodes to which the path was found are excluded from the non-visited list. The next starting node is chosen from the non-visited list. While the non-visited list is not empty, this operations repeats. This way DFS launched the same amount of time as there are connected components in the graph.

Two connected components found in the graph can be seen in the Figure 4.

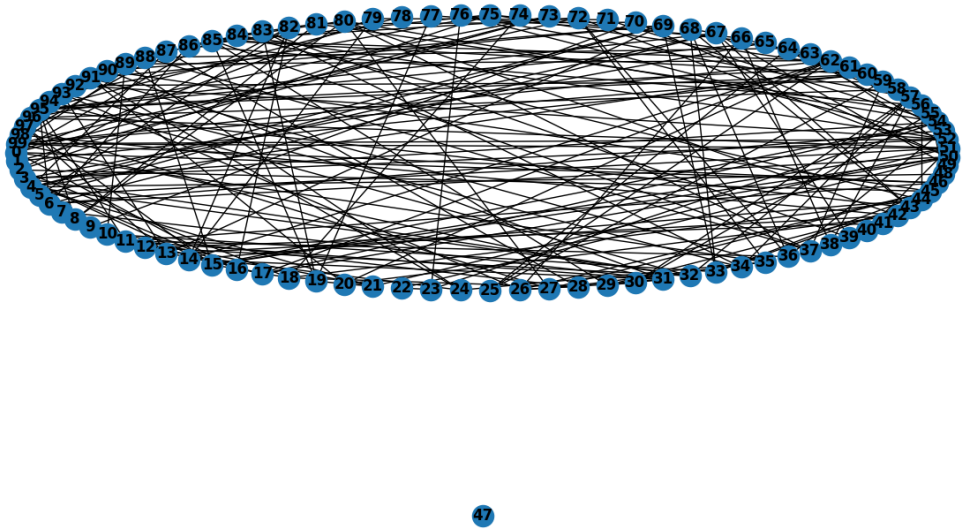

Figure 4: Connected components of the graph

The BFS was used to find the shortest path between pair of random nodes. Since the graph is undirected, the length of the path between two nodes is simply the depth level at which the target node was found.

The path found between nodes 95 and 87 can be sees in Figure 5.

Those two algorithms were compared on time complexity. Each algorithm was executed 1000 times. The results are present in the Table 1.

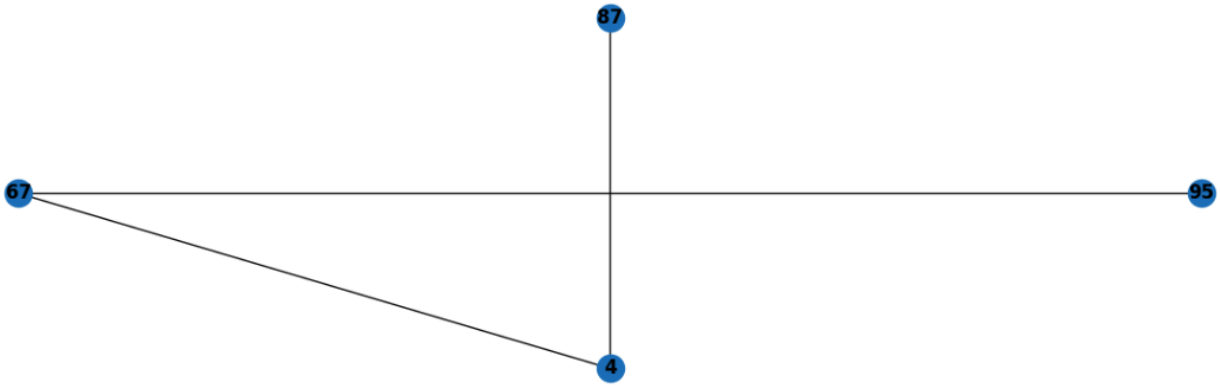

Figure 5: Connected components of the graph

Algorithm	Average execution time, ms
DFS	1.120
BFS	0.151

Table 1: Comparison of two algorithms

5 Conclusions

During this work the graphs and the main algorithms on them were studied.

The random graph with 100 vertices and 200 edges was generated. This graph was represented in two different ways: using adjacency matrix and adjacency list. The difference between them and time complexity of basic operations were described above.

The DFS was used to find the connected component of the generated graph. The BFS was used to find the shortest path between two random nodes in the graph.

The great difference between two algorithms (Table 1) can be explained as follows: `networkx`'s implementation of both DFS and BFS are optimized inside the module itself. The implementation of finding the connected components include some operations on lists and sorting, so there is a place for further optimization.

Appendix

The Python code for this task can be found here: GitHub.