FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report on the practical task No. 6
“Algorithms on graphs. Path search algorithms on weighted graphs”

Performed by Pavel Belenko, J4132C

Accepted by Dr Petr Chunaev

St. Petersburg
2021

Goal:

The use of path search algorithms on weighted graphs (Dijkstra's, A* and Bellman-Ford algorithms)

Formulation of the problem:

I. Generate a random adjacency matrix for a simple undirected weighted graph of 100 vertices and 500 edges with assigned random positive integer weights (note that the matrix should be symmetric and contain only 0s and weights as elements). Use Dijkstra's and Bellman-Ford algorithms to find shortest paths between a random starting vertex and other vertices. Measure the time required to find the paths for each algorithm. Repeat the experiment 10 times for the same starting vertex and calculate the average time required for the paths search of each algorithm. Analyze the results obtained.

II. Generate a 10×20 cell grid with 40 obstacle cells. Choose two random non-obstacle cells and find a shortest path between them using A* algorithm. Repeat the experiment 5 times with different random pair of cells. Analyze the results obtained.

III. Describe the data structures and design techniques used within the algorithms.

Brief theoretical part:

A brief introduction to graph theory and the corresponding definitions and notations were given in the practical task No. 5, “Algorithms on graphs. Introduction to graphs and basic algorithms on graphs”.

A weighted graph is a graph in which each child is assigned a weight (some number). Weighted graphs require special traversal algorithms. Let's look at some of them:

1) Dijkstra's algorithm (DA) solves the following problem: for a given graph (with positive weights) and a source vertex S, find the shortest paths from S to the remaining vertices.

   The main idea of Dijkstra's algorithm is that it generates a shortest path tree (SPT) with root S by processing two sets: one set contains vertices included in SPT, the other contains vertices not yet included in SPT. At each step, the algorithm finds a vertex that is not included in the SPT and has a minimum

distance from the source. The complexity of the algorithm is estimated from $O(V \cdot \log |V|)$ to $O(V^2)$ depending on the modification applied.

2) Algorithm A* solves the following problem: for graph data (with positive weights), source S and target T, find the shortest path from S to T.

   The main idea of the algorithm is that at each iteration it determines how to extend the path based on the cost of the current path from S to the extension point and the cost estimate of the path from the extension point to T (this is a heuristic in the A* algorithm). The time complexity of the algorithm is $O(|E|)$.

   It should be noticed that the algorithm A* yields Dijkstra's algorithm if the above estimate (heuristic) is omitted.

3) The Bellman-Ford algorithm (BFA) solves the following problem: for a given weighted graph (possibly with negative weights!) and a source S, find the shortest paths from S to all other vertices. If the graph contains a negative cycle C_- (i.e., a cycle in which the sum of edges is negative) reachable from S, then there is no shortest path: any path having a vertex in C_- can be made shorter by another traversal of C_-. In this case, the BFA reports about existence of the negative cycle C_-.

   The basic idea of BFA is as follows. At the i-th iteration, the BFA calculates the shortest paths that have at most i edges. Since there are at most $|V| - 1$ edges in any simple path, $i = 1, \dots, |V| - 1$. Assuming that there is no C_-, if we have calculated shortest paths from at most i edges, then iterating over all edges guarantees obtaining shortest paths from at most $i + 1$ edges. To check if there is a negative cycle C_-, BFA performs the $|V|$-th iteration. If at least one of the shortest paths becomes shorter, then there is a negative cycle C_-. The time complexity of BFA is very high and is estimated as $O(|V| \cdot |E|)$.

Results:

All calculations were performed in the Jupiter lab development environment. NumPy, random, time, matplotlib and networkx modules were used in the process.

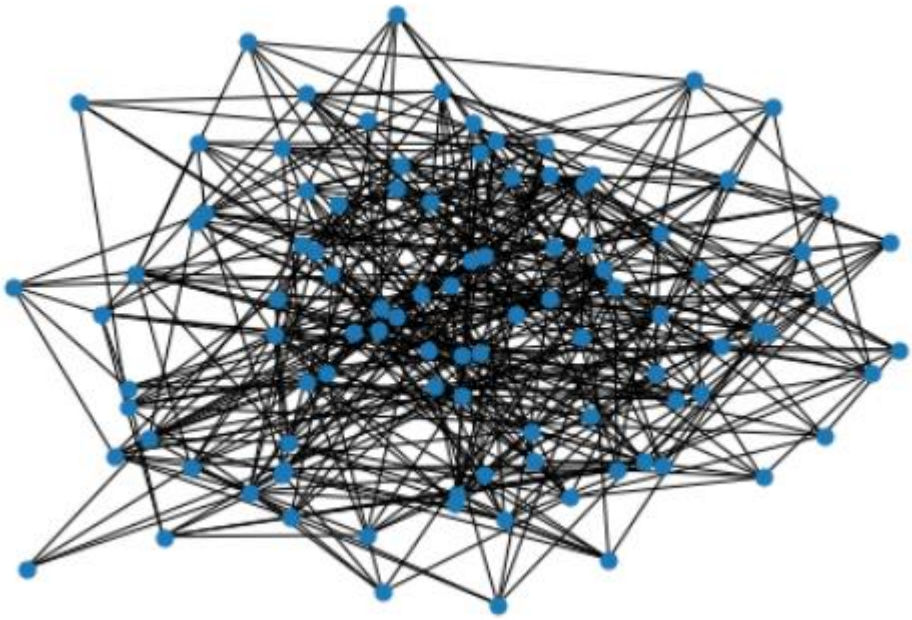

Figure 1 – visualization of the weighted graph with 100 vertices and 500 edges

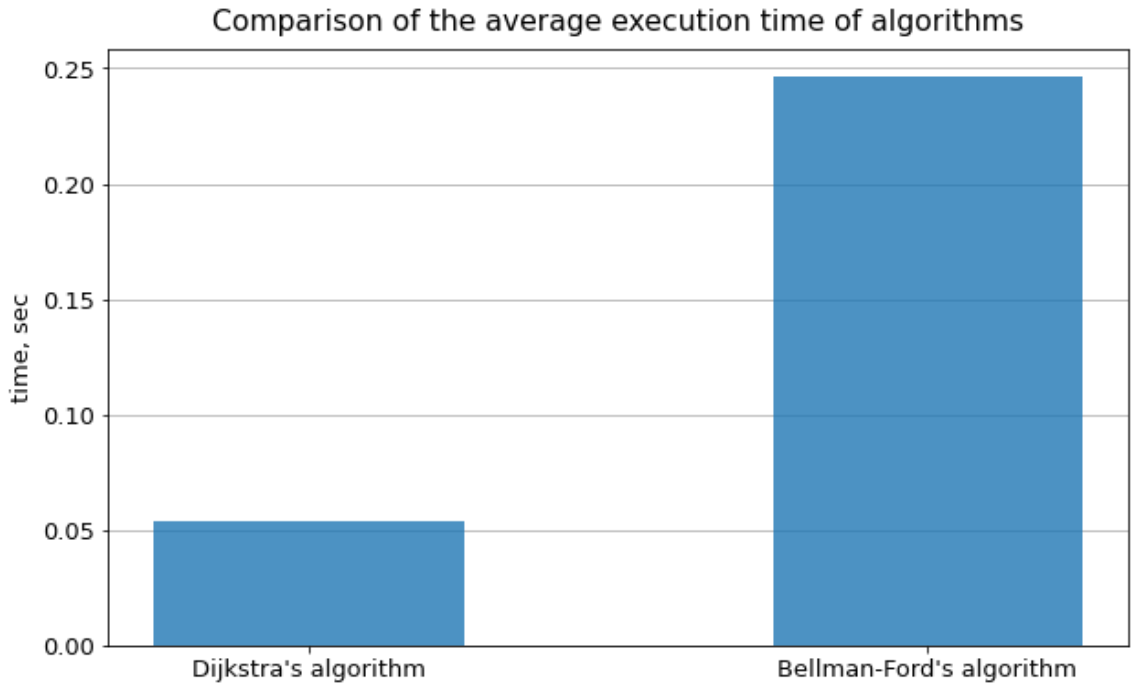

Figure 2 – comparison of the average execution time of algorithms applied to the graph from the Figure 1

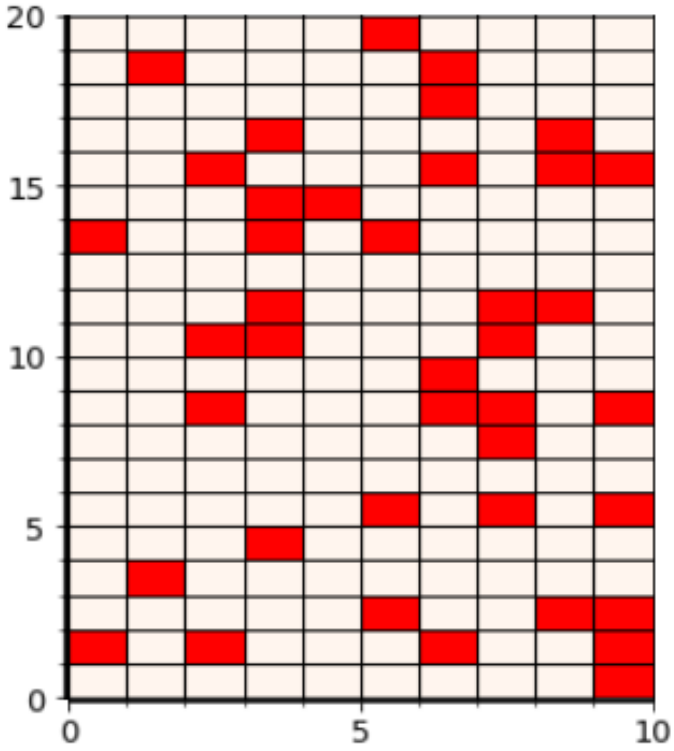

Figure 3 – visualization of the generated cell grid with 40 obstacle cells. The number of each cell corresponds to the coordinates of its lower left corner.

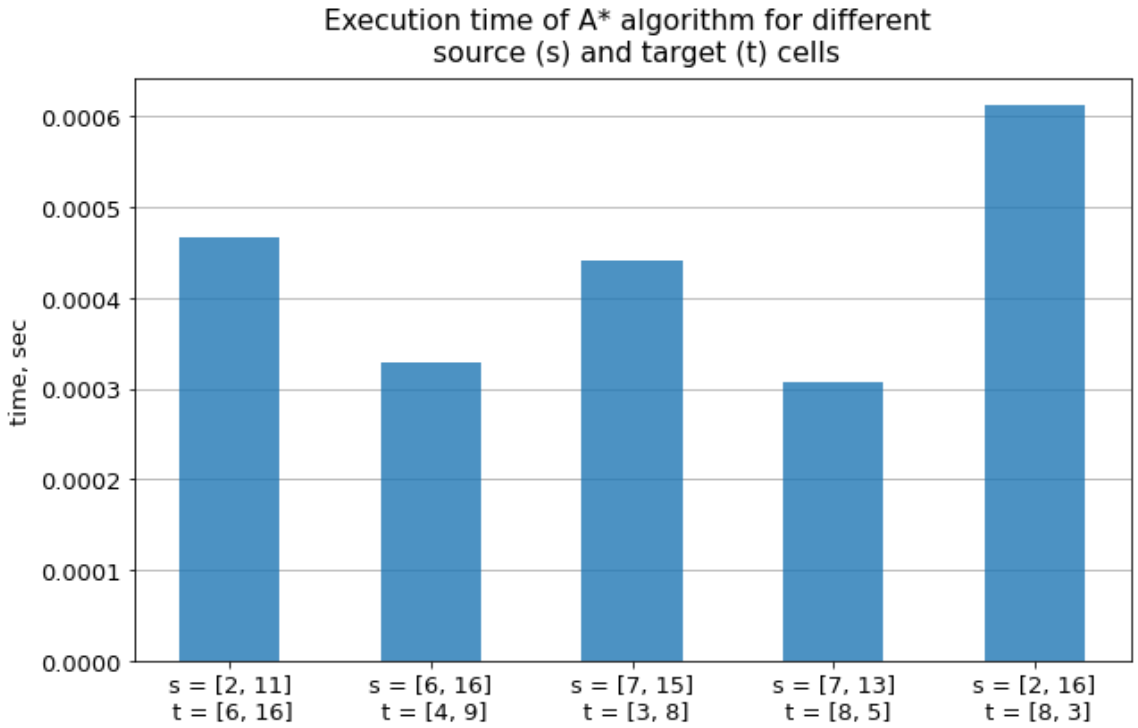

Figure 4 – comparison of the execution time of an A algorithm in five different cases applied to the graph from the Figure 3*

Conclusions:

I. As we can see from Figure 2, Bellman-Ford’s algorithm works several times slower than Dijkstra's algorithm. This was to be expected based on estimates of time complexity.

It is also worth mentioning that although the Bellman-Ford algorithm has shown worse results in this work, it is more applicable than the same Dijkstra algorithm, because it can also be used to search for shortest distances on weighted graphs with negative weights.

II. Regarding the results of the A* algorithm: in five different cases, it has approximately the same speed of operation, from which it can be assumed that the algorithm is quite stable.

Appendix:

Link to the code on the GitHub