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
Alexander Yamoldin
j4134c
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 thatthe 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. Analyse the results obtained.
II. Generate a 10x10 cell grid with 30 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. Analyse the results obtained.*
III. Describe the data structures and design techniques used within the algorithms.

Brief theoretical part

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph. It generates a shortest path tree (SPT) with the source as a root, with maintaining two sets: one set contains vertices included in SPT, other set includes vertices not yet included in SPT. At every step, it finds a vertex which is in the other set and has a minimum distance from the source. Algorithm has time complexity $O(|V|^2)$.

Bellman-Ford algorithm. The idea can be expressed as follows: at *i*-th iteration, Bellman-Ford calculates the shortest paths which has at most i edges. As there is maximum $|V| - 1$ edges in any simple path, $i = 1, \ldots, |V| - 1$. Assuming that there is no negative cycle, if we have calculated shortest paths with at most i edges, then an iteration over all edges guarantees to give shortest paths with at most $(i + 1)$ edges. To check if there is a negative cycle, make $|V|$-th iteration. If at least one of the shortest paths becomes shorter, there is such a cycle. The time complexity of this algorithm is $O(|V||E|)$ (which is $O(|V|^3)$ in the worst-case scenario).

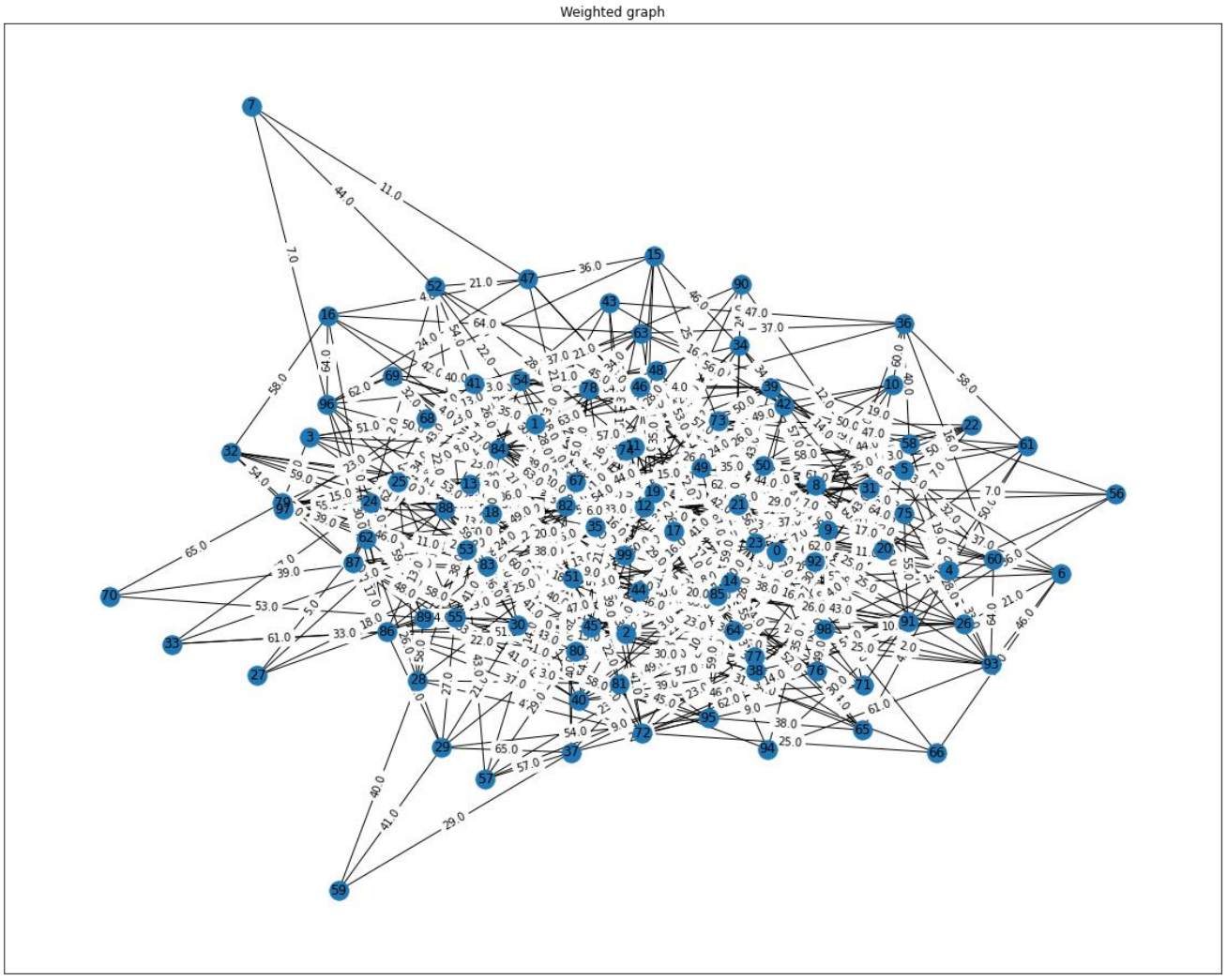
A* algorithm is algorithm for finding the shortest paths in a graph. **A*** is an informed search algorithm, or a best-first search: starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost. It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.

At each iteration of its main loop, **A*** needs to determine which of its paths to extend. It does so based on the cost of the path and an estimate of the cost required to extend the path all the way to the goal. **A*** exploits *heuristic technique* which in general helps it to work faster that Dijkstra's algorithm. Its estimated time complexity is $O(|E|)$.

Results

Task 1

Weighted graph containing 100 nodes and 500 edges with positive weights lying in the range from 1 to 66 was created.



First string in weight matrix looks like this:

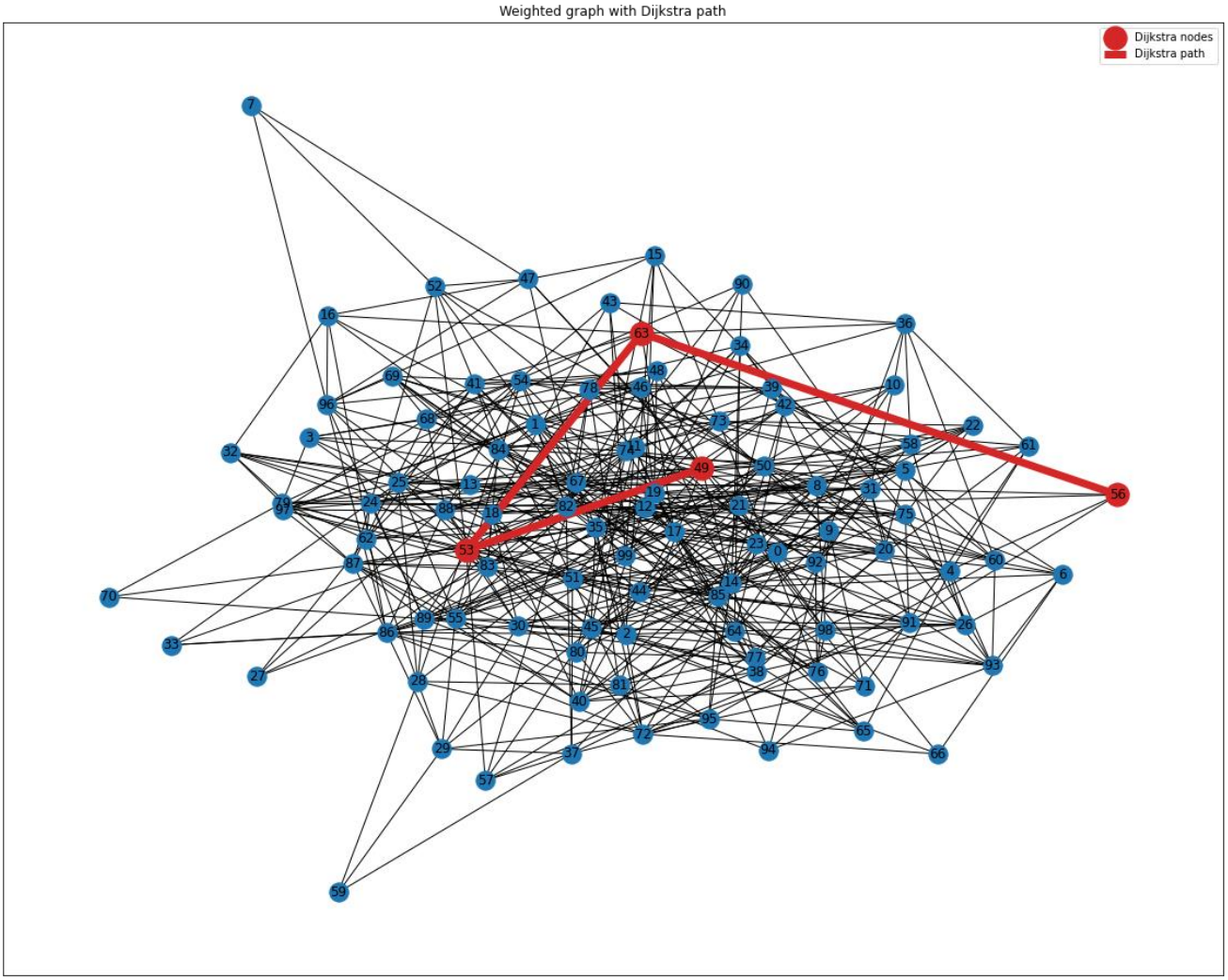
[0. 40. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 22. 0. 0. 0. 0. 0.
0. 0. 0. 0. 38. 0. 0. 0. 0. 0. 0. 0. 0. 56. 0. 0. 0. 63.
0. 0. 0. 0. 21. 0. 0. 0. 10. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 53. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 62. 0. 50. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

Dijkstra vs Bellman-Ford

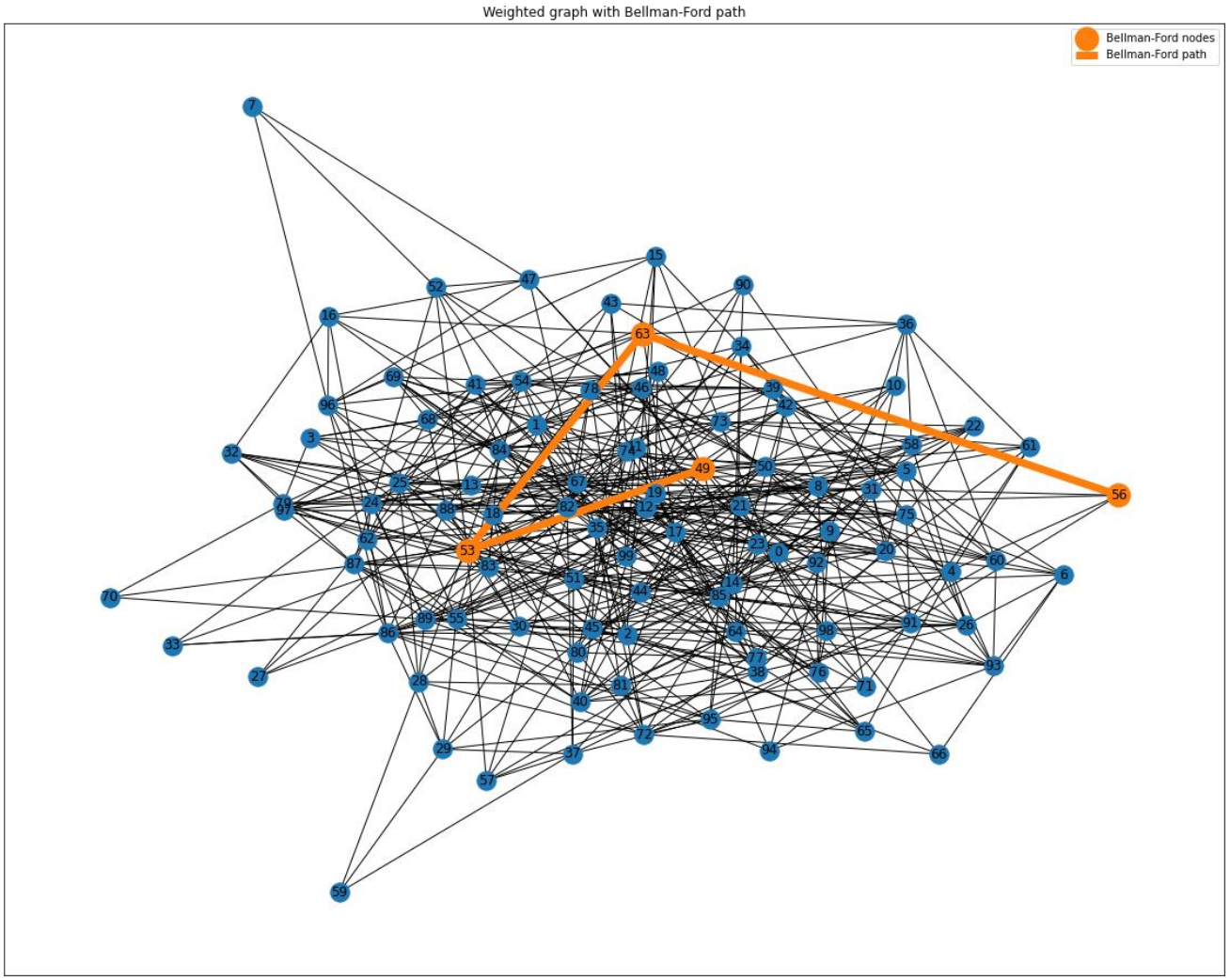
Both Dijkstra algorithm and Bellman-Ford algorithm were used to find shortest path between two random chosen nodes in this graph.
Our random chosen nodes was
noda_start = 49
noda_finish = 56
Both of algorithms chose the same path to reach noda_finish. The route was
noda_start -> 53 -> 63 -> noda_finish

The result of algorithms we visualizate in graphs below

Dijkstra



Bellman-Ford

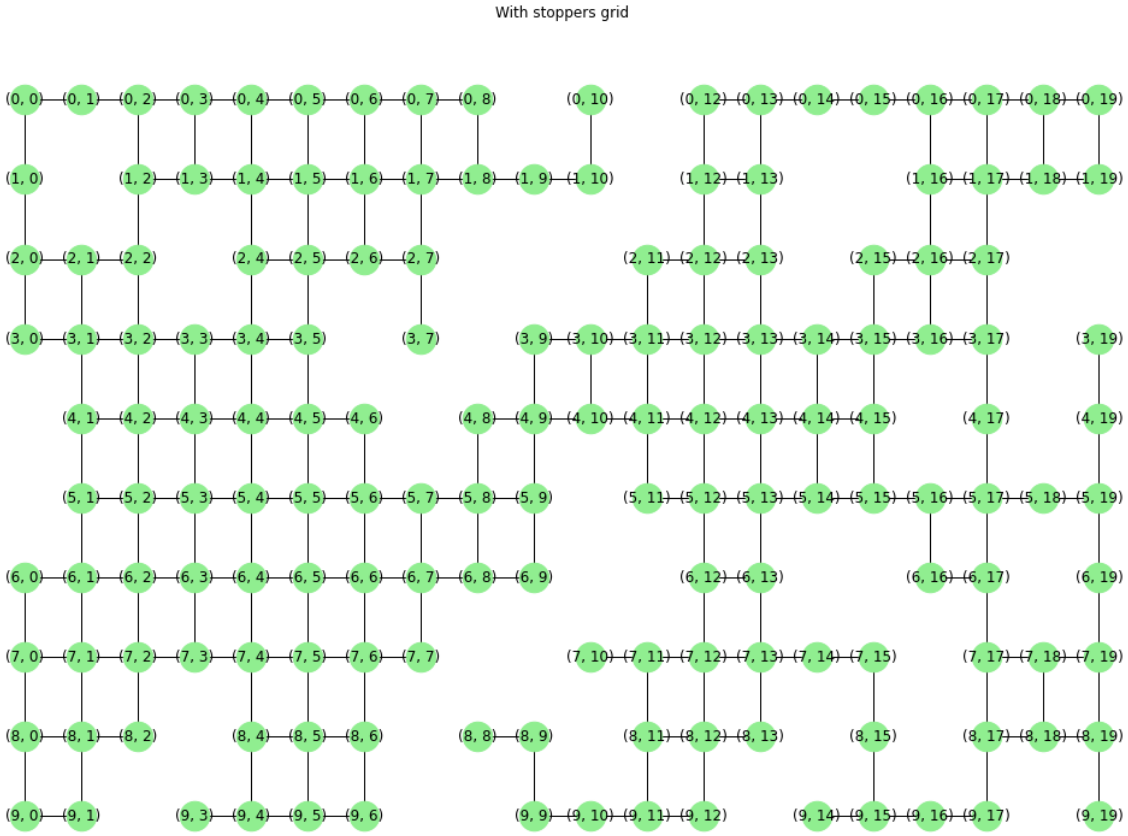


Time required for finding the paths was measured with *time.perf_counter()* function. It is seen that the time required for Bellman-Ford algorithm (1.8 ms) was **more than 5 times higher** than the time required for Dijkstra's algorithm (336 mcs).

Task 2

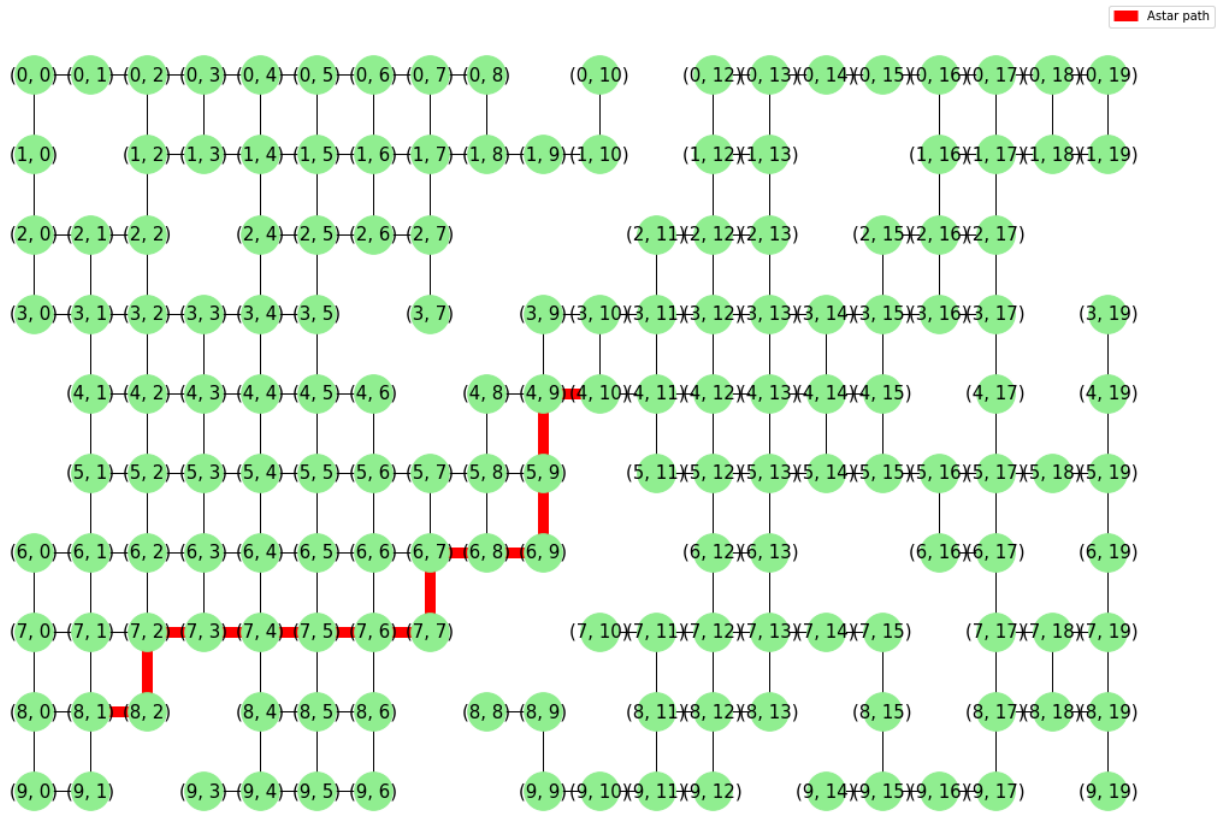
A* algorithm

The 10x20 cell grid with 40 obstacles was generated.

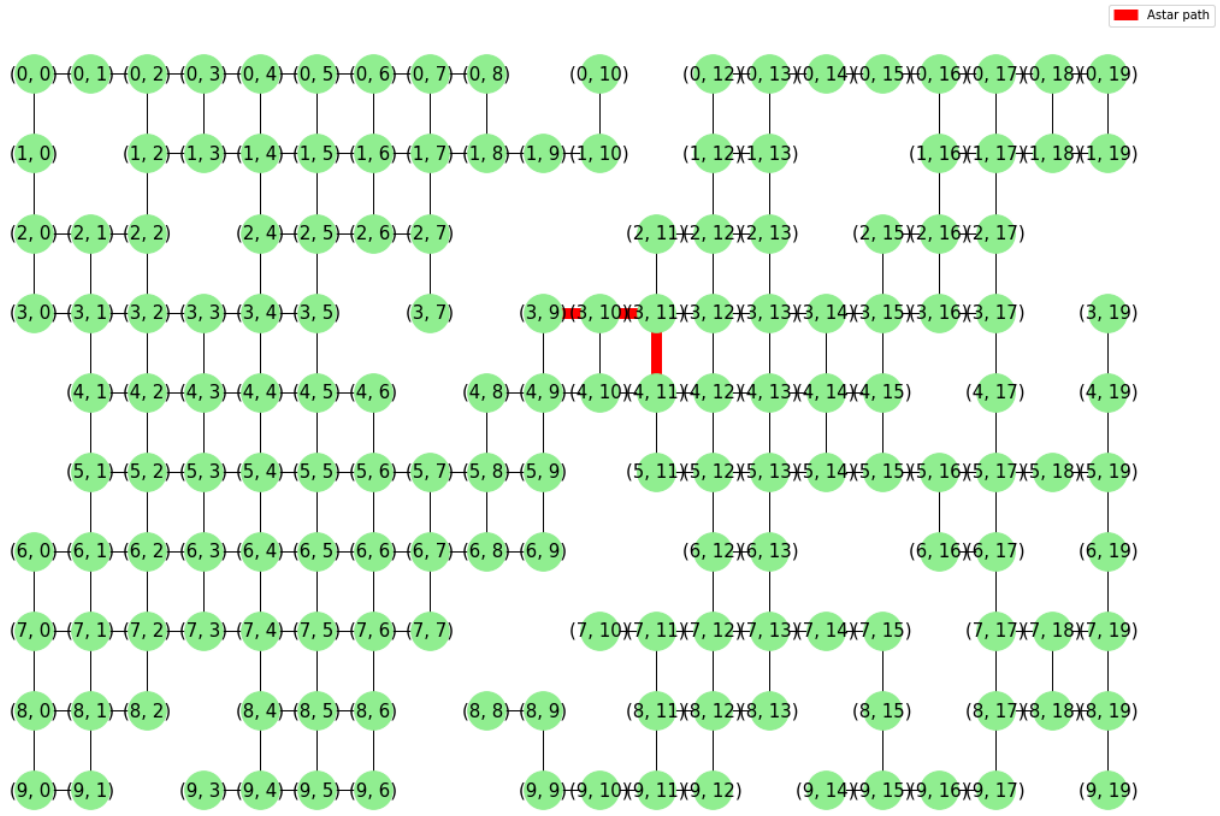


A* algorithm was used to find the shortest path from the one cell to the other cell. The experiment was doing 5 times. Each time we randomly chosen noda_start and noda_finish. Each experiment in graphs below:

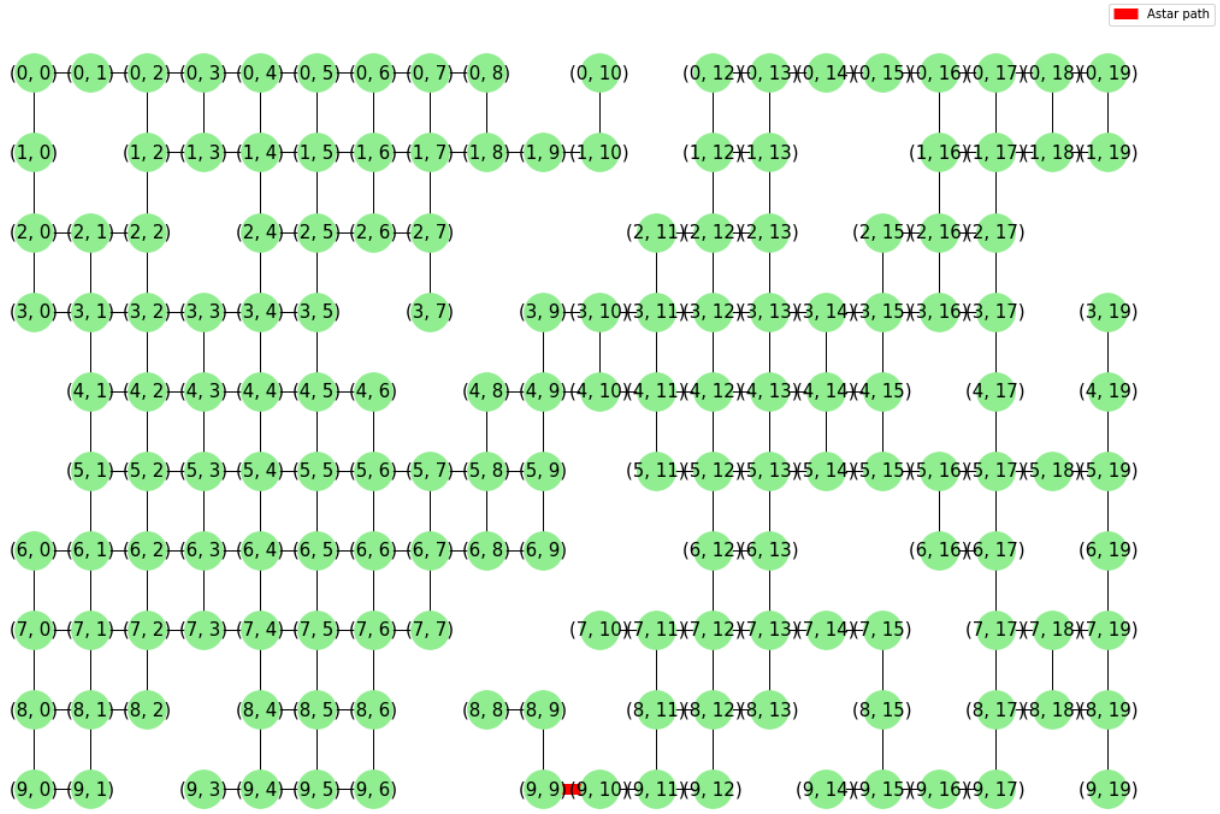
Time 803.9 ms. Distance 13. Chosen pair ((4, 10), (8, 1))



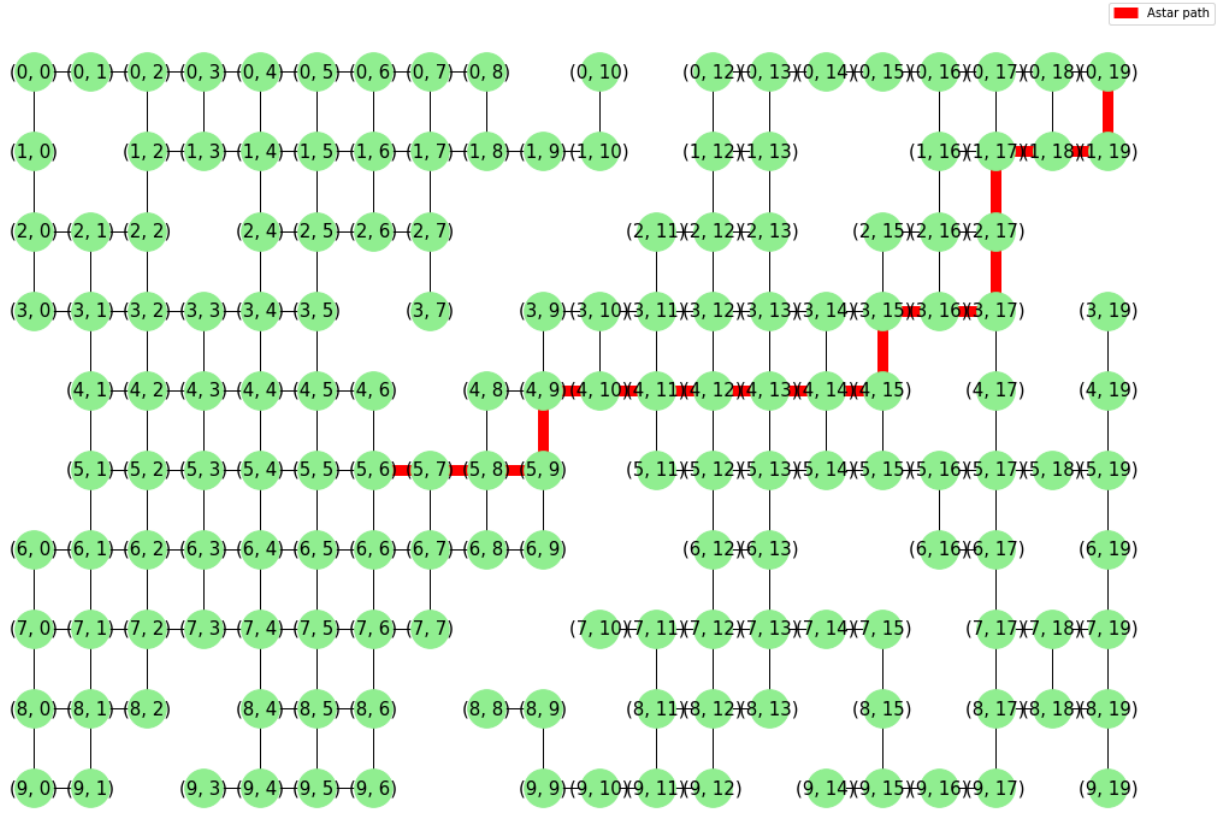
Time 104.0 ms. Distance 3. Chosen pair ((4, 11), (3, 9))

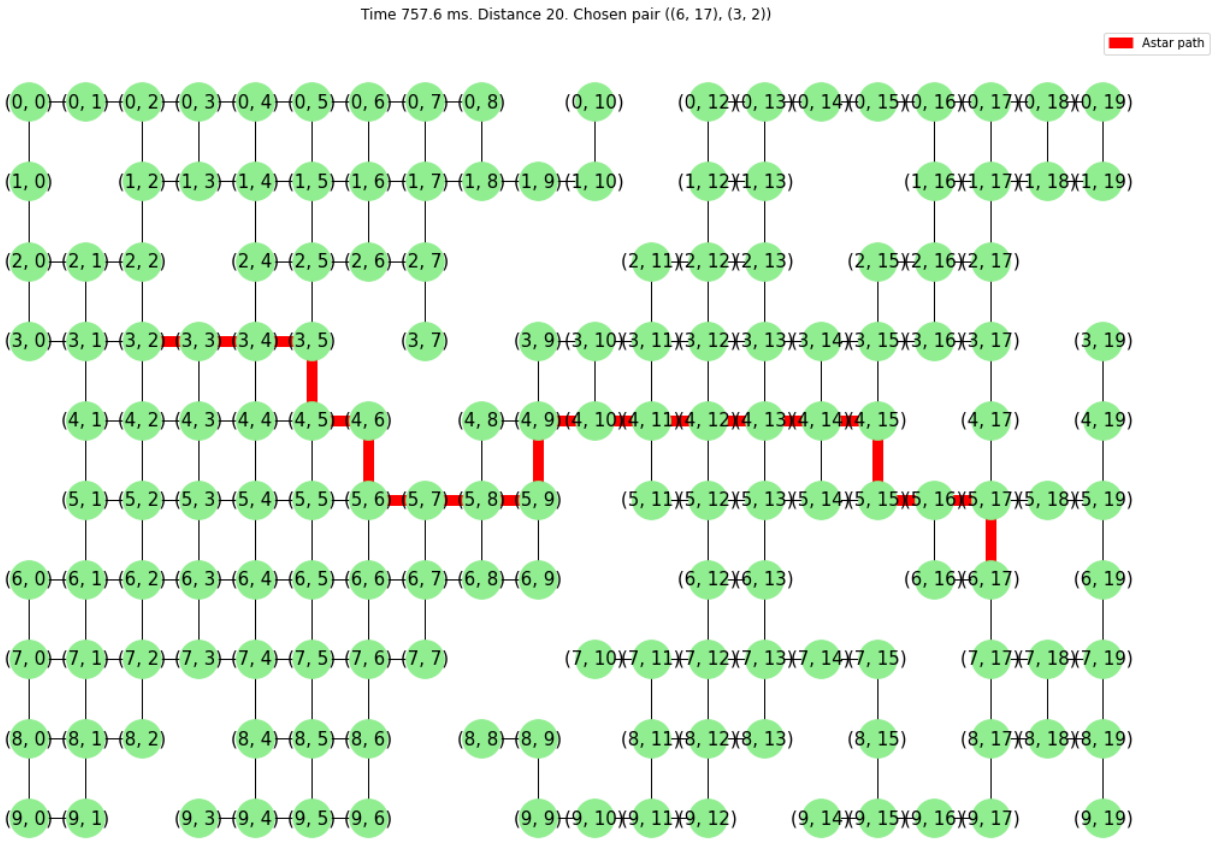


Time 29.1 ms. Distance 1. Chosen pair ((9, 10), (9, 9))

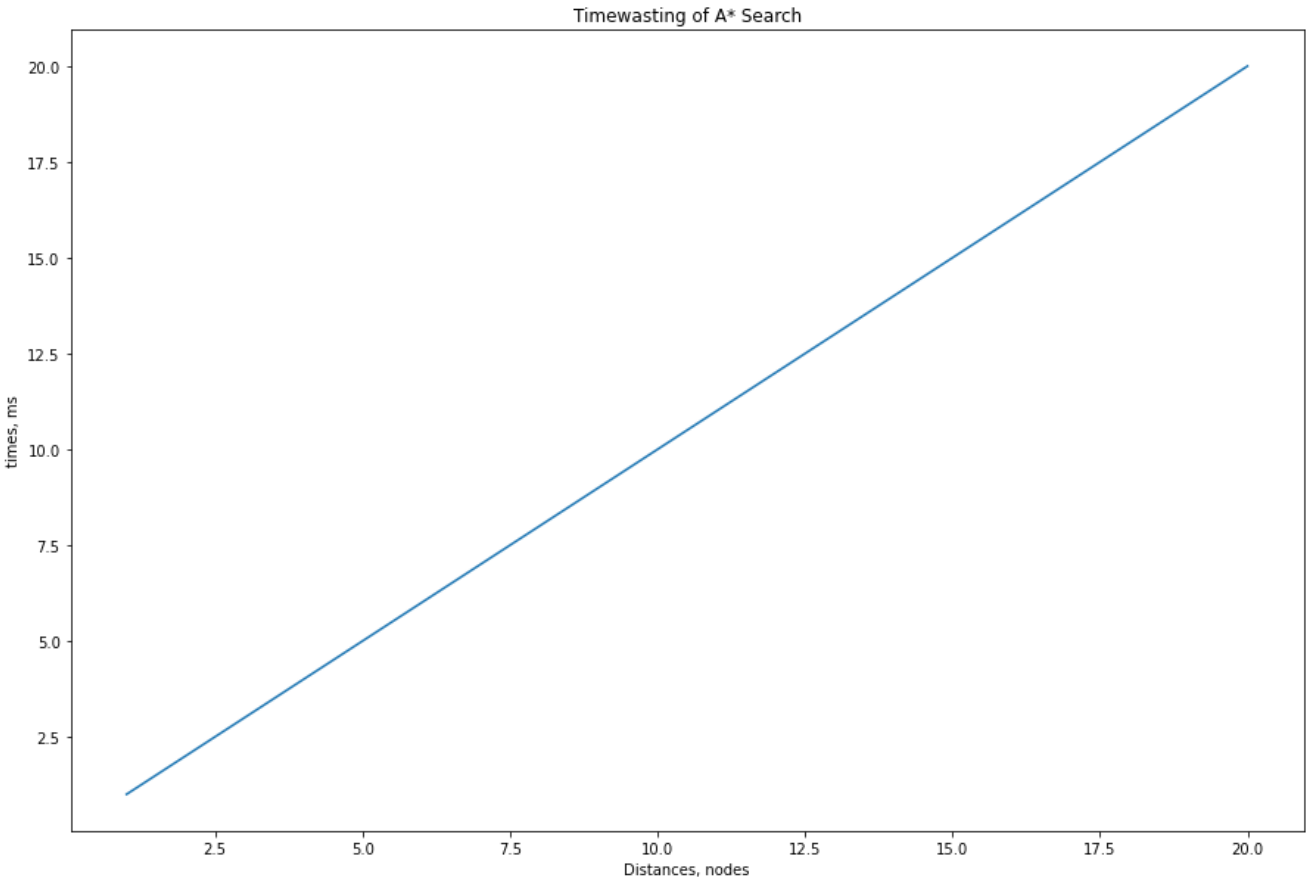


Time 483.3 ms. Distance 18. Chosen pair ((0, 19), (5, 6))





To investigate time complexity of the algorithm we plot a graph time(dist). Where dist is the number of edjes.



Conclusions

Three major algorithms for searching the shortest path in the graph were analyzed. ***Dijkstra's*** *algorithm is good in the case of graphs with positive edge weights.* ***Bellman-Ford*** *algorithm is useful for the case if the graph contains negative edge weights, but is slower if used in graph with positive edge weights. In our case Bellman-Ford was slower than Dijkstra's* ***more than by 5 times.***

A* *algorithm is an interesting Dijkstra's algorithm extension that utilized special heuristic that allows it to select the direction and find shortest paths in that direction without spending time on the other directions. In experiment* ***we have confirmed*** *the theoretical assumption about* ***A**** *algorithm estimated time complexity is O(|E|).*

Appendix

https://github.com/AAYamoldin/TrainingPrograms/blob/master/Python/ITMO_algorithms_lab/Task_6/Task6_Algorithms_on_graphs_path_search_algorithms_on_weighted_graps.ipynb