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
Dmitriy Prusskiy
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. Analyse the results obtained.

II. Generate a 10x20 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. 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, while 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 $O(|V|^2)$. Dijkstra’s algorithm is based on a greedy technique.

A* algorithm is another 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 than Dijkstra’s algorithm. Its estimated time complexity is $O(|E|)$.

Another algorithm for finding the shortest path is **Bellman-Ford algorithm**. The idea can be expressed as follows: at i-th iteration, Bellman-Ford calculates the shortest paths which have 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). Bellman-Ford uses dynamic programming techniques.

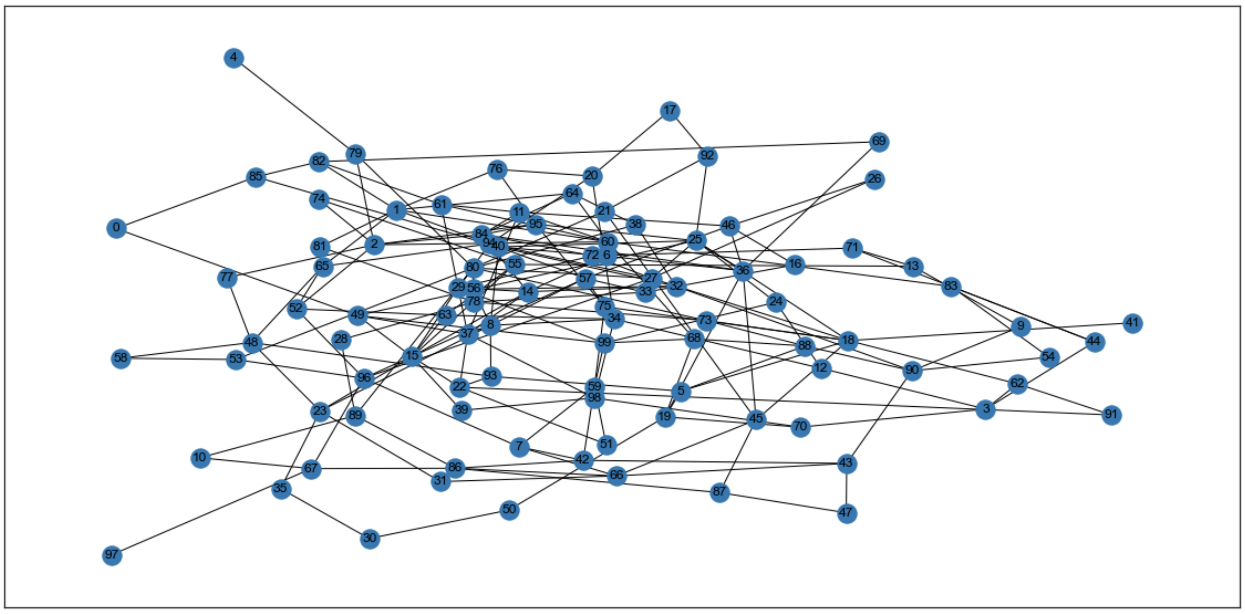
Results

The results were obtained using Python 3.8.5. The NetworkX library implementations of algorithms were used.

First part

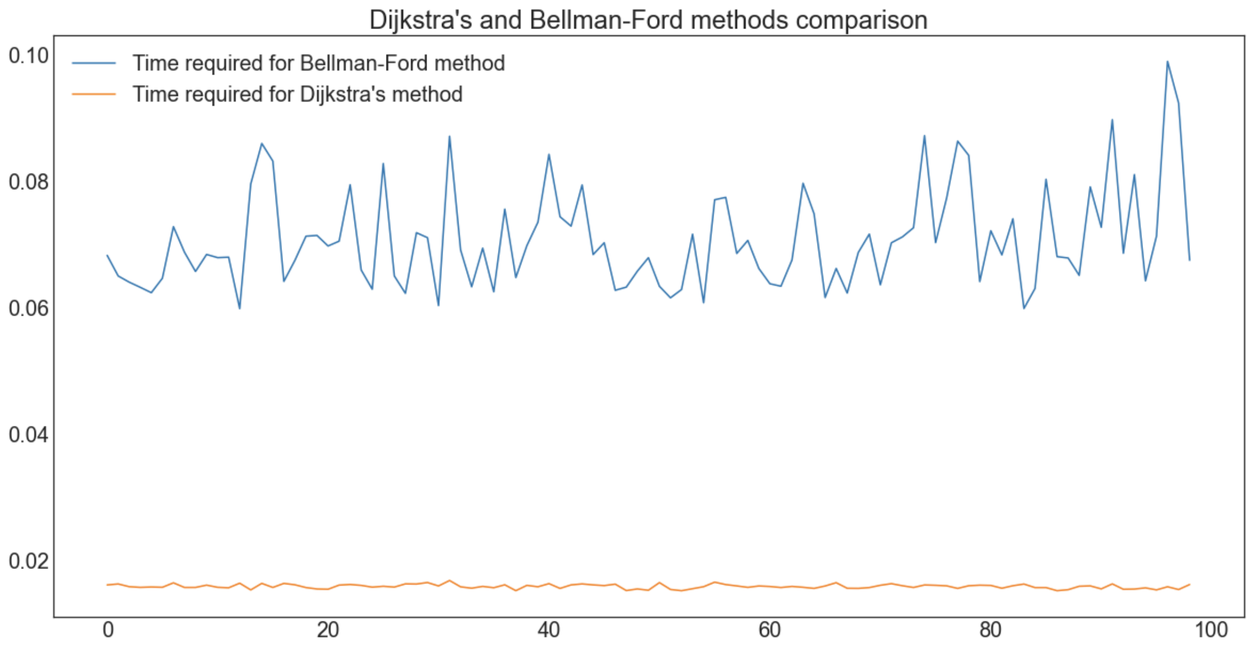
Weighted graph containing 100 nodes and 500 edges with positive weights lying in the range from 1 to 100 was created. The graph contained only one connected component.

Visualisation of the graph:



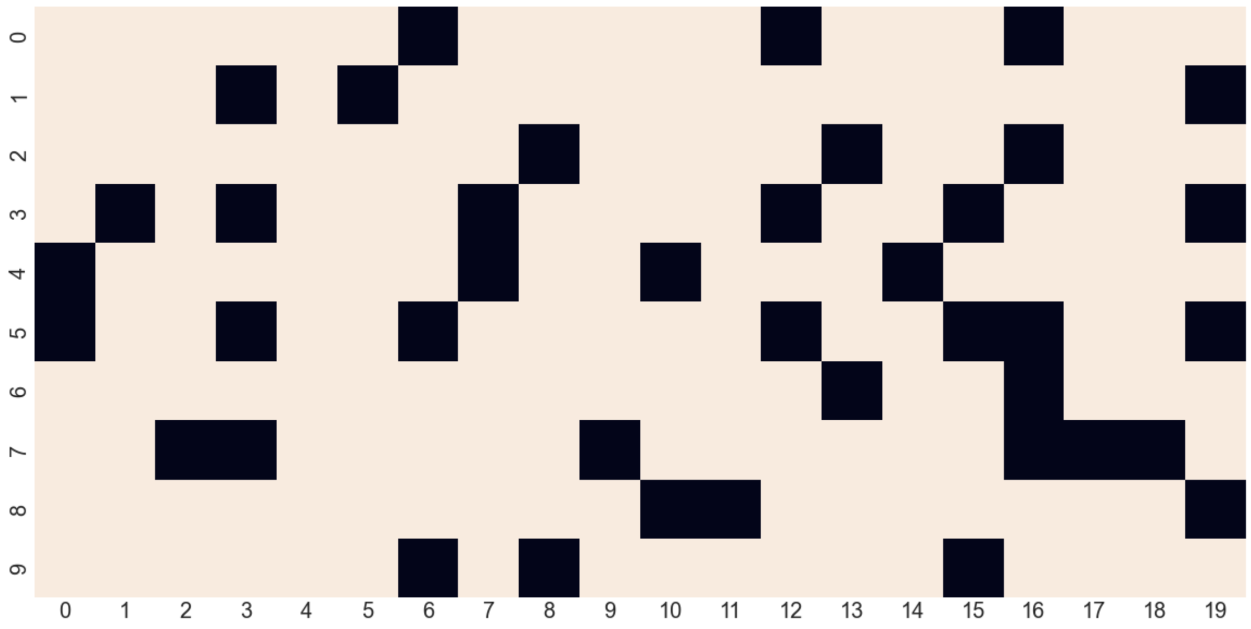
Both Dijkstra's algorithm and Bellman-Ford algorithm were used to find shortest paths between nodes in this graph. To visualize and analyze obtained results, the algorithms were executed with 100 random starting vertices.

It is seen that Bellman-Ford algorithm takes ~4.5 times more time than Dijkstra's algorithm. Interesting that Bellman-Ford algorithm always finds shortest path in sense of number of vertices in path. Dijkstra's algorithm does not always find paths with minimum number of vertices, but both algorithms find paths with equal minimum sum of weights.



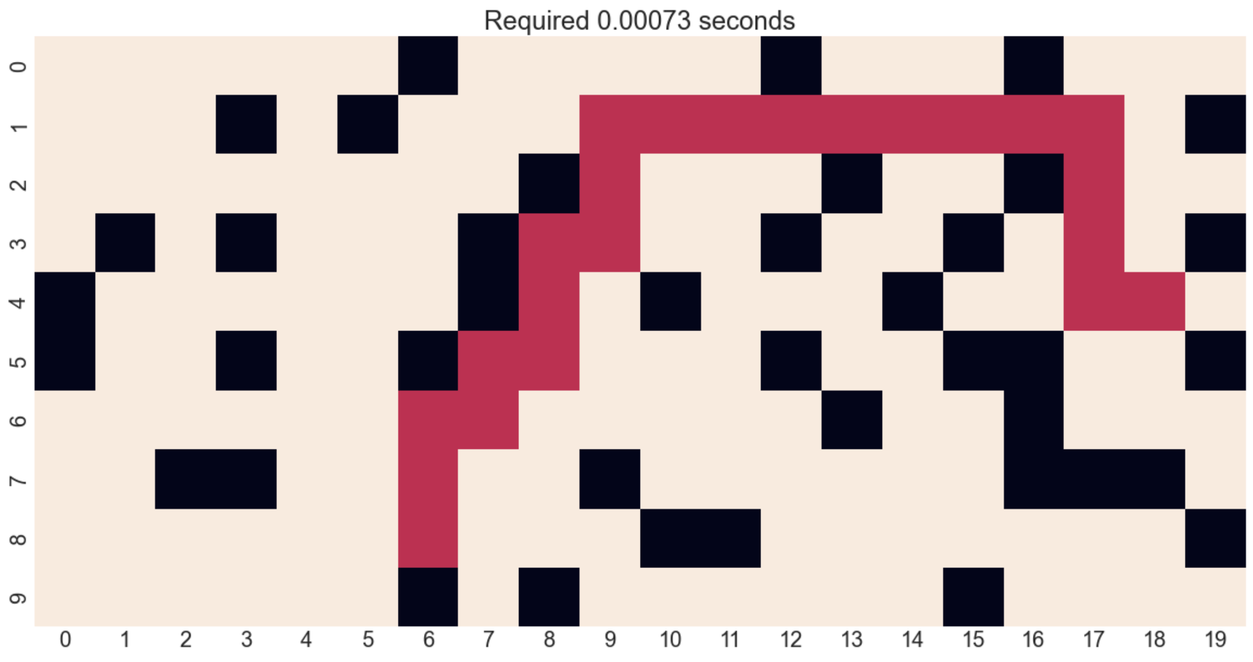
Second part

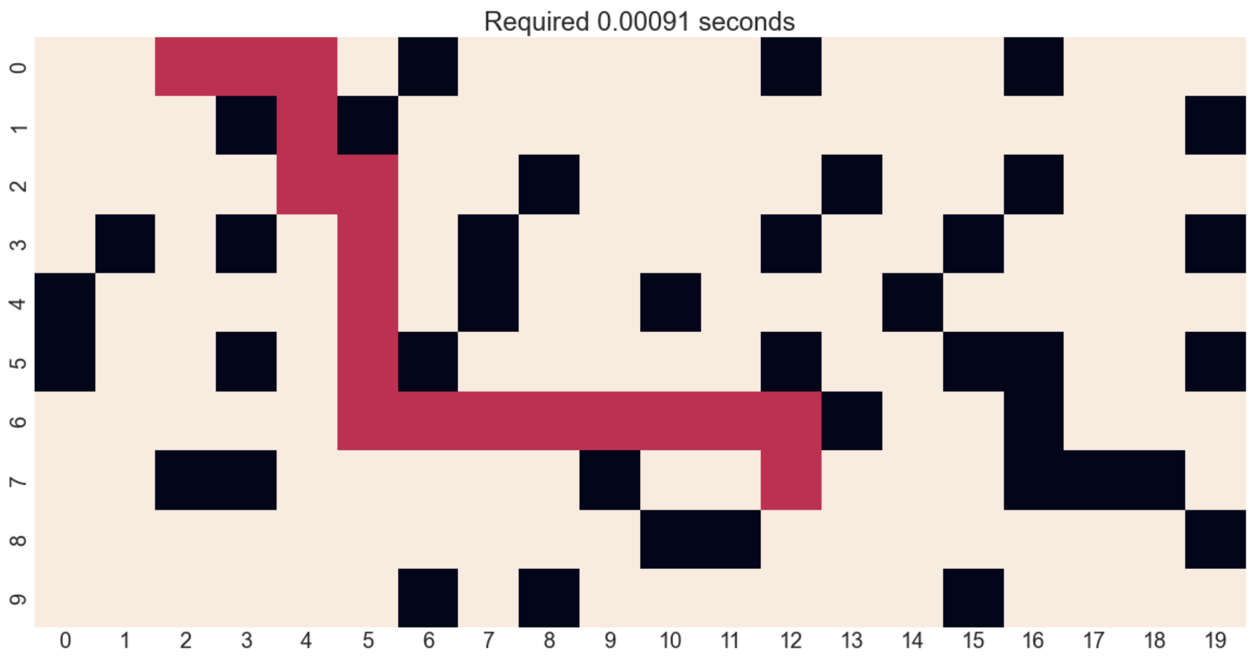
The 10x20 cell grid with 40 obstacles was generated. The seaborn.heatmap method was used to visualize the cell grid.



Algorithm A * was used to find the shortest paths between 5 different random pairs of cells.

An interesting fact is that finding the longest path does not always take the longest time. The longest path among 5 obtained paths contained 23 vertices; it was the path between cells (8, 6) and (4,18). The time required to find it is about $7 * 10^{-4}$ seconds. But there was one path required $9 * 10^{-4}$ seconds and contained 19 vertices. It happens since the algorithm probably spent some time when it chose the wrong direction. Below are visualizations of shortest paths on the cell grid.





Conclusions

The goal of this study was to use path search algorithms on weighted graphs and compare obtained results. Analysis of the obtained results shows that Dijkstra's algorithm works much faster then the Bellman-Ford algorithm. But the Bellman-Ford algorithm's advantage is finding the shortest path not only in the sense of sum of weights but also in the number of vertices. 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.

Appendix

Source code can be found at https://github.com/T1MAX/itmo_algorithms/tree/main/task_6