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

*Vdovkina Sophia*

*Syrchenko Arina*

*Academic group: J4133c*

Accepted by

Dr Petr Chunaev

St. Petersburg

2022

**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 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. Analyze the results obtained.

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

**Brief theoretical part**

*Dijkstra's algorithm*

Dijkstra's algorithm is used for finding the shortest paths between nodes in a graph. It generates a shortest path tree (SPT) with the source, which in this case is a root, while it has two sets, one set contains vertices included in SPT, other set contains 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’s time complexity ranges from O(|V| log|V|) to O(|V|2). Dijkstra’s algorithm is only applicable when all weights are positive as the weights of the edges are added to find the shortest path, so if there are negative weights, the algorithm will loop over them, as it will iteratively find shorter and shorter path. However, Bellman-Ford algorithm can be used in such cases.

*A\* algorithm*

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 source node, it aims to find a path to the goal node with the smallest cost. It does this by maintaining a tree of paths originating from the start node and extending those paths one edge at a time until the termination criterion is satisfied.

A\* algorithm has 3 parameters:

* *g* is the cost of moving from the initial cell to the current cell. It is the sum of all the cells that have been visited.
* *h* is also known as the heuristic value, it is the estimated cost of the path from the current cell to the final cell.
* *f* is the sum of g and h. So, 𝑓 = 𝑔 + ℎ. The way that the algorithm makes its decisions is by taking the f-value into account. The algorithm selects the smallest f-valued cell and moves to that cell. This process continues until the algorithm reaches its goal cell.

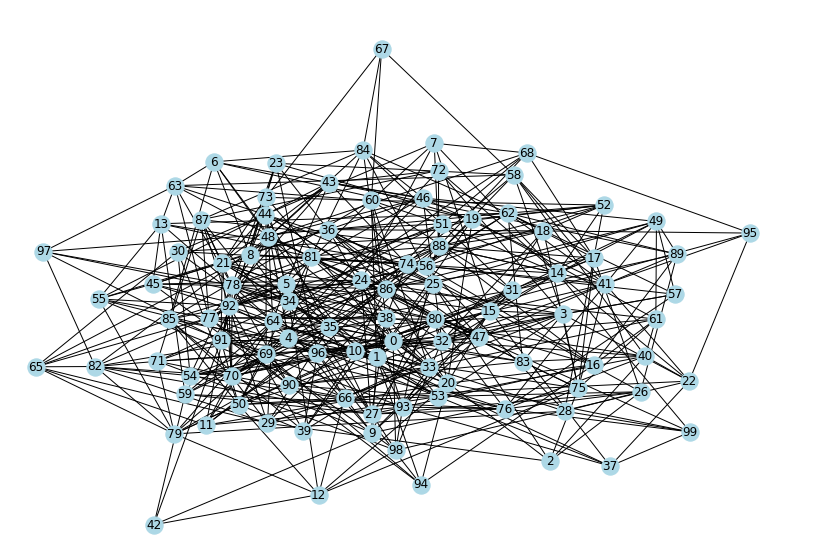
A\* exploits heuristic technique which helps it to work faster that Dijkstra’s algorithm. Its time complexity is O(|E|).

*Bellman-Ford algorithm*

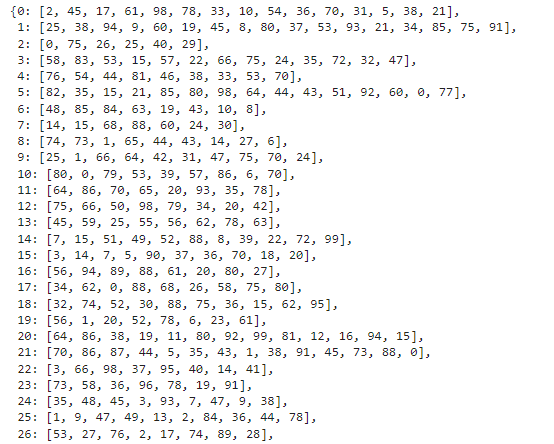
Bellman-Ford algorithm is used to find minimum distance from the source vertex to any other vertex. Bellman-Ford algorithm finds the distance in a bottom-up manner. At first, it finds those distances which have only one edge in the path. After that it increases the path’s length to find all possible solutions. So, at *i*-th iteration, Bellman-Ford calculates the shortest paths that has at most *i* edges. As there is maximum of edges in any simple path, . Assuming that there is no negative cycle and 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 a negative cycle. The time complexity of this algorithm is O(|V||E|) and O(|V|3) in the worst-case scenario.

**Results**

1. Weighted graph with 100 nodes and 500 edges was created, using generated random adjacency matrix. This graph can be seen in the picture 1 and a slice of its adjacency list is in the picture 2.



Picture 1 – graph



Picture 2 – adjacency list

Using Dijkstra’s and Bellman-Ford algorithms the shortest path from start node (6) to target node (54), the results are shown on pictures 3 and 4 respectively.



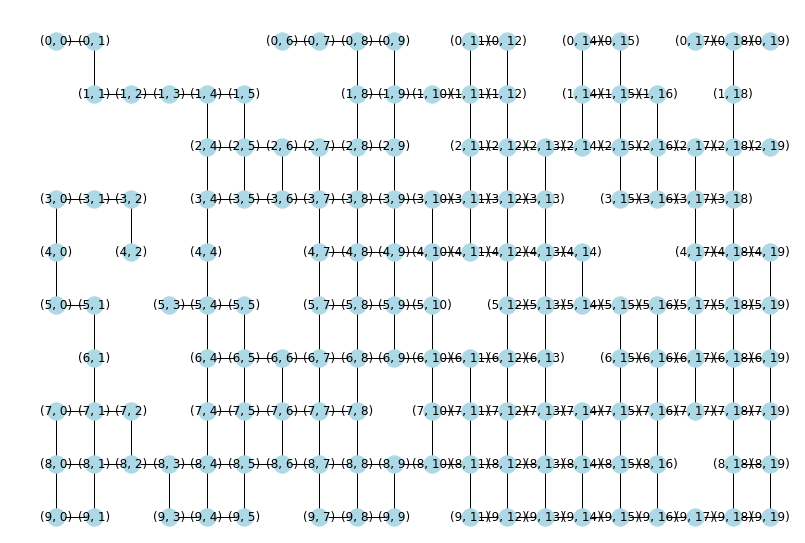
Picture 3 – result of Dijkstra's algorithm



Picture 4 – result of Bellman-Ford algorithm

Both algorithms found the same path. Dijkstra’s algorithm has completed the task faster than Bellman-Ford algorithm. Dijkstra’s algorithm is a greedy algorithm and its time complexity is much less than for the Bellman-Ford algorithm, that is why Dijkstra’s algorithm is faster.

1. A cell grid 10x20 with 40 obstacle cells was generated, it is shown on the picture 5.



Picture 5 – a cell grid

1. The data structure of a graph is a set of vertices that have data and are connected to other vertices.

**Conclusion**

Undirected random graph was generated to fins the shortest path between a random starting vertex and target vertex was found by Dijkstra's and Bellman-Ford algorithms. Dijkstra's algorithm has less time complexity. In the second part of the task шt was necessary to apply the A\* algorithm in the context of an obstacle grid represented as a weighted graph.

**Appendix**

Source code is available on <https://github.com/sophia-vdovkina/Analysis-and-development-of-algorithms/blob/main/Task%206/Task6.ipynb>