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
Mikhail Grigoryev (370852)
Semenova Valeria (370061)
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

Task 1. 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.

Task 2. 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. Analyse the results obtained.

Brief theoretical part

Path search algorithms on graphs are used to find shortest paths between two vertices. In case of unweighted graphs, those paths consist of the least number of edges. In this practical work, however, graphs are weighted which means that numbers (costs) are assigned to the edges. Here, the shortest path between two vertices is defined as the path with the least sum of edge costs.

Algorithms used in this practical work (implemented from NetworkX library):

1. Dijkstra's algorithm – a search algorithm for weighted graphs with positive costs only. The algorithm uses tentative distances of vertices of the graph (lengths of shortest paths between the starting node and the current one). At each step, all neighbors of the current vertex are taken into account. Their tentative distances are measured, and, if better than previous values, are updated (paths are logged). When the current node is the end node, tentative distance is updated and the shortest logged path is taken as the shortest one.
2. Bellman-Ford algorithm (although slower than Dijkstra's) can be used even on weighted graphs with negative costs. The algorithm initializes all nodes with infinite distances and then for all edges updates the distances to lower values if taking that edge lowers the distance to the destination. As the longest possible path (with no cycles) is $|V| - 1$ edges long, the edges must be scanned that many times to make sure that all of the paths to the nodes were optimized. This algorithm can detect negative loops (loops with costs adding up to a negative value) as in those cases no shortest path exists.
3. A* algorithm is complete and extremely fast but requires vast amounts of additional memory.It utilizes heuristics in a sense that at every iteration it estimates which paths have to be extended (based on the cost of the path to the current node plus heuristic function that estimates the cheapest path from the node to the goal). The algorithm terminates when the extended path is the one from the start to the goal or when there are no paths to be extended.

Results

Task 1. A random adjacency matrix for a simple undirected weighted graph of 100 vertices and 500 edges was generated. The edge costs were chosen randomly as integers from 1 to 20. The starting vertex from the graph was chosen randomly, then Dijkstra's and Bellman-Ford algorithms were applied to find the shortest (in terms of costs) path between the starting and all other vertices. The computation was repeated 9 more times and the runtimes were measured and averaged.

Presented below is an example path between the starting node (50) and the current target (61). The path consists of 6 edges and 7 vertices:

$$50 \xrightarrow{3} 64 \xrightarrow{1} 33 \xrightarrow{1} 84 \xrightarrow{1} 25 \xrightarrow{1} 52 \xrightarrow{2} 61$$

however, their costs add up to only 9.

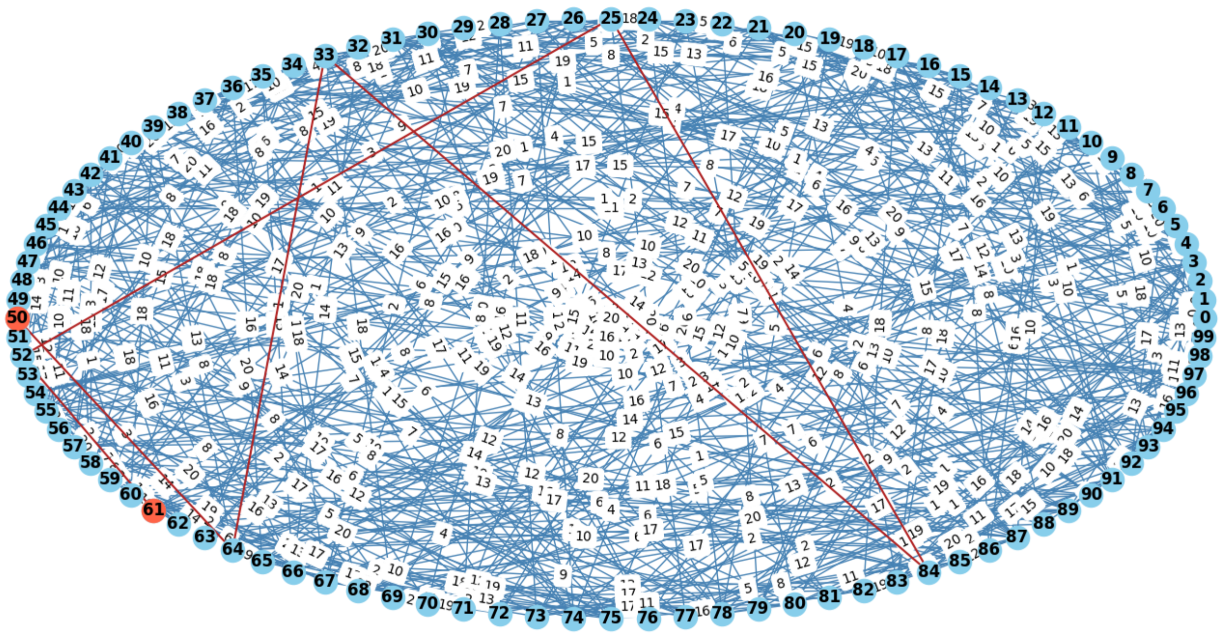

Figure 1: Example of a shortest path between two random vertices. Here the path consists of many edges, however, the costs of each are low.

In the task, shortest paths were computed from a randomly selected vertex to all other vertices in the graph by means of Dijkstra's and Bellman-Ford algorithms. Results for the starting vertex 50:

Dijkstra:	0.0162 s
Bellman-Ford:	0.0607 s

The results prove that Dijkstra's algorithm significantly faster than Bellman-Ford. The latter has an added benefit of working on weighted graphs with negative costs which was not covered in the present work.

Task 2. The random 10×20 grid with 40 obstacle cells was represented by a graph, where each cell is a node and edges exist only between adjacent cells in the grid. Obstacle cells were represented by the lack of vertex in its place. Two random non-obstacle cells were chosen and the shortest path between them was computed via A* algorithm. Such computations for different random pairs were executed five times. The computation runtimes were measured and presented on the graph below.

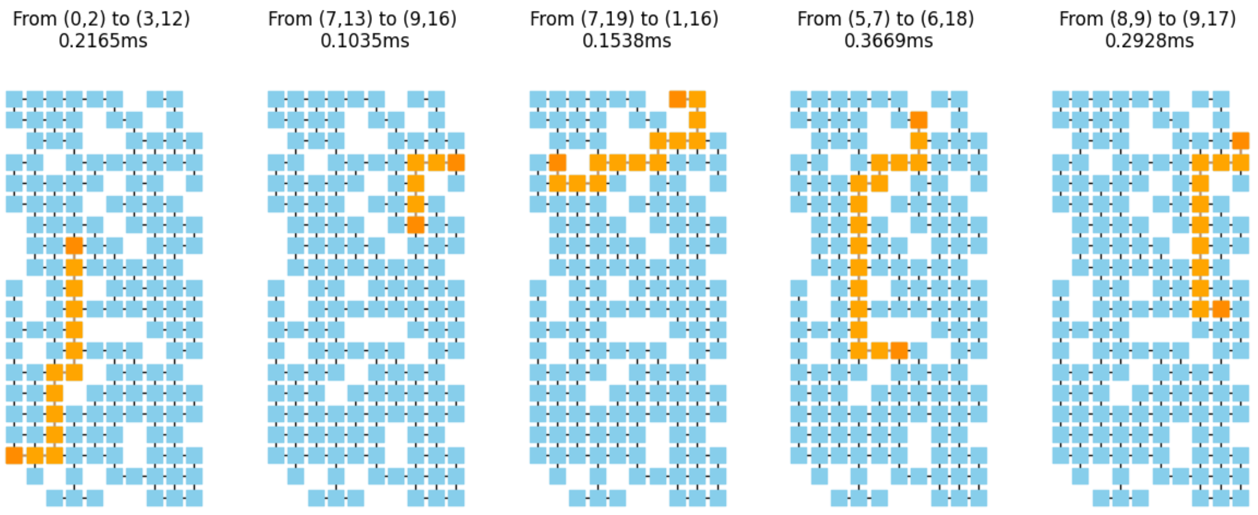

Figure 2: Five shortest paths (A*) in a random 10×20 grid with 40 obstacles with execution times.

According to the obtained data, the execution time is correlated with the length of the shortest path. The further the cells are located, the longer the algorithm runs in search of it.

Conclusions

Path search algorithms for weighted graphs such as Dijkstra's, Bellman-Ford and A* algorithms were applied for searching the shortest path between random vertices in random graphs, both weighted and unweighted. The former two algorithms were compared in performance.

Appendix

GitHub link (task 1): https://github.com/Dormant512/itmo_lab_listings/blob/main/lab6_1.py.

GitHub link (task 2): https://github.com/Dormant512/itmo_lab_listings/blob/main/lab6_2.py.