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
Alexandra Matveeva
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)**.**

**Problems**

**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 nonobstacle 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.** This algorithm makes a tree of the shortest path from the starting node, the source, to all other nodes (points) in the graph.

Dijkstra's algorithm makes use of weights of the edges for finding the path that minimizes the total distance (weight) among the source node and all other nodes. This algorithm is also known as the single-source shortest path algorithm.

Dijkstra’s algorithm is the iterative algorithmic process to provide us with the shortest path from one specific starting node to all other nodes of a graph. It is different from the minimum spanning tree as the shortest distance among two vertices might not involve all the vertices of the graph.

It is important to note that Dijkstra’s algorithm is only applicable when all weights are positive because, during the execution, the weights of the edges are added to find the shortest path.

And therefore if any of the weights are introduced to be negative on the edges of the graph, the algorithm would never work properly. However, some algorithms like the **Bellman-Ford Algorithm** can be used in such cases.

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 increase the path length to find all possible solutions.

**A*** is like Dijkstra’s Algorithm in that it can be used to find a shortest path. A* is like Greedy Best-First-Search in that it can use a heuristic to guide itself. In the simple case, it is as fast as Greedy Best-First-Search.

A* algorithm has 3 parameters:

- **g**: the cost of moving from the initial cell to the current cell. Basically, it is the sum of all the cells that have been visited since leaving the first cell.

- **h**: also known as the heuristic value, it is the estimated cost of moving from the current cell to the final cell. The actual cost cannot be calculated until the final cell is reached. Hence, h is the estimated cost. We must make sure that there is never an over estimation of the cost.

- **f**: it 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.

## Results

**I.** Adjacency matrix:

```
[[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0. 18.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0. 33.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. 20.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. 53.]
 [ 0. 18.  0.  0.  0. 33.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. 20.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  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.]]
```

Fig. 1 - Adjacency matrix

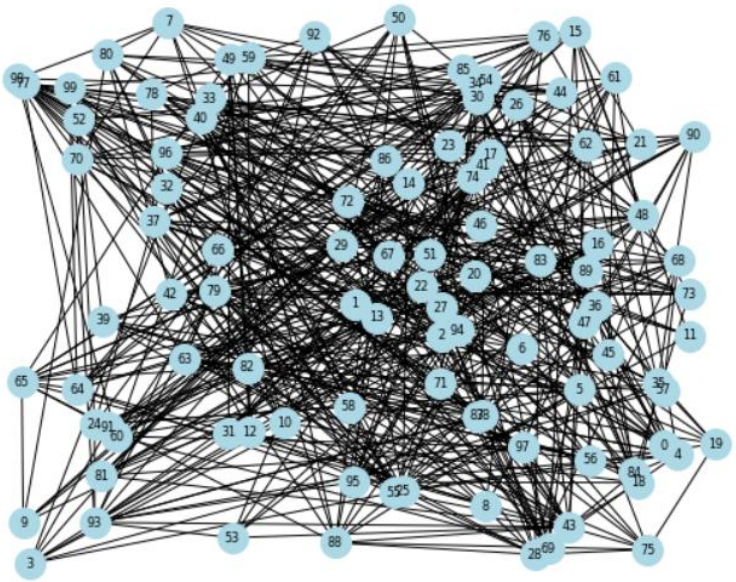


Fig. 2 – Graph visualization

The shortest path between start node (31) and target node (60) was found by methods:

- Dijkstra's algorithm (Fig. 3)

- Bellman-Ford algorithm (Fig. 4)

```
Dijkstra's algorithm:  [31, 93, 83, 98, 60]
Time:  0.0006903019997480442
```

Fig. 3 - Dijkstra's algorithm results

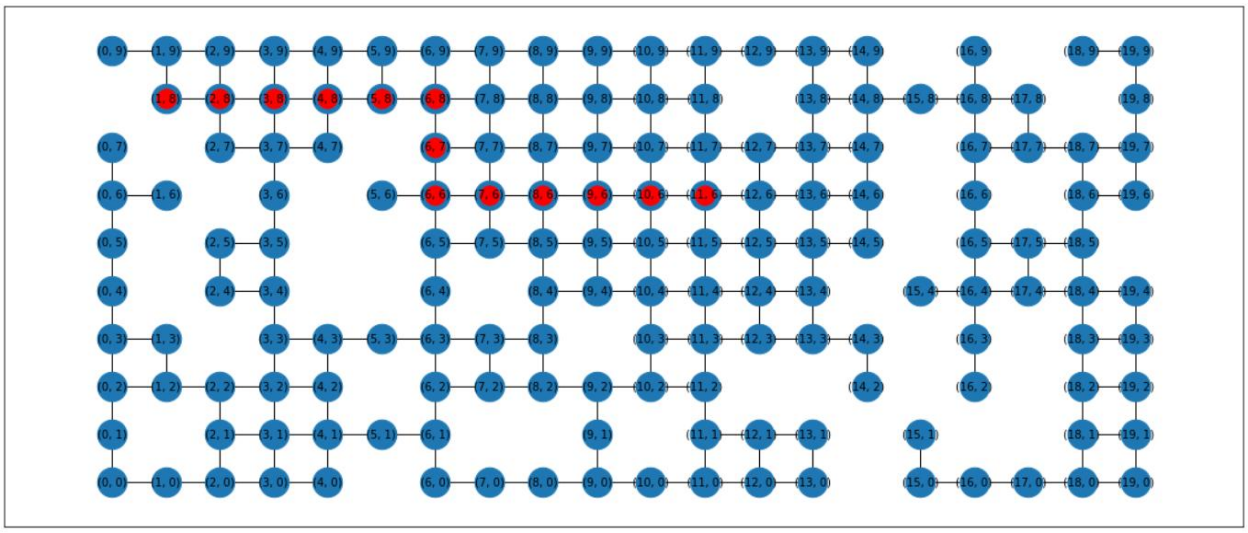
```
Bellman-Ford algorithm:  [31, 93, 83, 98, 60]
Time:  0.0044490176999715915
```

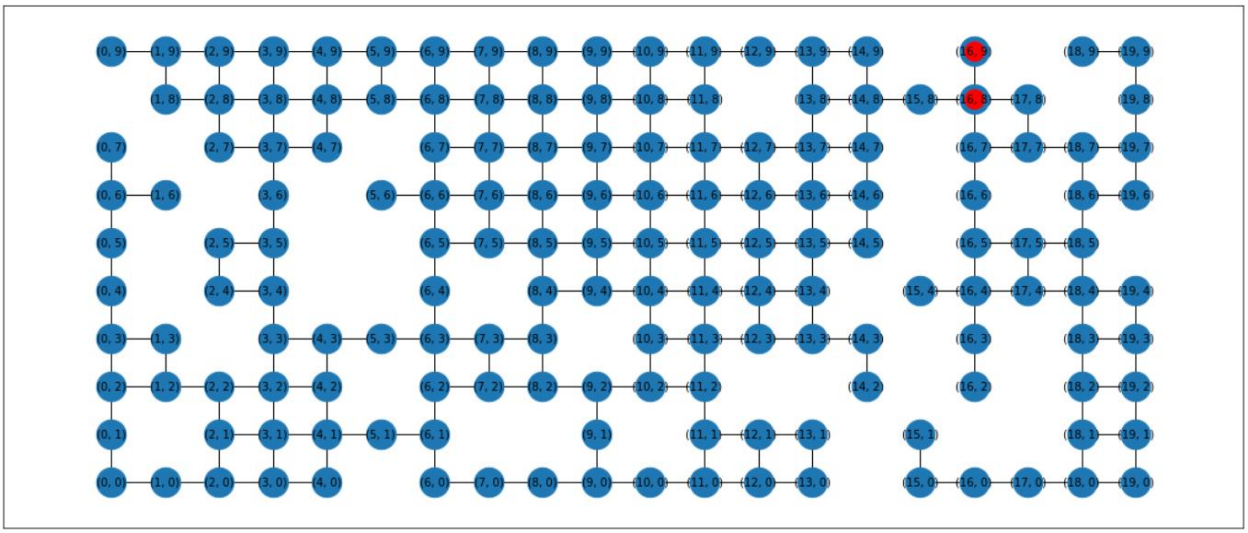
Fig. 4 - Bellman-Ford algorithm

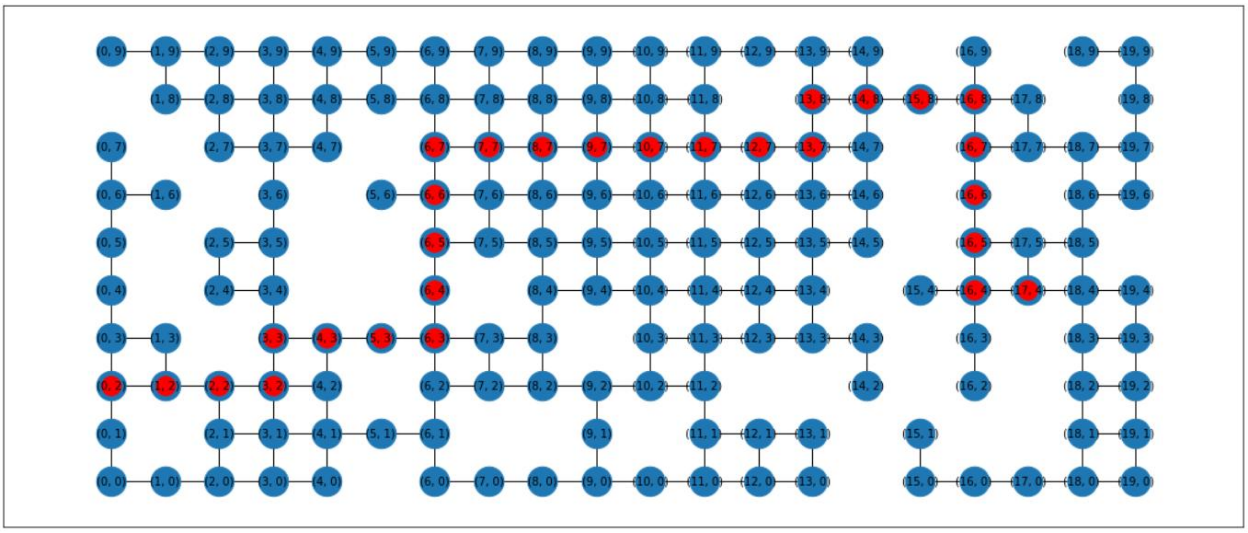
Both algorithms found the same paths, and Dijkstra's algorithm has completed the task faster than Bellman-Ford algorithm.

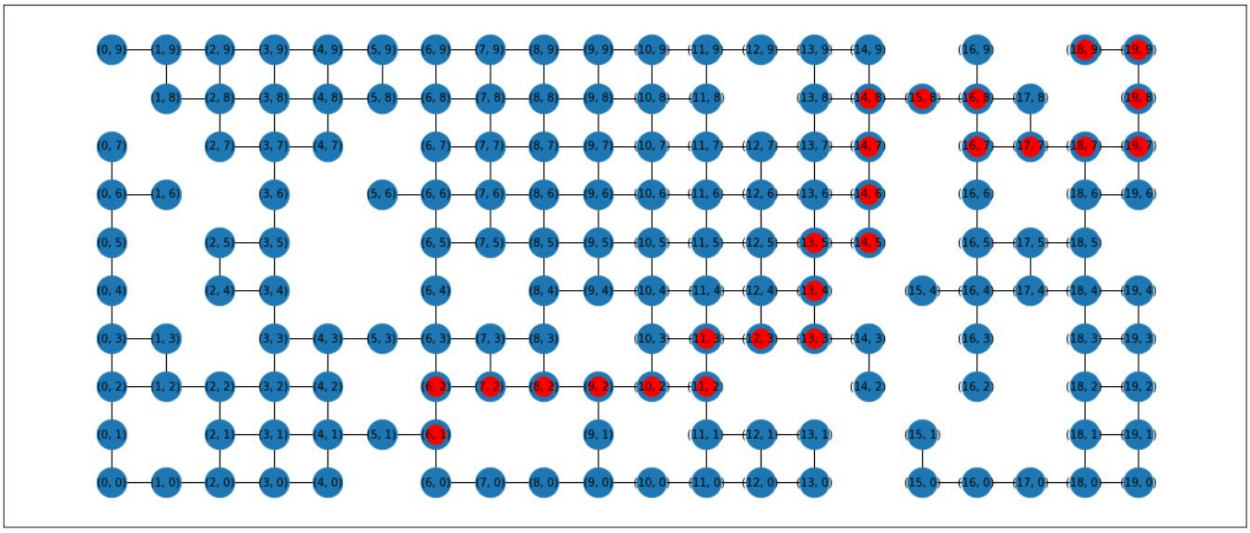
Dijkstra's algorithm is a Greedy algorithm and time complexity is $O(V + E\ LogV)$ (with the use of Fibonacci heap). Dijkstra doesn't work for Graphs with negative weight cycle, Bellman-Ford works for such graphs. Bellman-Ford is also simpler than Dijkstra and suites well for distributed systems. But time complexity of Bellman-Ford is $O(VE)$, which is more than Dijkstra.

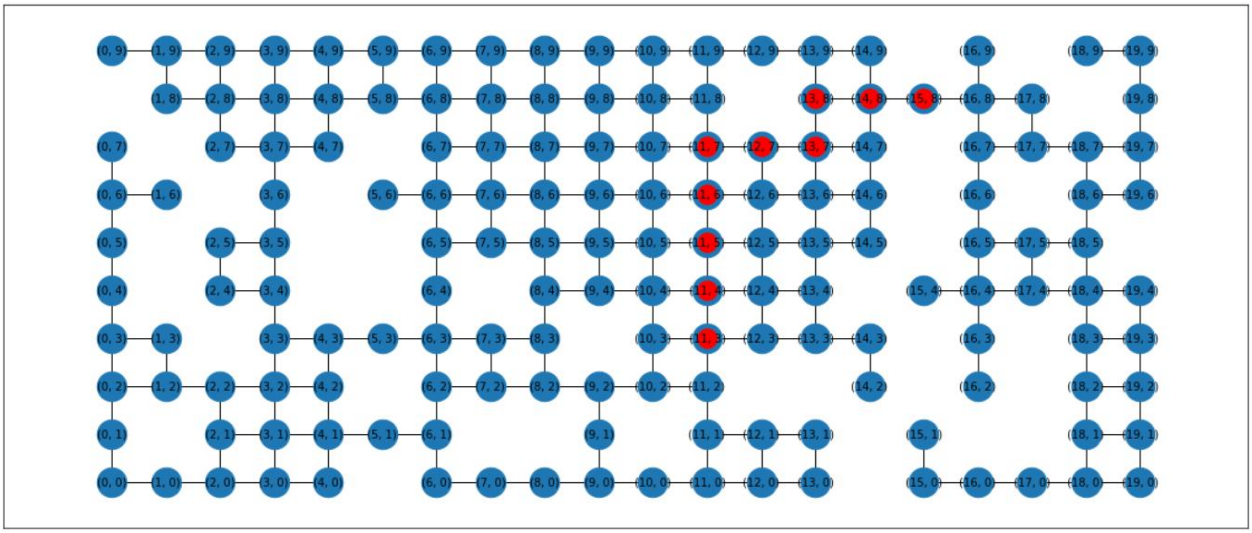
**II.** In the second part of task grapt-labyrinth was generated. Shortest path between two nodes was found (with method astar.astar_path).











**III.**

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

networkx.algorithms.shortest_paths.weighted.dijkstra_path(G, source, target, weight='weight') – returns the shortest weighted path from source to target in G. Uses Dijkstra’s Method to compute the shortest weighted path between two nodes in a graph.

networkx.algorithms.shortest_paths.weighted.bellman_ford_path(G, source, target, weight='weight') – returns the shortest path from source to target in a weighted graph G.

## Conclusion

During the execution of laboratory work, simple undirected unweighted random graph was generated. 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 ıt was necessary to apply the A* algorithm in the context of an obstacle grid represented as a weighted graph.

## Appendix

GitHub Link: https://github.com/alex-mat-s/Algorithms/blob/main/Lab6.ipynb