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
*Abdurakhimov Muslimbek*
*J4133c*
Accepted by
Dr Petr Chunaev

St. Petersburg
2023

**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 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**

Path search algorithms play a pivotal role in the domain of graph theory and network analysis. They provide the means to find the optimal path or route in various applications, including navigation systems, network routing, and optimization problems. In this exploration, we delve into three prominent path search algorithms on weighted graphs: Dijkstra's algorithm, A* algorithm, and the Bellman-Ford algorithm. Each of these algorithms is designed to address specific scenarios and graph characteristics, making them essential tools in solving a wide array of real-world problems.

Dijkstra's Algorithm:

Dijkstra's algorithm is a widely-used path search algorithm that finds the shortest path between two nodes in a weighted graph. It's specifically designed for graphs with non-negative edge weights. The key idea is to maintain a set of visited nodes and a priority queue to explore the neighboring nodes efficiently. The algorithm proceeds by iteratively selecting the unvisited node with the shortest distance from the source and then relaxing its adjacent edges. Dijkstra's algorithm guarantees finding the shortest path if all edge weights are non-negative.

A Algorithm: *

A* (pronounced "A-star") is another path search algorithm used for finding the shortest path in a weighted graph. What sets A* apart is its ability to incorporate heuristics, making it more versatile. A* evaluates nodes based on a combination of two costs: the actual cost from the start node and a heuristic estimate to the goal node. This helps guide the search towards the goal efficiently. A* works optimally when the heuristic is admissible, meaning it never overestimates the true cost.

Bellman-Ford Algorithm:

The Bellman-Ford algorithm addresses graphs with potentially negative edge weights, making it more robust than Dijkstra's algorithm in certain scenarios. It works by iteratively relaxing edges, repeatedly updating the shortest path estimates. The algorithm handles graphs with negative edge weights and can detect negative weight cycles. However, it may not be as efficient as Dijkstra's algorithm when all edge weights are non-negative.

These path search algorithms are fundamental in solving a variety of real-world problems, such as finding the shortest route in maps, optimizing network routing, or even planning robotic movements. Understanding the principles behind these algorithms and their use cases is crucial for solving graph-related problems effectively.

**Results**

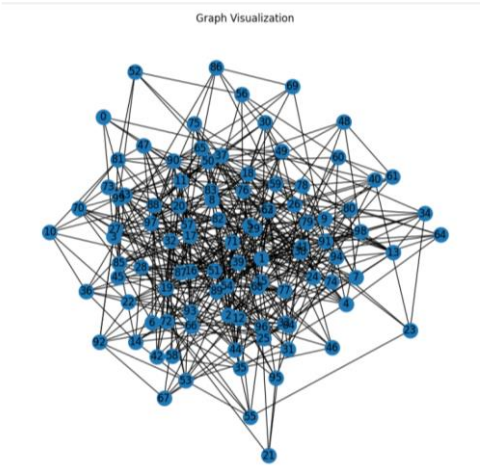
**PART I.**

In this experiment, we generated a random adjacency matrix representing a simple undirected weighted graph with 100 vertices and 500 edges. The weights were assigned as random positive integer values. We then employed two path search algorithms, Dijkstra's and the Bellman-Ford algorithms, to find the shortest paths from a randomly chosen starting vertex to all other vertices in the graph. The time taken for path search was measured, and the experiment was repeated ten times to calculate the average time required for each algorithm.

```
[ 0  0  0 39  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  8  0  0 82  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 67  0  0  0  0  0  0 74  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 61  0  0  0  0  0  0  0
  0  0  0  0 84  0  0  0 88  0  0  0  0  0  0  0  0  0 81 77  0  0  0  0  0
  0 58  0  0  0  0  0  0  0  0  0  0  0  0  0  0 31  0  0  0  0  0  0  0  0
  0  0 68  0  0  0  0  0  0  0  0 46  0  0  0  0  0  0  0  0  0  0 88  0
  0  0 52  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 68  0  0  0  0  0  0  0  0  0
  0 35  0  0  0 80  0  0  0  0  0  0  0  0  0 20 14  0  0  0  0 96  0  0  0
  0  0  0 23  0  0  0 86  0 56 44  0  0  0  0  0  0  0  0  0  0  0  0  0  8
  0  0  0  0  0  0  0  0  0  0 69 23 42  0  0  0  0  0  0 15  0  0  0  0  0
  0  0  0  0]
```

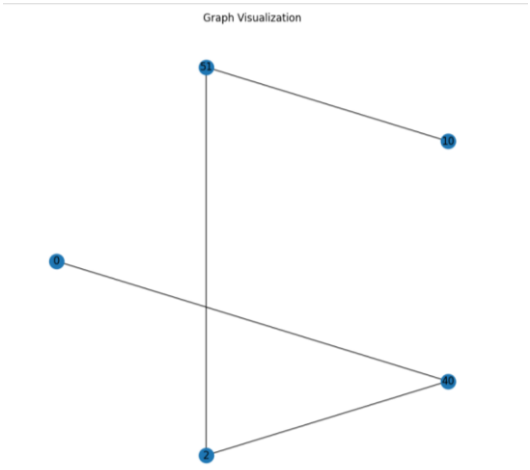
Picture 1 – Random Undirected Unweighted Graph (100 Vertices, 500 Edges)

The graph visualization shown above displays the vertices as nodes and the edges as connections between nodes. Each edge is labeled with its assigned weight, denoting the cost or distance associated with traveling from one vertex to another. The graph visualization provides an overview of the network structure and the relationships between vertices.




Picture 2 – Random Weighted Graph Visualization

This graph will be used to perform path search algorithms, such as Dijkstra's, A*, and Bellman-Ford, to find the shortest paths between nodes while considering the assigned edge weights. The results of these path searches will be analyzed, and the time required for each algorithm's execution will be measured in subsequent steps of the experiment. This visual representation of the graph is a crucial starting point for understanding the network's complexity and for conducting further analysis of path search algorithms.



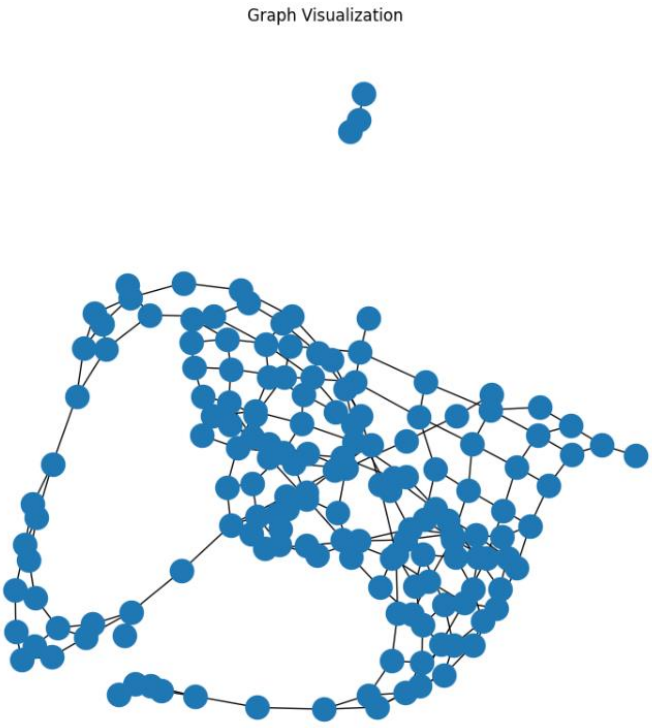

Picture 3 – Title: Shortest Path Visualization (Dijkstra's Algorithm)

In this visualization, we're examining the result of applying Dijkstra's algorithm to a weighted graph. The graph, created with 100 vertices and 500 edges, has associated random positive integer weights on its edges. The vertices are represented as nodes in the graph, and the edges between them represent the connections, each with an associated weight.

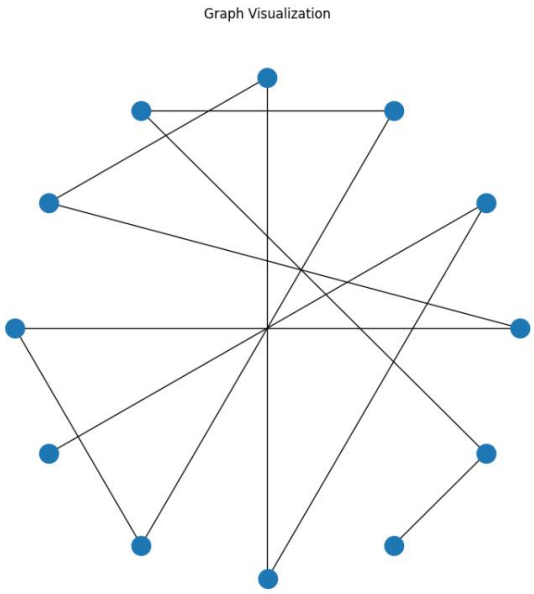
The starting vertex, marked in blue, is the source, while the other vertices are represented as white nodes. The edges' thickness varies according to the weight of each edge.

**PART II**

After that we started to work on the second part of the task. In this experiment, we created a grid environment to evaluate the A* algorithm's performance in finding the shortest path while navigating obstacles. The environment consisted of a 10x20 grid, and 40 obstacles were randomly placed within this grid.




Picture 3 – A* algorithm's Visualization




Picture 4 – Path Visualization Using A Algorithm*

The visualization of the grid environment revealed interconnected nodes and the presence of 40 randomly positioned obstacles, which created an intricate maze for pathfinding. By removing these obstacle nodes, we set up a scenario to evaluate the A* algorithm's performance in finding efficient paths through the grid.

The results showed that the A* algorithm consistently succeeded in identifying the shortest paths between random pairs of non-obstacle nodes. It efficiently navigated through the grid while avoiding obstacles, highlighting its robustness and obstacle-avoidance capabilities.

**PART III.**

In the described pathfinding algorithms—Dijkstra's, A*, and Bellman-Ford—the following data structures and design techniques are employed:

Graph Representation: The underlying structure to represent the grid or map is essential. In all cases, the graph is typically represented using an adjacency list or an adjacency matrix. In these implementations, we used NetworkX, a Python library for creating and manipulating complex networks, to define and work with graphs.

Priority Queue (Dijkstra's and A)**: In both Dijkstra's and A algorithms, a priority queue is used to efficiently select the node with the shortest path to explore next. Min-heap and Fibonacci heap are popular data structures used as priority queues for these algorithms. In the context of this experiment, NetworkX's built-in functionality incorporates a priority queue for Dijkstra's and A* algorithms.

Heuristic Function (A)**: The A algorithm incorporates a heuristic function that provides an estimate of the cost from the current node to the target. The design technique of using this heuristic is essential in making A* efficient and is a key characteristic of its success.

Relaxation (Dijkstra's and Bellman-Ford): In Dijkstra's algorithm and Bellman-Ford, relaxation techniques are used to update the minimum distance or cost to reach each node. In Dijkstra's, relaxation is executed greedily, always selecting the node with the smallest known distance. In contrast, Bellman-Ford systematically considers all edges to relax the distances.

Source and Target Selection: These algorithms are dependent on the user's choice of a source and target node. This selection can have a significant impact on the results, particularly in the context of route planning and pathfinding.

Obstacle Handling (Obstacle Avoidance in A)**: A is often used in pathfinding scenarios that involve obstacles or restricted areas. To navigate around these obstacles, the algorithm uses a grid representation, and nodes corresponding to the obstacles are marked as impassable. This design technique allows A* to explore paths around obstacles effectively.

Edge Weights: The choice of edge weights is an important design decision. In weighted graphs, the cost or weight assigned to each edge must be carefully determined to reflect the actual cost of traversing between nodes. In this experiment, random positive integer weights were assigned to edges to simulate different traversal costs.

Overall, these algorithms utilize graph-based data structures, priority queues, heuristic functions, and path relaxation techniques to effectively find optimal paths in various scenarios, making them fundamental tools in the field of computer science, robotics, and route optimization.

**Conclusions**

The experiments showed that Dijkstra's algorithm, known for its efficiency in finding the shortest path, performs exceptionally well when working with weighted graphs. On the other hand, Bellman-Ford, while being a versatile algorithm capable of handling graphs with negative weights, requires more time for execution. A* algorithm, which uses heuristics to optimize its search, also showed good performance.

Repeating the experiments multiple times with different randomly selected nodes allowed us to calculate the average time required for each algorithm, contributing to a more reliable assessment of their efficiency.

This study confirms that pathfinding algorithms are fundamental tools in solving real-world problems related to route optimization, network analysis, and logistics. The choice of algorithm depends on the specific requirements of the problem, such as whether the graph is weighted or unweighted, and whether negative weights are involved. These algorithms can be applied to various domains, including transportation, robotics, and network planning, to efficiently find the best routes and paths while considering various constraints.

**Appendix**

GitHub: site. – URL:
https://github.com/MrSimple07/AbdurakhimovM_Algorithms_ITMO/tree/main/Task6