FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report

on the practical task 6

“Algorithms on graphs. Path search algorithms on weighted graphs.”

Performed by

Ilya Lyalinov

Academic group 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)

Problem formulation

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

G = (V, E) -- graph.
Time complexity estimates for the corresponding algorithms from the "problems formulation" section:

1) Dijkstra's method: $O(|V|^2)$
2) Bellman-Ford method: $O(|V| * |E|)$. May be used for graphs with negative weights.
3) A*: $O(|E|)$. However, A* performs faster than Dijkstra's algorithm because it uses a function to calculate the approximate distance to the target vertex.

Results

I. We generated an adjacency matrix with 100 vertices and 500 random weighted edges. We measured average execution time for 10 runs of Bellman-Ford and Dijkstra's algorithms for the same starting node.

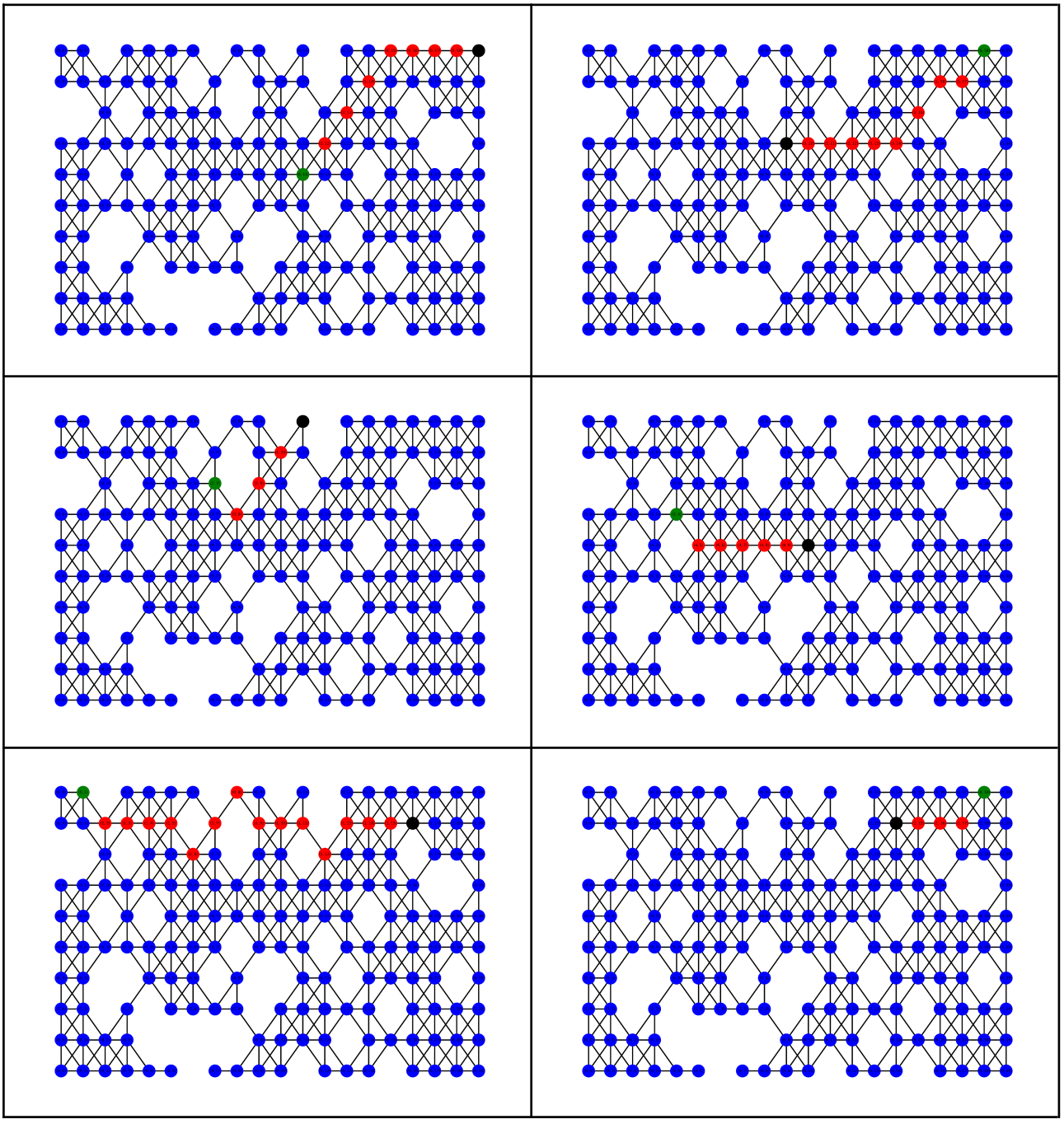
$avg\_time_{Dijkstra} = 0.0006\ seconds$

$avg\_time_{BF} = 0.0015$ secnods

As expected, BF method performed slower than Dijkstra's method.

II. We generated a 10x20 cell grid with 40 obstacle cells at random positions. We ran A* algorithm 6 times. Each time for a different randomly chosen pair of start and finish cells. We used euclidean distance between 2 points as a heuristic function for A* method.

Visualized results (green is the start vertex and black is the end; route marked in red):



At every iteration A* algorithm found an optimal solution which coincided with the solution obtained by applying the Dijkstra's algorithm to the same problem with the same initial pair of vertices. The final paths obtained by A* may have been different, but their lengths were optimal. A* ran 1.5-3 times faster than Dijkstra's algorithm on the same input data.

III.

Data structures:
Adjacency matrix n x n was used to store information about the graphs. Dijksta's method uses an array for storing distances, array for marking used/unused vertices and array for memorizing paths. BF algorithm uses an adjacency list, an array for storing distances and an array for memorizing paths. A* method uses priority queue for storing vertices that needs to be reviewed and an array for marking used/unused vertices.
Design techniques:
Dijkstra's and A* algorithms are greedy. BF method uses dynamic programming approach.

Conclusion

We generated random adjacency matrix for a simple undirected weighted graph of 100 vertices and 500 edges. We used BF and Dijktra's algorithms to find shortest paths. We measured average execution time for each algorithm for the same starting node. We analyzed the results in terms of execution time.
We generated 10x20 grid with 40 obstacle cells at random positions. We executed A* for 6 different random pairs (start and end) of cells to find the shortest path using euclidean distance as a heuristic function. We visualized the results, measured execution time and stated that A* executes faster than Dijkstra's method on the same input data on a grid graph.
We described data structures and design techniques used within the algorithms.

Appendix

https://github.com/ilyalinov/Algorithms