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
Denis Zakharov,
Maxim Shitilov
Academic group J4132c
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

The use of path search algorithms on weighted graphs.

Brief theoretical part

Firstly we have define what a Graph is; it's a structure amounting to a set of objects in which some pairs of the objects are in some sense "related". The objects correspond to mathematical abstractions called vertices (also called nodes) and each of the related pairs of vertices is called an edge. Typically, a graph is depicted in diagrammatic form as a set of dots or circles for the vertices, joined by lines or curves for the edges.

For this task we are going to work with a weighted graph in which a weight is assigned to each edge. Let's give short description for the path search algorithms on weighted graph that we are going to use:

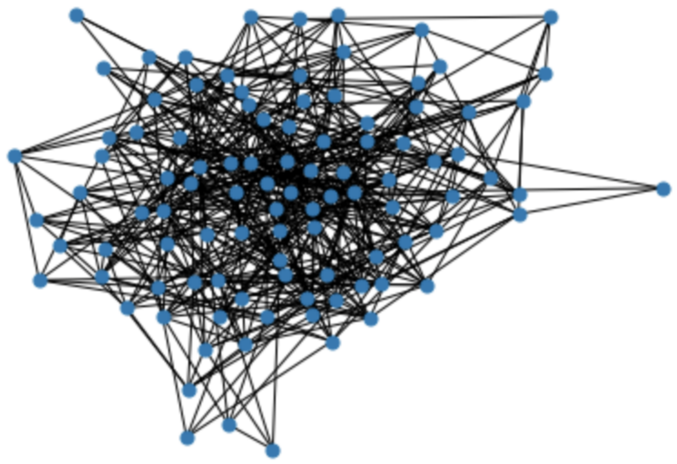
- **Dijkstra's algorithm** — generates a shortest path tree (SPT) with the source as a root, with maintaining two sets: one set contains vertices included in SPT, other set includes vertices not yet included in SPT. At every step, it finds a vertex which is in the other set and has a minimum distance from the source. **Important:** Dijkstra does not generally work with negative weights.
    - Time Complexity — time complexity is from $O(|V|\, log\, |V|)$ to $O(|V|^2)$, where $|V|$ − *cardinality of vertices.*
- **Bellman-Ford algorithm** — At $i - th$ iteration, Bellman-Ford calculates the shortest paths which have at most i edges. As there is maximum $|V| - 1$ edges in any simple path, $i = 1, ..., |V| - 1$.

  Assuming that there is no negative cycle, if 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.

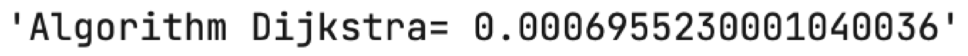
  To check if there is a negative cycle, make $|V|$-th iteration. If at least one of the shortest paths becomes shorter, there is such a cycle.
    - Time Complexity — $O(|V||E|)$, where $|V|$ − *cardinality of vertices*, $|E|$− *cardinality of edges.*
- **A* algorithm** — At each iteration, A* determines how to extend the path basing on the cost of the current path from the source and an estimate of the cost required to extend the path to the target.
    - Time complexity is $O(|E|)$, where $|E|$− *cardinality of edges.*

Results

1) Explain differences (if any) in the results obtained.

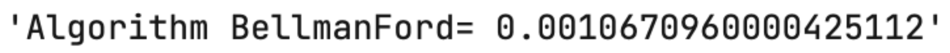


Pic. 1 — Visualization of generated graph



```
'Algorithm Dijkstra= 0.0006955230001040036'
```

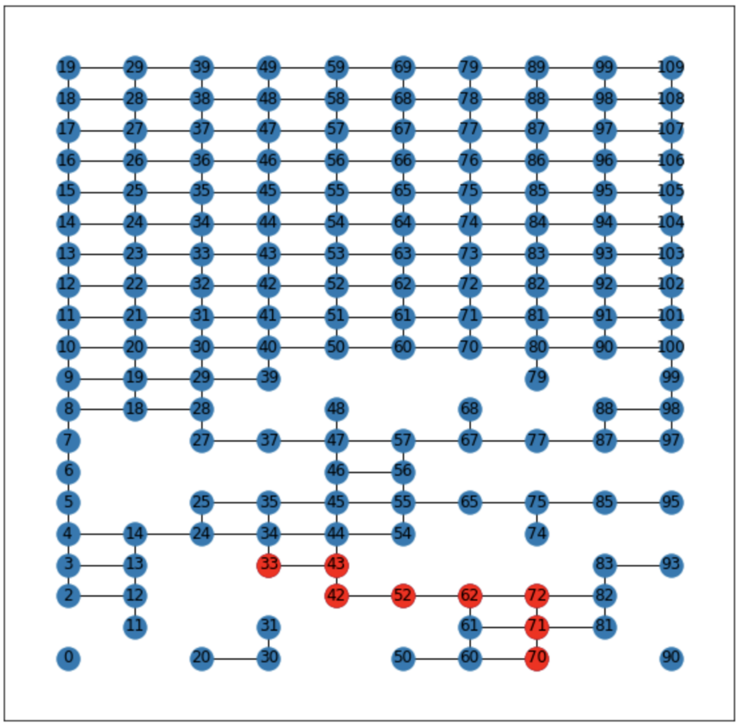
Pic. 2 — Algorithm Dijkstra mean time



```
'Algorithm BellmanFord= 0.0010670960000425112'
```

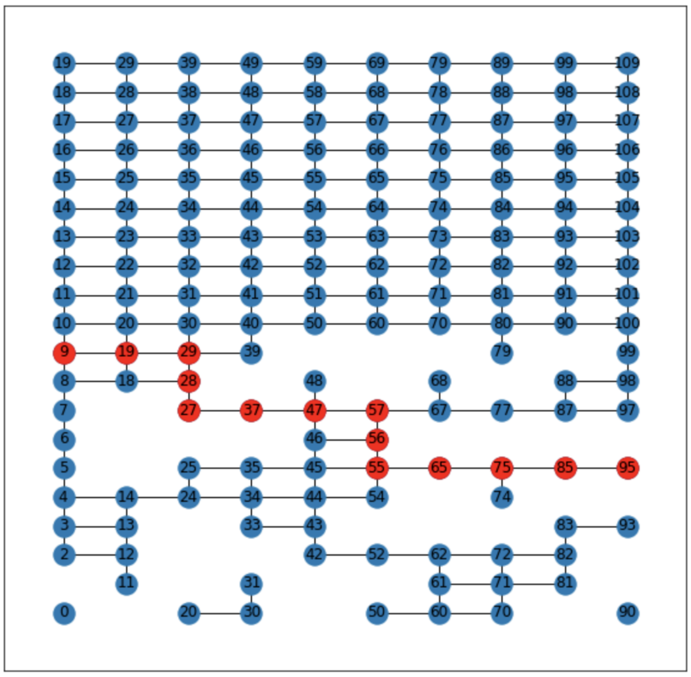
Pic. 3 — Algorithm BellmanFord mean time

```
[(3, 3), (4, 3), (4, 2), (5, 2), (6, 2), (7, 2), (7, 1), (7, 0)]
```



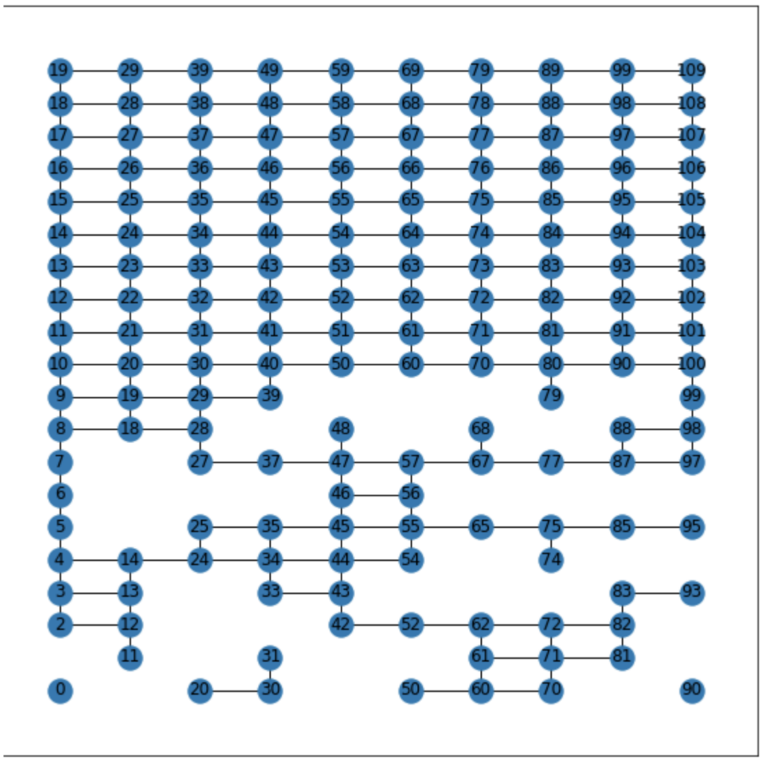
Pic. 4 — The first pair of random allowed cells A* method

```
[(0, 9), (1, 9), (2, 9), (2, 8), (2, 7), (3, 7), (4, 7), (5, 7), (5, 6), (5, 5), (6, 5), (7, 5), (8, 5), (9, 5)]
```



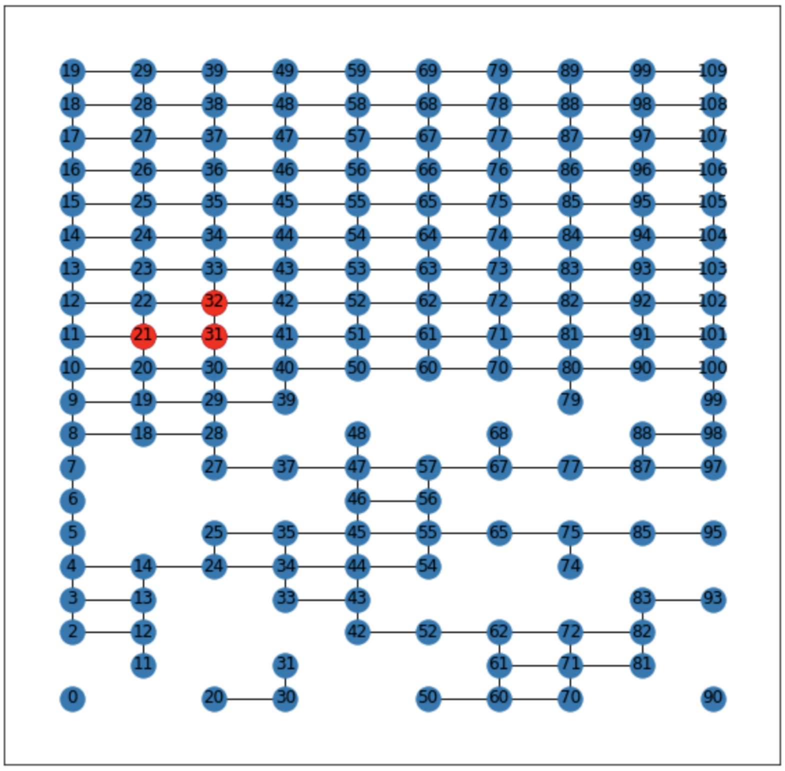
Pic. 5 — The second pair of random allowed cells A* method

```
printGrid(GRID, (0, 19), (9, 0))
```



Pic. 6 — The third pair of random allowed cells A* method

```
[(1, 11), (2, 11), (2, 12)]
```



Pic. 6 — The fifth pair of random allowed cells A* method

Conclusions

To sum up, we have used different use of path search algorithms on weighted graphs (Dijkstra's, A* and Bellman-Ford algorithms). And in our experiments Bellman-Ford works six times slower than Dijkstra's (0.001 and 0.0006 seconds) that confirms their theoretical time complexities. And it’s still hard to visualize a graph.

Questions for self-monitoring from Russian educational materials

1. Which algorithm for finding the shortest path – Dijkstra or Bellman-Ford – is usually used to work with weighted graphs with negative weights and possible negative cycles?

To work with weighted graphs with negative weights and possible negative cycles, the **Bellman-Ford** algorithm is usually used because Dijkstra's algorithm does not work for edges with negative weights. Good explanation is located on the Stack Overflow thread.

2. Is the A* algorithm applicable to a weighted graph associated with a geographical map (vertices are cities, weights on edges are distances between cities)? Why do we need heuristics in the A* algorithm?

Yes, we can calculate the distance in straight line between two cities, and then can use A* instead of Dijkstra, using the function of distance as heuristic. For sure we will expand fewer nodes than with Dijkstra.

Appendix

DataLore: site. – URL: https://datalore.jetbrains.com/notebook/RemqSkuJwmr1PM4Gc3cBqB/dIFKBWz6sRaBpispZoyV8u (circulation date: 05.10.2022)

Stack Overflow: “Why doesn't Dijkstra's algorithm work for negative weight edges?” by Madu site. – URL: https://stackoverflow.com/questions/13159337/why-doesnt-dijkstras-algorithm-work-for-negative-weight-edges