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

Chumakov Mike

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)*

Problems and methods

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 10x10 cell grid with 30 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

Weighted graph is a graph in which weight (some number) is assigned to each edge. Weighted graphs require special bypass algorithms. Let's consider some of them.

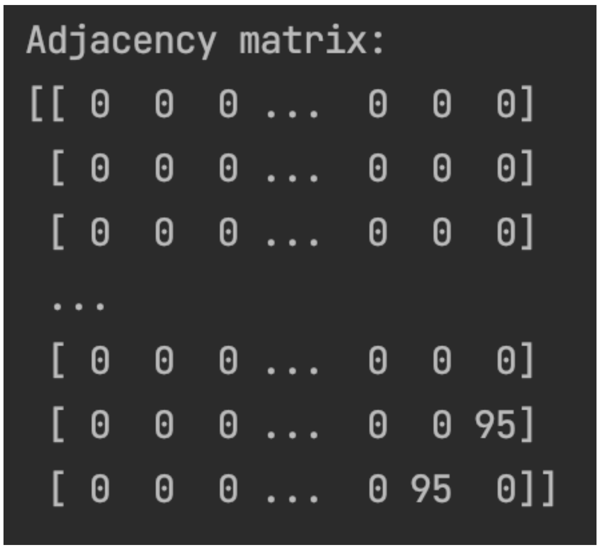
Dijkstra's algorithm (DA) solves the following problem: forgiven graph (with positive weights) and source vertex s, find the multiple the fastest paths from s to other vertices. The main idea behind Dijkstra's algorithm is that it generates the shortest path tree (SPT) rooted s by processing two sets: one set contains vertices, included in SPT, others are vertices not yet included in SPT. At each step, the algorithm finds a vertex that is not included in the SPT and has the minimum distance from the source. The complexity of the algorithm ranges from $O(|V| \log |V|)$ *to* $O(|V|^2)$ *depending on the applied modification.*

*Algorithm A * solves the following problem: for graph data (with positive weights), source s, and target t, find the shortest path from s to t. The main idea of the algorithm is that at each iteration it determines how to extend the path based on the cost of the current path from s to the point of extension and estimate the cost of the path from the point of extension to t (this is a heuristic in the A * algorithm). The time complexity of the algorithm is* $O(|E|)$*. algorithm A * in the context of a mesh with obstacles representable as a weighted graph. In this case, the heuristic cost estimate paths at each iteration of the algorithm are calculated without considering obstacles.*

The Bellman-Ford Algorithm (BFA) solves the following problem: for a given weighted graph (possibly with negative weights) and source s find the shortest paths from s to all other vertices. If the graph contains a negative cycle C_ (i.e., a cycle, the sum of the edges where is negative), reachable from s, then there is no shortest path: any path with a vertex in C_ can be made shorter by one more traversal of C_. In this case, the BFA reports a negative C cycle. The main idea behind BFA is as follows. At the i-th iteration, BFA calculates the shortest paths that have at most i edges. Because the in any simple path at most $|V| - 1$ *edges,* $i = 1,. ..., |V| - 1$*. Assuming that there is no C_, if we have computed the shortest paths from at most i edges, then iterating over all edges guarantees to obtain the shortest paths with at most* $i + 1$ *edges. To check if there is a negative cycle* $C-$*, BFA performs the* $|V|$ *th iteration. If at least one of the shortest paths becomes shorter, then there is a negative cycle C_.The time complexity of BFA is quite high and is estimated as* $O(|V||E|)$*.*

Results

Fragment of adjacency matrix for a graph with 100 vertices and 500 edges withrandom weight. (Figure 1).



```
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  0  0 ...  0  0 95]
 [ 0  0  0 ...  0 95  0]]
```

Figure 1: Fragment of adjacency matrix.

Dijkstra's Algorithm

For a random vertex of the graph, the shortest paths were found by Dijkstra's algorithm. 10 experiments were carried out to measure the running time of the algorithm. The average running time of the algorithm is 0.0349 sec(for vertex 0).

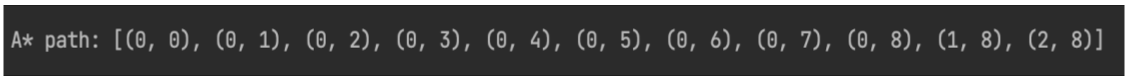
The Bellman-Ford Algorithm

For a random vertex of the graph, the shortest paths were found by Bellman-Ford Algorithm. 10 experiments were carried out to measure the running time of thealgorithm. The average running time of the algorithm is 0.0887 sec (for vertex 0).

As expected, Dijkstra's algorithm performed faster than the Bellman-Fordalgorithm.

*Algorithm A **

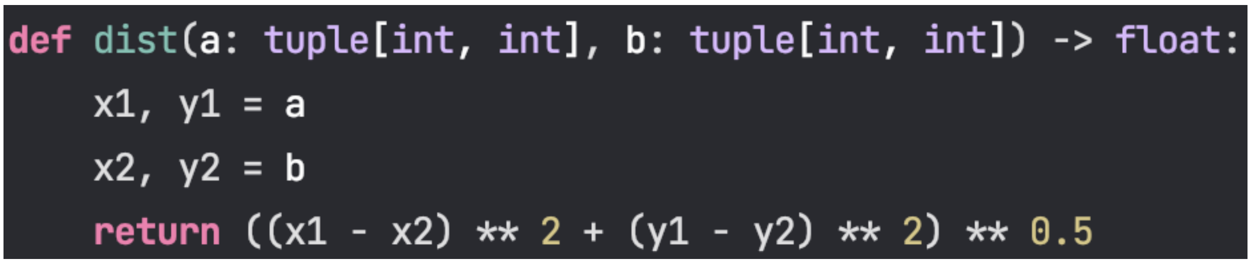
*A 10 × 10 grid was generated with 30 obstacle cells. Two random allowed cells are selected ((0,0), (2,8)) and the shortest path between them is found using the A *algorithm. (Figure 2)*



```
A* path: [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (1, 8), (2, 8)]
```

Figure 2: Shortest path from point (0,0) to point (2,8).

A heuristic function in python to evaluate the estimate of the distance from a node to thetarget see Figure 3:



```python
def dist(a: tuple[int, int], b: tuple[int, int]) -> float:
    x1, y1 = a
    x2, y2 = b
    return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5
```

Figure 3: distance formula.

Algorithms for finding the shortest path on the graph was done using the Pythonlanguage and the library networkx.

Conclusions

*In this laboratory work, pathfinding algorithms on weighted graphs were implemented and investigated. For a graph with 100 vertices and 500 edges, the algorithms of Dijkstra and Bellman-Ford were considered. As expected, Dijkstra's algorithm ran faster. For the graph, presented in the form of a 10x10 grid with 30 obstacle cells, the A * algorithm was implemented and investigated.*

Appendix

Code Link