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
Tiulkov Nikita
J4133c

Accepted by
Dr Petr Chunaev

St.Petersburg
2021

1. Goal

The use of path search algorithms on weighted graphs (Dijkstra's, A* and Bellman-Ford algorithms).

2. 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 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.

3. Brief theoretical part

3.1. Dijkstra's algorithm

Problem: given a weighted graph (with positive weights) and a source vertex, find shortest paths from the source to all other vertices.

Main idea: It 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.

Time complexity is from $O(|V|\log|V|)$ to $O(|V|^2)$.

Algorithm:

- Create an SPT set sptSet that keeps track of vertices included in SPT, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.
- Assign a distance value to all vertices in the input graph. Assign the distance value for the source vertex as 0. Assign the distance value for other vertices as ∞.
- While sptSet does not include all vertices:
  - Pick a vertex $u \notin$ sptSet that has a minimum distance value.
  - Include u in sptSet.
  - Update the distance values of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if the sum of distance value of u (from the source) and weight of edge (u, v) is less than the distance value of v, then update the distance value of v.

3.2. Bellman-Ford algorithm

Problem: Given a weighted graph (possibly directed and with negative weights) and a source vertex s, find shortest paths from s to all vertices in the graph. If a graph contains a negative cycle (i.e. a cycle whose edges sum to a negative value) that is reachable from s, then there is no shortest path: any path that has a point on the negative cycle can be made shorter by one more walk around the negative cycle. In such a case, Bellman–Ford can detect the negative cycle.

Note: Dijkstra does not work for negative weights. Bellman-Ford is simpler than Dijkstra but its time complexity is $O(|V||E|)$.

Idea: 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.

3.3. A* algorithm

Problem: given a weighted graph (with positive weights), a source vertex and a target vertex, find a shortest path from the source to the target.

Main idea: 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 $O(|E|)$.

One gets Dijkstra's algorithm if the above-mentioned estimate is skipped.

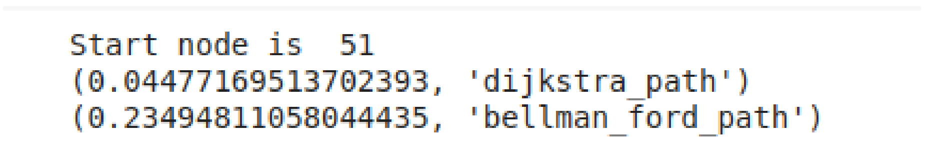
Consider a grid with obstacles that can be represented as a weighted graph (vertices=cells, weighted edges=shortest paths and their lengths).

Algorithm: We are given a square grid with obstacles, a source cell c_s and a target cell c_t. The aim is to reach c_t (if possible) from c_s as quickly as possible.

A* extends the path according to the value of $f(c_k) = g(c_k) + h(c_k)$, where g measures the cost to move from c_s to c_k by the path generated and h measures the cost to move from c_k to c_t.

4. Results

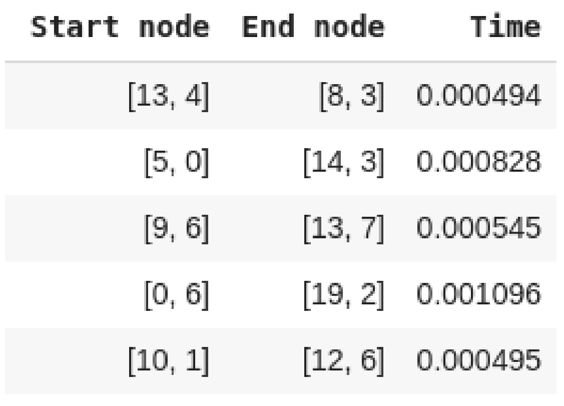
1. The result of using Dijkstra's and Bellman-Ford algorithms to find shortest paths between a random starting vertex and other vertices is shown on Figure 1. As we can see Dijkstra's algorithms works 5 times faster than Bellman-Ford algorithms. And it is explainable by their time complexity. Because we are using 500 edges and 100 vertexes.



```
Start node is  51
(0.04477169513702393, 'dijkstra_path')
(0.23494811058044435, 'bellman_ford_path')
```

Figure 1: Dijkstra's and Bellman-Ford's results

II. The results of using A* algorithms to find shortest paths between two random non-obstacle cells is shown on Figure 2. Since we need to find shortest path in a simple graph grid that algorithm is appropriate for this task. It is really depends on heuristic. In my case I don not specify heuristic. So algorithm was using default heuristic - h=0 - same as Dijkstra's algorithm. The average time is around 0.0007 sec.



Start node	End node	Time
[13, 4]	[8, 3]	0.000494
[5, 0]	[14, 3]	0.000828
[9, 6]	[13, 7]	0.000545
[0, 6]	[19, 2]	0.001096
[10, 1]	[12, 6]	0.000495

Figure 2: A* results

III. For working with graphs the «nx» Python library was used. For both algorithms we are using data structures as array and graph.

For Dijkstra's and A* algorithms greedy design technique is used. And for Bellman-Ford algorithm dynamic programming technique is used.

5. Conclusions

As a result of this work, main algorithms for finding shortest paths between graph's vertexes were investigated. Average time was calculated and compared.

Appendix

The code of algorithms you can find on GitHub: https://github.com/FranticLOL/ITMO_Algorithms