FEDERAL STATE AUTONOMOUS EDUCATIONAL
INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report
on the practical task No. 6
Task 6. Algorithms on graphs. Path search algorithms on weighted graphs.

Performed by

Anastasiia Pokasova

J4133c+

Accepted by

Dr Petr Chunaev

St. Petersburg
2020

Goal

The use of path search algorithms on weighted graphs (Dijkstra's, A* and BellmanFord algorithms)

Problems and methods

- 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.
- 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
- Describe the data structures and design techniques used within the algorithms.

Brief theoretical part

Graph Undirected graph is an ordered pair $G = (V, E)$, where V is a set of vertices and E is a set of edges.

Graph can be represented using any of listed structures:

- Adjacency list
- Adjacency matrix
- Incidency list
- Incidency matrix

Dijkstra's algorithm Dijkstra's algorithm, conceived by Dutch computer scientist Edsger Dijkstra in 1956 and published in 1959, is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree.

This algorithm is often used in routing and as a subroutine in other graph algorithms.

For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex.

Original algorithm time complexity: $O((|V| + |E|)logV)$

Bellman-Ford algorithm The Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted graph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers.

Time complexity: $O(|V| * |E|)$, which is slower than Dijkstra's algorithm.

A * algorithm A * is an heuristic algorithm for shortest path computing. It provides better results than Dijkstra's algorithm, but has bigger space complexity. Results of the algorithm are dependant on chosen heuristic and maze rules (i.e. availability of diagonal traversal).

Possible heuristics:

- Manhattan: $h(v) = |v.x - goal.x| + |v.y - goal.y|$
- Euclid: $h(v) = sqrt((v.x - goal.x)^2 + (v.y - goal.y)^2)$.
- Chebyshev: $max(|v.x - goal.x|, |v.y - goal.y|)$

Time complexity: $O(|E|)$.

Results

Design and data structures

Data structures used in this laboratory work are listed below:

- matrix
- linked list
- priority queue
- set

Matrix Matrix is a NXM table of elements.

Linked list A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. A linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.

Queue A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). Can be implemented via array or linked list.

A priority queue is an abstract data type similar to a regular queue or stack data structure in which each element additionally has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority. In some implementations, if two elements have the same priority, they are served according to the order in which they were enqueued, while in other implementations, ordering of elements with the same priority is undefined.

Set A set is an abstract data type that can store unique values, without any particular order.

Design Dijkstra's algorithm is greedy algorithm, which means that at each stage they make locally optimal choice.

Bellman-Ford uses dynamical programming for shortest path search. It is breaking a problem into simpler sub-problems in a recursive manner.

A* algorithm applies Best-First search design. It explores a graph by expanding the most promising node chosen according to a specified rule.

Results

Implementations of algorithms for this laboratory work follow original papers' implementations, so it is expected that Dijkstra's algorithm will perform faster than Bellman-Ford algorithm.

Measured time for each algorithm is shown in a table below:

	Dijkstra	Bellman-Ford
Estimated	$O(\|V\| + \|E\|log\|V\|)$	$O(\|V\| * \|E\|)$
Measured	7.78 microseconds	104.4 microseconds

As expected, Dijkstra's algorithm works much faster than Bellman-Ford.

A * algorithm results:

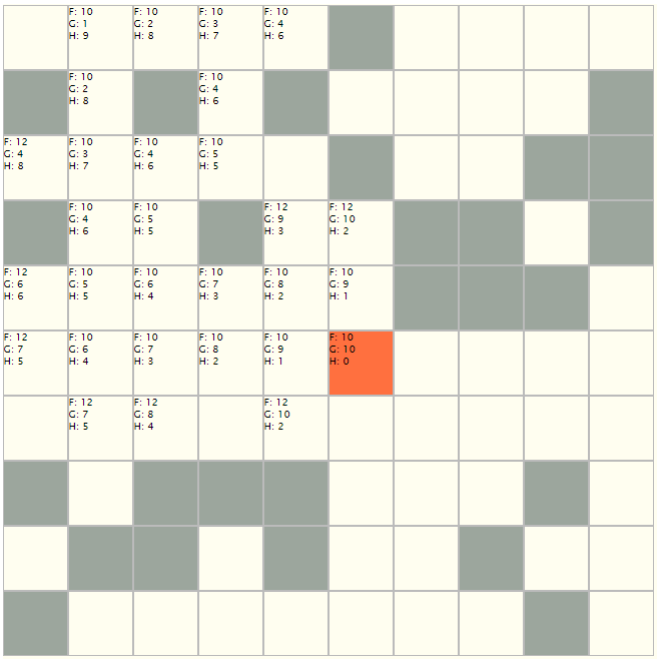

Figure 1: (0,0) to (5,5)

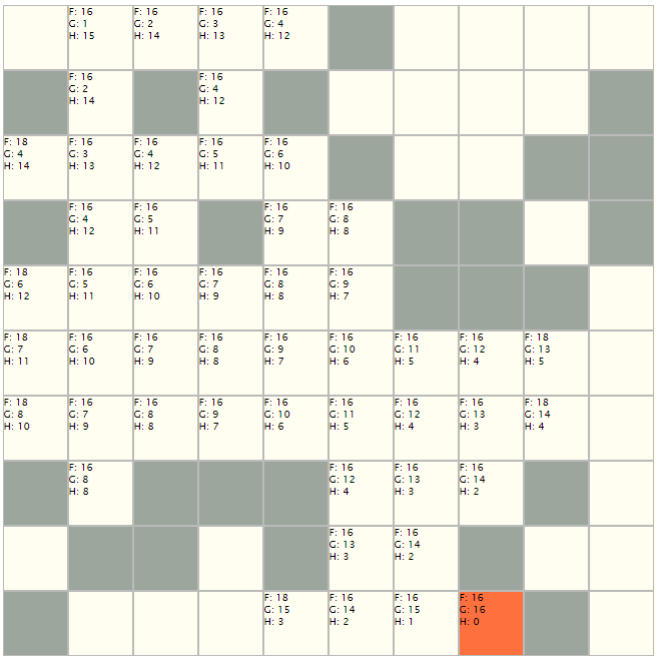

Figure 2: (0,0) to (7,9)

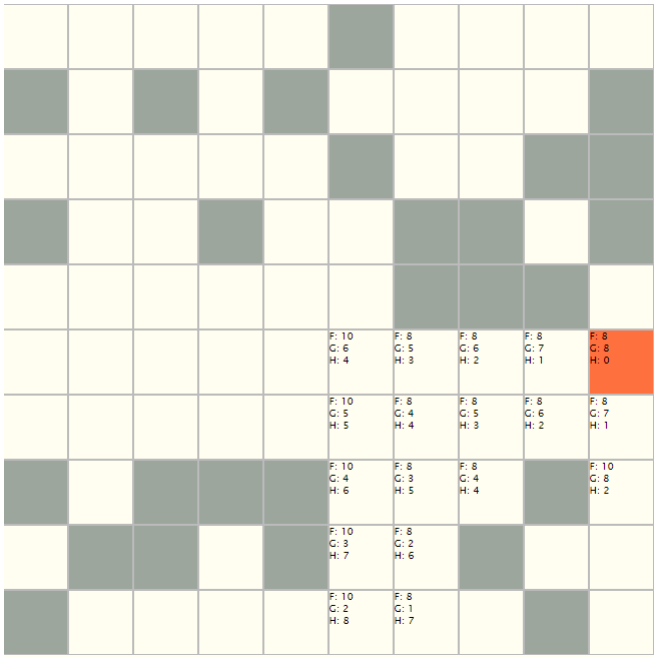

Figure 3: (7,9) to (9,5)

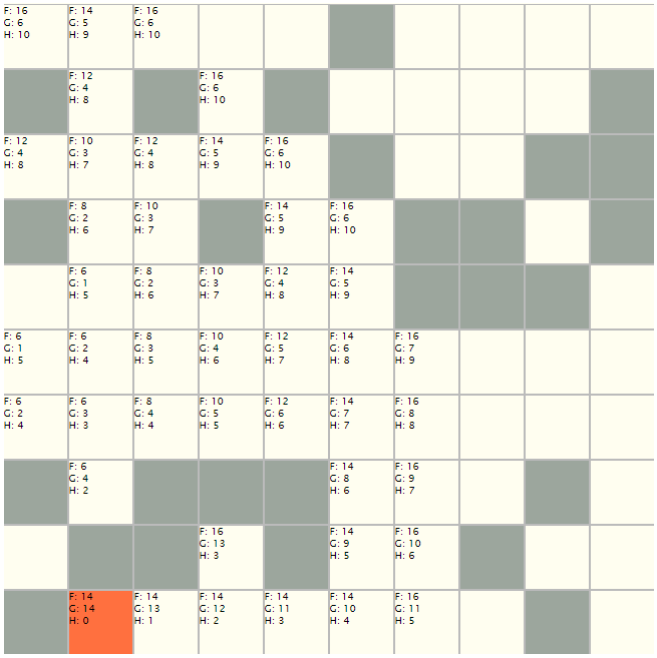

Figure 4: (0,4) to (1,9)

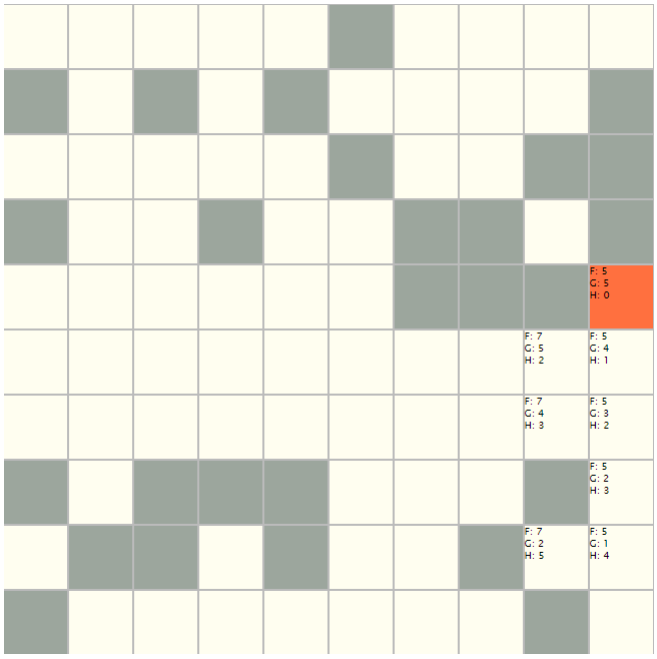

Figure 5: (9,9) to (9,4)

As we can see, if the target point is reachable from source point, A * will provide a resulting path independent from chosen heuristic. Only the points included in final candidate path will change.

Conclusion

As a part of a laboratory work, path search algorithms were compared. A program was implemented for the task of comparison.

Source code

Source code for the implementation is located here: here: `https://github.com/NeoIsALie/Algorithm_Analysis/tree/lab6`.