FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report enhancement

On the practical task number 6
“Algorithms on graphs. Path search algorithms on weighted graphs”

Performed by:
Patrik Marek Stronski
ICU number: 333976
J4133C
Accepted By:
Dr Petr Chunaev

St. Petersburg
2021

Goal
The goal of this task is to measure a few graph searching algorithms. The aim is to measure their application in solving two problems from the domain of shortest path finding:

- finding the shortest path between to nodes in a weighted graph
- finding the shortest path in a maze with obstacles

For this purpose 3 algorithms are about to be used:

- Dijkstra's algorithm
- Bellman-Ford algorithm
- A* algorithm

The goal is to measure the average time of their execution.

Formulation of the problem
The main task is to find the shortest path from some point A to other point B and measure how effective are the algorithms in finding the shortest path.

Brief theoretical part
In this examples I am about to analyze optimal path finding for graphs.

In the first example I am analyzing two algorithms – Dijkstra's and Bellman-Ford. Both of them are implemented using SciPy library.

Dijkstra's algorithm is rather an algorithm for finding the shortest path between a source and all other nodes. Nevertheless, it can be used for path finding between two points – its way of working simply makes it a bit longer. Its worst complexity is O(|v| ^2)

Bellman-Ford algorithm is a similar algorithm. Its big advantage, though, is capability of handling negative weights. It also finds route from the source to all other vertices, or simply stops upon the desired vertex is found. It is slower than Dijkstra's one. Its complexity is O(|v| * |E|)

For finding the shortest path in a auto-generated maze I used A* algorithm. This one I wrote myself, based on that, as Python language itself is not very fast, it takes a bit more time. The plus if my implementation is that I receive directly the path that was found – the shortest one. The algorithm itself uses a heuristic – it calculates the naive distance between the current node and start, end node. This way it does not waste time doing search for all paths, but only the one that is prospectively best. Its complexity is O(|E|) which makes it the most efficient one.

Used structures and methods
When it comes to the data structures for depicting graph I used adjacency matrix. This structure is really simple to use, as it is represented by square matrix. For each node the relations to other nodes are represented using amounts – 0 if the edge does not exist or some value, if the edge exists. The weights are usually positive, however negative ones can also happen (but not in my examples).

For my A* algorithm implementation I merged the functional and object-oriented approach. The map is represented by the list of lists (matrix), whereas the cell by the class *Cell*. Object Oriented approach made it really easy to both link the instances (via *parent* property) and calculate all the needed distances. The A* algorithm itself is a function which simply encapsulates the cells into classes. In the end the cells are expressed using tuples.

Results 1st part

For the first part I used both Dijkstra's and Bellman-Ford algorithms. I generated the non-directed graph which I showed using adjacency matrix.

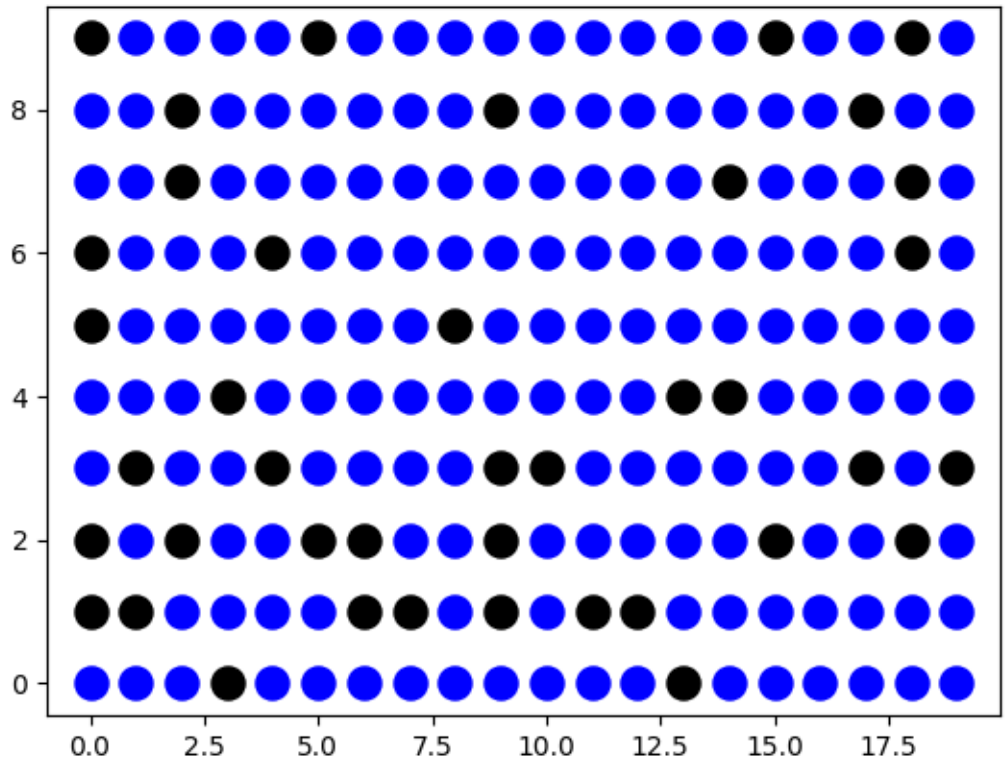
Algorithm name	Average time [ms]	Comment
Dijkstra's	0.24	As we can see it is faster than BF. The algorithm is more than 2 times better.
Bellman-Ford	0.55	As we can see the algorithm is worse. Nevertheless due to its optimization within SciPy package the results are also nearly immediate.

Results 2nd part

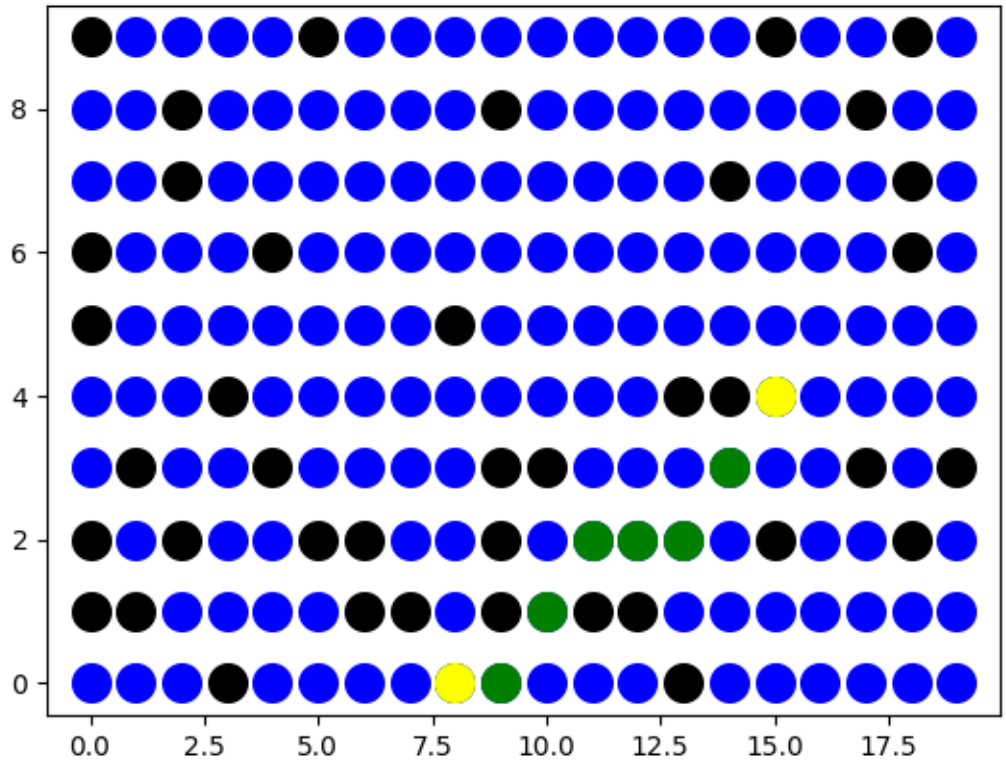
For the second part I wrote the algorithm myself. Here, because of the lack of optimizations, the results are far worse than for the previous algorithms. Nevertheless, in theory the algorithm should get better results for the same problem size, as it designed for exactly searching shortest path between two known points.

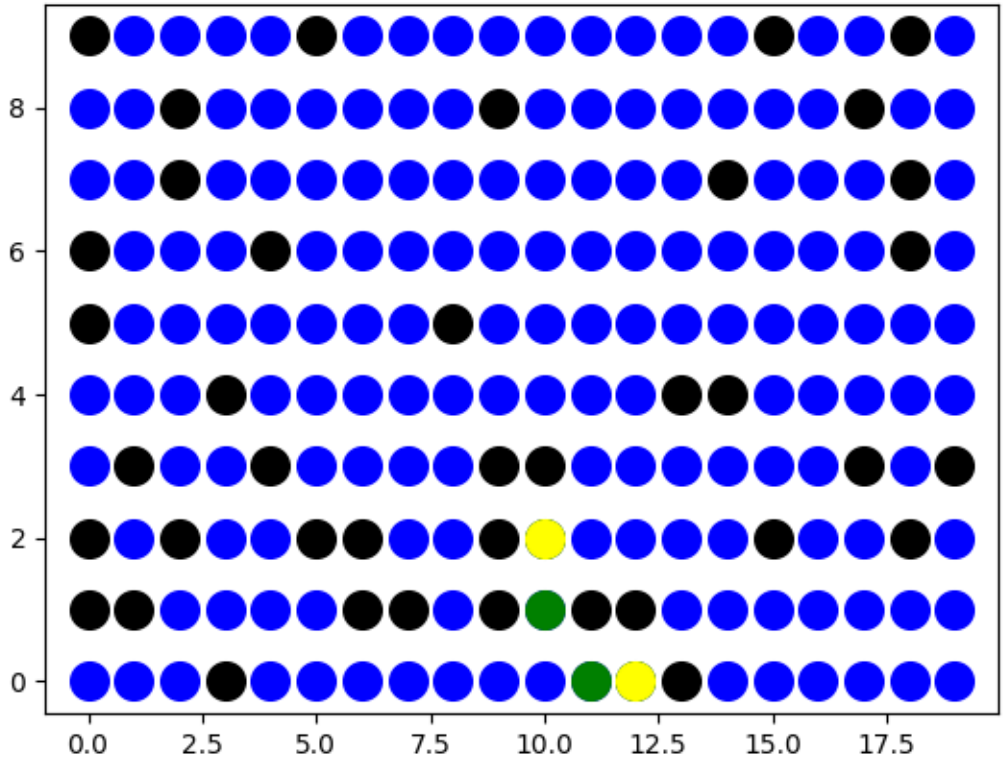
Here the points were, in fact, cells. The cells in the maze were adjacent to other cells apart from the case where there was an obstacle. I generated the obstacles randomly. Obstacle is the point through which there is no entry.

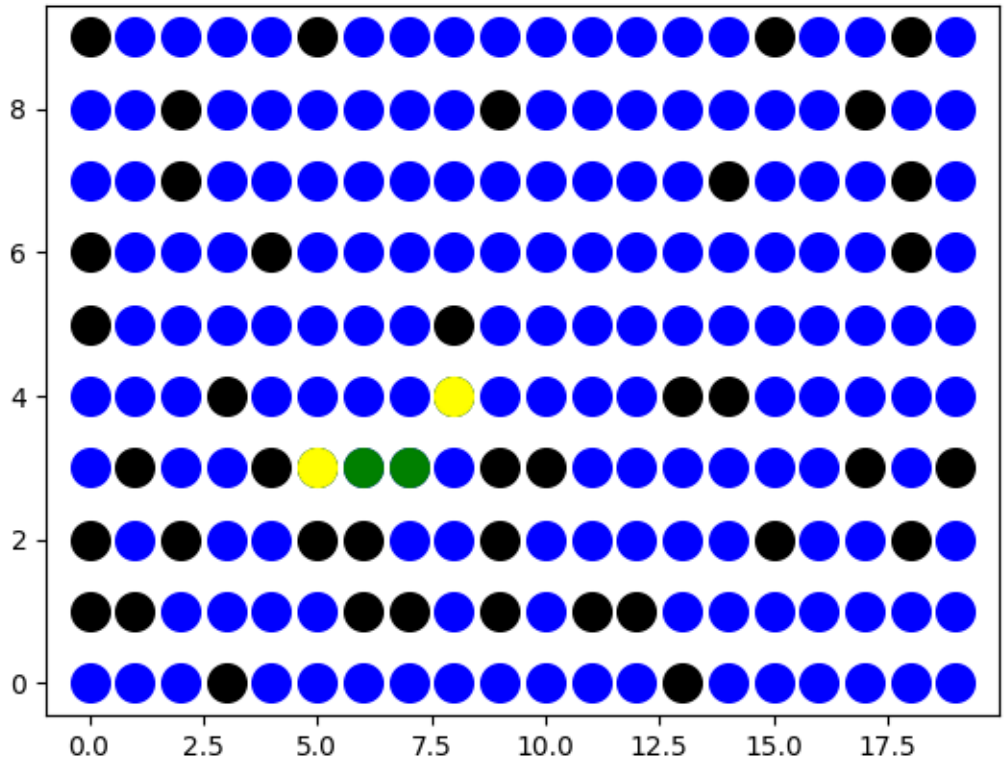
The maze in the beginning looked like this:

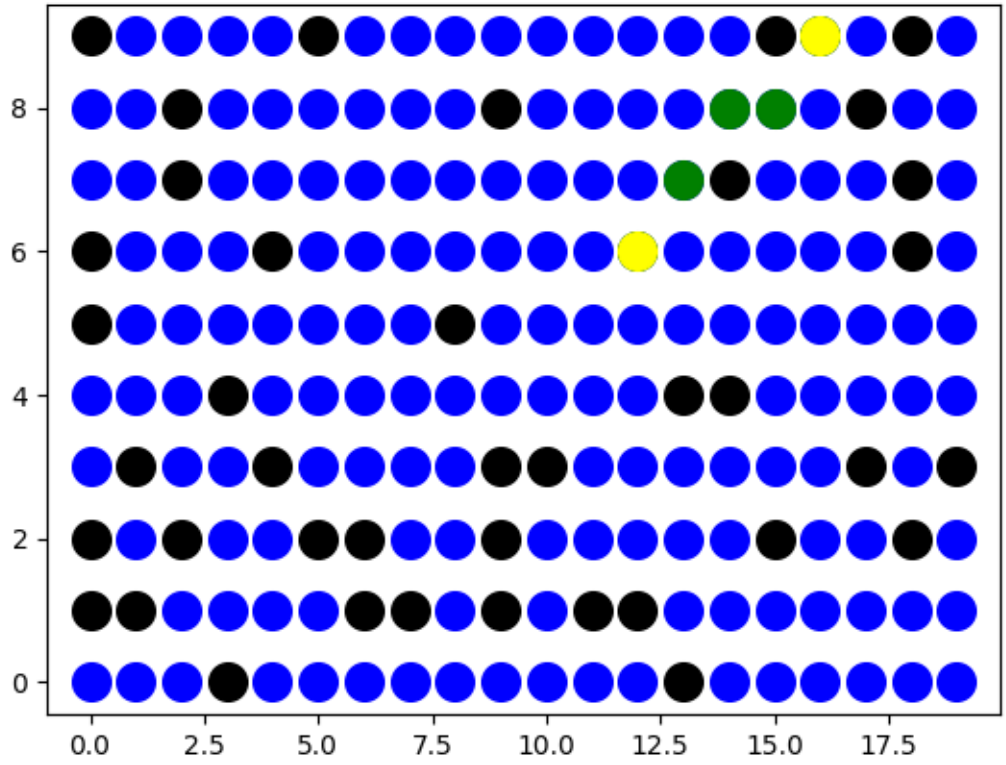


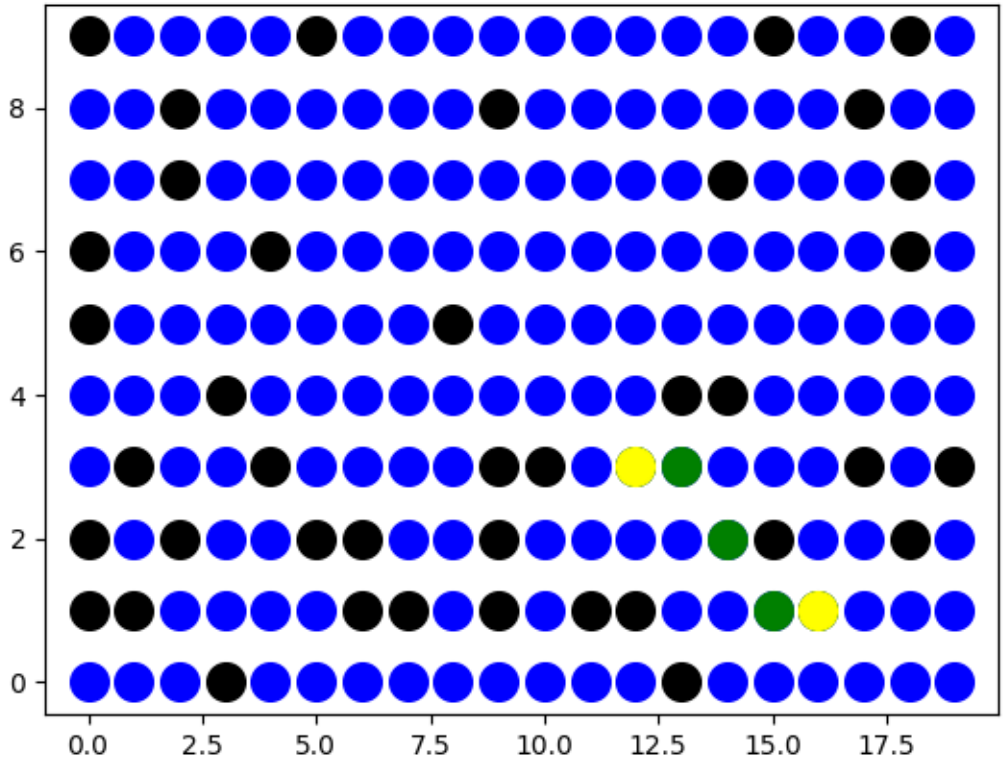
Some examples of routes:






8
6
4
2
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


8
6
4
2
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5



The average time was 12.32861328125ms.

Conclusion

As it can be seen there are numerous algorithms for solving the same task – shortest-path between two nodes. Nevertheless, each of those algorithms has some advantages.
Dijkstra's algorithm is good for evaluating distances between source and all places around. It is very effective in this case.
Bellman-Ford algorithm is also very good, but it is better than Dijkstra's approach only in case negative graph weights are concerned. Dijkstra's approach is not applicable when negative weights are used.
A* algorithm is really good for finding the path between two points. It is very simple to write and very effective.

References

https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csgraph.shortest_path.html
https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2

Appendix

My project is hosted here:
https://github.com/PatrykStronski/Algorithms_6