FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report
On the practical task number 8
"Practical analysis of 2 algorithms"

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

St. Petersburg
2021

Goal
The goal of the task is to analyze space and time complexity, used techniques end effectiveness of 2 chosen algorithms. For this purpose I chose **Edmonds-Karp** and **Preflow-Push** (also known as **Push-Relabel**) algorithms for solving the maximum flow problem in a flow network.

Formulation of the problem
The problem is to apply the two algorithms in the flow network in order to find the maximum flow. The network can represent a pipe or wire system and our aim is to find the way for the flow so that the substance from point A can reach point B at the fastest pace. The answer to to the problem is both the track the flow will take and the maximal flow value that can be achieved using this best track.

Brief theoretical part
Firstly, we need to define the network we will use. The network is a pipe or wire network with directed pipes/wires. Therefore it can be represented using a directed graph. The task is to ship the material at the fastest pace between start and destination, but without violating any capacity constraints. The maximum flow algorithms let us know on how much flow can we ship to the sink from the source, and how can the flow be distributed among the edges to do it efficiently.

The graph can be represented by $G= (V, E)$. Each edge $e \in E$ has a non-negative capacity $c(u, v)$. The graph is connected, providing many routes from start (also called source *s)* to finish (called sink *t*). On top of that, the graph is directed and provided there is an edge (v1, v2), there is no edge (v2, v1).
The flow is defined by function $f(v1, v2)$. The flow value is the product of the *f* function and it is attached to each edge. It ranges from 0 to maximum edge capacity, as it is the capacity that defines the upper bound for the flow.
The constraints for flows are the following:

$$\text{For all } v1, v2 \in V,\ 0 <= f(v1, v2) <= c(v1, v2)$$
$$\text{For all } v \in V - \{s, t\} \text{ Sum of } f(v1, v2) = \text{sum of flows } f(v2, v1)$$

Flow conservation in general needs to satisfy the property “flow in equals flow out”. This constraints simply originates from physics, as the problem’s application are networks of physical objects such as pipes.

The maximal flow is chosen under the assumption that each edge has the maximal *capacity*. The Capacity is simply the maximal amount of material the pipe can transmit. Based on that, it is possible that not the whole capacity is taken, can be less.

In case of maximal flow algorithms it is needed to also explain what is a **residual graph**. The residual graph is the graph that contains also the residual edges, which are the reverse edges to the original edges the network can have. As explained before, in a flow network only one edge can connect 2 nodes. The residual edges are used in order to correctly calculate the flow values, as well as to assess the goodness of the flow. They are also often denoted with *capacity* value equal 0. They are used in order to show how much flow has remained unused.

In case of my analysis I used two algorithms for this purpose – **Edmonds-Karp** and **push-relabel** (also known as **preflow-push**). Both algorithms return the network of residuals built upon the original network. The network of residuals is a processed input graph, which contains additional information, such as values of flow function $f(v1, v2)$ for each edge, residual edges (denoted as the edges in the backward direction to the flow the edges are directed), as well as the flow that arrives to sink *t*.

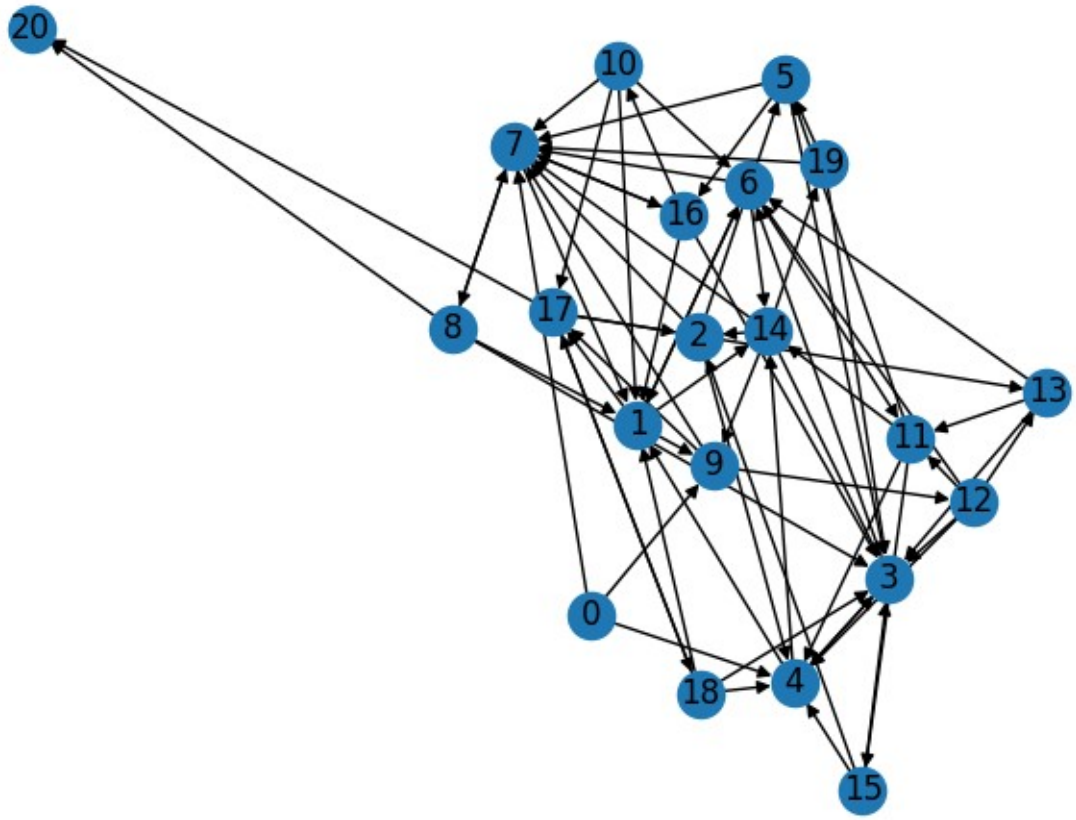
Edmonds-Karp algorithm is and extension of Ford-Fulkerson method. In general the Ford-Fulkerson method checks for the augumented paths using depth-first-search which, in this case, does not necessarily give the full results. In bread-first search the augumented paths are modified by iterating over the nodes that are adjacent to the node the algorithm wants to pass. The best solution is the solution when the maximal amount of flow can be pushed through the network. The number of best solutions can be more than one – it depends on the complexity of the network. The worst-case complexity of this algorithm is $O(|E|^2*|V|)$.

Preflow-Push algorithm, also called **Push-Relabel** uses different approach. The name preflow-push, comes from the basic idea behind the algorithm. It takes a preflow path and then modifies it by adding (pushing) or changing neighboring nodes in order to find the maximum flow value. Its worst case complexity is $O(|V|^2*SQRT(|E|))$, therefore it is considered better than Edmonds-Karp approach.

Used structures and methods

In order to create the flow network I used the assumption, that such network is a directed graph here only one edge can connect two vertices. I am creating such graph randomly. Firstly I provide the number of nodes that I would like the graph to have, and then I create the edges from the nodes, by iterating the nodes and appending the edges to the edge list. I can also specify the maximal amount of outcoming edges the vertex should have. There are also checks to prevent bidirectional edges and cycles. The graph is returned as the *networkx.DiGraph* instance, as I am basing all my program on the *networkx* basis.

The example of a graph I create randomly:

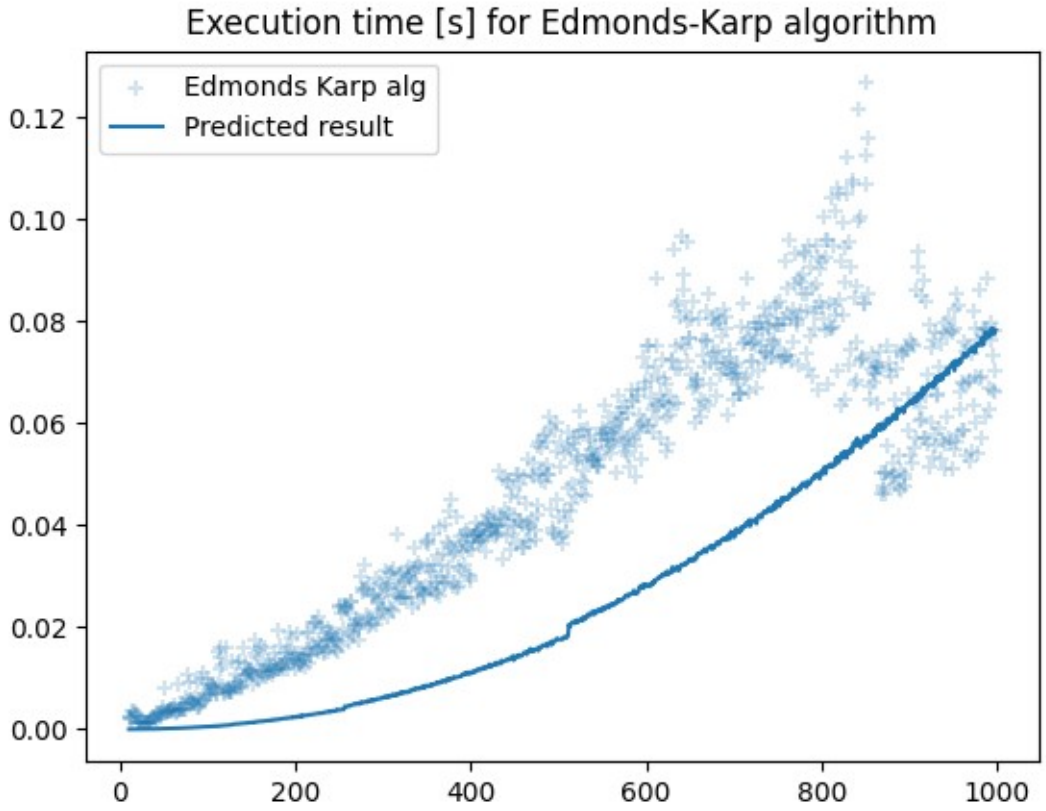


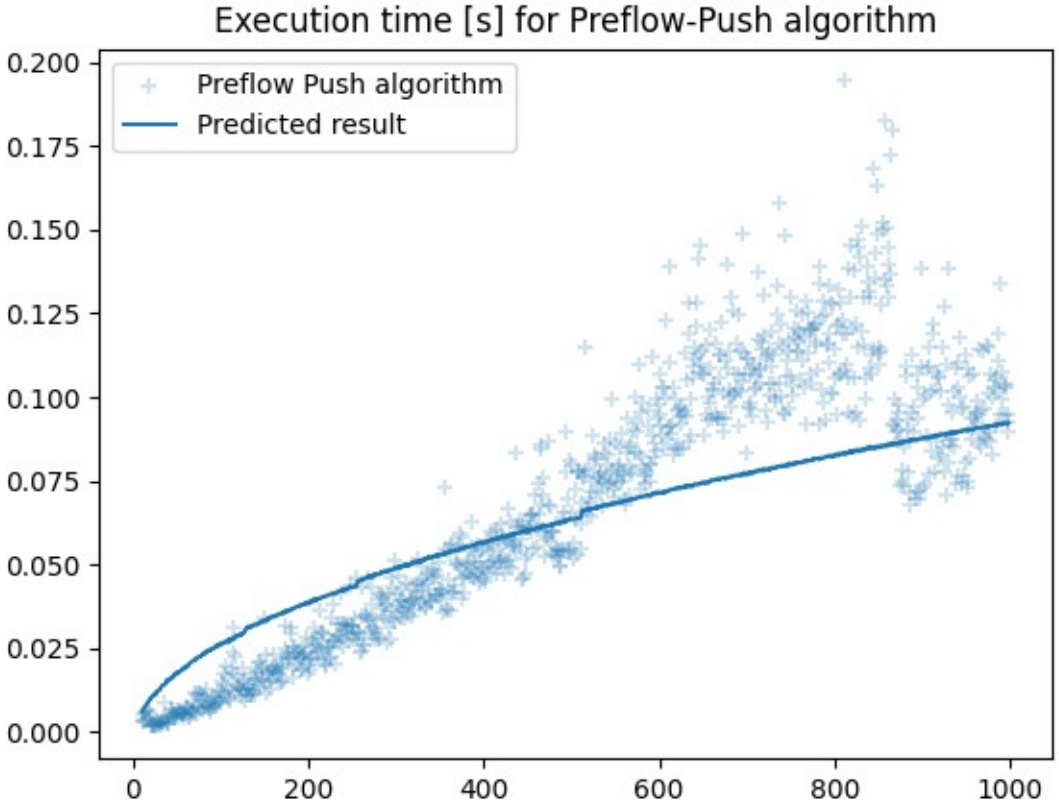
Then I am analyzing the algorithms – their execution time. I am plotting a graph to check whether the expected complexity is applicable to the realistic one. The sample time unit used for one algorithm iteration has been derived from the last function computation. Based on that the execution time here also carries some error, nevertheless we can assume it is negligibly small.

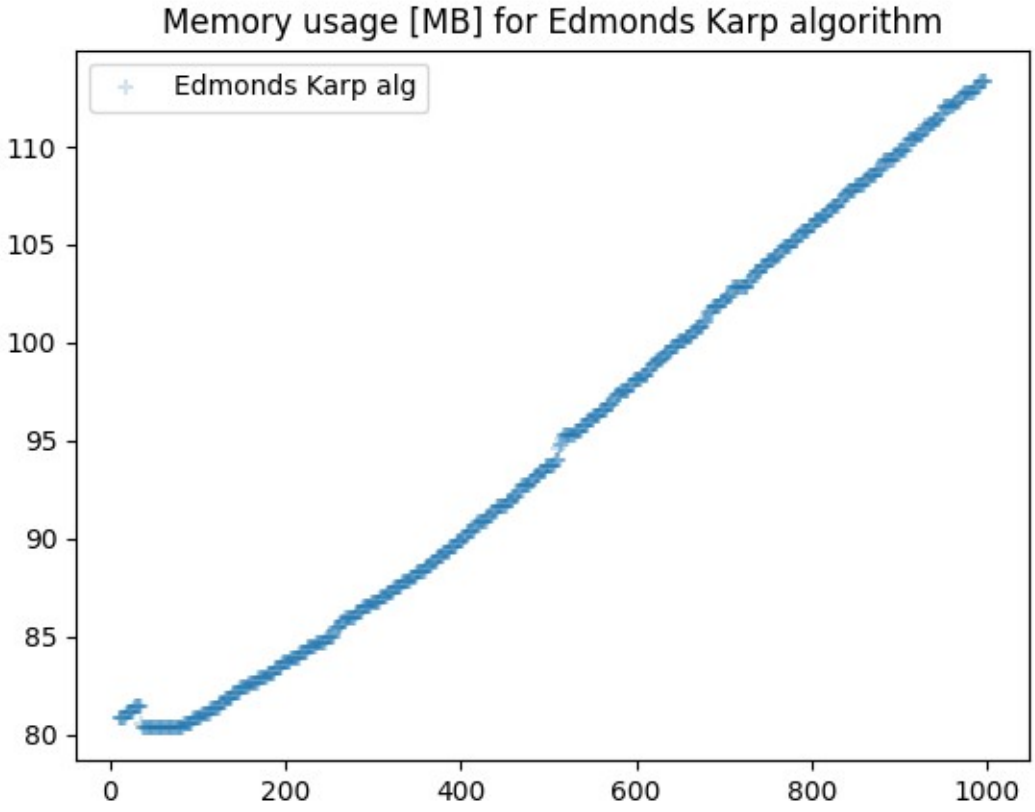
On top of that I am analyzing the memory consumption during the algorithm execution. This memory consumption is simply an estimation, as it is hard to grasp the memory precisely at the point when the algorithm finishes.

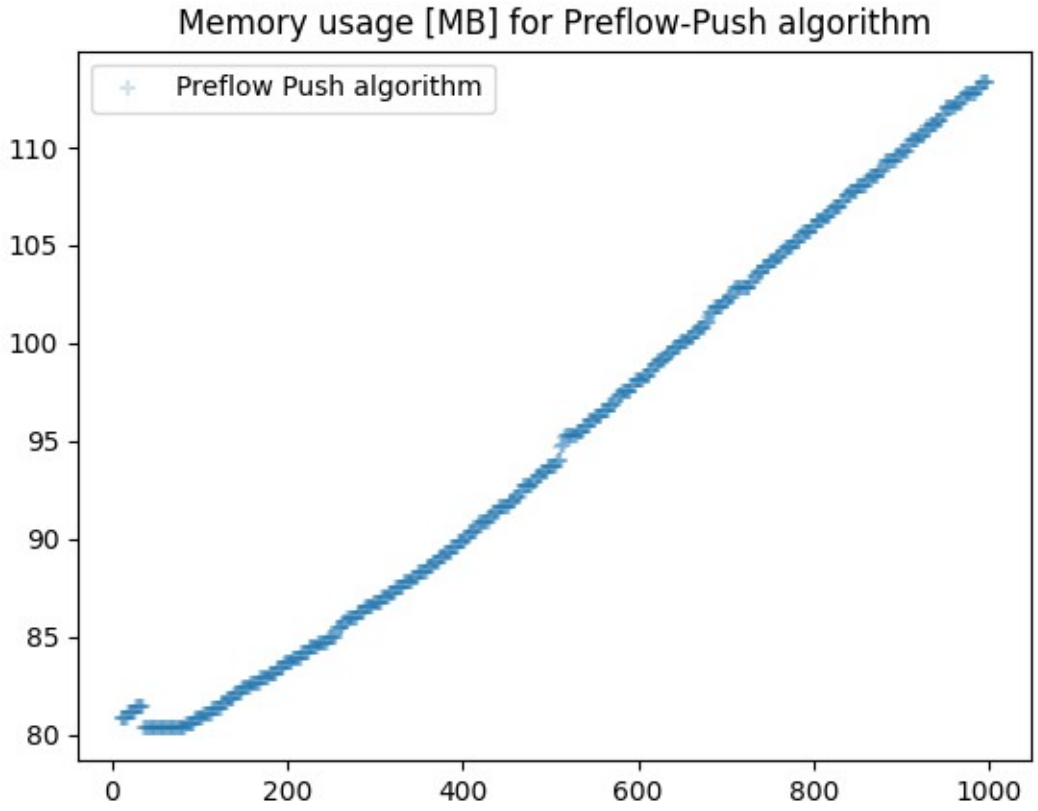
Results

I analyzed both algorithm by looking at their execution times. I analyzed graphs with amount of nodes from 10 to 1000. For each graph I repeated the algorithm calculation 5 times and calculated the mean. The mean value I treated as the actual time of algorithm execution for a particular graph.




Execution time [s] for Preflow-Push algorithm
Preflow Push algorithm
Predicted result
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
200
400
600
800
1000

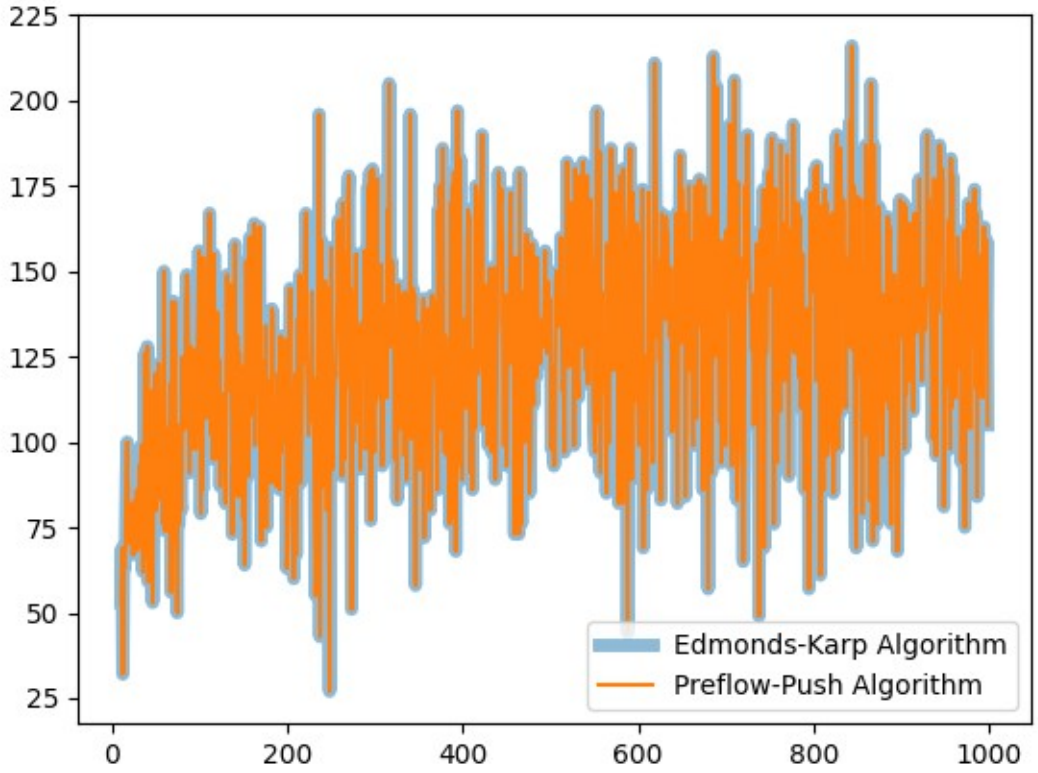

Memory usage [MB] for Edmonds Karp algorithm
Edmonds Karp alg
110
105
100
95
90
85
80
0
200
400
600
800
1000



As it can be seen on the graph, the Edmonds-Karp algorithm outperforms a bit the Preflow-Push one. This might be due to low amount of edges in the graph for example, because the number of outcoming edges is 19. Nevertheless, the algorithms have very similar execution times.

When it comes to the memory usage, it is also comparable in the graph. Nevertheless, as I mentioned, it is hard to measure exact memory needed for the algorithm execution, as Python has its own garbage collector which is launched at its own pace.

When it comes to the maximal flow values both algorithms get the same values, which can be seen on this diagram below.



Conclusion

As it can be seen there are two algorithms that do the task and receive the same good results. They solve the problem very quickly even for big networks (of about 1000 nodes). Maybe due to implementation or network specification the Edmonds-Karp algorithm renders a bit better. On top of that the given estimations are really pessimistic, sometimes the algorithms do perform better than the calculated value. On top of that, we need to take into account the current load on the machine I am running the algorithms on. Taking the mean from 5 repetitions is about to reduce this error, but still it is present.

Summing up, the flow network problem can be both solved by Edmonds-Kerp and Preflow-Push algorithms and the same results will be received within similar time and memory consumption.

References

Book: Thomas H. Cormen Charles E. Leiserson Ronald L. Rivest Clifford Stein “Introduction to Algorithms”
https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.flow.edmonds_karp.html#networkx.algorithms.flow.edmonds_karp
https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.flow.preflow_push.html#networkx.algorithms.flow.preflow_push
https://networkx.org/documentation/stable/tutorial.html#graph-attributes
https://en.wikipedia.org/wiki/Edmonds%E2%80%93Karp_algorithm
https://en.wikipedia.org/wiki/Push%E2%80%93relabel_maximum_flow_algorithm

Appendix

My project is hosted here:
https://github.com/PatrykStronski/Algorithms_8