FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION
ITMO UNIVERSITY

Report

on the practical task No. 8

“Practical analysis of advanced algorithms”

Performed by

Dmitriy Prusskiy

J4132c

Accepted by

Dr Petr Chunaev

St. Petersburg

2021

Goal

Practical analysis of advanced algorithms.

Formulation of the problem

I. Choose two algorithms (interesting to you and not considered in the course) from the above-mentioned book sections.

II. Analyse the chosen algorithms in terms of time and space complexity, design technique used, etc. Implement the algorithms and produce several experiments. Analyse the results.

Brief theoretical part

The algorithms of Kruskal and Prim were chosen to analyze. Both algorithms are greedy algorithms finding minimum spanning trees for a weighted undirected graph.

At each step, **Kruskal's algorithm** connects two vertices from different trees using the edge of minimum weight. The process continues as long as there are several not connected trees. To implement Kruskal's algorithm efficiently, one can store graph vertices in a disjoint set structure and sort graph edges by ascending weight. After connecting two vertices, one needs to union them in a disjoint set.

Asymptotic time complexity of this implementation adds up from disjoint set creating $O(V)$, then sorting graph edges ($O(E * log(E))$), then the algorithm performs $O(E)$ union and find operations on the disjoint set. Here E is the number of edges and V is the number of vertices in the graph. Total asymptotic time complexity is $O((E + V) * \alpha(V) + E * log(E))$, where α is the inverse of the Ackermann function, which grows very slowly. Because we assume that the graph is connected, so $V - 1 \leq E \leq V^2$, and $\alpha(V) = O(log(V)) = O(log(E))$ total time complexity can be reduced to $O(E * log(E)) = O(E * log(V))$.

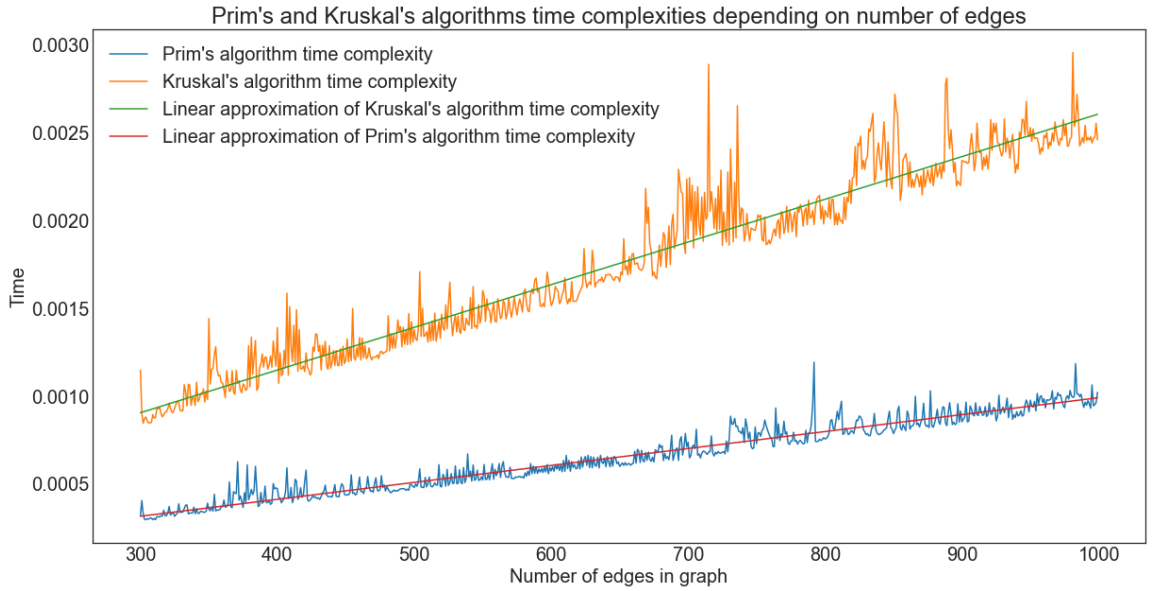
At each step, **Prim's algorithm** adds an edge of minimum weight to the current tree, provided that after adding this edge, the tree remains a tree. The process continues as long as there are vertices that do not belong to the constructed tree. In order to implement Prim's algorithm efficiently, one can store not visited vertices in a binary min-heap.

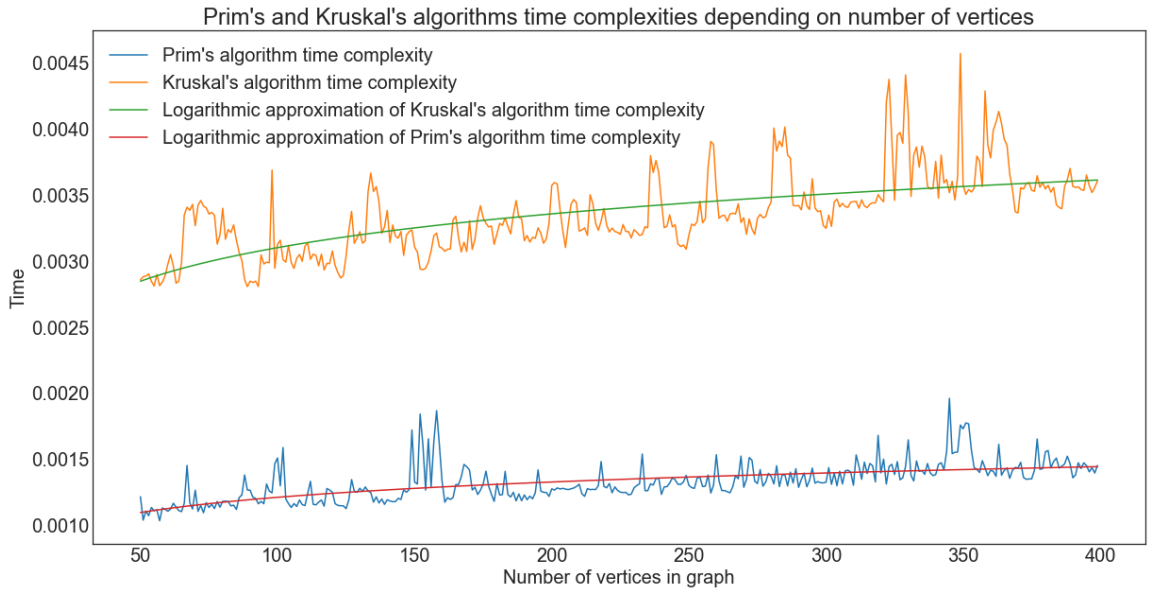
Asymptotic time required for effective implementation of Prim's algorithm adds up from creating binary min-heap $O(V)$, then on each of the V steps the algorithm pops the minimum element from heap $O(log(V))$, then push from heap $O(log(V))$ in inner loop, which will execute O(E) times altogether, since the sum of the lengths of all adjacency lists is 2*E. Total asymptotic time complexity is $O(V + V * log(V) + E * log(V)) = O(E * log(V))$.

Results

Algorithms were implemented using Python 3.8.5 programming language. For efficient implementation of Kruskal's algorithm a disjoint set structure was implemented, for Prim's algorithm - a *heapq* module from Python was used.

To check the correspondence between the theoretical and real time complexity of the algorithms, the running time of the algorithms was measured on graphs with a constant number of vertices and a changing number of edges, and then with a constant number of edges and a changing number of vertices. Below are results of these measurements along with corresponding approximations.

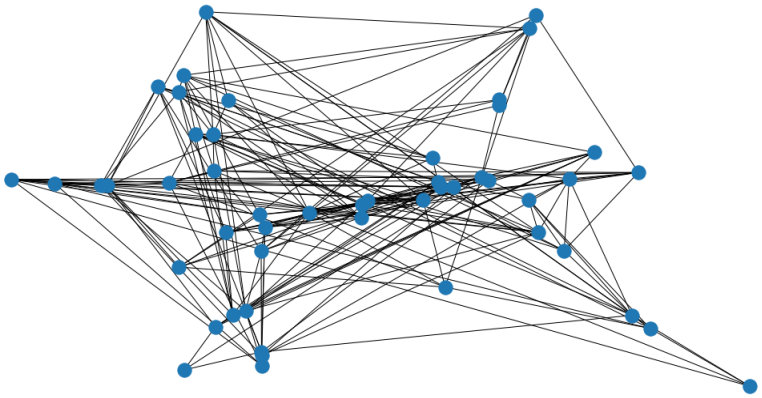




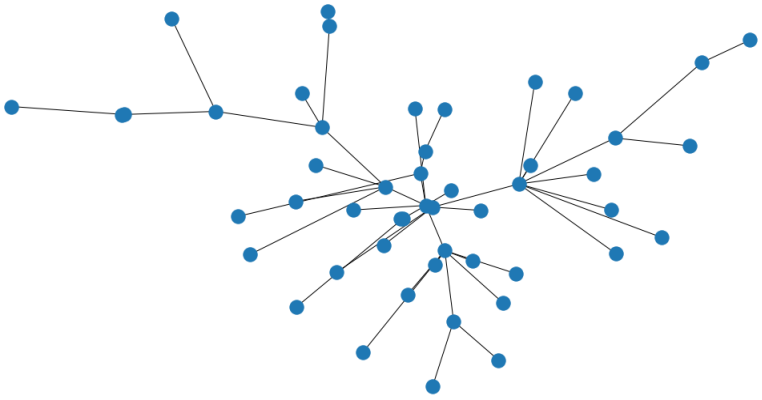
As can be seen from visualisations, real time complexities of these algorithms coincide with theoretical time complexities. Execution time depends linearly on the number of edges and logarithmically on the number of vertices.

In addition, it can be seen that Kruskal's algorithm takes almost 3 times longer than Prim's algorithm. However, Kruskal's algorithm in most cases gets better results than Prim's. Below are the results of both algorithms on a graph with 50 vertices and 200 edges. Kruskal's algorithm gets a spanning tree with 4 times less weight than Prim's algorithm. Choosing another starting vertex for Prim’s algorithm improves the result to weight 25.

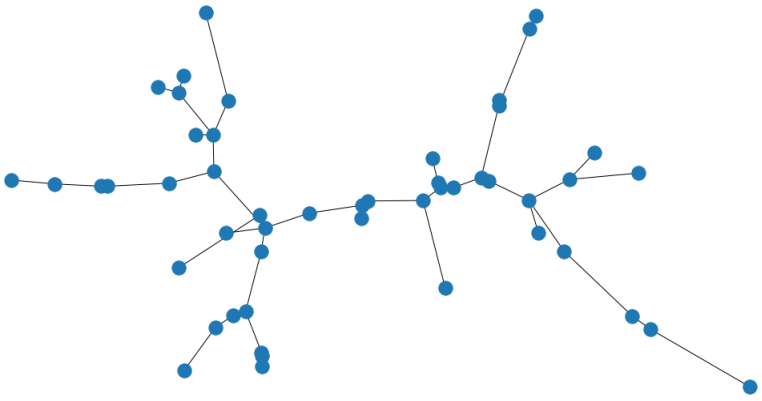
Original graph



Minimum spanning tree obtained using the Kruskal's algorithm
sum weight of the tree = 7.243218812215979



Minimum spanning tree obtained using the Prim's algorithm
sum weight of the tree = 28.706529455370553



Conclusions

The goal of this study was to analyze advanced algorithms and compare them. Kruskal's and Prim's algorithms were efficiently implemented and analyzed in terms of time complexity and algorithms performance. Analysis of results shows that Kruskal's algorithm takes 3 times more time than Prim's algorithm, but gets about 4 times better results on average.

Appendix

Source code can be found at https://github.com/T1MAX/itmo_algorithms/tree/main/task_8