FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION

OF HIGHER EDUCATION

ITMO UNIVERSITY

Report

on the practical task No. 8

“Practical analysis of advanced algorithms”

Performed by

*Vdovkina Sophia*

*Syrchenko Arina*

*Academic group: J4133c*

Accepted by

Dr Petr Chunaev

St. Petersburg

2022

**Goal**

Practical analysis of advanced algorithms.

**Formulation of the problem**

1. Choose two algorithms (interesting to you and not considered in the course) from the above-mentioned book sections.
2. 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**

Given an undirected and connected graph , a spanning tree of the graph *G* is a tree that spans *G* (that is, it includes every vertex of *G*) and is a subgraph of G (every edge in the tree belongs to G). The cost of the spanning tree is the sum of the weights of all the edges in the tree. Minimum spanning tree (MST) is the spanning tree where the cost is minimum among all the spanning trees. There also can be many minimum spanning trees.

Kruskal’s and Prim’s algorithms are two minimum-spanning-tree algorithms, that elaborate on the generic method. They each use a specific rule to determine a safe edge.

In Kruskal’s algorithm, the set *A* is a forest whose vertices are all those of the given graph. The safe edge added to *A* is always a least-weight edge in the graph that connects two distinct components. Kruskal's algorithm follows greedy approach as in each iteration it finds an edge that connects any two trees in the forest and has least weight and add it to the growing spanning tree.

In Kruskal's algorithm for MST, firstly, the edges of the given graph are sorted in ascending order. While making the tree, the edge with the least weight is added first, and such that no cycles are formed in the tree. This step is repeated till the number of edges in the MST becomes one less than the number of vertices *V*. That is, *E* becomes equal to *V-1*. Therefore, Kruskal's algorithm consists of the following main parts:

1. Sort the edges in the given graph in an ascending order.
2. Start making a spanning tree by adding the edge with the least weight first and check to make sure no cycles are formed.
3. Repeat the previous step till the number of edges in the spanning tree is one less than the number of vertices in the main graph.

The best data structures to use in this algorithm are a disjoint sets, that are sets whose intersection is the empty set so it means that they don't have any element in common. In Kruskal’s algorithm, most time-consuming operation is sorting because the total complexity of the Disjoint-Set operations will be  which is the overall time complexity of the algorithm.

While in Kruskal's algorithm an *edge* is added at each iteration, in Prim's algorithm a *vertex* is added to the growing spanning tree, so the set A forms a single tree. The safe edge added to A is always a least-weight edge connecting the tree to a vertex not in the tree. So, Prim’s Algorithm also use Greedy approach to find the minimum spanning tree.

The algorithm starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST. Thus, steps of Prim’s algorithm are the following:

1. Pick any vertex of the given network.
2. Choose the shortest weighted edge from this vertex.
3. Choose the nearest vertex *u* that is not included in the solution and has minimum weight.
4. Repeat steps 1-4 till all the vertices are visited, forming a minimum spanning tree.

The time complexity of the Prim’s Algorithm is because each edge is inserted in the priority queue only once and insertion in priority queue take logarithmic time. The asymptotic running time of Prim’s algorithm can be improved by using Fibonacci heaps to . A Fibonacci heap is a collection of trees satisfying the minimum-heap property, that is, the key of a child is always greater than or equal to the key of the parent. This implies that the minimum key is always at the root of one of the trees.

**Results**

Kruskal’s and Prim’s algorithms were implemented during this practical task.

# *Kruskal’s algorithm*

In order to implement Kruskal’s algorithm union-find data structure was created, it is also called disjoint sets, which is a crucial data structure for this algorithm.

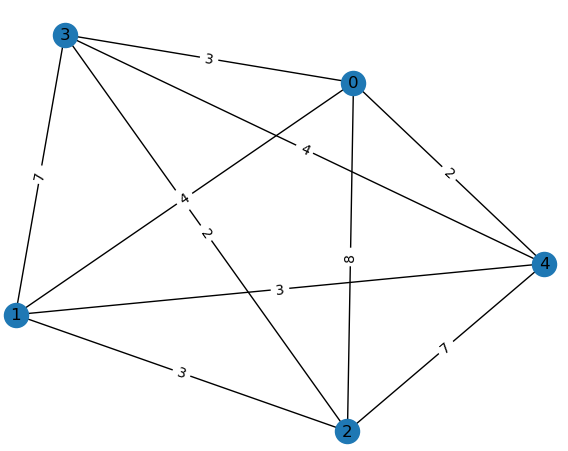
Each edge is stored as a set containing the two vertices and each of these sets contains elements. The disjoint set data structure then keeps track of all these sets.

Union by rank and find functions were created to maintain disjoint sets. Find function helps to find a particular's vertex set in the disjoint set structure. Using this function, if it is found that two vertices are in the same set, we avoid using the union function and don't merge the two sets as it implies overlapping elements were found. Thus, we eliminate cycles.

When union by rank us implemented, each vertex's subtrees and their heights are kept track of. When two sets don't contain overlapping elements and each element is part of a unique set, the two sets are merged into a single set such that the tree with greater height becomes the parent of the tree with the smaller height. This way the height of the overall tree structure is and it makes traversing and finding each vertex's set and parent node much easier and therefore faster.

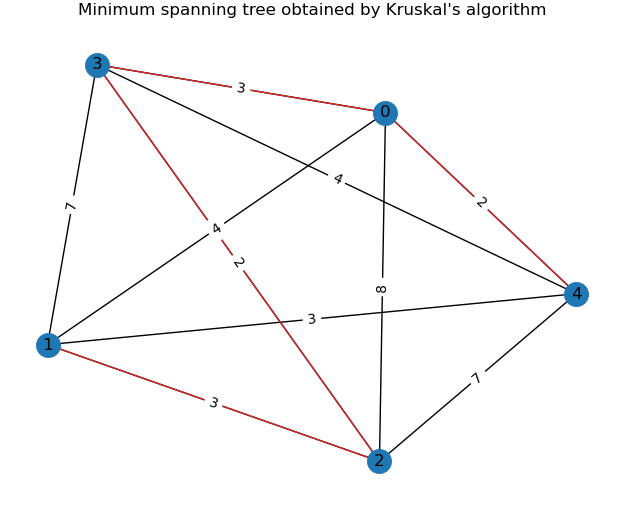
As the first step of the algorithm, edges of a graph are sorted in ascending order. Then a spanning tree is made by adding the edge with the least weight, while making sure no cycles are formed. Edges are added until the number of edges in the spanning tree is one less than the number of vertices in the main graph.

Complete graph with 5 nodes was generated in order to test algorithms, it is shown in picture 1.



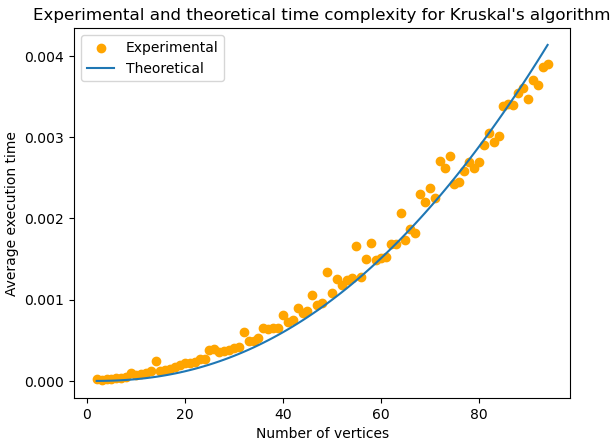
Picture 1 – complete graph

Then Kruskal’s algorithm was tested on the graph above. The visualization of obtained result is shown on picture 2.



Picture 2 – result visualization for Kruskal’s algorithm

After visualization, time complexity was tested. In order to do this, experiments were run on complete graphs that have 2 to 95 vertices, each graph was run with the algorithm 20 times to obtain mean execution time. Time complexity visualization graph can be seen in picture 3, where experimental complexity is compared with theoretical complexity.



Picture 3 – time complexity analysis for Kruskal’s algorithm

Implementation of union by rank function helps decrease the amount of time spent traversing the tree to find the root of a vertex and subset of the disjoint set structure it's in. Since the lower nodes are directly connected with the root, the height of the tree grows at a slow pace than . Therefore, the runtime of the function turns out to be a constant in practice: . The total running time of Kruskal’s algorithm is . Observing that it can be concluded that and the running time of Kruskal’s algorithm can be restated as Experimental running time seems to be close to the theoretical one, taking into account implementation specifics.

The Disjoint set structure, requires to keep track of all vertices in the beginning and the respective subsets and to keep track of all the valid sorted edges to be included in the final MST. Therefore, the total space complexity turns out to be .

# *Prim’s algorithm*

The first step in Prim’s algorithm is initializing the key values of the root as 0 and key values of other vertices as infinity. This initialization takes . After the first step, the neighbors of this vertex are found and decrease key operation is executed which takes Then the root node is deleted (takes ) and the minimum weighted edge is chosen. The above procedure is repeated till all vertices are visited.

In Prim’s two sets are maintained, one set contains list of vertices already included in MST, other set contains vertices not yet included. With adjacency list representation, all vertices of a graph can be traversed in time using Breadth First Search. The idea is to traverse all vertices of graph using BFS and use a Min Heap to store the vertices not yet included in MST. Min Heap is used as a priority queue to get the minimum weight edge from the cut. Min Heap is used as time complexity of operations like extracting minimum element and decreasing key value is in Min Heap.

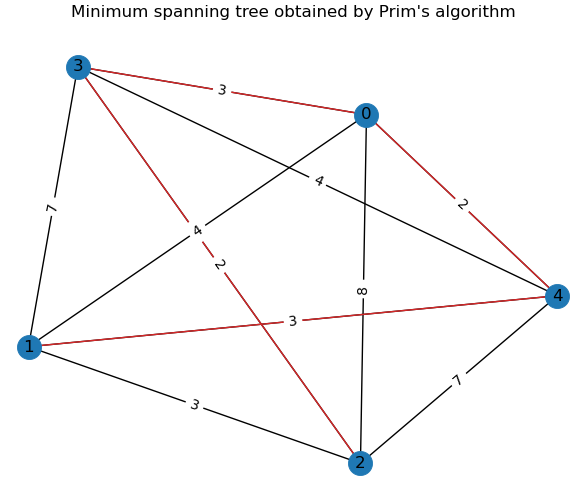
Min Heap is used in a following manner:

1. Min Heap of size V is created. Every node of min heap contains vertex number and key value of the vertex.
2. Min Heap is initialized with first vertex as root (the key value assigned to first vertex is 0). The key value assigned to all other vertices is INF (infinite).
3. While Min Heap is not empty:

* Extract the min value node from Min Heap. Let the extracted vertex be .
* For every adjacent vertex *v* of *u*, check if *v* is in Min Heap (not yet included in MST). If *v* is in Min Heap and its key value is more than weight of *u-v*, then update the key value of *v* as weight of *u-v*

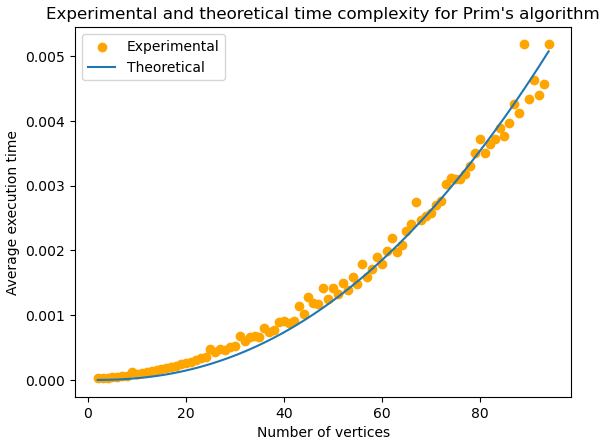
Fibonacci Heaps is a more sophisticated implementation of heaps. They have some advantages, which greatly reduce their operation cost. In fact, all operations where deletion of an element is not involved, they run in . However, due to the complicated nature of Fibonacci Heaps, various overheads in maintaining the structure are involved which increase the constant term in the order. In this work normal min heap was implemented

Prim’s algorithm was tested on the same graph as Kruskal’s algorithm. Result turned out to be the same. The visualization of obtained result is shown on picture 4.



Picture 4 – result visualization for Prim’s algorithm

In terms of time complexity Prim’s algorithm was tested in the same manner as Kruskal’s algorithm. Time complexity visualization graph can be seen in picture 5.



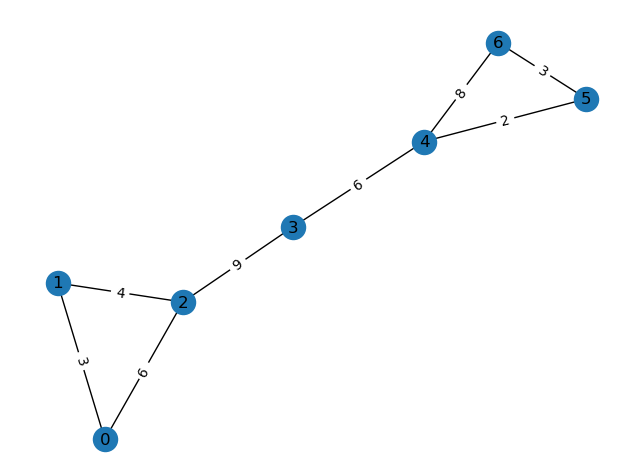
Picture 5 – time complexity analysis for Prim’s algorithm

Two nested while loops in the algorithm seem to take . But, the statements in inner loop are executed times and the inner loop operation takes time. So, overall time complexity is which is . Thus, in this method, the time complexity of Prim's algorithm is . It makes it the same with Kruskal’s algorithm.

In Prim's algorithm, the heap holds V elements. An array, that stores visited vertices takes up space V. Array, that stores the path and adjacency list also take up V space each. Adding all these will give us a total space complexity of Prim's algorithm

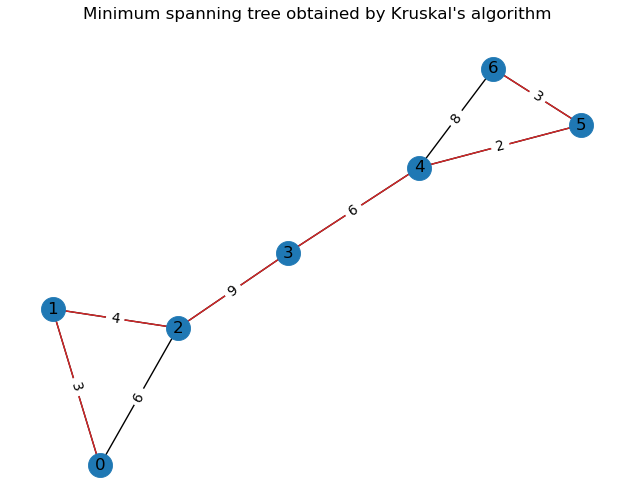
# Comprasion

In order to experiment more with the algorithms let’s test them on another type of graphs. For example, Barbell graph, that is shown on picture 6.

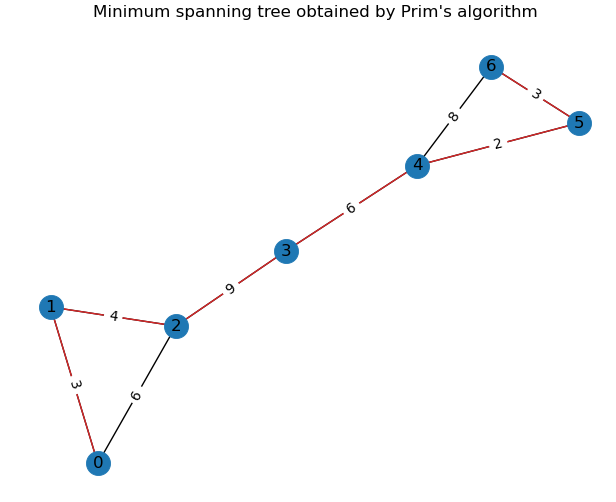


Picture 6 – the Barbell graph

Obtained results for Kruskal’s and Prim’s algorithms are shown on pictures 6-7 respectively.



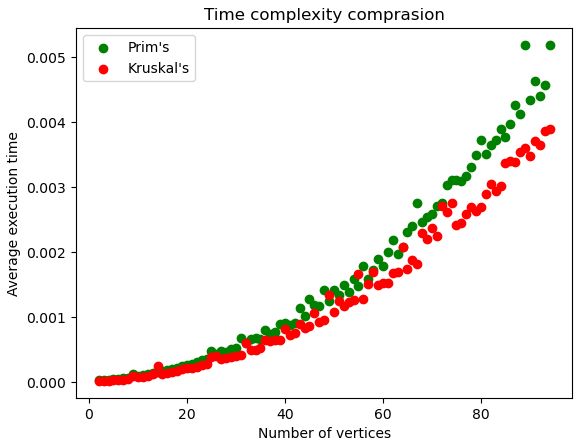
Picture 7 – result visualization for Kruskal’s algorithm on the Barbell graph



Picture 8 – result visualization for Prim’s algorithm on the Barbell graph

So, both algorithms work in a right way.

The theoretical time complexity is the same for both algorithms, experimental complexity comparison is shown on picture 9.



Picture 9 – average execution time comparison

Kruskal’s seem to be a little faster on larger number of vertices, that can be explained with the implementation on python, while also falling within the margin of error.

It can be stated that Prim’s nevertheless might have smaller space complexity.

Examining the main flow, structure of the algorithms, several differences were obtained:

* In Prim's algorithm a vertex is added at each iteration, while in Kruskal's algorithm an edge is added.
* Prim’s algorithm starts to build the MST from any vertex in the graph, while Kruskal’s algorithm, starts to build the MST from the vertex carrying minimum weight in the graph.
* Kruskal’s algorithm traverses one node only once, while Prim’s can traverse one node many times.
* Prim’s algorithm generates a tree, which is a connected component, so works only on connected graph, Kruskal’s algorithm generates forest, so it can work with disconnected components

Also, some papers observe that Prim’s algorithm works better with dense graphs, while Kruskal’s algorithm operates better with sparse graphs.

**Conclusion**

Twominimum-spanning-tree algorithms Kruskal’s and Prim’s algorithms were implanted in this task. Both algorithms were programmed using python language, chosen techniques led to the same time complexity of both algorithms. Experimental time complexities for both algorithms were obtained and compared with theoretical time complexities, the results showed that average execution time coincided both within the algorithms and between them. Algorithms were tested on different graph structures. Also, comparing analysis of structure of the algorithms was conducted. Optimal data structures and design techniques were analyzed and implemented.

**Appendix**

Project file is available on <https://github.com/sophia-vdovkina/Analysis-and-development-of-algorithms/blob/main/Task%208/Task8.ipynb>