**Ministerul Educaţiei și Cercetării al Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**REPORT**

Laboratory Work nr.7

*at Algorithms Analysis*

Elaborated by:

st. gr. FAF-211 Hariton Dan

Verified by:

asist. univ. Fiștic Cristofor

Chişinău - 2023

**ALGORITHM ANALYSIS**

**Subject:** Empirical analysis of algorithms: Prim and Kruskal.

**Tasks:**

1. Implement the algorithms listed below in a programming language
2. Establish the properties of the input data against which the analysis is performed
3. Choose metrics for comparing algorithms
4. Perform empirical analysis of the proposed algorithms
5. Increase the number of nodes in graph and analyze how this influences the algorithms. Make a graphical presentation of the data obtained.
6. Make a conclusion on the work done.

**Establish Comaparation:**

We will compare these 2 algorithms we used adjacency matrices.

**Comparison Metric:**

The comparison metric for this laboratory work will be considered the time of execution of each algorithm (T(n))

**IMPLEMENTATION**

**Prim:**

Prim's algorithm starts with a single vertex and gradually grows the minimum spanning tree by adding edges to connect the vertices with the smallest edge weights until all the vertices are connected. Prim's algorithm is a greedy algorithm that builds the tree incrementally, always choosing the next edge that connects the tree to the vertex with the smallest weight.

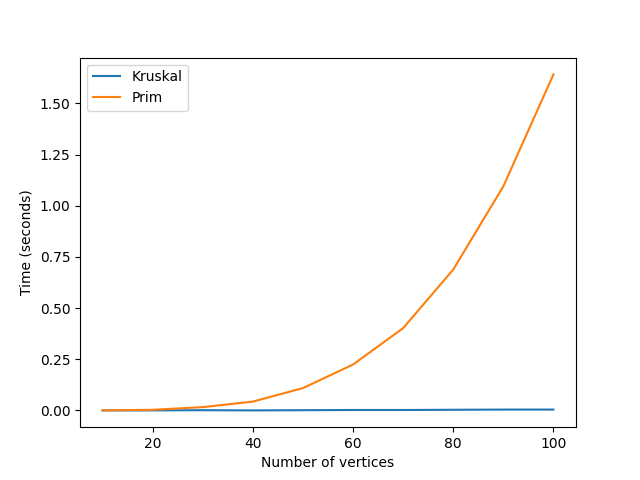
def prim(graph):  
 mst = set()  
 visited = {0}  
 while len(visited) < graph.num\_vertices:  
 candidates = []  
 for vertex in visited:  
 for neighbor in graph.get\_neighbors(vertex):  
 if neighbor not in visited:  
 candidates.append((vertex, neighbor))  
 edge = min(candidates, key=lambda uv: graph.weights[uv])  
 mst.add(edge)  
 visited.add(edge[1])  
 return mst

## Kruskal:

## Kruskal's algorithm sorts all the edges by weight and adds the edges to the minimum spanning tree one by one, as long as they do not form a cycle. Kruskal's algorithm, on the other hand, is also a greedy algorithm, but it selects edges based on their weight, not on the vertices they connect.

def kruskal(graph):  
 mst = set()  
 forest = {vertex: {vertex} for vertex in graph.vertices}  
 edges = list(graph.weights.keys())  
 edges.sort(key=lambda uv: graph.weights[uv])  
 for uv in edges:  
 u, v = uv  
 if forest[u] != forest[v]:  
 mst.add(uv)  
 old\_tree = forest[u]  
 new\_tree = forest[v]  
 for w in old\_tree:  
 forest[w] = new\_tree  
 return mst

**Results:**

****

**Conclusion:**

The proposed work aimed to analyze the time-based and theoretical complexities of the Prim and Kruskal algorithms. We used two different algorithms to determine their temporal complexity and highlight the most efficient algorithm that will display the desired result using the least time that passed by. From the analysis, it can be concluded that in our dataset, the Prim algorithm is slower than the Kruskal algorithm in finding the minimum spanning tree in a graph. This is due to the fact that Prim's algorithm is a vertex-based algorithm that starts with a single vertex and gradually grows the minimum spanning tree, while Kruskal's algorithm is an edge-based algorithm that sorts all the edges by weight and adds the edges to the minimum spanning tree one by one.

On the other hand, the Prim algorithm is more efficient than the Kruskal algorithm in certain scenarios. If the graph is dense with many edges, then the Prim algorithm would be more efficient as it only processes the vertices that are needed to build the minimum spanning tree. If the graph is sparse with few edges, then the Kruskal algorithm would be more efficient as it sorts all the edges by weight and processes them one by one, without considering the vertices.

In conclusion, both Prim and Kruskal algorithms are very efficient and the choice of algorithm to use depends on the specific problem at hand and the characteristics of the graph. If the goal is to find the minimum spanning tree in a sparse graph with few edges, then the Kruskal algorithm is the better choice in our dataset. If the graph is dense with many edges, then the Prim algorithm may be more efficient.

**Link to GitHub**: <https://github.com/haritondan/AA-Labs>