**Ministerul Educaţiei și Cercetării al Republicii Moldova Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

Laboratory work 5:

Prim and Kruskal Algorithms

Elaborated:

st. gr. FAF-221 Gavriliuc Tudor

Verified:

asist. univ. Fiștic Cristofor

prof. univ. Andrievschi-Bagrin Veronica

Chişinău - 2024

**Objective:**

Study and analyze different graph traversing algorithms.

**Tasks:**

1. Study the greedy algorithm design technique
2. To implement in a programming language algorithms Prim and Kruskal.
3. Empirical analyses of the Kruskal and Prim
4. Increase the number of nodes in graphs and analyze how this influences the algorithms. Make a graphical presentation of the data obtained

Theoretical Notes:

Empirical analysis provides an alternative approach to understanding the efficiency of algorithms when mathematical complexity analysis is impractical or insufficient. This method proves beneficial in various scenarios:

1. Initial Insights: It offers preliminary insights into an algorithm's complexity class, aiding in the understanding of its efficiency characteristics.
2. Comparative Analysis: It facilitates the comparison of multiple algorithms tackling the same problem, allowing for informed decisions regarding efficiency.
3. Implementation Comparison: Empirical analysis enables the comparison of different implementations of the same algorithm, providing insights into which may perform better in practice.
4. Hardware-specific Evaluation: It helps in assessing an algorithm's efficiency on a particular computing platform, taking into account hardware constraints and capabilities.

The empirical analysis of an algorithm typically involves the following steps:

Establishing Analysis Goals: Clearly define the objectives and scope of the analysis.

1. Choosing Efficiency Metrics: Select appropriate metrics, such as the number of operations executed or the execution time, based on the analysis goals.
2. Defining Input Data Properties: Determine the characteristics of the input data relevant to the analysis, including data size or specific attributes.
3. Implementation: Develop the algorithm in a programming language, ensuring it accurately reflects the intended logic.
4. Generating Input Data Sets: Create multiple sets of input data to cover a range of scenarios and edge cases.
5. Execution and Data Collection: Execute the program for each input data set, recording relevant performance metrics.
6. Data Analysis: Analyze the collected data, either by computing synthetic quantities like mean and standard deviation or by plotting graphs to visualize the relationship between problem size and efficiency metrics.
7. The choice of efficiency measure depends on the analysis's objectives. For instance, if assessing complexity class or verifying theoretical estimates, counting the number of operations may be suitable. Conversely, if evaluating algorithm implementation behavior, measuring execution time becomes more relevant.

8. Post-execution, recorded results undergo analysis. This involves computing statistical measures or plotting graphs to visualize the algorithm's performance characteristics in terms of problem size and efficiency metrics. Such analyses aid in making informed decisions regarding algorithm selection and optimization strategies.

**Introduction:**

Kruskal and Prim algorithms are fundamental techniques used in the field of graph theory for solving the minimum spanning tree (MST) problem. Both algorithms aim to find the smallest set of edges that connects all vertices in a weighted undirected graph without forming any cycles, thereby creating a spanning tree with the minimum possible total edge weight.

Kruskal's algorithm, developed by Joseph Kruskal in 1956, follows a greedy approach by iteratively selecting the smallest edge from the set of available edges that does not create a cycle when added to the growing spanning tree. It operates efficiently by maintaining a forest of disjoint sets, where each component initially consists of a single vertex. As the algorithm progresses, it merges these components until all vertices belong to the same tree.

In contrast, Prim's algorithm, proposed by Czech mathematician Vojtěch Jarník in 1930 and later independently rediscovered by Robert C. Prim in 1957, also employs a greedy strategy but operates from a single starting vertex. It continuously expands the spanning tree by adding the smallest-weight edge that connects a vertex already included in the tree to a vertex not yet included. This process gradually grows the MST until it spans all vertices in the graph.

Both Kruskal's and Prim's algorithms guarantee the construction of a minimum spanning tree, but they differ in their implementation details and performance characteristics. Kruskal's algorithm typically utilizes a priority queue or a sorting mechanism to efficiently select edges, while Prim's algorithm often employs a data structure such as a priority queue or a Fibonacci heap to manage candidate edges efficiently.

## **Comparison Metric:**

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

## **Input Format:**

As input, each algorithm will receive 8 series of numbers of nodes 4, 8, 16, 32, 64, 128, 256, ,512.

Next, using this numbers of nodes, it will be generated randomly graphs with that amount of nodes.

**Implementation**

All algorithms will be implemented in their naïve form in python an analyzed empirically based on the time required for their completion. While the general trend of the results may be similar to other experimental observations, the particular efficiency in rapport with input will vary depending on memory of the device used.

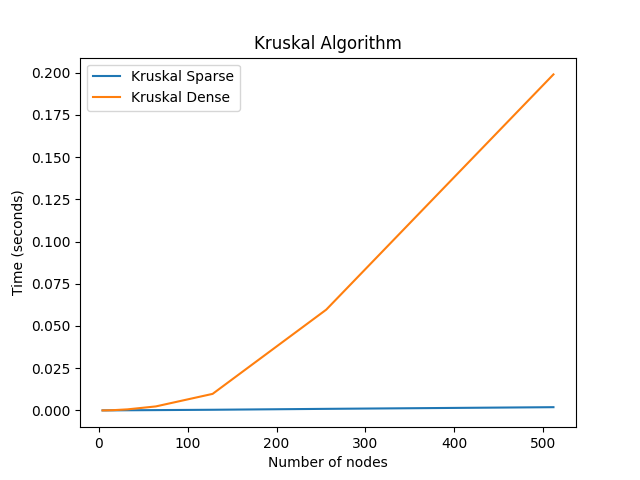
Kruskal Algorithm:

def kruskal(self, graph):  
 result = [] # To store the resultant MST  
 graph\_edges = self.get\_edges(graph) # Retrieve all edges from the graph  
 graph\_edges.sort(key=lambda x: x[2]) # Sort edges by weight  
  
 parent = {node: node for node in graph} # Initialize parent for union-find  
  
 # Process smallest edges and check for cycle formation  
 for u, v, w in graph\_edges:  
 if self.find(parent, u) != self.find(parent, v):  
 result.append([u, v, w])  
 self.union(parent, u, v)  
  
 return result



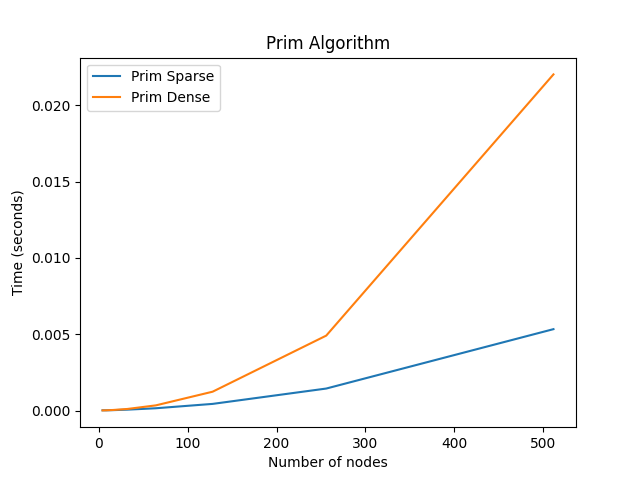
**Figure 1. Time table for algorithms**

As observed in the table, Prim algorithm is faster than Kruskal in all cases. For a small number of nodes, either the graph is sparse of dense, the algorithms are executed almost in the same time. But as the number of nodes increase, the time also increase considerably for Kruskal.

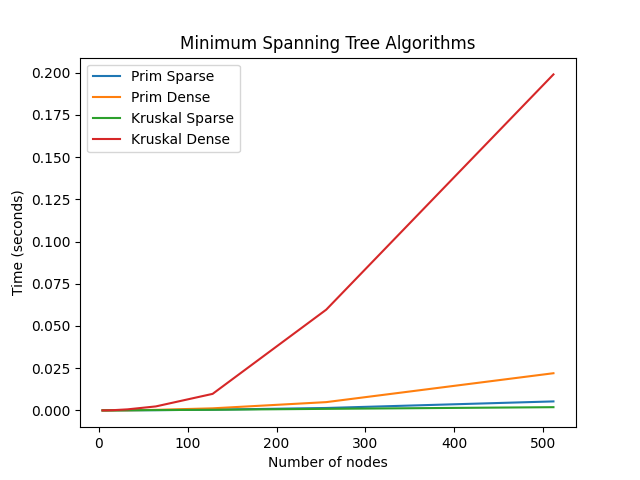
 **Figure 2. Graph for Kruskal algorithm**

Prim:

def prim(self, graph):  
 mst = [] # To store the resultant MST  
 keys = {node: float('inf') for node in graph} # Priority queue for vertices  
 keys[0] = 0 # Start from vertex 0  
 parent = {node: None for node in graph} # Parent dictionary to store MST  
  
 # While there are vertices not yet included in MST  
 while keys:  
 u = min(keys, key=keys.get) # Vertex with the smallest key value  
 del keys[u] # Remove vertex from queue  
  
 # For each adjacent vertex v, check if it should be updated  
 for v in graph[u]:  
 if v in keys and graph[u][v] < keys[v]:  
 keys[v] = graph[u][v]  
 parent[v] = u  
  
 # Create MST list from parent dictionary  
 for v, p in parent.items():  
 if p is not None:  
 mst.append([p, v, graph[p][v]])  
  
 return mst

**Figure 3. Graph for Prim**

**Conclusion**



The analysis of Prim's and Kruskal's algorithms reveals their crucial roles in effectively addressing the minimum spanning tree (MST) problem within graph theory. Both methods provide robust solutions for constructing a spanning tree with the minimal sum of edge weights, but they differ significantly in their implementation techniques and performance traits. Kruskal's algorithm adopts a greedy approach by selecting the smallest edges first, ensuring the tree expands without forming cycles and encompasses all vertices. It leverages disjoint sets and methodical union operations, making it a straightforward and efficient choice for computing the MST. Conversely, Prim's algorithm also utilizes a greedy strategy but starts from a specific vertex, incrementally building the MST by adding the lowest weight edge that connects to the tree without closing a cycle. This technique ensures that the tree is both connected and acyclic from the beginning to the end of its construction. Kruskal's algorithm typically performs better on sparse graphs with many edges, while Prim's algorithm is more effective on denser graphs or when initiating from a particular vertex is beneficial. Moreover, the efficiency of Prim's algorithm can be enhanced through the use of advanced data structures like priority queues or Fibonacci heaps, which can optimize its performance under certain conditions. Choosing between Prim's and Kruskal's algorithms depends on the graph's specific attributes and the precise needs of the problem. A deep understanding of these algorithms' details, strengths, and limitations allows practitioners to select the most appropriate method for solving MST challenges effectively in various practical scenarios, such as network design, optimization, logistical planning, and resource management.