**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 CalugareanuAna

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:**

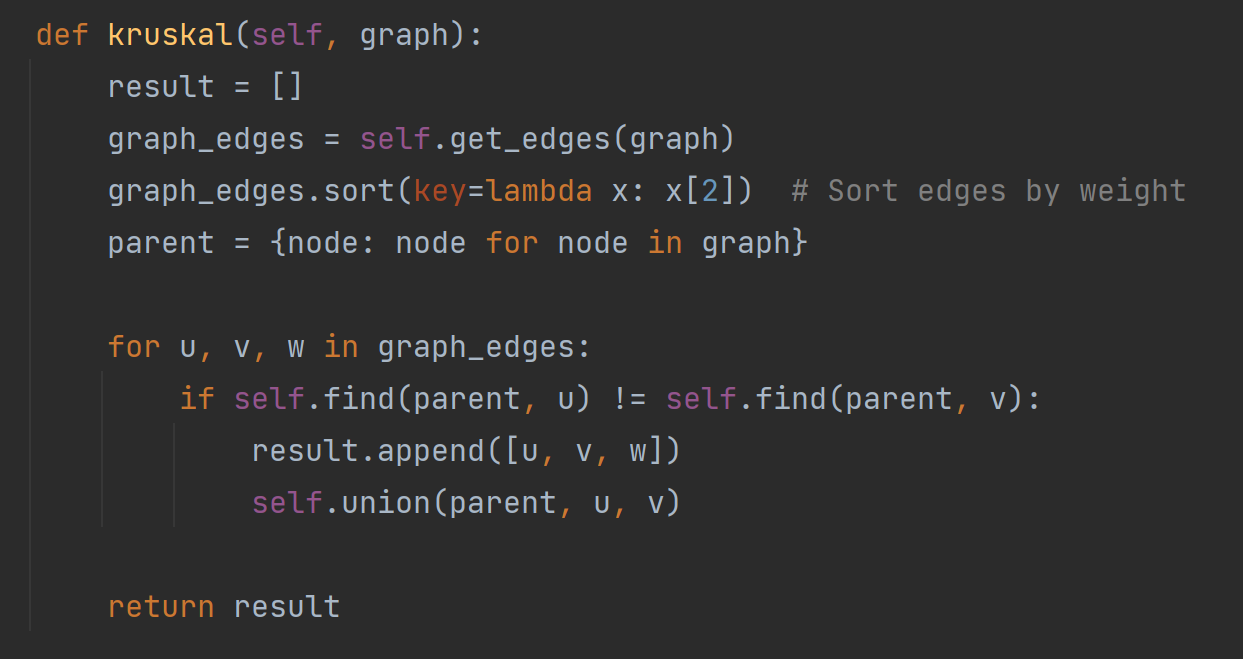
As input, each algorithm will receive 8 series of numbers of nodes 4, 8, 16, 32, 64, 128, 256, ,512.

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

**Implementation**

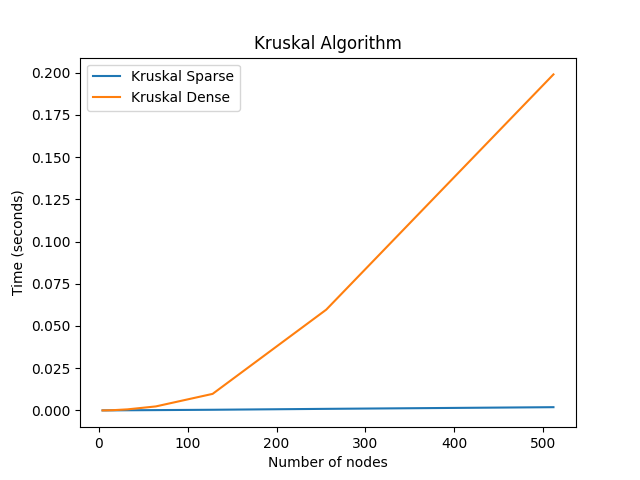
All algorithms will be implemented in their naïve form in python an analyzed empirically based on the time required for their completion. While the general trend of the results may be similar to other experimental observations, the particular efficiency in rapport with input will vary depending on memory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

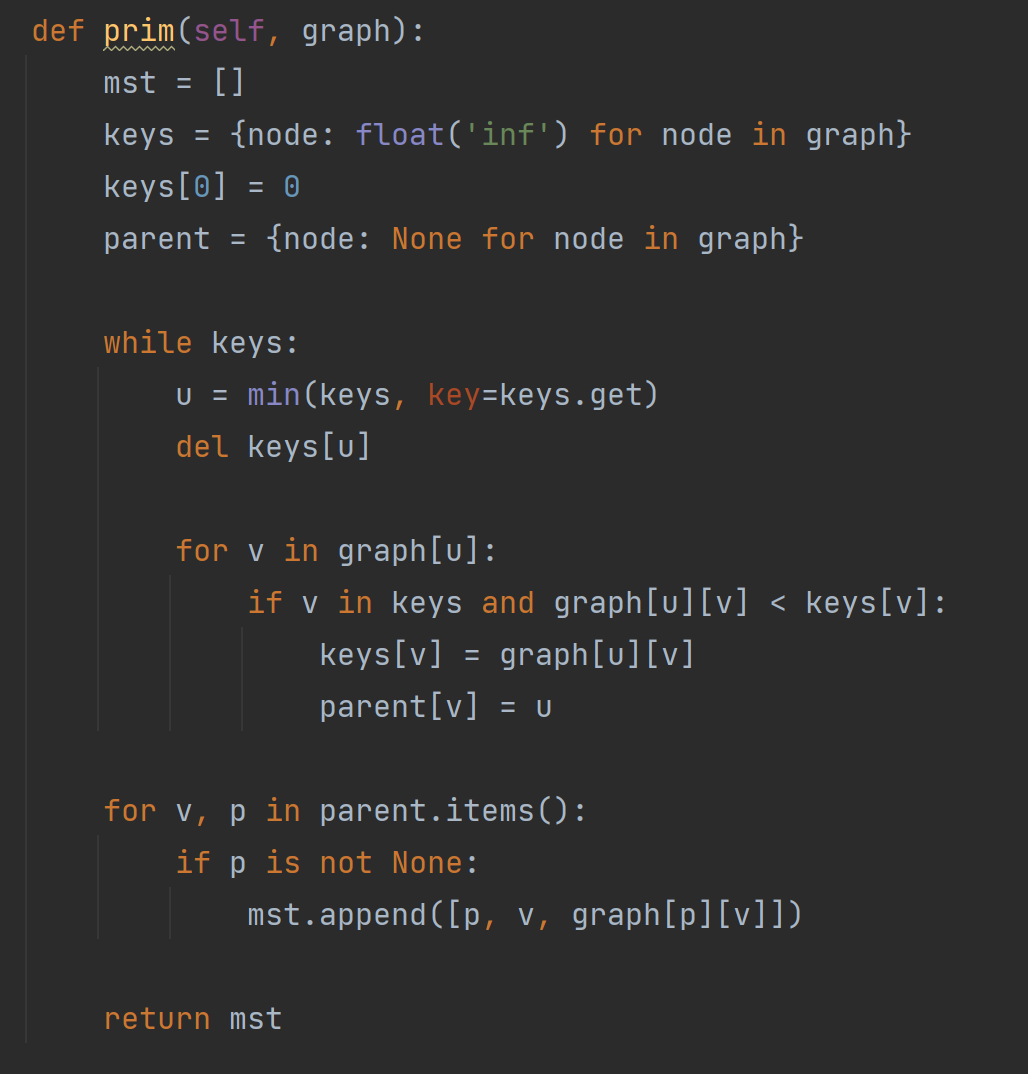
Kruskal Algorithm:

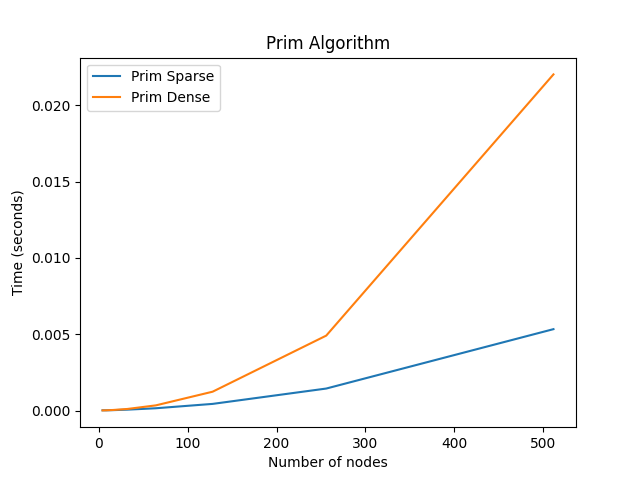


**Figure 1.** Time table for algorithms

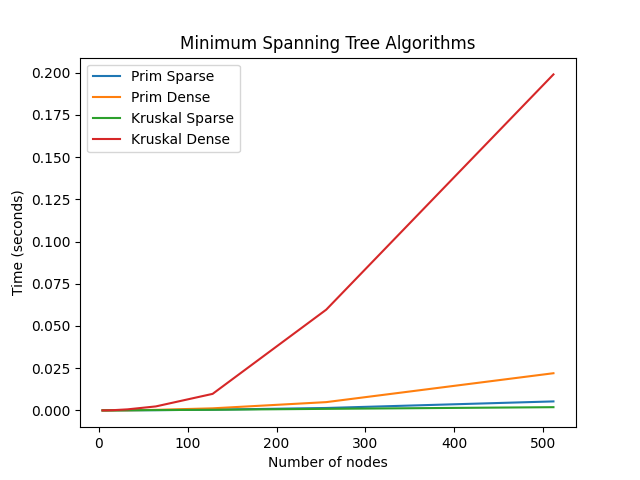
As observed in the table, Prim algorithm is faster than Kruskal in all cases. For a small number of nodes, either the graph is sparse of dense, the algorithms are executed almost in the same time. But as the number of nodes increase, the time also increase considerably for Kruskal.

 **Figure 2.** Graph for Kruskal algorithm



**Figure 3.** Graph for Prim

**Conclusion**



In summary, exploring Prim's and Kruskal's algorithms reveals their critical importance in effectively addressing the minimum spanning tree (MST) challenge within graph theory. Both methods provide robust mechanisms to construct a spanning tree that minimizes the total edge weight, yet they differ significantly in their approaches and performance metrics.

Kruskal's algorithm adopts a greedy technique, selecting the lowest-weight edges first, which ensures the tree expands without forming cycles, ultimately encompassing all vertices. It utilizes disjoint set data structures to manage and merge sets of vertices efficiently, facilitating a simple and effective resolution to the MST problem.

Conversely, Prim's algorithm also employs a greedy strategy but starts from a designated vertex, progressively adding the smallest available edge connecting to the expanding tree. This procedure guarantees the tree is both acyclic and continuously connected as it develops.

Kruskal's algorithm is typically more effective for sparse graphs with numerous edges, whereas Prim's algorithm is preferable in denser graphs or when initiating from a specific vertex offers a strategic advantage. Prim’s can also benefit from advanced data structures like priority queues or Fibonacci heaps to boost its efficiency in particular settings.

The decision between using Prim’s or Kruskal’s algorithms should be based on the graph’s specific attributes and the precise demands of the problem. Recognizing the subtle distinctions and strengths of each algorithm allows practitioners to choose the most appropriate method to address MST problems effectively in a variety of practical situations, from designing efficient networks to optimizing resource distribution and logistical operations.