**Ministry of Education and Research of the Republic of Moldova**

**Technical University of Moldova**

**Faculty of Computers, Informatics and Microelectronics**

**REPORT**

Laboratory work no. 7

*Empirical analysis of Greedy Algorithms*

Did:

St. gr. FAF-211 Andrei Ceban

Checked:

asist. univ. Cristofor Fistic

Chişinău - 2023

**Laboratory work no. 7**

**Objective:**

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. Make a graphical presentation of the data obtained
5. Increase the number of nodes in graph and analyze how this influences the algorithms.
6. Make a conclusion on the work done.

**INTRODUCTION**

A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result.The algorithm never reverses the earlier decision even if the choice is wrong. It works in a top-down approach.This algorithm may not produce the best result for all the problems. It's because it always goes for the local best choice to produce the global best result.However, we can determine if the algorithm can be used with any problem if the problem has the following properties:

Greedy Choice Property - If an optimal solution to the problem can be found by choosing the best choice at each step without reconsidering the previous steps once chosen, the problem can be solved using a greedy approach. This property is called greedy choice property.

Optimal Substructure - If the optimal overall solution to the problem corresponds to the optimal solution to its subproblems, then the problem can be solved using a greedy approach. This property is called optimal substructure.

Kruskal's algorithm finds a minimum spanning forest of an undirected edge-weighted graph. If the graph is connected, it finds a minimum spanning tree. (A minimum spanning tree of a connected graph is a subset of the edges that forms a tree that includes every vertex, where the sum of the weights of all the edges in the tree is minimized. For a disconnected graph, a minimum spanning forest is composed of a minimum spanning tree for each connected component.) It is a greedy algorithm in graph theory as in each step it adds the next lowest-weight edge that will not form a cycle to the minimum spanning forest.

In computer science, Prim's algorithm (also known as Jarník's algorithm) is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm operates by building this tree one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.

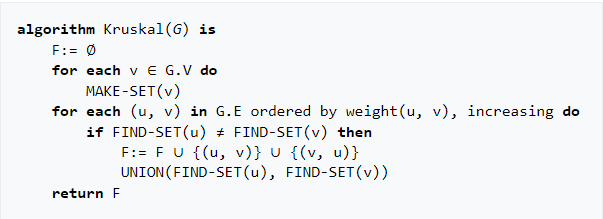
**IMPLEMENTATION**

**The Kruskal's algorithm**

T Kruskal's algorithm is inherently sequential and hard to parallelize. It is, however, possible to perform the initial sorting of the edges in parallel or, alternatively, to use a parallel implementation of a binary heap to extract the minimum-weight edge in every iteration. As parallel sorting is possible in time O(n) on O(log n) processors,the runtime of Kruskal's algorithm can be reduced to O(E α(V)), where α again is the inverse of the single-valued Ackermann function.

A variant of Kruskal's algorithm, named Filter-Kruskal, has been described by Osipovand is better suited for parallelization. The basic idea behind Filter-Kruskal is to partition the edges in a similar way to quicksort and filter out edges that connect vertices of the same tree to reduce the cost of sorting. The following pseudocode demonstrates this.

**Pseudocode**



**Algorithm explanation :**

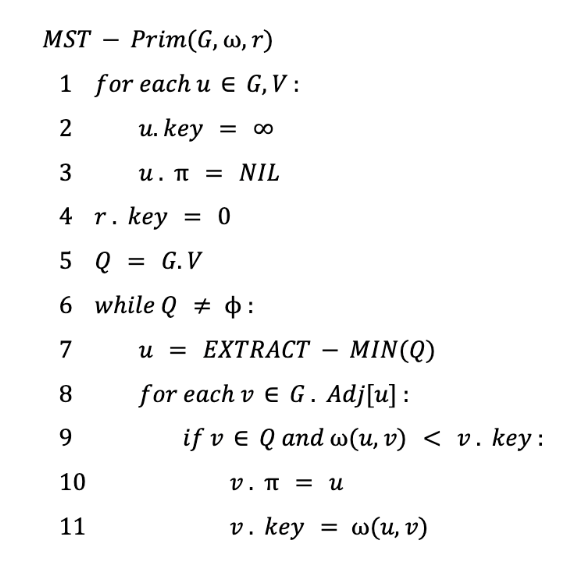
1. Sort all the edges of the graph in non-decreasing order of their weights. Create an empty set that will eventually contain the edges of the minimum spanning tree.
2. Iterate through the sorted edges and add each edge to the tree if it does not create a cycle: Start iterating through the sorted edges from the smallest weight to the largest weight. For each edge:
   * Check if adding the edge to the set of selected edges creates a cycle in the minimum spanning tree that has been built so far.
   * To check for cycles, you can use a disjoint-set data structure), which keeps track of the connected components and their relationships.
   * If adding the edge does not create a cycle, add it to the minimum spanning tree set.
3. Continue this process until all vertices are included in the minimum spanning tree: Repeat step 3 until all the vertices of the graph are included in the minimum spanning tree. In other words, keep iterating through the sorted edges and adding edges to the tree until there are V-1 edges in the set, where V is the number of vertices in the graph.

**The Prim's algorithm**

This algorithm always starts with a single node and moves through several adjacent nodes, in order to explore all of the connected edges along the way. 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, and 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. A group of edges that connects two sets of vertices in a graph is called cut in graph theory. So, at every step of Prim’s algorithm, find a cut, pick the minimum weight edge from the cut, and include this vertex in MST Set

**Pseudocode**



**Algorithm explanation :**

Step 1: Determine an arbitrary vertex as the starting vertex of the MST.

Step 2: Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).

Step 3: Find edges connecting any tree vertex with the fringe vertices.

Step 4: Find the minimum among these edges.

Step 5: Add the chosen edge to the MST if it does not form any cycle.

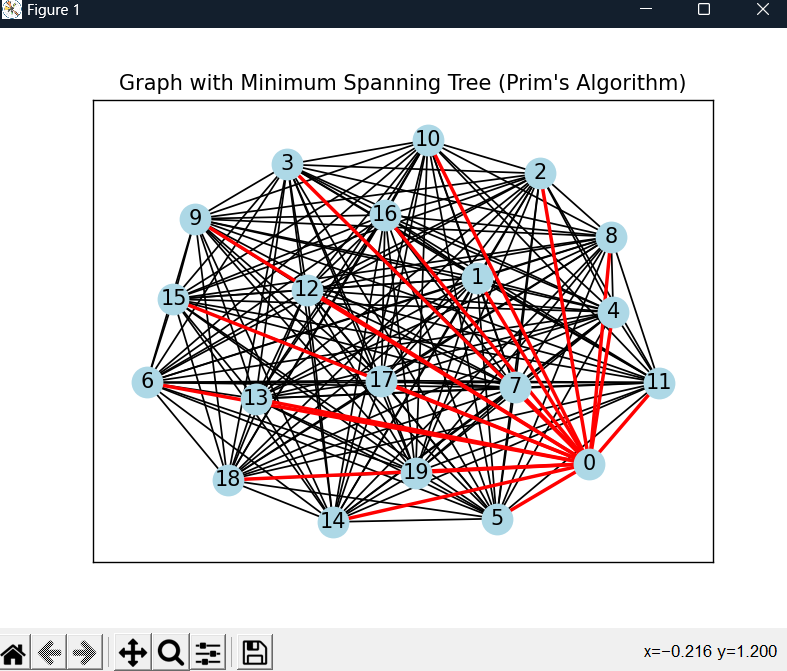
Step 6: Return the MST and exit

**RESULTS**

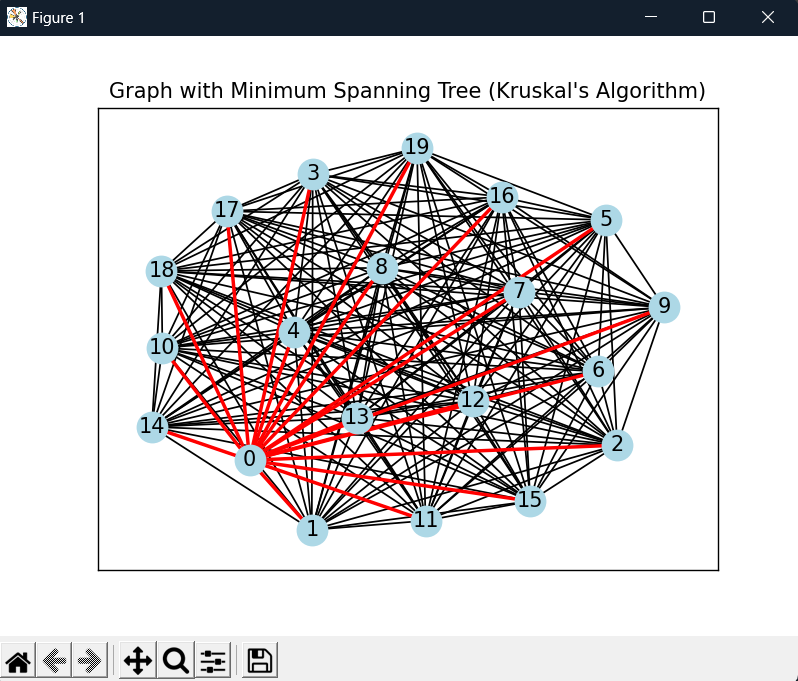
Prim's Algorithm: Time Complexity: O(V^2) for the adjacency matrix representation of the graph and O(E log V) for the adjacency list representation, where V is the number of vertices and E is the number of edges in the graph. It starts with an arbitrary vertex and grows the MST by adding the shortest edge connecting the MST to a new vertex until all vertices are included. It relies on maintaining a priority queue to efficiently select the next vertex to add to the MST.

Kruskal's Algorithm: Time Complexity: O(E log E) or O(E log V), depending on the implementation, where V is the number of vertices and E is the number of edges in the graph. It initially treats each vertex as a separate component and repeatedly selects the minimum-weight edge from the remaining edges until all vertices are connected. It uses a disjoint-set data structure to efficiently keep track of the connected components and detect cycles.

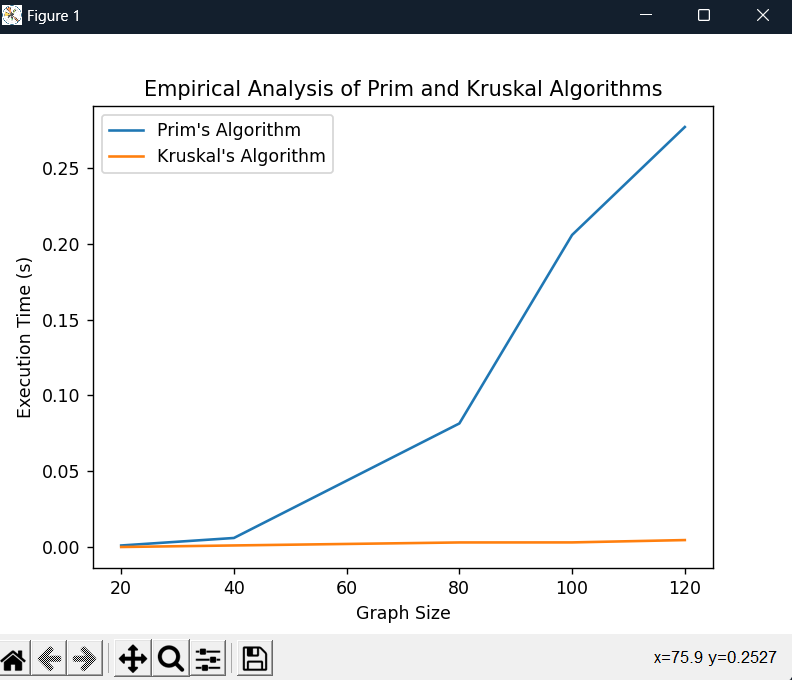
**Spanning Tree Prim’s Algorithm**



**Spanning Tree Kruskal’s Algorithm**



**Time Comparison for Prim and Kruskal Algorithm**



**CONCLUSION**

During this lab experiment, I analyzed and contrasted the effectiveness of Prim's and Kruskal's algorithms in determining the minimum spanning tree (MST) of a graph. These algorithms employ diverse methods for constructing the MST and present unique benefits and factors to consider. Prim's algorithm follows a greedy approach, commencing with a random vertex and gradually expanding the MST by including the edge with the lowest weight that connects a vertex within the MST to a vertex outside.

When comparing the two algorithms, it becomes evident that Prim's algorithm tends to exhibit superior performance in dense graphs because it effectively utilizes an adjacency matrix for representation. The time complexity of O(V^2) associated with Prim's algorithm enables it to handle larger dense graphs more efficiently compared to Kruskal's algorithm.

The advantage of Prim's algorithm is its complexity, which is better than Kruskal's algorithm. Therefore, Prim's algorithm is helpful when dealing with dense graphs that have lots of edges. However, Prim's algorithm doesn't allow us much control over the chosen edges when multiple edges with the same weight occur.

To summarize, both Prim's and Kruskal's algorithms provide dependable solutions for discovering the minimum spanning tree. When determining which algorithm to utilize, it is important to take into account the graph's properties, such as its level of sparsity. Prim's algorithm is a preferable option for graphs with many connections, whereas Kruskal's algorithm performs exceptionally well in graphs with fewer connections. By comprehending these distinctions, we can choose the most suitable algorithm for our particular graph structure and enhance the efficiency of our minimum spanning tree calculations.

Git Repo : https://github.com/andeiceban0352/Labs-Anul2/tree/main/Lab%20APA/Lab7