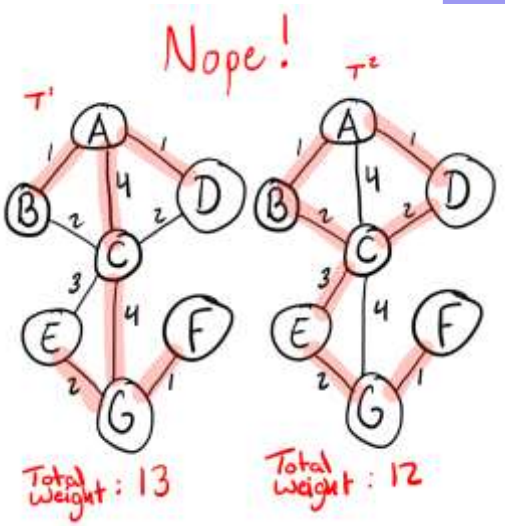
Overview

Finding Critical and Pseudo-Critical Edges in Minimum Spanning Tree

- Project by :
- Student Name: CH. Chandra Naga Sai Kumar
- Course Code: CSA0656
- Course Name: DAA
- Slot: A

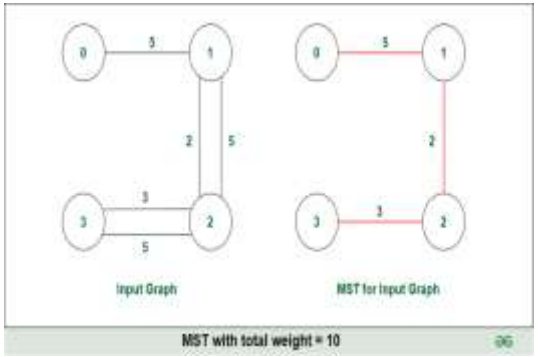


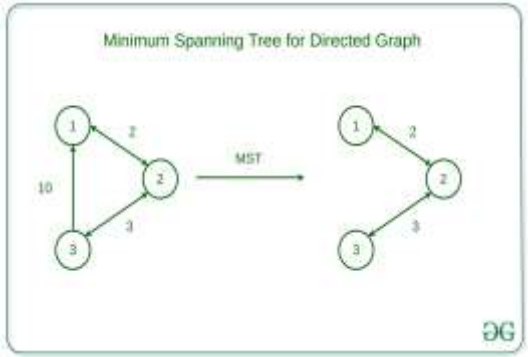
Abstract

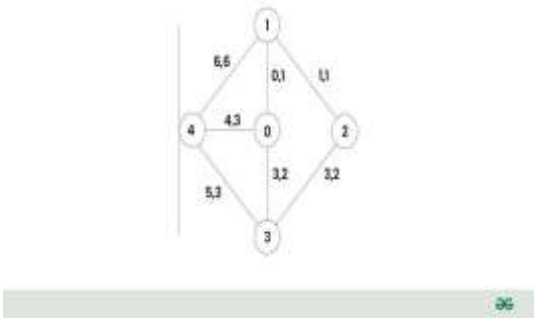
- The goal is to identify critical and pseudo-critical edges in a Minimum Spanning Tree (MST) of a weighted, undirected, and connected graph.
- A graph is defined by 'n' vertices and an array of edges, where each edge is represented as an array of three values: [vertex1, vertex2, weight].
- Critical edges are those that, if removed, will increase the total weight of the MST, while pseudo-critical edges can appear in some MSTs but not all.
- This classification is essential for optimizing network design, ensuring the most efficient connections while minimizing costs.

Example

problem







Input Details

Given a graph with n = 5 vertices and edges as follows: [[0,1,1], [1,2,1], [2,3,2], [0,3,2], [0,4,3], [3,4,3], [1,4,6]].

Output Interpretation

Output: [[0, 1], [2, 3, 4, 5]]. Critical edges are [0,1] and [1,2] as they appear in all MSTs.

Understanding Edge Types

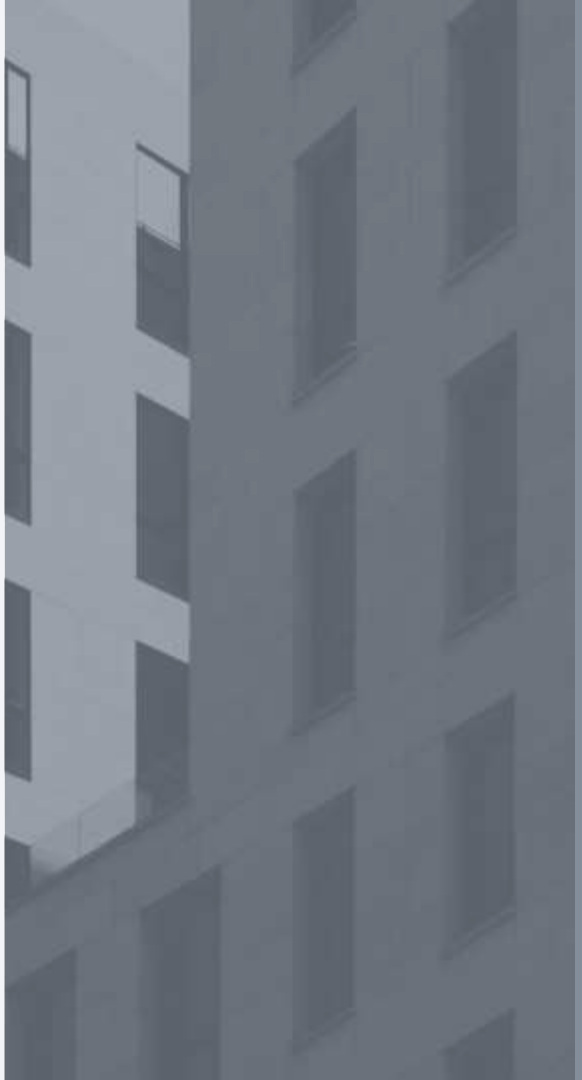
Critical edges are essential for every MST, while pseudo-critical edges may appear in some but not all MSTs.

Initial Inefficient Approach

- Compute the MST using all edges initially.
- For each edge, check if its removal increases the MST weight.
- For each edge, check if forcing its inclusion maintains the same MST weight.

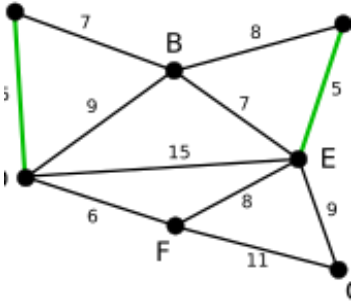
Efficient Approach Using MST

- Utilize Kruskal's or Prim's algorithm to compute the initial MST.
- For each edge, remove it and check if the MST weight increases for critical edges.
- For pseudo-critical edges, include each edge and check if the MST weight remains unchanged.



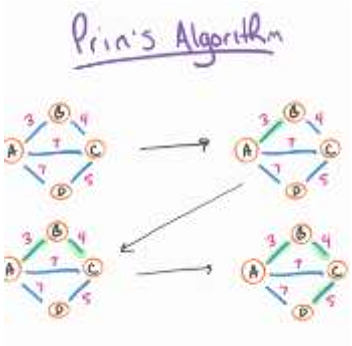
Algorithms

MST Techniques



Kruskal's Algorithm

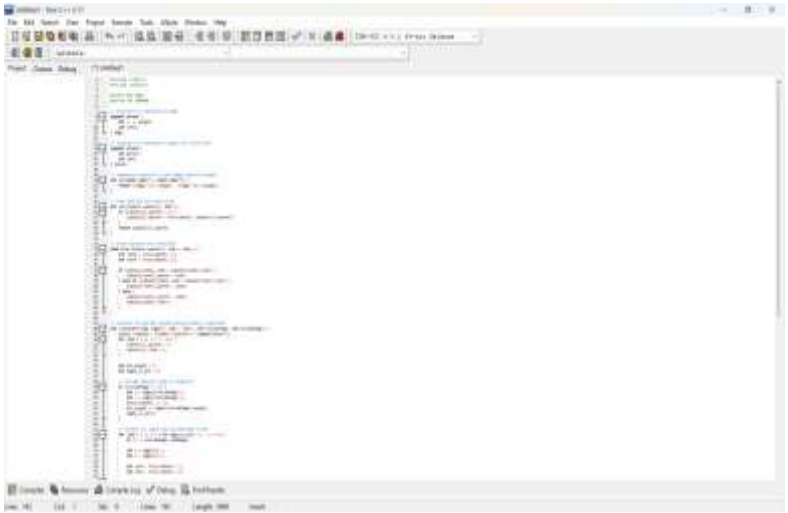
Sort edges by weight, then add edges to the MST using a union-find structure while avoiding cycles, stopping when n-1 edges are included.

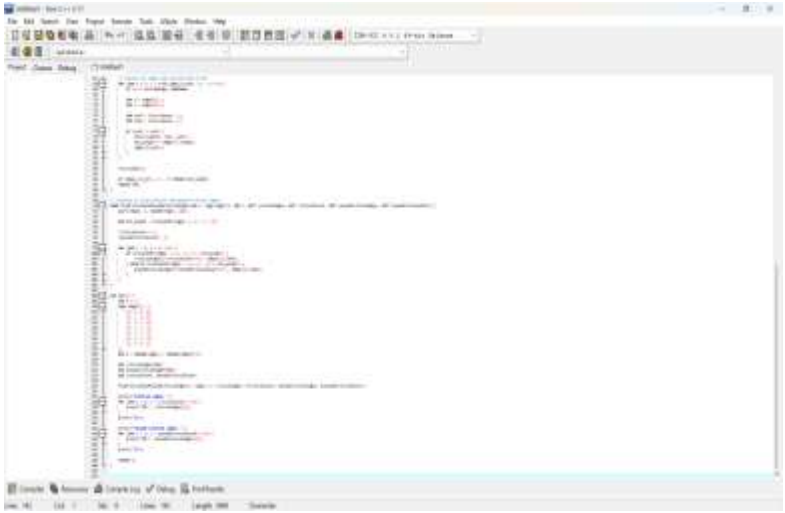


Prim's Algorithm

Start from an arbitrary vertex and continuously add the minimum weight edge connecting the tree to a new vertex until all vertices are included.

Code:





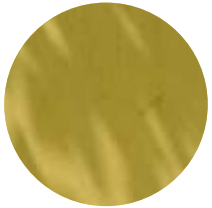
Output :

Analysis

Complexity Analysis

Best Case Complexity

In the best case, sorting edges and initial MST calculation require O(E log E) + O(E log V). If early edges form a complete MST, checks for critical or pseudo-critical edges can be minimized.



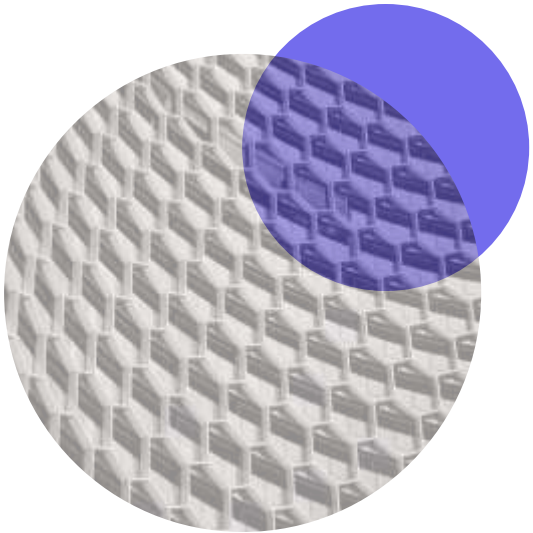
Worst Case Complexity

In the worst case, the complexity is O(E log E) for sorting and O(E log V) for MST calculations. Each edge may need two full Kruskal's algorithm runs, leading to O(E^2 log V).

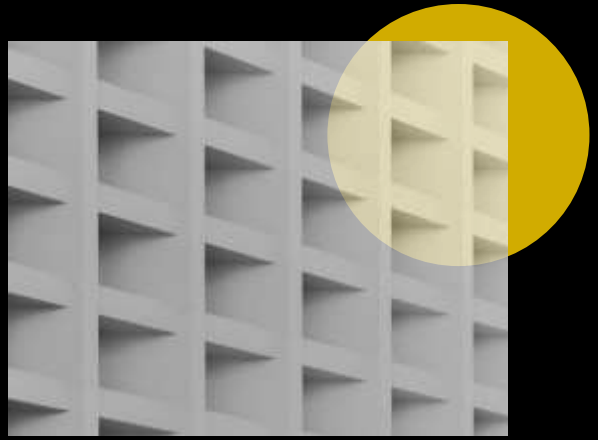
Average Case Complexity

On average, the complexity combines best and worst scenarios, approximated as O(E log E) + O(E^2 log V). The dominant term is O(E^2 log V), assuming a moderate number of vertices and edges.

Conclusion



- The project successfully identified critical and pseudo-critical edges in the Minimum Spanning Tree (MST) of a weighted, undirected graph using efficient algorithms.
- Kruskal's algorithm, combined with a Union-Find data structure, allowed for efficient calculations of MSTs and edge classifications.
- Critical edges are those that must be included in all MSTs, while pseudo-critical edges may appear in some but not necessarily all MSTs, allowing for flexibility in edge selection.



Thank you.