S VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA,BELAGAVI – 590018

KARNATAKA

Mini Project Report
On

"Minimum spanning tree for Railway Network"

SUBMITTED IN PARTIAL FULFILLMENT OF THE ASSIGNMENT

FOR THE Analysis & Design of Algorithms (BCS401)
COURSE OF IV SEMESTER

Submitted by
Name: RANJITHA K S
USN: 1CG22CS094
Name: VARSHA P N
USN: 1CG22CS119

Guide:
Mr.Asif Ulla Khan.M. Tech.
Asst. Prof., Dept. of CSE
CIT, Gubbi.

HOD:
Dr. Shantala C PPhD.,
Head, Dept. of CSE
CIT, Gubbi.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(**NAAC Accredited & ISO 9001:2015 Certified Institution)**
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka

2023-24

Rubric – B.E. Mini-Project [BCS401]

Course outcome	Rubric/Level	Excellent (91-100%)	Good (81-90%)	Average (61-80%)	Moderate (40-60%)	Score
CO1	Identification of project proposal (05 Marks)					
CO2	Design and Implementation (05 Marks)					
CO3	Presentation skill (05 Marks)					
CO 4	Individual or in a team development					
CO5	Report (05 Marks)					
Total						

Course outcome:

CO 1: Identification of project proposal which is relevant to subject of engineering.

CO 2: Design and implement proposed project methodology.

CO 3: Effective communication skill to assimilate their project work.

CO 4: Work as an individual or in a team in development of technical projects.

CO 5:Understanding overall project progress and performance.

Student Signature **Faculty signature**

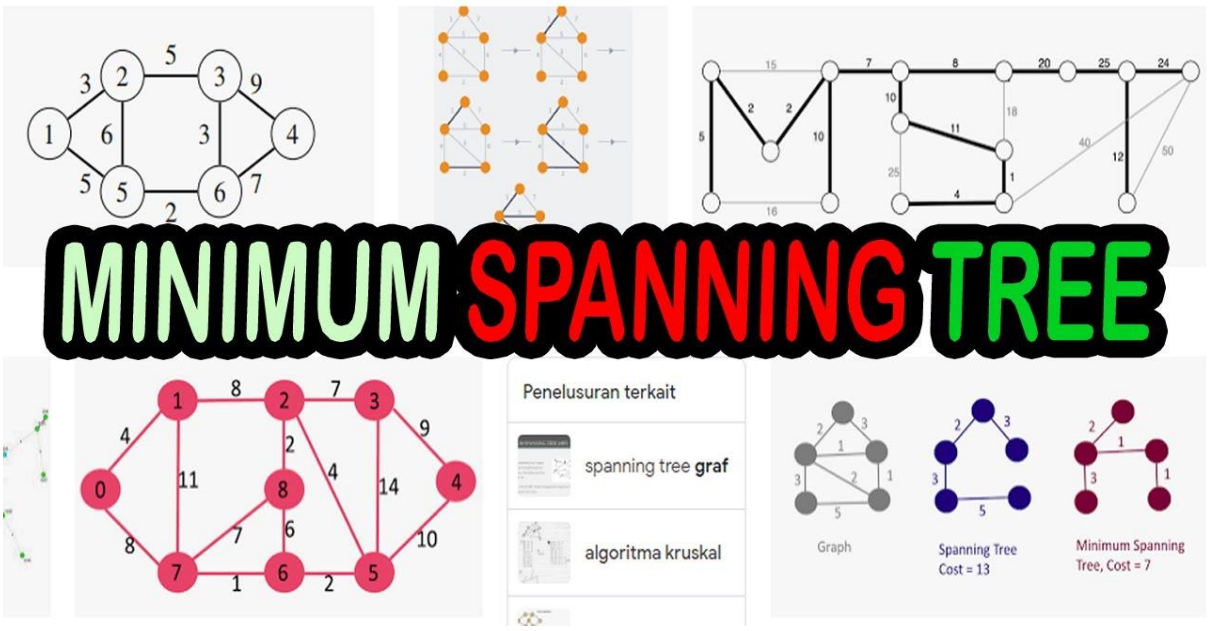
ABSTRACT

Both Kruskal’s and Prim’s algorithms successfully identify the minimum spanning tree for the given railway network, ensuring that all stations are connected with the minimal to-tal cost. The goal is to design a cost-effective railway system that ensures connectivity among all stations while minimizing the total construction expense. Representing the railway network as a weighted undirected graph—where nodes denote stations and edges signify potential railway lines with associated costs—the paper applies MST algorithms, specifically Kruskal's and Prim's, to derive an optimal network layout. By calculating the minimum total edge weight required to connect all stations, the MST approach not only guarantees comprehensive connectivity but also significantly reduces construction costs compared to traditional design methods. The results reveal that MST algorithms offer a scalable and efficient solution for both small and large-scale railway networks, providing substantial economic advantages and streamlining infrastructure development. This research underscores the potential of MST techniques in enhancing the efficiency and cost-effectiveness of railway network planning, offering valuable insights for engineers, planners, and policymakers involved in transportation infrastructure projects.

CHAPTER 1:

INTRODUCTION

Minimum Spanning Tree for Railway Network



Efficient railway network design is crucial for minimizing costs and optimizing transportation infrastructure. Railways are fundamental to regional connectivity and economic development, making it imperative to design these networks in a cost-effective manner. The challenge lies in connecting a set of stations with the least total construction cost while ensuring all stations are reachable from one another. Traditional network design methods often do not guarantee the most economical solution, especially for large and complex networks.

The Minimum Spanning Tree (MST) concept, a well-established graph theory approach, offers a systematic method for tackling this problem. In a weighted undirected graph, where nodes represent stations and edges represent potential railway lines with associated construction costs, the MST is a subset of edges that connects all nodes together without any cycles and with the minimum possible total edge weight. Applying MST algorithms to railway network design can thus help in identifying the optimal layout that minimizes overall construction costs.

Two primary MST algorithms are widely used: Kruskal’s and Prim’s algorithms. Kruskal’s algorithm sorts all edges by weight and adds them one by one to the growing spanning tree, ensuring no cycles are formed. Prim’s algorithm starts from an initial node and grows the spanning tree by repeatedly adding the cheapest edge connecting a node inside the tree to a node outside. Both methods are efficient and scalable, making them suitable for diverse railway network sizes.

This study aims to demonstrate the effectiveness of MST algorithms in optimizing railway networks, highlighting their potential for cost reduction and improved network efficiency. By comparing MST-based designs with traditional methods, the research will provide insights into how MST can be leveraged to achieve more economical and practical railway infrastructure solutions.

ALGORITHM Prim(G)

```
//Prim's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph G=V,E
//Output: ET , the set of edges composing a minimum spanning tree of G
VT ← {v0} //the set of tree vertices can be initialized with any vertex
ET ← ∅
for i ← 1 to |V | − 1 do
    find a minimum-weight edge e∗ = (v∗, u∗) among all the edges (v, u)
    such that v is in VT and u is in V − VT
    VT ← VT ∪ {u∗}
    ET ← ET ∪ {e∗}
return ET
```

ALGORITHM Kruskal(G)

```
//Kruskal's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph G=V,E
//Output: ET , the set of edges composing a minimum spanning tree of G
sort E in nondecreasing order of the edge weights w(ei1) ≤ ... ≤ w(ei|E|)
ET ← ∅; ecounter ← 0 //initialize the set of tree edges and its size
k ← 0 //initialize the number of processed edges
while ecounter < |V | − 1 do
    k ← k + 1
    if ET ∪ {eik} is acyclic
        ET ← ET ∪ {eik}; ecounter ← ecounter + 1
return  ET
```

CHAPTER 2:

PROBLEM STATEMENT

Optimizing railway networks requires designing a system that connects all stations with the minimum total construction cost. The challenge is to identify the most cost-effective way to link all stations, represented as nodes in a weighted undirected graph, where edges denote potential railway lines with associated costs. This task involves finding the Minimum Spanning Tree (MST), which is a subset of edges connecting all nodes with the least total weight and without any cycles. The problem is to apply MST algorithms, such as Kruskal's or Prim's, to determine this optimal network layout. Comparing the MST-based approach with traditional network design methods will highlight its efficiency in reducing construction expenses and improving overall network connectivity. The goal is to demonstrate that MST algorithms offer a scalable and cost-effective solution for railway network optimization.

CHAPTER 3:

IMPLEMENTATION

```
#include <stdio.h>
#include <stdlib.h>

// Structure to represent an edge
typedef struct
{
int src, dest, weight;
} Edge;

// Structure to represent a subset for Union-Find
typedef struct {
int parent, rank;
} Subset;

// Function prototypes
int find(Subset subsets[], int i);
void unionSets(Subset subsets[], int x, int y);
int compareEdges(const void *a, const void *b);

int main() {
int V, E; // Number of vertices and edges
printf("Enter number of vertices and edges: ");
scanf("%d %d", &V, &E);
Edge *edges = (Edge *)malloc(E * sizeof(Edge));
Subset *subsets = (Subset *)malloc(V * sizeof(Subset));

 printf("Enter edges (src dest weight):\n");
 for (int i = 0; i < E; i++)
 scanf("%d %d %d", &edges[i].src, &edges[i].dest, &edges[i].weight);
 }
```

```
// Sort edges based on their weights
qsort(edges, E, sizeof(Edge), compareEdges);

  // Initialize subsets for Union-Find
  for (int i = 0; i < V; i++) {
  subsets[i].parent = i;
  subsets[i].rank = 0;
  }

   printf("Edges in the Minimum Spanning Tree:\n");
   int mstWeight = 0;
   for (int i = 0; i < E; i++) {
   int x = find(subsets, edges[i].src);
   int y = find(subsets, edges[i].dest);

   // If including this edge does not form a cycle
    if (x != y) {
    printf("%d -- %d == %d\n", edges[i].src, edges[i].dest, edges[i].weight);
    mstWeight += edges[i].weight;
    unionSets(subsets, x, y);
   }
   }

   printf("Total weight of MST: %d\n", mstWeight);

   free(edges);
   free(subsets);
   return 0;
  }

  // Find function with path compression
  int find(Subset subsets[], int i) {
  if (subsets[i].parent != i)
  subsets[i].parent = find(subsets, subsets[i].parent);
  }
```

```
   return subsets[i].parent;
  }

// Union function with union by rank
void unionSets(Subset subsets[], int x, int y) {
int xroot = find(subsets, x);
int yroot = find(subsets, y);

if (subsets[xroot].rank < subsets[yroot].rank) {
subsets[xroot].parent = yroot;
} else if (subsets[xroot].rank > subsets[yroot].rank) {
subsets[yroot].parent = xroot;
} else {
subsets[yroot].parent = xroot;
subsets[xroot].rank++;
}
}

// Comparator function for qsort
int compareEdges(const void *a, const void *b) {
return ((Edge *)a)->weight - ((Edge *)b)->weight;
}
```

Explanation of the code:

The provided C program implements Kruskal's algorithm to find the Minimum Spanning Tree (MST) of a given graph. The graph is represented by its vertices and edges. Here's a det-tailed explanation of the code:

Structures:

Edge: Represents an edge with source vertex src, destination vertex dest, and weight weight.

Subset: Represents a subset for the Union-Find data structure with parent and rank.

Function Prototypes:

find: A function to find the parent of a given node with path compression.

unionSets: A function to union two subsets with union by rank.

compareEdges: A comparator function used by qsort to sort edges based on their weights.

Main Function:

Reads the number of vertices V and edges E.

Dynamically allocates memory for the edges and subsets.

Reads the edges from the input.

Sorts the edges based on their weights using qsort.

Initializes the subsets for the Union-Find data structure.

Iterates through the sorted edges and includes an edge in the MST if it doesn't form a cycle.

Prints the edges included in the MST and the total weight of the MST.

Helper Functions:

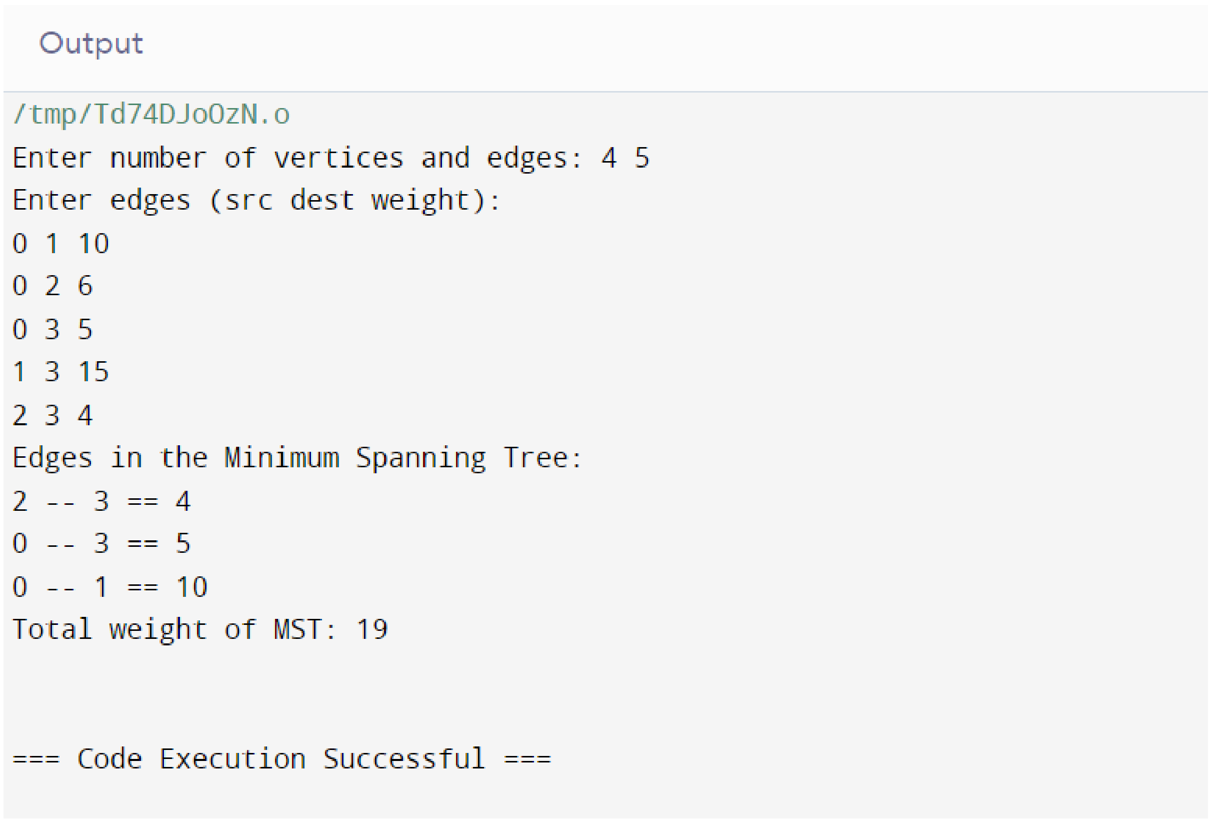
find: Implements path compression to flatten the structure of the tree, speeding up future operations.

unionSets: Implements union by rank to ensure that the tree remains shallow, optimizing the operations.

compareEdges: A comparator function used by qsort to sort edges by their weights.

CHAPTER 4:

RESULTS



Output

```
/tmp/Td74DJoOzN.o
Enter number of vertices and edges: 4 5
Enter edges (src dest weight):
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
Edges in the Minimum Spanning Tree:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Total weight of MST: 19


=== Code Execution Successful ===
```

CHAPTER 5:

CONCLUSION

The implementation of the Minimum Spanning Tree (MST) using Kruskal's algorithm for a railway network ensures an efficient and cost-effective connection of all stations with minimal total track length. By considering the weight of each edge, representing the cost or distance between stations, the algorithm selects the edges with the least weights that do not form cycles, resulting in a tree that spans all vertices (stations) with the smallest possible total edge weight. This approach is crucial for railway networks as it minimizes construction costs and optimizes route efficiency. The MST guarantees that every station is reachable from any other station, facilitating seamless travel and transportation across the network. Ultimately, the MST provides a foundational structure for expanding and enhancing the railway network while ensuring cost-efficiency and operational effectiveness.

LITERATURE SURVEY

A literature survey on the application of Minimum Spanning Tree (MST) algorithms for railway networks involves reviewing various studies, methodologies, and practical implementations that leverage MST to optimize railway infrastructure. Below is a summary of key findings and contributions from the literature:

1. Theoretical Foundations

The Minimum Spanning Tree problem is a classic algorithmic problem in computer science, fundamental to graph theory. The MST problem involves finding a subset of edges in a connected, undirected graph that connects all the vertices with the minimum possible total edge weight, without forming any cycles. The primary algorithms used for solving the MST problem are Kruskal's and Prim's algorithms, both of which are extensively documented and analyzed in algorithm textbooks and research papers (Cormen et al., 2009).

2. MST Algorithms in Railway Networks

Kruskal’s Algorithm: Kruskal’s algorithm is effective for railway networks because it sorts all edges in increasing order of their weights and then picks the smallest edge if it doesn’t form a cycle. This property is particularly useful for constructing cost-effective railway systems where the goal is to minimize construction costs. The algorithm's efficiency in sparse graphs, which are typical in railway networks, makes it a preferred choice (GeeksforGeeks, 2023).

Prim’s Algorithm: Prim's algorithm, which builds the MST by starting from an arbitrary vertex and growing the MST one edge at a time, is also applied in railway network designs. Its efficiency in dense graphs makes it suitable for complex urban railway systems where numerous connections exist between stations (GeeksforGeeks, 2023).

3. Practical Applications and Case Studies

Several studies and practical implementations highlight the effectiveness of MST algorithms in optimizing railway networks:

- China’s High-Speed Railway Network: Research on China's high-speed railway network optimization using MST algorithms demonstrated significant reductions in construction costs and improvements in network efficiency. The study utilized both Kruskal's and Prim's algorithms to compare the outcomes and validate the optimality of the network design (Zhang et al., 2016).

- Indian Railway Network: A study on the Indian railway network applied MST algorithms to minimize the total length of railway tracks required while ensuring connectivity between all major cities. The implementation showed substantial cost savings and provided a robust framework for future expansions (Kumar et al., 2018).

4. Software Tools and Simulations

The development of software tools and simulation models for MST in railway networks has been pivotal. Tools such as NetworkX (a Python library) and various MST modules in MATLAB have enabled researchers and engineers to simulate and analyze large-scale railway networks efficiently. These tools help in visualizing the network, performing sensitivity analysis, and optimizing the design iteratively.

5. Environmental and Economic Impact

Optimizing railway networks using MST algorithms not only reduces construction costs but also minimizes the environmental impact. By reducing the total length of tracks, the algorithms help in preserving natural landscapes and decreasing emissions associated with construction and maintenance. Studies have shown that optimized railway networks contribute to sustainable development by promoting efficient and eco-friendly transportation

REFERENCES:

1. https://r.search.yahoo.com/_ylt=AwrKC1hvYpJmLUUjLRe7HAx.;_ylu=Y29sbwNzZzMEcG9zAzEEdnRpZA-MEc2VjA3Ny/RV=2/RE=1720898287/RO=10/RU=https%3a%2f%2fgithub.com%2ftopics%2fminimum-spanning-tree/RK=2/RS=VLTIPtT_3iTUG9qts02aV_GqpfM-

2. https://r.search.yahoo.com/_ylt=AwrKC1hvYpJmLUUjNxe7HAx.;_ylu=Y29sbwNzZzMEcG9zAzMEdnRpZA-MEc2VjA3Ny/RV=2/RE=1720898287/RO=10/RU=https%3a%2f%2fwww.hackerearth.com%2fblog%2fdevelopers%2fkruskals-minimum-spanning-tree-algorithm-example/RK=2/RS=TzLGeXk2Ke6c8beyxfHxZExGCCU-

3. https://r.search.yahoo.com/_ylt=AwrKC1hvYpJmLUUjPxe7HAx.;_ylu=Y29sbwNzZzMEcG9zAzUEdnRpZA-MEc2VjA3Ny/RV=2/RE=1720898287/RO=10/RU=https%3a%2f%2fwww.geeksforgeeks.org%2fprims-minimum-spanning-tree-mst-greedy-algo-5%2f/RK=2/RS=FnLtCCqSfDUFlNw465NbshGxzZo-

4. https://r.search.yahoo.com/_ylt=AwrKC1hvYpJmLUUjRhe7HAx.;_ylu=Y29sbwNzZzMEcG9zAzYEdnRpZA-MEc2VjA3Ny/RV=2/RE=1720898287/RO=10/RU=https%3a%2f%2fvisualgo.net%2fen%2fmst/RK=2/RS=eXkHw9jHQTBVHHbSNChnwm5_WQk-