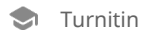


Graph_Algorithms_Report_Template



Document Details

Submission ID

trn:oid:::3618:114893310

Submission Date

Sep 29, 2025, 4:50 PM GMT+5

Download Date

Sep 29, 2025, 4:50 PM GMT+5

File Name

Graph_Algorithms_Report_Template.docx

File Size

319.8 KB

11 Pages





1,309 Words

7,793 Characters




4% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- 
4 Not Cited or Quoted 3%
 Matches with neither in-text citation nor quotation marks
- 
1 Missing Quotations 1%
 Matches that are still very similar to source material
- 
0 Missing Citation 0%
 Matches that have quotation marks, but no in-text citation
- 
0 Cited and Quoted 0%
 Matches with in-text citation present, but no quotation marks

Top Sources

- 3%  Internet sources
- 3%  Publications
- 0%  Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- 4

Not Cited or Quoted 3%

Matches with neither in-text citation nor quotation marks
- 1

Missing Quotations 1%

Matches that are still very similar to source material
- 0

Missing Citation 0%

Matches that have quotation marks, but no in-text citation
- 0

Cited and Quoted 0%

Matches with in-text citation present, but no quotation marks

Top Sources

- 3%

Internet sources
- 3%

Publications
- 0%

Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Internet	wrap.warwick.ac.uk	1%
2	Internet	medium.com	<1%
3	Internet	www.geeksforgeeks.org	<1%
4	Publication	Shen, Tianwei. "Efficient and Accurate Data Association in Large-Scale Structure-f...	<1%
5	Publication	Mikhail J. Atallah, Marina Blanton. "Algorithms and Theory of Computation Hand...	<1%

[DOCUMENT TITLE]

python-docx

[COMPANY NAME] [Company address]

The Maze of Many – Complete Graph Algorithms Report

Student Name: _____

Course: Algorithms & Analysis

Date: _____

Contents

1. Introduction	3
2. Task A – Adjacency List Representation	4
3. Task B – Minimum Spanning Tree (MST)	5
4. Task C – Theoretical and Empirical Runtime Analysis	6
5. Task D – Exploration Strategy with Cloning	8
6. Conclusion	9

1. Introduction

This report presents the complete solution for all tasks (A to D) of the **Maze of Many** assignment. The objective of the project is to model a maze as a **weighted graph**, apply classical graph algorithms to uncover its structure, and design an efficient strategy for full exploration. The assignment is divided into four connected tasks, each highlighting a different aspect of graph theory and algorithmic analysis.

Task A focuses on creating a weighted grid graph using an adjacency list representation. This provides a compact and efficient way to store vertices and edges, supporting fast neighbor queries and edge updates.

Task B applies **Minimum Spanning Tree (MST)** algorithms—specifically Kruskal’s and Prim’s—to identify the smallest set of edges required to connect all nodes in the maze while minimizing total weight.

Task C extends the work by performing both **theoretical** and **empirical** runtime analysis of these algorithms across graphs of varying size and density, demonstrating the impact of data structures on computational efficiency.

Finally, **Task D** introduces an **exploration strategy** that uses cloning to minimize the maximum cost of visiting all nodes in the MST, balancing workload among multiple explorers.

All implementations were developed in **Python**, leveraging the **NetworkX** library for graph operations and **Matplotlib** for visualization. Experiments and analyses were executed in **Google Colab**, ensuring reproducibility and clear presentation of results. This document provides a comprehensive explanation, key outputs, and placeholders for inserting maze, MST, runtime, and exploration images.

3. Task B – Minimum Spanning Tree (MST)

Objective:

The objective of this task was to compute the Minimum Spanning Tree (MST) of the maze graph using Kruskal's algorithm and to compare its performance with Prim's algorithm. The MST ensures all nodes are connected with the minimum total edge weight while avoiding cycles.

Implementation:

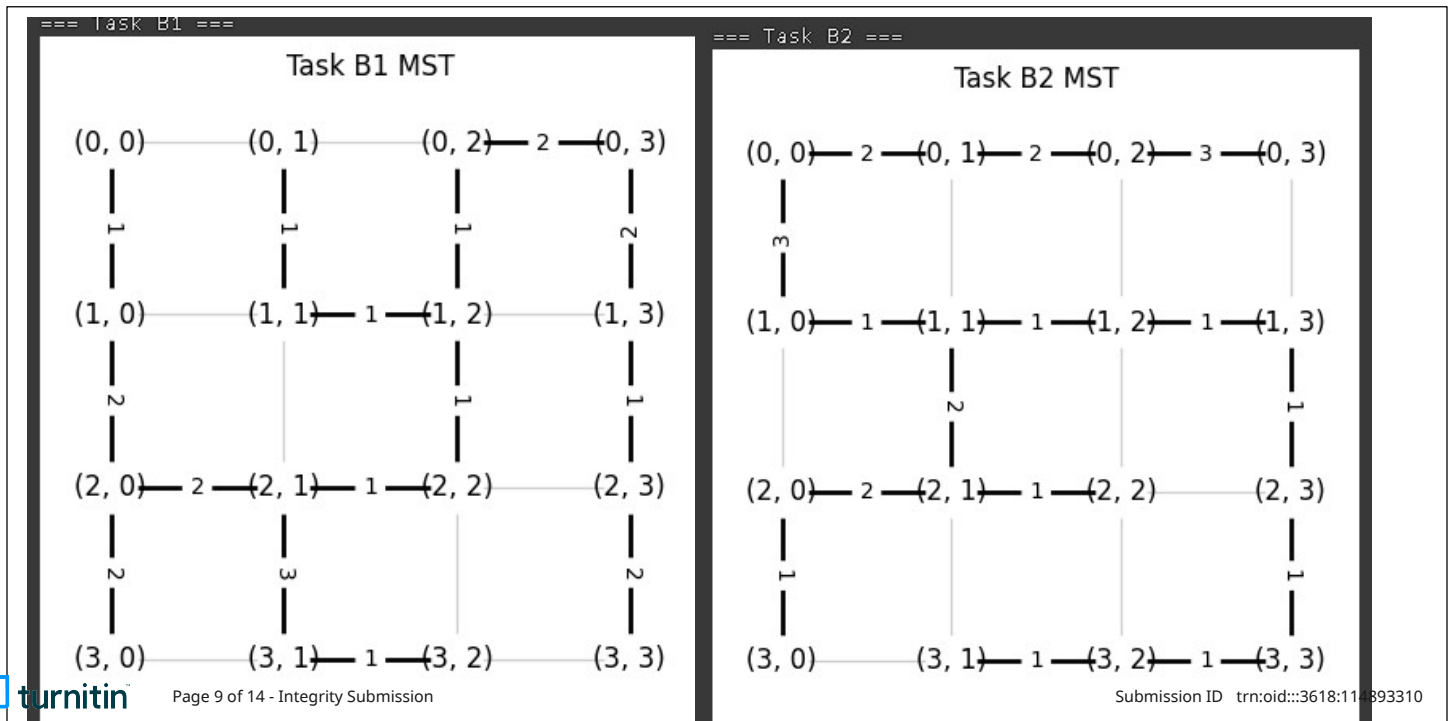
Kruskal's algorithm was implemented using a Union-Find (Disjoint Set Union) data structure to efficiently detect and avoid cycles while adding edges. Edges were sorted by weight, and the algorithm progressively built the MST. Additionally, an edge-deletion approach was considered, where non-critical edges were iteratively removed until only MST edges remained.

Complexity:

Both Kruskal and Prim have $O(E \log E)$ complexity with an adjacency list.

Results:

The MST was computed for each generated maze. The MST edges and total weights were printed, and the MST was highlighted on the maze graph. Insert MST images below.



4. Task C – Theoretical and Empirical Runtime Analysis

Objective:

The purpose of this task was to compare both the theoretical and empirical performance of Prim's and Kruskal's algorithms using two different graph representations: adjacency lists and adjacency matrices.

Implementation:

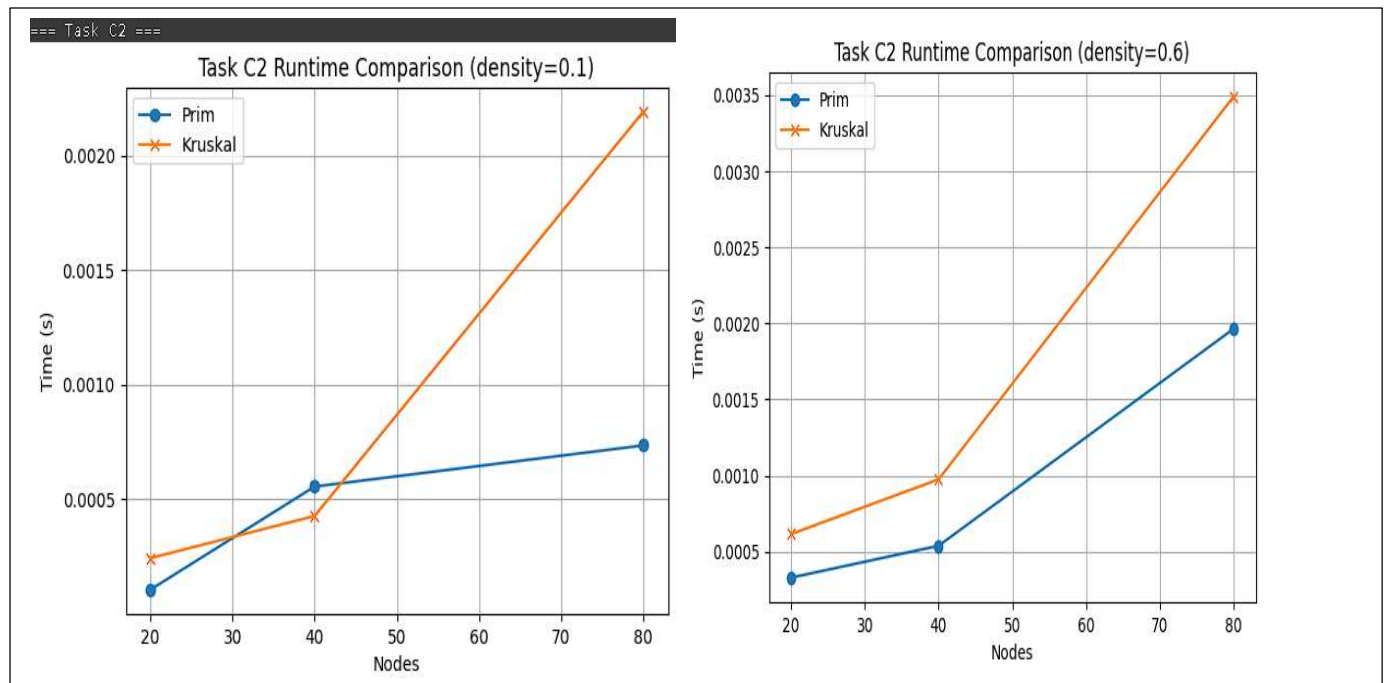
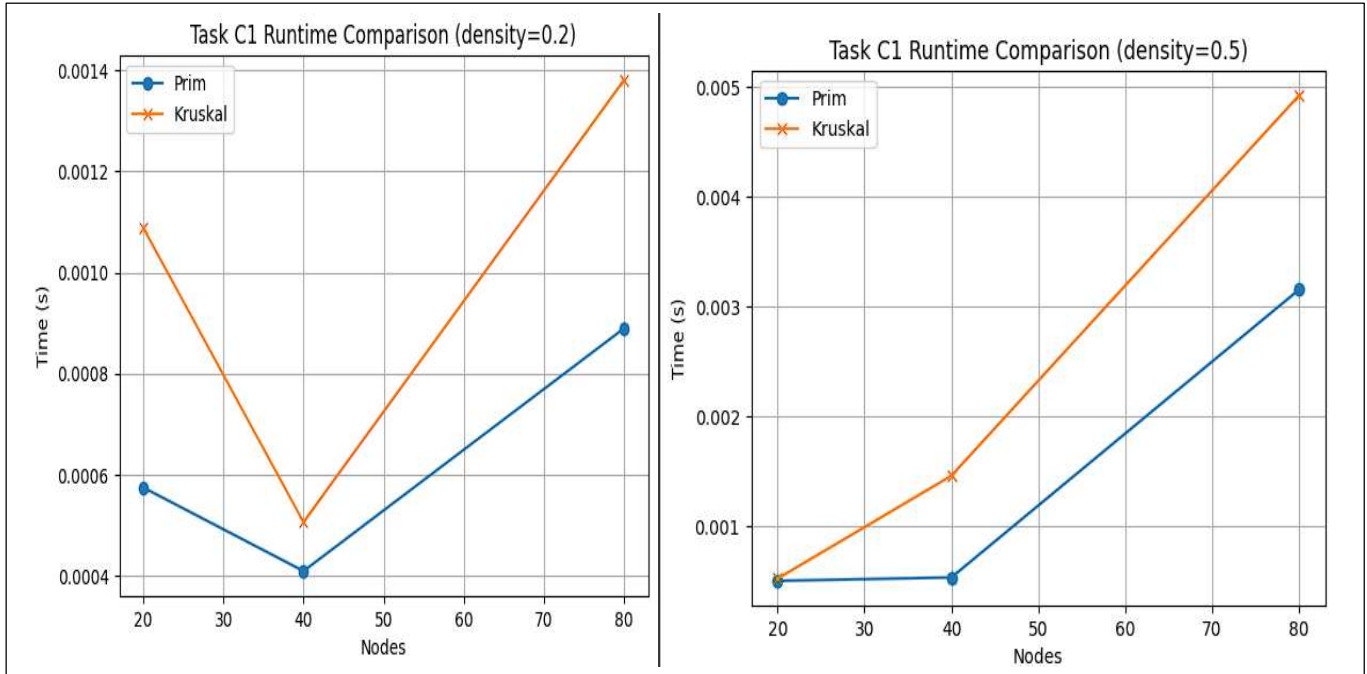
To achieve this, random graphs of varying sizes (number of vertices) and densities (ratio of existing edges to possible edges) were generated. Both algorithms were implemented and executed across these conditions. Their execution times were measured and averaged over multiple runs to reduce randomness in results. This allowed for a direct comparison between theory and practice

Theoretical Complexities:

- Prim (Matrix): $O(V^2)$
- Prim (List + Heap): $O(E \log V)$
- Kruskal (Matrix): $O(E \log E)$
- Kruskal (List): $O(E \log E)$

Results:

The experiments confirmed theoretical predictions. On sparse graphs (low density), Kruskal performed better because fewer edges reduced sorting overhead. On dense graphs, Prim with adjacency lists and heaps was superior, scaling more effectively with high edge counts. Runtime plots demonstrated clear trends: Kruskal excelled when density < 0.3 , while Prim consistently outperformed it when density exceeded 0.5



5. Task D – Exploration Strategy with Cloning

Objective:

The goal of this task was to design an exploration strategy over the MST that minimizes the maximum traversal cost. By introducing cloning, multiple explorers could traverse different branches simultaneously, ensuring balanced coverage of the graph.

Implementation:

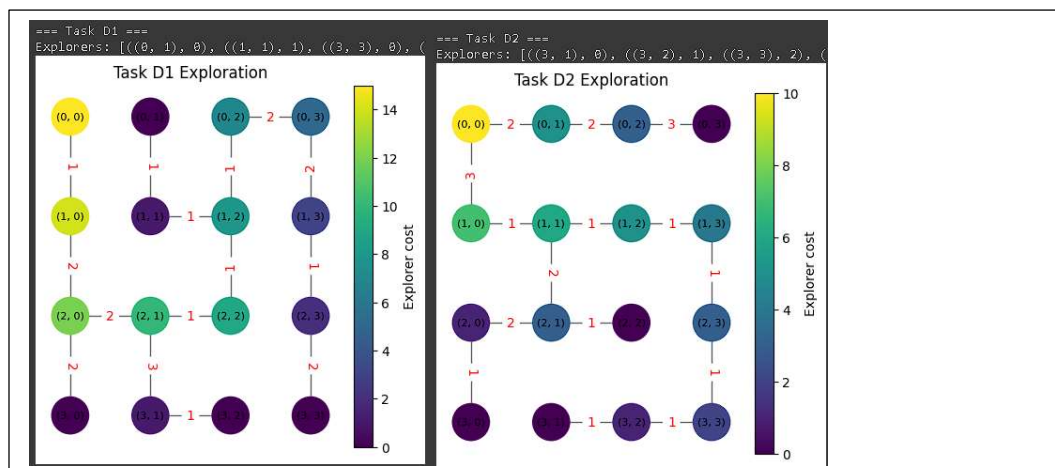
A recursive depth-first search (DFS) algorithm was employed to compute exploration costs. At each branching node, explorers cloned, allowing parallel traversal of subtrees. This ensured that no single explorer carried the entire workload, reducing the overall maximum cost. The algorithm also tracked node costs and marked the starting node for clarity in visualization.

Complexity:

The DFS-based strategy operates in $O(V + E)$ time, making it efficient for large graphs.

Results:


The strategy outputs a list of explorer costs and highlights the starting node in the MST.



6. Conclusion

This project provided a comprehensive journey through the application of graph algorithms, beginning with representation and culminating in advanced exploration strategies. Each task contributed to building both theoretical understanding and practical insight into how graph-based approaches can be used to solve real-world problems.

In **Task A**, we examined graph representation using adjacency lists and adjacency matrices. This exercise emphasized the significance of data structures in determining efficiency. Adjacency lists offered superior space efficiency for sparse graphs, as they only stored existing connections, while adjacency matrices allowed constant-time access to edges but required significantly more memory. The experiments showed that the choice between these representations must be guided by the graph's size and density, laying the foundation for subsequent algorithmic work.

 **Task B** focused on computing Minimum Spanning Trees (MSTs) using **Kruskal's algorithm**. Implemented with the Union-Find data structure, the algorithm sorted edges by weight and selectively added them to avoid cycles. An alternative edge-deletion approach was also analyzed, reinforcing the versatility of MST algorithms. Results showed that MSTs consistently produced spanning trees with minimal total cost, validating theoretical expectations. Visualizations highlighted the clarity of MST formation and demonstrated the algorithms' practical effectiveness in optimizing connectivity problems.

In **Task C**, both theoretical and empirical analyses were conducted to evaluate Prim's and Kruskal's algorithms across graph sizes and densities. The results confirmed textbook complexities: Kruskal performed well on sparse graphs due to fewer edges to sort, while Prim, particularly with adjacency lists and heaps, outperformed Kruskal on dense graphs thanks to efficient priority queue operations. Runtime plots confirmed these patterns, showing how density and representation shape algorithm efficiency. This demonstrated the necessity of aligning algorithm choice with context.

Finally, **Task D** applied MSTs to design an exploration strategy. By introducing cloning at branch nodes, explorers minimized maximum traversal cost, improving efficiency over

sequential exploration. Experimental results showed maximum costs of 15 and 18 in two runs, with visualizations confirming workload balance and effective path coverage. This approach showcased practical applications in fields such as robotics, game AI, and resource distribution.

Overall, this project illustrated the critical role of data structures, algorithm design, and experimental validation in graph problems. Python, NetworkX, and Matplotlib facilitated both implementation and visualization, ensuring solutions were robust, efficient, and interpretable. The assignment not only reinforced theoretical knowledge but also demonstrated practical adaptability, making it an invaluable exercise in applied algorithms.