

Detailed Design and Analysis of Kruskal's Minimum Spanning Tree (MST) Algorithm

Use case: Determining the optimum airline network over a span of Airports

Github repository: <https://github.com/intercogni/hub-and-spokes>

Members:

Taib Izzat Samawi 50% [Code Implementation + Algorithm Concept]

Fawwaz Abdulloh Al-Jawi 25% [Algorithm Concept]

Muhammad Reza Octavianto 25% [Game Concept]

Introduction

In the realm of air transportation, determining the most efficient network of flight routes that connects various airports is an interesting, yet nonetheless crucial task for minimizing operational costs and enhancing connectivity.

The Minimum Spanning Tree (MST) problem in graph theory provides a framework for addressing this challenge. An MST ensures that all airports (nodes) are connected with the minimum possible total flight route distance (edge weights), without forming any cycles.

The application of the Minimum Spanning Tree (MST) algorithm can be seen by how real-life airlines use the **Hub-and-Spokes** connection scheme, in which some airports are reserved to be the hubs to connect multiple edge/peripheral airports.

In this paper, this team is interested in using Kruskal's algorithm—a well-known greedy algorithm that efficiently finds the MST of a graph. This report will detail the design, implementation, and analysis of Kruskal's MST algorithm, specifically applied to finding the optimal network of flight routes between airports.

Libraries and Dependencies

To implement Kruskal's algorithm and visualize them into an easy-to-view interface, this team decided to use these frameworks and or libraries to support its development:

- **python**, used because of its expansive documentation and ease-of-use



- **numpy**, used to facilitate numerical computation and algorithms, and to provide a visualization endpoint for the graph

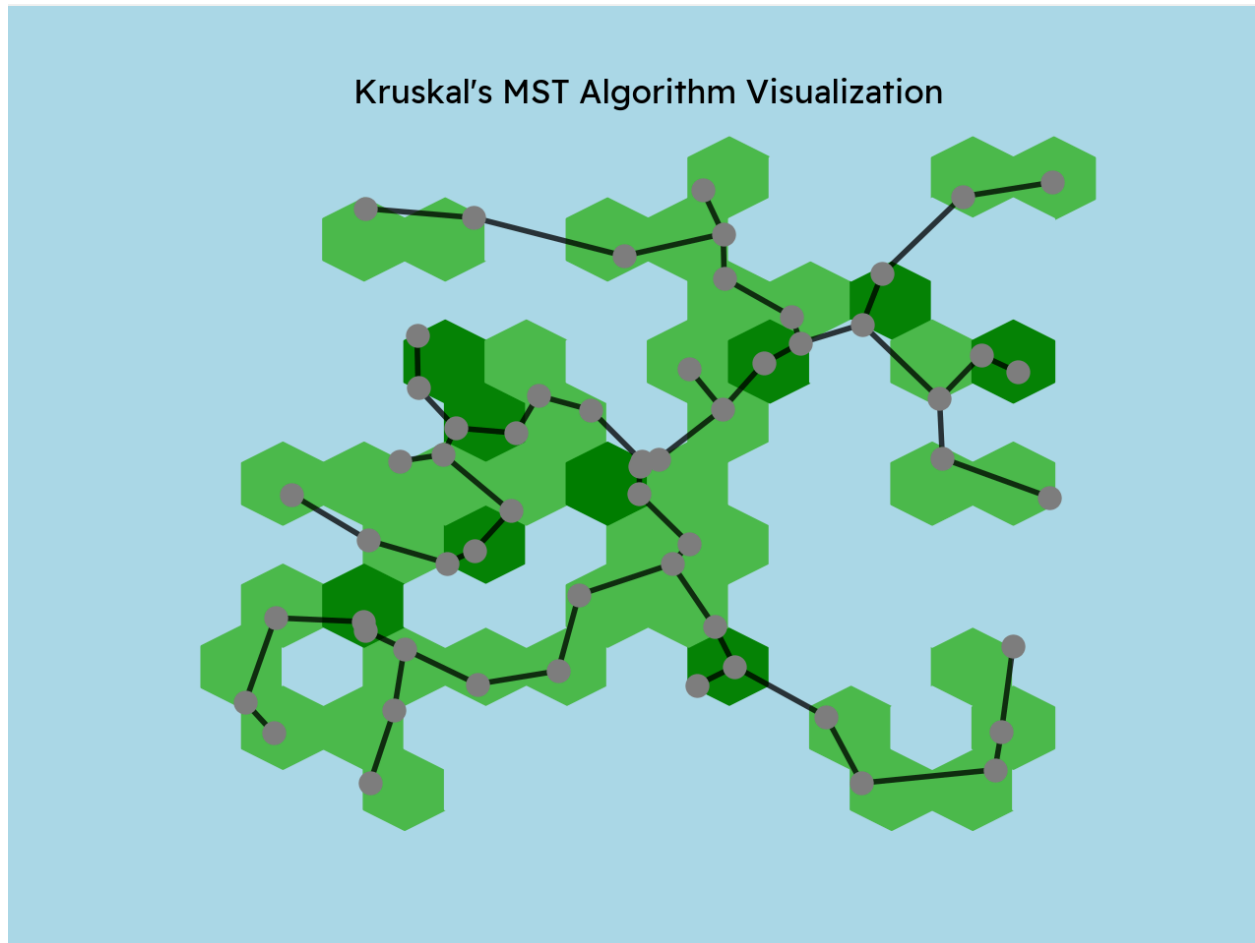


- **networkx**, to create, manipulate, and visualize graphs



Graph Representation

Airports are represented as nodes, each with a random position in a 2D space to facilitate visualization. Flight routes between each pair of airports are represented as edges, with weights determined by the Euclidean distance between the airport positions.



We also used a hexbin to create a pseudo-map visualizing the continents where the airports are located on.

Below is the program for the initial setup of airports and flight routes generation:

```
import networkx as nx
import matplotlib.pyplot as plt
import math
import random
import numpy as np
import matplotlib.colors as mcolors
from netgraph import Graph

# Custom color map for visualization
colors = [(0, 'lightblue'), (0.01, 'lightgreen'), (0.7, 'green'), (1, 'green')]
custom_cmap = mcolors.LinearSegmentedColormap.from_list("", colors)

def euclidean_distance(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

# Create a graph and add nodes with random positions
G = nx.Graph()
graph_data = []
edge_data = []
for i in range(ord('A'), ord('z') + 1):
    graph_data.append(
        (chr(i), {'pos': (random.randint(1, 10000), random.randint(1, 10000))})
    )
G.add_nodes_from(graph_data)

# Add edges with weights based on Euclidean distance
for i in range(ord('A'), ord('z') + 1):
    for j in range(ord('A'), ord('z') + 1):
        if i != j:
            edge_data.append((
                chr(i),
                chr(j),
                euclidean_distance(
                    G.nodes[chr(i)]['pos'],
                    G.nodes[chr(j)]['pos']
                )
            ))
G.add_weighted_edges_from(edge_data)
```

Kruskal's MST Algorithm

Kruskal's algorithm finds the MST by sorting all flight routes in non-decreasing order of their distances and adding them to the MST if they do not form a cycle. This requires an efficient method to manage the connected components of the airport network, typically achieved using a union-find data structure.

Steps of Kruskal's Algorithm

1. **Edge Sorting:** All flight routes are sorted based on their distances.
2. **Union-Find Data Structure:** This structure keeps track of the connected components to ensure no cycles are formed when adding routes.
3. **MST Construction:** Iterate through the sorted routes and add each route to the MST if it connects two different components.

The following code implements Kruskal's algorithm and visualizes the process:

```
def kruskal_mst(graph):
    edges = sorted(graph.edges(data=True), key=lambda x: x[2]['weight'])
    mst = nx.Graph()
    mst.add_nodes_from(graph.nodes())
    plt.gcf().set_facecolor('lightblue')

    # Union-Find structure for components
    components = {node: {node} for node in graph.nodes()}
    for edge in edges:
        u, v, weight = edge
        if components[u] != components[v]:
            mst.add_edge(u, v, weight=weight['weight'])
            components[u] |= components[v]
            for node in components[v]:
                components[node] = components[u]

    # Visualization
    plt.clf()
    x = [graph.nodes[node]['pos'][0] for node in graph.nodes()]
    y = [graph.nodes[node]['pos'][1] for node in graph.nodes()]

    hb = plt.hexbin(x, y, gridsize=10, cmap=custom_cmap, extent=(0,
10000, 0, 10000))

    pos = nx.get_node_attributes(G, 'pos')
    nx.draw_networkx_nodes(G, pos, node_size=60, node_color='gray')
    nx.draw_networkx_edges(
```

```

        mst,
        pos,
        width=2.0,
        alpha=0.75,
        edge_color='black',
    )
    csfont = {'fontname': 'Readex Pro'}
    plt.title("Kruskal's MST Algorithm Visualization", **csfont)
    plt.axis('off')
    plt.pause(0.33333)
    return mst

fig = plt.figure()
mst = kruskal_mst(G)
plt.show()

```

Detailed Analysis

Complexity Analysis

Time Complexity

- **Sorting the Routes:** Sorting the flight routes takes $O(E \log E)$, where E is the number of routes.
- **Union-Find Operations:** Union and find operations are nearly constant time, $O(\alpha(V))$, where α is the **inverse Ackermann** function, making it extremely efficient.
- **Overall Time Complexity:** The overall time complexity is $O(E \log E)$.

Space Complexity

- **Graph Storage:** The algorithm requires space for storing the airport network, routes, and the union-find data structure.
- **Overall Space Complexity:** The space complexity is $O(V + E)$, where V is the number of airports.

Visualization

Dynamic Updates

The visualization updates dynamically to reflect the addition of routes to the MST.

Hexbin Plot

A hexbin plot is utilized to create a pseudo-map of the continents that the airports are going to be located on.

Node and Edge Drawing

Airports are drawn as gray circles, and MST routes are drawn as black lines.

Observations

- **Greedy Approach:** Kruskal's algorithm's greedy nature ensures that at each step, the minimum distance route is considered, leading to an optimal solution.
- **Union-Find Efficiency:** The union-find structure efficiently manages the components, preventing cycles and maintaining the MST properties.
- **Scalability:** The algorithm handles large networks of airports efficiently due to its $O(E \log E)$ complexity, making it suitable for extensive flight route analysis.

Customization and Enhancements

- **Font and Titles:** Customized fonts and titles improve the readability and aesthetics of the visual output.
- **Random Positions:** Each run generates random positions for airports, showcasing the algorithm's adaptability to different network structures.

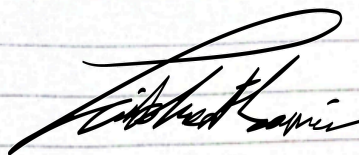
Conclusion

Kruskal's MST algorithm is a robust and efficient method for finding the optimal network of flight routes in a weighted graph. Its greedy strategy, coupled with the efficient union-find data structure, ensures optimal performance. The detailed implementation and visualization demonstrate the algorithm's step-by-step process, providing insights into its working mechanism and efficiency. This report offers a comprehensive overview of Kruskal's algorithm, highlighting its theoretical foundations, practical implementation, and analytical insights, tailored specifically for determining the best flight routes between airports.

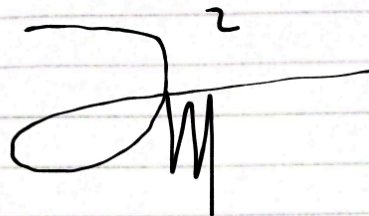
Date

By the name of Allah Almighty, herewith We pledge and truly declare that We worked on Quiz 2 by ourselves, did not do any cheating by any means, did not do any Plagiarism, and did not accept anyone's help by any means. I am going to accept all of the consequences by any means if it has proven that We have done any cheating and/or plagiarism.

Surabaya, 24 May 2024



Taib Izzat
Samawi
5025221085



Fawwaz Abdulloh
Al-jawi
5025221075



Muhammad Reza
Octavianito
5025221074