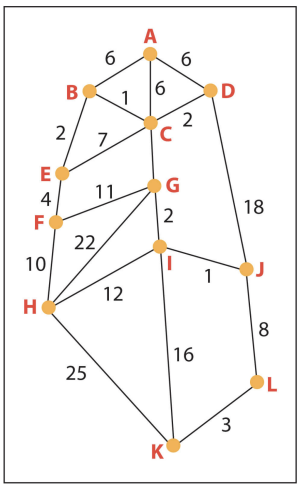
Design Analysis and Algorithem, Lab Assignment 10

Bhuvana Kanakam - SE21UCSE035

November 16, 2023

Question

Assignment Consider a scenario where a city planner is tasked with connecting a set of neighborhoods through a network of roads. Each road between neighborhoods has an associated cost, representing the expenses involved in construction. The objective is to design a road network that connects all neighborhoods while minimizing the total construction cost. Use Kruskal's algorithm for optimizing road network connectivity. Also, analyze the time complexity of the implemented algorithm.



Answer

Kruskal's greedy algorithm finds a minimum spanning tree for a weighted, undirected graph. The algorithm starts with a forest consisting of the individual nodes of the graph and then finds the cheapest edge from each node and adds it to the forest. This process is repeated until only one tree is in the forest, the minimum spanning tree.

Kruskal's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph. This input forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm works by sorting the graph's edges by weight, then taking the edge with the lowest weight from the graph and adding it to the tree. This process repeats until all the vertices are included in the tree.

Python Code

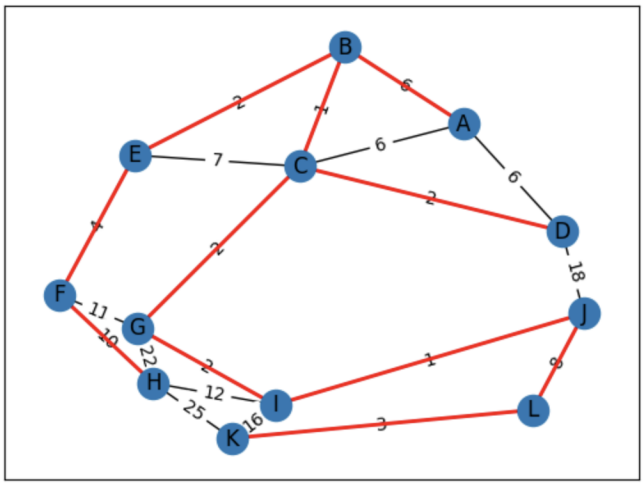
```
import networkx as nx
import matplotlib.pyplot as plt

class UnionFind:
    def __init__(self, vertices):
        self.parent = {vertex: vertex for vertex in vertices}
```

```
        self.rank = {vertex: 0 for vertex in vertices}

    def find(self, vertex):
        if self.parent[vertex] != vertex:
            self.parent[vertex] = self.find(self.parent[vertex])
        return self.parent[vertex]

    def union(self, root1, root2):
        if self.rank[root1] < self.rank[root2]:
            self.parent[root1] = root2
        elif self.rank[root1] > self.rank[root2]:
            self.parent[root2] = root1
        else:
            self.parent[root1] = root2
            self.rank[root2] += 1

def kruskal(graph):
    edges = [(cost, u, v) for (u, v), cost in graph.items()]
    edges.sort()

    minimum_spanning_tree = set()
    vertices = set()

    for edge in edges:
        cost, u, v = edge
        root1 = union_find.find(u)
        root2 = union_find.find(v)

        if root1 != root2:
            minimum_spanning_tree.add((u, v, cost))
            union_find.union(root1, root2)
            vertices.add(u)
            vertices.add(v)

    return minimum_spanning_tree, vertices

graph = {
    ('A', 'B'): 6, ('A', 'C'): 6, ('A', 'D'): 6,
    ('B', 'C'): 1, ('B', 'E'): 2,
    ('C', 'E'): 7, ('C', 'D'): 2, ('C', 'G'): 2,
    ('D', 'J'): 18,
    ('E', 'F'): 4,
    ('F', 'G'): 11, ('F', 'H'): 10,
    ('G', 'H'): 22, ('G', 'I'): 2,
    ('H', 'I'): 12, ('I', 'J'): 1, ('I', 'K'): 16, ('H', 'K'): 25,
    ('J', 'L'): 8,
    ('K', 'L'): 3
}

union_find = UnionFind(set(node for edge in graph.keys() for node in edge))

minimum_spanning_tree, vertices = kruskal(graph)

print("Minimum Spanning Tree:")
for edge in minimum_spanning_tree:
    print(edge)

print("\nVertices in the Minimum Spanning Tree:", vertices)
```

```
def plot_graph(graph, minimum_spanning_tree):
    G = nx.Graph()

    for edge, cost in graph.items():
        G.add_edge(edge[0], edge[1], weight=cost)

    pos = nx.spring_layout(G, seed=42)

    labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_nodes(G, pos)
    nx.draw_networkx_edges(G, pos)
    nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
    nx.draw_networkx_labels(G, pos)

    mst_edges = [(edge[0], edge[1]) for edge in minimum_spanning_tree]
    nx.draw_networkx_edges(G, pos, edgelist=mst_edges, edge_color='r', width=2)

    plt.show()

plot_graph(graph, minimum_spanning_tree)
```

Output of the Code:

```
Minimum Spanning Tree:
('C', 'G', 2)
('E', 'F', 4)
('B', 'E', 2)
('K', 'L', 3)
('A', 'B', 6)
('J', 'L', 8)
('G', 'I', 2)
('F', 'H', 10)
('C', 'D', 2)
('B', 'C', 1)
('I', 'J', 1)
Vertices in the Minimum Spanning Tree: {'A', 'L', 'G', 'C', 'E', 'K', 'D', 'H', 'J', 'F', 'B', 'I'}
```



Analysis of the Algorithm:

Kruskal's algorithm is an algorithm to find the minimum spanning tree of a graph. It makes use of the fact that the edges of a minimum spanning tree must form a subset of the edges of any other spanning tree.

The time complexity of Kruskal's Algorithm is **O(ElogE)**, where E is the number of edges in the graph.

Written Working of the Algorithm