Kruskals

class DisjointSet:

def \_\_init\_\_(self, n):

self.parent = list(range(n))

def find(self, x):

if self.parent[x] != x:

self.parent[x] = self.find(self.parent[x]) # Path compression

return self.parent[x]

def union(self, x, y):

xroot = self.find(x)

yroot = self.find(y)

if xroot == yroot:

return False

self.parent[yroot] = xroot

return True

def kruskal\_mst(edges, n):

# Sort edges based on weight

edges.sort(key=lambda x: x[2])

ds = DisjointSet(n)

mst = []

total\_weight = 0

for u, v, weight in edges:

if ds.union(u, v):

mst.append((u, v, weight))

total\_weight += weight

return mst, total\_weight

# Define edges (u, v, weight)

edges = [

(0, 1, 4),

(0, 7, 8),

(1, 2, 8),

(1, 7, 11),

(2, 3, 7),

(2, 8, 2),

(2, 5, 4),

(3, 4, 9),

(3, 5, 14),

(4, 5, 10),

(5, 6, 2),

(6, 7, 1),

(6, 8, 6),

(7, 8, 7),

]

n = 9 # Number of vertices (0 to 8)

mst, total\_weight = kruskal\_mst(edges, n)

print("Edges in MST:")

for u, v, weight in mst:

print(f"{u} - {v}: {weight}")

print(f"Total weight of MST: {total\_weight}")

theory

Kruskal’s Algorithm – Theory

Kruskal’s Algorithm is a Greedy algorithm used to find the Minimum Spanning Tree (MST) of a connected, undirected, weighted graph. It is an alternative to Prim's Algorithm and is often preferred when the graph is sparse.

Steps of Kruskal’s Algorithm:

Sort all edges of the graph in increasing order of weight.

Initialize a Disjoint Set (Union-Find) to keep track of connected components.

Iterate through the sorted edges:

For each edge (u, v) with weight w:

If u and v are in different sets (i.e., not connected yet):

Add the edge to the MST.

Perform a union operation to merge the sets of u and v.

If u and v are already in the same set, skip the edge to avoid a cycle.

Stop when the MST contains exactly V-1 edges (where V is the number of vertices).

Return the MST and its total weight.