Name : Eslam Mohamed Abbas Abbas Hammad
Section : 2
Department : CS
ID:1000263795

Name : Hanin Baher Elsaid Ali Hamza
Section : 2
Department : CS
ID:1000263927

(a) Algorithms for Kruskal's Algorithm

To find the Minimum Spanning Tree (MST) using Kruskal's Algorithm, we require the following algorithms:

Kruskal's Algorithm: Finds the MST by repeatedly adding the smallest edge to the tree while ensuring no cycles are formed.

Union-Find (Disjoint Set): Ensures cycle detection and efficient merging of sets.

Algorithm 1: Kruskal's Algorithm

—-------------------------------------

Kruskal(G)
Input: A graph G with vertices V and edges E (each edge has weight w).
Output: A Minimum Spanning Tree (MST).

1. Initialize MST as an empty set.
2. Sort all edges in non-decreasing order of their weights.
3. Create a disjoint-set for each vertex in G.
4. for each edge (u, v) in the sorted edge list:
5. if Find(u) ≠ Find(v):
6. Add (u, v) to MST.
7. Union(u, v).
8. Return MST.

Algorithm 2: Union-Find (Disjoint Set)

—---

The union-find data structure is used for cycle detection and managing connected components.
Find with Path Compression

Find(x)
Input: An element x.
Output: The representative of the set containing x.

1. if parent[x] ≠ x:
2. parent[x] = Find(parent[x]) # Path compression

3. Return parent[x].

Union by Rank

Union(x, y)
Input: Two elements x and y.
Output: Combines the sets containing x and y.

1. rootX = Find(x).
2. rootY = Find(y).
3. if rootX ≠ rootY:
4. if rank[rootX] > rank[rootY]:
5. parent[rootY] = rootX.
6. else if rank[rootX] < rank[rootY]:
7. parent[rootX] = rootY.
8. else:
9. parent[rootY] = rootX.
10. rank[rootX] += 1.

—---

(b) Analysis of Kruskal's Algorithm

1. Time Complexity

Sorting edges: O(ElogE)O(ElogE), where EE is the number of edges.

Union-Find operations: Each union and find operation runs in O(α(V))O(α(V)), where αα is the inverse Ackermann function (very small, effectively constant). For EE edges, this contributes O(Eα(V))O(Eα(V)).

Overall complexity:
O(ElogE+Eα(V))=O(ElogE)O(ElogE+Eα(V))=O(ElogE)
Since E≥V−1E≥V−1 for a connected graph, the complexity can also be written as O(ElogV)O(ElogV).

2. Space Complexity

Space for storing edges: O(E)O(E).

Space for disjoint-set data structure (parent and rank arrays): O(V)O(V).

Total space complexity: O(V+E)O(V+E).

3. Properties

Greedy: Kruskal’s algorithm works by greedily adding edges of the smallest weight while avoiding cycles.

Suitable for sparse graphs: EE is much smaller than V2V2.Output for Example

For the input graph with edges:
(0,1,10),(0,2,6),(0,3,5),(1,3,15),(2,3,4)(0,1,10),(0,2,6),(0,3,5),(1,3,15),(2,3,4)

The output is:
Edges in MST: [(2, 3, 4), (0, 3, 5), (0, 1, 10)]
Total weight of MST: 19

—---

(c) Code Of MST using Kruskal’s Algorithm

```
class DisjointSet:
	def _init_(self, n):
	self.parent = list(range(n))
	self.rank = [0] * n

	def find(self, x):
	"""Find with path compression."""
	if self.parent[x] != x:
		self.parent[x] = self.find(self.parent[x])  # Path compression
	return self.parent[x]

	def union(self, x, y):
	"""Union by rank."""
	rootX = self.find(x)
	rootY = self.find(y)

	if rootX != rootY:
		if self.rank[rootX] > self.rank[rootY]:
			self.parent[rootY] = rootX
		elif self.rank[rootX] < self.rank[rootY]:
			self.parent[rootX] = rootY
		else:
			self.parent[rootY] = rootX
			self.rank[rootX] += 1

def kruskal(edges, n):
	"""Finds the MST using Kruskal's algorithm."""
	# Sort edges by weight
	edges.sort(key=lambda edge: edge[2])
	ds = DisjointSet(n)
	mst = []
	mst_weight = 0

	for u, v, w in edges:
	# If u and v belong to different sets, add the edge to MST
```

```python
    if ds.find(u) != ds.find(v):
        mst.append((u, v, w))
        mst_weight += w
        ds.union(u, v)

    return mst, mst_weight

# Example Usage
edges = [
    (0, 1, 10),
    (0, 2, 6),
    (0, 3, 5),
    (1, 3, 15),
    (2, 3, 4)
]
n = 4  # Number of vertices

mst, total_weight = kruskal(edges, n)
print("Edges in MST:", mst)
print("Total weight of MST:", total_weight)
```

——---