### Algorithm: Kruskal's Algorithm

Kruskal's algorithm is a greedy algorithm that finds a minimum spanning tree for a connected weighted graph.

#### Initialize Data Structures:

* Use a list to keep track of the edges in the minimum spanning tree (MST).
* Use a union-find data structure to keep track of the connected components.

#### Sort and Select Edges:

* Sort the edges by weight.
* Add the smallest edge to the MST if it doesn't form a cycle.

#### Repeat:

* Repeat the process until the MST contains (V-1) edges, where (V) is the number of vertices.

#### Implementation:

| **class** UnionFind:  **def** \_\_init\_\_(self, size):  self.parent = list(range(size))  self.rank = [0] \* size  **def** find(self, u):  **if** self.parent[u] != u:  self.parent[u] = self.find(self.parent[u])  **return** self.parent[u]  **def** union(self, u, v):  root\_u = self.find(u)  root\_v = self.find(v)  **if** root\_u != root\_v:  **if** self.rank[root\_u] > self.rank[root\_v]:  self.parent[root\_v] = root\_u  **elif** self.rank[root\_u] < self.rank[root\_v]:  self.parent[root\_u] = root\_v  **else**:  self.parent[root\_v] = root\_u  self.rank[root\_u] += 1  **def** kruskal(graph: List[Tuple[int, int, int]], V: int) -> List[Tuple[int, int, int]]:  mst = []  uf = UnionFind(V)  graph.sort(key=**lambda** x: x[2])  **for** u, v, w **in** graph:  **if** uf.find(u) != uf.find(v):  uf.union(u, v)  mst.append((u, v, w))  **return** mst  *# Example usage:*  graph = [(0, 1, 10), (0, 2, 6), (0, 3, 5), (1, 3, 15), (2, 3, 4)]  V = 4  print(kruskal(graph, V)) *# Output: Minimum Spanning Tree edges* |
| --- |

#### Explanation:

Initialize:

* mst: A list to keep track of the edges in the MST.
* uf: A union-find data structure to keep track of the connected components.

Sort and Select Edges:

* Sort the edges by weight.
* Add the smallest edge to the MST if it doesn't form a cycle.

Repeat:

* Continue the process until the MST contains (V-1) edges