Physics class

LAB ASSIGNMENT - 7

21BCE9905

K. MadhuSudhana Rao

1. Write a program to find Minimum Spanning tree using Kruskal Algorithm. (using the Greedy approach).

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def add_edges(self, u, v, w):
        self.graph.append([u, v, w])

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

    def union(self, parent, rank, x, y):
        root_x = self.find(parent, x)
        root_y = self.find(parent, y)
        if rank[root_x] < rank[root_y]:
            parent[root_x] = root_y
        elif rank[root_x] > rank[root_y]:
            parent[root_y] = root_x
        else:
            parent[root_y] = root_x
            rank[root_x] += 1
```

```
def kruskal_mst(self):                      # 21B.CE0090
    result = []
    self.graph.sort(key = lambda item: item[2])
    parent = [0]* self.v

    for u, v, w in self.graph:
        root_u = self.find(parent, u)
        root_v = self.find(parent, v)

        if (root_u != root_v:
            result.append([u, v, w])
            self.union(parent, rank, root_u, root_v)

    return result.
# Example usage:
g = Graph(4)
g.add_edge(0, 1, 10)
g.add_edge(0, 2, 6)
g.add_edge(0, 3, 5)
g.add_edge(1, 3, 15)
mst = g.kruskal_mst()
print("Minimum Spanning Tree edges:")
for u, v, w in mst:
    print(f"{u} - {v}: {w}")
```

2. Write a program to implement the single source shortest problem (using the greedy approach) i.e.,

wri

Class Graph:

21BEE9??

```
    def __init__(self, vertices):
        Self.V = vertices
        Self.graph = [[] for _ in range(vertices)]

    def add_edge(self, u, v, w):
        Self.graph[u].append((v, w))

    def dijkstra(self, src):
        dist = [float('inf')] * self.V
        dist[src] = 0
        pq = [(0, src)]
        while pq:
            d, u = heapq.heappop(pq)
            for v, w in self.graph[u]:
                if dist[v] > dist[u] + w:
                    dist[v] = dist[u] + w
                    heapq.heappush(pq, (dist[v], v))

        return dist.

g = Graph(5).
g.add_edge(0, 1, 10)
g.add_edge(0, 2, 5)
g.add_edge(1, 3, 2)
g.add_edge(2, 1, 3)
g.add_edge(2, 3, 9)
Shortest_paths = g.dijkstra(0).
print("Shortest paths from vertex 0:")
for i, dist in enumerate(Shortest_paths):
    print(f"vertex{i}: {dist}")
```

3. Write a program to solve 0/1 knapsack problem using Dynamic programming algorithm.

```
def knapsack(weights, values, capacity):   # 2us(E990(
    n= len(weights)
    dp = [[0]* (capacity+1) for_in. range(n+1)]
  for i in range(n+1):
      for w in rang(capacity +1):
          if i==0 or w==0:
              dp[i][w] =0
          elif weights[i-1]<=w;
              dp[i][w] = max(values[i-1]+dp[i-1][w-
                         weights[i-1]], dp[i-1][w])
          else:
              dp[i][w] = dp[i-1][w]

  return dp [n][capacity]

weights = [2, 3, 4,5]
values = [3, 4, 5, 6]
capacity = 5
max_value = Knap Saia (weigha, values, capacity)
print (f" Maximum value achieveble: {max_values}")
```