Practical work nr.4

```
def find(self, parent, i): # this function finds the parent of a node
    if parent[i] != i:
        parent[i] = self.find(parent, parent[i]) # we go recursively, untill we
find the root of the subtree
    return parent[i]

def union(self, parent, rank, x, y): # this function unites two subtrees, the
one which has the bigger rank will be the root

    if rank[x] < rank[y]:
        parent[x] = y
    elif rank[x] > rank[y]:
        parent[y] = x

    else:
        parent[y] = x
        rank[x] += 1

def Kruskal(self):
    result = []
    costs = sorted(self.__costs.items(), key=lambda cost: cost[1])

    new_cost=[]
    for i in costs:
        new_cost.append([i[0][0],i[0][1],i[1]]);
    for i in new_cost:
        j=[i[1],i[0],i[2]]
        if j in new_cost:
            new_cost.remove(j)
    for i in new_cost:
        print(i)
        # in the functions above we transformed the cost dictionary to a list,
containing [start,end,cost] pairs ordered by cost
    e=0
    i=0
    parent=[]
    rank=[]
    for node in range(self.__vertices):
        parent.append(node)
        rank.append(0)
    while e<self.__vertices-1: # the MST should have nr. vertices - 1 edges
        u,v,w=new_cost[i] # we choose the yet unvisited edge with the smallest
cost
        i = i + 1
        x = self.find(parent, u) # we find the parents of the nodes -> the roots
of their corresponding subtrees
        y = self.find(parent, v)

        if x != y: # if the two nodes are not part of the same subtree, we unite
the two subtrees
            e = e + 1
            result.append([u, v, w])
            self.union(parent, rank, x, y)
        # Else discard the edge

    minimum_cost = 0
    print("Edges in the constructed MST")
    for u, v, weight in result:
        minimum_cost += weight
        print("%d -- %d == %d" % (u, v, weight))
    print("Minimum Spanning Tree", minimum_cost)
```