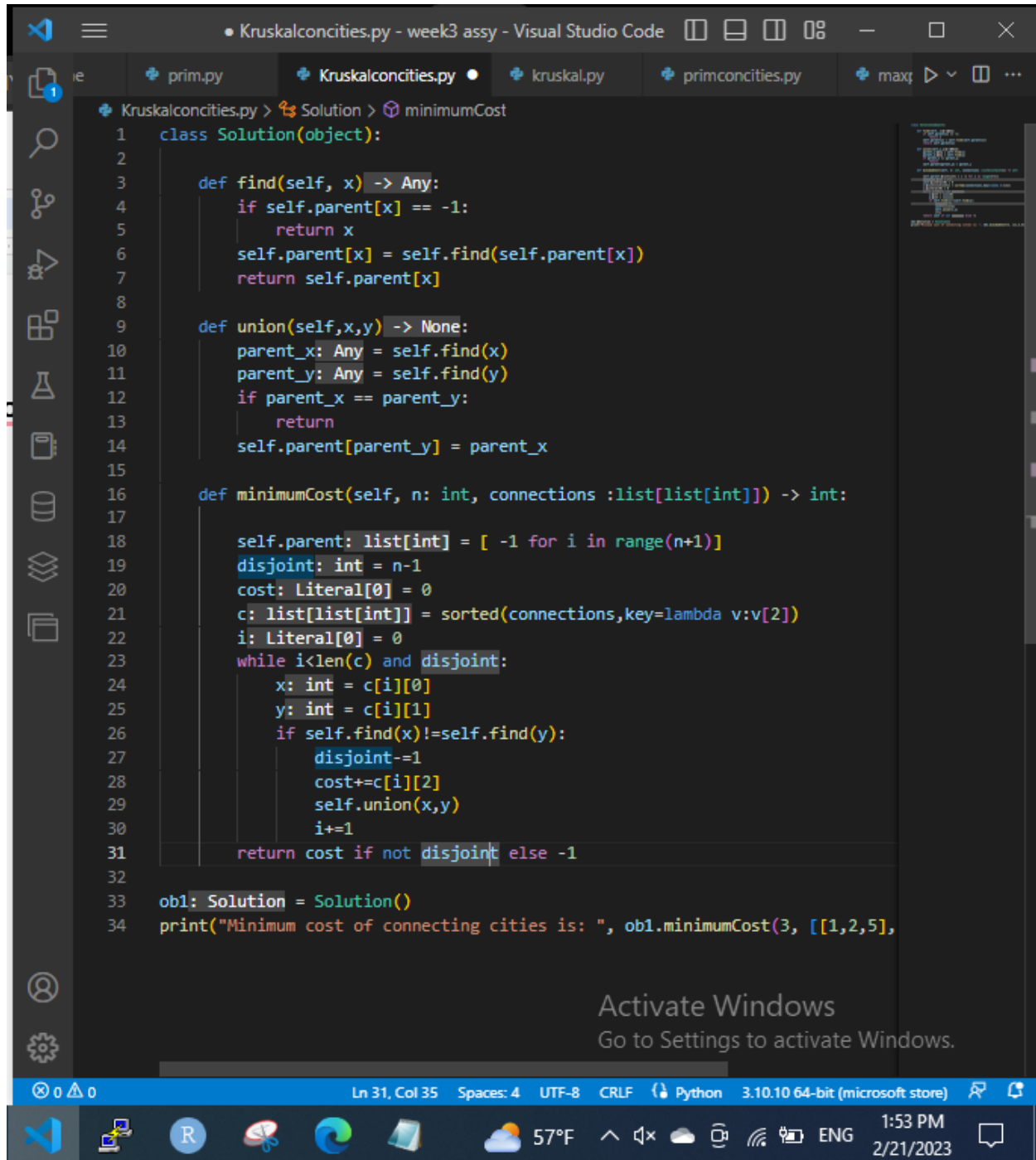


Week 5 Homework 2: Greedy Algorithm : Minimum Spanning Tree (Kruskal) - LC

Step 1



```
1 class Solution(object):
2
3     def find(self, x) -> Any:
4         if self.parent[x] == -1:
5             return x
6         self.parent[x] = self.find(self.parent[x])
7         return self.parent[x]
8
9     def union(self,x,y) -> None:
10        parent_x: Any = self.find(x)
11        parent_y: Any = self.find(y)
12        if parent_x == parent_y:
13            return
14        self.parent[parent_y] = parent_x
15
16    def minimumCost(self, n: int, connections :list[list[int]]) -> int:
17
18        self.parent: list[int] = [-1 for i in range(n+1)]
19        disjoint: int = n-1
20        cost: Literal[0] = 0
21        c: list[list[int]] = sorted(connections,key=lambda v:v[2])
22        i: Literal[0] = 0
23        while i<len(c) and disjoint:
24            x: int = c[i][0]
25            y: int = c[i][1]
26            if self.find(x)!=self.find(y):
27                disjoint-=1
28                cost+=c[i][2]
29                self.union(x,y)
30                i+=1
31        return cost if not disjoint else -1
32
33    ob1: Solution = Solution()
34    print("Minimum cost of connecting cities is: ", ob1.minimumCost(3, [[1,2,5],
```

Kruskalconcities.py - week3 assy - Visual Studio Code

prim.pyKruskalconcities.pyXkruskal.pyprimconcities.pymaxg

Kruskalconcities.py > Solution > minimumCost

1 class Solution(object):

PROBLEMSOUTPUTDEBUG CONSOLETERMINAL

Python + -

PS C:\Users\melan\Desktop\Algorithms\week3 assy> cd "c:/Users/melan/Desktop/Algorithms/week3 assy"

PS C:\Users\melan\Desktop\Algorithms\week3 assy> & C:/Users/melan/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/melan/Desktop/Algorithms/week3 assy/Kruskalconcities.py"

Minimum cost of connecting cities is: 6

PS C:\Users\melan\Desktop\Algorithms\week3 assy>

Activate Windows

Error running mypy in c:\Users\melan\Desktop\Algorithms\week3 assy...
Go to Settings to activate Windows.

Ln 31, Col 35Spaces: 4UTF-8CRLFPython3.10.10 64-bit (microsoft store)

1:54 PM2/21/2023

Code

```
class Solution(object):

    def find(self, x):
        if self.parent[x] == -1:
            return x
        self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self,x,y):
        parent_x = self.find(x)
        parent_y = self.find(y)
        if parent_x == parent_y:
            return
        self.parent[parent_y] = parent_x

    def minimumCost(self, n: int, connections :list[list[int]]) -> int:

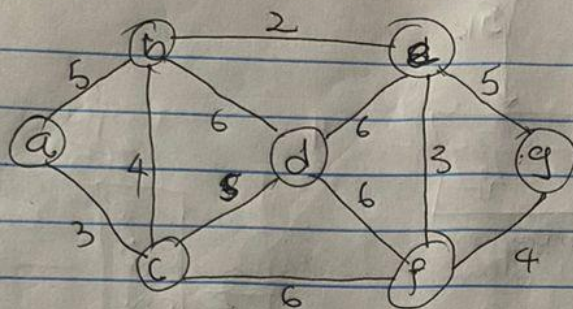
        self.parent = [-1 for i in range(n+1)]
        disjoint = n-1
        cost = 0
        c = sorted(connections,key=lambda v:v[2])
        i = 0
        while i<len(c) and disjoint:
            x = c[i][0]
            y = c[i][1]
            if self.find(x)!=self.find(y):
                disjoint-=1
                cost+=c[i][2]
                self.union(x,y)
                i+=1
        return cost if not disjoint else -1

ob1 = Solution()

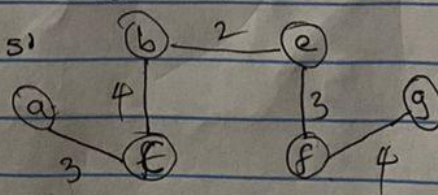
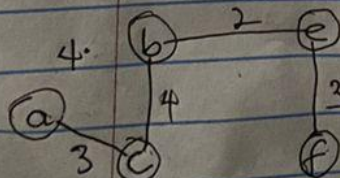
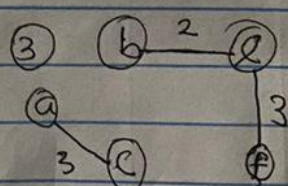
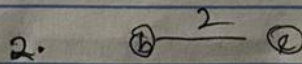
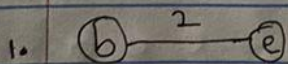
print("Minimum cost of connecting cities is: ", ob1.minimumCost(3, [[1,2,5], [1,3,6], [2,3,1]]))
```

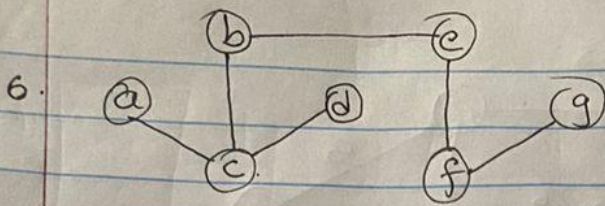
STEP 2

Kruskal



Weight	Src	Dest	
2	b	e	✓
3	a	c	✓
3	e	f	✓
4	b	c	✓
4	f	g	✓
5	a	b	x
5	c	d	✓
5	e	g	x
6	b	d	x
6	d	e	x
6	d	f	x
6	c	f	x





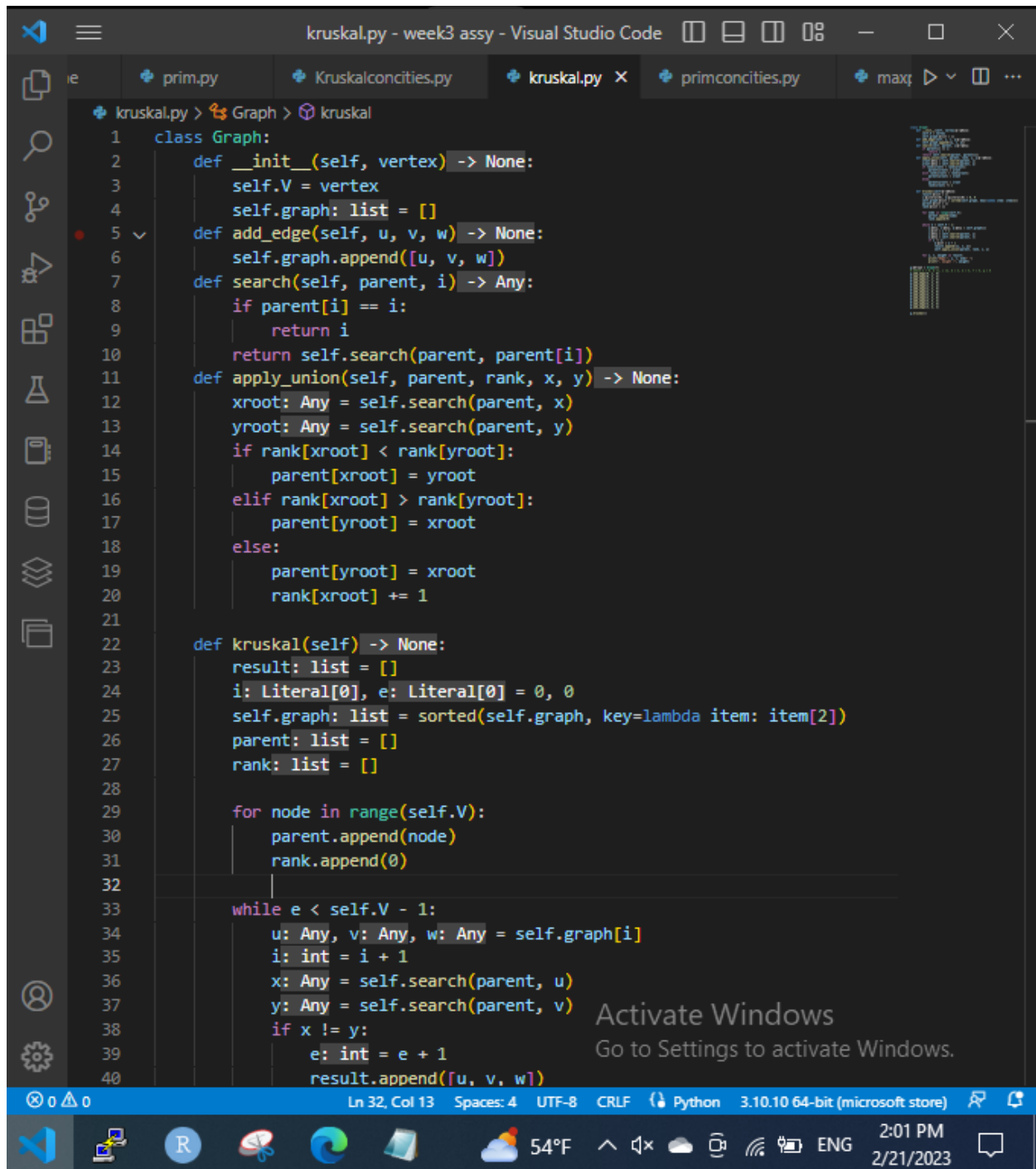
Every other step results in a loop being formed
So we skip them.

Step 6 is the final answer

~~Time complexity = $O(E)$~~

Time complexity = $O(E \log V)$ because in Kruskal's algorithm, most time consuming operation is sorting because of the total complexity of the disjoint set operations.

STEP 3



```
kruskal.py - week3 assy - Visual Studio Code
prim.py  Kruskalconcities.py  kruskal.py x  primconcities.py  maxt

kruskal.py > Graph > kruskal
1 class Graph:
2     def __init__(self, vertex) -> None:
3         self.V = vertex
4         self.graph: list = []
5     def add_edge(self, u, v, w) -> None:
6         self.graph.append([u, v, w])
7     def search(self, parent, i) -> Any:
8         if parent[i] == i:
9             return i
10        return self.search(parent, parent[i])
11    def apply_union(self, parent, rank, x, y) -> None:
12        xroot: Any = self.search(parent, x)
13        yroot: Any = self.search(parent, y)
14        if rank[xroot] < rank[yroot]:
15            parent[xroot] = yroot
16        elif rank[xroot] > rank[yroot]:
17            parent[yroot] = xroot
18        else:
19            parent[yroot] = xroot
20            rank[xroot] += 1
21
22    def kruskal(self) -> None:
23        result: list = []
24        i: Literal[0], e: Literal[0] = 0, 0
25        self.graph: list = sorted(self.graph, key=lambda item: item[2])
26        parent: list = []
27        rank: list = []
28
29        for node in range(self.V):
30            parent.append(node)
31            rank.append(0)
32
33        while e < self.V - 1:
34            u: Any, v: Any, w: Any = self.graph[i]
35            i: int = i + 1
36            x: Any = self.search(parent, u)
37            y: Any = self.search(parent, v)
38            if x != y:
39                e: int = e + 1
40                result.append([u, v, w])
```

Activate Windows
Go to Settings to activate Windows.

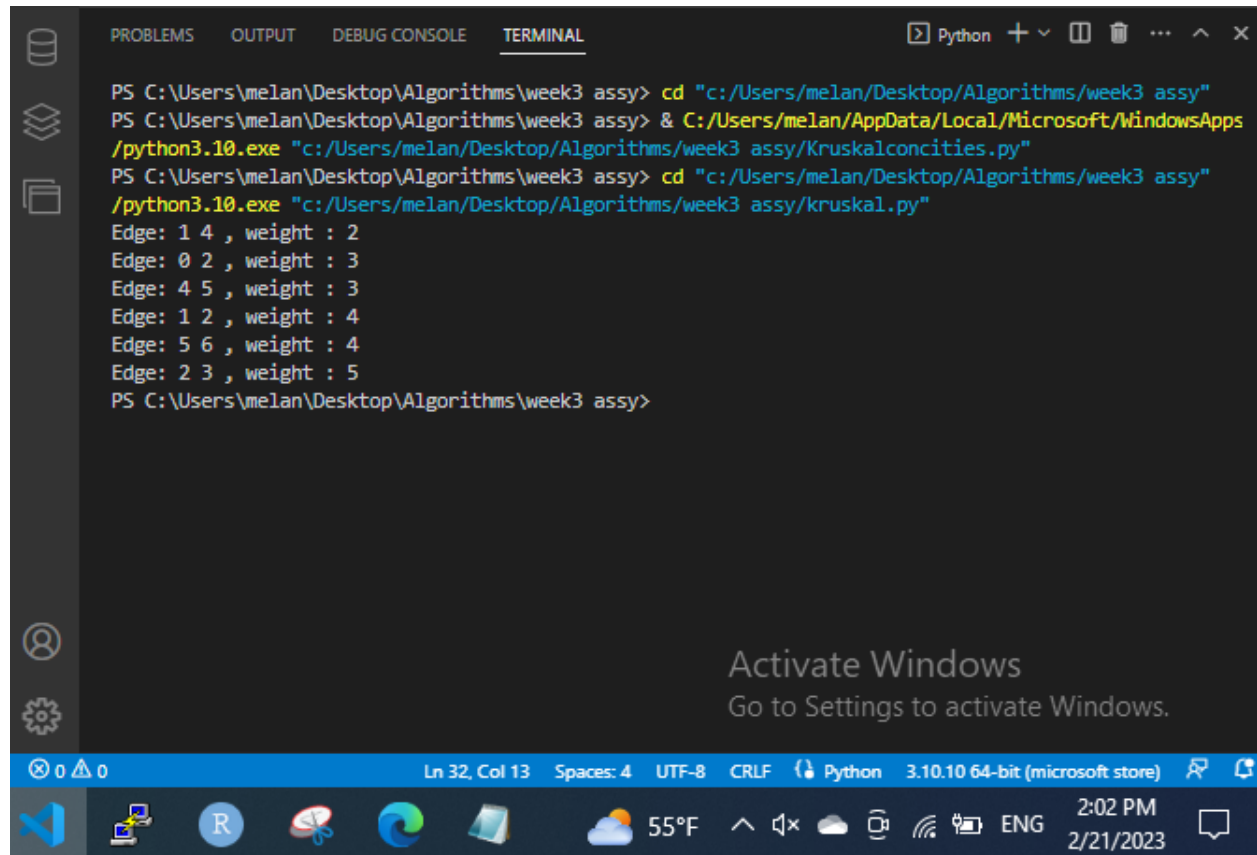
Ln 32, Col 13 Spaces: 4 UTF-8 CRLF Python 3.10.10 64-bit (microsoft store)

2:01 PM 2/21/2023

```
kruskal.py - week3 assy - Visual Studio Code
kruskal.py x primconcities.py maxq
kruskal.py > Graph > kruskal
27 rank: list = []
28
29 for node in range(self.V):
30     parent.append(node)
31     rank.append(0)
32
33 while e < self.V - 1:
34     u: Any, v: Any, w: Any = self.graph[i]
35     i: int = i + 1
36     x: Any = self.search(parent, u)
37     y: Any = self.search(parent, v)
38     if x != y:
39         e: int = e + 1
40         result.append([u, v, w])
41         self.apply_union(parent, rank, x, y)
42
43 for u, v, weight in result:
44     print("Edge:", u, v, end=" ")
45     print(", weight :", weight)
46
47 g: Graph = Graph(7)
48 # where a = 0, b = 1, c = 2, d = 3, e = 4, f = 5, g = 6
49 g.add_edge(0, 1, 5)
50 g.add_edge(0, 2, 3)
51 g.add_edge(1, 2, 4)
52 g.add_edge(1, 3, 6)
53 g.add_edge(1, 4, 2)
54 g.add_edge(2, 3, 5)
55 g.add_edge(2, 5, 6)
56 g.add_edge(3, 4, 6)
57 g.add_edge(3, 5, 6)
58 g.add_edge(4, 5, 3)
59 g.add_edge(4, 6, 5)
60 g.add_edge(5, 6, 4)
61
62 g.kruskal()

Activate Windows
Go to Settings to activate Windows.

Ln 32, Col 13 Spaces: 4 UTF-8 CRLF Python 3.10.10 64-bit (microsoft store)
2:01 PM 2/21/2023
```



The screenshot shows a Windows terminal window with the following content:

```
PS C:\Users\melan\Desktop\Algorithms\week3 assy> cd "c:/Users/melan/Desktop/Algorithms/week3 assy"
PS C:\Users\melan\Desktop\Algorithms\week3 assy> & C:/Users/melan/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/melan/Desktop/Algorithms/week3 assy/Kruskalconcities.py"
PS C:\Users\melan\Desktop\Algorithms\week3 assy> cd "c:/Users/melan/Desktop/Algorithms/week3 assy"
PS C:\Users\melan\Desktop\Algorithms\week3 assy> python3.10.exe "c:/Users/melan/Desktop/Algorithms/week3 assy/kruskal.py"
Edge: 1 4 , weight : 2
Edge: 0 2 , weight : 3
Edge: 4 5 , weight : 3
Edge: 1 2 , weight : 4
Edge: 5 6 , weight : 4
Edge: 2 3 , weight : 5
PS C:\Users\melan\Desktop\Algorithms\week3 assy>
```

The terminal window has a dark theme and a sidebar on the left with icons for Explorer, Search, and Task View. The status bar at the bottom shows the file path, line and column numbers, and the Python version.

CODE

```
class Graph:
    def __init__(self, vertex):
        self.V = vertex
        self.graph = []
    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])
    def search(self, parent, i):
        if parent[i] == i:
            return i
        return self.search(parent, parent[i])
    def apply_union(self, parent, rank, x, y):
        xroot = self.search(parent, x)
        yroot = self.search(parent, y)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
```



```
rank[xroot] += 1
```

```
def kruskal(self):
    result = []
    i, e = 0, 0
    self.graph = sorted(self.graph, key=lambda item: item[2])
    parent = []
    rank = []

    for node in range(self.V):
        parent.append(node)
        rank.append(0)

    while e < self.V - 1:
        u, v, w = self.graph[i]
        i = i + 1
        x = self.search(parent, u)
        y = self.search(parent, v)
        if x != y:
            e = e + 1
            result.append([u, v, w])
            self.apply_union(parent, rank, x, y)

    for u, v, weight in result:
        print("Edge:", u, v, end=" ")
        print(", weight :", weight)
```

```
g = Graph(7)
# where a = 0, b =1, c =2, d = 3, e = 4, f = 5, g = 6
g.add_edge(0, 1, 5)
g.add_edge(0, 2, 3)
g.add_edge(1, 2, 4)
g.add_edge(1, 3, 6)
g.add_edge(1, 4, 2)
g.add_edge(2, 3, 5)
g.add_edge(2, 5, 6)
g.add_edge(3, 4, 6)
g.add_edge(3, 5, 6)
g.add_edge(4, 5, 3)
g.add_edge(4, 6, 5)
g.add_edge(5, 6, 4)

g.kruskal()
```