# CHAPTER 5
# IMPLEMENTATION AND TESTING

## 5.1 Implementation

In this project, the data structure is placed in a separate class. The data structure used will form a dictionary. Data structure class contains 3 methods. There are add, find, and union.

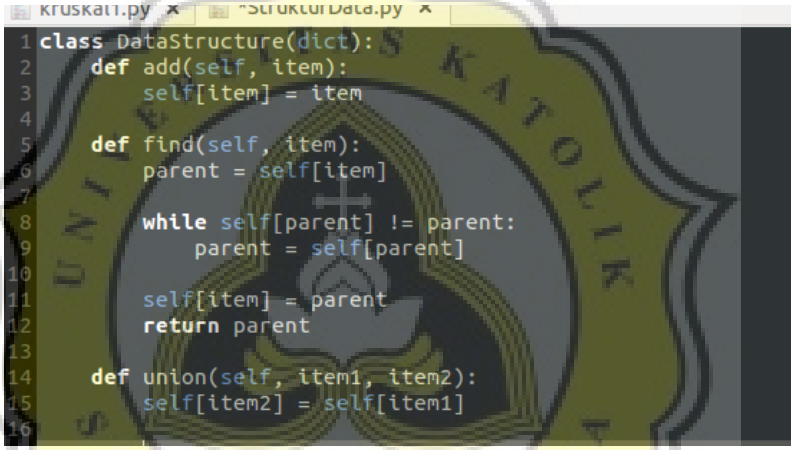
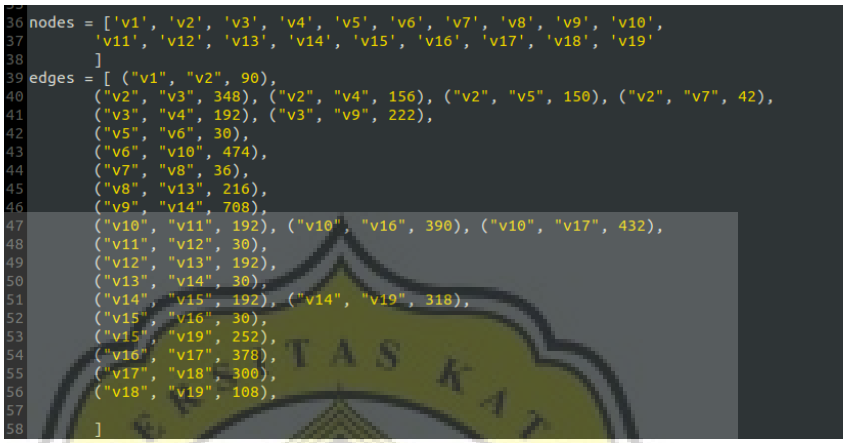



Illustration 5.1: Data Structure Class

Piece of code above comes from the DataStructure class. The **add** method works to add nodes to the dictionary. The **find** method is used to determine the nodes by finding the parent of the nodes. And the **union** method serves to replace the data nodes contained in the dictionary.

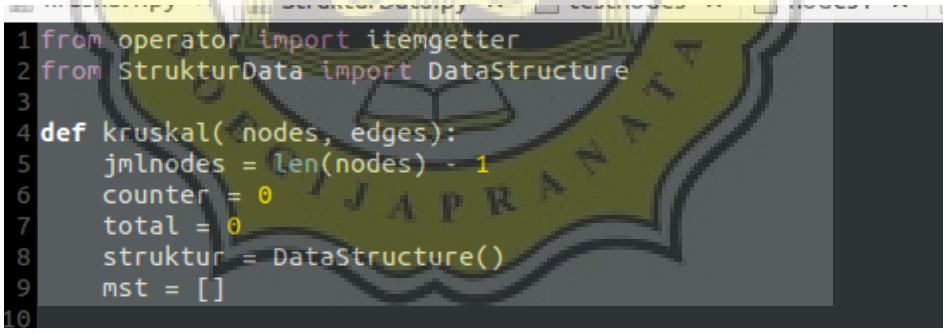


```
nodes = ['v1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8', 'v9', 'v10',
         'v11', 'v12', 'v13', 'v14', 'v15', 'v16', 'v17', 'v18', 'v19'
         ]
edges = [ ("v1", "v2", 90),
         ("v2", "v3", 348), ("v2", "v4", 156), ("v2", "v5", 150), ("v2", "v7", 42),
         ("v3", "v4", 192), ("v3", "v9", 222),
         ("v5", "v6", 30),
         ("v6", "v10", 474),
         ("v7", "v8", 36),
         ("v8", "v13", 216),
         ("v9", "v14", 708),
         ("v10", "v11", 192), ("v10", "v16", 390), ("v10", "v17", 432),
         ("v11", "v12", 30),
         ("v12", "v13", 192),
         ("v13", "v14", 30),
         ("v14", "v15", 192), ("v14", "v19", 318),
         ("v15", "v16", 30),
         ("v15", "v19", 252),
         ("v16", "v17", 378),
         ("v17", "v18", 300),
         ("v18", "v19", 108),

         ]
```

Illustration 5.2: Nodes & Edges Data

Nodes and edges data are stored into txt-extensioned files. This data will be retrieved by the parameters that exist in the kruskal method to be processed.



```
from operator import itemgetter
from StrukturData import DataStructure

def kruskal( nodes, edges):
    jmlnodes = len(nodes) - 1
    counter = 0
    total = 0
    struktur = DataStructure()
    mst = []
```

Illustration 5.3: Initiation of Kruskal Algorithm

The Kruskal calculation begins with the creation of a method that uses two parameters to retrieve data nodes and edges. The **jmlnodes** variable contains the length of nodes - 1. **Counter** and **total** variables in the declare with a value of 0. Next the **DataStructure** class is imported into an object named **struktur**. An empty list named **mst** will be used to store path data that has been selected by Kruskal.

```
17      for e in sorted( edges, key=itemgetter( 2 ) ):
18          n1, n2, x = e
19          t1 = struktur.find(n1)
20          t2 = struktur.find(n2)
21
22          if t1 != t2:
23              mst.append(e)
24              total = total + x
25              counter = counter + 1
26              if counter == jmlnodes:       #proses berhenti
27                  print '======================================='
28                  print 'Total panjang kabel =', total,'centimeter'
29                  print 'Dengan rincian jalur sebagai berikut :'
30                  return mst
```

Illustration 5.4: Process of Kruskal Algorithm

The existing edges data are then sorted in the loop by using the **itemgetter** library. Itemgetter itself is a library in python that serves to retrieve certain values in the list. In this code **itemgetter (2)** means taking the value in the list in the 2nd column to then sorting.

Lines 18-22 of the program code contains commands to check for the existence of circuits in the selection of edges by using the find method found on the **struktur** object. If the selected edges do not form the circuit, then the edges are appended to the **mst** list. The **total** variable serves to add up the edges stored in the mst list. Each program finishes append, then the counter variable increases 1. And when the **counter** count is equal to the variable **jmlnodes** then the minimum spanning tree has been found and the program stops.

## 5.2 Testing

Testing on this project is done on 2 pieces of the plan has been obtained. The plan will be processed to produce the graph that will be examined by Kruskal algorithm. Testing begins with the assumption of the path to be examined by Kruskal.

**SAMPEL I**

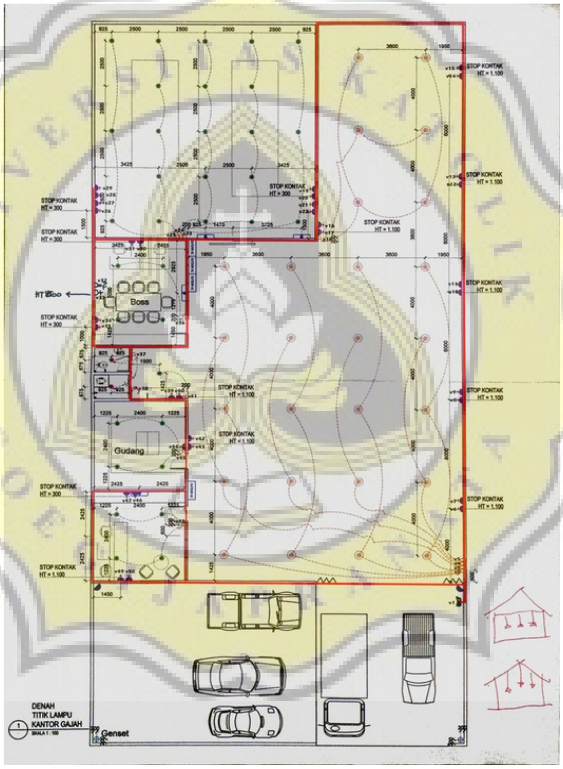



Illustration 5.5: Sample Data 1

The picture above is a 1st floor plan that has been given a path. Further data nodes and edges of the plan are stored into the program. Data nodes and sample edges 1 can be seen in table 5.1

Table 5.1: Data Nodes and Edges Sampel 1

| Nodes & Edges | Weight | Nodes & Edges | Weight |
|---|---|---|---|
| v1 v2 | 150 | v26 v27 | 40 |
| v1 v50 | 1870 | v26 v32 | 410 |
| v2 v3 | 20 | v27 v28 | 40 |
| v3 v4 | 20 | v28 v29 | 40 |
| v4 v5 | 20 | v30 v36 | 480 |
| v5 v6 | 280 | v30 v31 | 40 |
| v6 v7 | 40 | v31 v32 | 450 |
| v7 v8 | 550 | v31 v26 | 350 |
| v8 v9 | 40 | v32 v33 | 40 |
| v9 v10 | 550 | v33 v34 | 150 |
| v10 v11 | 40 | v34 v35 | 40 |
| v11 v12 | 550 | v35 v37 | 340 |
| v12 v13 | 40 | v36 v37 | 520 |
| v13 v14 | 550 | v37 v38 | 180 |
| v14 v15 | 40 | v38 v39 | 370 |
| v15 v19 | 1860 | v39 v40 | 40 |
| v16 v17 | 40 | v40 v41 | 30 |
| v17 v18 | 40 | v41 v42 | 250 |
| v18 v23 | 720 | v42 v43 | 40 |
| v19 v20 | 40 | v43 v44 | 40 |
| v20 v21 | 40 | v44 v45 | 30 |
| v22 v16 | 70 | v45 v46 | 480 |
| v22 v23 | 850 | v45 v48 | 400 |
| v23 v24 | 20 | v46 v47 | 40 |
| v24 v25 | 20 | v47 v49 | 790 |
| v25 v30 | 200 | v49 v50 | 40 |
| v25 v26 | 350 | v50 v48 | 600 |

The above data is then processed by the program. The program will read the stored data to produce the graph. Based on the graph that has been formed, the program gets the result of cable length 10.470 cm. Similar results were obtained when Kruskal calculations were performed manually without program assistance, which was 10,470 cm long. The details of the path chosen by the program can be seen in Illustration 5.6.

Illustration 5.6: Result of Kruskal Algorithm of Sample 1

Based on the results of the path that has been obtained, then the visualization of the path can be done. Visualization on this project is still done manually. The result of the visualized path into the sample plan 1 can be seen in Illustration 5.7.

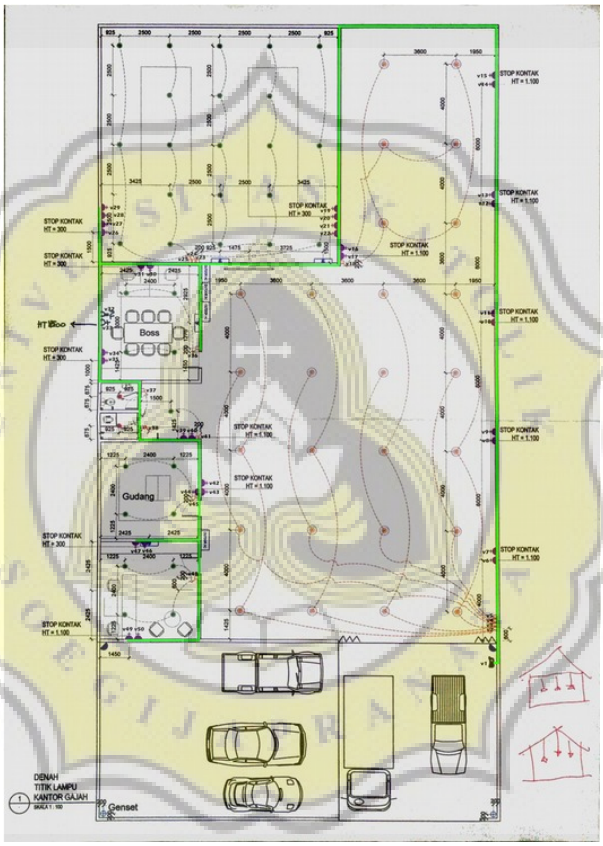



Illustration 5.7: Visualization of Sample Results 1

**SAMPEL 2**

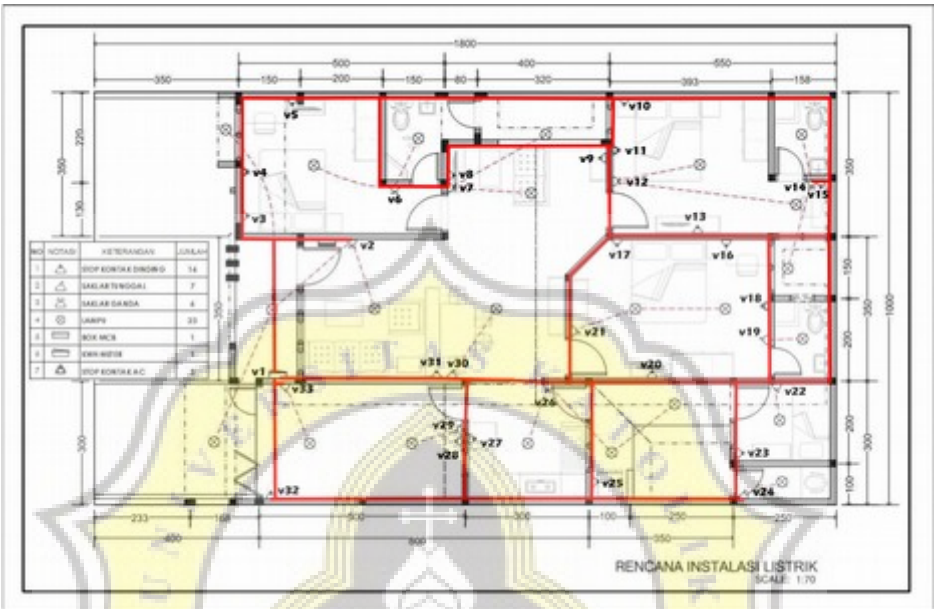



Illustration 5.8: Sample Data 2

The picture above is a sample 2 plan that has been given a path. Further data nodes and edges of the plan are stored into the program. Data nodes and sample 2 edges can be seen in table 5.2

Table 5.2: Data Nodes and Samples Edges 2

| **Nodes & Edges** | **Weight** | **Nodes & Edges** | **Weight** |
|---|---|---|---|
| v1 v2 | 385 | v16 v17 | 203 |
| v1 v3 | 343 | v16 v18 | 196 |
| v1 v31 | 280 | v17 v21 | 210 |
| v1 v32 | 224 | v18 v19 | 98 |
| v1 v33 | 21 | v19 v20 | 280 |
| v3 v4 | 84 | v19 v22 | 98 |
| v4 v5 | 238 | v20 v26 | 161 |
| v5 v6 | 322 | v20 v22 | 231 |
| v5 v9 | 686 | v21 v26 | 175 |
| v6 v7 | 105 | v21 v20 | 231 |
| v7 v8 | 21 | v22 v23 | 203 |
| v8 v9 | 371 | v23 v24 | 84 |
| v9 v10 | 140 | v24 v25 | 308 |
| v9 v11 | 21 | v25 v26 | 259 |
| v9 v12 | 42 | v25 v27 | 357 |
| v10 v11 | 112 | v26 v27 | 238 |
| v10 v15 | 532 | v26 v30 | 175 |
| v11 v12 | 56 | v27 v28 | 21 |
| v12 v13 | 245 | v27 v29 | 35 |
| v12 v17 | 105 | v28 v29 | 21 |
| v13 v14 | 371 | v28 v32 | 455 |
| v13 v16 | 56 | v29 v30 | 98 |
| v13 v17 | 147 | v30 v31 | 28 |
| v14 v15 | 21 | v31 v33 | 287 |
| v15 v16 | 252 | v32 v33 | 455 |
| v15 v22 | 462 | | |

The program will read the stored data to produce the graph. Based on the graph that has been formed, the program gets the result of cable length 4,606 cm. The same results were obtained when Kruskal calculations were performed manually without program assistance, which was 4,606 cm long. The details of the path chosen by the program can be seen in Illustration 5.7.

```
=======================================
Total panjang kabel = 4606 centimeter
Dengan rincian jalur sebagai berikut :
[('v1', 'v33', 21), ('v7', 'v8', 21), ('v9', 'v11', 21), ('v14', 'v15', 21), ('v27', 'v28',
 21), ('v28', 'v29', 21), ('v30', 'v31', 28), ('v9', 'v12', 42), ('v13', 'v16', 56), ('v3',
 'v4', 84), ('v23', 'v24', 84), ('v18', 'v19', 98), ('v19', 'v22', 98), ('v29', 'v30', 98),
 ('v6', 'v7', 105), ('v12', 'v17', 105), ('v10', 'v11', 112), ('v13', 'v17', 147), ('v20',
'v26', 161), ('v21', 'v26', 175), ('v26', 'v30', 175), ('v16', 'v18', 196), ('v22', 'v23',
203), ('v17', 'v21', 210), ('v1', 'v32', 224), ('v4', 'v5', 238), ('v15', 'v16', 252), ('v2
5', 'v26', 259), ('v1', 'v31', 280), ('v5', 'v6', 322), ('v1', 'v3', 343), ('v1', 'v2', 385
)]
```

Illustration 5.9: Result of Kruskal Algorithm of Sample 2

Based on the results of the path that has been obtained, then the visualization of the path can be done. Visualization on this project is still done manually. The results of the visualized path into the sample 2 plan can be seen in Illustration 5.10.

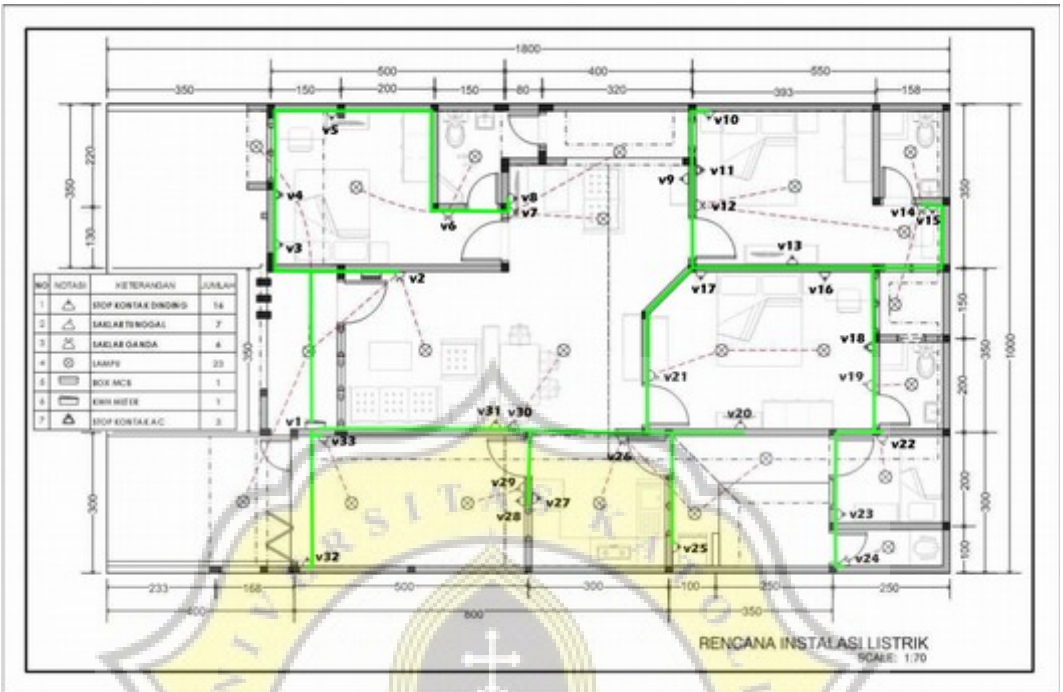



Illustration 5.10: Visualizing Sample Results 2

**SAMPEL 3**

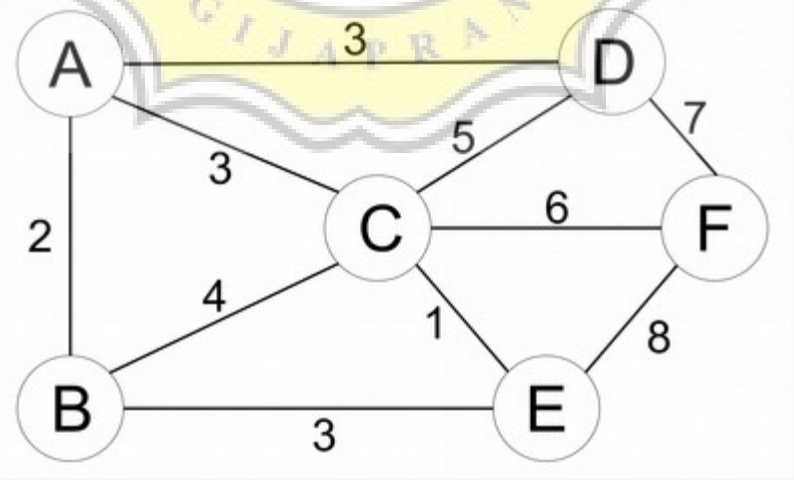



Illustration 5.11: Sample Data 3

This test is performed on a simple graph. The purpose of this test is to know how the behavior of the Kruskal algorithm when taking the side with the same weight.

```
36
37 nodes = ['A', 'B', 'C', 'D', 'E', 'F']
38 edges = [ ("A", "B", 2), ("A", "D", 3), ("A", "C", 3),
39          ("B", "C", 4), ("B","E", 3),
40          ("C", "D", 5), ("C", "F", 6), ("C", "E", 1),
41          ("D", "F", 7),
42          ("E", "F", 8),
43        ]
```

Illustration 5.12: Data Nodes and Samples Edges 3

Based on the result of sample 3 test, Kruskal's behavior was taken in taking the side with the same weight. Kruskal will take sides according to the stored sequence. The result of sample 3 test can be seen in Illustration 5.12.

```
==============================================
Total panjang kabel = 15 centimeter /
Dengan rincian jalur sebagai berikut :
[('C', 'E', 1), ('A', 'B', 2), ('A', 'D', 3), ('A', 'C', 3), ('C', 'F', 6)]
```

Illustration 5.13: Sampling Test Results 3