IJID International Journal on Informatics for Development, *e-ISSN :2549-7448*
Vol. 3, No. 2, 2014

Minimum Spanning Tree for the Implementation of Kruskal's Algorithm

Paryati
Department of Informatics Engineering
Faculty of Industrial Engineering
Universtas Pembangunan Nasional "Veteran" Yogyakarta
Jl.Babarsari No.2 TambakBayan Yogyakarta 55281. Telp (0274) 485323
yaya_upn_cute@yahoo.com

Ahmad Subhan Yazid
Informatics Department
Islamic State University (UIN) of Sunan Kalijaga Yogyakarta, Indonesia
yazid.anfalah@gmail.com

***Abstract*—Kruskal's algorithm is an algorithm used to find a minimum spanning tree in graph connectivity which gives an option to keep processing the edge limit with the least weight. In a Kruskal's algorithm, the weight sorting of edge value will ease the finding of the shortest path. This algorithm has independent nature which will ease and enhance the path track making.**

Keywords-Algorithm; Graph; Kruskal's; Spanning Tree.

I. INTRODUCTION

The theories about graphs began in 1736 and several important discoveries in graph theory were obtained in the 19th century, but not until the 1920s that the interest in graph theory bloomed. The first text about graph theory emerged in 1936. Undoubtedly, one of the reasons for interest in graph theory is due to its application in many fields, including computer science, chemistry, operation research, electrical engineering, linguistics, and economy [1].

The research will address the definition of a graph and some terminologies along with a basic example of a graph. The shortest path algorithm which will be used is Kruskal's algorithm. It will show us how to find the shortest path between two given points.

Fig 1. shows a highway system that must be supervised by the investigator officer. Particularly, this investigator must make trips to all roads and make reports about their condition, the clarity of roads path, the condition of traffic signs, and so on. Because she lives in Semarang, the most economical way of examining all roads must be started in Semarang.

Figure 1. The Highway System that must Be Supervised

II. LITERATURE REVIEW

A. Graph and Digraph

A graph (*G*) is a set of nodes/ vertices/ points (*V*) joined by a set of lines or arrows [2]. In a directed graph, arrows join nodes. On the other hand, in an undirected graph, undirected lines join nodes. Both of the arrows and undirected lines are called edges (*E*).

In undirected graph, every edge is a member of set E, $e \in E$ and is associated with the unordered vertex pair. If there is an *e* edge that connects vertices *v* and *w*, it is written as $e = (v, w)$ or $e = (w, v)$. In this context, (v, w) suggests an edge between *v* and *w* is in an undirected graph and not an ordered pair.

A directed graph or digraph G consists of a set V from vertices (nodes or points) and an *E* set from edge as such that every edge of $e \in E$ connects the ordered vertex pair. If there is a single *e* edge that connects the ordered pair (v, w) from vertices, it is written as $e = (v, w)$, which suggests an edge from *v* to *w*.

An *e* edge in a graph (directed or undirected) which connects the vertices pair *v* and *w* is said to be incidental on *v* and *w*, then *v* and *w* are said to be incidental on *e* and referred to as adjacent vertices.If *G* is a graph (directed or undirected) with vertices *v* and edges *E*, then it is written as $G = (V, E)$. If it is not specifically mentioned, the set *E* and *V* are assumed as finite and *V* is assumed as not empty.

In the case in Fig. 1 above, the model making it will appear as shown in Fig. 2.

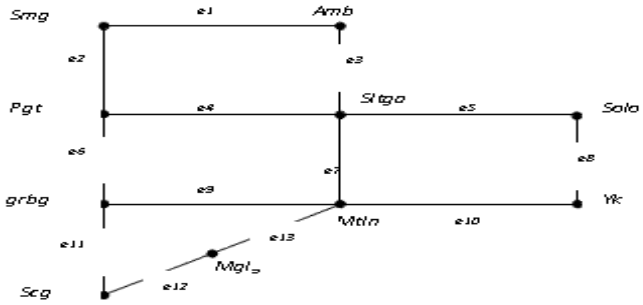

Figure 2. Gaph Model Form a Highway System

The points in figure 1 are called vertices and the lines which connect those vertices are called edge. Every vertex is named by taking the first three letters of the appropriate city. The edges are signed by $e_1, \ldots, e_{13}$. In drawing a graph, the only important information is the imaginary vertices linked by the imaginary edges as well. If we start from a vertex v_0, walk along a edge to vertex v_1, walk along another edge to vertex v_2 and so on, and finally arrive at vertex v_n, we refer to such complete trip as path from v_0 to v_n.

B. Spanning Tree

A *T* tree is a Spanning Tree of graph *G* if *T* is a subgraph of *G* which contains all vertices of *G*. The characteristics of Spanning Tree are:

1. *All graph G links in this tree must not form a circle.*
2. *A graph G is said to be a tree if and only if G is linked or form a path.*
3. *A linked graph G is said to be a tree if and only if it is every edge forms a edge.*
4. *A linked graph G is said to be a tree if and only if it has an N peak and N-1 edge.*

The description of spanning tree is shown in Fig. 3.

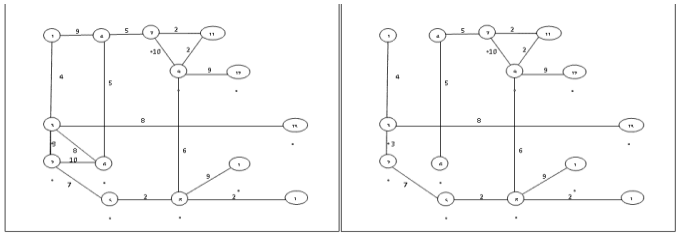

Figure 3. Tree

From Fig. 3, it can be stated that figure (A) is a wrong figure because it is not fit the characteristics of a tree as mentioned earlier. Figure (B) is a correct tree because it fulfills the characteristics of a tree.

C. *Kruskal's Algorithm*

Kruskal's algorithm is one to find a minimum spanning tree in graph connectivity which gives options of always processing the edge limit with the least weight [2]. This algorithm is run by considering the graph's biggest edge limit when searching for track in node in a graph that has been put into a spanning tree. If an edge limit is considered will integrate (with one of the points not in the spanning tree), or the integration of a point in spanning tree (one of which is not in the spanning tree), then the edge limit and the final point is included in the spanning tree. Considering one of the edge limits, algorithm continues by considering the next larger edge limit weight. In this term, when the edge limit weight values are the same, then it is unnecessary to determine which weight must be chosen first. The algorithm will stop when every point has been included in the spanning tree. Note that, when spanning tree was made, there was a possibility that the edge limit is not connected with the part of the main tree, although in the end, they will integrate with the main tree.

III. METHODS

The first research conducted used the testing method on Kruskal's algorithm step by step. The algorithms are as follows:

$E_{(1)}$ is a set of edges from Minimum Genetic Tree

$E_{(2)}$ is a set of the remaining edges

V is a set of vertices

n is node or point

Steps:

$E_{(1)}=0$, $E_{(2)}=E$

WHILE $E_{(1)}$ consists less than $n-1$ edges and $E_{(2)}=0$ DO

From the side of E(2) choose one with the lowest value→ e(ij)

E(1)=E(2)-{ e(ij)}

IF V(i), V(j) is not included in the same tree, THEN

Unite tree from V(i) and V(j) become one single tree

END of IF

END of WHILE

END of Algorithm

IV. IMPLEMENTATION

Here is the source code for implementing the algorithm above which is obtained from [3]:

```
public class Algorithms
{
   public static Graph KruskalsAlgorithm (Graph g)
   {
     int n = g.getNumberOfVertices();
     Graph result = new GraphAsLists (n);
     for (int v = 0; v < n ; ++v )
        result.addVertex (v);
     PriorityQueue queue =
        new BinaryHeap (g.getNumberOfEdges());
     Enumeration p= g.getEdges();
     while (p.hasMoreElements())
     {
        Edge edge = (Edge) p.nextElement();
        Int weight= (Int) edge.getWeight();
        queue.enqueue (new Assosiation (weight, edge));
     }
     Partition partition = new PartitionAsForest(n);
     while(!queue.isEmpty() && partition.getCount() > 1)
     {
        Association assoc = (Association)
        queue.dequeueMin();
        Edge edge = (Edge) assoc.getValue ();
        int n0 = edge.getV0 ().getNumber ();
        int n1 = edge.getV1 ().getNumber ();
        Set s = partition.find (n0);
        Set s = partition.find (n1);
        If (s !=t)
        {
                partition.join (s,t);
                result.addEdge (n0,n1);
        }
     }
        return result;
   }
}
```

V. EXPERIMENT AND PRODUCT ANALYSIS

In conducting analysis for Kruskal's algorithm a test is employed by having implementation overall two algorithms with the case example as follows:

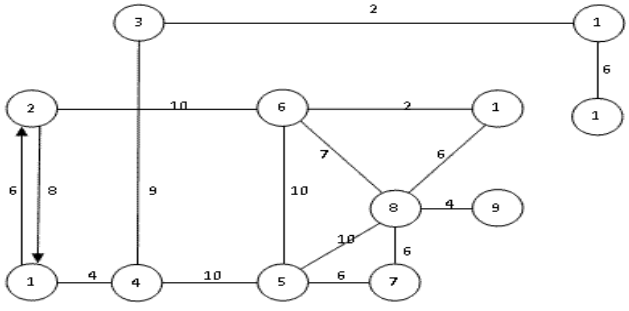

Figure 4. Tree Representation

Initial step: if we look at the characteristics of a tree, then the above graph does not fulfill the characteristics of a graph,

because the track is circular. Thus, we will include Kruskal's algorithm to solve the problem above. The first step to run Kruskal's algorithm is arranging *e* (edge weight) in the right order from the minimum value to the maximum. After the arrangement (Exhibit 5.1) then we make the track for both vertices with minimum weight.

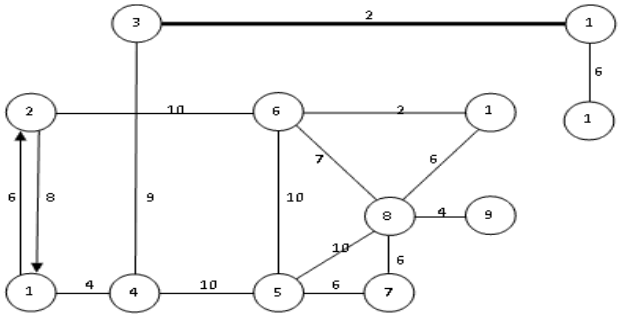

Figure 5. Initial Step

Step 1: in the line with the table order, then what to do first is making track $V_3 - V_{11}$ with the edge weight 2 (track with bold lines).

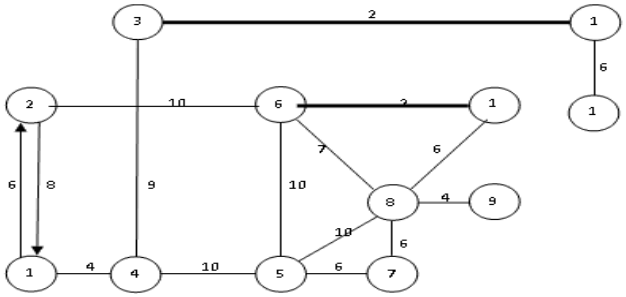

Figure 6. Step 1

Step 2: the next order is $V_6 - V_{10}$ with the edge weight 2. It appears that the track is separated from step 1. However, considering the characteristics of the independent Kruskal's algorithm which means that there is a possible edge limit (vertex) unconnected with the part of the main tree, although eventually will join the main tree, it can happen.

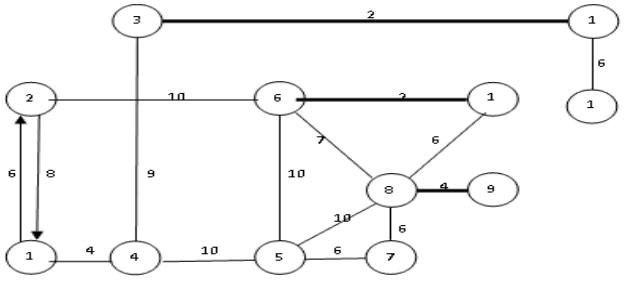

Figure 7. Step 2

Step 3: the next order is $V_8 - V_9$ with the edge weight 4.

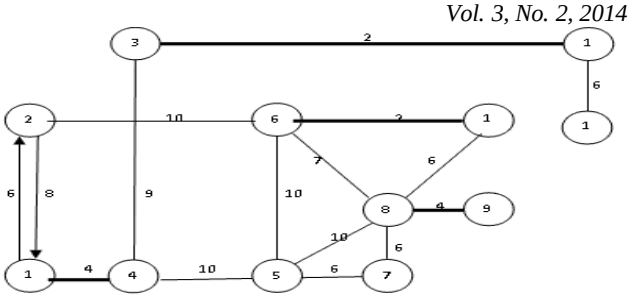

Figure 8. Step 3

Step 4: the next order is $V_1 - V_4$ with the edge weight 4.

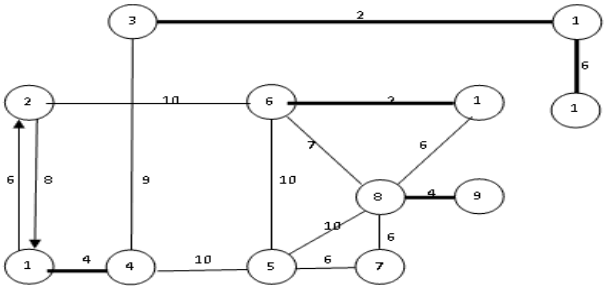

Figure 9. Step 4

Step 5: the next order is $V_{11} - V_{12}$ with the edge weight 6.

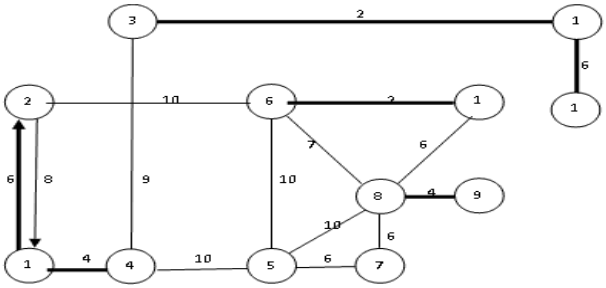

Figure 10. Step 5

Step 6: the next order is $V_1 - V_2$ with the edge weight 6.

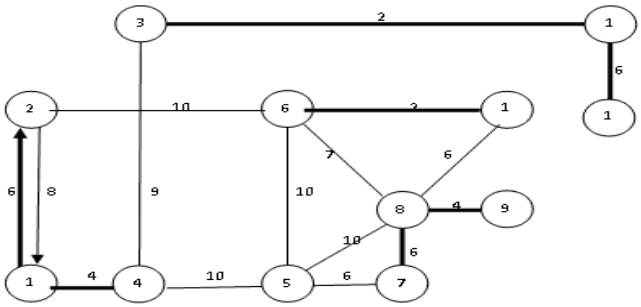

Figure 11. Step 6

Step 7: the next order is $V_8 - V_7$ with the edge weight 6.

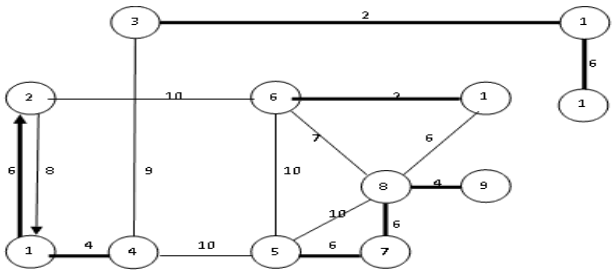

Figure 12. Step 7

Step 8: the next order is $V_5 - V_7$ with the edge weight 6.

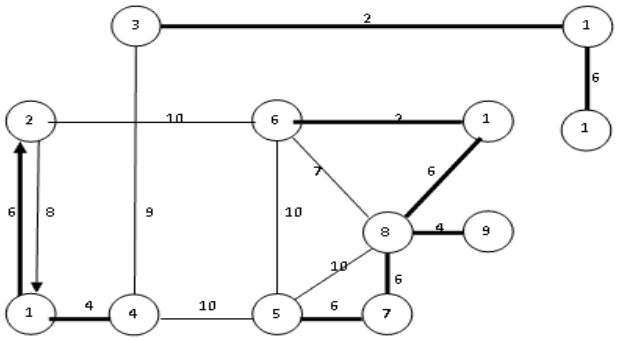

Figure 13. Step 8

Step 9: the next order is $V_8 - V_{10}$ with the edge weight 6.

Step 10: the next order is $V_6 - V_8$ with the edge weight 7. This path is not allowed because it will form a circle in path $V_6 - V_8 - V_{10}$.

Step 11: the next order is $V_2 - V_1$ with the edge weight 8. This path is not allowed because vertex $V_2 - V_1$ has been passed. In fact, the requirement of a tree only allows it to be passed once.

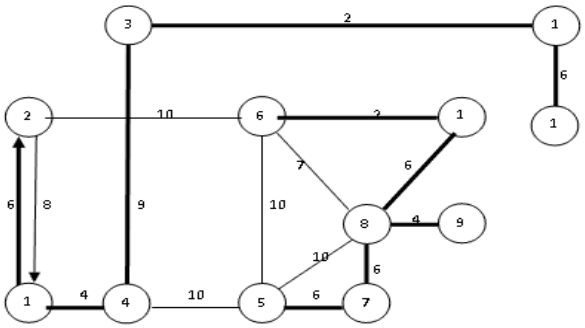

Figure 14. Step 11

Step 12: the next order is $V_3 - V_4$ with the edge weight 9.

Step 13: the next order is $V_2 - V_6$ with the edge weight 10.

Step 14-16: this order is unnecessary to be done because it forms a circle.

TABLE I. STEP OF PATH DETERMINATION IN KRUSKAL'S ALGORITHM

Step	v	e	
1	3-11	2	Add to tree
2	6-10	2	Add to tree
3	8-9	4	Add to tree
4	1-4	4	Add to tree
5	11-12	6	Add to tree
6	1-2	6	Add to tree
7	8-7	6	Add to tree
8	5-7	6	Add to tree
9	8-10	6	Add to tree
10	6-8	7	Reject because it forms a circuit
11	2-1	8	Reject because it forms is turned back
12	3-4	9	Add to tree
13	2-6	10	Add to tree
14	6-5	10	Reject because it forms a circuit
15	4-5	10	Reject because it forms a circuit
16	5-8	10	Reject because it forms a circuit

Total weight ----------- 61

Other examples of Kruskal's Algorithm are as follows:

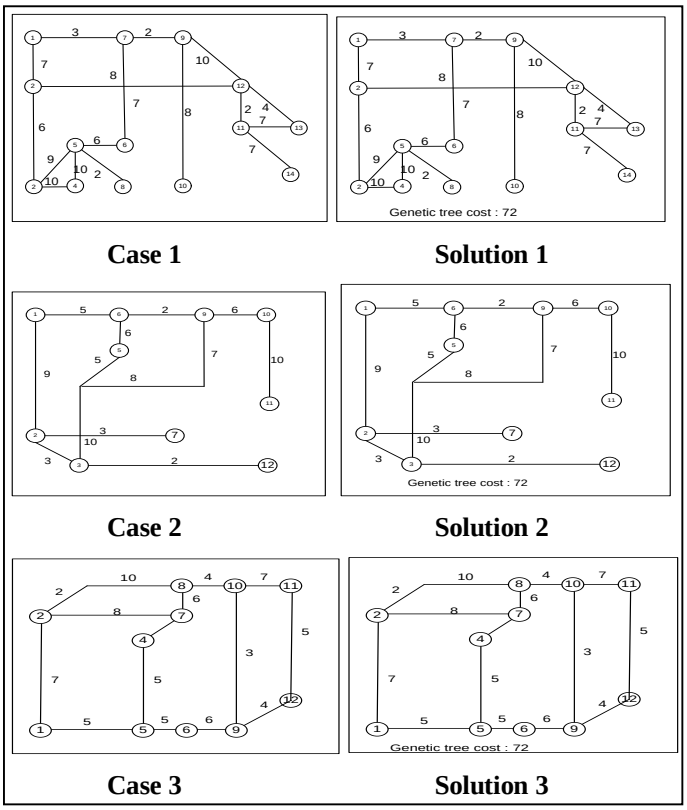

Figure 15. Other Examples

VI. CONCLUSION

After testing and comparison of the crucial algorithm, then the conclusion can be drawn as follows:

A. Strength

1) The existence of weight sorting will ease the searching for the shortest path.

2) Considering the characteristics of Kruskal's independent algorithm, it will ease and enhance the formation of path track.

B. Weaknesses

If the number of vertices is very large, it will slower than the Dijkstra's algorithm because it must sort thousands of vertices first, then forming the path.

REFERENCES

[1] R. Johnsonbaugh, Discrete Mathematics, Seventh. Pearson Prentice Hall, 2009.

[2] G. Brassard and P. Bratley, Fundamental of Algorithmics. Pearson, 1995.

[3] B. R. Preiss, Data Structures and Algorithms with Object-Oriented Design Patterns in Java. John Wiley & Sons, 1999.