Greedy Algorithm for Electricity Distribution Solution in Indonesia

Yogi Salomo Mangontang Pratama 13511059
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
yogi.sinaga@students.itb.ac.id

***Abstract*—Indonesia has a wide area and also a lot of Resources that can be used to produce electricity power to serve its areas. But up until now the distribution of electricity in Indonesia is still unable to be done evenly. Not every region in Indonesia is served with electricity and the distribution is too focused in the big cities. This paper will explain how computer programming can help to solve this country's problem. By using greedy algorithm, especially Minimum Spanning Tree Problem, the program explained in this paper will calculate the minimum cost of electricity distribution in Indonesia and the path of the distribution itself. The will prove that Greedy Algorithm is efficient enough to solve this problem and solution of this problem is able to be found with Computer Programming.**

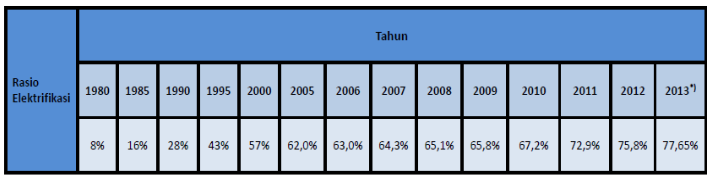
***Index Terms*—Power Plant, Minimum Spanning Tree, Prim's, Kruskal's**

I. Introduction

Indonesia is the biggest archipelago on Earth. With approximately 17,000 islands scattered from Sabang (West most) to Merauke (East most), Indonesia covered for about 5,000 km area of the surface, making it one of the broadest nation in the world.

Its broad territory has given Indonesia a lot of benefits. Indonesia has a rich variety of natural resources. With 2,957 animal species and 8,000 plant species discovered, and about 80% species still undiscovered, majority of the wildlife around the world exist in Indonesia. This country also has a wide variety of human culture. With about 1,340 ethnic group lives across its area, Indonesia has one of the biggest number of tradition and culture and it makes Indonesia more and more interesting to live in and study about.

But beside of those benefits given by its broad and archipelago area, Indonesia must also face some problems that caused by the very same reason of those benefits. One of the problems that become the main concern on this paper is the distribution of electricity in every area across all islands within its territory. Because as we know, being an archipelago country means that Indonesia has a broad area of islands and also a lot of water terrain within it. These factors made an even electricity distribution become a challenging problem for the government to solve.



Rasio Elektrifikasi	Tahun													
	1980	1985	1990	1995	2000	2005	2006	2007	2008	2009	2010	2011	2012	2013*)
	8%	16%	28%	43%	57%	62,0%	63,0%	64,3%	65,1%	65,8%	67,2%	72,9%	75,8%	77,65%

Table 1.1 The Electrification Table of Indonesia(1980-2013)

As described on the table above, the distribution of electricity hasn't been reaching every area in Indonesia completely (75.8% in 2012). And though the percentage is quite high, the distribution is not even yet since the highest Electrification is in the province of the capital city, DKI Jakarta, with 99.99% while the lowest one is in the Papua region with only 35.89% of its area already served with electricity. These statistics indicates that the government is still unable to determine the most effective method to distribute the electricity evenly.

The electricity source itself shouldn't be any problem since Indonesia with its wide variety of natural resources that can be used to produce electric power. There are about 45 power plants that is powered by water (PLTA), air (PLTU), natural gas (PLTG), geothermal(PLTB), Solar, Nuclear Energy(PLTN), Coal (PLTB), Renewable Energy (or also called Bio Mass) spread across Indonesia. The main problem now is to determine which power plant should serve certain area so that every area in Indonesia could receive electricity service and the total cost of distribution is as low as possible.

This paper will explain how programming can solve this problem. In this context, Indonesia can be represented as a graph where the nodes will represent the power plants, cities, villages, and all other important sites of the country, and the edges of the graph represents the cables and other infrastructures that connect one area to another. The cable and infrastructures itself must have cost to build and maintain, so the cost itself will be the weight of the edges in the graph.

Because the main concern is to make sure every area in the country is served with electricity, every node in the graph must be connected with at least one node that represents a power plant except the nodes that represent the power plant itself. A site is considered connected to a power plant if and only if there is a path which consists of only cables from site to a power plant.

Since every city, village and other sites only need to be connected with one power plant, the Minimum Spanning Tree Problem is most accurate model to this problem. The nature of Minimum Spanning Tree Problem is to select a subset of Edges in a graph such that the graph is still connected and the total weight of the selected Edges is minimal. This is exactly what the problem of electricity distribution needed. Minimum Spanning Tree Problems can be solved using several well-known Greedy Algorithms like Prim's and Kruskal's which are going to be explained in this paper.

II. SOME THEORIES

II.I GREEDY ALGORITHM

The word Greedy itself is an adjective which means "desiring excessively". The principle of Greedy is to form the solution step by step. In every step, there exists a lot of options that need to be explored. So in every step a best decision must be made to determine the best option. In every step a **local optimum solution** must be formed. By assuming that the rest of the step will lead to **global optimum solution.**

The elements of Greedy Algorithm are:

1. Candidate set (C)
   The set of possible values that will be used as the value to determine the optimum solution.
2. Solution set (S)
   The combination set of candidates that fulfill the requirement of current problem.
3. Selection Function
   The function used to determine the most optimum solution to be picked from the rest of candidate value.
4. Feasibility Function
   The function used to check whether current solution to be picked still fulfilling the requirement of current problem.
5. Objective Function
   The function used to make sure that the solution to be picked is the most optimum.

The generic scheme of Greedy Algorithm is explained in the following pseudo-code:

```
function greedy(input C:
candidate_set)? Candidate_set
{ Return the solution of Optimum
Decision of Greedy Algorithm
  Input: candidate_set C
  Output: soltion_set with relevant
type with candidate_set
}
Declaration
    x : candidate
    S : candidate_set

Algorithm:
      S <- {}    { Initialization of S
with  empty }
      while (not SOLUTION(S)) and (C
!= {} ) do
       x <- SELECTION(C)           {
Choose a candidate from C}
       C <- C - {x}           { Reduce
the candidate set by one element }
      if FEASIBLE(S AND {x}) then
                S <- S AND {x}
          endif
     endwhile
{SOLUTION(S) or C = {} }

if SOLUSI(S) then
   return S
else
   write('No Solution')
endif
```

Global Optimum Solution is not always the most optimum solution. It is only sub-optimum or pseudo-optimum. There are two reasons to support this statement:

1. Greedy Algorithm is not operating in every possible solution alternatives that exist.
   There is a possibility where a solution that is not the most optimum in a step will produce the global solution in the following steps.
2. There consists different Selection Functions, so that we need to choose the right function to produce optimum solution.

If the absolute best solution is not required, then Greedy Algorithm is often useful to produce an approximation solution because Greedy Algorithm is quite simple compared than another algorithms. Greedy Algorithm is also useful when local optimum solution can be reassured as part of global optimum solution.

II.II MINIMUM SPANNING TREE

A spanning tree of a graph is a sub-graph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A graph can also have a *weight* at each edge, which is a number representing how unfavorable it is. Those weights are used to compute the total cost of a spanning tree by computing the sum of the weights of the edges in that spanning tree.

A **Minimum Spanning Tree** (**MST**) or **Minimum Weight Spanning tree** is a spanning tree with cost less than or equal to the cost of every other spanning tree. More generally, any undirected graph (not necessarily connected) has a **Minimum Spanning Forest**, which is a union of minimum spanning trees for its connected components.

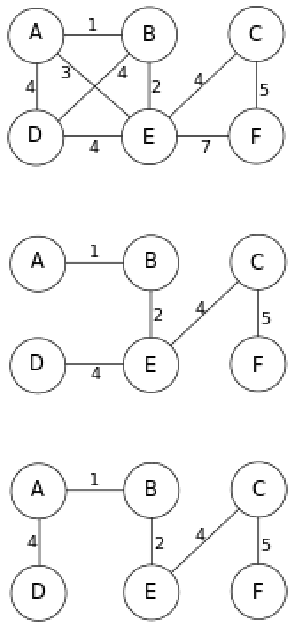


Image 2.1 The Example Of Minimum Spanning Tree of a Graph. Source: www.wikipedia.org

Minimum Spanning Tree (MST) has many practical applications. For example, we can model a problem of building road networks in remote villages as an MST Problem. The Vertices are the villages. The edges are the potential roads that may be built between those villages. The cost of building a road that connects village *i* and *j* is the weight of edge(*i*, *j*).

This MST problem can be solved with several well-known algorithms i.e. Prim's and Kruskal's, both are going to be explained and used as experiment in this paper.

II.IV PRIM'S ALGORITHM

Robert Clay Prim's Algorithm fist takes a starting vertex, flags it as 'taken', and en queue a pair of information into a priority queue. The weight(w) and the other end point (u) of the chosen edge that is not taken yet. Those pairs are sorted in the priority queue based on the increasing weight, and if more than one pair has similar weight, sorted by increasing vertex number. Then Prim's Algorithm *greedily* select the pair in front of the priority queue – which has the minimum weight w – if the end point of the edge has not been taken before. This validation must be done to prevent a cycle. If the pair is valid, then the weight w is added into the MST cost, and marked the selected vertices as 'taken'. This process is repeated until the priority queue is empty.

The implementation of Prim's Algorithm in a pseudo-code is explained in the following section.

```
Prim(G, w, s)
{Input: undirected connected weighted
graph G = (V,E) in adj list
representation,
source vertex s in V
Output: p[1..|V|], representing the
set of edges composing an MST of G}

      for each v in V
            color(v) <- WHITE
            key(v) <- infinity
            p(v) <- NIL
      Q <- empty list // Q keyed by
key[v]
      color(s) <- GRAY
      Insert(Q, s)
      key(s) <- 0
      while Q != empty
            u <- Extract-Min(Q)
            for v in Adj[u]
                  if color(v) = WHITE
                        then color(v)
<- GRAY
                        Insert(Q,v)
                        key(v) <-
w(u,v)
                        p(v) <- u
                  elseif color(v) =
GRAY
                        then if key(v)
> w(u,v)

      then key(v) <- w(u,v)

      p(v) <- u
            color(v) <- BLACK
      return(p)
```

Prim's Algorithm has one key problem that need to be solved, which is : How to Efficiently Find The Crossing Edge of Minimal Weight ?

There are two implementations that can be used to solve the question:

1. Lazy implementation
   Use a priority queue to hold the crossing edges and find one of minimal weight. Each time an edge is added to the tree, also add a vertex to the tree. To maintain the set of crossing edges, add all edges from that vertex to any non-tree vertex to the priority queue. Any edge connecting the vertex just added to a tree vertex that is already on the priority queue now becomes *ineligible* (it is no longer a crossing edge because it connects two tree vertices). The lazy implementation leaves such edges on the priority queue, deferring the ineligibility test to when they are removed.

2. Eager Implementation
   To improve the lazy implementation of Prim's algorithm, delete ineligible edges from the priority queue, so that the priority queue contains only the crossing edges. The key is to note that the only interest is in the *minimal* edge from each non-tree vertex to a tree vertex. When adding a vertex v to the tree, the only possible change with respect to each non-tree vertex w is that adding v brings w

closer than before to the tree. In short, keeping on the priority queue all of the edges from w to vertices tree—we just need to keep track of the minimum-weight edge and checking whether the addition of v to the tree necessitates that is updated that minimum (because of an edge v-w that has lower weight), which can be done while processing each edge in s adjacency list. In other words, the maintaining on the priority queue just one edge for each non-tree vertex: the shortest edge that connects it to the tree.

The time complexity is O(V log V + E log V) = O(E log V), making it the same as Kruskal's algorithm. The following image gives visual explanation about how Prim's Algorithm works to make a Minimum Spanning Tree of a Graph.

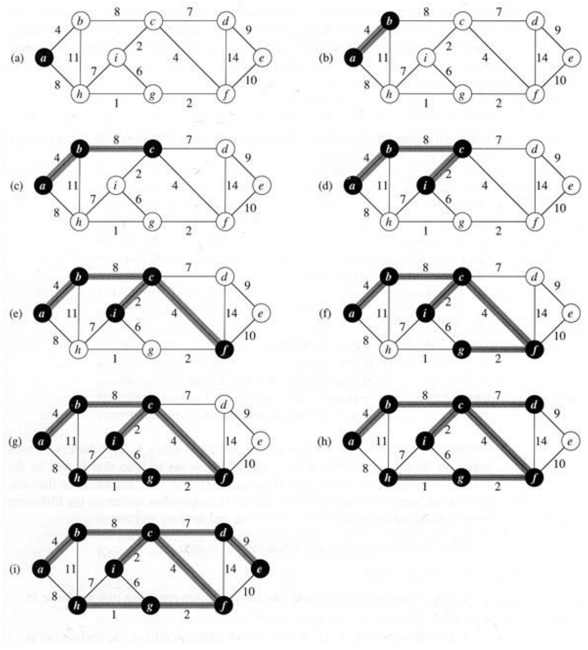


Image 2.2 The Animation of Prim's Algorithm for an MST Problem. Source: vis.berkeley.edu

II.IV KRUSKAL'S ALGORITHM

Joseph Bernard Kruskal Jr.'s algorithm first sort *E* edges based on non-decreasing weight in *O*(*E* log *E*). This can be easily done using priority queue (or alternatively, use vector & sort). Then, it *greedily* tries to add *O*(*E*) edges with minimum costs to the solution as long as such addition does not form a cycle. This cycle check can be done easily using Union-Find Disjoint Sets.

The implementation of Kruskal's Algorithm in a pseudo-code is explained below.

```
KRUSKAL(G):
 A = Ø
 foreach v MemberOf G.V:
   MAKE-SET(v)
 foreach (u, v) ordered by weight(u, v), increasing:
     if FIND-SET(u) ? FIND-SET(v):
        A = A JOIN {(u, v)}
        UNION(u, v)
  return A
```

Given *E* as the number of edges in the graph and *V* is the number of vertices, Kruskal's algorithm can be shown to run in *O*(*E* log *E*) time, or equivalently, *O*(*E* log *V*) time, all with simple data structures. These running times are equivalent because:

- *E* is at most V^2 and $\log \log V^2 = 2 \log V$ so the complexity is *O*(log *V*).
- Each isolated vertex is a separate component of the minimum spanning forest. If we ignore isolated vertices we obtain $V \leq E+1$, so complexity is *O*(log *E*).

We can achieve this bound as follows:

1. Sort the edges by weight using a comparison sort in *O*(*E* log *E*) time;
2. Remove an edge with minimum weight from Graph(*S)* to operate in constant time.
3. Use Disjoint set Data Structure (Union & Find) to keep track of which vertices are in which components. This operation has O(*E*) complexity. There consist two 'find' operations and possibly one union for each edge. Even a simple disjoint-set data structure such as disjoint-set forests with union by rank can perform O(*E*) operations in *O*(*E* log *V*) time. Thus the total time is *O* (*E* log *E*) = *O*(*E* log *V*).

Provided that the edges are either already sorted or can be sorted in linear time (for example with counting sort or radix sort), the algorithm can use more sophisticated disjoint-set data structure to run in *O*(*E* α(*V*)) time, where α is the extremely slowly growing inverse of the single-valued Ackermann function.

The following image gives visual explanation about how Kruskal's Algorithm works to make a Minimum Spanning Tree of a Graph.

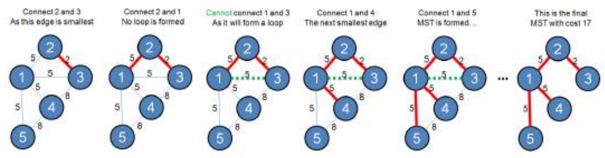


Image 2.3 The Animation of Kruskal's Algorithm for an MST Problem. Source: Halim, Steven. Increasing Your Lower Bound 1st Edition

III. Implementation

As mentioned in the previous part of this paper, the main concern of the problem is to determine which Power Plant should serve a certain area so that the cost of cable and infrastructure maintenance is as low as possible.

To solve the problem, the first task that is need to be done is to represent Indonesia as a graph where the area (cities, villages, power plant, etc.) presented as vertices, while the cable and another infrastructures to connect each area represented with edges. The weight of each edge is measured by the cost that is needed to build and maintain every infrastructure.

Every area (cities, villages, power plant, etc.) only need to be connected to one Power Plant or another area that is already connected with Electricity. But Power Plant is an exception, since a Power Plant doesn't need to be connected to another Power Plant.

This characteristic of Problem makes Minimum Spanning Tree Problem the best Abstraction of The Electricity Distribution Problem in Indonesia. Since the nature of Minimum Spanning Tree is to connect every vertices in the graph with only one edge for each with the minimum total weight of the edges.

But a simple modification need to be done because there are some vertices that must be prevented from being connected one to another, which is the vertices of Power Plant. To outsmart this problem, the graph must be modified with these steps:

1. Construct a Graph that is similar with the Original Graph of Indonesia (for example Graph G').
2. Add an additional vertex for a dummy (for example x).
3. Add edges from x to every Power Plant that exists in the graph with weight of zero (0).
4. Solve the Graph with Prim's or Kruskal's Algorithm as a normal Minimum Spanning Tree.

The reason of the addition a dummy vertex and edges with weight of zero is to make sure a Power Plant is not connected one to another without affecting the total cost of The Minimum Spanning Tree. The effect of this addition will be explained more clearly with an example in the following section of this paper.

The pseudo-code of solution offered by this paper is as following:

A. Prim's Algorithm

```
process(int vtx)
{Input : Number of Vertex that want
to be processed
Output : Priority Queue Added with
edges of the vertex
}
      taken[vtx] <- 1
      for each v connected to 0
            pair_integer tempPair <-
make_pair(vtx, v)
            if NOT taken[0] THEN
                  push( pq, -
tempPair.second, -tempPair.first )

PrimPowerPlant()
{Input: Three Integers denoting the
number of sites, the number of
cables, the number of power plants
Output: The Cables that use to
connect one site to another and the
total cost}

      integer site, cable, nPlant,
plant
      integer src, dst, weight,
totalCost
      graph Indonesia
      vector integer taken
      priority_queue pq

      INPUT (site, cable, plant)
      for i = 0, i < nPlant
            INPUT(plant) {Input
Number of Nodes which is Power Plant}
            Indonesia <- MAKEEDGE(0,
plant, 0)
      for i = 0, i < cable
            INPUT(src, dst, weight) {
Input Another Sites }
            Indonesia <-
MAKEEDGE(src, dst, weight)
      for each v in Indonesia
            color(v) <- WHITE
            key(v) <- infinity
            p(v) <- NIL

      ASSIGN(taken, 0, 0) // Assign
first Vertex to Vector
      process(0)
      totalCost <- 0
      WHILE NOT EMPTY(pq) DO
            pair_integer front <-
top(pq)
            integer u <- -
front.second, w <- -front.first
            if NOT taken(u) THEN
                  WRITEPATH(u,w)
                  totalCost <-
totalCost + w
                  process(u)
  WRITE(totalCost)
```

B. Kruskal's Algorithm

```
KruskalPowerPlant()
{Input: Three Integers denoting the
number of sites, the number of cables,
the number of power plants
Output: The Cables that use to connect
one site to another and the total
cost}

      integer site, cable, nPlant,
plant
      integer src, dst, weight,
totalCost
      vector_of_pair_integer Indonesia

      INPUT (site, cable, plant)
      for i = 0, i < nPlant
            INPUT(plant) {Input Number
of Nodes which is Power Plant}
            Indonesia <- MAKEEDGE(0,
plant, 0)
      for i = 0, i < cable
            INPUT(src, dst, weight) {
Input Another Sites }
            Indonesia <- MAKEEDGE(src,
dst, weight)
      totalCost <- 0
      SORT(Indonesia)
      WRITE(totalCost)
      INITSET(V)
      for i = 0, i < SIZE(Indonesia)
            pair_of_integer_pair front
<- Indonesia[i]
            if NOT ISSAMESET(front)
THEN
                  WRITEPATH(front)
                  totalCost <-
totalCost + front.first
                  UNIONSET(front)
      WRITE(totalCost)
```

IV. EXPERIMENTS

To check the validity of the program that already explained in the previous section, a simple experiment will be done in this section of paper. Let's say that there are nine areas in Indonesia and three of them are Power Plants as described in the following image:

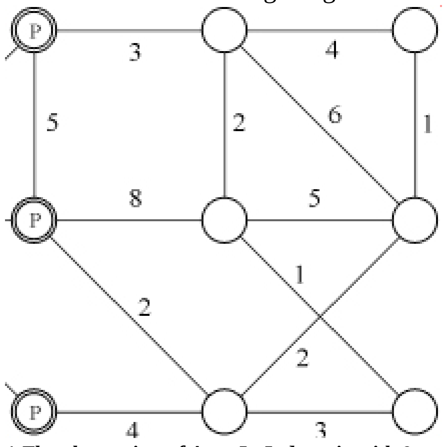


Image 4.1 The abstraction of Area In Indonesia with 9 areas and 3 Power Plants

The input to the program is:

```
1 2 5
1 3 2
2 3 5
3 5 4
2 5 7
2 4 4
4 5 5
5 7 10
4 7 4
4 6 6
6 8 3
6 7 11
7 8 8
7 9 10
```

The illustration of the solution of this example is described in the following image:

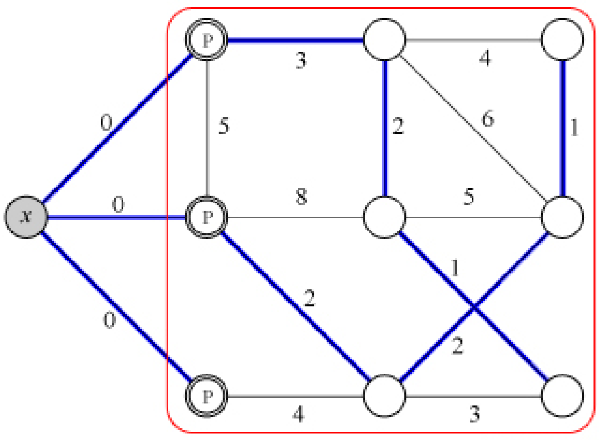
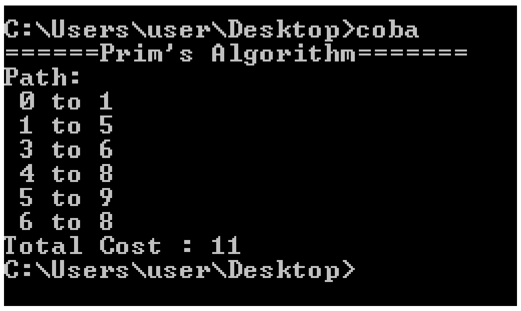


Image 4.2 The abstraction of Solution of the Example Problem

X is a dummy edge to connect with all the power plants avoiding any power plant to connect with each other.

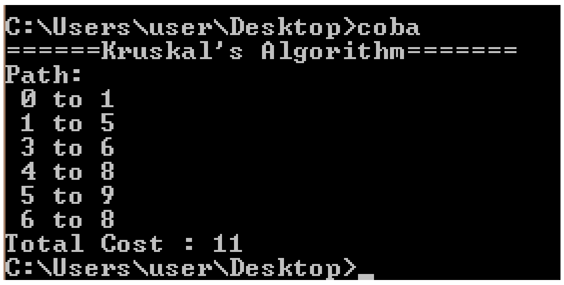
The result of the program itself for Prim's Algorithm is as following:



```
C:\Users\user\Desktop>coba
======Prim's Algorithm=======
Path:
 0 to 1
 1 to 5
 3 to 6
 4 to 8
 5 to 9
 6 to 8
Total Cost : 11
C:\Users\user\Desktop>
```

Image 4.3 Screenshot of Compile Result of Prim's Algorithm

The result of the program for Kruskal's Algorithm:



```
C:\Users\user\Desktop>coba
======Kruskal's Algorithm=======
Path:
 0 to 1
 1 to 5
 3 to 6
 4 to 8
 5 to 9
 6 to 8
Total Cost : 11
C:\Users\user\Desktop>_
```

Image 4.4 Screenshot of Compile Result of Kruskal's Algorithm

V. Algorithm Analysis

For the algorithms that are used in this paper, there are some analysis to determine whether the algorithm is effective from aspect of complexity and also the cost that is produced by the program.

From the aspect of complexity, as explained in the previous section of this paper, the complexity of the program is O(log n) since this program is implementing Prim's and Kruskal's Algorithm. This is still the most efficient method acknowledged until this paper is done. As comparison the brute force algorithm has O (n!) complexity which is so much more inefficient than the algorithm used in this paper.

From the aspect of Total Weight of the Minimum Spanning Tree produced, or in this problem described as the total cost of maintenance, for every test case that is given to the program, the program could always return the most minimum value of the Spanning Tree. The example in this program is one of the test cases that is solved successfully by the program.

But this program is simplifying the problem of Electronic Distribution in Indonesia into a graph that only concern about the cost of maintenance. While in the real life problem, there are a lot of factor to be considered when government want to build or serve an area with a certain Power Plant. Those factors are the condition of nature, ethics in the certain area, the law that is implemented in the certain area, and many more. Those factors can also be calculated to be the weight of the graph, so this algorithm is still capable to be implemented in the real life problem with some more modifications.

VI. Conclusion

Greedy with Prim's and Kruskal's Algorithm is suitable to determine which Power Plant to serve a certain area in Indonesia with the lowest maintenance cost. By using Minimum Spanning Tree, the path of the electricity distribution can be analyzed to determine whether the path is the most efficient path or not. After all the paths are determined, the total cost can be calculated. But the cost of the maintenance must be pre-computed before the program is executed, and a lot of factors must be considered in the calculation of the cost itself.

VII. Acknowledgment

This paper is made to fulfill the assignment of IF2211. The writer say thank you to Mister Rinaldi Munir and Mrs Masayu as Lecturer in this Subject that is always guide the writer in the process of finishing this paper.
The writer is also want to express the gratitude to every person that is helping the making of this paper. The writer want to say thank you to the parent, to D Brotherhood, to another friends. Because of all of you this paper can be done in the right time and in a good form.

References

[1] Munir, Rinaldi. *Diktat Strategi Algoritma*. 2009. Bandung.
[2] Halim, Steven & Halim, Felix. *Competitive Programming 2. This Increases the Lower Bound of Programming Contest. Again*. 2012. Singapore.
[3] www.satwa.org accessed at: Tuesday, 17 December 2013 19.00
[4] www.cs.auckland.ac.nz accessed at: Thursday, 19 December 2013 20.00
[5] www.vis.berkeley.edu accessed at: Thursday, 19 December 2013 21.00
[6] www.esdm.go.id accessed at: Friday, 20 December 2013 10.00

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Desember 2013

[illegible]

Yogi Salomo Mangontang Pratama