Graph Modeling on Intercity Transportation Modes and the Application of Bellman-Ford Algorithm in Determining the Route of Two Cities at Lowest Cost

Aidil Rezjki Suljztan Syawaludin 13517070
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
riskisultan@yahoo.com

***Abstract*—Transportation has always been one of the vital aspects in human lives. Moreover, human, as social creatures, needs other humans. However, most people are separated by geographical features. This encourages transportation to develop. In this modern era, transportation systems have become more complex as technology progresses. This paper will create a simplistic graph model of intercity transportation modes and determine the route of two cities in the system that has the lowest cost. The model will be based on weighted and directed graph, with a few adjustments. To determine the route of two cities at the lowest cost, Bellman-Ford algorithm will be used.**

***Keywords*—Graph, human, lowest cost, transportation**

I. Introduction

Transportation has always been one of the vital aspects in human lives. In this era, people are living more connected than ever before. Most people on earth are living geographically separated. However, this proves to not be an obstacle as mankind progresses. The invention of various transportation means, from the use of animals such as horses, the invention of wheels, the first sailing boat, to the sophisticated means of travel such as airplanes, allows people from all around the world to travel from places far away.

Human is by nature a social creature. That is, human needs other humans. This simple reason might be the one that mostly encourages transportation. Most people are disconnected by geographic features. This is where transportation comes into play. To allow people to get to different places that is either too far away to walk, or is just simply disconnected geographically, people need means of transportation.

Trade is also a factor. As people are separated in different kind of locations, resources available are also different, depending on the location. For example, people living in Indonesia might be able to obtain woods and logs easily but might not be able to get much crude oil and gas. Since those resources are needed, yet is not obtainable locally, people need to trade. Transportation is needed for people to trade easily, since it allows people to move goods from different places.

Naturally, transportation comes with a cost. Most of the time, the distance needed to be traveled matters the most in deciding the cost of transportation. The further it is, the higher the cost gets. Other factors that may influence the cost are namely speed, safety, and comfort.

As of the current time, there are various modes of transportation available for people to use, with different advantages and disadvantages, at different costs. Thus, a way to help people determine the most suitable mode of transportation is needed. This will be useful in applications that need to present its users the cheapest route, for example in applications that specializes on selling tickets of transportation modes.

On this paper, will be discussed the modeling of intercity transportation modes on a weighted graph. The graph model will represent several different cities with edges that represent connection between cities and type of transportation, and weights that represent cost of travel. In determining the route between two different cities at the lowest cost, will be used the Bellman-Ford Algorithm.

II. Definition and Types of Graphs

A. Definition of Graph [1], [2]

A graph consists of a nonempty set of *vertices V* (or *nodes*) and a set of *edges E*. An edge may connect either one vertex or two vertices. Those vertices are called *endpoints* of an edge. A graph is represented by a couple of sets, a set of vertices and a set of edges. In this case, we may define a graph $G = (V, E)$. This means that graph G consists of a set of vertices V and a set of edges E. The number of vertices in a graph is called as *graph's cardinality*, which is denoted as $n = |V|$, and the number of edges in a graph is denoted as $m = |E|$.

As defined before, a graph consists of a nonempty set of vertices, which means that a graph must consist of at least one vertex. However, a graph may consist of an empty set of edges. A graph that consist of one vertex and an empty set of edges is called as *trivial graph*.

Vertices of a graph may be denoted by alphabet letters, such as *a, b, c, ...*, by natural numbers, such as *1, 2, 3, ...*, or by a combination of both. An edge is expressed by a pair of vertices. For example, an edge that connects vertex a to vertex b is expressed as (a, b). An edge may also be denoted as $e_1, e_2, e_3, \ldots$ with each edge representing a pair of vertices. In this case, we may define $e_1 = (a, b)$, which means that edge e_1 connects vertex a to vertex b. The order of vertices pair with same vertices in an edge might differentiate an edge if the graph is a *directed graph*. This will be discussed in the next part.

B. Types of Graph [2]

There are several ways to classify a graph. The first classification of graphs is based on the edges. Based on its edges, a graph may be classified into two types:

1. Simple graph

A graph which consists of no loops and no multi-edge is considered as a simple graph. A simple graph consists of *unordered pair* edges, which means that any edges in a simple graph has no direction. This also means that in a simple graph, an edge (a, b) is the same edge as (b, a). Based on this definition, we may define a graph G = (V, E) that consists of a nonempty set of vertices V and a set of *unordered pairs* of vertices referred as edges E.

2. Unsimple graph

A graph which consists of at least a loop or at least a multi-edge, or both, is considered as an unsimple graph. A loop itself is an edge that has the same endpoints. For example, an edge (a, a) is a loop. Multi-edge itself means that in a set of edges, there is at least two edges that connects the same vertices. For example, a set of edges E that contains edge (a, b) and (a, b) is considered a multi-edge set. There are two types of unsimple graph (not mutually exclusive, a graph may be both type) which are **multigraph** and **pseudograph**. A multigraph is a graph that consists of at least one multi-edge, while a pseudograph is a graph that consists of at least one loop.

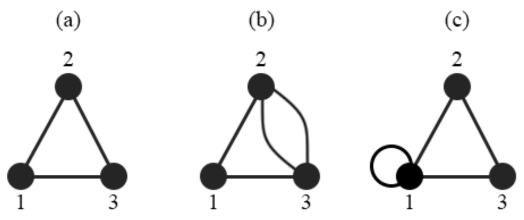


Fig. 1 Graph (a) simple graph, (b) multigraph, (c) pseudograph

The second classification of graphs is based on the orientation of the edges. Based on the direction of its edges, a graph may be classified into two types:

1. Undirected graph

A graph that consists of edges with no directions is considered as an undirected graph. In this type of graph, the edges are a set of *unordered pairs*. This means that if given two edges (a, b) and (b, a), both edges are considered as same.

2. Directed graph (*digraph*)

A graph that consists of edges with direction orientation is considered as a directed graph. These directed edges may be referred to as *arcs*. In a directed graph, the set of edges E consists of *ordered pairs*. This means that if given two edges (a, b) and (b, a), those edges are considered different, which also means that (a, b) $\neq$ (b, a). In an arc (a, b), vertex a is called as *initial vertex* while vertex b is called as *terminal vertex*. A directed graph may contain a loop edge, but it cannot contain multi-edges.

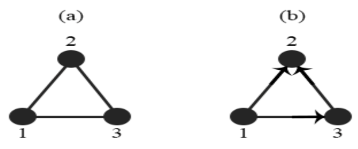


Fig.2 Graph (a) undirected graph, (b) directed graph

III. Finding Path of Two Vertex With Lowest Total Weight Using Bellman-Ford Algorithm

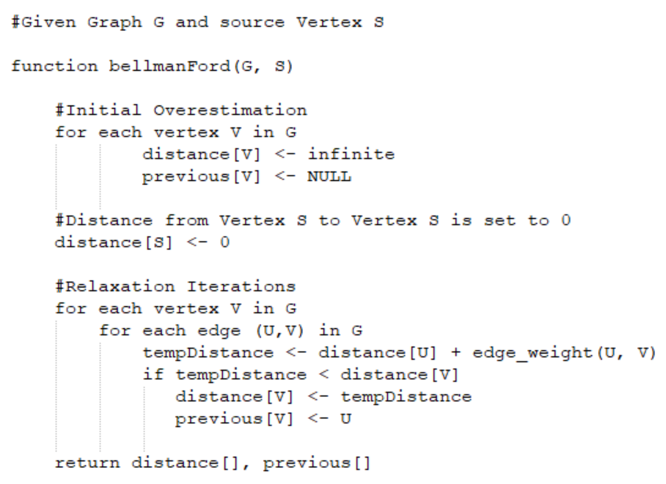
A. Bellman-Ford Algorithm [3]-[5]

Bellman-Ford algorithm is used to find a path from a single vertex in a weighted-directed graph (*single source shortest path*) to any other vertices in the graph. This algorithm allows for negative weights on the edges of the graph. However, there cannot exist *negative cycles*, which is a circuit in a graph with total weights less than 0. This algorithm is designed for directed graphs, but it is also applicable on undirected graphs.

To use Bellman-Ford algorithm on an undirected graph, some adjustments are needed. For every edges E in an undirected graph G, a directed edge is created from both endpoints. For example, for an edge E = (a, b) in an undirected graph G, edge (a, b) and edge (b, a) is created, both with the same weight as the weight of E.

B. Finding Path of Two Vertex with Lowest Weight [4], [5]

Bellman-Ford algorithm works by doing an overestimation on the shortest path from a vertex to the other vertices in the graph. Then, the algorithm does an iteration |V| - 1 times. These iterations are called relaxations. In these relaxations, the algorithm finds new paths with lower weights from the source vertex to all other vertices. The pseudocode of Bellman-Ford algorithm is given below.



```
#Given Graph G and source Vertex S

function bellmanFord(G, S)

    #Initial Overestimation
    for each vertex V in G
            distance[V] <- infinite
            previous[V] <- NULL

    #Distance from Vertex S to Vertex S is set to 0
    distance[S] <- 0

    #Relaxation Iterations
    for each vertex V in G
        for each edge (U,V) in G
            tempDistance <- distance[U] + edge_weight(U, V)
            if tempDistance < distance[V]
               distance[V] <- tempDistance
               previous[V] <- U

    return distance[], previous[]
```

Fig. 3 Pseudocode of Bellman-Ford algorithm.
Source: https://www.programiz.com/dsa/bellman-ford-algorithm

The given pseudocode also keeps track of the previous vertex of the path to a vertex.

The algorithm may be divided into two parts, *initialization* and *relaxation*. On the initialization, the algorithm overestimates the distance from source Vertex S to any other vertices by setting all distance value to infinity except the distance to Vertex S itself, which is set to 0. It also sets the previous vertex to *NULL*. On the relaxation, the algorithm starts to find the lower-weighted paths, which will lead to the lowest-weighted paths in the end. The iterations in relaxation part should be done only |V| - 1 times, and each of these iterations represents the number of edges involved in the lowest-weighted path produced. This means that the-n_{th} iteration will produce the lowest-weighted paths that at most involves n edges. The illustration of how this algorithm works will be shown later.

IV. Modeling Of Intercity Transportation Into A Graph

A. Problem Explanation

In this paper, an example problem will be given and modeled. Given a set of cities C_1, C_2, C_3, C_4, C_5. Each of these cities are connected through different modes of intercity transportations. In this example, the cities are connected as follows:

- C_1 is connected to C_2 by a plane with the cost of Rp. 700.000,00- , connected to C_5 by a train with the cost of Rp. 200.000,00-, and connected to C_4 by a plane with the cost of Rp. 800.000,00-.
- C_2 is connected to C_1 by the same plane before that goes in reverse, connected to C_3 by a bus with the cost of Rp. 100.000,00-, connected to C_4 by a train with the cost of Rp. 250.000,00-, and connected to C_5 by a plane with the cost of Rp. 1.300.000,00-.
- C_3 is connected to C_2 by the same bus that goes in reverse and connected to C_4 by bus with the cost of Rp. 50.000,00-.
- C_4 is connected to C_1 by the same plane that goes in reverse, connected to C_2 by the same train that goes in reverse, connected to C_3 by the same bus that goes in reverse, and connected to C_5 by plane with the cost of Rp. 1.600.000,00-.
- C_5 is connected to C_1 by the same train that goes in reverse, connected to C_2 by the same plane that goes in reverse, and connected to C_4 by the same plane that goes in reverse.

The following table will show the connections between each cities which will be denoted by the cost of travels and the transportation modes. Each element represents a pair of transportation modes and cost of travel (in thousand rupiahs).

	C_1	C_2	C_3	C_4	C_5
C_1	(-, 0)	(P, 700)	(-, -)	(P, 800)	(T, 200)
C_2	(P, 700)	(-, 0)	(B, 100)	(T, 250)	(P, 1300)
C_3	(-, -)	(B, 100)	(-, 0)	(B, 50)	(-, -)
C_4	(P, 800)	(T, 250)	(B, 50)	(-, 0)	(P, 1600)
C_5	(T, 200)	(P, 1300)	(-, -)	(P, 1600)	(-, 0)

Table. I Connection between cities in the example.
B: Bus
P: Plane
T: Train

B. Modeling the Intercity Transportation Modes into Graph

For the given example problem before, will be created a model that may explain the situation. In order to model the problem into a graph, the vertices and and the edges need to be determined first. In this problem, the cities will be modeled as vertices on the graph and the connections between cities will be modeled as edges between each vertex.

By modeling the cities from C_1, C_2, C_3, C_4, and C_5 into vertices, we will get a graph with five vertices, which will mean that $|V| = 5$. For the edges, a few adjustments are needed since the data does not only contain weight for the edges, but also the type of transportation mode represented by the edge itself. For this, a pair of transportation mode and cost will become the component of an edge in the graph.

Using the model explained before, a graph model of the example problem may be created as follows.

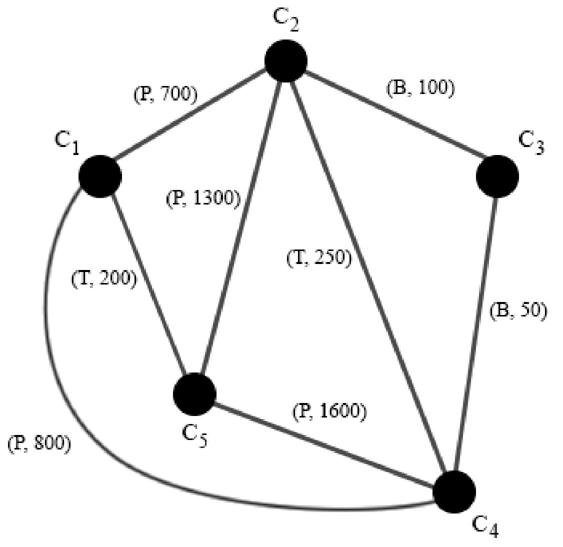


Fig. 4 Graph model on the example problem
B: Bus
P: Plane
T: Train

In Fig. 4, the graph model of the example problem can be seen, with each vertices representing each cities, and each edges representing connections between each cities, containing the transportation mode and weighted cost.

V. Application of Bellman-Ford Algorithm in Determining the Route Between Two Cities at Lowest Cost

By using Bellman-Ford algorithm, the path between vertices in a weighted-directed graph may be determined. The algorithm will be used on the previous graph model of intercity transportation modes.

However, since the previous graph model is still a weighted-undirected graph, it still needs to be converted into a weighted-directed graph that still holds the exact same information as before. To do this, for every edge in the graph model, will be created a directed edge that connects both endpoints, with the exact same weight. For example, the undirected edge (C_1, C_2) may be converted into two edges, (C_1, C_2) and (C_2, C_1), with the exact same weight for both edges, which is (P, 700). The resulting weighted-directed paragraph is as follows.

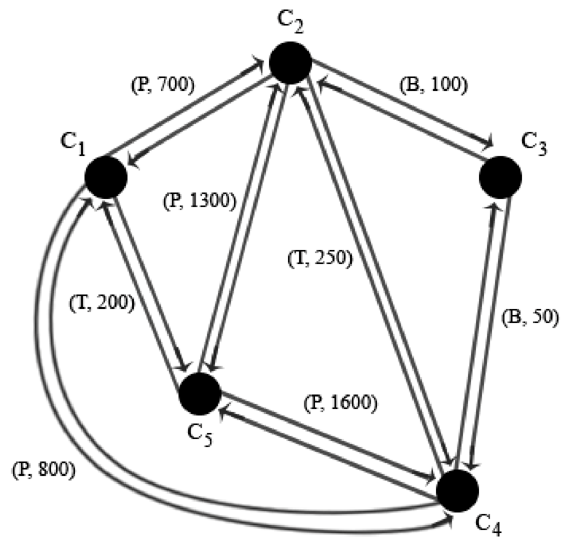


Fig. 5 Graph model on the example problem

B: Bus
P: Plane
T: Train

The graph in Fig. 5 is now a weighted-directed graph version of the previous graph model in Fig. 4. With this graph, the Bellman-Ford algorithm may be used to find the path with lowest cost from a vertex to any other vertices.

In this example, the path with the lowest cost from city C_5 to city C_3 will be sought. As explained before, the Bellman-Ford algorithm will first overestimate the cost from the source vertex to any other vertices, and set the distance to itself to 0. In order to make the process easier to understand, an illustration will be used. The table shows the distances from C_5 to all vertices in the graph and the previous vertex to reach the target vertex.

Vertex	C_5	C_1	C_2	C_3	C_4
Distance	0	~	~	~	~
Previous	-	-	-	-	-

Table. II Initialization process.

After the initialization process, the Bellman-Ford algorithm continues to the relaxation part. In this part, the algorithm will do iterations $|V|$ - 1 times. In this example problem, since we have five vertices ($|V|$ = 5), the algorithm will iterate the relaxation $|V|$ - 1 times which is four times.

The iteration of this relaxation part will iterate each vertex in the weighted-directed graph, and will run through all the edges. The way to illustrate this is by imagining a pointer to the current vertex. The pointer will iterate through all vertices in the graph. In that iteration, the algorithm will check all edges that connects the current vertex pointed to any other vertices, and check the cost.

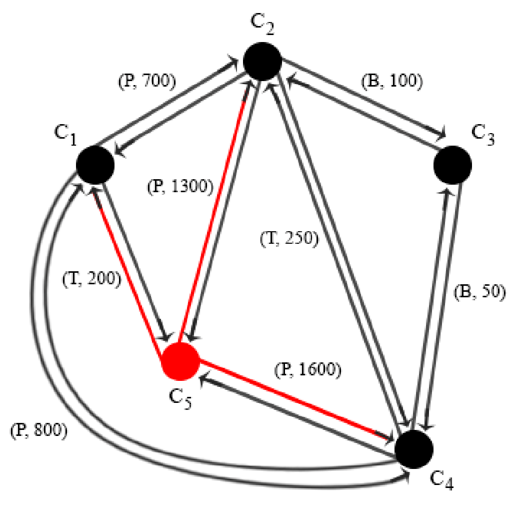


Fig. 6 Pointer is currently pointing to Vertex C_5.

In Fig.6, the pointer is currently pointing to Vertex C_5. The Bellman-Ford algorithm then checks every edges that originates from Vertex C_5 to any other vertices. In Fig. 5, it shown that there are three edges from Vertex C_5 to other vertices, namely edges (C_5, C_1), (C_5, C_2), and (C_5, C_4), all with different weights. The algorithm then loops through those edges, and updates the distance table if the distance from the current pointed vertex to the target vertex added with current pointed vertex cost is lower than the one currently saved in the table. For example, the edge (C_5, C_1) has the cost of 200 and the current pointed vertex has the cost of 0. From Table. II it is known that the currently saved distance from C_5 to C_1 is ~ (infinite). That means the condition is fulfilled and the table is updated. The process of updating the table involves changing the distance and the previous vertex, in this case, the distance is 200 and previous vertex is set to C_5. Continuing the next edge (C_5, C_2), this edge costs 1300 and the current cost to currently pointed vertex is 0, these adds to 1300, which is also less than ~ (infinite), thus the table is updated again. To the next edge (C_5, C_4), this edge has the weight of 1600 and added with currently pointed vertex cost becomes 1600, which is again less than ~ (infinite), and the table is updated. All the edges originating from the currently pointed vertex, C_5, has already been iterated. After this iteration, the table becomes as follows.

Vertex	C_5	C_1	C_2	C_3	C_4
Distance	0	200	1300	~	1600
Previous	-	C_5	C_5	-	C_5

Table. III First relaxation, iterating on Vertex C_5.

The algorithm continues the iteration. The pointer changes to another vertex. The order of this vertex iteration does not really matter, since it is guaranteed that it will always produce the same path with lowest weight. However, if the algorithm is optimized, it may affect the order may affect performance since the iteration may stop earlier in some orders, but this is not a concern in this paper, because the algorithm used in this paper is the basic Bellman-Ford algorithm.

Continuing the iteration, the pointer changes to C_1. This currently pointed vertex has edges directing towards vertices C_2, C_4, and C_5. This means that the table values that may change in this iteration are only those involving vertex C_2, C_4, and C_5. From the previous table, the currently pointed vertex costs 200. Observing edge (C_1, C_2), the cost is 700, added with currently pointed vertex, the cost becomes 900, which is less than 1300 from the previous table (the cost from C_5 to C_2), so the table is updated. To the next edge, (C_1, C_4), it costs 800, added with current cost, the cost becomes 1000. This cost is lower than the previous cost, which is 1600, thus the table is updated. To the last edge on the currently pointed vertex, (C_1, C_5), which costs 200, added with currently cost, it becomes 400. Checking that cost to the previous value, which is 0, definitely does not satisfy the condition, and the table is not updated. From this, it is also discovered that all edges directing toward the source vertex, in this case C_5, does not need to be checked, since it will always fail the condition, because the graph has no negative weight. After this current iteration, the table is updated as follows.

Vertex	C_5	C_1	C_2	C_3	C_4
Distance	0	200	900	~	1000
Previous	-	C_5	C_1	-	C_1

Table. IV First relaxation, iterating on Vertex C_1.

The iteration continues onto the third vertex, C_2. This vertex has edges directing to C_1, C_3, C_4, and C_5. From the previous table, this vertex costs 900. For the first edge, (C_2, C_1), the cost is 700, added with the current cost, it gets to 1600. This costs more than what the previous table has, thus the table is not updated. To the next connection, (C_2, C_3), it costs 100 and with current costs becomes 1000. Compared to the previous value on the table, 1000 is less than ~ (infinite), so the table is updated. Next, (C_2, C_4), this costs 250 and added with the current cost becomes 1050, which is not less than the previous value toward C_4, 1000, so the table is not updated. To the final edge in this vertex, (C_2, C_5), this edge is going toward source vertex, so it may be skipped. After this iteration, the table becomes as follows.

Vertex	C_5	C_1	C_2	C_3	C_4
Distance	0	200	900	1000	1000
Previous	-	C_5	C_1	C_2	C_1

Table. V First relaxation, iterating on Vertex C_2.

The algorithm then continues to the next vertex, C_3. This vertex is connected to C_2 and C_4, and this current vertex costs 1000. For the first pair, (C_3, C_2), this costs 100, and added with the current cost, this costs 1100. Comparing this cost to the previous value on the table, 1100 is not less than 900, thus the table stays the same. To the pair (C_3, C_4), it costs 50, and with the current vertex cost, it becomes 1050. Again, comparing this value to the previous value on the table, 1050 is not less than 1000, and the table is not updated. The table stays the same as before, shown as follows.

Vertex	C_5	C_1	C_2	C_3	C_4
Distance	0	200	900	1000	1000
Previous	-	C_5	C_1	C_2	C_1

Table. VI First relaxation, iterating on Vertex C_3.

Onto the last vertex in the first relaxation, the pointer now points to Vertex C_4, which cost 1000. This vertex is connected to C_1, C_2, C_3, and C_5. The first edge, (C_4, C_1), this edge costs 800, plus the current cost, it gets to 1800. This cost is higher than the previous cost, which is 200, so the table is left untouched. To the next edge, (C_4, C_2), this costs 250 and with the current cost becomes 1250. The table holds previous value of 900, so this edge does not satisfy the condition and the table is unchanged. For the next edge, (C_4, C_3), this edge has weight of 50, added with the current cost, it now costs 1050, while the table holds previous value of 1000, so the table is not updated. Similar to the previous vertex, this iteration does not update any value on the table, and the table stays as follows.

Vertex	C_5	C_1	C_2	C_3	C_4
Distance	0	200	900	1000	1000
Previous	-	C_5	C_1	C_2	C_1

Table. VII First relaxation, iterating on Vertex C_4.

Because all of the vertices have already been iterated, the first relaxation is finished. The algorithm now enters the second relaxation, starting all over again, from Vertex C_5.

Starting back from C_5, this vertex costs 0, and is connected to C_1, C_2, and C_4. Beginning from pair (C_5, C_1) which costs 200, with the current costs, it stays 200. Compared to the previous value on the table, 200 is not less than 200, thus the table is not updated. To the next edge (C_5, C_2) with the cost of 1300, added with current cost of 0, it still costs 1300. However, the previous value on the table is 900, so it does not satisfy the condition, and the table is unchanged. To the edge (C_5, C_4), this edge weighs 1600, with current cost, it stays 1600, compared to the previous value of 1000, it also does not satisfy the condition. Because none of the edge satisfy the condition, the table is unchanged in the current iteration.

Vertex	C_5	C_1	C_2	C_3	C_4
Distance	0	200	900	1000	1000
Previous	-	C_5	C_1	C_2	C_1

Table. VIII Second relaxation, iterating on Vertex C_5.

Now, the pointer is pointing to the next vertex, C_1. This vertex costs 200, and has edges towards C_2, C_4, and C_5. The edge (C_1, C_2) costs 700, with current vertex cost, it now costs 900, and compared to previous value, it is not less than 900, so the table is unchanged. The edge (C_1, C_4) costs 800, with current cost becomes 1000, and is not less than previous value of 1000, so the table is not changed. The next edge (C_1, C_5) is heading toward the source vertex and may be skipped. After this iteration, the table still stays the same, as follows.

Vertex	C_5	C_1	C_2	C_3	C_4
Distance	0	200	900	1000	1000
Previous	-	C_5	C_1	C_2	C_1

Table. IX Second relaxation, iterating on Vertex C_1.

The pointer then moves to C_2, this vertex costs 900, while being connected to C_1, C_3, C_4, and C_5. The edge of (C_2, C_1) costs 700, and becomes 1600 because of current vertex cost. This does not satisfy the condition, since the previous value is 200. Next, edge (C_2, C_3) with the cost of 100, plus the current vertex cost becomes 1000, and is still not less than 1000. The next pair (C_2, C_4) is costing 250, and with the current cost it gets to 1150. This value is not less than the previous one, which is 1000, thus the table is not updated. The table is not updated in this iteration either and stays the same as follows.

Vertex	C_5	C_1	C_2	C_3	C_4
Distance	0	200	900	1000	1000
Previous	-	C_5	C_1	C_2	C_1

Table. X Second relaxation, iterating on Vertex C_2.

The pointer now is pointing to C_3, with the current cost of 1000. This vertex has edges heading to C_2 and C_4. The first pair of (C_3, C_2) costs 100, and costs 1100 with the current vertex cost, which is not less than the previous 900. Now, the pair (C_3, C_4) weighs 50, and becomes 1050 because of the current vertex cost. This is no less than 1000, the previous value. The table is not changed in this iteration and stays the same as follows.

Vertex	C_5	C_1	C_2	C_3	C_4
Distance	0	200	900	1000	1000
Previous	-	C_5	C_1	C_2	C_1

Table. XI Second relaxation, iterating on Vertex C_3.

The algorithm continues to Vertex C_4, with current cost of 1000. This vertex is connected to C_1, C_2, C_3, and C_5. The edge of (C_4, C_1) costs 800, and becomes 1800 with current vertex cost. It is definitely not less than 200. For edge (C_4, C_2) that costs 250, added with 1000 from current vertex cost, now it costs 1250. Compared to 900 from the previous value, it does not satisfy the condition. To the edge (C_4, C_3), this edge weighs 50, with total of 1050 because of the current vertex cost. Comparing it to 1000 from the previous value shows that it does not satisfy the condition, thus the table is not updated. This iteration does not change the table, and it stays as follows.

Vertex	C_5	C_1	C_2	C_3	C_4
Distance	0	200	900	1000	1000
Previous	-	C_5	C_1	C_2	C_1

Table. XII Second relaxation, iterating on Vertex C_4.

This ends the second relaxation. The Bellman-Ford algorithm will continue to the third relaxation, starting from the beginning vertex again. However, since the second relaxation does not update the table at all, the third and fourth relaxations will not update.

Vertex	C_5	C_1	C_2	C_3	C_4
Distance	0	200	900	1000	1000
Previous	-	C_5	C_1	C_2	C_1

Table. XIII The table after third relaxation.

Vertex	C_5	C_1	C_2	C_3	C_4
Distance	0	200	900	1000	1000
Previous	-	C_5	C_1	C_2	C_1

Table. XIV The table after fourth relaxation.

After doing $|V|$ - 1 relaxations, which is four times, the algorithm finally found the path from source Vertex C_5 to any other vertices on the graph with the lowest total of weight. From Table. XIV, it is shown that the path with lowest total weight from Vertex C_5:

1. C_5 to C_5 (itself) costs 0,
2. C_5 to C_1 costs 200,
3. C_5 to C_2 costs 900,
4. C_5 to C_3 costs 1000, and
5. C_5 to C_4 costs 1000.

To find out the route with lowest weight, a stack data structure will be used. For example, the route from C_5 to C_3 with the lowest weight may be traced through this method. First, push C_3 to the stack and go to the data in the table that holds C_3, then check the previous vertex of that target vertex, which is C_2. Push Vertex C_2 to the stack. Now continue to find the data of C_2 on the table and check the previous vertex. Vertex C_1 is the previous vertex in this route that leads to C_2, thus C_1 is pushed to the stack. Continuing by finding the data of C_1, the previous vertex is C_5. Thus, C_5 is pushed to the stack. This process is actually looped from the target vertex until the source vertex is found and pushed to the stack. Now, a stack is produced as follows.

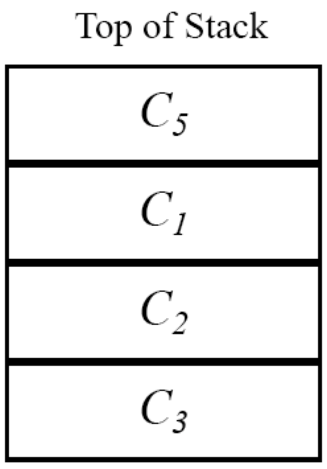


Fig. 7 Stack produced for path from C_5 to C_3.

The stack now represents the vertex that is passed by the route with the lowest weight from Vertex C_5 to C_3. To access the path, the stack is popped, until the stack is empty and the target vertex is reached.

With the Bellman-Ford algorithm, the route for intercity transportation modes with the lowest cost may be determined. From the example, it has been found out that from city C_5 to city C_3, the route that has the lowest cost is through city C_5 to city C_1, then go through city C_5, and arrive at city C_5. This route will cost Rp. 1.000.000,00-. The transportation mode may also be found out since the edge has been adjusted to suit the needs of the graph model of this intercity transportation modes. As example again, to go from city C_5 to C_3, the best route that costs the least is by Train from C_5 to C_1, then continues by plane to C_2, and finally arrive to C_3 by bus.

VI. Conclusion

Graph has many applications in modeling various problems. In this paper, graph is used to model intercity transportation modes. This graph model makes the system easier to understand and allows for procedural and mathematical approach to find solutions for various problem involving the modeled system. For example, the graph model of intercity transportation modes allows procedural approach to determine the route between two cities that costs the least. The graph model in this paper still only involves costs and is still a very simplistic approach to the problem. However, this also proves that graph modeling allows for better approach in finding solutions and this model may also be developed even further to consider other factors of transportations.

VII. Acknowledgment

The author of this paper thanks the lecturer of the authors class of Discrete Mathematics, Dr. Rinaldi Munir, for the knowledge shared and taught in the class and also the guidance that allows the author to finish this paper. The author also thanks all the authors of the references, also to others that may have helped the making of this paper directly and indirectly.

References

[1] Rosen, K. H. (2011). *Discrete Mathematics and Its Applications 7th Edition*. New York, NY: McGraw-Hill Education.
[2] Munir, R. (2009). *Matematika Diskrit*. Bandung: Informatika Bandung.
[3] Bellman, R. (1958). On a routing problem. *Quarterly of Applied Mathematics*. 16(1). 87-90. DOI: 10.1090/qam/102435
[4] Bellman-Ford Algorithm | DP-23. Retrieved from https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/
[5] Bellman Ford's Algorithm. Retrieved from https://www.programiz.com/dsa/bellman-ford-algorithm

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, 10 Desember 2018

[illegible]

Aidil Rezjki Suljztan Syawaludin
13517070