Proceedings of the First





University Journal of Research and Innovation

December, 2019

Organized by
University of Computer Studies (Pakokku)

Proceeding of

The First University Journal of Research and Innovation 2019

December , 2019

Organized by

University of Computer Studies (Pakokku)

Department of Higher Education ,

Ministry of Education , Myanmar

University Journal of Research and Innovation

Volume 1, Issue 1 2019

University Journal of Research and Innovation 2019

Volume 1 , Issue 1 , 2019

This journal and individual papers published at www.ucspkku.edu.mm.

Copies:110

All research papers in this journal have undergone rigorous peer-reviewed which is published annually. Full papers submitted for publication are refereed by the Associate Editorial Board through an anonymous referee process.

The authors of the paper bear the responsibility for their content.

Papers presented at the First University Journal of Research and Innovation(UJRI), University of Computer Studies (Pakokku), December 2019.

UJRI 2019 Editorial Board

UJRI 2019 Editorial Board

Proceedings of

The First University Journal of

Information and Computing Science 2019

December, 2019

Contents

Parallel & Distributed Computing

Image Processing

Human Computer Interaction

Database Management System & Information Retrieval

Network & Security

Data Mining & Machine Learning

Digital Business Management

Electronics

Embedded System

Cloud Computing

Software Engineering and Web Engineering

Digital Signal Processing

Theoretical Nuclear Physics

Material Science

Mathematics

Experimental Nuclear Physics

English Language

Myanmar Language in Literature

Author Index

Author	**Page No.**

Application of Dijkstra's Shortest Path Algorithm for Road Map Estimation in Sagaing Region

Thin Thin Swe
Department of Engineering Mathematics
Mandalay Technological University
Mandalay, Myanmar
thinthinswe.dr.mtu@gmail.com

San San Maw
Department of Engineering Mathematics
Mandalay Technological University
Mandalay, Myanmar.
dr.sansanmaw2017@gmail.com

Abstract

***This research paper concerns with the application of the Dijkstra's algorithm in graph theory for the estimation of shortest road map to eleven destinations in Sagaing region. This algorithm can be used to find the shortest paths with positive weight graph from a single source node to a single destination node by denoting permanent if it has been determined. In this paper, Sagaing is used as a source node and the other ten towns in Sagaing region are used as destination nodes. The goal of this paper is to find the shortest paths from Sagaing to other towns by comparing the weighted values between any two different paths with their edge lengths that assigned by actual values. The values of distances and time between any two destinations are taken from Myanmar distance calculator website and compared the accuracy of the results using those values in Google Map*.**

Keywords— Path Finding, Weighted Graph, Shortest Paths, Dijkstra's Algorithm. Myanmar distance calculator.

1. Introduction

Sagaing region is an administrative region of Myanmar, located in the north-western part of the country. The region has an area of 93527 km^2 and the largest region among the seven regions in Myanmar. The capital city of Sagaing region is Monywa. Sagaing region consists of 10 districts divided into 34 townships with 198 wards and villages. The major towns in this region are Monywa, Shwebo, Sagaing, Yinmarbin, Kanbalu, Katha, Tigyaing, Kale, Tamu, Mawlaik and Pinlebu [1]. In many fields of applications, graphs theory [2] plays vital role for various modelling problems in real world such as travelling, transportation, traffic control, communications, and various computer applications and so on. This paper provides the shortest paths from one place to another by using Dijkstra's Algorithm in graph theory [3], [4]. Firstly, the descriptions of the algorithm are presented. Then the steps of the algorithm are explained. Finally the detailed implementation of the algorithm is illustrated to find the optimal paths for travelling to the eleven major towns in Sagaing region with shortest distances and minimum time.

2. Resources and Method

A graph *G* consists of two finite sets, a set *V* of points, called vertices, and a set *E* of connecting lines, called edges, such that each edge connects two vertices, called the endpoints of the edge [1]. Although straight lines are used to represent the edges, they are not straight in practice. Any place can be visited from any other place directly because all the destinations are linked by roads to each other. In this case, it is very important to find the shortest route, i.e., the route with the shortest total mileage or minimal travel time for overall trip. In this paper we consider one-to-all shortest path problem for determining the shortest path from a start town, Sagaing, to all the other ten famous destinations in Sagaing region.

2.1 Problem Definition

The problem of finding the shortest path between two intersections on a road map may be modeled as a special case of the shortest path problem in graphs, where the vertices correspond to intersections and the edges correspond to road segments, each weighted by the length of the segment [1]. For a traveller, who wants to visit all interesting and famous places in a region, it is important to know the effective way. In this situation, not every road is equal. Some of them are longer, some aren't in good shape, and some have more traffic. i.e., you need more time for traversing some roads than other. We may represent that time with weights that we assign to the roads. When you plan a journey, there are different factors you might consider such as the shortest distance, the minimum time and the lowest cost for effective travelling. In this paper, we consider to optimize distance and time. The travelling cost from one place to another is not considered because costs are changed according to time period we collected.

2.1.1. Dijkstra's Iterative Shortest Path Algorithm

Dijkstra's algorithm, published in 1959 and named after its creator Dutch computer scientist Edsger Dijkstra, can be applied on a weighted graph [1]. The graph can either be directed or undirected. The following are the simple steps of the Dijkstra's algorithm:

Step 1. Initialization

- Label the start vertex as 0.
- Box this number (permanent label).
- Label each vertex that is connected to the start vertex with its distance (temporary label).

Step 2. Box the minimum number.

- From this vertex, consider the distance to each connected vertex.
- If a distance is less than a distance already at this vertex, cross out this distance and write in the new distance. If there was no distance at the vertex, write down the new distance.

Step 3. Repeat from step 2 until the destination vertex is boxed.

When a vertex is boxed you do not reconsider it. You need to show all temporary labels together with their crossing out. The Dijkstra's algorithm for shortest paths is as follows: [2];

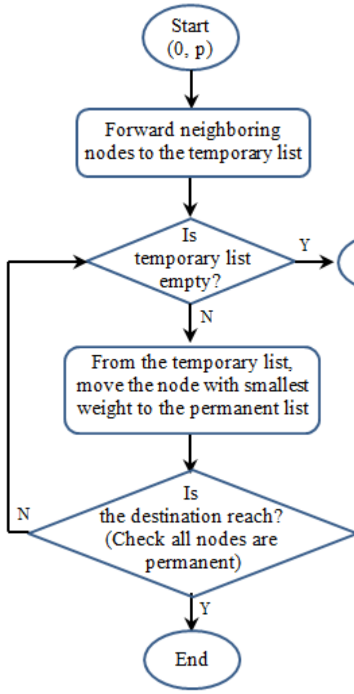


Figure 1. Flowchart for Dijkstra's shortest path algorithm

3. Result and Discussion

In this paper, the application of Dijkstra's algorithm to effective travelling from Sagaing to other ten towns in that region has been illustrated. Everyone who wants to travel needs to optimize the route for saving time and money. Although the detailed calculation for optimal result of distance has been focused, the other factors such as travel time that may effect in travelling have also been considered in this paper. Firstly, the weights on the links are referred as distances (km) for corresponding route.

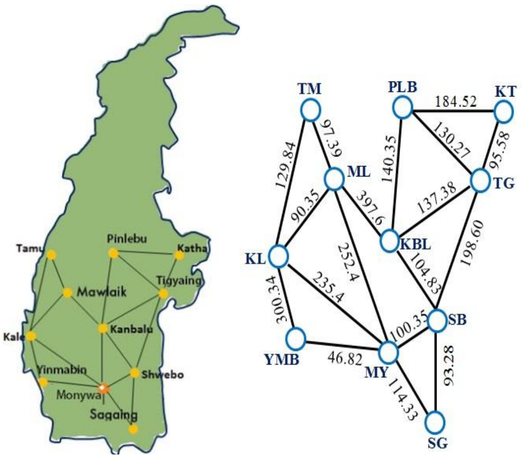


Figure 2. Representation of towns in Sagaing region by weighted distance (km) connected graph [5]

3.1. Implementation of the Algorithm

The implementation steps of the Dijkstra's shortest path algorithm are presented as follows:

Step 1– To define the initial (start) town and label that point as number zero and box it (named it as permanent or current).

Step 2 – To search the towns that can be reached from permanent (current) and select the nearest town by using the formula;

$$new\ d_j = \min\{d_j, d_i + d_{ij}\}$$

where d_j is the distance of adjacent town j. d_i is the distance value for the index of the current town, d_{ij} is the distance values between i^{th} town and j^{th} town.

Step 3– Repeat the step 2 until all destination towns are permanent by changing the updated town with shortest distance to permanent list and box it. If we cannot reach any temporary labelled node from the current node, then all the temporary labels become permanent.

Step 4– Construct a minimum spanning tree to describe the shortest distances from start town to other destination towns in given network.

In this paper, we illustrate to find the optimal path (minimum spanning tree for shortest paths) from Sagaing to other destination towns in Sagaing region as follows.

Step 1

Sagaing (SG) is designed as the current town and the state of the SG is (0, p). Every other town has state (∞, t).

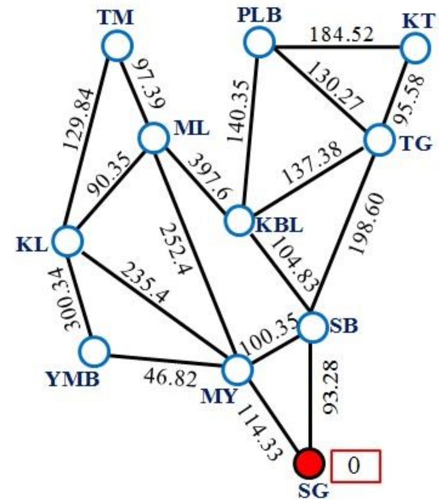


Figure 3. The states of current town Sagaing (SG) as permanent

Step 2

The towns named Monywa (MY) and Shwebo (SB) can be reached from the current town SG. Update distance values for these cities;

$$d_{Mon} = \min\{\infty, 0 + 114.33\} = 114.33$$
$$d_{SB} = \min\{\infty, 0 + 93.28\} = \boxed{93.28}$$

Hence the shortest distance is 93.28 at Shwebo (SB).

Box the number 93.28 at Shwebo and the state label at SB changes to permanent so its state is (93.28, p) while the states of Monywa (MY) remain temporary.

SB becomes the current town as shown in Figure 4.

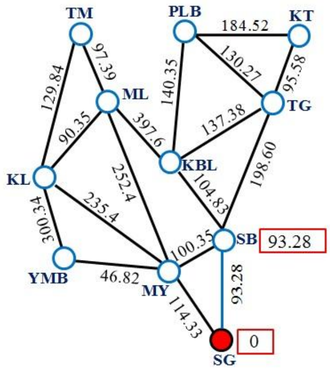


Figure 4. Showing the current town Shwebo

Step 3

We are not done, not all towns have been reached from SG, so we perform another iteration step (back to step 2).

Another implementation of step 2:

Monywa (MY), Kanbalu (KBL) and Tigyaing (TG) can be reached from the current city SB.

Update distance value for these towns.

$d_{MY} = \min\{114.33, 93.28 + 100.35\}$
$= \boxed{114.33}$

$d_{KBL} = \min\{\infty, 93.28 + 104.83\} = 198.11$

$d_{TG} = \min\{\infty, 93.28 + 198.60\} = 291.88$

Hence the shortest distance is 114.33 at Monywa (MY).

Box the number114.33 at Monywa and the state label at MY changes to permanent so its state is (114.33, p) while the states of KBL and TG remain temporary.

Monywa becomes the current town as shown in Figure 5.

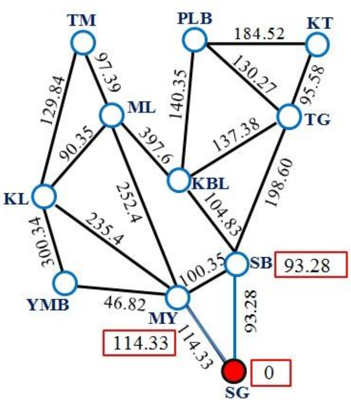


Figure 5. Showing current town Monywa

Another Step 2

Yinmabin (YMB), Kale (KL) and Mawlaik (ML) can be reached from the current town Monywa (MY).

Update distance value for these towns.

$d_{YMB} = \min\{\infty, 114.33 + 46.82\} = \boxed{161.15}$

$d_{KL} = \min\{\infty, 114.33 + 235.4\} = 349.73$

$d_{ML} = \min\{\infty, 114.33 + 252.43\} = 366.76$

Now between the towns YMB and ML, the shortest distance is 161.15 at Yinmabin (YMB).

Box the number 161.15 at Yinmabin and the state label at YMB changes to permanent so its state is (161.15, p) while the states of Mawlaik (ML) remain temporary.

YMB becomes the current town as shown in Figure 6.

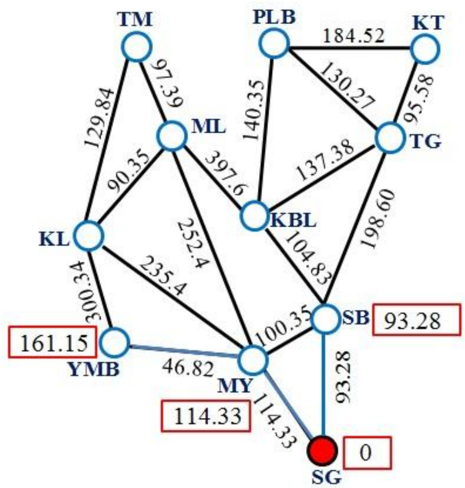


Figure 6. The states of current town Yinmarbin (YMB) as permanent

Similar repeated calculation can be done for step 2 again and again until all towns have been changed to permanents.

Step 4

The final graph showing the minimum spanning tree of distances from start town, Sagaing, to other towns can be seen as follows:

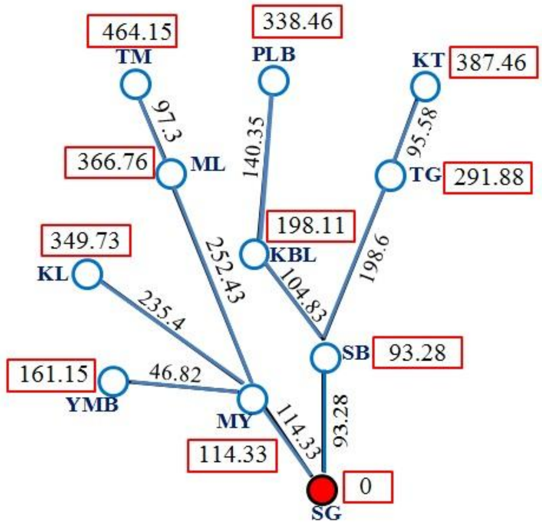


Figure 7. The minimum spanning tree of shortest paths (km) from Sagaing to other towns in Sagaing region

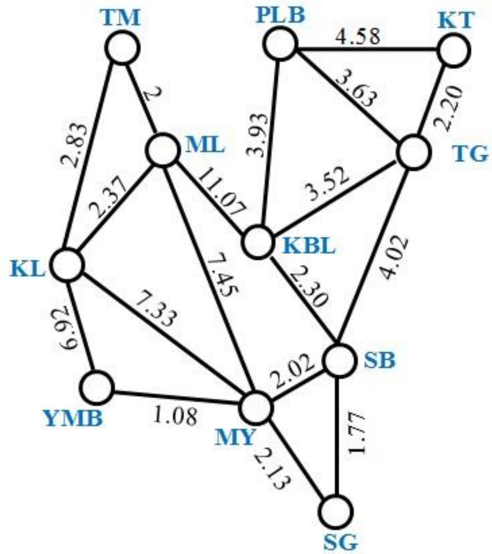


Figure 8. Travel time (hour) between all pair towns in Sagaing region

The values assigned at each town (node) shown in graph with red-boxes, Figure 7, are the shortest distances to travel from Sagaing. Likewise, we can start any town and using the same algorithm to travel any other towns in effective way [6]. The order of towns forming permanent or current destinations for each iteration steps is Sagaing, Shwebo, Monywa, Yinmabin, Kalay, Mawlaik, Kanbalu, Tigyaing, Pinlebu, and Katha. Tamu is an end town of the network because its adjacent towns, Kale and Mawlaik, are already selected as permanent towns. So it is automatically selected as permanent by implementation step 3.

We can also calculate for the minimum travel time started from Sagaing to other towns using the actual travel time (hour) between any two pair of towns from Myanmar distance calculator as weights and described on each road (edges) respectively [7].

Similar algorithm can be done to find the shortest path for minimum time travel and the final result is illustrated in figure 9.

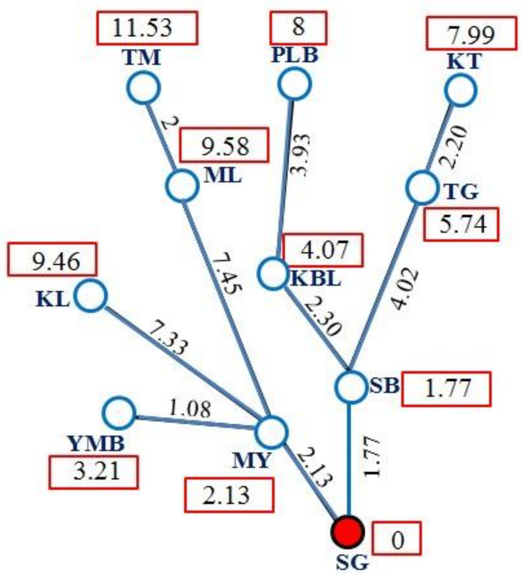


Figure 9. The minimum spanning tree of shortest paths (hour) from Sagaing to other towns in Sagaing region

The values assigned at each town shown in given network with red-boxes, Figure 9, are the minimum time for travelling from Sagaing to respective towns. We can see that the final minimum spanning tree for minimum time is the same as that for shortest distance. But the order of forming permanent town of Katha and Pinlebu is changed because of road qualities between those towns.

Table 1. Optimum results and shortest paths from Sagaing to other destinations in Sagaing region

Start	Destination	Shortest Distance (km)	Minimum Time (hr)	Shortest Paths
SG	MY	114.33	2.13	SG-MY
	SB	93.28	1.77	SG-SB
	YMB	161.15	3.21	SG-MY-YMB
	KBL	198.11	4.07	SG-SB-KBL
	KL	349.73	9.46	SG-MY-KL
	ML	366.76	9.58	SG-MY-ML
	TG	291.88	5.74	SG-SB-TG
	KT	387.46	7.99	SG-SB-TG-KT
	PLB	338.46	8	SG-SB-KBL-PLB
	TM	464.15	11.53	SG-MY-ML-TM

The last column of above table shows the shortest paths for each pair of places on both distance and time. All of the shortest paths for distance and time are the same. This means that the qualities of all roads on those routes are almost the same. The results obtained from research using Dijkstra's shortest paths algorithm have been compared with the data from Google Map and described in the following table 2.

Table 2. Comparison for the result in table 1 with Results from Google map [8]

Start Town	Destination town	Results from research (Dijkstra)		Results from Google Map		Difference	
		Shortest distance (km)	Time (hr)	Distance (km)	Time (hr)	Distance (km)	Time (hr)
SAGAING	MY	114.33	2.13	114	2.08	0.33	0.05
	SB	93.28	1.77	93.3	1.78	-0.02	-0.01
	YMB	161.15	3.21	156	2.98	5.15	0.23
	KBL	198.11	4.07	198	4.01	0.11	0.06
	KL	349.73	9.46	345	9.26	4.73	0.2
	ML	366.76	9.58	362	9.33	4.76	0.25
	TG	291.88	5.74	266	5.68	25.88	0.06
	KT	387.46	7.99	361	7.97	26.46	0.02
	PLB	338.46	8	335	7.67	3.46	0.33
	TM	464.15	11.53	454	11.33	10.15	0.2

The last two columns in the table 2 show the difference between shortest paths and minimum time from algorithm and Google map. According to the difference data from above table, the accuracy of the results from Dijkstra's algorithm is reliable for practical situation.

3.2. Comparison and Recommendation

There are many algorithms to find the shortest paths in various situations but the special features of these algorithms are different. For example, Dijkstra's algorithm and Bellman Ford algorithm are both single-source shortest path algorithms but Dijkstra can only be used to find the shortest paths for positive weights and Bellman Ford is to handle for negative weight and circle in a graph. On the other hand, Floyd Warshall algorithm is for all sources to all destinations. The calculation time of Dijkstra algorithm is $n^2 + m$, while Bellman Ford and Floyd Warshall algorithms are n^3 and nm

respectively where *n* is the number of nodes and *m* is the number of edges in the network [9].

Dijkstra's algorithm is the most efficient one to find the shortest paths. It can be applied to both directed and undirected graphs, and calculation time is considerably faster than other algorithms. For these reasons and above comparison, we strongly recommend that Dijkstra's algorithm is an algorithm to get the best solution for finding shortest paths [6].

4. Conclusion and Future Works

In this work, Dijkstra's Algorithm is used to get the optimal results using actual distances and travel time between any two nodes (towns). Detailed descriptions and step by step procedure of the algorithm has been described with a flowchart and illustrated by determining the shortest paths started from Sagaing to other towns in Sagaing region using actual weighted values between any two towns. The accuracy of the results obtained from the research has been proved by comparing the results from Google Map. For further studies, this algorithm can also be applied to find the optimal results for traffic control, path finding in social networks, computer games, transportation systems, and operations research etc. Moreover, based on the flowchart and detailed calculation procedure, one can create computer codes such as C/C++ or JAVA, running these codes using various weighted values from actual information and data to solve general Dijkstra's shortest path problems.

Acknowledgments

The author would like to acknowledge her thank to Dr. Lin Lin Naing, Professor and Head, Faculty of Computing, University of Computer Studies, Hinthada, Ayeyarwady region and Dr. San San Maw, Professor, Department of Engineering Mathematics, Mandalay Technological University, Mandalay, Myanmar for their encouragement, invaluable comments and perfect supervision throughout my research work.

References

[1] Dijkstra's algorithm (2019), [Online]. Available: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

[2] E. Kreyszig, *Advanced Engineering Mathema - tics*,10th edition. John Wiley & Sons, Inc., 2011, pp.970-990.

[3] P. Sharma, A. Planiya, "Shortest Path Finding of Wireless Optical Network using Dijkstra Algorithm and Analysis of Delay and Blocking Probability", International Journal of Research and Scientific Innovation (IJRSI). Vol. 3, Issue 4, April 2016.

[4] *Solving Shortest Path Problem: Dijkstra's Algorithm*, Lecture Note on Operations Research Methods. October 23, 2009. [Online]. Available: https://www.ifp.illinois.edu/~angelia/ge330fall09_dijkstra_l18.pdf

[5] Myanmar Map [Online]: Available: http://www.un.org/Depts/Cartographic/map/profile/myanm-ar.pdf

[6] S. S. Maw, K. S. Lin, L. L. Naing, "Dijkstra's Algorithm for Effective Travelling to the Most Famous Destinations in Myanmar", International Journal of Mathematics Trends and Technology (IJMTT). Volume 65 Issue 8, August 2019. pp-4-12. [Online]: Available: https://www ijmttjournal.org/ Volume-65/Issue-8/IJMTT-V65I8P502.pdf.

[7] *Myanmar Distance Calculator* [Online] Available: https://distance calculator.globefeed.com/Myanmar_ Distance_ Calculator.asp

[8] Google Map (2019). [Online]: Available: http://maps.google.com

[9] Z. Ali, "Comparison of Dijkstra's Algorithm with other proposed algorithms". International Academic Journal of Science and Engineering, Vol. 3, No. 7, 2016, pp. 53-66.

A Study for Kruskal's MST Algorithm Based on Design and Analysis of Computer Algorithms Courses

Aye Aye Naing, Soe Moe Aye
University of Computer Studies (Pakokku)
aanaing85@gmail.com

Abstract

Many problems in engineering and science can be formulated in terms of undirected or directed graph. To solve these problems, there are many algorithms. These are minimum spanning tree algorithms, transitive closure algorithms, shortest path algorithms, and so on. Among these, finding a minimum spanning tree (MST) of a graph is also a well known problem in graph theory with many practical applications. For example: travelling from one city to another city, designing the electronic circuitry, designing the telecommunication network and so on. In this paper, we focused on AA (Design and Analysis of Computer Algorithm) theory to teach algorithms. This paper discussed Kruskal's Minimum Spanning Tree (MST) algorithm.

KEYWORDS: *graph, spanning tree, MST, vertices, edges*

1. Introduction

Let G= (V, E) be a connected, undirected graph with a cost function mapping edges to real numbers. A spanning tree is an undirected tree that connects all vertices in V. The cost of a spanning tree is just the sum of the cost of its edges. MST of G is a subset of E that forms a spanning tree of G with the least cost [**1**]. MST is one of the well-known classical graph problems which has many critical applications in network organization, VLSI layout and routing, touring problems, partitioning data points into clusters and various other fields. It was in 1926 that **Boruvka [2]** produced the first fully algorithm to find the MST. At later time, **Kruskal** and **Prim** developed the two most commonly used MST algorithms, Kruskal's algorithm [**3**] and Prim's algorithm [**4**], respectively. Kruskal's algorithm works on both connected and disconnected graph while Prim's algorithm can work only on the connected graph.

In this paper, Kruskal's MST algorithm is discussed and the main core work is presenting its implementation in C++ language and its time complexity. Kruskal's algorithm is one of the most known algorithms that address the MSF problems. The strictly ordered examination of the graph's edges in order to decide whether they are part of the MSF or not, prohibits the usage of well known parallel strategies, like data partitioning.

The rest of the paper is organized as follows: section 2 looks at the works relating to Kruskal's algorithm. Section 3 explains about how Kruskal's algorithm works with example. Section 4 presents the implementation with the source code and explains the time complexity. Section 5 concludes the paper and gives its future work.

2. Related Works

Prim's algorithm starts with a tree that has only one edge, the minimum weight edge [**5**]. The edges (*j*, *q*) is added one by one such that node *j* is already included, node *q* is not included and weight *wt*(*j*, *q*) is the minimum amongst all the edges (*x*, *y*) for which *x* is in the tree and *y* is not. In order to execute this algorithm efficiently, we have a node index *near*(*j*) associated with each node *j* that is not yet included in the tree. If a node is included in the tree, *near*(*j*) = 0. The node *near*(*j*) is selected into the tree such that *wt*(*j*, *near*(*j*)) in the minimum amongst all possible choices for *near*(*j*).

In [**6**], **Munier et al.** presented shared memory-based parallel implementations of Kruskal's and Prim's algorithms. To program the shared memory parallel machines, they used serial code with compiler directives method. They did the analysis the parallel MST programs executed using classical multi threaded and openMP-based execution models. The experiment showed that their proposed parallel algorithms achieve better performance than serial ones.

[**7**] proposed a simple modification of Kruskal's algorithm that avoids sorting edges that are obviously not in MST. This algorithm runs in time $(m + nlognlog \frac{m}{n}$) for arbitrary graphs with random edge weights. The experiment showed that this proposed algorithm outperforms the original Kruskal's algorithm when the number of edges is increased.

V. Loncˇar et al. [**8**] proposed parallel variants of Kruskal's and Prim's algorithm and made massage passing parallel machine with distributed memory. They considered the large graphs that can not fit into memory of one process. The experimental result showed that Prim's algorithm is a good choice for dense graphs while Kruskal's algorithm is better for sparse ones. Poor scalability of Prim's algorithm comes from its high communication cost while Kruskal's algorithm showed much better scaling to larger number of processes.

In [**9**], **A. Katsigiannis et al.** considered helper threading scheme used to parallelize efficiently Kruskal's Minimum Spanning Forest algorithm. First of all, this scheme employs a main thread that executes the regular, sequential Kruskal's algorithm and at each iteration, examines the edge with the next minimum weight. At the same time, a number of helper threads run concurrently with the main one and examine edges of bigger weight, checking whether they create a cycle if added to current MSF. Whenever a cycle is discovered, the corresponding edge is marked as discarded. As these edges have been safely excluded from the MSF, the main thread needs to check only the edges that weren't rejected by the helper threads, thus performing less work compared to the sequential implementation. The more cycles found by the helper threads, the more offloading will be accomplished for the main thread.

3. Minimum Spanning Tree

Given a connected and undirected graph, a *spanning tree* of that 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 *minimum spanning tree (MST)* or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree. A minimum spanning tree is a special kind of tree that minimizes the lengths (or "weights") of the edges of the tree. This paper discusses Kruskal's Algorithm and Prim's Algorithm. Both are greedy algorithm to find the MST.

3.1 Kruskal's MST Algorithm

Kruskal's Algorithm builds the spanning tree by adding edges one by one into a growing spanning tree. Kruskal's algorithm follows greedy approach as in each iteration it finds an edge which has least weight, and adds it to the growing spanning tree. Below are the steps for finding MST using Kruskal's algorithm:

- Sort all edges with respect to their weights
- Pick the smallest edge. Check whether it forms a cycle with the spanning tree formed or not. If cycle is not formed, include this edge. Else, discard it.
- Repeat step 2 until there is no edge in the sorted list.

The step 2 uses Union-Find algorithm to detect cycle.

3.1.1 Union-Find Algorithm

Union-Find algorithm performs on disjoint-set data structure. A disjoint-set data structure is a data structure that keeps track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets [10]. Union-Find algorithm has two operations:

- Find: Determine which subset a particular element is in. This can be used for determining if two elements are in the same subset.
- Union: Join two subsets into a single subset.

3.2 Explanation with Example for of Kruskal's MST algorithm

Now, the explanation of how MST algorithm works is given with the example. Figure 1 shows the step by step procedures of Kruskal's MST algorithm.

begin

1. ***T ← Ø;***
2. ***VS ← Ø;***
3. ***construct a priority queue Q containing all edges in E;***
4. ***for each vertex v ∈ V do add {v} to VS;***
5. ***while || VS|| >1 do***
   begin
6. ***choose (v,w), an edge in Q of lowest cost;***
7. ***delete (v,w) from Q;***
8. ***if v and w are in different sets*** W_1 ***and*** W_2 ***in VS then***
   begin
9. ***replace*** W_1 ***and*** W_2 ***in VS by*** $W_1 \cup W_2$***;***
10. ***add (v,w) to T***
    end
    end

end

Figure 1.Minimum-cost Spanning Tree Algorithm

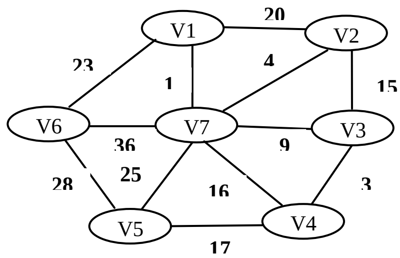
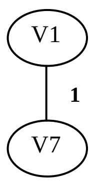


Figure 2.An Undirected Graph with Cost on Edge

Figure 2 is considered as undirected graph input. According to step 3 of MST algorithm, the sorted list of edges is generated as shown in Table 1.

The followings steps are demonstration of from step 4 to step 10 of the algorithm. When all of the vertices have been added to the tree, the minimum spanning tree is finally outputted with cost 57 as shown in Figure 3.

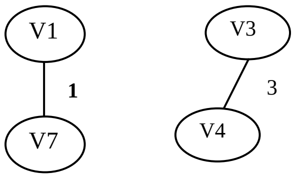
1. Pick the edge v1-v7. No cycle is formed. Include it



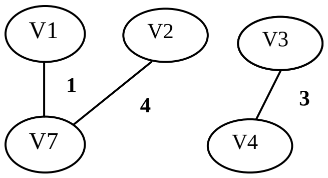
Edge	Cost
V1,v7	1
V3, v4	3
V2,v7	4
V3,v7	9
V2,v3	15
V4,v7	16
V4,v5	17
V1,v2	20
V1,v6	23
V5,v7	25
V5,v6	28
V6,v7	36

Table 1. Priority Queue List (Sorted List) of Edges

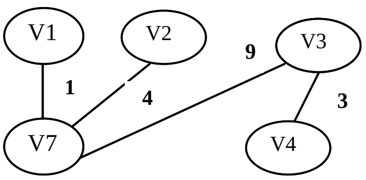
2. Pick the edge v3-v4. No cycle is formed. Include it.



3. Pick the edge v2-v7. No cycle is formed. Include it.



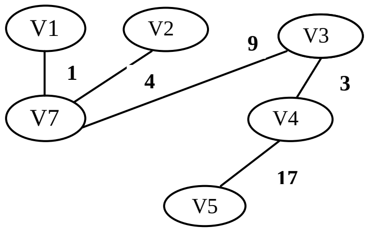
4. Pick the edge v3-v7. No cycle is formed. Include it.



5. Pick the edge v2-v3. This edge results in cycle. Discard it.

6. Pick the edge v4-v7. This edge also results in cycle. Discard it.

7. Pick the edge v4-v5. No cycle is formed. Include it.



8. Pick the edge v1-v2. This edge results in cycle. Discard it.

9. Pick the edge v1-v6. No cycle is formed. Include it.

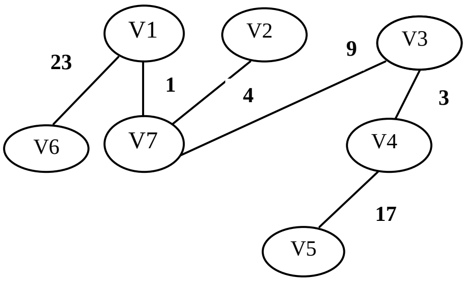


Figure 3. Minimum-cost Spanning Tree

3.3. Prim's Algorithm

The Prim's Algorithm is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far. Let us understand it with an example: Consider the below input graph.

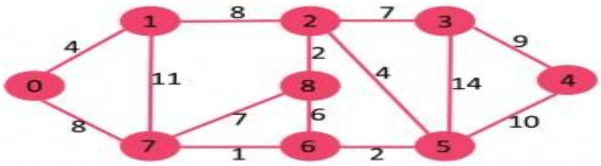
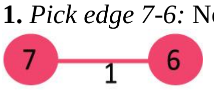


Figure 4: Undirected Graph with 9 Vertices and 14 Edges

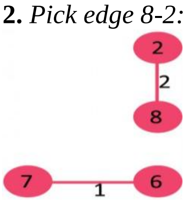
The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

Now pick all edges one by one from sorted list of edges.

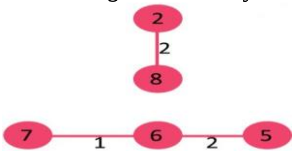
1. *Pick edge 7-6:* No cycle is formed, include it.



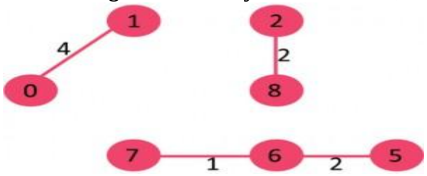
2. *Pick edge 8-2:* No cycle is formed, include it.



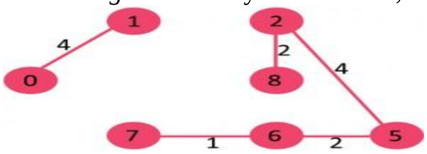
3. *Pick edge 6-5:* No cycle is formed, include it.



4. *Pick edge 0-1:* No cycle is formed, include it.

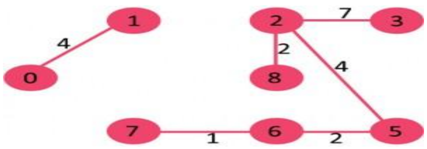


5. *Pick edge 2-5:* No cycle is formed, include it.



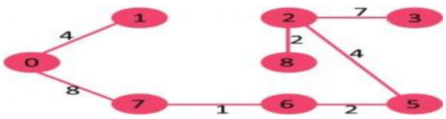
6. *Pick edge 8-6:* Since including this edge results in cycle, discard it.

7. *Pick edge 2-3:* No cycle is formed, include it.



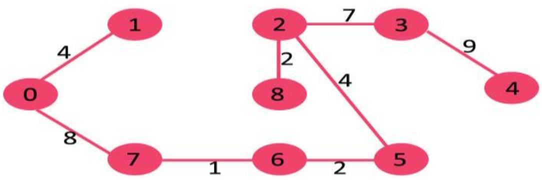
8. *Pick edge 7-8:* Since including this edge results in cycle, discard it.

9. *Pick edge 0-7:* No cycle is formed, include it.



10. *Pick edge 1-2:* Since including this edge results in cycle, discard it.

11. *Pick edge 3-4:* No cycle is formed, include it.



Since the number of edges included equals (V – 1), the algorithm stops here.

4. Example Case Study for Traveling Salesperson Problem

A less obvious application is that the minimum spanning tree can be used to approximately solve the traveling salesman problem. A convenient formal way of defining this problem is to find the shortest path that visits each point at least once. Note that if you have a path visiting all points exactly once, it's a special kind of tree. For instance, twelve of sixteen spanning trees are actually paths. If you have a path visiting some vertices more than once, you can always drop some edges to get a tree. So in general the MST weight is less than the TSP weight, because it's a minimization over a strictly larger set.

On the other hand, if you draw a path tracing around the minimum spanning tree, you trace each edge twice and visit all points, so the TSP weight is less than twice the MST weight. Therefore this tour is within a factor of two of optimal.

5. Implementation and Analysis

In this section, implementation for Kruskal's MST algorithm is presented and its time complexity is also considered in big O notation. For the implementation, C++ language with GCC 32 bit release compiler is used.

5.1Implementation Interface

Input : Undirected graph in Figure 2
Method: Pseudo Code in Figure 1
Output: Minimum spanning tree with cost 57 as shown in Figure 5.

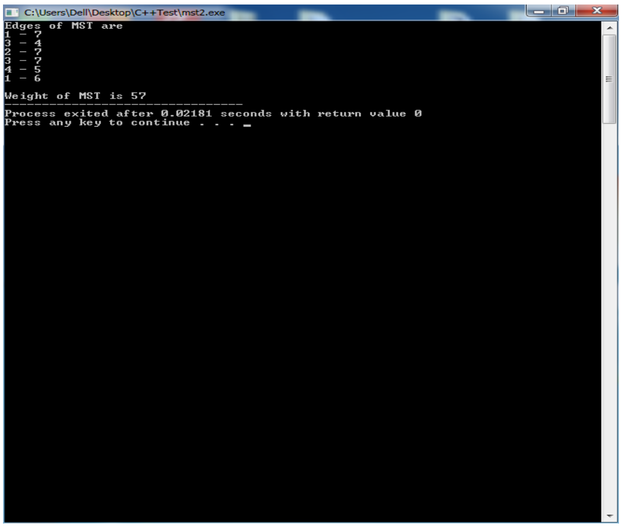


Figure 5: Minimum Spanning Tree with Cost 57 Displayed in Console Mode

For this implementation, the node names of the input graph are redefined as follows:
v1⇒1, v2⇒2, v3⇒3, v4⇒4, v5⇒5, v6⇒6, v7⇒7

When the next undirected graph in Figure 7 is considered as input, the minimum spanning tree with cost 37 is generated as shown in Figure 6.

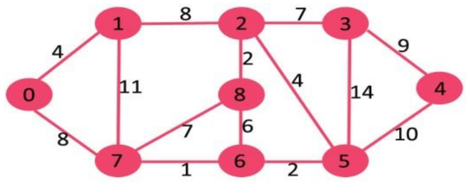


Figure6: Undirected Graph with 9 Vertices and 14 Edges

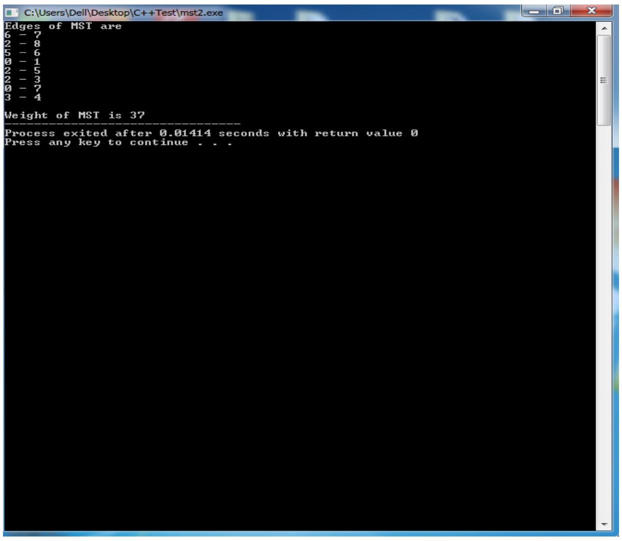


Figure 7: Minimum Spanning Tree with Cost 37 and 8 Edges

5.2 Difference in Kruskal's algorithm and Prim's algorithm

Both are greedy algorithm to find the MST. However, let me show the difference with the help of table:

Prims Algorithm	Kruskal Algorithm
It start to build the MST from any of the Node.	It start to build the MST from Minimum weighted vertex in the graph.
Adjencary Matrix , Binary Heap or Fibonacci Heap is used in Prims algorithm	Disjoint Set is used in Kruskal Algorithm.
Prims Algorithm run faster in dense graphs	Kruskal Algorithm run faster in sparse graphs
Time Complexity is O(EV log V) with binay heap and O(E+V log V) with fibonacci heap.	Time Complexity is **O(E log V)**
The next Node included must be connected with the node we traverse	The next edge include may or may not be connected but should not form the cycle.
It traverse the one node saveral time in order to get it minimum distance	It travese the edge only once and based on cycle it will either reject it or accept it,
Greedy Algorithm	Greedy Algorithm

5.3 Time Complexity and Experimental Run Time

For E edges and V vertices,

Sorting the edges : $O(ELogE)$

Find operations : $O(2E)$ for at most $2E$ find operations

Union operation : O(VlogV) for at most $V-1$ union operations

Thus, total run time = $O(ELogE) + O(2E) +$ O(VlogV)

$= O(ElogE)$ since the maximum value E can be V^2.

According to my experiment, the running time is 0.1 sec for 9 vertices and 14 edges implemented by using C++ language with GCC 32 bit release compiler, under Intel(R) Core(TM) i3CPU@2.10GHz and RAM-4.00GB.

The following is the source code for MST algorithm implementation.

```
#include<bits/stdc++.h>
using namespace std;
typedef  pair<int, int> iPair;
struct Graph
{
   int V, E;
   vector< pair<int, iPair> > edges;
   Graph(int V, int E)
   {
      this->V = V;
      this->E = E;
   }
   void addEdge(int u, int v, int w)
   {
      edges.push_back({w, {u, v}});
   }
    int kruskalMST();
};
struct DisjointSets
{
   int *parent, *rnk;
   int n;

   DisjointSets(int n)
   {
      this->n = n;
      parent = new int[n+1];
      rnk = new int[n+1];
      for (int i = 0; i <= n; i++)
      {
         rnk[i] = 0;
         parent[i] = i;
      }
   }
   int find(int u)
```

```
    {
        if (u != parent[u])
            parent[u] = find(parent[u]);
        return parent[u];
    }
     void merge(int x, int y)
    {
        x = find(x), y = find(y);
        if (rnk[x] > rnk[y])
            parent[y] = x;
        else
            parent[x] = y;

        if (rnk[x] == rnk[y])
            rnk[y]++;
    }
};
int Graph::kruskalMST()
{
    int mst_wt = 0;
    sort(edges.begin(), edges.end());
    DisjointSets ds(V);
    vector< pair<int, iPair> >::iterator it;
    for (it=edges.begin(); it!=edges.end(); it++)
    {
        int u = it->second.first;
        int v = it->second.second;
        int set_u = ds.find(u);
        int set_v = ds.find(v);
        if (set_u != set_v)
        { cout << u << " - " << v << endl;
        mst_wt += it->first;
        ds.merge(set_u, set_v);
         }
    }

    return mst_wt;
}
int main()
{int V = 9, E = 14;
    Graph g(V, E);
g.addEdge(0, 1, 4);
    g.addEdge(1, 2, 8);
    g.addEdge(2,3, 7);
    g.addEdge(3, 4, 9);
  g.addEdge(4, 5, 10);
   g.addEdge(5, 6, 2);
    g.addEdge(6, 7, 1);
   g.addEdge(0, 7, 8);
    g.addEdge(1, 7, 11);
    g.addEdge(7, 8, 7);
    g.addEdge(2, 8, 2);
   g.addEdge(2, 5, 4);
   g.addEdge(3, 5, 14);
    g.addEdge(6, 8, 6);
    cout << "Edges of MST are \n";
    int mst_wt = g.kruskalMST();

    cout << "\nWeight of MST is " << mst_wt;

    return 0;
}
```

6. Conclusions

By using Kruskal's MST algorithm, the shortest path of a graph can be found. To travel from city A to City B, for example, the shortest path between these cities can be achieved by using this algorithm. However, in order to find the shortest path of a graph by hand, it is time consuming and this work is so tedious. This paper explains about Kruskal's MST algorithm in detail and implements it by using a programming language. According to the experiment, the resulted minimum spanning tree of the undirected graph that has 9 vertices and 14 edges is generated by taking about 0.1 seconds. In addition to the work of this paper, its time complexity and analysis is also considered in order to lead to the research work.

As the future work, I want to perform the analysis this algorithm, not in serially, but in parallel for distributed memory architecture.

Acknowledgment

I would like to express my gratitude and my heartfelt thanks to UJRI (2019) for submitting this paper. I am also grateful to all my teachers from the University of Computer Studies (Pakokku) who has taught and guided as during the period of study for this paper. Finally, I also thank all of my friends and colleagues for supporting me in various ways.

References

[1] Alfred V. Aho, John E. Hopcroft, Jeffery D. Ullman, "The Design and Analysis of Computer Algorithms", Addison-Wesley publishing, pp. 172-176.

[2] Jaroslav, Helena, "The Origins of Minimum Spanning Tree Algorithms", Mathematical Subject Classification, 2010, pp. 127-141.

[3] Kruskal, J.B., " On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem", Proceedings of the American Mathematical Society 7(1), February 1956, pp. 48-50.

[4] Prim, R.C. "Shortest connection networks and some generalizations", Bell System Technology Journal, vol. 36, Issue 6, 1957, pp. 1389-1401.

[5] S.K. Sathua, M.R. Kabat, R. Mohanty, "Lecture Notes on Design and Analysis of Algorithms", VEER SURENDRA SAI UNIVERSITY OF TECHNOLOGY, India.

[6] B. Munier, M. Aleem, M. A. Islam, M. A Iqbal, "A Fast Implementation of Minimum Spanning Tree Method and Applying it to Krusal's and Prim's Algorithms", Vol. 1, No. 1, 2017 Sukkur IBA, pp. 57-66.

[7] V. OSipov, P. Sanders, and J. Singler, "The Filter-Kruskal Minimum Spanning Tree Algorithm", Proceeding of the Workshop on Algorithm Engineering and Experiments, ALENEX 2009, New York, New York, USA, January 3, 2009.

[8] V. Loncar, S. Škrbic´, and A. Balazˇ, "Parallelization of Minimum Spanning Tree

Algorithms Using Distributed Memory Architectures”, book chapter from Transactions on Engineering Technologies: Special Volume of the World Congress on Engineering 2013, pp. 543-554.
[9] A. Katsigiannis, N. Anastopoulos, K. Nikas, and N. Koziris, “An approach to parallelize Kruskal’s algorithm using Helper Threads”, proceeding of 2012 IEEE 26th International Conference on Parallel and Distributed Processing Symposium &Ph D Forum (IPDPSW).
[10] https://www.geeksforgeeks.org/union-find/

A Spanning Tree with Minimum Weight of the One City and Six Towns in Mandalay Region

Mon Yee Aye
Department of Engineering Mathematics
Mandalay Technological University
Mandalay, Myanmar.
cherry.monyi@gmail.com

Abstract

One of the possibilities when modelling a transport network is to use a graph with vertices and edges. They represent the nodes and arcs of such a network respectively. This paper proposes a novel network-reduction techniques, based on a network-flow procedure, which is referred to as Minimum Spanning Tree (MST) with additional capabilities. In networking, we use minimum spanning tree algorithm often. So the problem is as stated here, given a graph with weighted edges, find a tree of edges with the minimum total weight that satisfies these three properties: connected, acyclic and consisting of | V | – 1 edges. In this paper, we present the minimum weight of a spanning tree using Prim's Algorithm. Firstly, the basic definitions and notations in graph theory are introduced. Then some properties of the tree are expressed. Next the concept of Prim's Algorithm is described. Finally, a minimum weight spanning tree of the one city and six towns in Mandalay Region is observed.

Keywords*— *Weighted graph, MST, Prim's Algorithm, Adjacency matrix.

1. Introduction

In the study of graph theory, the problem of finding a minimum spanning tree is interesting and difficult. In a number of literature the problem of finding the best spanning tree has been studied, for example, spanning tree with minimum diameter, minimum cost (weight) spanning tree or the minimum degree spanning tree. We consider the minimum weight spanning tree (MST) of a weighted graph; that is, those graphs where weights are preserved in every connected graph. A MST of a graph G is a spanning tree with minimum weight. In this paper, first we discuss some structured properties of MST. Then we present an algorithm to find a MST of a weighted graph.

2. Definitions and Notations

A **graph** G = (V(G), E(G)) or G = (V, E) consists of two finite sets, V(G) or V, the vertex set of the graph, which is a non-empty set of elements called **vertices** and E(G) or E, the edge set of the graph, which is a possibly empty set of elements called **edges**, such that each edge e in E is assigned as an unordered pair of vertices (u,v), called the **end vertices** of e [1]. Two nonparallel edges are said to be **adjacent** if they are incident on a common vertex. A **walk** in a graph G is a finite sequence whose terms are alternately vertices and edges. In a walk, there may be repetition of vertices and edges. If the edges $e_1,e_2,\ldots,e_k$ of the walk $W \equiv v_o\ e_1\ v_1\ e_2\ v_2 \ldots e_k v_k$ are distinct then W is called a **trail**. A trail is a walk in which no edge is repeated. A nontrivial closed trail in a graph G is called a **cycle** if its orgin and internal vertices are distinct [2]. A graph with no cycle is an **acyclic graph**. A **tree** is a connected acyclic graph. If the vertices v_o, $v_1, \ldots, v_k$ of the walk $W \equiv v_o\, e_1\, v_1\, e_2\, v_2 \ldots e_k v_k$ are distinct then W is called a **path**. A graph G is called **connected** if every two of its vertices are connected. A graph will real number on the edges is called a **weighted graph** [3].

2.1. Some Properties of Trees

2.1.1 Theorem

Every pair of vertices in a tree is connected by one and only one path.

2.1.2 Theorem

If there is one and only one path between every pair of vertices in a graph G, then G is a tree.

2.1.3 Theorem

A tree with n number of vertices has n – 1 number of edges.

2.1.4 Theorem

A connected graph with n vertices and n – 1 edges is a tree.

2.1.5 Theorem

A graph G has a spanning tree if and only if G is connected [4].

2.2. Definitions

A tree T is called a **spanning tree** of a connected graph G if T is a subgraph of G and if T contains all the vertices of G. A **minimum spanning tree** (MST) or **Minimum weight spanning tree** is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weight is as small as possible. The **adjacency matrix** of graph G with n vertices and no parallel edge is an n by n symmetric binary matrix $X = (x_{ij})_{n \times n}$ defined over the ring of integers such that $x_{ij} = 1$, if there is an edge between i^{th} and j^{th} vertices = 0, if there is no edge between them [5].

3. Applications of Minimum Weight or Cost Spanning Tree (MST)

The standard application is to a problem like phone network design. We have a business with several offices; we want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. We want a set of lines that connects all offices with a minimum total cost. There are quite a few use cases for minimum spanning trees. One example would be a telecommunications company trying to lay cable in a new neighbourhood. If it is constrained to bury the cable only along certain paths, then there would be a graph containing the houses connected by those paths. Some of the paths might be more expensive, because they are longer, or require the cable to be buried deeper; these paths would be represented by edges with larger weights. A spanning tree for that graph would be a subset of those paths that has no cycles but still connects every house; there might be several spanning trees possible. A minimum spanning tree would be one with the lowest total cost, representing the least expensive path for laying the cable ([5], [6]).

We now present Prim's algorithm for finding a minimal spanning tree for a connected weighted graph where no weight is negative.

3.1. Prim's Algorithm

Prim's Algorithm also use Greedy approach to find the minimum spanning tree. In Prim's Algorithm, we grow the spanning tree from a starting position.

Let T be a tree in a connected weighted graph G represented by two sets: the set vertices in T and set of edges in T.

Step 1 : We start with a vertex v_0 (say) in G and no edge such that $T = \{\{v_0\}, \phi\}$.

Step 2 : We find the edge $e_1 = (v_0,v_1)$ in G such that the end vertex v_0 is in T and its weight is minimum, i.e., $w(e_1)$ is minimum. Adjoin v_1 and e_1 to T, i.e., $T = \{\{v_0,v_1\}, e_1\}$.

Step 3 : We choose the next edge $e_{ij} = (v_i,v_j)$ in such a way that end vertex v_i is in T and end vertex v_j is not in T and weight of e_{ij} is as small as possible. Adjoin v_j and e_{ij} to T.

Sept 4 : We repeat step 3 until T contains all the vertices of G. The set T will give minimal spanning tree of G. [7]

Now we apply to Prim's Algorithm, we can find a minimum spanning tree connecting the one city and six towns in Mandalay Region. We collect the distance between two towns from Google map and Ministry of Construction in Mandalay Region. Firstly, we show the distances, in kilometers, between one city and six towns in Mandalay Region. Then we get the connected graph G for the one city and six towns. Next , we show the adjacency matrix X(G) for connecting in one city and six towns. Finally , we find a minimum spanning tree connecting the one city and six towns.

We denote Mandalay by A, Madayar by B, Patheingyi by C, Sintgaing by D, Kyaukse by E, Amarapura by F and Sintgu by G.

Table 1. The distances, in kilometers, between one city and six downs

	A	B	C	D	E	F	G
A	-	40.4	13.7	31.0	45.4	7.0	92.8
B	40.4	-	41.4	67.2	81.6	45.4	54.2
C	13.7	41.4	-	39.7	53.6	20.7	93.8
D	31.0	67.2	39.7	-	14.8	26.8	119.6
E	45.4	81.6	53.6	14.8	-	40.7	134.0
F	7.0	45.5	20.7	26.8	40.7	-	97.8
G	92.8	54.2	93.8	119.6	134.0	97.8	-

Then the connected graph G follows.

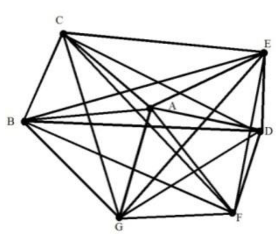


Figure 1. The connected graph G for one city and six towns

The adjacency matrix X (G) is

	A	B	C	D	E	F	G
A	0	40.4	13.7	31.0	45.4	7.0	92.8
B	40.4	0	41.4	67.2	81.6	45.4	54.2
C	13.7	41.4	0	39.7	53.6	20.7	93.8
D	31.0	67.2	39.7	0	14.8	26.8	119.6
E	45.4	81.6	53.6	14.8	0	40.7	134.0
F	7.0	45.5	20.7	26.8	40.7	0	97.8
G	92.8	54.2	93.8	119.6	134.0	97.8	0

Initially, we start with a vertex A in G and no edge such that $T = \{\{A\}, \phi\}$. After choosing the root vertex A, (A,B), (A,C), (A,D), (A,E), (A,F) and (A,G) are six edges with 40.4, 13.7, 31.0, 45.4, 7.0 and 92.8, respectively. We choose the edge (A,F) as it is lesser than the other.

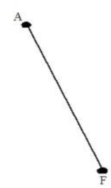


Figure 2. Corresponding tree T

Then we have T ={{A,F},{(A,F)}}. Now the tree A-7-F is treated as one vertex and we check for all edges going out from it. We select the one which has the smallest distance and include it in the tree. We choose the edge (A,C) as it is lesser than the other and the minimum weight is 13.7 km.

Figure 3. Corresponding tree T

Then we have T = {{A,F,C}, {(A,F), (A,C)}}. After this step, A-F-A-C tree is formed. Now we'll again treat it as a vertex and will check all the edges again. However, we will choose only the least distance edge. In this case, (A,D) is the new edge with minimum weight is 31 km, which is lesser than other edges' distance 40.4, 45.4 and 92.8 km.

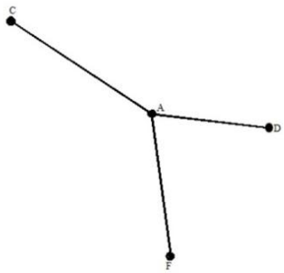


Figure 4. Corresponding tree T

Then we have T = {{A, F, C, D}, {(A,F). (A, C), (A, D)} }. After adding vertex D to the spanning tree, we'll treat it as a vertex and will check all the edges again. In this case, (D,E) is the new edge and the minimum weight is 14.8 km.

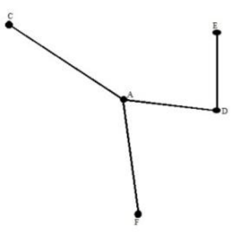


Figure 5. Corresponding tree T

Then we have T = { {A,F,C,D,E}, {(A,F), (A,C), (A,D), (D,E)}}. After this step, A-F-A-C-A-D-E tree is formed. Now we'll treat it as a vertex and will check all the edges. We will choose only the least distance edge. In this case, (A,B) is the new edge with distance 40.4 km.

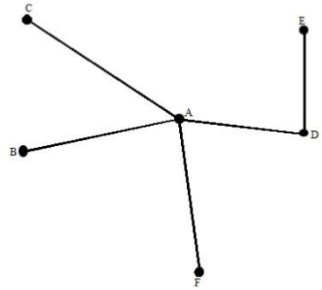


Figure 6. Corresponding tree T.

Then we have T = { {A,F,C,D,E,B} , {(A,F), (A,C), (A,D), (D,E), (A,B)}}. After adding vertex B to the spanning tree, we now have one edge (B,G). Then, we can add vertex G. (B,G) is the new edge with distance 54.2 km.

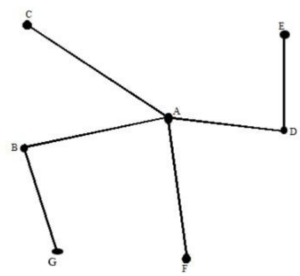


Figure 7. Corresponding tree T

Finally we have T = { {A,F,C,D,E,B,G}, {(A,F),(A,C),(A,D,(D,E),(A,B), (B,G)} }.

Total weight = 7 +13.7 + 31 + 14.8+ 40.4 + 54.2 = 161.1 kilometers which is the minimum weight of the spanning tree appears in Figure 7.

4. Conclusion

Spanning tree find applications in many field, including computer network, calling trees and organization charts. The minimum weight spanning tree can be used to approximately solve the travelling saleman problem. For further studies, one can observe the minimum weight spanning trees for all towns in Mandalay Region with the aids of computer codes such as C/C^{++}, MATLAB or JAVA language running these codes by using various weight values from actual information and data to solve Prim's Algorithm. MST have direct applications in the design of networks, including computer networks, telecommunications networks, transportation networks, water supply networks , and electrical grids.

References

[1] Bondy, J.A and Murty, U.S.R, "Graph Theory with Applications", Macmillan Press Ltd., London, 1976.

[2] Chin, F., Houck, D., "Algorithms for updating minimal spanning trees", Journal of Computer and System Sciences, 16(3):333-344, doi:10.1016/0022-0000(78) 90022-3, 1978.

[3] Gross, J. and Yellon, J., "Graph Theory and Its Applications", CRC Press Company, London , 1999.

[4] Harray, F., "Graph Theory", Narosa Publishing House, New Delhi, 2000.

[5] Parthasarathy, K. R., "Basic Graph Theory", Tata McGraw-Hill Publishing Company Limited, New Delhi, 1994.

[6] Rosen, K.H., "Discrete Mathematics and Its Application", McGraw-Hill, New York , 2012.

[7] Saha Ray, S., " Graph Theory with Algorithms and Its Applications", DOI : 10.1007/978-81-322-0750-4,© Springer Verlag, India, 2013.

University of Computer Studies (Pakokku)
Department of Higher Education
Ministry of Education
Myanmar