International Journal of Computer Architecture and Mobility
(ISSN 2319-9229) Volume 1-Issue 10, August 2013

An Algorithmic Approach to Graph Theory

Neetu Rawat
nrwt12345@gmail.com, *Assistant Professor, Chameli Devi Group of Institutions, Indore. India.*

***Abstract*— This paper compares two different minimum spanning tree algorithms, namely Prim's and Kruskal's algorithm. It introduces the relevant graph theory concepts that are needed and the working principles behind the algorithms are explained briefly. Next an experiment is conducted in which the order of the starting graph is changed and each algorithm's runtime is then measured. As a result it is possible to draw an order versus time graph. It becomes apparent that the algorithms follow their established time complexities fairly well and that out of the two, Prim's algorithm is the slowest, while Kruskal's is very fast.**

***Keywords*— Spanning tree, minimum spanning tree, Prims algorithm, Kruskal algorithm, time complexity.**

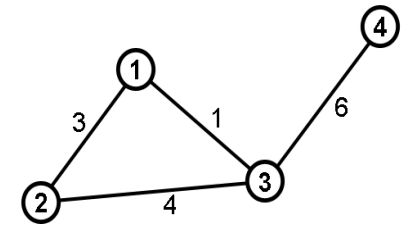
I. INTRODUCTION

Graph theory is an important field of mathematics that has been studied carefully ever since its inception, which is commonly regarded as the 18th century. It was Euler who famously introduced his historical problem of "Seven Bridges of Königsberg", in which the goal is to cross every bridge in the town only once. In his paper he came to the conclusion that no solution exists and this consequently laid the foundations to the study of graphs. The subject has spawned many other fascinating problems nearly all of which are rather concrete in their nature and quite easily comprehensible, but the different algorithms for solving them pose an interesting field of study offering a source for analysis and research. These examples include the travelling salesman, the four colours problem and the minimum spanning tree question, which will be the topic of research in this paper. The algorithms that were made to search for the minimum spanning tree were already developed before the introduction of computers. In fact the first minimum spanning tree algorithm was hypothesised by a Czech mathematician named Otakar Boruvka in his paper published in 1926 [1]. His purpose was to connect the Moravian electric grid as efficiently as possible. Kruskal proved that the shortest spanning tree of the graph is unique [2]. Later prim's has interconnected a given set of terminals with a shortest possible network of direct links [3]. Today there are a number of these algorithms and new ones are being worked on as well in the hope of finding more efficient solutions. Graph theory has been applied in various areas of studies, such as dealing with electronic circuitry, links between entities, such as airline paths and also quite importantly in computer science.

II. DEFINITIONS AND TERMS

A graph is defined as a set of points, or more formally called vertices, some or all of which are connected with a set of lines, or edges. A graph G is denoted as follows:
$G = \{V, E\}$ where V is the set of vertices and E is the set of edges. Often vertices are also called nodes.

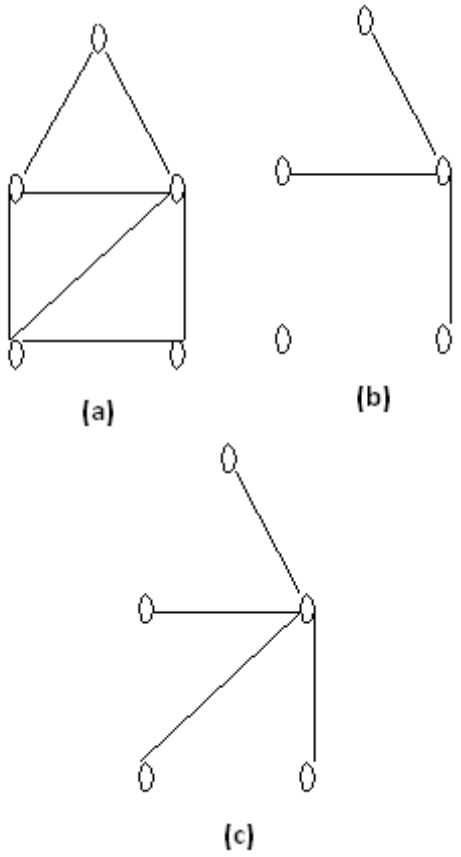
The **adjacency matrix** is an $n \times n$ square matrix, in which each element represents a connection between two vertices. If no such connection exists the element is left empty. In our case, the graph is undirected, which implies that there is a connection between from vertex u to v and vice versa. This means that the entries aij and aji are equal and so the matrix becomes symmetrical.



From \ *To*	1	2	3	4
1	x	**3**	**1**	x
2	**3**	x	**4**	x
3	**1**	**4**	x	**6**
4	x	x	**6**	x

A simple graph with its adjacency matrix

Spanning tree: A spanning tree of a connected graph is a spanning subgraph of the graph which is a tree.
For example, figure (b) shows a tree and figure (c) shows a spanning tree of the graph in figure (a).



Minimal spanning tree: A minimal spanning tree is a spanning tree with minimum weight. More specifically we say that in a weighted graph G, a spanning tree T is a minimal spanning tree if and only if there exists no other spanning tree at a distance one from T whose weight is smaller than that of T.

III. A CLOSER LOOK AT THE ALGORITHMS

In this section of the essay the algorithms will be explained in greater detail and in addition the ways of performing the tasks related to graph theory using computer language will be addressed. Also the algorithms' time complexities will be presented.

Kruskal's algorithm

We adopt the following steps for finding minimum spanning tree using Kruskal's algorithm:

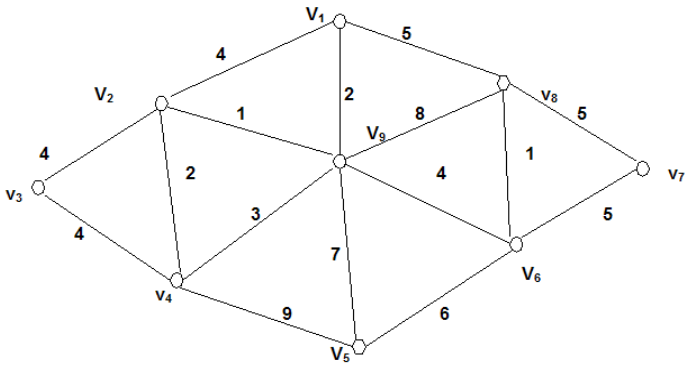
Step1: List all the edges of graph G in the increasing order of weights.

Step2: Select the smallest edge (having minimum weight) from the list and add it to the spanning tree, (which is initially empty), if the inclusion of the edge doesn't make a circuit. If the selected edge makes a circuit, then remove it from the list.

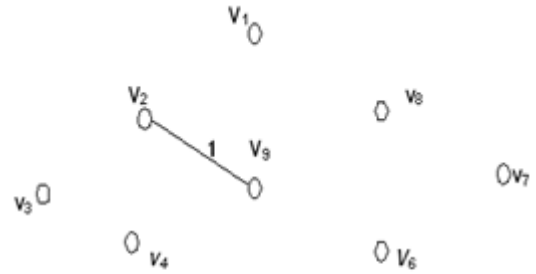
Step3: Repeat steps 1 and 2 until the tree contains v-1 edges(where v is the number of vertices) or the list is empty.

Step4: Now if the tree contains less than v-1 edges and the list is happened to be empty then no spanning tree is possible, else it gives the minimum spanning tree.

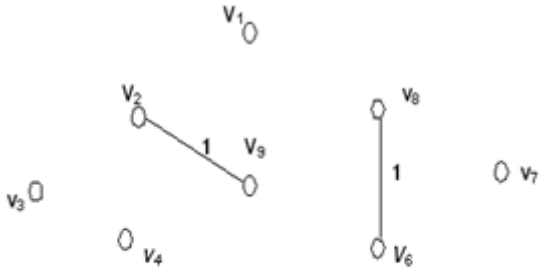
For example:



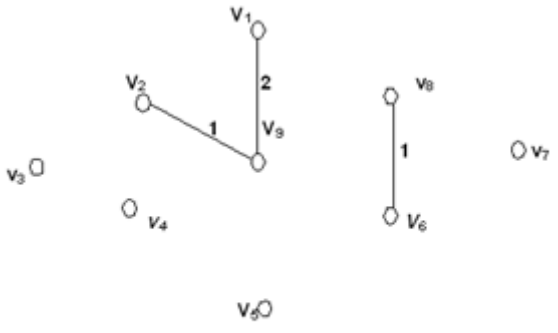
Graph G



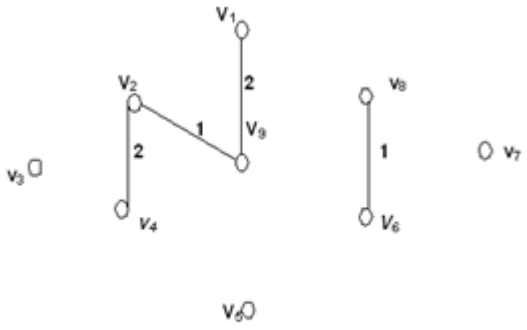
step1 : edge(V2,V9)



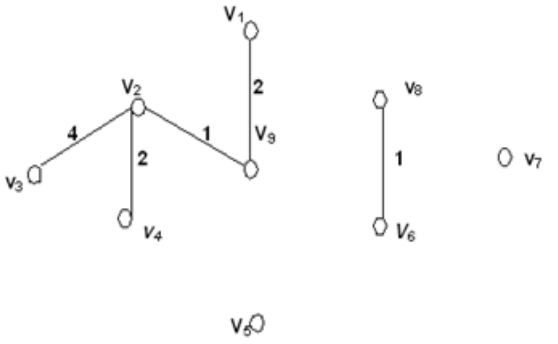
step2 : edge(V6, V8)



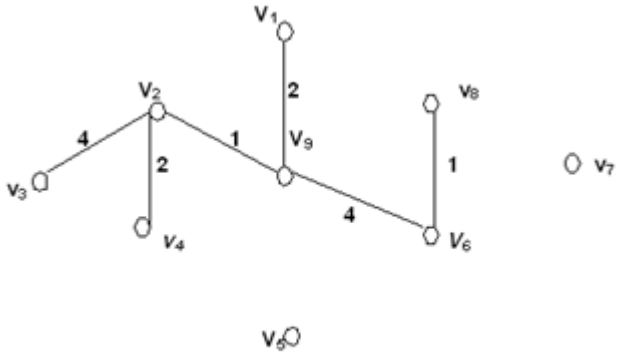
step3: edge(V_1, V_9)



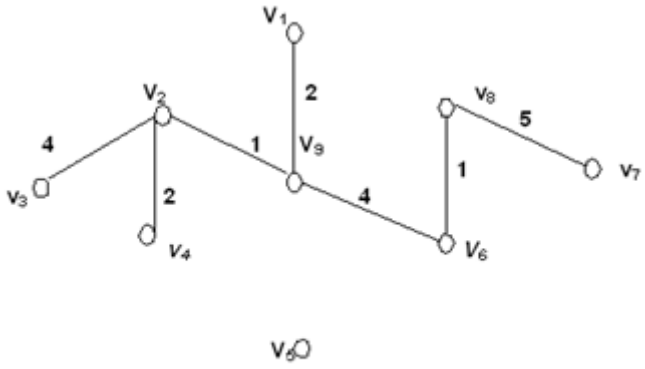
step4: edge(V_2, V_4)



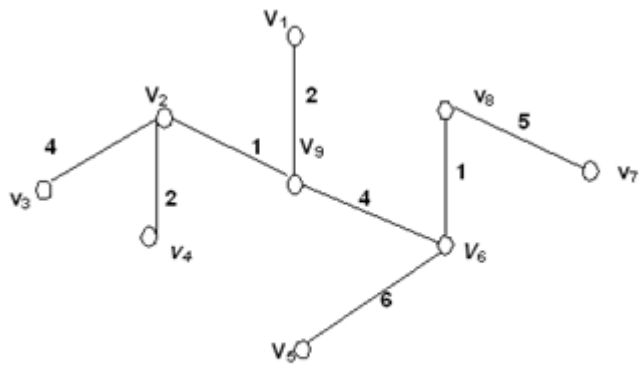
step5: edge(V_2, V_3)



step6: edge(V_6, V_9)



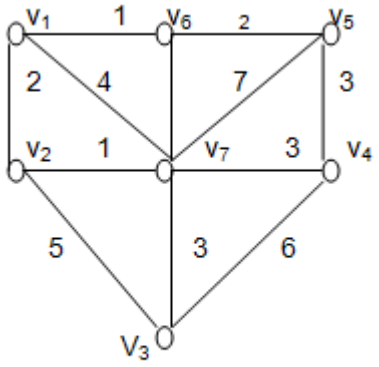
step7: edge(V_7, V_8)



step8: edge(V_5, V_6)

Minimum Spanning tree of the graph G

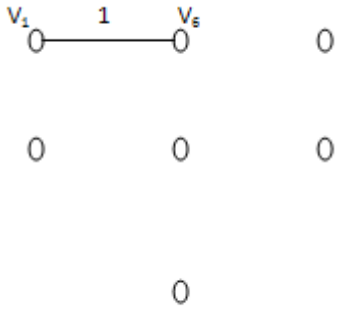
Edges in increasing order of their weight	Weights of edges	Selection of an edge
v_2-v_9	1	Yes
v_6-v_8	1	Yes
v_1-v_9	2	Yes
v_2-v_4	2	Yes
v_4-v_9	3	No
v_2-v_3	4	Yes
v_3-v_4	4	No
v_1-v_2	4	No
v_6-v_9	4	Yes
v_1-v_8	5	No
v_8-v_7	5	No
v_5-v_6	6	Yes
v_5-v_9	7	No
v_8-v_9	8	No
v_4-v_5	9	No



Graph G

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	–	2	–	–	–	1	6
V_2	2	–	5	–	–	–	1
V_3	–	5	–	6	–	–	3
V_4	–	–	6	–	3	–	3
V_5	–	–	–	3	–	2	7
V_6	6	1	3	3	7	4	–

Adjacency matrix of graph G



step1: edge(V_1, V_6)

Prim's algorithm:

Kruskal's algorithm requires sorting of all the edges in order of increasing weight. Because of the sorting involved, Kruskal's algorithm is not efficient. Also it requires verifying at each step whether a newly selected edge forms a circuit or not. To overcome these limitations, **Robort Prim** has proposed this algorithm.

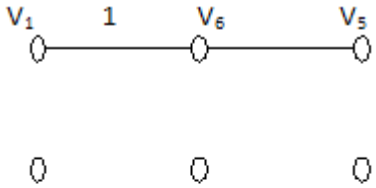
We adopt the following steps for finding minimum spanning tree using Kruskal's algorithm:

Step1: Represent the given weights of the edges of graph G in an n × n matrix, where n is the number of vertices.

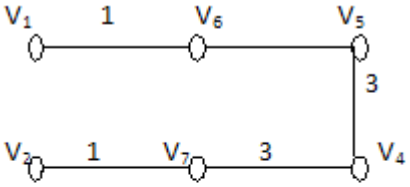
Step2: Start from vertex v_1 and connect it to its nearest neighbours (i.e., to the vertex which has the smallest entry in row 1 of the matrix), say v_k. If more than one smallest entry is there, then arbitrarily select one of them.

Step3: Consider v1 and v_k as one subgraph and connect the subgraph to its closest neighbours (i.e., to a vertex other than v_1 and v_k) which has the smallest entry in row 1 and k.

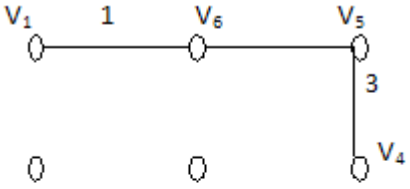
For example:



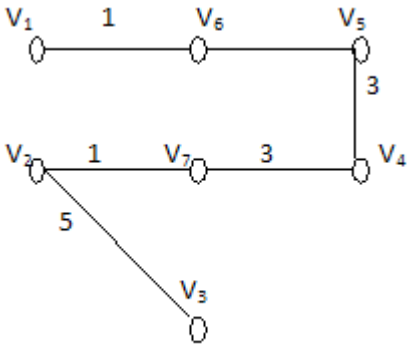
step2: edge(V_6, V_5)



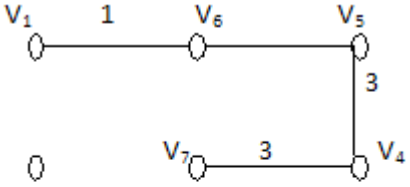
step5: edge(V_7, V_2)



step3: edge(V_5, V_4)



step6: edge(V_2, V_3)



step4: edge(V_4, V_7)

Proof of correctness of Kruskal's and Prim's algorithm:

Let G be a connected, weighted graph with n vertices. Let T be the spanning tree for G produced by Prim's algorithm. Suppose that the edges of T, in the order in which they were selected are $t_1,t_2,\ldots\ldots,t_{n-1}$. For each i from 1 to n-1, we define T_i to be the tree with edges $t_1,t_2,\ldots\ldots,t_i$ and $T_0=\{\ \}$. Then $T_0 \subset T_1 \subset \ldots \subset T_{n-1}=T$. We now prove, by mathematical induction, that each T_i is contained in a minimal spanning tree for G.

Basis Step:

Clearly P(0): $T_0=\{\ \}$ is contained in every minimal spanning tree for G.

Induction step

Let P(k): T_k is contained in a minimal spanning tree T' for G. We use P(k) to show P(k+1): T_{k+1} is contained in a minimal spanning tree for G. We have

$\{t_1,t_2,\ldots\ldots,t_k\} \subseteq T'$. If $t_{k+1} \in T'$, then $T_{k+1} \subseteq T'$ and we have P(k+1) is true. If $t_{k+1} \notin T'$, then since T' is minimal spanning tree, $T' \cup \{t_{k+1}\}$ must contain a cycle for some edges $s_1,s_2,\ldots\ldots,s_r$ in T'. Clearly the edges of this cycle cannot all be from T_k or T_{k+1} would contain this cycle. Let s_l be edge with smallest index l that is not in T_k. Then s_l has one vertex in T_k and one not in T_k. This means that when t_{k+1} was chosen by Prim's(or Kruskal's) algorithm, s_l was also available. Thus the weight of s_l is at least as large as that of t_{k+1}.The spanning tree

$(T'-\{s_l\})\cup\{t_{k+1}\}$ contains T_{k+1}. The weight of this tree is less than or equal to the weight of T', so it is a minimal spanning tree for G. Thus, P(k+1) is true. So T_{n-1}=T is contained in a minimal spanning tree and in fact it is a minimal spanning tree.

Time Complexity

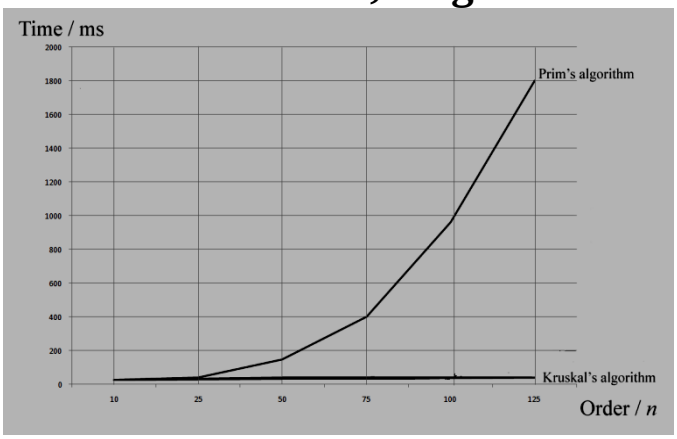
It has been shown that the time complexity of Kruskal's algorithm is O(E log E) or equivalently O(E log V), where E is the number of edges in the graph and V is the number of vertices. E is how many times we loop on the edges and this is multiplied by log V, which is how long it takes to perform the check for cycle task for each edge [6]. The time complexity for Prim's algorithm is O(V2) and O(E + V logV) depending on data structure [7].

Graphing runtime versus order

After ensuring that the algorithms do indeed give out the correct minimum spanning tree, an experiment was conducted to see the connection between the runtime in milliseconds and the order of a randomly generated graph. This is essentially the time complexity curve. In the experiment a random graph with order varying time was created and the minimum spanning tree was found by using both the algorithm. The number of edges was set to be equal to the order of the graph. Some experimentation was conducted beforehand and it became clear the number of vertices was more important than the number of edges. Doubling the edges increased runtime only by a few milliseconds. It is also evident from the time complexities that V has a greater effect. For each item, the test was run two or three times and the average was taken, although the deviations were nominal. As a result the following data table was produced:

Order *n*	*n*=10	n=25	*n*=50	*n*=75	*n*=100	*n*=125
Prim's	27	41	150	400	960	1800
Kruskal's	27	30	32	34	36	40

For a graph with an order of 10 the results were almost identical. When the order was increased to 25, slight differences became apparent and it appeared that Kruskal's algorithm was the fastest. For an order of 50, Prim's algorithm took more than five times the time than during an order of 10, while Kruskal's had increased by just five milliseconds. The next columns proceed in the same fashion and in the final graph test of an order of 125 vertices it took Prim's algorithm nearly two seconds to complete, while Kruskal's algorithm completed the same task in just 40 milliseconds. If drawn on a graph, with order in the x-axis and also being the independent variable and y-axis as the dependant variable time, the plot would look as follows:



The time complexity of Prim's algorithm clearly appears exponential just as it was given O(V2). In addition Kruskal's algorithms had the same time complexity and the graph clearly shows this trend, although Kruskal's gradient is slightly lower. The difference probably lies in the implementation.

IV. Conclusions

Kruskal's algorithm builds a minimum spanning tree by adding one **edge** at a time. The next line is always the shortest (minimum weight) only if it does not create a cycle. Prim's algorithm builds a minimum spanning tree by adding one **vertex** at a time. The next vertex to be added is always the one nearest to a vertex already on the graph. Prim's algorithm is the simplest of the two to implement. It also has the least number of code lines. While in Kruskal's algorithm edges are added, in Prim's on the other hand it is the vertices. In this algorithm the useful tool is the array which holds the vertices that have been added. During every iteration all of the vertices that are part of the array are checked and the lightest edge is selected and then added to the array. It is quite clear that after each round the number of vertices to be checked increases and since each vertex has a number of edges, we have a loop within a loop. For the last iteration the function must go through almost the entire graph. This greatly hinders the runtime of the algorithm and it was also visible during experimentation.

References

[1] http://citeseer.ist.psu.edu/old/413530.html.
[2] Kruskal, J. B., Jr., "On the Shortest Spanning Subtree of Graph and the Travelling Salesman Problem," Proc. Am. Math. Soc., Vol. 7, 1956.
[3] Prim, R. C., "Shortest Connection Networks and some Generalizations," Bell system Tech. J., Vol. 36, Nov. 1957.
[4] Kolman, Busby, Ross, Rehman, Discrete Mathematical Structures, Pearson education, 2006.

[5] C. L. Liu, Elements of Discrete Mathematics, Tata McGraw Hill, Special Indian Edition 2008.
[6] http://en.wikipedia.org/wiki/Kruskal%27s_algorithm
[7] http://en.wikipedia.org/wiki/Prim%27s_algorithm
[8] www.wikipedia.org – Online encyclopaedia.