

AN APPROACH TO GENERATE MST WITHOUT CHECKING CYCLE

Prof. Sharadindu Roy
University of Calcutta

sharadinduroysonarpur@gmail.com

Prof. Samar sen Sarma
University of Calcutta

sssarma2001@yahoo.com

Abstract: A minimum spanning tree of an undirected graph can be easily obtained using classical algorithms by Prim or Kruskal. MST generation is a NP hard problem. Now this paper represents an algorithm to find minimum spanning tree without checking cycle. Good time and space complexities are the major concerns of this algorithm. Running Time (complexity) of this algorithm = $O(E \cdot \log V)$ (E = edges, V = nodes), which is obviously better than prim's algorithm (complexity- $E + V \log V$). This algorithms operate at $O(E \cdot \log(V))$ time, though Prim's can be optimized to $O(E + V \log V)$ by using specialized data structures(heap). For large graphs, these algorithms can take significant amount of time to complete. This algorithm is important in many real world applications. One example is an internet service provider determining the best way to install underground wires in a neighbourhood in order to use the least amount of wire and dig the least amount of ground.

Keyword : MST(Minimum spanning tree)

Introduction:

A minimum spanning tree of a given graph is a sub-graph such that all vertices are connected, there are no cycles, and it has the minimum total weights possible in the graph. Minimum spanning trees are non-unique for many non-trivial graphs, so different algorithms many give wildly different trees.

However, by definition, any two algorithms which claim to produce a minimum spanning tree must result in a tree with the same sum of weights.

This algorithm is important in many real world applications. One example is an internet service provider determining the best way to install underground wires in a neighbourhood in order to use the least amount of wire and dig the least amount of ground.

Many algorithms exist to find these minimum spanning trees in a sequential manner. For example: Prim's Algorithm and Kruskal's algorithm now determine the minimum spanning tree by without checking cycle. This algorithm like Prim's, it does not need to worry about detecting cycles. These algorithms operate at $O(E \cdot \log(V))$ time, though Prim's can be optimized to $O(E + V \log V)$ by using specialized data structures. For large graphs, these algorithms can take significant amount of time to complete.

Definition:

Graph: A graph G consists of a vertex set V and an edge set E . Every element of E is an unordered pair of vertices. We will write undirected edges as $\{u,v\}$.

Subgraph: A subgraph H of G has an edge set E' is subset E and a vertex set induced by E' .

Path: A path in G is a sequence of vertices $v_0, v_1, v_2, \dots, v_k$ such that there is

an edge between any two adjacent vertices v_i, v_{i+1} in the sequence. We will sometimes

refer to the edges in the path, although it is formally defined as a sequence of vertices.

Connected graph: A graph is connected if there is a path between any two vertices in the graph.

Tree: A tree is a graph that is minimally connected – that is, any tree T is a connected graph, but removing any edge will disconnect it.

Spanning Trees: A spanning tree of a graph is just a subgraph that contains all the vertices and is a tree.

A graph have many spanning trees

Minimum Spanning Trees

The Minimum Spanning Tree for a given graph is the Spanning Tree of minimum cost for that graph.

Forest: A forest is a set of trees

Cycle: A cycle in G is a path whose endpoint is the same as its start point.

Since a spanning tree T of G is connected, there is a path involving only edges in T between any two vertices in G , and since it is a tree, this path is unique.

An *undirected graph* G is defined as a pair (V, E) , where V is a set of *vertices* and E is a set of *edges*. Each edge connects two vertices, i.e. $E = \{(u, v) | u, v \in V\}$. An undirected, *weighted* graph has a *weighting function* $w: E \rightarrow \mathcal{R}$, which assigns a weight to each edge. The weight of an edge is often called its cost or its distance.

A *tree* is a subgraph of G that does not contain any circuits. As a result, there is exactly one path from each vertex in the tree to each other vertex in the tree. A *spanning tree* of a graph G is a tree containing all vertices of G . A *minimum spanning tree* (MST) of an undirected, weighted graph G is a spanning tree of which the sum of the edge weights (costs) is minimal.

There are several greedy algorithms for finding a minimal spanning tree M of a graph. The algorithms of Kruskal and Prim are well known.

Kruskal's algorithm. Repeat the following step until the set M has $n-1$ edges (initially M is empty). Add to M the shortest edge that does not form a circuit with edges already in M .

Prim's algorithm. Repeat the following step until the set M has $n-1$ edges (initially M is empty): Add to M the shortest edge between a vertex in M and a vertex not in M (initially pick any edge of shortest length).

Although both are greedy algorithms, they are different in the sense that Prim's algorithm grows a tree until it becomes the MST, whereas Kruskal's algorithm grows a forest of trees until this forest reduces to a single tree, the MST.

A spanning tree s can be represented by a set of $n-1$ edges. An edge can be represented by an unordered couple of vertices.

$$S = \{(a_1, b_1), \dots, (a_{n-1}, b_{n-1})\}$$

The Minimum Spanning Tree Problem

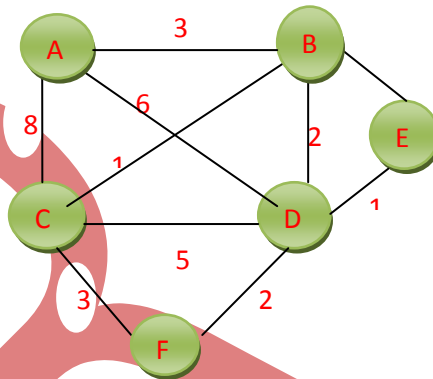
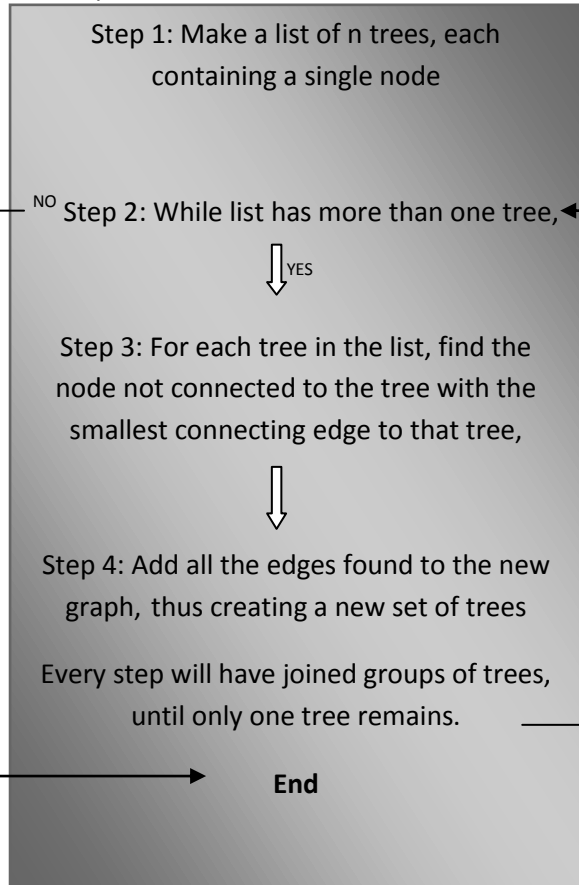
SOLUTION METHODOLOGY:

This algorithm is similar to Prim's, but nodes are added to the new graph in parallel all around the graph. It creates a list of trees, each containing one node from the original graph and proceeds to merge them along the smallest-weight connecting edges until there's only one tree, which is, of course, the MST. It works rather like a merge sort.

There are 10 edges. An edge connecting the vertices i and j may be represented by tuple (i,j) . The list of edges of the graph are $(a,b),(a,c),(b,c),(b,d),(b,e),(a,d),(c,d),(d,f),(c,f),(d,e)$.

Trees Of The Graph At Beginning Of Round 1

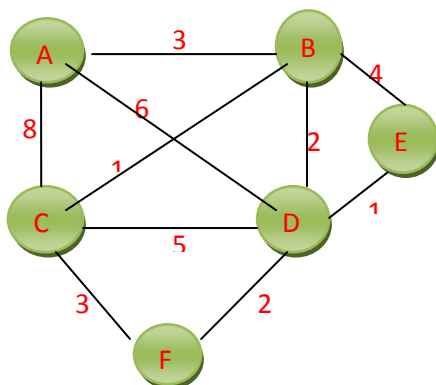
The steps are:



List of tree

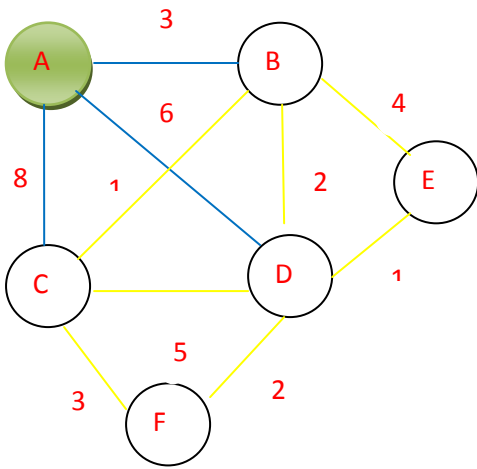
- A
- B
- C
- D
- E
- F

Complete Graph

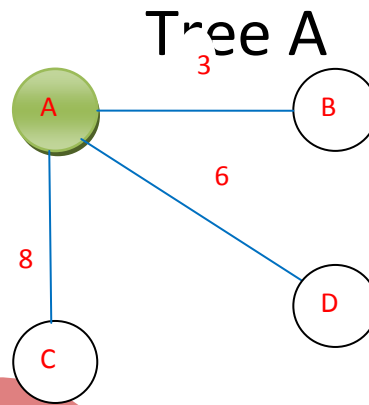


Consider a graph shown in the above figure

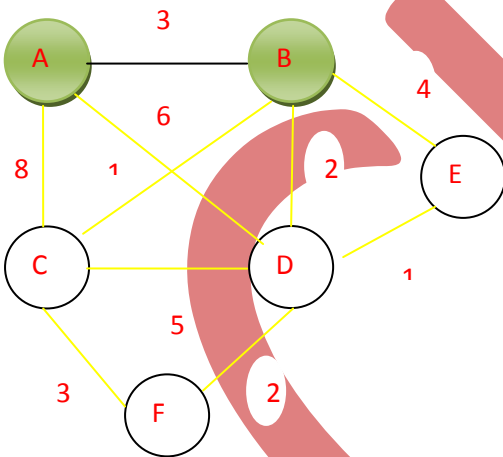
Round 1



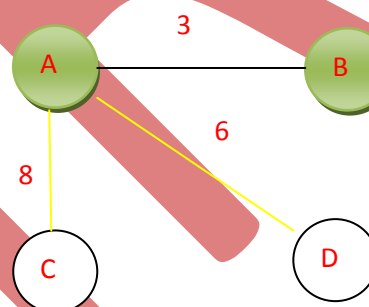
Tree A



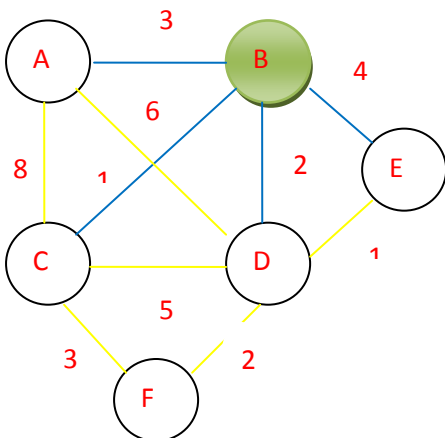
Round 1



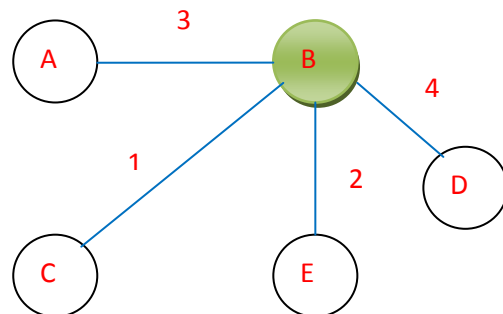
Edge A-B



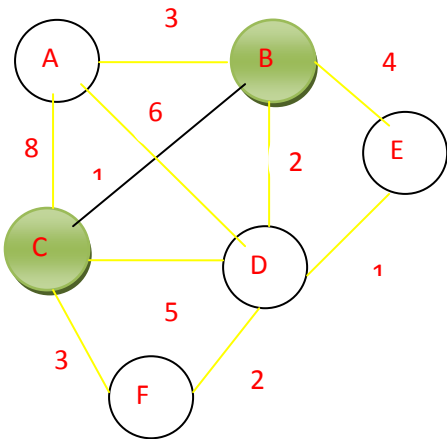
Round 1



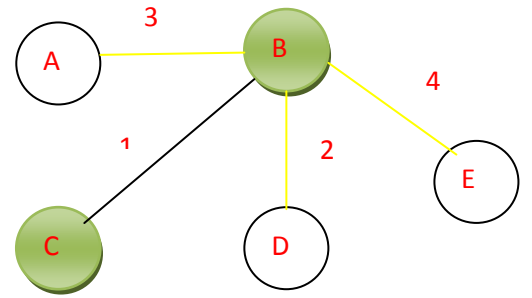
Tree B



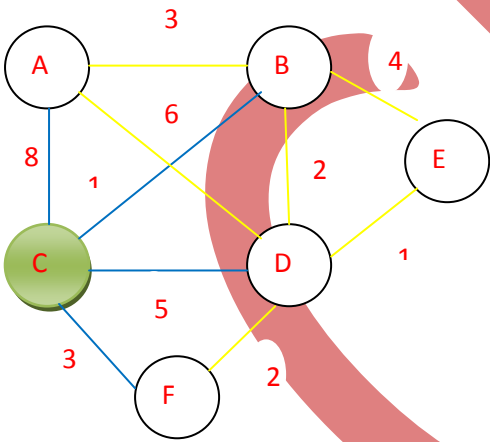
Round 1



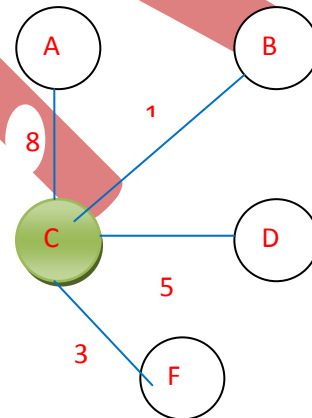
Edge B-C



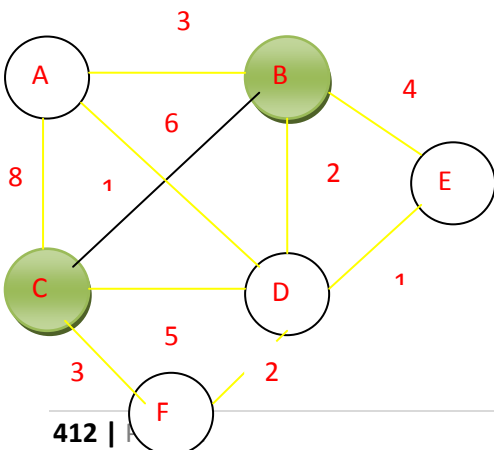
Round 1



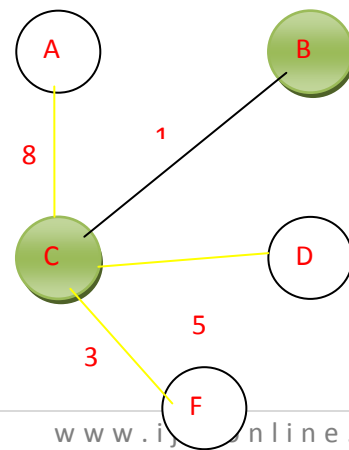
Tree C



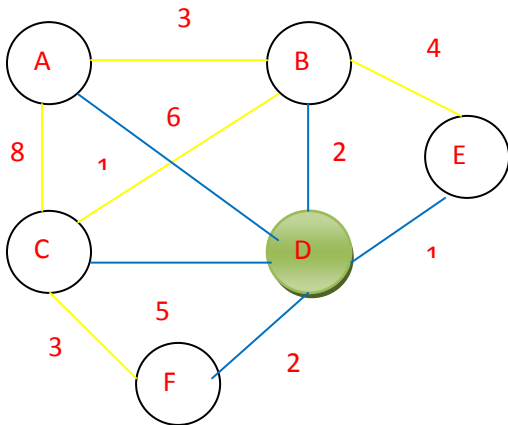
Round 1



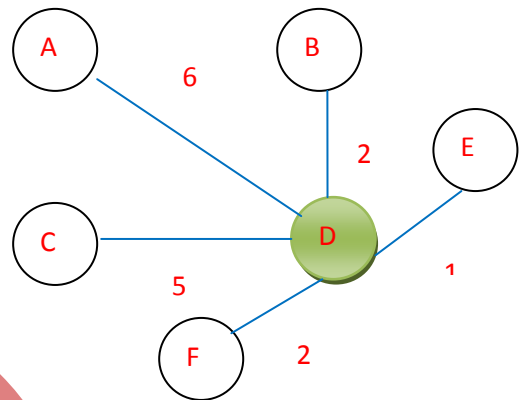
Edge C-B



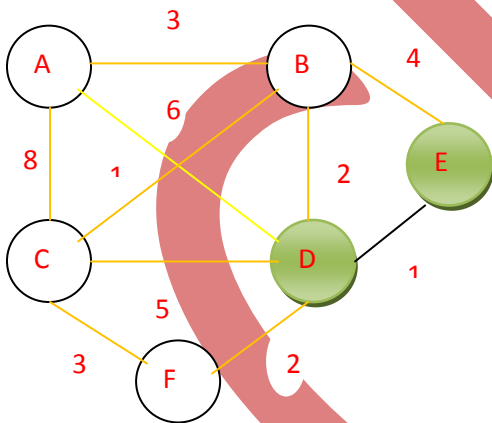
Round 1



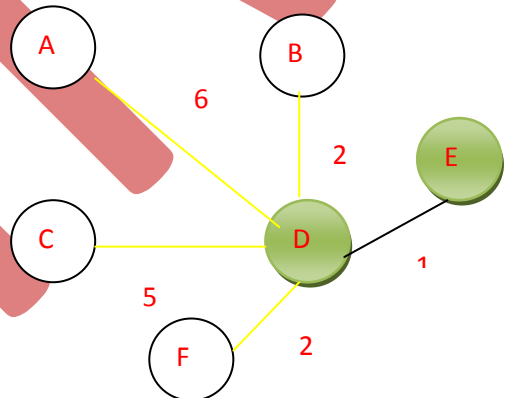
Tree D



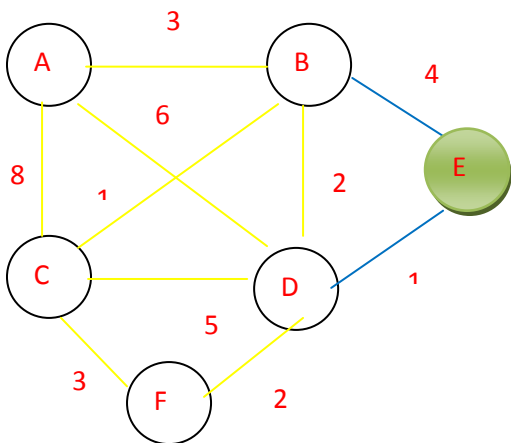
Round 1



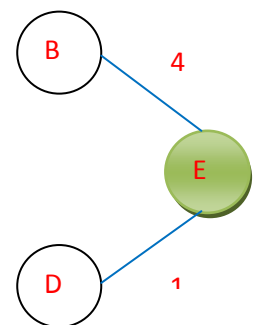
Edge D-E



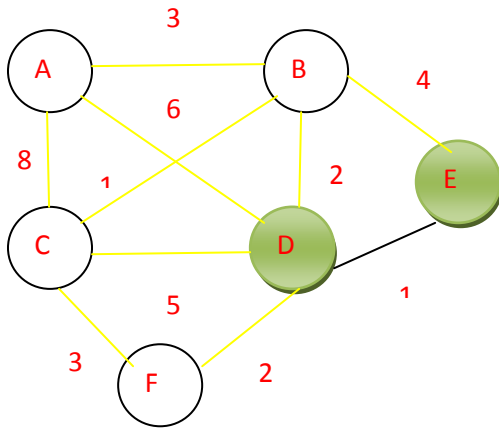
Round 1



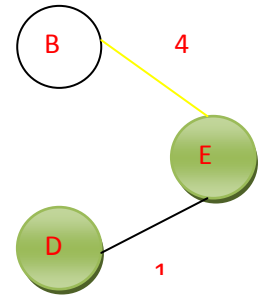
Tree E



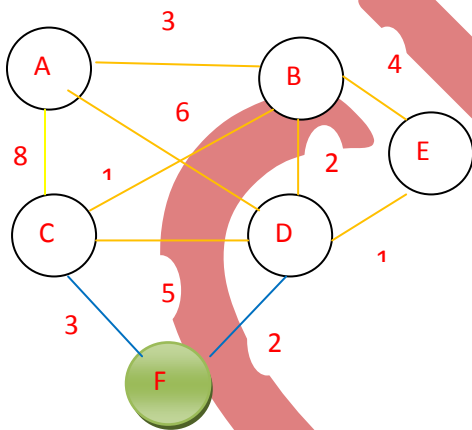
Round 1



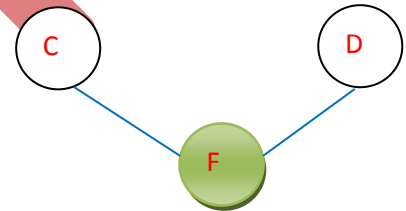
Edge E-D



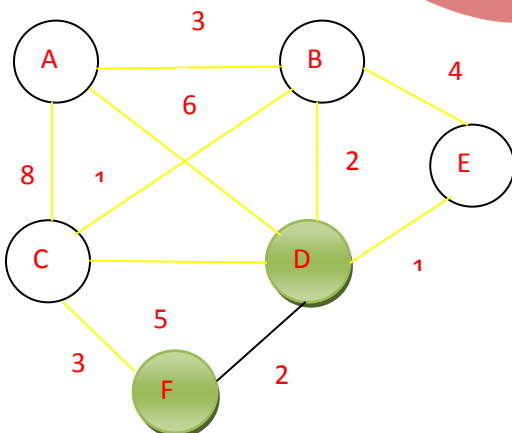
Round 1



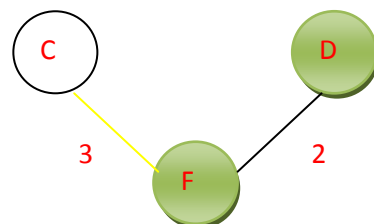
Tree F



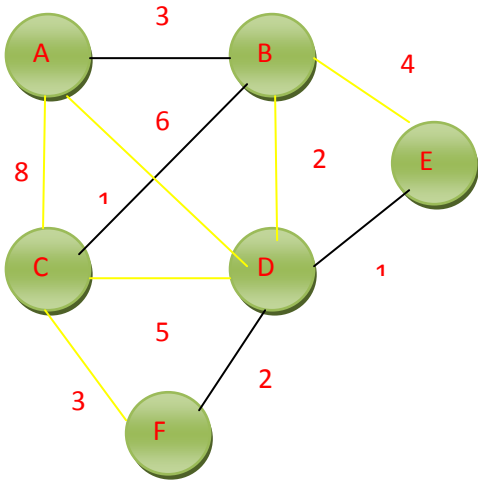
Round 1



Edge F-D



Round 1 Ends-Add Edges



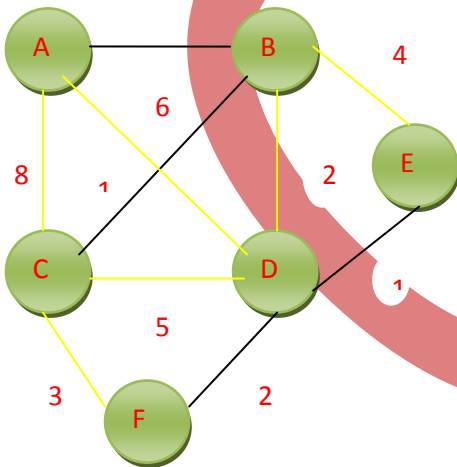
List of Edges to Add

After completion of round 1 we get the following edges

- A-B
- B-C
- C-B
- D-E
- E-D
- F-D

Trees Of Graph At Beginning

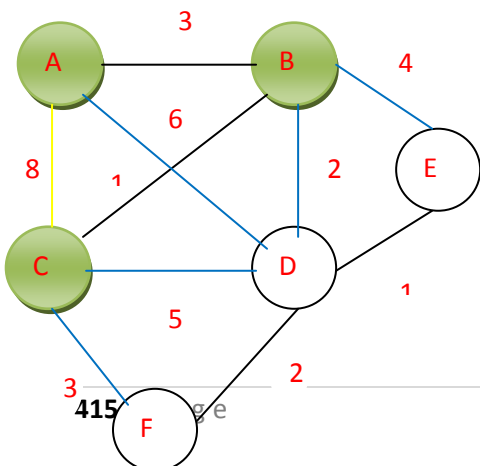
Of Round 2



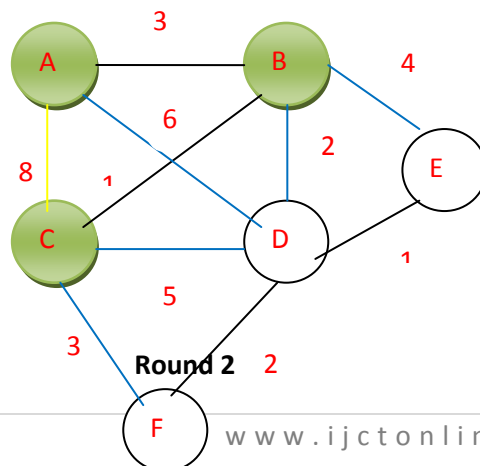
List of Trees

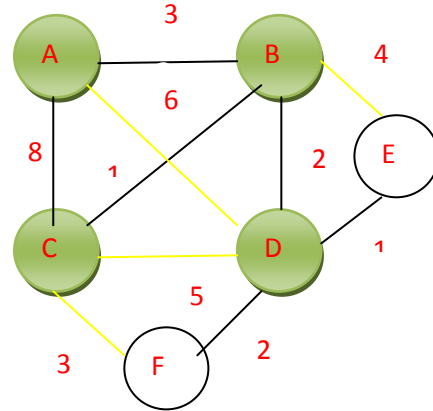
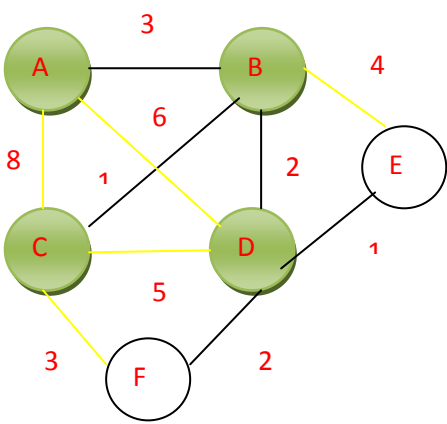
- A-B-C
- F-D-E

Round 2



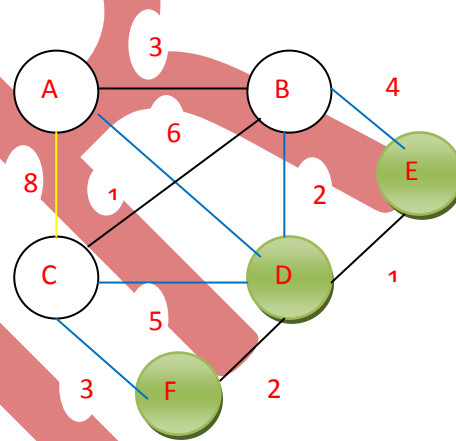
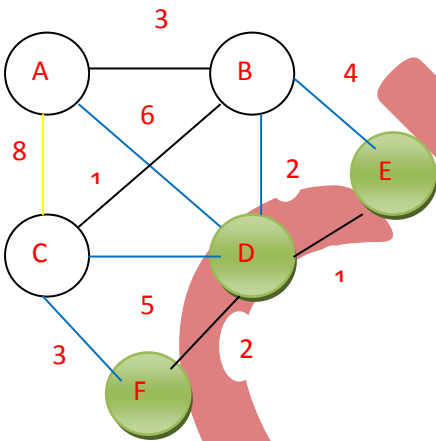
Tree A-B-C





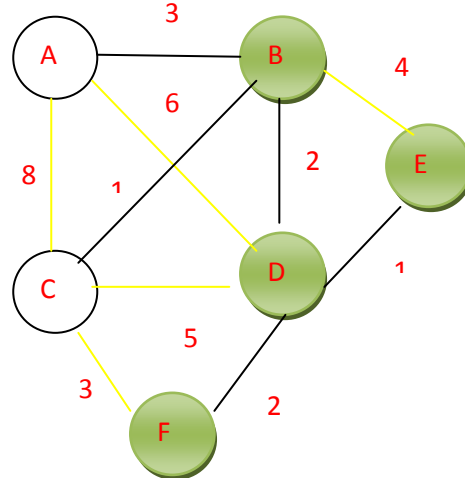
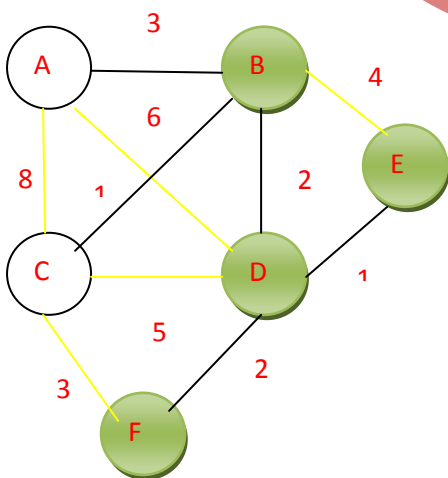
Round 2

Tree F-D-E



Round 2

Edge D-B




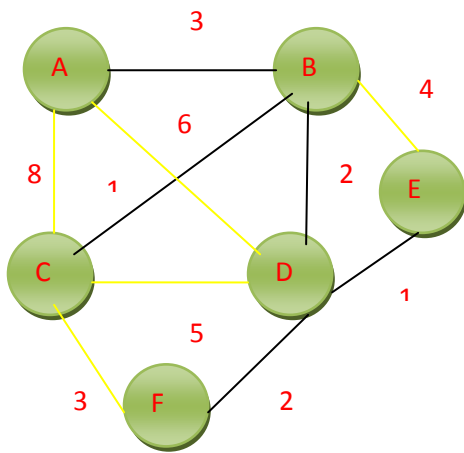
Edge B-D

Round 2 Ends-Add Edges

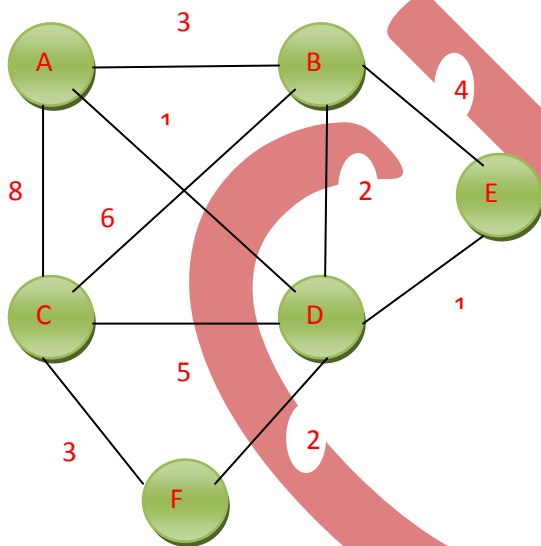
List of Edges To Add

After completion of round 2 we get the following edges

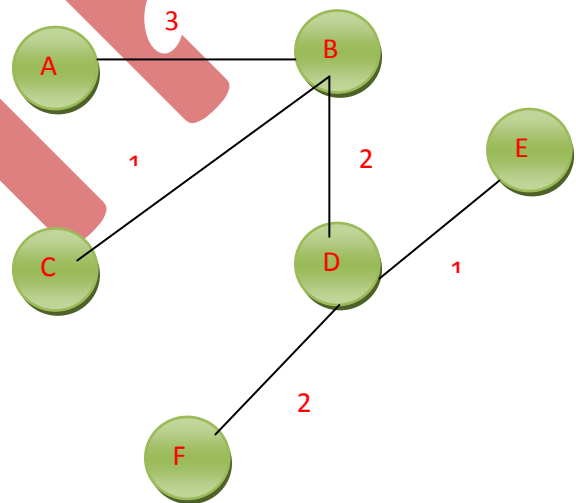
-  B-D
-  D-B



Complete Graph



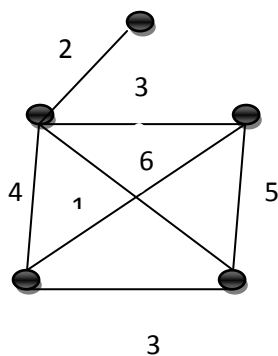
Minimum Spanning Tree



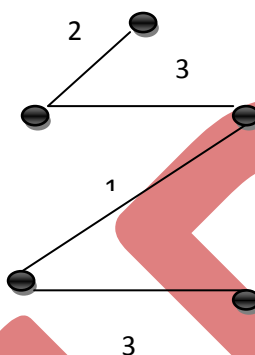
RESULT AND DISCUSSION:

This algorithm runs by c language using adjacency-list data structure and array.

Given graph



result (MST)



Running Time(complexity) of this algorithm = $O(E \log V)$ (E = edges, V = nodes), which is obviously better than prim's algorithm (complexity- $E + V \log V$).

Let $|E|$ be the number of edges, $|V|$ the number of vertices of a given graph G .

Although this algorithm is difficult to explain, unlike the two preceding algorithms, it does not require a complicated data structure.

Like Prim's, it does not need to worry about detecting cycles. It does, however, need to see the whole graph, but it only examines pieces of it at a time, not all of it at once.

Like Kruskal's it is best if edges are kept to a minimum (though it doesn't hurt to keep the nodes to a minimum as well).

Conclusion: It is obvious that algorithm has better running times regardless the number of edges within the graph. Main advantage of this algorithm is that it avoids cycle testing. So complicated data structures

which are needed for two algorithm (Prim's and kruskal's), is not needed here.

So, of course, this algorithm is best it minimize the cost of complex data structures.

It does $O(\log \log n)$ passes and then switches to Prim's, resulting in a running time of $O(m \log \log n)$. So, it's the fastest algorithm, but would, of course, require the Fibonacci heap for Prim's which this algorithm avoids when used by itself. However, in order to keep things simple, I did not explore it here.

REFERENCES:

- [1] Bader, D., & Cong, G. (2006). Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs. *Journal of Parallel and Distributed Computing*, 66(11), 1366-1378. doi:10.1016/j.jpdc.2006.06.001
- [2] Setia, R., Nedunchezian, A., & Balachandran, S. (2009). A new parallel algorithm for minimum spanning tree problem. *Proc. International Conference on High Performance Computing (HiPC)* (pp. 1-5). Retrieved from http://111.hipc.org/hipc2009/documents/HIPC_CSS09Papers/1569250351.pdf
- [3] Osipov, V., Sanders, P., & Singler, J. (2009). The filter-Kruskal minimum spanning tree algorithm. *ALLENEX* (pp. 52-61). SIAM. Retrieved from http://72.32.205.185/proceedings/alenix/2009/alx09_005_osipov.pdf
- [4] Knowles J.D., Corne D.W., A Comparison of Encodings and Algorithms for Multiobjective Minimum Spanning Tree Problems, University of Reading, UK, <http://www.rdg.ac.uk/~ssr97jdak>.
- [5] Lin G-H., Xue G., Steiner problem with minimum number of Steiner points and bounded edge-length, *Information Processing Letters*, 69, 1999, 53-57.