CS 302: DESIGN AND ANALYSIS OF ALGORITHMS

"PROJECT REPORT"

Project Title: **"Implementation and Evaluation of Graph Theory Algorithms"**

Submitted by:

1. Murad Popattia (17K-3722) (Section B)
2. Muhammad Samiullah (17K-3745) (Section B)

Date of Submission: 01 - 12 - 2019

Abstract:

The following project has been built on the basis of benchmarking graph algorithms based on increasing number of nodes. This benchmarking has been achieved by implementing and testing the algorithms in an IDE and a programming language in order to simulate and evaluate the working of the algorithms.

Introduction:

The problem at hand was to evaluate the output of various graph algorithms including:

- Prim's Algorithm for Minimum Spanning Tree
- Kruskal's Algorithm for Minimum Spanning Tree
- Dijkstra Algorithm for Single Source Shortest Path
- Bellman Ford's Algorithm for Single Source Shortest Path
- Floyd Warshall's Algorithm for All Pair Shortest Path
- Clustering Coefficient of Graph (Local Clustering)

For increasing number of nodes and benchmark the performance of each algorithm as per increasing number of nodes(10 to 100 nodes specifically in this scenario).

Proposed System:

System consists of various components in order to make the interface as user friendly as possible.

- The first component is the sidebar component that enables user to navigate among various panes / component. Fig A.1
- The second component is the Overview Pane that enables the user to make selection of input file on which he / she would benchmark various algorithms. This runs the files a single time for all algorithms to evaluate the time taken for each to give an output for undirected graphs. Fig A.2
- The third component is for the shortest path algorithm pane (Bellman and Dijsktra). As they both have the same flow, hence shown as one. Fig A.3

- The fourth component is for the All Pair Shortest Path (Floyd Warshall) Algorithm. Fig A.4
- The fifth component is for the minimum spanning tree algorithms. (Kruskal and Prims) Fig A.5
- The sixth component is for the Clustering Coefficient where the CC for all the nodes and the whole graph can be viewed. Fig A.6

Experimental Setup:

The system takes input in the form of ".txt" that ensure a certain format to be followed. The file must contains, number of nodes, coords of each node, all edges with their cost that must be displayed on the graph, and the source vertex for shortest path algorithms. Further details of the software and frameworks used include:

- IDE: IntelliJ IDEA Build 2019.2.3
- Programming Language: Java 1.8 JDK
- Framework for graph visualization: JUNG Framework (2.0.1)

Results:

After the input file has been read, the system saves the graph in the form of "Adjacency List" (a list of multiple linked lists) in order to store nodes and their connections in the graph.

The algorithms mentioned above have been implemented using various approaches. These approaches have been mentioned as such:

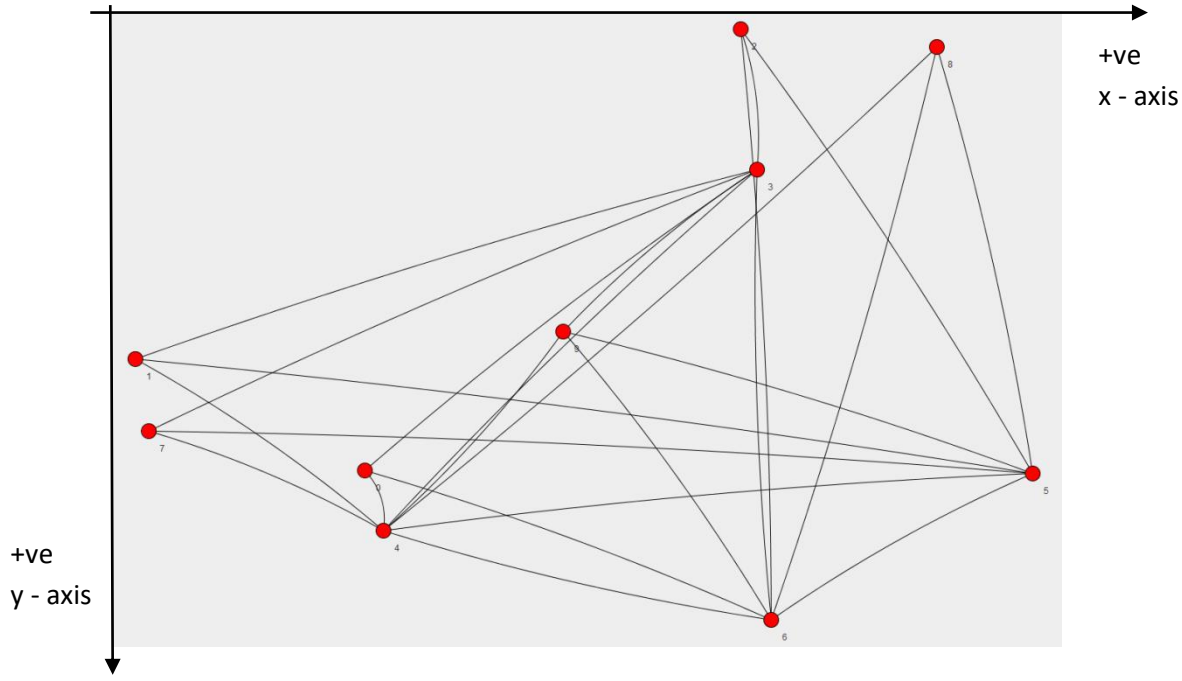
	Algorithm	Approach	Time Complexity	Time (µs)	Status
	Prims	Adjacency List and Min Heap	O(E Log V)		Inactive
	Kruskal	Adjacency List and Priority Queue	O(E Log V)		Inactive
	Dijkstra	Adjacency List and Min Heap	O(E Log V)		Inactive
	Bellman Ford	Adjacency List	O(VE)		Inactive
	Floyd Warshall	Adjacency Matrix	O(n^3)		Inactive
	Clustering Coefficient	Adjacency List	O(n^3)		Inactive

The system hence calculates the time it takes for each algorithm in (µs) to evaluate the result based upon increasing number of input nodes.

	10	***20***	***30***	***40***	***50***	***60***	***70***	***80***	***90***	***100***
Prims	22	38	29	54	45	54	57	63	69	108
Kruskal	17	15	23	50	41	57	72	94	92	141
Dijkstra	23	62	59	61	50	58	69	77	90	117
Bellman Ford	26	23	32	45	58	67	79	89	116	263
Floyd	40	34	41	56	86	132	213	335	388	521
Clustering Coeff	37	40	48	66	84	109	186	217	262	409

Even though, the time complexities differ from O(ElogV) to $O(n^3)$, we see that the difference is in time is not as the complexities, for smaller inputs as 10, 20, 30 nodes. But for nodes > 70 we see a significant difference in their elapsed times hence justifying their complexities.

Before going onto the benchmark, the display of the graph is as follows:



Weights have not been shown on the graph, but instead they have been shown on the console due to limitation of framework to effectively show weights as “EdgeWeightLabellers” (decoraters) are present in jung 1.7.6 jar. However we have used jung 2.0.1.jar.

For undirected:

Benchmark	***Prim's***	***Kruskal***	***Clustering Coefficient***
Input 10	24.45	24.45	0.6583
Input 20	51.45	51.45	0.5828
Input 30	87.6	87.6	0.6981
Input 40	137.10	137.10	0.771
Input 50	133.05	133.05	0.6314
Input 60	201.15	201.15	0.7437
Input 70	195.75	195.75	0.6997
Input 80	249.30	249.30	0.7384
Input 90	287.10	287.10	0.8192
Input 100	304.5	304.5	0.7294

Details regarding table:

- The columns for Prim's and Kruskal's show the total sum of all weights of edges present in the minimal spanning tree. We see that both give the same answer.
- Clustering coefficient here represents the average of clustering coefficients of all the nodes present in the graph
- Assumption for clustering coefficient: If there are no neighbors or one neighbor then, clustering coefficient is trivially defined to be 1.

Benchmark	***Djisktra (Source node = 1) Undirected***	***Djisktra (Source node = 1) Directed***	***Bellman (Source node = 1) Undirected***	***Bellman (Source node = 1) Directed***	***Floyd's (Source node = 1) Undirected***	***Floyd (Source node = 1) Directed***
Input 10	52.8	∞	52.8	∞	453.6	181.95 + ∞
Input 20	122.25	∞	122.25	∞	2274.9	1120.5 + ∞
Input 30	162.9	∞	162.9	∞	6271.8	1587.3 + ∞
Input 40	344.55	∞	-	∞	13504.2	2376.9 + ∞
Input 50	260.4	∞	-	∞	15542.1	1667.4 + ∞
Input 60	557.25	∞	-	∞	28674.6	4015.8 + ∞
Input 70	475.35	∞	-	∞	36364.5	3007.35 + ∞
Input 80	427.35	∞	-	∞	50689.5	4002.45 + ∞
Input 90	765.3	∞	-	∞	65643	4173.9 + ∞
Input 100	693.15	∞	-	∞	80038.8	5497.35

Details regarding table:

- The table shows results for both the directed and undirected versions of the algorithms.
- For dijkstra and bellman ford, the answer is basically sum of all the costs where each cost represents the shortest cost for source node to reach every node except itself.
- For Floyd's it is the sum of all the costs where each cost is shortest cost of all pair of vertex present in the graph.
- (-) in bellman ford represents, that the graph contains negative weighted cycles, hence algorithm will not produce an accurate result.
- (∞) shows that there is no path to any vertex using the source vertex.
- (number + ∞) shows that there are possible pairs present (the graph has some connections)

Conclusion:

Hence, using this system, we are easily able to visualize how graph algorithms can be used in various applications such as networks, and other domains where shortest paths, cycle detections and minimal spanning connections are vital. We are also able to benchmark algorithms and affirm there complexities based on variety of inputs.

References:

- https://www.geeksforgeeks.org/ (For Algorithms)
- https://www.tutorialhorizon.com/ (For Algorithms)
- https://www.youtube.com/watch?v=K2WF4pT5pFY (For Clustering Coefficient)
- https://stackoverflow.com/
- http://jung.sourceforge.net/doc/manual.html (For JUNG Framework)

Appendix: (Data Flow Diagrams for all panes)

Fig A.1:

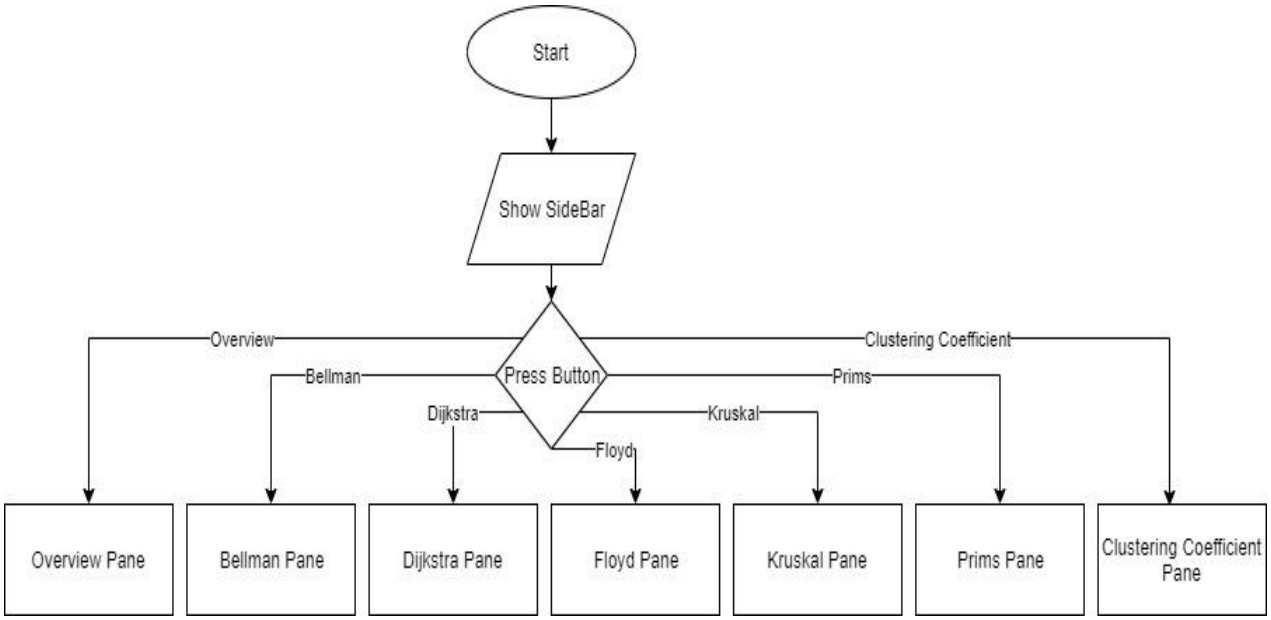


Fig A.2:

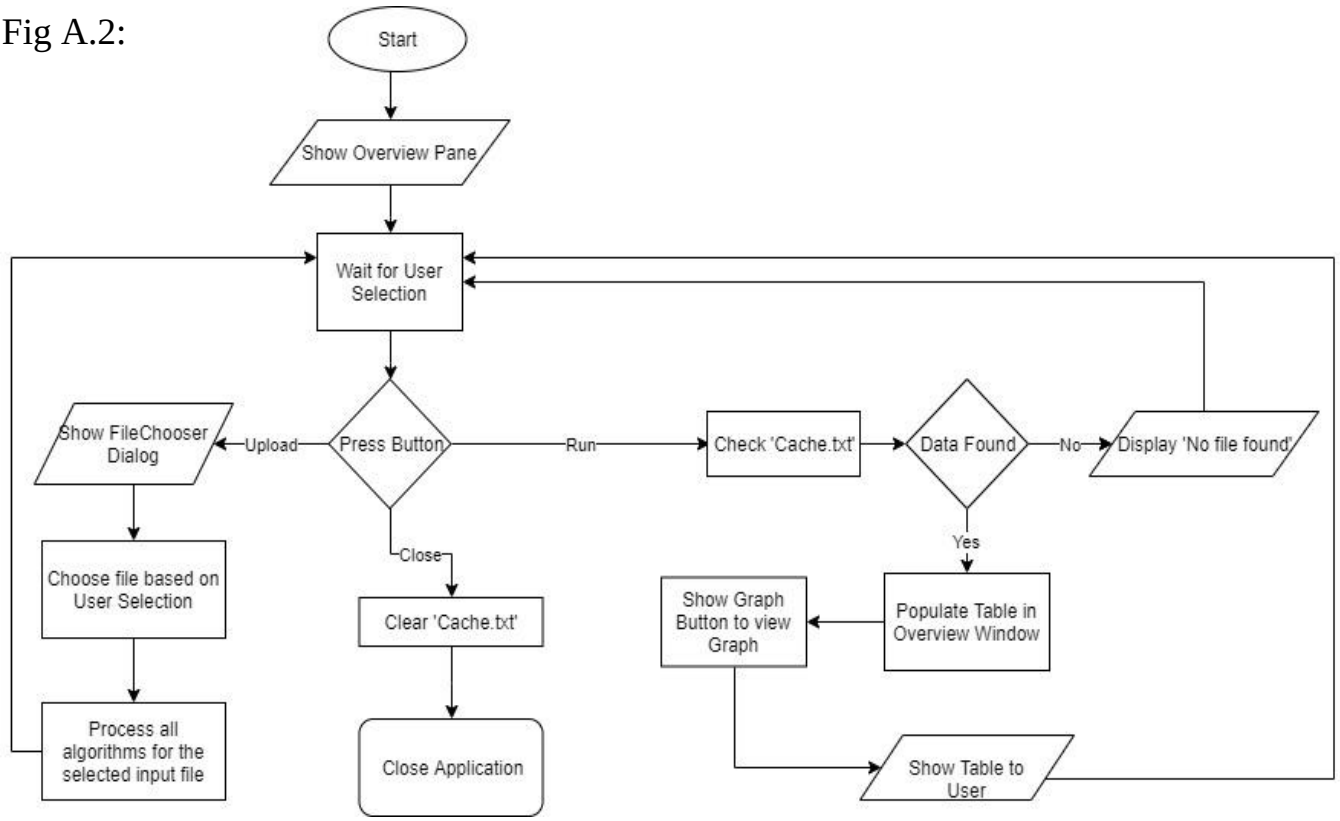


Fig A.3:

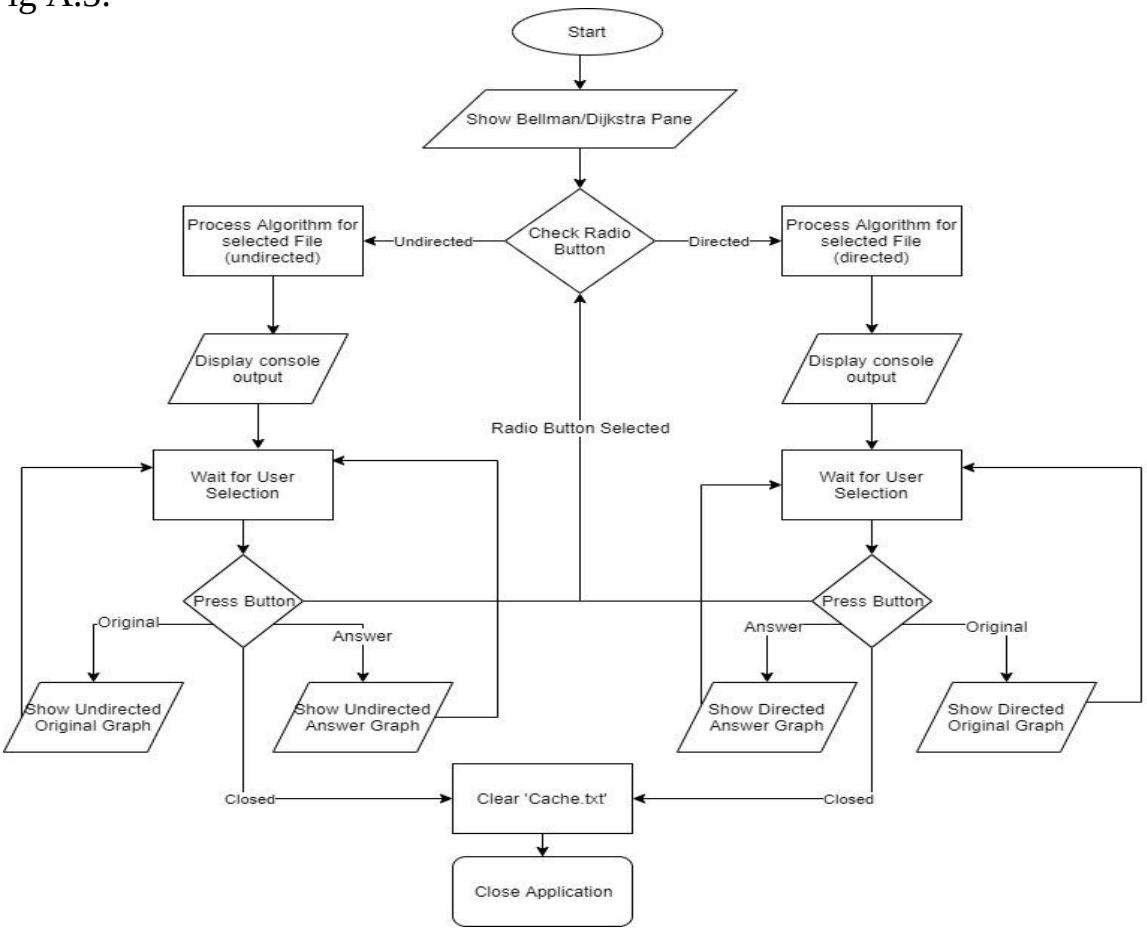

Start
Show Bellman/Dijkstra Pane
Check Radio Button
Undirected
Directed
Process Algorithm for selected File (undirected)
Process Algorithm for selected File (directed)
Display console output
Display console output
Radio Button Selected
Wait for User Selection
Wait for User Selection
Press Button
Press Button
Original
Answer
Answer
Original
Show Undirected Original Graph
Show Undirected Answer Graph
Show Directed Answer Graph
Show Directed Original Graph
Closed
Clear 'Cache.txt'
Closed
Close Application

Fig A.4:

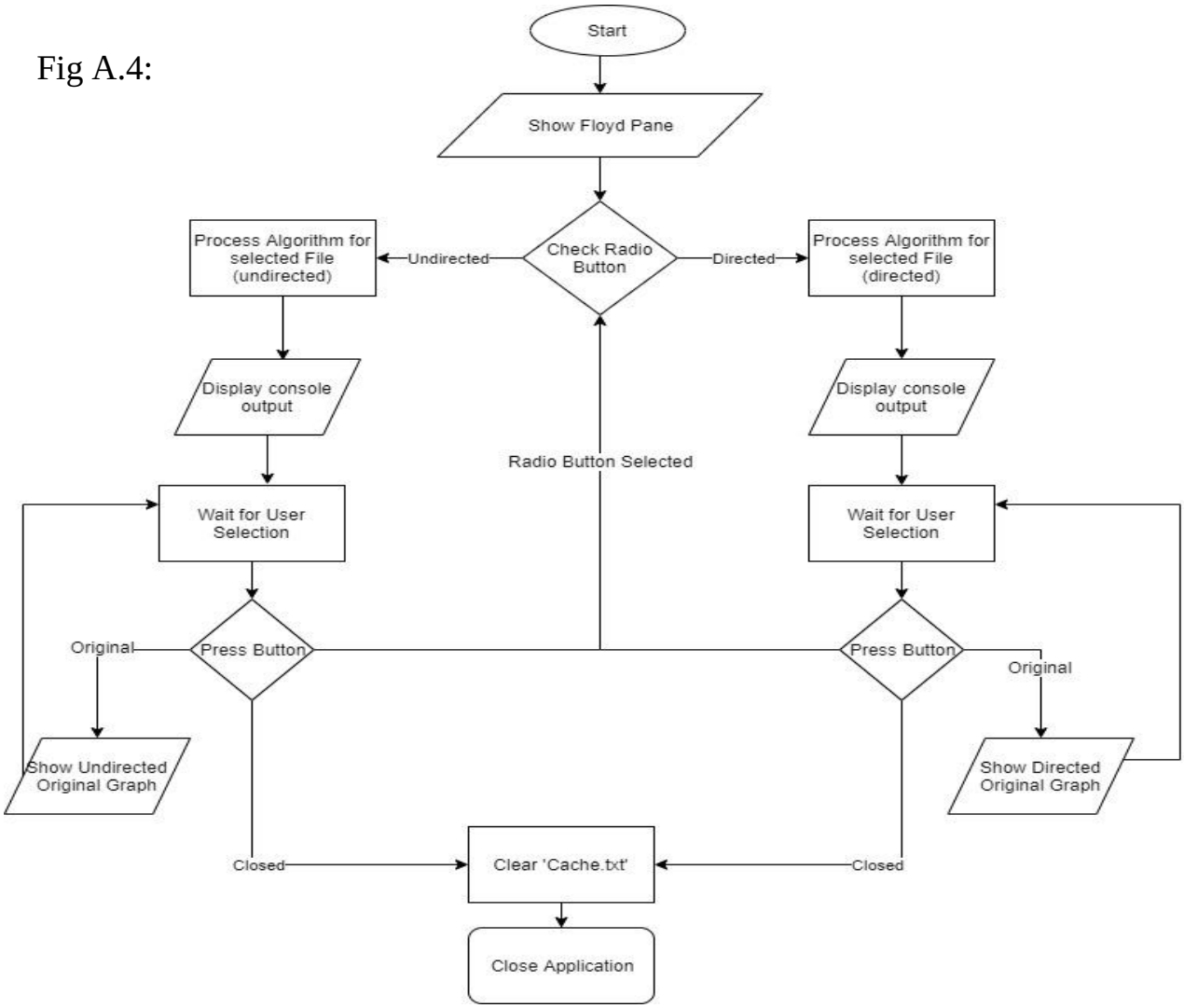

Start
Show Floyd Pane
Check Radio Button
Undirected
Directed
Process Algorithm for selected File (undirected)
Process Algorithm for selected File (directed)
Display console output
Display console output
Radio Button Selected
Wait for User Selection
Wait for User Selection
Press Button
Press Button
Original
Original
Show Undirected Original Graph
Show Directed Original Graph
Closed
Clear 'Cache.txt'
Closed
Close Application

Fig A.5:

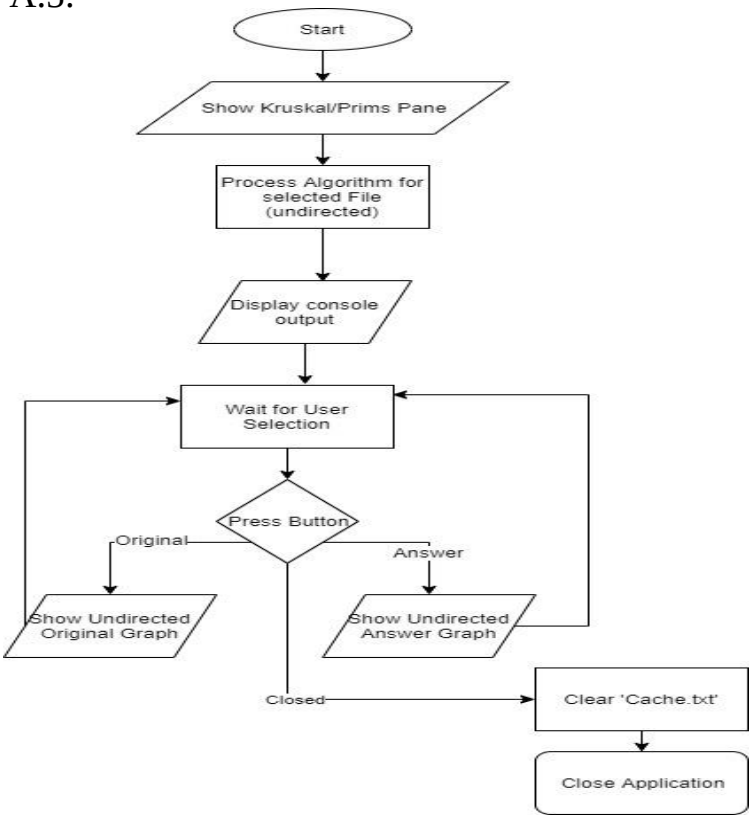
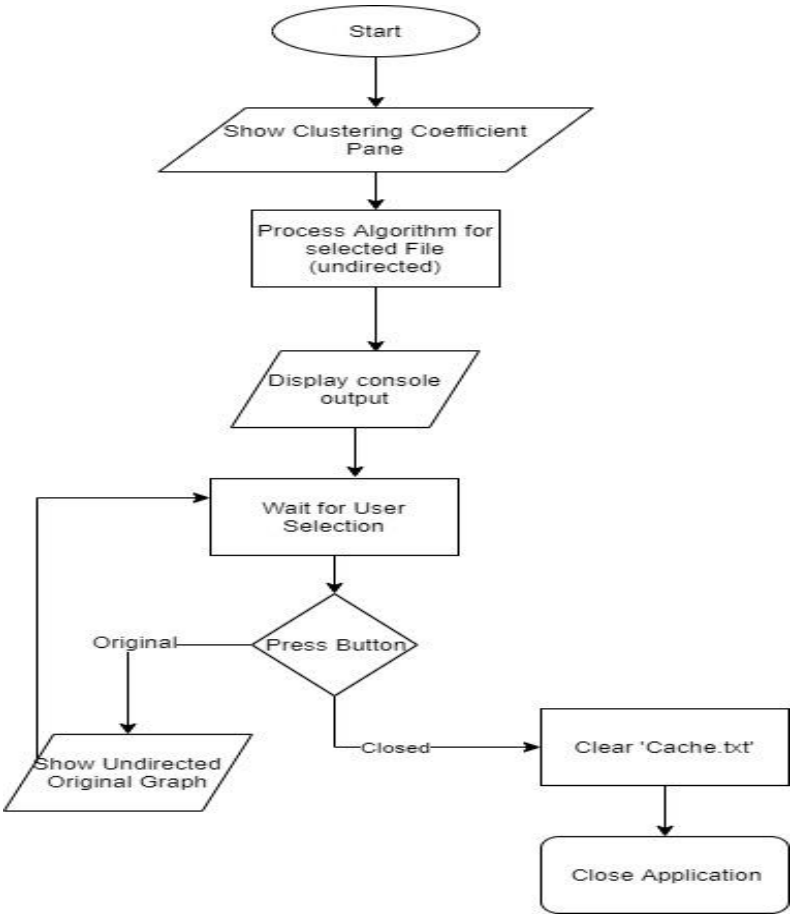

Start
Show Kruskal/Prims Pane
Process Algorithm for selected File (undirected)
Display console output
Wait for User Selection
Press Button
Original
Answer
Show Undirected Original Graph
Show Undirected Answer Graph
Closed
Clear 'Cache.txt'
Close Application

Fig A.6:


Start
Show Clustering Coefficient Pane
Process Algorithm for selected File (undirected)
Display console output
Wait for User Selection
Press Button
Original
Show Undirected Original Graph
Closed
Clear 'Cache.txt'
Close Application