PROJECT REPORT

DESIGN AND ANALYSIS OF ALGORITHM

DEPARTMENT OF COMPUTER SCIENCE

Submitted to:

SIR WAQAS SHEIKH

Submitted by:

Rumaisa Mansoor (17K-3601)

Eisha Tir Raazia (17k-3730)

Sec: C

National University of Computer and Emerging Sciences-FAST

Karachi Campus

Abstract:

In our project, we will be implementing six different algorithms:

1. Prims
2. Kruskal
3. Dijisktra
4. Bellman-Ford
5. Floyd Warshall
6. Clustering Coefficient (Local Clustering)

To find the shortest path between the initial and the final node. Total Cost calculation in running the algorithm will also be determined at the end of the program.
We are using 10 different sets of input files.

Introduction:

we are using 6 different algorithms to find the shortest path between root to the final node. Shortest path algorithms are basically a problem about finding a path between 2 vertices in a graph such that the total sum of the weights of the edges is minimum.

The biggest advantage of using these algorithms is that all the shortest distances between any vertices could be calculated in O (V3), where is the number of vertices in a graph.

Similarly, we are using different algorithms to compare the total cost and total time needed to implement the algorithm

1. **Prims**: Prim's (also known as Jarník's) algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.

2. **Kruskal**: Kruskal's algorithm is a minimum-spanning-tree algorithm that finds an edge of the least possible weight that connects any two trees in the forest. It is a greedy algorithm in graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step.

3. **Dijkstra**: Dijkstra's algorithm (or Dijkstra's Shortest Path First algorithm, SPF algorithm) is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. ... For a given source node in the graph, the algorithm finds the shortest path between that node and every other.

4. **Bellman-Ford**: The Bellman-Ford algorithm is an algorithm that computes the shortest paths from a single source vertex to all of the other vertices in a weighted digraph.

5. **Floyd Warshall**: is an algorithm for finding shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles). A single execution of the algorithm will find the lengths of the shortest paths between all pairs of vertices.

6. **Clustering Coefficient**: Triangle counting is a community detection graph algorithm that is used to determine the number of triangles passing through each node in the graph. A triangle is a set of three nodes, where each node has a relationship to all other nodes.

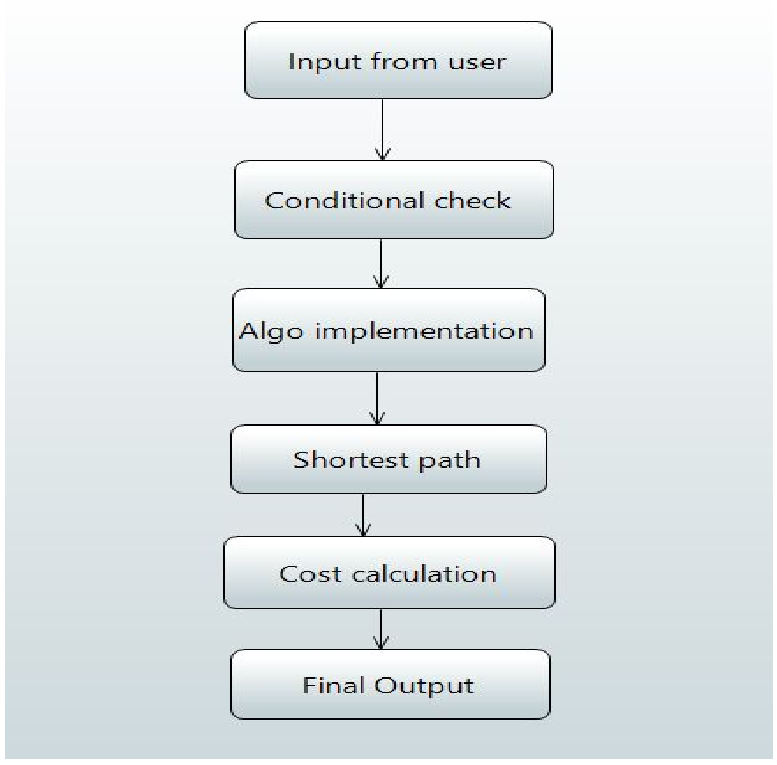
Proposed system:

we ask the user input, based on the input and choice of the user, the respective benchmark file is opened after the conditional check of switch case which is applied for 10 different input options.

On the outcome of the Switch case, the respective Algorithm reads the file and starts executing to find the shortest path among different nodes and calculate the total cost from one node to another.

GUI representation shows the outcome of the algorithm to the user.

Flow Diagram:



Experimental Setup:

We have 10 different input options:

1. 10 as 1
2. 20 as 2
3. 30 as 3
4. 40 as 4
5. 50 as 5
6. 60 as 6
7. 70 as 7
8. 80 as 8
9. 90 as 9
10. 100 as 10

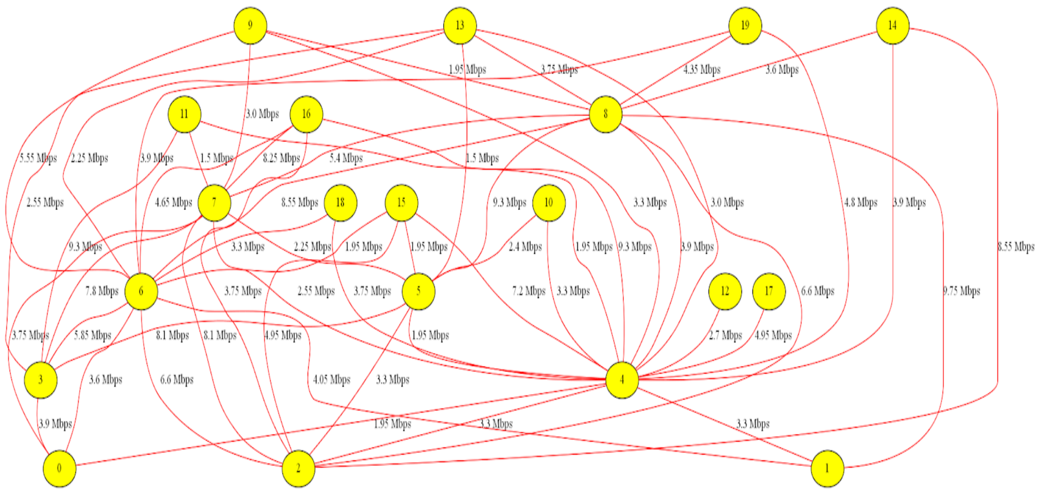
Based on the user's choice, the input is then matched to switch case conditions which are again 10.
When the case is matched, the respective benchmark file is opened to read the input from that file and the algorithm starts its execution.

TIME COMPLEXITIES OF ALGORITHMS:

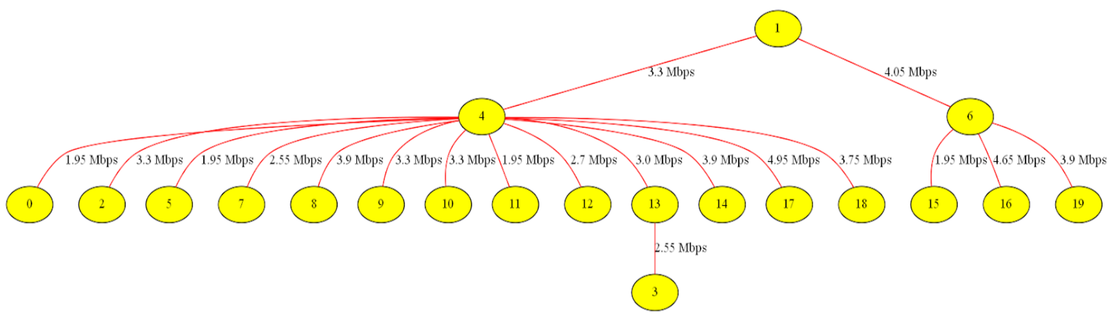
1. Prims: O(VlogV + ElogV) = O(ElogV)
2. Kruskal: ElogE
3. Dijkstra: O((v+e) log v)
   O(e log v)
4. Bellmen Ford: *O(VE)*
5. Floyd Warshall: O(n^3)

Graphical Representation:

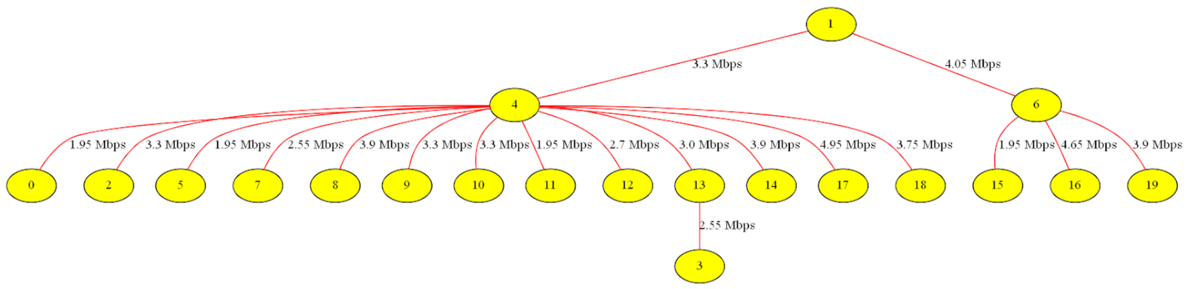
- **Sample of 20 Nodes:**
  - **INPUT**



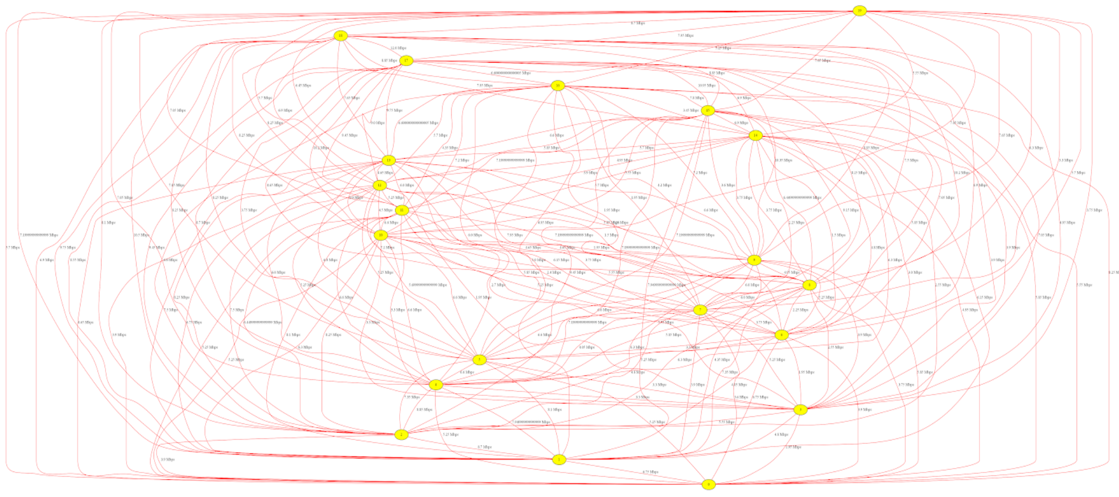
- **DIJKSTRA**



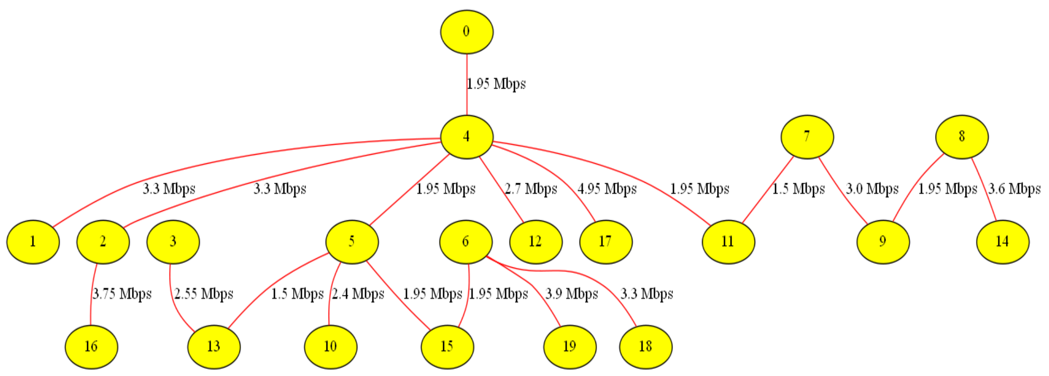
- **BELLMAN-FORD**



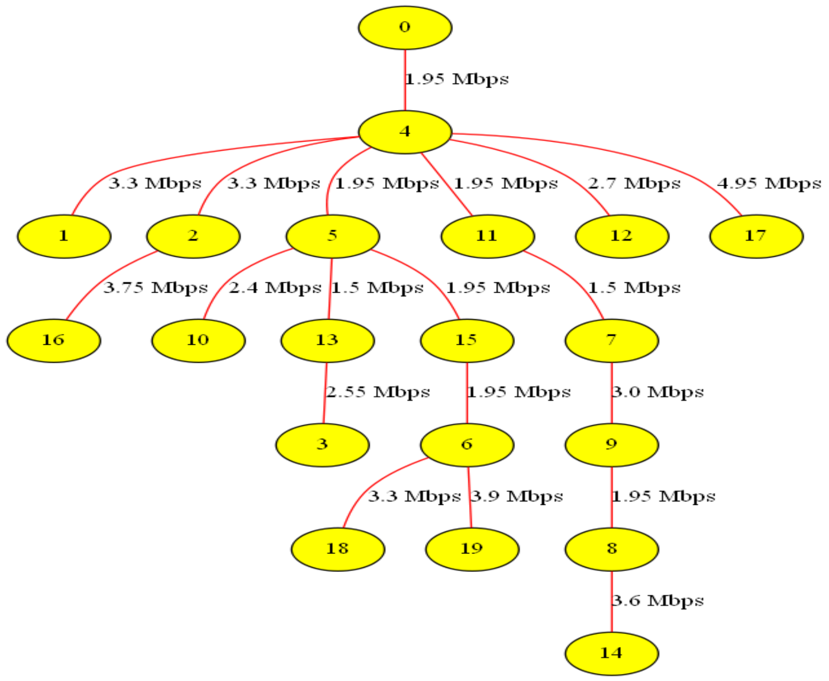
- **FLOYD-WARSHALL**



- **KRUSKAL**



- **PRIMS**



- **OUTPUT GENERATED IN FILE:**
  - The total cost of Kruskal is 51.45 Mbps
  - The total cost of Prim is 51.45 Mbps
  - The input node is 1 and cost by Dijkstra   is 122.25 Mbps
  - The input node is 1 and cost by Bellman ford is 122.24999999999999 Mbps
  - Total cost by Floyd Warshall is 2274.900000000001 Mbps
  - Local Clustering Coefficient of node 0 is 0.5
  - Local Clustering Coefficient of node 1 is 0.6666666666666666
  - Local Clustering Coefficient of node 2 is 0.5357142857142857
  - Local Clustering Coefficient of node 3 is 0.4
  - Local Clustering Coefficient of node 4 is 0.15441176470588236
  - Local Clustering Coefficient of node 5 is 0.4642857142857143
  - Local Clustering Coefficient of node 6 is 0.16363636363636364
  - Local Clustering Coefficient of node 7 is 0.4166666666666667
  - Local Clustering Coefficient of node 8 is 0.4222222222222222
  - Local Clustering Coefficient of node 9 is 0.6666666666666666
  - Local Clustering Coefficient of node 10 is 1.0
  - Local Clustering Coefficient of node 11 is 0.6666666666666666
  - Local Clustering Coefficient of node 12 is 0
  - Local Clustering Coefficient of node 13 is 0.6
  - Local Clustering Coefficient of node 14 is 1.0
  - Local Clustering Coefficient of node 15 is 0.6666666666666666
  - Local Clustering Coefficient of node 16 is 0.6666666666666666
  - Local Clustering Coefficient of node 17 is 0

- Local Clustering Coefficient of node 18 is 0
- Local Clustering Coefficient of node 19 is 0.6666666666666666
- The final Clustring coefficient is 0.48284685086155676

Result/Costs:

Benchmark	**Prims Total cost in Mbps**	**Kruskal**	**Dijisktra (Choose any input node e.g. 5)**	**Bellman-Ford (Choose any input node e.g. 5)**	**Floyd Warshall Algorithm**	**Clustering Coefficient (Local Clustering)**
Input 10	24.45	24.45	52.8	52.8	453.6	0.6583
Input 20	51.45	51.45	122.25	122.25	2274.9	0.48284
Input 30	87.6	87.6	162.9	162.9	6271.79	0.69809
Input 40	137.1	137.1	344.55	344.55	13504.2	0.77097
Input 50	133.05	133.05	260.4	260.4	15542.09	0.6114
Input 60	201.15	201.15	557.249	557.249	28674.6	0.7103
Input 70	195.749	195.749	475.35	475.35	36364.5	0.68545
Input 80	249.3	249.3	427.35	427.35	50689.5	0.70086
Input 90	287.1	287.1	765.3	765.3	65643.0	0.78583
Input 100	304.5	304.5	693.149	693.149	80038.79	0.69942

Conclusion:

From the above table of results, we could conclude by comparing the time complexity of the algorithm of

same problem domain i.e MST or shortest path. For example, Prims and Kruskal can be compared

because they belong to the same problem domain and from the table, we can clearly see that both these algorithms produces the same bandwidth but thy differ in their time complexities. The time complexity

of prims is O(E + logV) (WHEN USING FIBONACCI HEAP)and kruskal is O(ElogV) so we can say that Prims

will perform better due to its time complexity.

On the other side when we talk about dijkstra or bellman ford for finding shortest path between two

given points their results are also same but they differ in their time complexities so on looking upon the

time complexity of dijkstra which is O(VlogV)(WHEN USING FIBONACCI HEAP) and bellman Ford which is

O(VE) we can conclude that dijkstra will perform better against the input provided by the user.

References:

1) https://docs.python.org/3/library/tk.html

2) https://www.youtube.com/watch?v=uqJZWLlnSqs

3) https://docs.scipy.org/doc/numpy-1.13.0/reference/arrays.ndarray.html

4) https://www.youtube.com/watch?v=ktyW-kOqGpY

5) https://www.youtube.com/watch?v=Uh2ebFW8OYM

6) https://graphviz.gitlab.io/documentation/

7) https://networkx.github.io/documentation/stable/reference/algorithms/shortest_paths.html

8) https://networkx.github.io/documentation/stable/reference/algorithms/clustering.html

9)https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.tree.mst.maximum_spanning_tree.html