**MINI PROJECT REPORT**

**Submitted to the faculty of Engineering and Technology**

**VI Semester**, **B. Tech**

***(Autonomous Batch)***

**IMPLEMENTATION OF ALL PAIRS SHORTEST PATH ALGORITHM**



BY

**A. VARUN**

**B17CS038**

Under the Guidance of

**Syed Abdul Moeed**

**Asst. Professor**

**Department of Computer Science & Engineering**

**Kakatiya Institute of Technology & Science**

**(An Autonomous Institute under Kakatiya University)**

**Warangal (Telangana State)**

**2020-21**

****

**C E R T I F I C A T E**

This is to certify that **A. VARUN** bearing roll no: **B17CS038** of the VI Semester B.Tech. Computer Science and Engineering (Autonomous) has satisfactorily completed the mini project entitled **“IMPLEMENTATION OF ALL PAIRS SHORTEST PATH ALGORITHM”.**

**Supervisor**

Syed Abdul Moeed

Asst. Professor

**Project Coordinator Head of the Department**

Dr. Kumar Dorthi V. Shankar

Asst. Professor

**ACKNOWLEDGMENT**

I extend my sincere and heartfelt thanks to our esteemed guide, **Syed Abdul Moeed, Asst. Professor** for his exemplary guidance, monitoring and constant encouragement throughout the course at crucial junctures and for showing us the right way.

I am grateful to respected coordinator **Dr. Kumar Dorthi, Asst. Professor** for permitting me to utilize all the necessary facilities of the Institute.

I would like to extend thanks to our respected head of the department, **Prof. V. Shankar** for allowing us to use the facilities available. I would like to thank other faculty members also.

I would like to thank my friends and family for the support and encouragement that they have given us during the mini project work.

1. **VARUN**

**B17CS038**

**ABSTRACT**

The ALL PAIR SHORTEST PATH is an algorithm for finding shortest path in weighted graph with positive or negative edge weights. A program with proper GUI which takes the number of nodes that must be created for the implementation of algorithm. And it must take the matrix values. And it checks the intermediate vertices cost and provides us the optimal path.

**LIST OF CONTENTS**

**Page No.**

**ABSTRACT i**

**CONTENTS ii**

**LIST OF FIGURES iii**

**CHAPTER 1 INTRODUCTION 01**

1.1 Introduction 01

1.2 Objectives of the Project 04

1.3 Applications 04

1.4 Literature Survey 04

**CHAPTER 2 DESIGN AND ANALYSIS 05**

2.1 Methodology 05

2.2. Software and Hardware Requirements 06

2.3 Flow Chart 06

**CHAPTER 3 IMPLEMENTATION 07**

3.1 Module-1 07

3.2 Module-2 08

**CHAPTER 4 TESTING AND RESULTS 09**

4.1 Number of Nodes Validation Testing 09

4.2 No Edge between Vertices Testing 11

4.3 Empty Text Field Warning Testing: 12

**CHAPTER 5 CONCLUSIONS** 12

5.1 Conclusion 12

5.1 Future Scope of Work 14

**REFERENCES 15**

**APPENDIX-1 SOURCE CODE 16**

**APPENDIX-3 SCREENSHOTS OF OUTPUTS 18**

**LIST OF FIGURES**

Fig. No. Title Page No.

1.1 A figure to explain All Pairs Shortest Path 04

4.1 A sample of number of nodes testing step-1 09

4.2 A sample of number of nodes testing step-2 10

4.3 A sample of number of nodes testing step-3 10

4.4 Giving a value zero if there is no edge between them 11

4.5 Changing its value to ∞ 11

4.6 Alert when text fields are empty 12

Appendix 6.1.1 GUI asking for number of nodes 19

Appendix 6.1.2 Entering number of nodes 20

Appendix 6.1.3 Asking for inputting matrix values 21

Appendix 6.1.4 Inputting matrix values 22

Appendix 6.1.5 Updated matrix values 23

**CHAPTER 1**

**INTRODUCTION**

* 1. **Introduction:**

SHORTEST PATH algorithms are the algorithms which are used to find the shortest path between the vertices.

There are two types of shortest path algorithms:

1.Single Source Shortest Path Algorithm

2.All Pairs Shortest Path Algorithm

**Single Source Shortest Path Algorithm:**

It is a type of algorithm in which it finds the shortest path between the vertex v and all the remaining vertices.

Ex: -1. Bellman–Ford Algorithm

2. Dijkstra's Algorithm with list

3. Dijkstra's Algorithm with binary heap

**All Pairs Shortest Path Algorithm:**

It is a type of algorithm in which it finds the shortest path between all the vertices of the graph.

Ex: -1. Floyd-Warshall Algorithm

2. Seidel's algorithm

3. Williams 2014

In this, we use Floyd-Warshall Algorithm to implement the ALL PAIRS SHORTEST PATH because of its optimality.

**1.1.1 ALL PAIRS SHORTEST PATH**

The ALL PAIRS SHORTEST PATH 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 (summed weights) of shortest paths between all pairs of vertices. Although it does not return details of the paths themselves, it is possible to reconstruct the paths with simple modifications to the algorithm. Versions of the algorithm can also be used for finding the transitive closure of a relation • or (in connection with the Schulze voting system) widest paths between all pairs of vertices in a weighted graph.

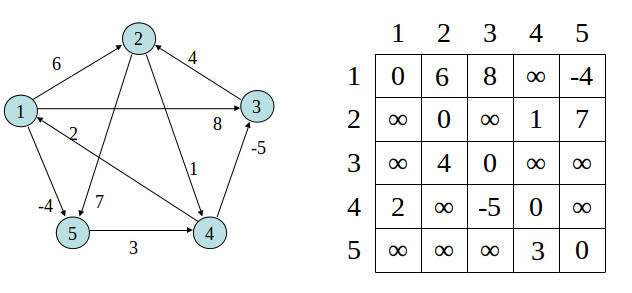


Figure 1.1-A figure to explain All Pairs Shortest Path

**ALGORITHM:**

• K is not an intermediate vertex in shortest path from I to j. We keep the value of dist[j] as it is

• K is an intermediate vertex in shortest path from I to j. we update the value of dist[i][j] as dist[i][k] +dist[k][j] if dist[i][j]>dist[i][j] +dist[k][j] the above figure shows the above optimal substructure property in the all-pair shortest path problem

1. Let dist be a |V| x |V| array) of minimum distances initialized to co (infinity)

2. for each edge (u, v)

3. dist[u][v] <----w (u, v) // the weight of the edge (u.,)

4. for each vertex v

5. dist[v][v] <--0

6. for k from 1 to |V|

7. for i from 1 to |V|

8. for j from 1 to |V|

9. if dist[i][j] > dist[i][k] + dist[k][j]

10. dist[i][j] <---dist[i][k] + dist[k][j]

11. end if

Input:

Graph [] [] = {{0, 5,2,1, i},

{3,0, 6, i,4}, {6, 7,0, 3,1}, {5,4, i,0, i}, {I,5, i, i, 0}}

Output:

0 5 2 1 3

3 0 5 4 0

6 6 0 3 1

5 4 7 0 8

8 5 10 9 0

**1.2 Objective of the Project**

To find the distance between every pair of nodes, decompose the given problem into subproblem and determine the shortest path between the vertices. Implementation of the algorithm and developing a Graphical User Interface using Java Programming.

**1.3 Applications**

* Shortest Path Algorithms are applied in finding the directions between two physical locations in mapping applications like Google Maps, Apple Maps etc.
* It is used in a non-deterministic abstract machine as a graph where vertices are described as states and edges are described as the possible transitions, shortest path algorithms can be used to find an optimal sequence of choices to achieve a certain goal. For example, if vertices represent the states of a puzzle like a Rubik's Cube and each directed edge corresponds to a single move or turn, shortest path algorithms can be used to find a solution that uses the minimum possible number of moves.
* It is also used in the inversion of matrices (Gauss-Jordan method).
* Computing the similarities between the graphs
* Optimal routing. In this application one is interested in finding the path with the maximum flow between two vertices. This means that, rather than taking minima as in the pseudocode above, one instead takes maxima.

**1.4 Literature Survey**

The Floyd–Warshall algorithm is an example of dynamic programming, and was published in its currently recognized form by Robert Floyd in 1962.However, it is essentially the same as algorithms previously published by Bernard Roy in 1959 and also by Stephen Warshall in 1962 for finding the transitive closure of a graph, and is closely related to Kleene's algorithm (published in 1956) for converting a deterministic finite automaton into a regular expression. The modern formulation of the algorithm as three nested for-loops was first described by Peter Ingerman, also in 1962.

**CHAPTER 2**

**DESIGN AND ANALYSIS**

**2.1 Methodology**

Step-1: Analysing the Problem Statement

Step-2: Gathering the requirements and Inputs

Step-3: Deciding the Flow of Control

Step-4: Division of the project into modules

Step-5: Create a JFrame, JPanel, JTextField and two JButtons and add them to the Container.

Step-6: Set the layout of the panel as the GridBagLayout to align the textboxes similar to that of matrix input

Step-7: Add Panel to the frame

Step-8: Use Textfield for taking the number of vertices as inputs

Step-9: Implement Button Event for the Button Pressed by implementing Button Listener with Action Listener

Step-10: Use GridBagConstraints for setting various attributes of the GridBagLayout

Step-11: The Method pack () is applied to the frame.

Step-12: Method panel. updateUI () is the method used.

**2.2 Software and Hardware Requirements**

**Software Requirements Specification:**

This section describes the implementation of algorithms using NetBeans software to find the minimum cost from source to destination by using a java language.

**Software Requirements:**

Microsoft Windows (XP/Vista/7/8/10)

NetBeans (or any other IDE with JDK installed in it).

**Hardware Requirements:**

Intel Atom or above

2 GB RAM

2 GB of available disk space

**2.3 Flow Chart**

**CHAPTER 3**

**IMPLEMENTATION**

**3.1 Module-1**

Module-1 is the ALGORITHM IMPLEMENTATION MODULE where n\*n matrix is given as the input to the algorithm where n is the number of vertices.

This Module Works as follows: -

In this module, the required inputs like number of vertices is stored in an integer variable, depending on that we store the corresponding edge values as the matrix values and store them in the form of 2-D array. We denote the case of not having an edge between two vertices in the matrix values is by assigning them the value of infinity, this means that there is no edge between the vertices.

In this we can determine the number of edges for a graph by the number of vertices in the graph. Number of edges possible is one more than number of vertices in the graph. Now we use loops for taking the matrix inputs and storing them in the 2-dimensional array. As it is the 2-dimensional array we need 2 loops for taking the input.

The important step in this module is that we need to check the costs between two vertices and also the cost if we traverse to the destination through the intermediate vertex. The least cost will be selected and the corresponding value is updated in the 2-dimensional array. This can be done programmatically by taking 3 loops because along with the source and destination vertex, the intermediate vertices values must also be checked

The case where the source vertex and destination vertex are the same must also be checked. There, the value “ZERO” must be assigned to the corresponding matrix value even in the case where the user inputs the non-zero value to that element.

**3.2 Module-2**

Module-1 is the GRAPHICAL USER INTERFACE MODULE where the text inputs are given as the cost of the inputs. First, a panel is where the UI is updated depending on the number of vertices which is provided by the user as the input.

Create a frame by instantiating an object of the class JFrame and select its Layout as the FlowLayout (). Create a label by instantiating an object of the class JLabeland add it to the frame.

Create a Panel with its Layout as the GridBadLayout () so that its convenient to create the Text Fields for matrix input. Create 2 buttons by instantiating an object of JButton class.

Next, ButtonListener is implemented by ActionListener and an event is created when the button is clicked. One Button, named “Generate I/P Matrix” is used for generating the row column matrix along with gridbag layout. This is updated on the panel when the button is clicked which leads to start of an event.

Similarly, another button is created for storing the values of a text fields into an array.

**CHAPTER 4**

**TESTING AND RESULTS**

**4.1 Number of Nodes Validation Testing:**

The program must notify the user if we provide the number of nodes input as any other character other than numbers.

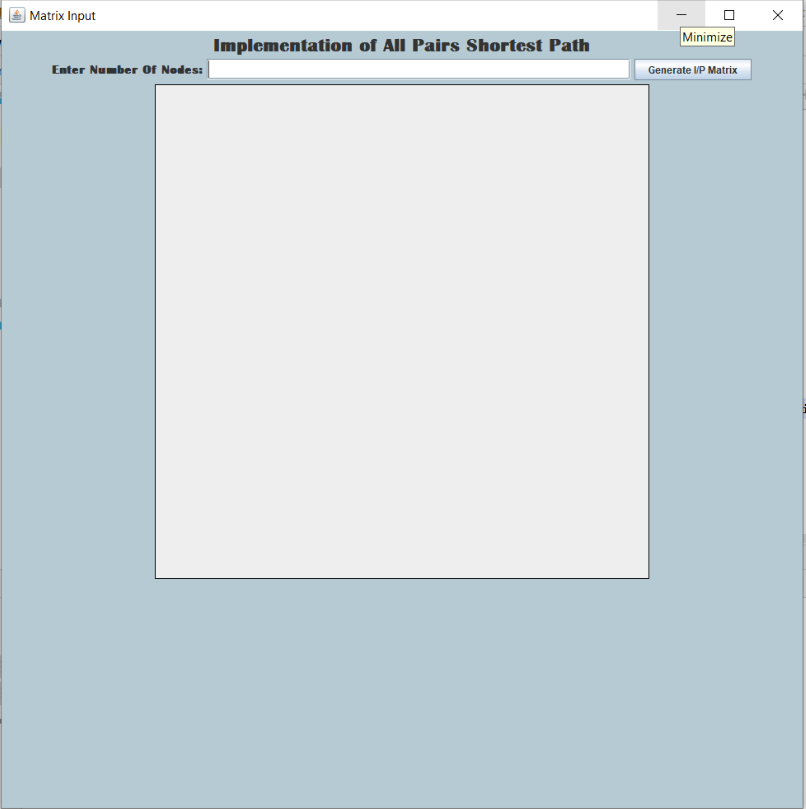


Figure 4.1: - A sample of number of nodes testing step-1

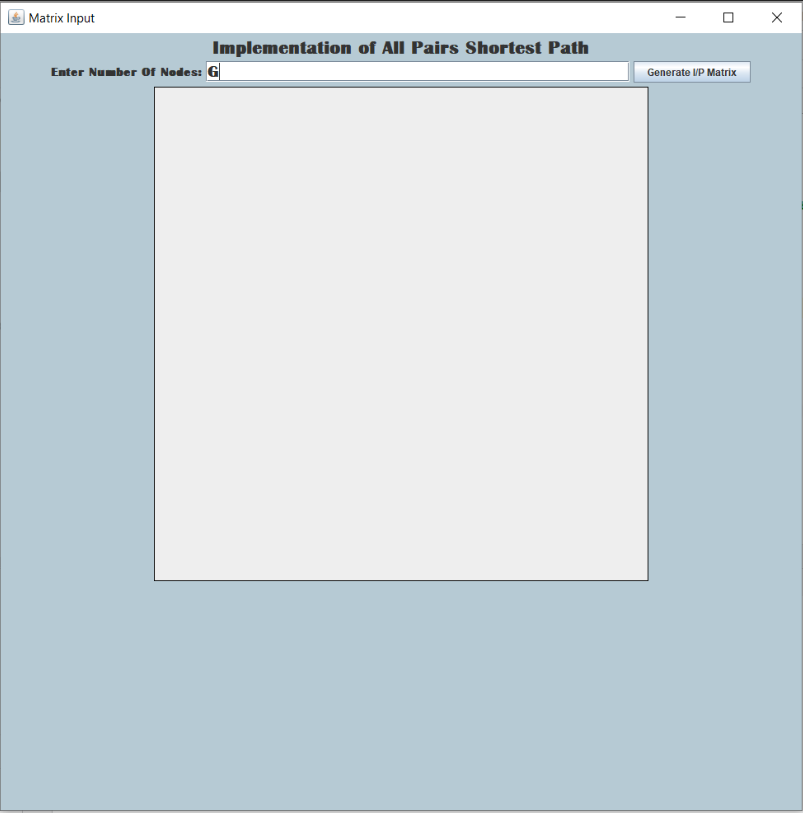


Figure 4.2: - A sample of number of nodes testing step 2

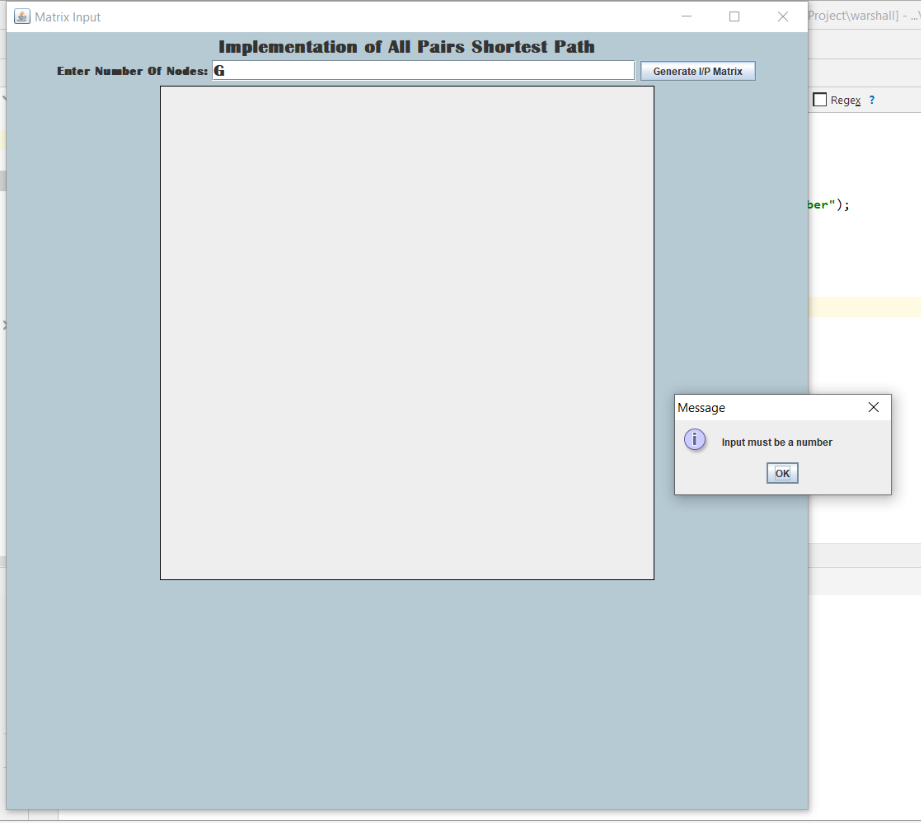


Figure 4.3: - A sample of number of nodes testing step 3

**4.2 No Edge between Vertices Testing:**

The program must make sure that when there is no edge between the vertices, the corresponding matrix must be considered as infinite.

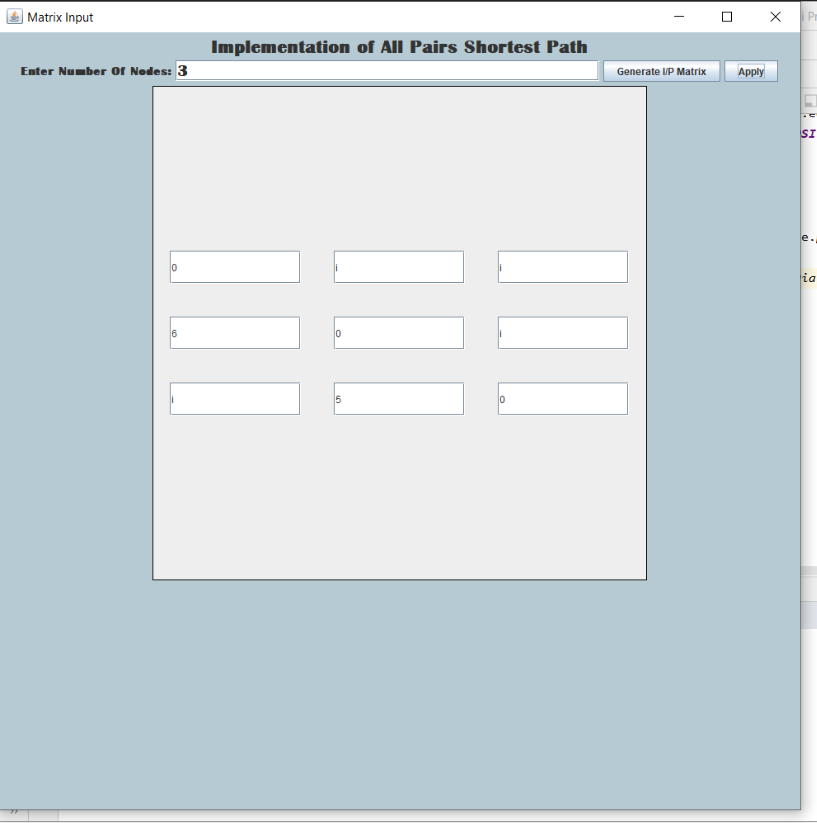


Figure 4.4-Giving a value zero if there is no edge between them

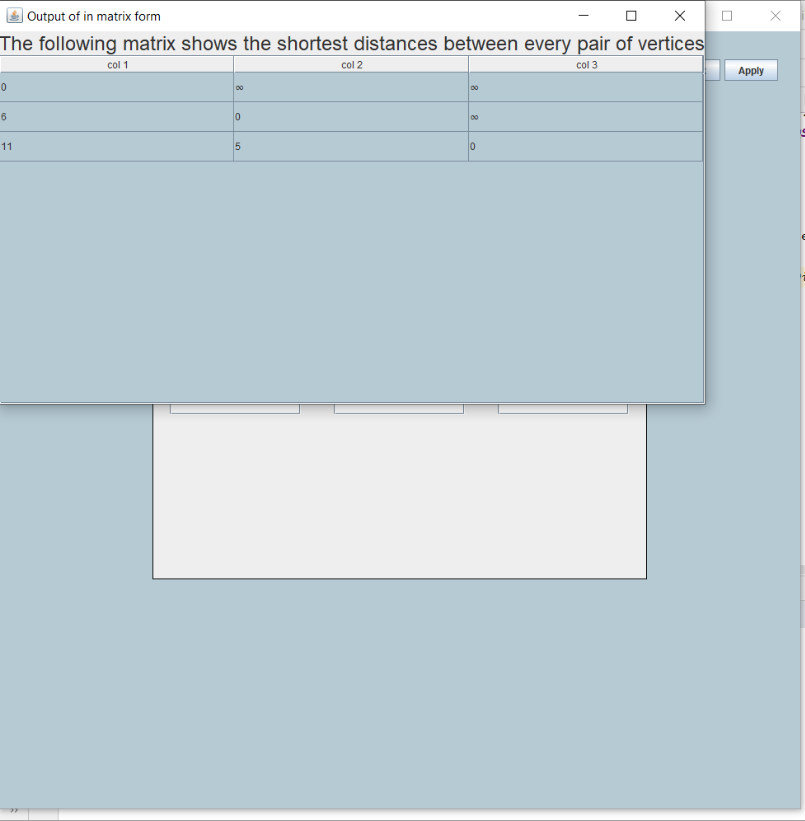


Figure 4.5-Changing its value to ∞

**4.3 Empty Text Field Warning Testing:**

The program must make sure that when there if in case the user leaves any of the text field empty, it most prompt the user that one or more text fields have been left empty.

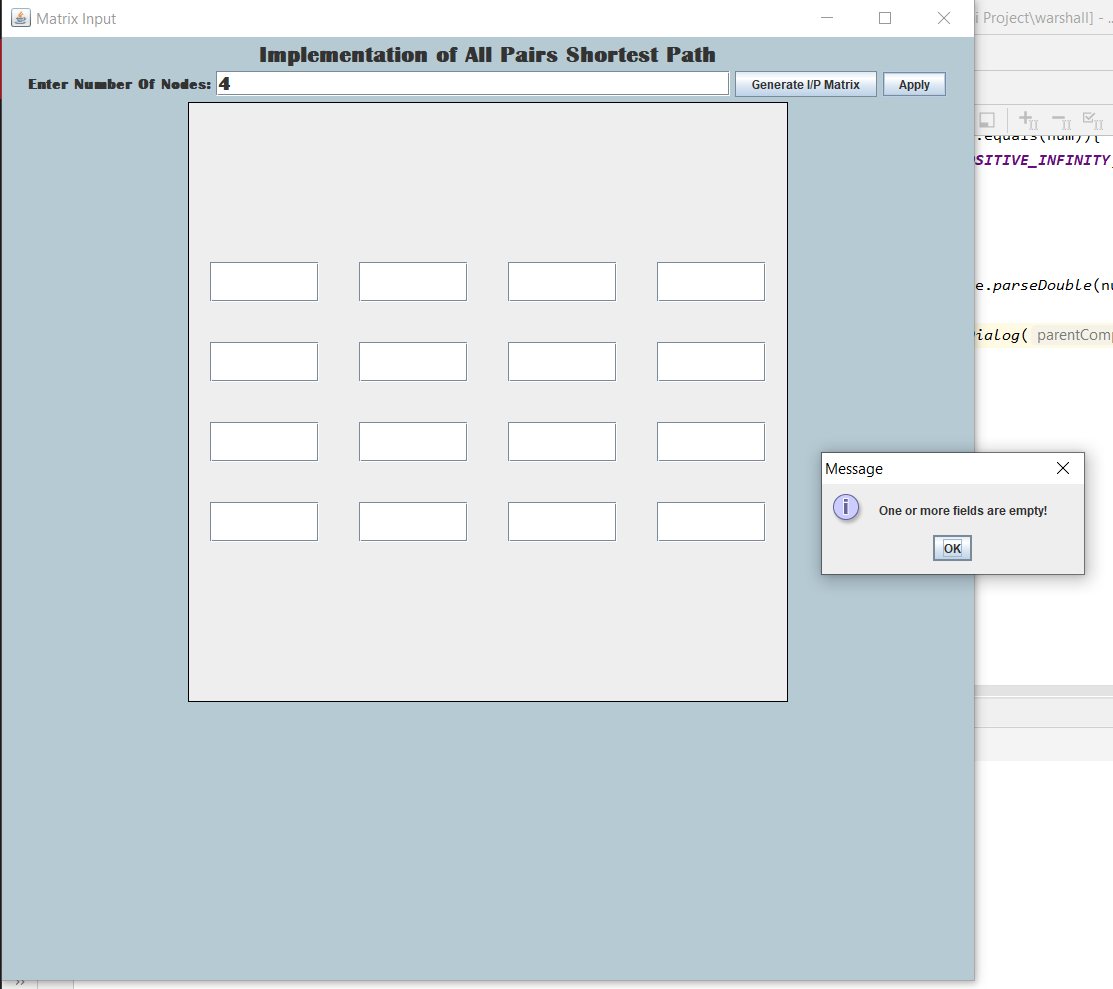


Figure 4.6-Alert when text fields are empty

**CHAPTER 5**

**CONCLUSION**

**5.1 Conclusions**

The Floyd Warshall Algorithm for the All Pairs Shortest Path is the better choice among the other algorithms for computing the paths between all pairs of vertices in which most of the vertices are connected by the edges.

Another important differentiating factor between the algorithms is their working towards distributed systems. Unlike Dijkstra’s algorithm, Floyd Warshall can be implemented in a distributed system, making it suitable for data structures such as Graph of Graphs (Used in Maps).

Floyd Warshall works for negative edge but not for negative cycle, whereas in case of Dijkstra algorithm it doesn’t work for negative edges.

**5.2 Future Scope of Work**

The Floyd Warshall Algorithm has wide range of applications such as finding the shortest path for the directions, finding the inversion of the matrix etc.

This algorithm can be implemented by adding some more functionalities such as simulating the process of algorithm (creation of a node, and creation of an edge between the 2 vertices). Using mouse events to select the source node and the destination node.

**REFERENCES**

# “Java the Complete Reference, 8th Edition”, by Herbert Schildt

# Rakesh Agarwal and Ramakrishnan Srikant "Fast algorithms for mining association rules in large databases”. Proceedings of the 20th International Conference on 1994 on Very Large Data Bases, pages 478-499, Santiago, Chile, September 1994

# Piatetsky-Shapiro, Gregory. "Discovery, analysis and presentation of strong rules." 3. Knowledge discovery in databases (1991): 229-248

# APPENDIX-I

**SOURCE CODE**

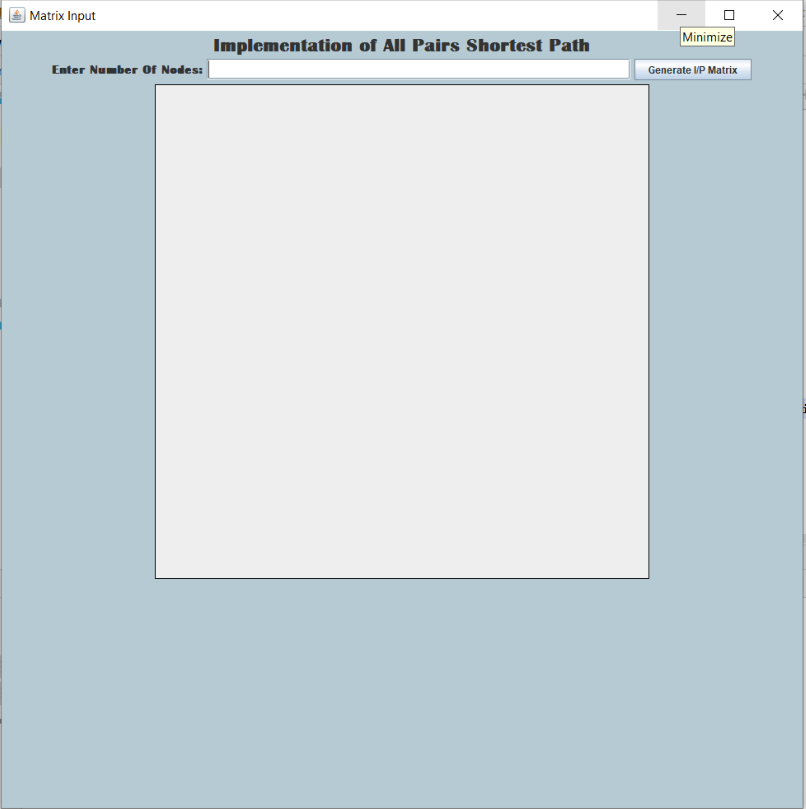
**Module-1:**

**import** warshall.APSP;  
**import** warshall.Test1;  
**import** java.awt.\*;  
**import** java.util.\*;  
**import** java.lang.\*;  
**import** java.io.\*;  
**import** javax.swing.\*;  
**import static** warshall.Test1.*rowCnt*;  
**public class** APSP1  
{  
 **final static** Double ***INF*** = 99999.0;  
 **public static** JFrame *frame*;  
 **static** String[] *columns* = **new** String[Test1.*rowCnt*];  
 **static** Object [][] *data*= **new** Object[Test1.*rowCnt*][Test1.*rowCnt*];  
  
 **void** floydWarshall(Double graph[][])  
 {  
 Double dist[][] = **new** Double[Test1.*rowCnt*][Test1.*rowCnt*];  
 **int** i, j, k;  
  
 **for** (i = 0; i < *rowCnt*; i++)  
 **for** (j = 0; j < *rowCnt*; j++)  
 dist[i][j] = graph[i][j];  
  
 **for** (k = 0; k < *rowCnt*; k++)  
 {**for** (i = 0; i < *rowCnt*; i++)  
 {**for** (j = 0; j < *rowCnt*; j++)  
 {**if** (dist[i][k] + dist[k][j] < dist[i][j])  
 dist[i][j] = dist[i][k] + dist[k][j];  
 }  
 }  
 }display(dist);  
 }  
  
 **void** display(Double dist[][]) {  
  
 **for** (**int** i = 0; i < *rowCnt*; i++) {  
 *columns*[i] = **"col"** + **" "** + (i + 1);  
  
 }  
  
 **for** (**int** i = 0; i< *rowCnt*; ++i)  
 {  
 **for** (**int** j = 0; j< *rowCnt*; ++j)  
 {  
 **if** (dist[i][j]==Double.***POSITIVE\_INFINITY***)  
 {  
 *data*[i][j]=**"∞"**;  
 }  
 **else** *data*[i][j]=(**int**)Math.*round*(dist[i][j]);  
  
 }}  
   
**Module-2:**

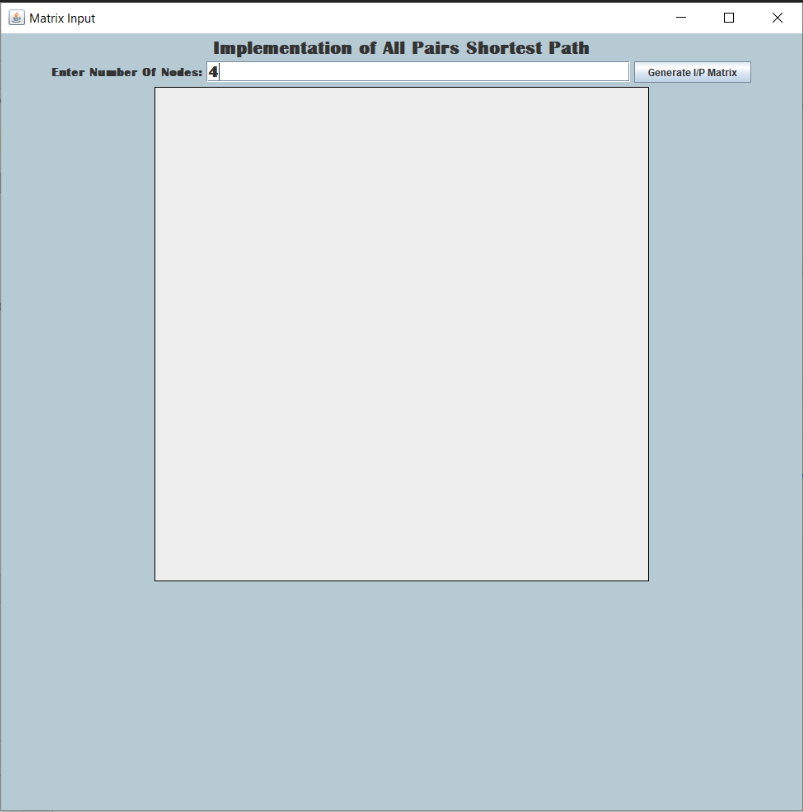
**public class** Test1  
{  
 *// Field members* **static** JPanel *panel* = **new** JPanel();  
 **static** Integer *indexer* = 1;  
 **static** List<JTextField> *listOfTextFields* = **new** ArrayList<JTextField>();  
 **static** JTextField *jtb1*;  
 **public static void** main(String[] args)  
 {  
*// Construct frame* JFrame frame = **new** JFrame();  
 frame.setLayout(**new** FlowLayout());  
 frame.setPreferredSize(**new** Dimension(990, 990));  
 frame.setTitle(**"Matrix Input"**);  
 frame.getContentPane().setBackground(**new** java.awt.Color(182, 202, 212));  
 frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
  
 JLabel jlb=**new** JLabel(**" "**+**"Implementation of All Pairs Shortest Path"**+**" "**,SwingConstants.***CENTER***);  
 JLabel jlb1=**new** JLabel(**"Enter Number Of Nodes:"**,SwingConstants.***LEFT***);  
 jlb.setFont(**new** Font(**"Britannic Bold"**,Font.***BOLD***,23));  
 jlb1.setFont(**new** Font(**"Britannic Bold"**,Font.***BOLD***,15));  
 frame.add(jlb);  
 frame.add(jlb1);  
  
 *// Frame constraints* GridBagConstraints frameConstraints = **new** GridBagConstraints();  
 *jtb1*=**new** JTextField(**""**,30);  
 *jtb1*.setFont(**new** Font(**"Britannic Bold"**,Font.***BOLD***,20));  
  *// Construct button* JButton addButton = **new** JButton(**"Generate I/P Matrix"**);  
 addButton.addActionListener(**new** ButtonListener());  
  
  
 *// Add button to frame* frameConstraints.**gridx** = 0;  
 frameConstraints.**gridy** = 0;  
 frame.add(*jtb1*);  
 frame.add(addButton, frameConstraints);  
  
 *// Construct panel  
 panel*.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 *panel*.setAlignmentY(Component.***BOTTOM\_ALIGNMENT***);  
 *panel*.setPreferredSize(**new** Dimension(600, 600));  
 *panel*.setLayout(**new** GridBagLayout());  
 *panel*.setBorder(**new** EmptyBorder(40,20,20,20));  
 *panel*.setBorder(LineBorder.*createBlackLineBorder*());  
  
  
 frame.add(*panel*); frame.pack();  
  
 frame.setVisible(**true**);  
 }

**APPENDIX-II**

**SCREENSHOTS OF OUTPUTS**

**

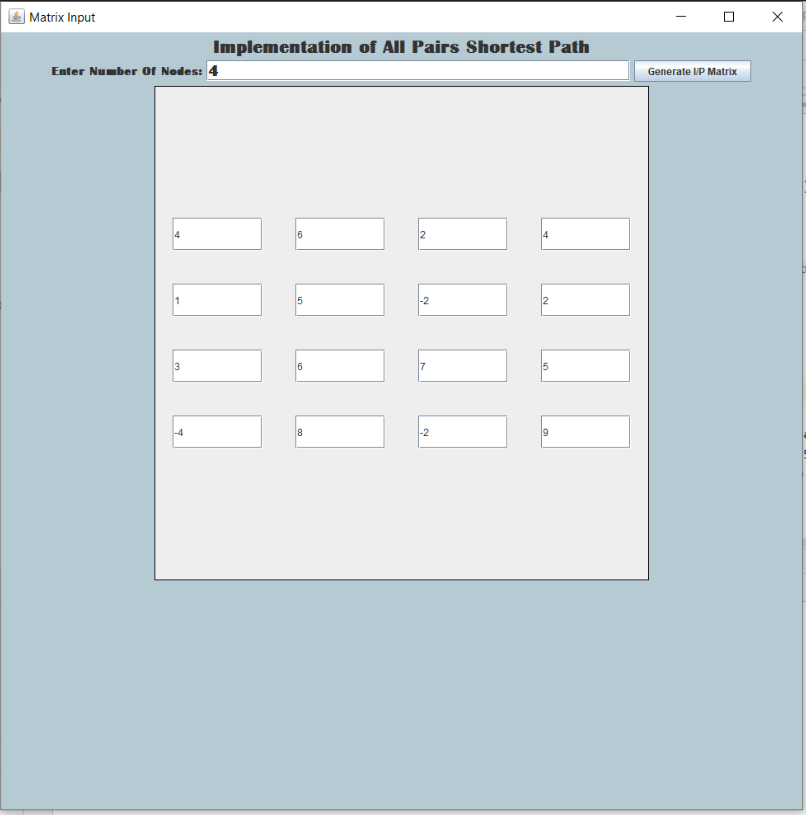
Appendix Figure 6.1.1 GUI asking for number of nodes



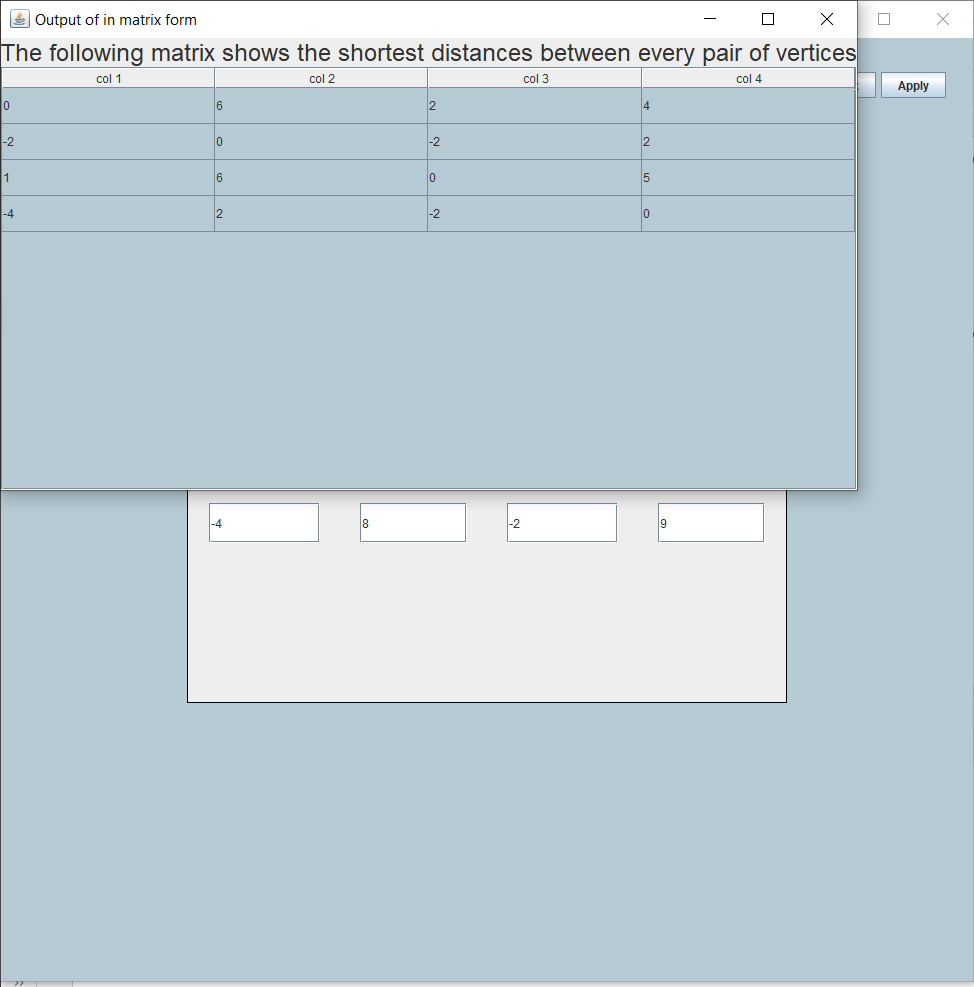
Appendix Figure 6.1.2 Entering number of nodes



Appendix Figure 6.1.3 Asking for inputting matrix values



Appendix Figure 6.1.4 Inputting matrix values

**

Appendix Figure 6.1.5 Updated matrix values