

A Project Report on

**GRASP**

**Graphical representation of algorithms**

**Structures and processes**

in partial fulfilment of the requirement for the award of the degree of

Bachelor of Technology

In

Computer Science & Engineering

Under the Supervision of

**Mr. Ankit Bansal (Assistant professor-CSE Department)**

By

**Rishabh Dubey (1203510083) Preeti Kumari (1203510072)**

Babu Banarsi Das Institute of Technology, Ghaziabad Dr.A.P.J Abdul Kalam Technical University, Uttar Pradesh, Lucknow

Session 2015-16

DECLARATION

I/We hereby declare that SRS entitled”

……………………………………………………………..….……………………………

…………… ……… ………..”Which is submitted by me/us in partial fulfillment of the requirement for the award of degree B.Tech. in Computer Sc. & Engineering to BBDIT, Ghaziabad affiliated to Dr. A.P.J Abdul Kalam Technical University, Uttar Pradesh, Lucknow comprises only my/our own work and due acknowledgement has been made in the text to all other material used.

*Date: Name(s) of Student(s):*

Certificate

This is to certify that SRS entitled “……………………………………………....” which is submitted by ………………………… …………………….. ………………….in partial fulfillment of the requirement for the award of degree of Bachelor of Technology in Computer Science & Engineering from BBDIT, Ghaziabad affiliated to Dr. A.P.J Abdul Kalam Technical University, Uttar Pradesh, Lucknow is a record of the candidates’ own work carried out by them under my supervision. The matter embodied in this report is original and has not been submitted for the award of any other degree.

**Date: -**

**Project Guide: Head of Department:**

Mr/Ms Dr. Amit Singhal Designation Professor Department of CSE Department of CSE

**ACKNOWLEDGEMENT**

It is a matter of immense pleasure and heartfelt gratitude for us to present report of the project GRASP (General representation of algorithm structure and processes) as our final year project in B.Tech (CSE Department). There are many people we would like to thanks for their help in writing this report and implementing this concept. At the outset we thank for their help in writing reports. We are very thankful to **Mr Kedarnath Singh** (Assistant professor) and **Mr Sachin Minocha** (Assistant professor) in BBDIT, Ghaziabad for assigning us on this project and believing in us. I am very much thankful to **Mr Ankit Bansal** (Assistant professor) in BBDIT, Ghaziabad who give me guideline and provide me good suggestion. Thanks you for sharing his time and knowledge withus and helped in completing this project successfully. Thanks also to our colleagues who let us know about the projects.

We are also thankful to our head of department **Mr Amit Singhal**

**(HOD, CSE Department)** and we are also thankful to our department faculty for encouraging us and guiding through the projects.

We are very much grateful to our parents for their support and faith

they showed in us. Last but not the least our friends for being there round the clock whenever we needed them.

**Rishabh Dubey**

**Preeti Kumari**

**ABSTRACT**

The GRASP project (Graphical Representations of Algorithms Structures and Processes) has successfully created and prototyped a new algorithmic level graphical representation for software, the graphical Diagram. The primary impetus for creation of the graphical representation was to improve the comprehension efficiency of algorithms and, as a result, improve reliability and reduce costs.

The emphasis has been on the automatic generation of the graphical diagram. The graphical constructs were created and applied manually to several small algorithm programs. A user interface was designed and partially implemented. The user interface provides the user, prototype for the generation of graphical

diagram of the algorithm and they can see the summary of designing of graphical diagram.

It shows the shortest path and other modules as per algorithm basis.The basic purpose of GRASP is graphical representation. In this project user can see output of the graph in graphical form as well as in summary form. This is what gives a better understanding for user to learn about algorithms.

**DECLARATION CERTIFICATE**

**TABLE OF CONTENT**

**ACKNOWLEDGEMENT ABSTRACT**

**LIST OF FIGURES**

**IST OF SYMBOLS, ABBREVIATIONS**

**1. INTRODUCTION**

1.1 GRASP………………………………………………………………………...8

1.2 Purpose……………………...…………………………………………………8

1.3 Scope…………..……………………………………………............................8

1.4 Application Design…………………………………………………………….8

1.5 Overview……………………………………………………………………….8

**2. OVERALL DESCRIPTION**

2.1 Product Perspective…………………………………………………………….9

2.2 Software Interface……………………………………………………………...10

2.3 Hardware Interface……………………………………………………………..10

2.4 Function model and description ………..……………………………..……….10

2.4.1 Graphical input ….……………………………………………………….11

2.4.2 Algorithm selection………………………………………………………11

2.4.3 Help/Tutorial……………………………………………………………..11

2.4.3.1 BFS (Breadth first search) …...……… ……..……………………12

2.4.3.2 DFS (Depth first search) ...………………………………………..12

2.4.3.3 Dijikstra algorithm…………………………..……………………13

2.4.3.4 Prims algorithm…………….……………………………………...14

2.4.3.5 Kruskal algorithm………………………………………...……......15

2.4.3.6 Topological Sort……………………………………...……………15

2.4.3.7 Longest path algorithm……………………………...……………..16

2.4.3.8 Maximum flow algorithm…………………………..……………..16

2.4.3.9 Critical path analysis……………….………………….………..…17

2.4.3.10 Graph Colouring……….………………………..…………...…...17

2.4.3.11 Travelling Salesman……………………………………………...18

2.4.3.12 Hamilton tour……………..……………………………….……..18

2.4.4 Complexity……………………..………………………………………......18

2.4.4.1 Time Complexity………………………………………………….19

2.4.4.2 Space Complexity…………...…………………………………….19

2.4.5 Description module…………….………………………………………….20

2.4.5.1 Summarisation…………………………………………………….20

2.4.5.2 Graphical output…………………………………………………..20

2.5 Constraints………………………………………………………………………21

2.6 Use Case Model Description……………………………………………………21

2.7 Sequence Diagram…………………..…………………………………………..22

2.8 Data Flow Diagram………………..……………………………………………23

2.8.1 Level-0 DFD………..………………………..…………………………..22

2.8.2 Level-1 DFD…………..……………………………………..…………..23

2.8.3 Level-2 DFD……………..………………………………...…………….24

2.9 Software interface description…..…………………………………………..….24

2.9.1 Proposed system…….………..………………………………………….24

2.9.2 Advantage of proposed system…..…..………………………………......24

2.10 Assumption and Dependencies………………………………………………..24

2.11 Database Design……………………………………………………………….25

2.11.1 ER Diagram…………...………………………………………………..25

**3. SPECIFIC REQUIREMENTS**

3.1 Use Case Reports……………………………………………………………….25

3.2 Feasibility Study………………………………………………………………..26

3.2.1 Economic Feasibility……………………………………………………..26

3.2.2 Operational Feasibility…………………………………………………....26

3.2.3 Technical Feasibility……………………………………………………...26

**4. TESTING AND MANTINANCE**

4.1 Introduction……………………………………………………………………..27

4.2 Type of Testing………………………………………………………………....27

4.2.1 Unit testing……………………………………………………………….27

4.2.2 Integration Testing……………………………………………………….27

4.2.3 Functional Testing……………………………………………………......27

4.2.4 System Testing…………………………………………………………...28

4.2.5 White Box Testing…………………………………………………….....28

4.2.6 Black Box Testing……………………………………………………..…28

4.3 Unit Testing……………………………………………………………………..28

4.4 Integration Testing……………………………………………………………...29

4.5 Acceptance Testing……………………………………………………………..29

**5. SAMPLE CODE**

5.1 Graphical Input………………………………………………………………….30

5.2 Graphical Output………………………………………………………………..33

**6. SNAPSHOTS**……………………………………………………………………37

Input screen after running projects

Input screen after creating node by mouse click

Input screen after creating edges between each nodes

Graphical output

Choice for algorithm selection Output in form of summary Snapshots of the coding

Snapshots for GRASP problem domain

**7. APPENDICES**

A. Bibliography/ List of reference…………………………………………………56

B. List of figures……………………………………………………………………56

C. List of tables……………………………………………………………………..56

**1. Introduction: -**

**1.1 GRASP: -**GRASP stands for graphical representations of algorithms structures and processes. The main purpose of GRASP is to show the graphical representation of the algorithm of DAA (Design and analysis of algorithm). The purpose of designing GRASP is to provide a tutorial series for how the algorithm processed from one node to another. How they determine the shortest path between each node?

**1.2 Purpose: -** Our goal for the implementation of GRASP (graphical representation of algorithms structures and processes) is to provide a tutorial series for beginning level user to understand basic concepts of the graph theory or algorithms representation. In this we are planning to provide a graphical way to understand algorithms from graphical representation.

 User can also learn about algorithms from here how algorithms works.

 Lack of understanding in the algorithm in theoretical way.

 To reduce paper work.

 To save time.

 It will be easy to learn algorithm by GRASP.

**1.3 Scope: -** The scope of the GRASP is good in the following ways.

 This project can be used broadly to learn about algorithms because it will be useful in learning the algorithm easily from graphical representation. In GRASP we are implementing the following algorithms:

o BFS ([Breadth-first search)](https://en.wikipedia.org/wiki/Breadth-first_search)

o DFS (Depth-first search)

o Dijikstra algorithm

o Prims algorithm

o Kruskal algorithm

o Topological sort

o Euler tour

o Longest path

o Maximum flow

o Critical path analysis

o Graph colouring

o [Travelling salesman problem](https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CBwQFjAAahUKEwis9aCl3MLIAhXEno4KHTrsCUQ&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FTravelling_salesman_problem&usg=AFQjCNHHeqQL_wgjok2-NTUVuoNOORofXw&bvm=bv.105039540,d.c2E)

o Hamilton tour

 It is reusable.

 It can be enhanced if required to add some extra features.

**1.4 Application Design: -** GRASP is a software from where beginner level user can learn about the execution of algorithm and the process of execution of algorithms. It will provide a learning platform for users. It will show the graphical output of the graphical input given by the user.

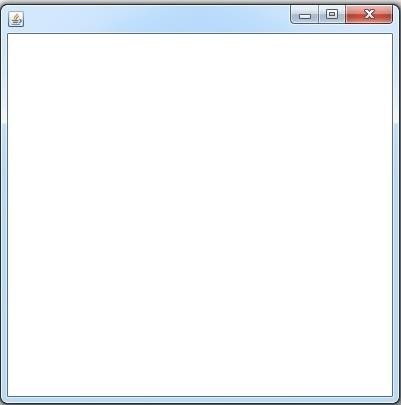
**1.5 Overview: -** GRASP is nothing but a learning platform for the user. It will be beneficial for beginners because they can learn about basics of algorithm and the execution of Algorithm. GRASP takes graph as input. It includes various modules like

Graphical input, algorithm selection, help/tutorial, complexity, graphical output. In first module GRASP take graph as input. In algorithm selection module there are various algorithms from which user can run their algorithms. In help/tutorial modules it will include tutorial and algorithm of the various algorithms. Complexity module includes the complexity of various algorithm in best case and worst case scenario. It will show the graphical output along with the summarisation of each step processed in the algorithms.

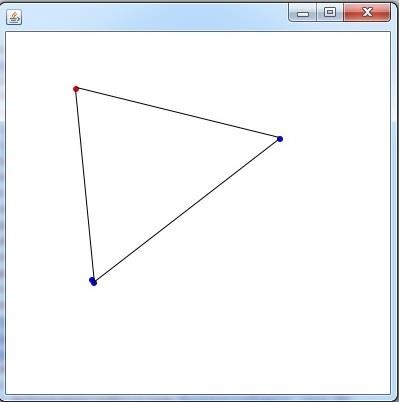
**2. Overall Description: -**

**2.1 Product Perspective: -** The main perspective of our project is to give output in the

graphical form. As you can see after running project the following screen popup from where it takes input by mouse in first screen you are seeing a GUI interface. When you click on mouse it will create a node. When you created enough nodes than click on a node and drag line till another node. Make sure that the click on middle of nodes otherwise it will create a new node.



 **Cross platform support:** Offers operating support for most of the known and commercial operating system.



 **Tutorial series:** It will include the tutorial series of the algorithms in the form of pdf and videos.

 **Complexity:** It will show the complexity in each case of the algorithms.

**2.2 Software Interface:**

This project GRASP will run on windows and any other operating system. It can be accessed

as open source software when it will be deployed on the server.

 **Operating System: -** Windows (any version), Linux, Mac and other OS also.

 **Browser: -** Google chrome, Mozilla Firefox, Internet Explorer and others.

 **Web Server:** This software is being designed to run on Apache Tomcat Server 7.2.1.

 **Database:** The system will access MYSQL database developed by Microsoft for providing the backend support for various feature included later like tutorial series.

|  |  |
| --- | --- |
| Operating system | Windows, Linux, Mac etc. |
| Front end | Java |
| Back end | MySQL |
| Documentation | Microsoft word 2013 |
| Presentation | Microsoft PowerPoint 2013 |

**2.3 Hardware Interface: -**

 Device will work when the java environment variable is set up in the device.

 The hardware interface of the system is handled by the windows server operating system. No hardware dependent code will be written by the development team.

**Technology Used:**

 **J2EE:** J2EE is a platform independent, java centric environment from Oracle for developing, building and deploying web based enterprise application online.

 **HTML:** It will be used to develop tutorial series page in the project and giving eye catching look to this site.

 **MySQL:** It will be used to store the pdf and video files of tutorial series after adding these modules.

**Performance requirements:**

 **Client Hardware:** The performance of system will also depend on the client

machine.

 **Java Supported Device:** For supporting the GRASP it must be java supportable.

**2.4 Functional Model and Description:** - The GRASP contains the following modules as given below: -

1. Graphical Input

2. Algorithm Selection

3. Help/Tutorial

4. Complexity

5. Description module

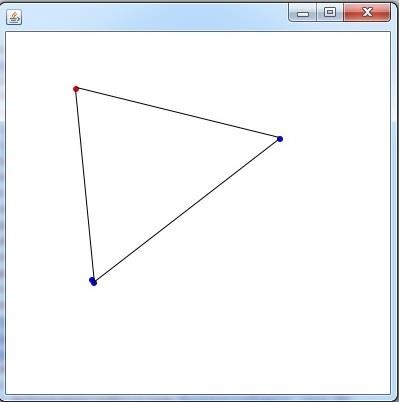
6. Summarisation

7. Graphical output

**2.4.1 Graphical Input: -**

In graphical input we are using mouse click for giving the input. By clicking mouse user can

draw any graph and its nodes by assigning weight from one node to another. For this purpose, we will be using some of the JAVA API. The graphical input will be seen like the below diagram.



**2.4.2 Algorithm Selection: -**

After giving graphical input user can select the algorithm from which they want to execute

their graph. In this module following algorithm has been included from which they can execute their graph.

 BFS ([Breadth-first search)](https://en.wikipedia.org/wiki/Breadth-first_search)

 DFS (Depth-first search)

 Dijikstra algorithm

 Prims algorithm

 Kruskal algorithm

 Topological sort

 Euler tour

 Longest path

 Maximum flow

 Critical path analysis

 Graph colouring

 [Travelling salesman problem](https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CBwQFjAAahUKEwis9aCl3MLIAhXEno4KHTrsCUQ&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FTravelling_salesman_problem&usg=AFQjCNHHeqQL_wgjok2-NTUVuoNOORofXw&bvm=bv.105039540,d.c2E)

 Hamilton tour

**2.4.3 Help/Tutorial:-**

In this module user can get help how the process work and in this section user can see the

algorithm of various shorting algorithm.

**2.4.3.1 BFS** [**(Breadth-first search)**](https://en.wikipedia.org/wiki/Breadth-first_search)

Breadth-first search (BFS) is a[n algorithm for](https://en.wikipedia.org/wiki/Algorithm) traversing or searching [tree](https://en.wikipedia.org/wiki/Tree_data_structure) or [graph](https://en.wikipedia.org/wiki/Graph_(data_structure)) data

structures. It starts at the [tree root](https://en.wikipedia.org/wiki/Tree_(data_structure)#Terminology) (or some arbitrary node of a graph, sometimes referred to as a 'search key') and explores the neighbour nodes first, before moving to the next level neighbours.

**Breadth-First-Search (G, v):**

1. for each node n in G:

2. n.distance = INFINITY

3. n.parent = NIL

4. create empty queue Q

5. v.distance = 0

6. Q.enqueue(v)

7. while Q is not empty:

8. u = Q.dequeue()

9. for each node n that is adjacent to u:

10. if n.distance == INFINITY:

11. n.distance = u.distance + 1

12. n.parent = u

**13.** Q.enqueue(n)

**Complexity:**

Worst case performance: O(|E|)

Worst case space complexity: O(|V|)

**2.4.3.2 DFS (Depth-first search): -**

Depth-first search (DFS) is an [algorithm](https://en.wikipedia.org/wiki/Algorithm) for traversing or searching [tree](https://en.wikipedia.org/wiki/Tree_data_structure) or [graph](https://en.wikipedia.org/wiki/Graph_(data_structure)) data

structures. One starts at [the root (s](https://en.wikipedia.org/wiki/Tree_(data_structure)#Terminology)electing some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before [backtracking.](https://en.wikipedia.org/wiki/Backtracking)

A recursive implementation of DFS

1. procedure DFS(G,v):

2. label v as discovered

3. for all edges w in G.adjacentEdges(v) do

4. if vertex w is not labeled as discovered

**5.** then recursively call DFS(G,w)

A non-recursive implementation of DFS:

1. procedure DFS-iterative(G,v):

2. let S be a stack

3. S.push(v)

4. while S is not empty

5. v = S.pop()

6. if v is not labeled as discovered:

7. label v as discovered

8. for all edges from v to w in G.adjacentEdges(v) do

9. S.push(w)

**Complexity:**

Worst case performance: O(|E|)

Worst case space complexity: O(|V|)

**2.4.3.3 Dijikstra algorithm: -**

Dijkstra's algorithm is an [algorithm for](https://en.wikipedia.org/wiki/Algorithm) finding t[he shortest paths b](https://en.wikipedia.org/wiki/Shortest_path_problem)etwee[n nodes in](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) a [graph,](https://en.wikipedia.org/wiki/Graph_(abstract_data_type)) which may represent, for example, road networks. Dijkstra's original variant found the shortest path between two nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a [shortest path tree.](https://en.wikipedia.org/wiki/Shortest_path_tree)

For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, if the nodes of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path algorithm is widely used in netw[ork routing protocols,](https://en.wikipedia.org/wiki/Routing_protocol) most notably [IS-IS a](https://en.wikipedia.org/wiki/IS-IS)nd Open Shortest Path First [(OSPF).](https://en.wikipedia.org/wiki/OSPF)

In the following algorithm, the code u ← vertex in *Q* with min dist[u], searches for the vertex *u* in the vertex set *Q* that has the least dist[*u*] value. Length (*u*, *v*) returns the length of the edge joining (i.e. the distance between) the two neighbor-nodes *u* and *v*. The variable *alt* on line 19 is the length of the path from the root node to the neighbor node *v* if it were to go through *u*. If this path is shorter than the current shortest path recorded for *v*, that current path is replaced with this *alt* path. The prev array is populated with a pointer to the "next-hop" node on the source graph to get the shortest route to the source.

**function Dijkstra(Graph, source):**

1. create vertex set Q

2. for each vertex v in Graph: // Initialization

3. dist[v] ← INFINITY // Unknown distance from source to v

4. prev[v] ← UNDEFINED // Previous node in optimal path from source

5. add v to Q // All nodes initially in Q (unvisited nodes)

6. dist[source] ← 0 // Distance from source to source

7. while Q is not empty:

8. u ← vertex in Q with min dist[u] // Source node will be selected first

9. remove u from Q

10. for each neighbor v of u: // where v is still in Q.

11. alt ← dist[u] + length(u, v)

12. if alt < dist[v]: // A shorter path to v has been found

13. dist[v] ← alt

14. prev[v] ← u

15. return dist[], prev[]

If we are only interested in a shortest path between vertices *source* and *target*, we can terminate the search after line 13 if *u* = *target*. Now we can read the shortest path from *source* to *target* by reverse iteration:

1. S ← empty sequence

2. u ← target

3. while prev[u] is defined: // Construct the shortest path with a stack S

4. insert u at the beginning of S // Push the vertex onto the stack

5. u ← prev[u] // Traverse from target to source

6. insert u at the beginning of S // Push the source onto the stack

**Complexity:**

Worst case performance: O(|E|+|V|log|V|)

**2.4.3.4 Prims algorithm: -**

Prim's algorithm is a [greedy algorithm that](https://en.wikipedia.org/wiki/Greedy_algorithm) finds [a minimum spanning tree f](https://en.wikipedia.org/wiki/Minimum_spanning_tree)or

a [weig](https://en.wikipedia.org/wiki/Weighted_graph)[hted undirected graph.](https://en.wikipedia.org/wiki/Undirected_graph) This means it finds a subset of the [edges](https://en.wikipedia.org/wiki/Edge_(graph_theory)) that forms [a tree that](https://en.wikipedia.org/wiki/Tree_(graph_theory)) includes every [vertex, w](https://en.wikipedia.org/wiki/Vertex_(graph_theory))here the total weight of all the [edges](https://en.wikipedia.org/wiki/Graph_theory) in the tree is minimized. The algorithm operates by building this tree one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.

Algorithm:

The algorithm may informally be described as performing the following steps: Initialize a tree with a single vertex, chosen arbitrarily from the graph.

1. Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree.

2. Repeat step 2 (until all vertices are in the tree).

In more detail, it may be implemented following the [pseudo code](https://en.wikipedia.org/wiki/Pseudocode) below.

**1.** Associate with each vertex *v* of the graph a number *C*[*v*] (the cheapest cost of a connection to *v*) and an edge *E*[*v*] (the edge providing that cheapest connection). To initialize these values, set all values of *C*[*v*] to +∞ (or to any number larger than the maximum edge weight) and set each*E*[*v*] to a special [flag value](https://en.wikipedia.org/wiki/Flag_value) indicating that there is no edge connecting *v* to earlier vertices.

**2.** Initialize an empty forest *F* and a set *Q* of vertices that have not yet been included in *F* (initially, all vertices).

**3.** Repeat the following steps until *Q* is empty:

a. Find and remove a vertex *v* from *Q* having the minimum possible value of *C*[*v*]

b. Add *v* to *F* and, if *E*[*v*] is not the special flag value, also add *E*[*v*] to *F*

c. Loop over the edges *vw* connecting *v* to other vertices *w*. For each such edge, if *w* still belongs to *Q* and *vw* has smaller weight than *C*[*w*], perform the following steps:

i. Set *C*[*w*] to the cost of edge *vw*

ii. Set *E*[*w*] to point to edge *vw*.

**4.** Return *F*

**2.4.3.5 Kruskal algorithm: -**

Kruskal's algorithm is a minimum-spanning-tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest.[[1] I](https://en.wikipedia.org/wiki/Kruskal%27s_algorithm#cite_note-:0-1)t is a [greedy algorithm in](https://en.wikipedia.org/wiki/Greedy_algorithm) [graph theory a](https://en.wikipedia.org/wiki/Graph_theory)s it finds [a minimum spanning tree f](https://en.wikipedia.org/wiki/Minimum_spanning_tree)or a [connected](https://en.wikipedia.org/wiki/Connectivity_(graph_theory)) [weighted graph a](https://en.wikipedia.org/wiki/Glossary_of_graph_theory#Weighted_graphs_and_networks)dding increasing cost arcs at each step.[[1] T](https://en.wikipedia.org/wiki/Kruskal%27s_algorithm#cite_note-:0-1)his means it finds a subset of the [edges](https://en.wikipedia.org/wiki/Edge_(graph_theory)) that forms a tree that includes every [vertex,](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) where the total weight of all the edges in the tree is minimized. If the graph is not

connected, then it finds a *minimum spanning forest* (a minimum spanning tree for each [connected component).](https://en.wikipedia.org/wiki/Connected_component_(graph_theory))

**Algorithm:**

KRUSKAL(G):

1. A = ∅

2. foreach v ∈ G.V:

3. MAKE-SET(v)

4. foreach (u, v) ordered by weight(u, v), increasing:

5. if FIND-SET(u) ≠ FIND-SET(v):

6. A = A ∪ {(u, v)}

7. UNION(u, v)

8. return A

**2.4.3.6 Topological sort: -**

In the field [of computer science, a](https://en.wikipedia.org/wiki/Computer_science) topological sort (sometimes abbreviated topological sort)

or topological ordering of [a directed graph](https://en.wikipedia.org/wiki/Directed_graph) is a linear ordering of its [vertices su](https://en.wikipedia.org/wiki/Vertex_(graph_theory))ch that for every directed edge *uv* from vertex *u* to vertex *v*, *u* comes before *v* in the ordering. For instance, the vertices of the graph may represent tasks to be performed, and the edges may represent constraints that one task must be performed before another; in this application, a topological ordering is just a valid sequence for the tasks. A topological ordering is possible if and only if the graph ha[s no directed cycles](https://en.wikipedia.org/wiki/Directed_cycle)[, that is, if it is a directed acyclic graph (](https://en.wikipedia.org/wiki/Directed_acyclic_graph)DAG). Any DAG has at least one topological ordering, and [algorithms a](https://en.wikipedia.org/wiki/Algorithm)re known for constructing a topological ordering of any DAG in [linear](https://en.wikipedia.org/wiki/Linear_time) time.

For topological sort there are many algorithms like Kahn’s algorithm or Tarzan’s algorithm.

Kahn’s algorithm

L ← Empty list that will contain the sorted elements

S ← Set of all nodes with no incoming edges

1. while S is non-empty do

2. remove a node n from S

3. add n to tail of L

4. for each node m with an edge e from n to m do

5. remove edge e from the graph

6. if m has no other incoming edges then

7. insert m into S

8. if graph has edges then

9. return error (graph has at least one cycle)

10. else

11. return L (a topologically sorted order)

**Complexity: -**

The usual algorithms for topological sorting have running time linear in the number of nodes plus the number of edges, asymptotically O(|V|+|E|).

**2.4.3.7 Longest path algorithm: -**

The longest path problem is the problem of finding a sim[ple path](https://en.wikipedia.org/wiki/Path_(graph_theory)) of maximum length in a given

graph. A path is called simple if it does not have any repeated vertices; the length of a path may either be measured by its number of edges, or (in [weighted graphs)](https://en.wikipedia.org/wiki/Weighted_graph) by the sum of the weights of its edges. In contrast to the [shortest path problem,](https://en.wikipedia.org/wiki/Shortest_path_problem) which can be solved in polynomial time in graphs without negative-weight cycles, the longest path problem is [NP-hard,](https://en.wikipedia.org/wiki/NP-hard) meaning that it cannot be solved in [polynomial time for](https://en.wikipedia.org/wiki/Polynomial_time) arbitrary graphs unless [P = NP.](https://en.wikipedia.org/wiki/P_%3D_NP) Stronger hardness results are also known showing that it is difficult to [approximate.](https://en.wikipedia.org/wiki/Approximation_algorithm) However, it has a [linear time solu](https://en.wikipedia.org/wiki/Linear_time)tion for [directed acyclic graphs,](https://en.wikipedia.org/wiki/Directed_acyclic_graph) which has important applications in finding the [critical](https://en.wikipedia.org/wiki/Critical_path_method) path in scheduling problems.

**NP-hardness: -**

The NP-hardness of the un-weighted longest path problem can be shown using a reduction

from [the Hamiltonian path problem:](https://en.wikipedia.org/wiki/Hamiltonian_path_problem) a graph *G* has a Hamiltonian path if and only if its longest path has length *n* − 1, where *n* is the number of vertices in *G*. Because the Hamiltonian path problem is NP-complete, this reduction shows that the [decision version of](https://en.wikipedia.org/wiki/Decision_problem) the longest path problem is also NP-complete. In this decision problem, the input is a graph *G* and a number *k*; the desired output is "yes" if *G* contains a path of *k* or more edges, and *no* otherwise. If the longest path problem could be solved in polynomial time, it could be used to solve this decision problem, by finding a longest path and then comparing its length to the number *k*. Therefore, the longest path problem is NP-hard. It is not NP-complete, because it is not a decision problem.

**2.4.3.8 Maximum flow algorithm: -**

Maximum flow problems involve finding a feasible flow through a single-source, single-

sink [flow network that](https://en.wikipedia.org/wiki/Flow_network) is maximum. The maximum flow problem can be seen as a special case

of more complex network flow problems, such as t[he circulation problem. T](https://en.wikipedia.org/wiki/Circulation_problem)he maximum value of an s-t flow (i.e., flow from [source s](https://en.wikipedia.org/wiki/Glossary_of_graph_theory#Direction) to [sink t)](https://en.wikipedia.org/wiki/Glossary_of_graph_theory#Direction) is equal to the minimum capacity of an [s-t cut](https://en.wikipedia.org/wiki/Cut_(graph_theory)) (i.e., cut severing s from t) in the network, as stated in the [max-flow min-cut theorem.](https://en.wikipedia.org/wiki/Max-flow_min-cut_theorem)

**Algorithm:**

Let be a network with being the source and the sink of respectively.



The capacity of an edge is a mapping, denoted by or . It represents the maximum amount of flow that can pass through an edge.



A flow is a mapping, denoted by or, subject to the following two constraints:



1., for each (capacity constraint: the flow of an edge cannot exceed its capacity)



2., for each (conservation of flows: the sum of the flows entering a node must equal the sum of the flows exiting a node, except for the source and the sink nodes)



The value of flow is defined by, where is the source of . It represents the amount of flow passing from the source to the sink.



The maximum flow problem is to maximize, that is, to route as much flow as possible from to .



**Solution:**

We can define the Residual Graph, which provides a systematic way to search for forward-

backward operations in order to find the maximum flow.

Given a flow network *G*, and a flow *f* on *G*, we define the residual graph of *G* with respect to *f* as follows.



The node set of is the same as that of *G*.



|  |  |  |  |
| --- | --- | --- | --- |
| Each edge | of | is with a capacity of | . |
| Each edge | of | is with a capacity of f(e). |  |

**2.4.3.9 Critical path analysis: -**

A project-management technique that lays out all the activities needed to complete a task, the

time it will take to complete each activity and the relationships between the activities. Also called the "critical path method", [critical path analysis c](http://www.investopedia.com/terms/c/critical-path-analysis-cpa.asp)an help predict whether a project can be completed on time and can be used to reorganize the project both before starting it, and as it progresses, to keep the project's completion [on track](http://www.investopedia.com/terms/o/ontrack.asp) and ensure that [deliverables a](http://www.investopedia.com/terms/d/deliverables.asp)re ready on time. Either manually or using computer software, the project manager first lists each activity, the order it must be completed in and how long it is expected to take, and then diagrams the process.

**2.4.3.10 Graph colouring: -**

In [graph theory, g](https://en.wikipedia.org/wiki/Graph_theory)raph colouring is a special case of [graph labelling;](https://en.wikipedia.org/wiki/Graph_labeling) it is an assignment of

labels traditionally called "colours" to elements of a [graph](https://en.wikipedia.org/wiki/Graph_(mathematics)) subject to certain constraints. In its simplest form, it is a way of colouring the vertices of a graph such that no two adjace[nt vertices](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) share the same colour; this is called a vertex colouring. Similarly, an [edge colouring](https://en.wikipedia.org/wiki/Edge_coloring) assigns a colour to each edge so that no two adjacent edges share the same colour, and a face colouring of a planar graph assigns a colour to each face or region so that no two faces that share a boundary have the same colour.

Vertex colouring is the starting point of the subject, and other colouring problems can be transformed into a vertex version. For example, an edge colouring of a graph is just a vertex colouring of it[s line graph,](https://en.wikipedia.org/wiki/Line_graph) and a face colouring of a graphics just a vertex colouring of its [dual.](https://en.wikipedia.org/wiki/Dual_graph) However, non-vertex colouring problems are often stated and studied *as is*. That is partly for perspective, and partly because some problems are best studied in non-vertex form, as for instance is edge colouring.

The convention of using colours originates from colouring the countries of a map, where each face is literally coloured. This was generalized to colouring the faces of a graph [embedded](https://en.wikipedia.org/wiki/Graph_embedding) in the plane. By planar duality it became colouring the vertices, and in this form it generalizes to all graphs. In mathematical and computer representations, it is typical to use the first few positive or nonnegative integers as the "colours". In general, one can use any finite set as the "colour set". The nature of the colouring problem depends on the number of colours but not on what they are.

**2.4.3.11 Traveling salesman: -**

The travelling salesman problem (TSP) asks the following question: Given a list of cities and

the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city? It is an [NP-hard probl](https://en.wikipedia.org/wiki/NP-hard)em in [combinatorial optimization,](https://en.wikipedia.org/wiki/Combinatorial_optimization) important in [operations research a](https://en.wikipedia.org/wiki/Operations_research)nd [theoretical computer science.](https://en.wikipedia.org/wiki/Theoretical_computer_science)

TSP is a special case of t[he travelling purchaser problem a](https://en.wikipedia.org/wiki/Traveling_purchaser_problem)[nd the Vehicle routing problem.](https://en.wikipedia.org/wiki/Vehicle_routing_problem)

In [the theory of computational complexity, the](https://en.wikipedia.org/wiki/Computational_complexity_theory) decision version of the TSP (where, given a length L, the task is to decide whether the graph has any tour shorter than L) belongs to the class [of NP-complete](https://en.wikipedia.org/wiki/NP-complete) problems. Thus, it is possible that the [worst-case](https://en.wikipedia.org/wiki/Best,_worst_and_average_case) [running time for](https://en.wikipedia.org/wiki/Running_time) any algorithm for the TSP increases [super polynomially (](https://en.wikipedia.org/wiki/Time_complexity#Superpolynomial_time)perhaps, specifically, exponentially) with the number of cities.

**2.4.3.12 Hamilton tour: -**

A Hamiltonian path (or traceable path) is [a path](https://en.wikipedia.org/wiki/Path_(graph_theory)) in an undirected or directed graph that visits eac[h vertex e](https://en.wikipedia.org/wiki/Vertex_(graph_theory))xactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian path that is a [cycle. D](https://en.wikipedia.org/wiki/Cycle_(graph_theory))etermining whether such paths and cycles exist in graphs is the [Hamiltonian path](https://en.wikipedia.org/wiki/Hamiltonian_path_problem) problem, which is [NP-complete.](https://en.wikipedia.org/wiki/NP-complete_problem)

Hamiltonian paths and cycles and cycle paths are named after [William Rowan Hamilton who](https://en.wikipedia.org/wiki/William_Rowan_Hamilton) invented [the icosian game,](https://en.wikipedia.org/wiki/Icosian_game) now also known as *Hamilton's puzzle*, which involves finding a Hamiltonian cycle in the edge graph of the [do decahedron.](https://en.wikipedia.org/wiki/Dodecahedron) Hamilton solved this problem using [the icosian calculus, a](https://en.wikipedia.org/wiki/Icosian_calculus)n [algebraic structure](https://en.wikipedia.org/wiki/Algebraic_structure) based on [roots of unity with](https://en.wikipedia.org/wiki/Root_of_unity) many similarities to [the quaternions (](https://en.wikipedia.org/wiki/Quaternion)also invented by Hamilton). This solution does not generalize to arbitrary graphs. However, despite being named after Hamilton, Hamiltonian cycles in polyhedral had also been studied a year earlier by [Thomas Kirkman.](https://en.wikipedia.org/wiki/Thomas_Kirkman)

**2.4.4 Complexity: -**

In this module user can see the best case and worst case performance of the algorithm. It includes the time complexity of all algorithm.

Algorithmic complexity is concerned about how fast or slow particular algorithm performs. We define complexity as a numerical function T(n) - time versus the input size n. We want to define time taken by an algorithm without depending on the implementation details.

But you agree that T(n) does depend on the implementation! A given algorithm will take different amounts of time on the same inputs depending on such factors as: processor speed; instruction set, disk speed, brand of compiler and etc.

**2.4.4.1 Time Complexity: -** The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the string representing the input. The time complexity of an algorithm is commonly expressed using big O notation, which excludes coefficients and lower order terms.

When expressed this way, the time complexity is said to be described asymptotically, i.e., as the input size goes to infinity. For example, if the time required by an algorithm on all inputs of size n is at most (5n) ^3 + 3n for any n (bigger than some n0), the asymptotic time complexity is O(n)^3.

Time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, where an elementary operation takes a fixed amount of time to perform. Thus the amount of time taken and the number of elementary operations performed by the algorithm differ by at most a constant factor.

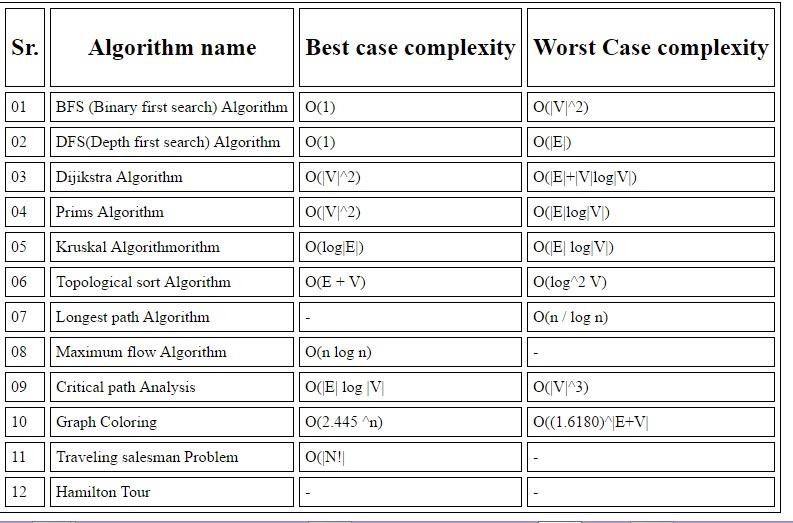
Since an algorithm's performance time may vary with different inputs of the same size, one commonly uses the worst-case time complexity of an algorithm, denoted as T(n), which is defined as the maximum amount of time taken on any input of size n. Less common, and usually specified explicitly, is the measure of average-case complexity.

**2.4.4.2 Space Complexity: -** Space Complexity of an algorithm is total space taken by the algorithm with respect to the input size. Space complexity includes both Auxiliary space and space used by input.

For example, if we want to compare standard sorting algorithms on the basis of space, then

Auxiliary Space would be better criteria than Space Complexity.

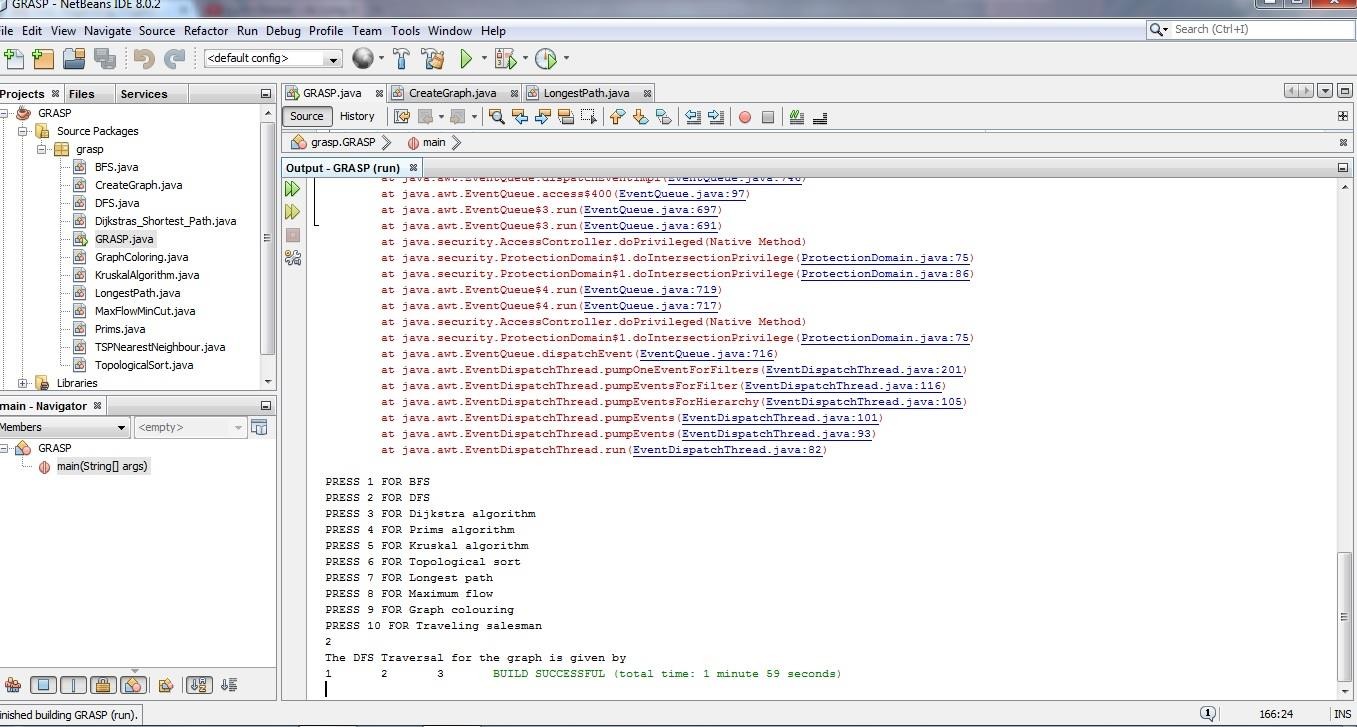
Merge Sort uses O(n) auxiliary space, Insertion sort and Heap Sort Use O (1) auxiliary space. Space complexity of all these sorting algorithms is O(n) though.



**2.7 Description module: -** In this module user get the final graphical output. The given output

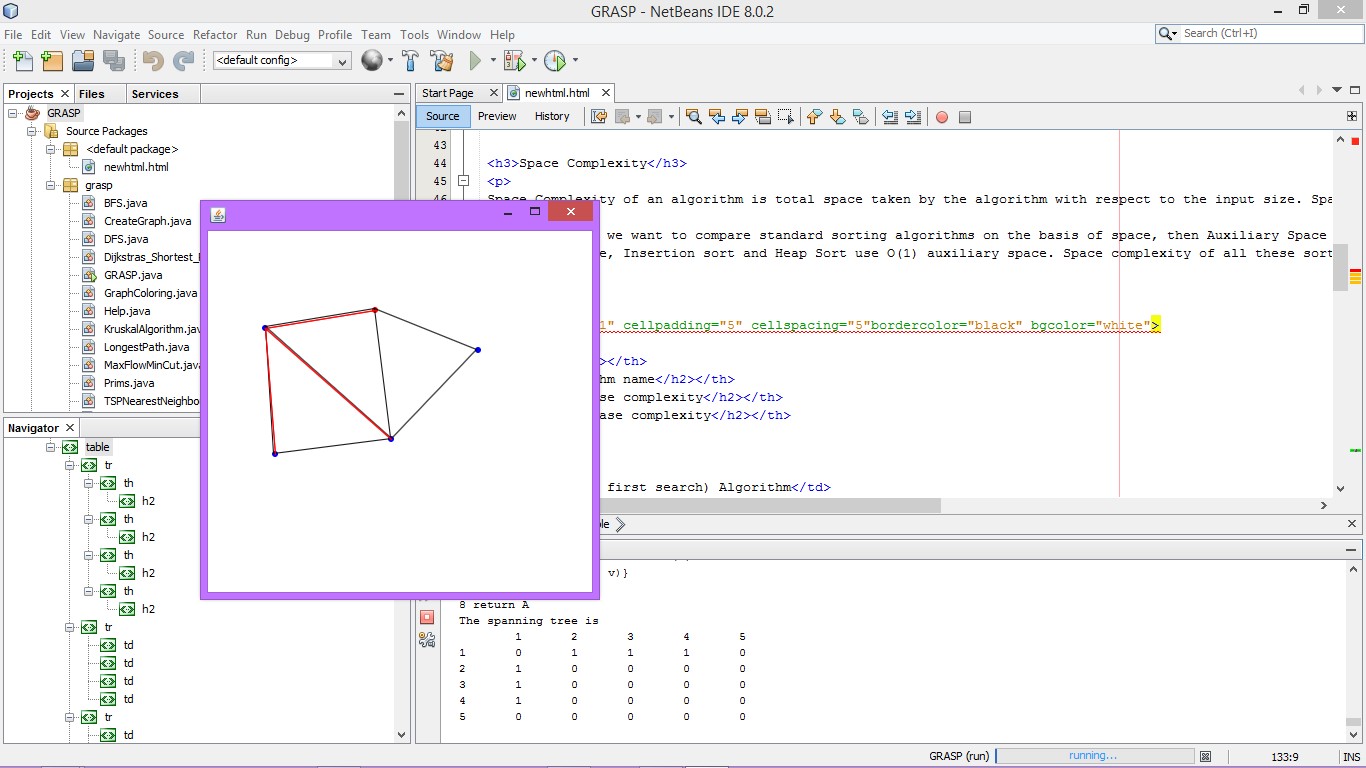
Is shown by highlighting the shortest path in the graph. This module includes two sub module

**2.4.5 Summarisation:** In this sub module the whole process shows up how algorithm executed step by step.



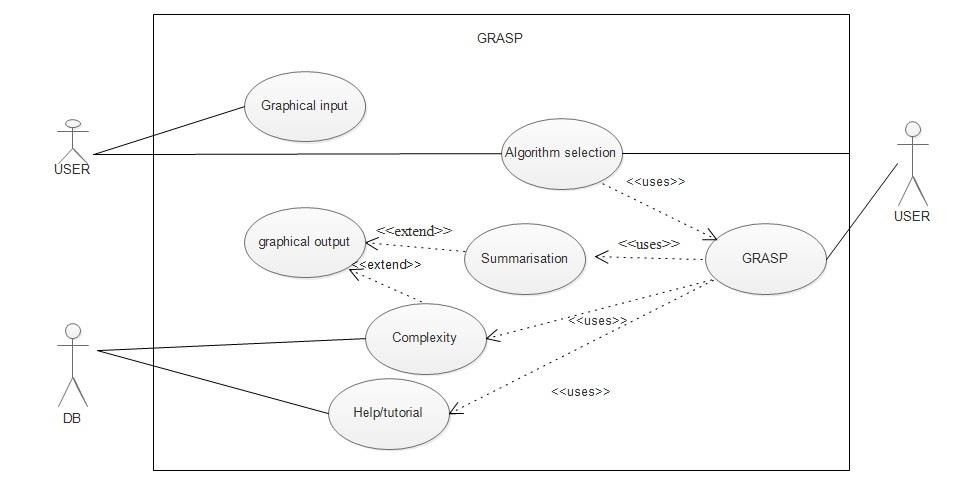
**2.4.6 Graphical output:** This sub module shows the graphical output of the given graphical input by highlighting the shortest path between the nodes. The graphical output is shown as below of the particular input:

**2.5 Constraints: -** The GRASP mainly depends on the constraints that while giving the input you must put node where you want and draw line by clicking on the selected node in the middle

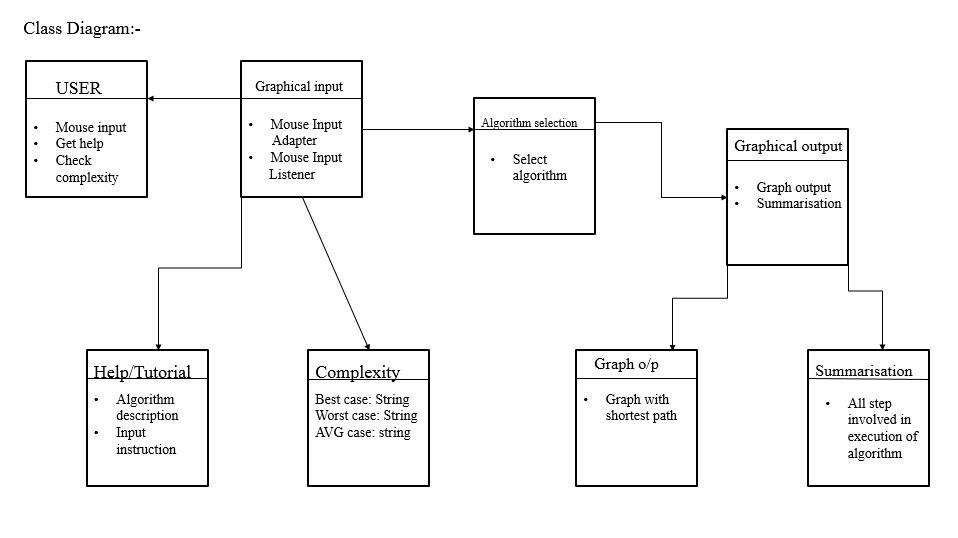


till the next node, by clicking on any node its colour changes to red than you can draw a line to next node.

**2.6 Use case Model description: -**The use case diagram is shown below



**Data objects: -** The class diagram, sequence diagram is shown below showing the process of GRASP.



Data model diagram: In [software engineering, a](https://en.wikipedia.org/wiki/Software_engineering) class diagram in the [Unified Modelling Language (](https://en.wikipedia.org/wiki/Unified_Modeling_Language)UML) is a type of static structure diagram that describes the structure of a system by showing the system's [classes,](https://en.wikipedia.org/wiki/Class_(computer_science)) their attributes, operations (or methods), and the relationships among objects.

The class diagram is the main building block of [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) modelling. It is used both for general [conceptual modelling](https://en.wikipedia.org/wiki/Conceptual_model) of the systematics of the application, and for detailed modelling translating the models into [programming code. Class](https://en.wikipedia.org/wiki/Programming_code) diagrams can also be used for [data modelling.](https://en.wikipedia.org/wiki/Data_modeling) The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

In the diagram, classes are represented with boxes that contain three compartments:

The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.

The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.

The bottom compartment contains the operations the class can execute. They are also left- aligned and the first letter is lowercase.

In the design of a system, a number of classes are identified and grouped together in a class diagram that helps to determine the static relations between them. With detailed modelling, the classes of the conceptual design are often split into a number of subclasses.

In order to further describe the behaviour of systems, these class diagrams can be complemented by a [state diagram or](https://en.wikipedia.org/wiki/State_diagram) [UML state](https://en.wikipedia.org/wiki/UML_state_machine)

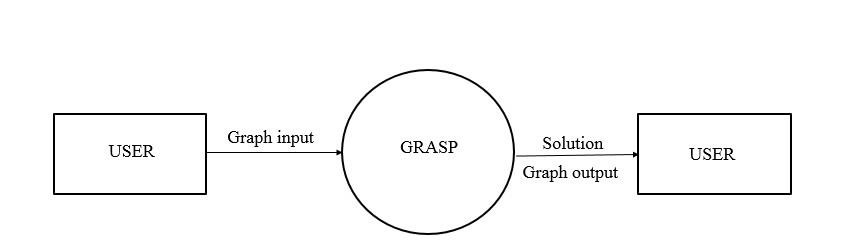
**2.7 Sequence Diagram: -**



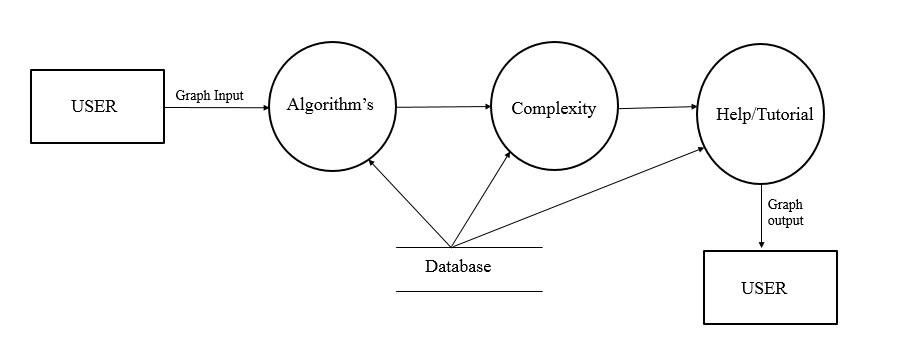
**2.8 Data flow diagrams: -** A data flow diagram (DFD) is a graphical representation of the "flow" of data through an [information system, m](https://en.wikipedia.org/wiki/Information_system)odelling its process aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated. DFDs can also be used for [the visualization of](https://en.wikipedia.org/wiki/Data_visualization) [data processing](https://en.wikipedia.org/wiki/Data_processing) (structured design).

A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of process or information about whether processes will operate in sequence or in parallel (which is s[hown on a flowchart).](https://en.wikipedia.org/wiki/Flowchart)

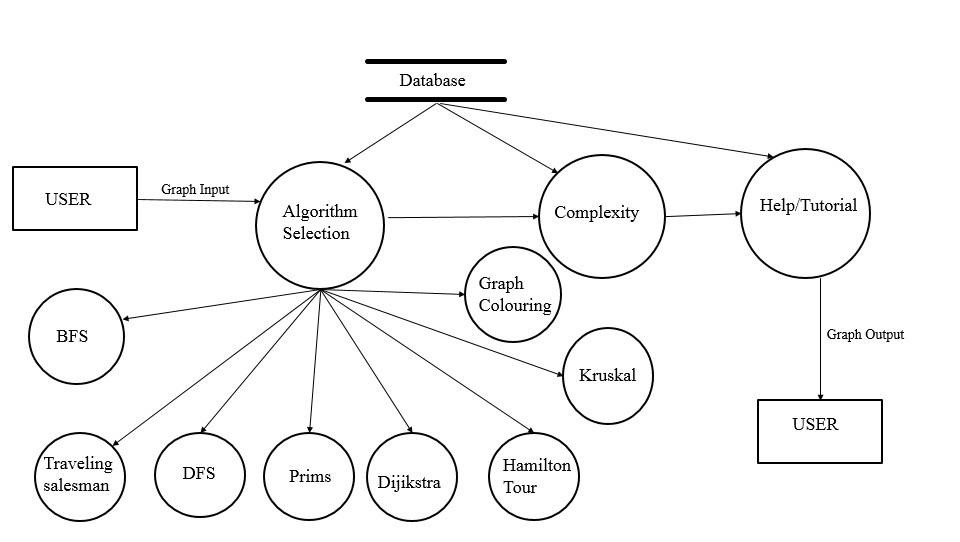
**2.8.1 Level-0 DFD: -** Level-0 DFD shows the very basic structure of the problem domain. As in the below figure level-0 DFD is shown.



**2.8.2 Level-1 DFD: -**



**2.8.3 Level-2 DFD: -**



**2.9 Software Interface Description: -**

**2.9.1 Proposed system:** The proposed system **“GRASP”** includes-

o Graphical representation

o Algorithm selection

o Summarisation

o Graphical output

**2.9.2 Advantages of proposed system: -** The proposed system has following advantages-

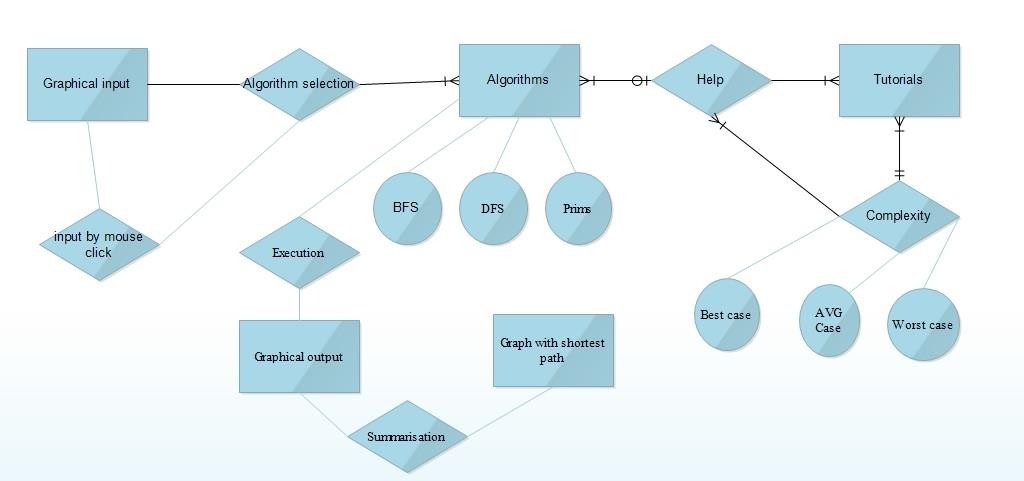
o It shows the graphical representation of the algorithms.

o It also shows the process involved in graphical representation of algorithm.

**2.10 Assumption and Dependencies: -** In the project of GRASP we did not consider any type of assumption. As some of the problem occurs in our project so we have to assume that which node is entering first kind of remember node which is created first.

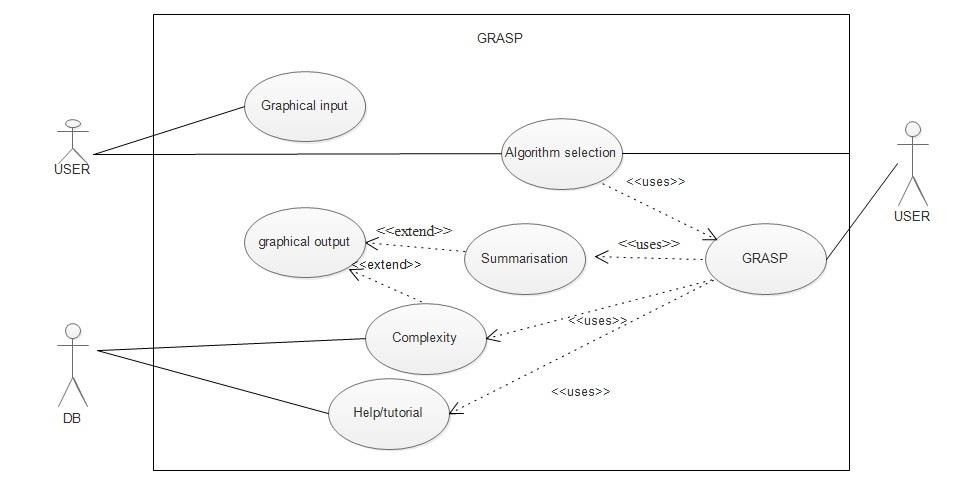
**2.11 Database Design: -**

**2.11.1 ER-diagram:**



**3.0 Specific Requirements: -**

**3.1 Use case reports: -** Use case report shows us that which actor is involved in which process. As in this proposed model there are two actor user and DB. Which are involved in various action like Graphical input, Algorithm selection, Help/tutorials, Complexity and graphical output. The use case diagram is given below.



**3.2 FEASIBILITY STUDY**

Feasibility study is an important phase in the software development process. It enables the developer to have an assessment of the product being developed. It refers to the feasibility study of the product in terms of outcomes of the product, operational use and technical support required for implementing it. Feasibility study should be performed on the basis of various criteria and parameters. The various feasibility studies are:

 Economic Feasibility

 Operational Feasibility

 Technical Feasibility

**3.2.1 ECONOMIC FEASIBILITY**

It refers to the benefits or outcomes we are deriving from the product as compared to the total cost we are spending for developing the product. If the benefits are more or less the same as the older system, then it is not feasible to develop the product. This project is very less costly because all the required software is easily available.

**3.2.2 OPERATIONAL FEASIBILITY**

It refers to the feasibility of the product to be operational. Some products may work very well design and implementation but may fail in the real environment. It includes the study of additional human resource required and their technical expertise. In this project it will operate in the college environment where the users can operate easily.

**3.2.3 TECHNICAL FEASIBILITY**

It refers to whether the software that is available in the market fully supports the present application. It studies the pros and cons of using particular software for the development and its feasibility. It also studies the additional training needed to be given to the people to make the application work.

This software will run well on those devices which supports java and browsers like chrome and Mozilla Firefox and also supports others browsers. These browsers are easily available in the internet. The main feature of this project is that user can learn about algorithms easily in the graphical way.

**4.0 TESTING AND MAINTINANCE: -**

**4.1 Introduction:** The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies and/or a finished product. It is the process of exercising with the intent of ensuring that the

Software system meets its requirements and user expectation and does not fail in an unacceptable manner. There are various types of test. Each test type address a specific testing requirement.

**4.2 TYPES OF TESTING**

**4.2.1 UNIT TESTING: -** Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that the programs inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software unit of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit test performs basic test at component level and test a specific business process, application, and/ or system configuration. Unit tests ensures that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**4.2.2 INTEGRATION TESTING: -** Integration test is designed to test integrated software components to determine if they actually runs as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration test demonstrate that although the components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

**4.2.3 FUNCTIONAL TESTING: -** Functional tests provide systematic demonstrations that function tested are available as specified by business and technical requirements, system documentation, and user manuals.

Functional testing is centred on the following items:

Valid Input : Identifies classes of valid input must be accepted. Invalid input : Identified function must be rejected.

Functions : Identified functions must be exercised.

Output : Identified classes of application outputs must be exercised. System/procedures : Interfacing system or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or Special test cases. In addition, systematic coverage pertaining to identify Business process flows; Data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**4.2.4 SYSTEM TESTING-** System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

**4.2.5 WHITE BOX TESTING-** White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least it’s Purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**4.2.6 BLACK BOX TESTING-** Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box. you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

**4.3 Unit testing**

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach Field testing will be performed manually and functional tests will be written in detail.

Test objectives

 All nodes entries must work properly.

 Pages must be activated from the identified link.

 The input screen, messages and responses must not be delayed. Features to be tested

 Verify that the entries are of the correct format.

 No duplicate entries should be allowed.

 All nodes should take the user to the next node.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.**  **no.** | **Input** | **Expected output** | **Actual Output** | **Status** |
| 01 | Input page  screen | A GUI screen pop ups  for taking input | A GUI screen pop ups  for taking input | Pass |
| 02 | Click to create  node | Node created | Node created | pass |
| 03 | Click on centre  of node to draw edges | Node colour must  changed | Colour changed to red | pass |
| 04 | Draw an edge  from clicked node to another | Edge created | Edge created | Pass |
| 05 | Algorithm  selection | After input it asked for  choice of your algorithm | Asked for enter choice  of algorithm | Pass |
| 06 | Graphical output | After selection of  algorithm it shows graphical output | Graphical output by  highlighting shortest path | pass |
| 07 | Output Summary | Shows the summary and  time taken | Shows the summary  steps and time taken | pass |

**4.4 Integration testing**

Software integration testing is the incremental integration testing of two or more integrated Software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications, e.g. components in a software system or – one steps up – software applications at the company level – interact without error.

**Test Results -** All the test cases mentioned above passed successfully. No defects encountered.

**4.5 Acceptance testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**5.0 Sample code: -** the code for the graphical input and output is given below.

**5.1 Graphical Input: -**

package grasp;

import java.awt.\*; import java.awt.event.\*; import java.awt.geom.\*; import java.util.\*;

import java.util.List;

import javax.swing.\*;

public class CreateGraph {

static int adj[][] = new int[10][10];

public static void create() { JFrame f = new JFrame();

//f.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); f.getContentPane().add(new PointPanel()); f.setSize(400,400);

f.setLocation(200,200);

f.setVisible(true);

// return adj;

}

}

class PointPanel extends JPanel

{

List pointList; List lineList;

Color selectedColor; Ellipse2D selectedPoint;

public PointPanel()

{

pointList = new ArrayList(); lineList = new ArrayList(); selectedColor = Color.red;

addMouseListener(new PointLocater(this));

setBackground(Color.white);

}

protected void paintComponent(Graphics g)

{ super.paintComponent(g); Graphics2D g2 = (Graphics2D)g;

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

Ellipse2D e; Color color;

for(int j = 0; j < pointList.size(); j++)

{

e = (Ellipse2D)pointList.get(j);

if(e == selectedPoint) color = selectedColor; else

color = Color.blue; g2.setPaint(color); g2.fill(e);

}

Line2D l;

color = Color.BLACK;

for(int j = 0; j < lineList.size(); j++)

{

l = (Line2D)lineList.get(j); g2.setPaint(color); g2.draw(l);

}

}

public List getPointList()

{

return pointList;

}

public List getLineList()

{

return lineList;

}

public void setSelectedPoint(Ellipse2D e)

{

selectedPoint = e;

repaint();

}

public void addPoint(Point p)

{

Ellipse2D e = new Ellipse2D.Double(p.x - 3, p.y - 3, 6, 6);

pointList.add(e); selectedPoint = null; repaint();

}

public void addLine(Point p1,Point p2)

{

Line2D e = new Line2D.Double(p1.x, p1.y, p2.x, p2.y);

lineList.add(e);

repaint();

}

}

class PointLocater extends MouseAdapter

{

PointPanel pointPanel; boolean dragging = false; Point p1,p2;

int x,y;

public PointLocater(PointPanel pp)

{

pointPanel = pp;

}

public void mousePressed(MouseEvent e)

{

Point p = e.getPoint();

dragging=true;

boolean haveSelection = false;

List list = pointPanel.getPointList(); Ellipse2D ellipse;

for(int j = 0; j < list.size(); j++)

{

ellipse = (Ellipse2D)list.get(j);

if(ellipse.contains(p))

{ pointPanel.setSelectedPoint(ellipse); p1 = p;

x = j;

haveSelection = true;

break;

}

} if(!haveSelection) pointPanel.addPoint(p);

}

public void mouseReleased(MouseEvent e)

{

Point p = e.getPoint();

List list = pointPanel.getPointList(); Ellipse2D ellipse;

for(int j = 0; j < list.size(); j++)

{

ellipse = (Ellipse2D)list.get(j);

if(ellipse.contains(p))

{

p2 = p; pointPanel.addLine(p1, p2); y = j;

CreateGraph.adj[x][y] = 1; CreateGraph.adj[y][x] = 1; break;

}

}

}

}

**5.2 Graphical Output: -**

package grasp;

import java.io.\*;

public class GRASP{

public static void main(String args[])throws IOException

{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in)); System.out.println("input the graph");

CreateGraph.create();

System.out.println("PRESS ENTER WHEN INPUT DONE"); String str=br.readLine();

int adj[][] = CreateGraph.adj;

int flag,n=0;

for(int i=0;i<10;i++)

{

flag=0;

for(int j=0;j<10;j++)

{

if(adj[i][j]==1)

{

flag = 1;

break;

}

}

if(flag==0)

{

n = i;

break;

}

}

int a[][]=new int[n][n];

int b[][]=new int[n+1][n+1]; int c[][]=new int[n+1][n+1]; int d[][]=new int[n+1][n+1]; int e[][]=new int[n+1][n+1];

boolean f[][]=new boolean[n][n];

for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++)

{

a[i][j] = adj[i][j];

b[i+1][j+1] = adj[i][j]; c[i+1][j+1] = adj[i][j]; d[i+1][j+1] = adj[i][j]; e[i+1][j+1] = adj[i][j]+1; if (i == j)

{

c[i+1][j+1] = 0; d[i+1][j+1] = 0; continue;

}

if (c[i+1][j+1] == 0)

{

c[i+1][j+1] = 999;

d[i+1][j+1] = Integer.MAX\_VALUE;;

}

if(adj[i][j]==1)

f[i][j]=true;

else f[i][j]=false;

//System.out.print(a[i][j]+" ");

}

//System.out.println();

}

System.out.println("PRESS 1 FOR BFS"); System.out.println("PRESS 2 FOR DFS"); System.out.println("PRESS 3 FOR Dijkstra algorithm"); System.out.println("PRESS 4 FOR Prims algorithm "); System.out.println("PRESS 5 FOR Kruskal algorithm"); System.out.println("PRESS 6 FOR Topological sort"); System.out.println("PRESS 7 FOR Longest path"); System.out.println("PRESS 8 FOR Maximum flow"); System.out.println("PRESS 9 FOR Graph colouring"); System.out.println("PRESS 10 FOR Traveling salesman");

int ch=Integer.parseInt(br.readLine());

switch(ch)

{

case 1:System.out.println("The BFS traversal of the graph is "); BFS bfs = new BFS();

bfs.bfs(b, 1);

break;

case 2:System.out.println("The DFS Traversal for the graph is given by "); DFS dfs = new DFS();

dfs.dfs(b, 1);

break;

case 3:System.out.println("Enter the source ");

int source = Integer.parseInt(br.readLine());

System.out.println("Enter the destination ");

int destination = Integer.parseInt(br.readLine());

Dijkstras\_Shortest\_Path dijkstrasAlgorithm = new Dijkstras\_Shortest\_Path(n);

dijkstrasAlgorithm.dijkstra\_algorithm(d, source);

is: ");

System.out.println("The Shorted Path from " + source + " to " + destination + "

for (int i = 1; i <= dijkstrasAlgorithm.distances.length - 1; i++)

{

if (i == destination)

System.out.println(source + " to " + i + " is "+ dijkstrasAlgorithm.distances[i]);

}

break;

case 4:Prims prims = new Prims(n); prims.primsAlgorithm(c); prims.printMST();

break;

case 5:KruskalAlgorithm kruskalAlgorithm = new KruskalAlgorithm(n);

kruskalAlgorithm.kruskalAlgorithm(c);

break;

case 6: System.out.println("The Topological sort for the graph is given by "); TopologicalSort toposort = new TopologicalSort();

int topological\_sort[] = toposort.topological(b, 1); System.out.println();

for (int i = topological\_sort.length - 1; i > 0; i-- )

{

if (topological\_sort[i] != 0) System.out.print(topological\_sort[i]+"\t");

}

break;

case 7:System.out.println("Enter the source ");

int from = Integer.parseInt(br.readLine());

System.out.println("Enter the destination ");

int to = Integer.parseInt(br.readLine());

LongestPath lp = new LongestPath();

int path = lp.getLongestPath(from, to, f);

break;

case 8:System.out.println("Enter the source of the graph");

source= Integer.parseInt(br.readLine()); System.out.println("Enter the sink of the graph");

int sink = Integer.parseInt(br.readLine());

MaxFlowMinCut maxFlowMinCut = new MaxFlowMinCut(n);

int maxFlow = maxFlowMinCut.maxFlowMinCut(b, source, sink);

System.out.println("The Max Flow is " + maxFlow); System.out.println("The Cut Set is "); maxFlowMinCut.printCutSet();

break;

case 9:System.out.println("\nEnter number of colors"); int col = Integer.parseInt(br.readLine()); GraphColoring gc = new GraphColoring(); gc.graphColor(a, col);

break;

case 10:System.out.println("the citys are visited as follows"); TSPNearestNeighbour tspNearestNeighbour = new TSPNearestNeighbour(); tspNearestNeighbour.tsp(e);

break;

default: System.out.println("wrong choice");

}

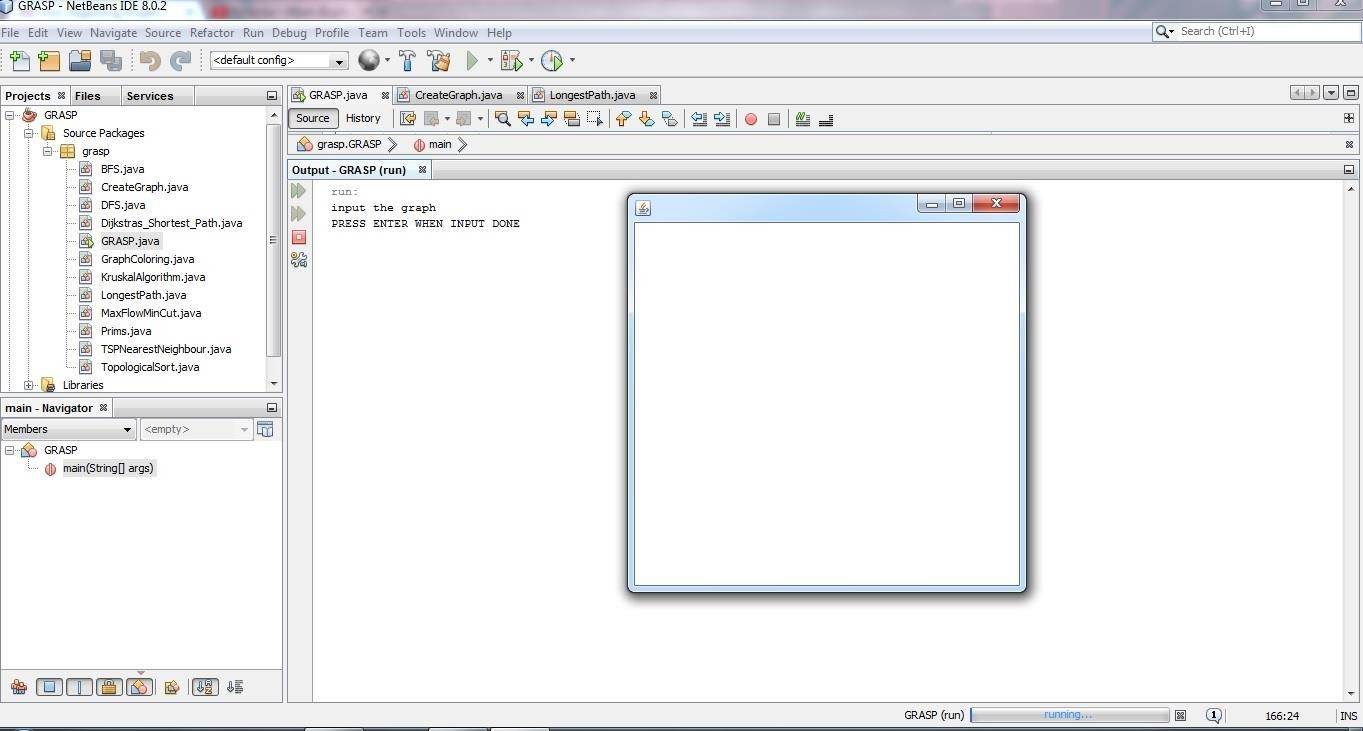
System.exit(0);

}

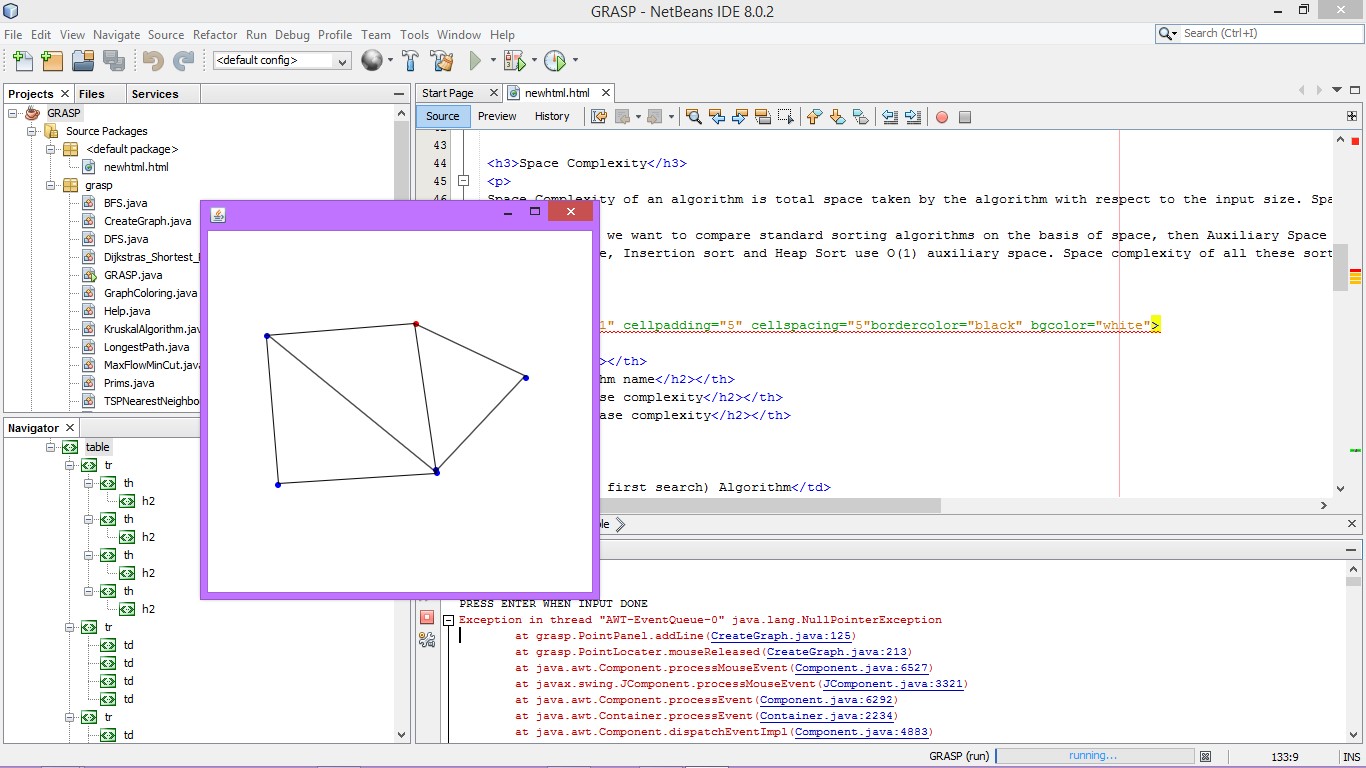
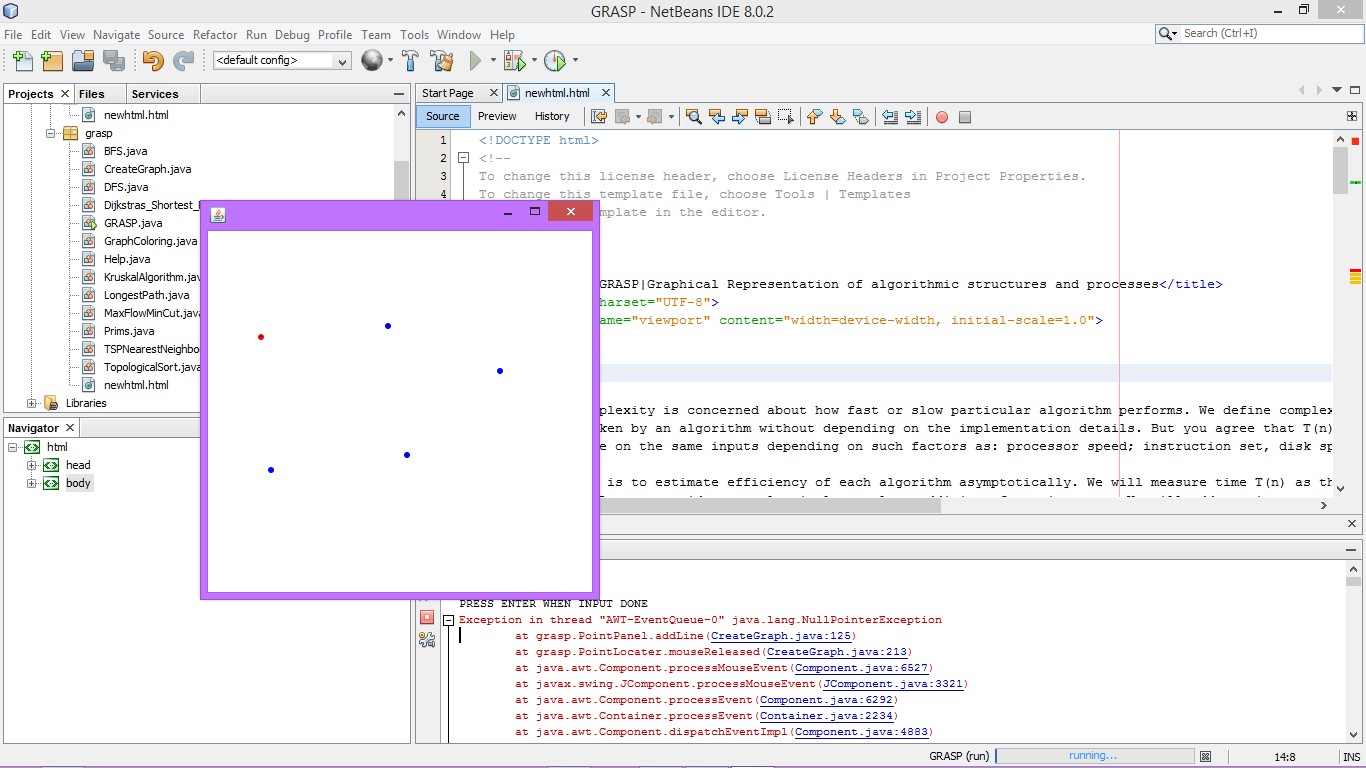
}

**6.0 Snapshots: -**

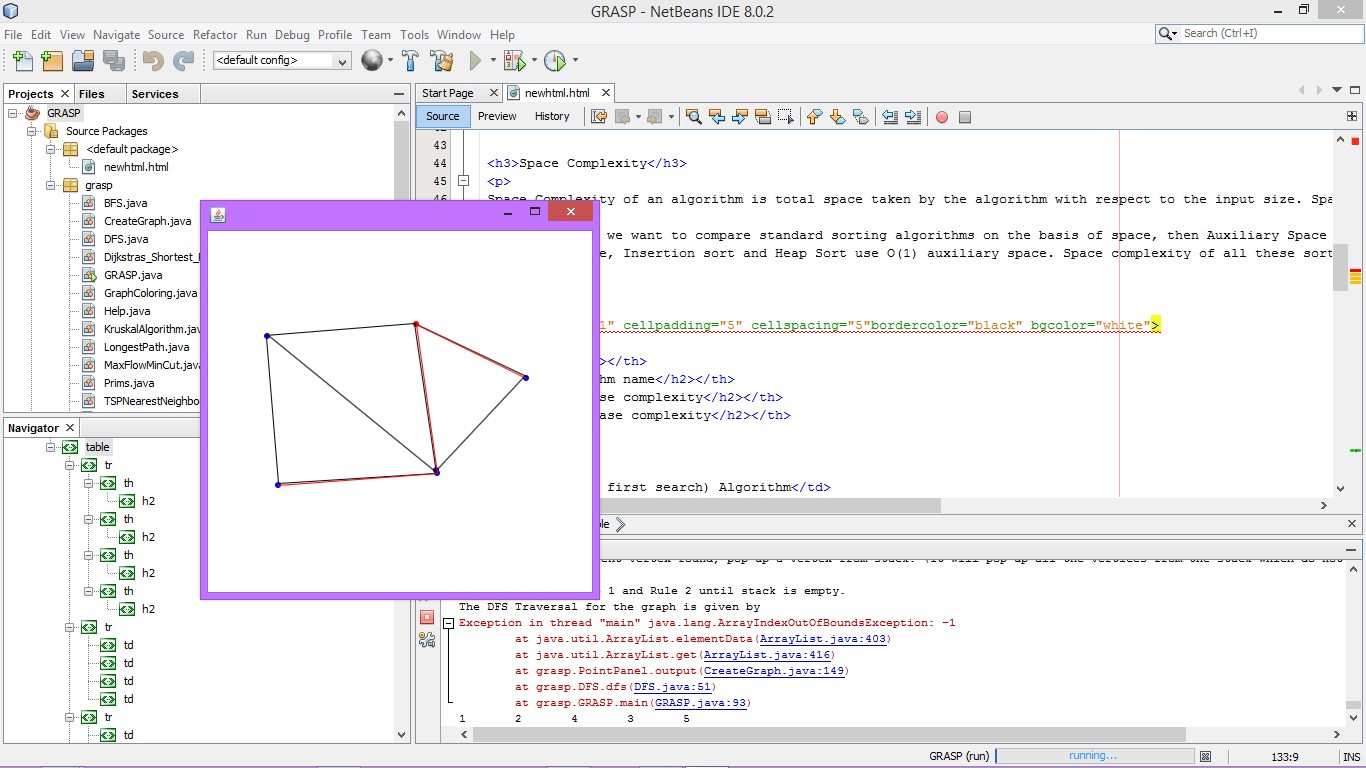
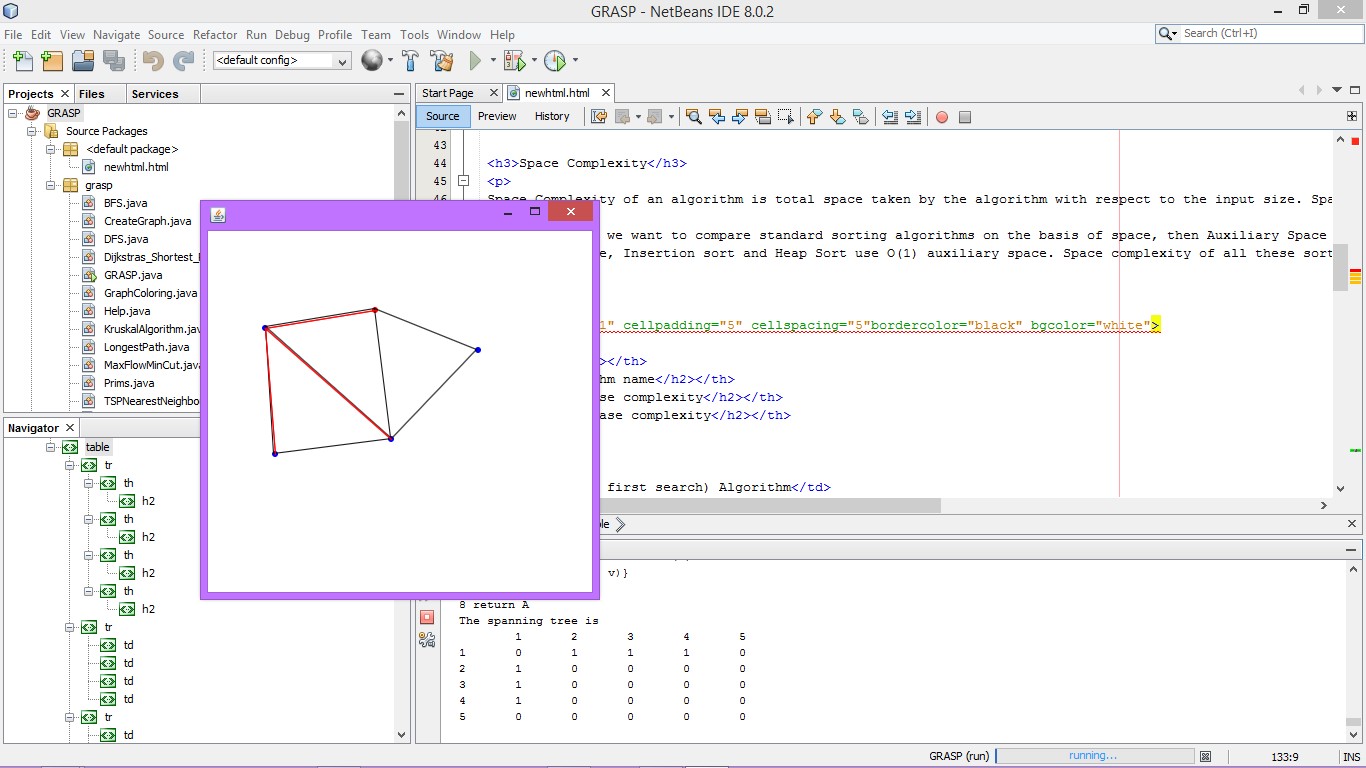
Input screen after running the projects:



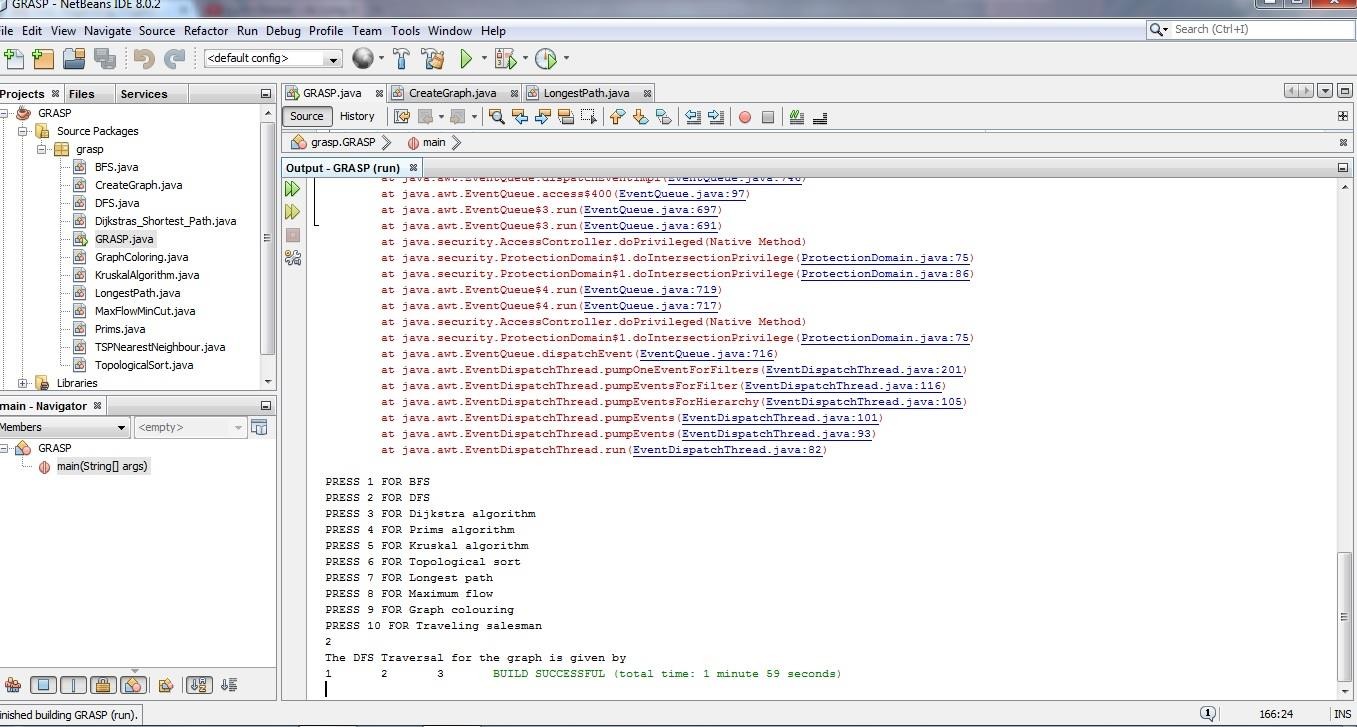
Input screen after creating node by mouse click:



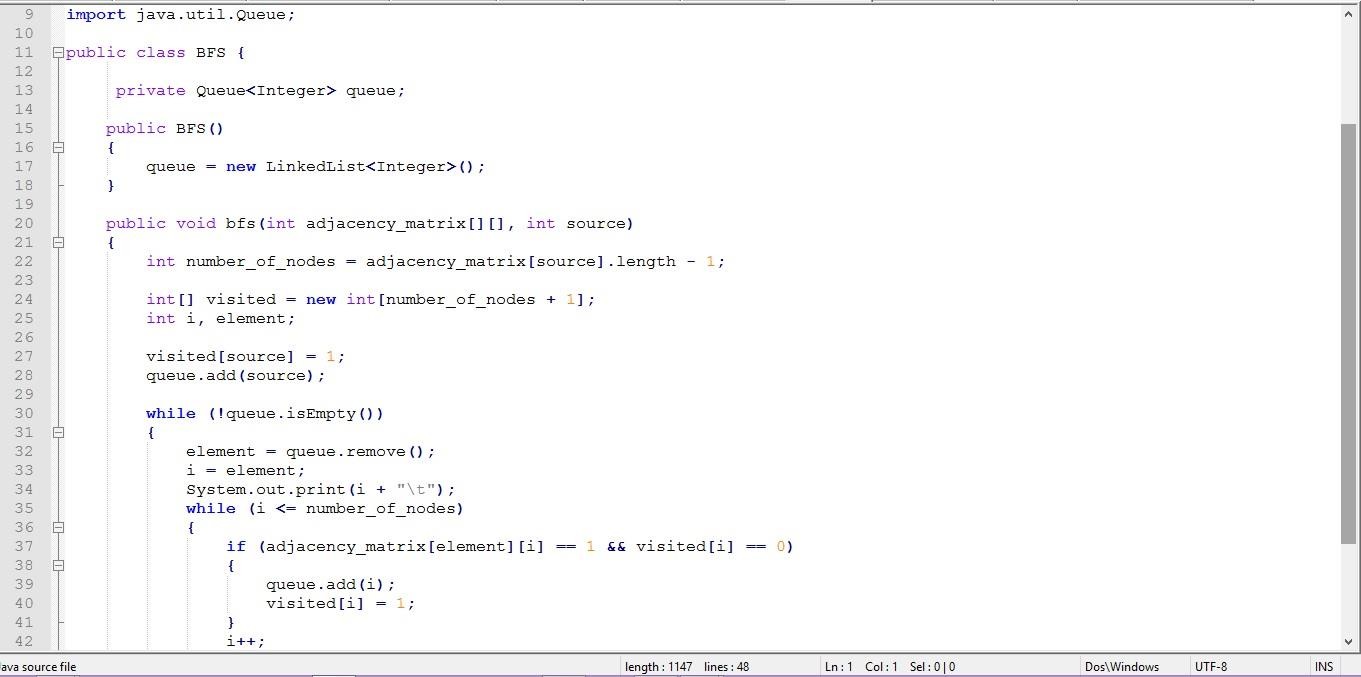
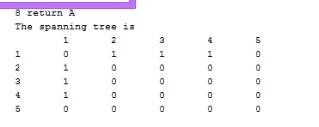
**Graphical output after running through Dijikstra algorithm:**



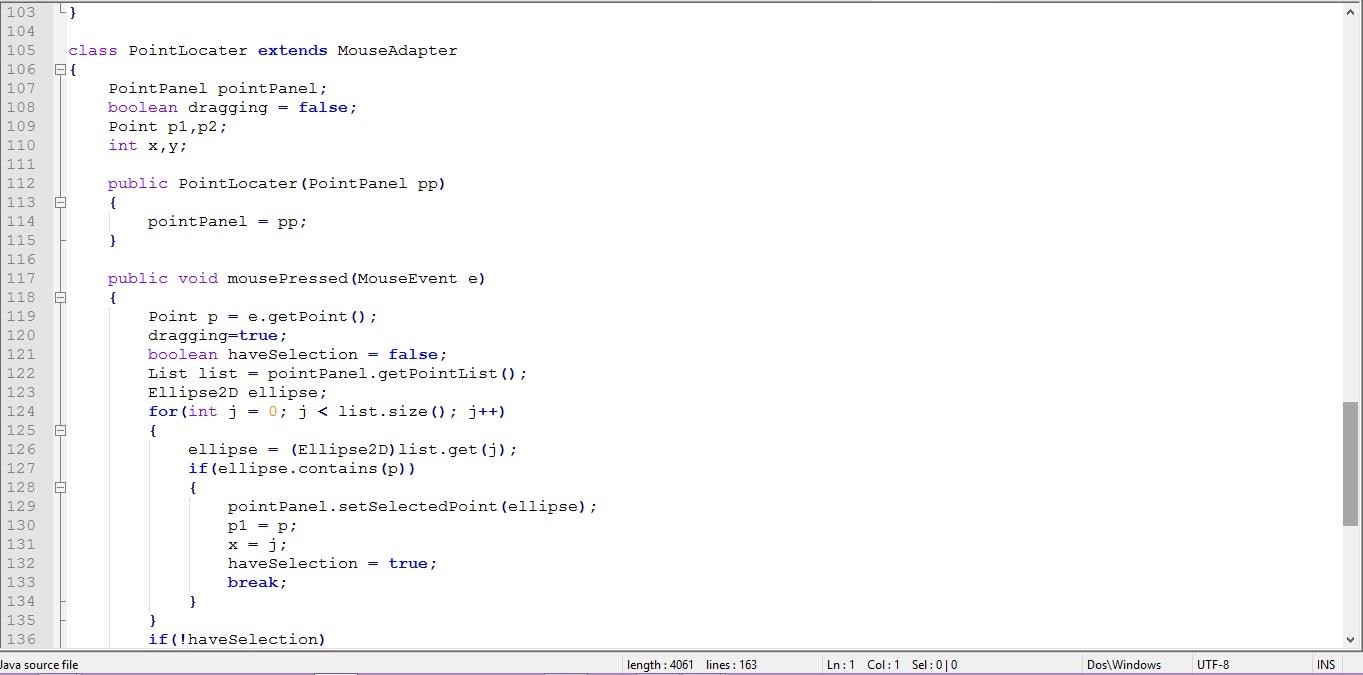
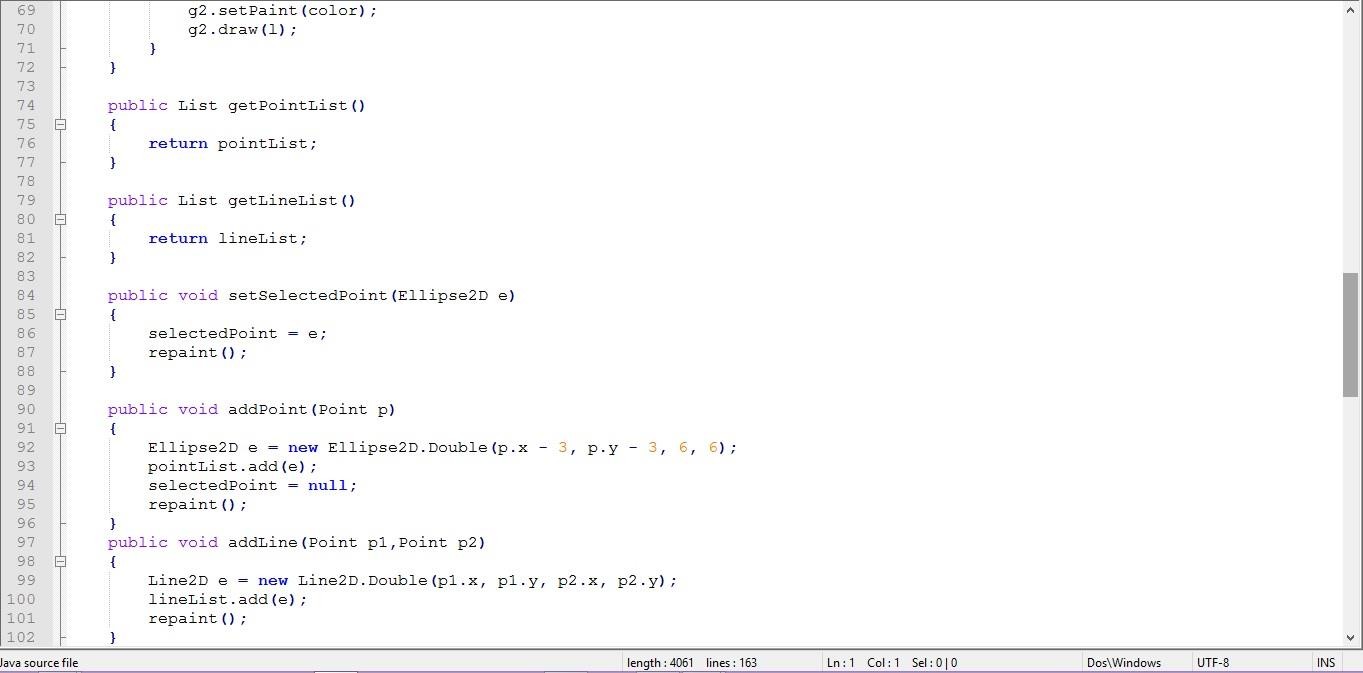
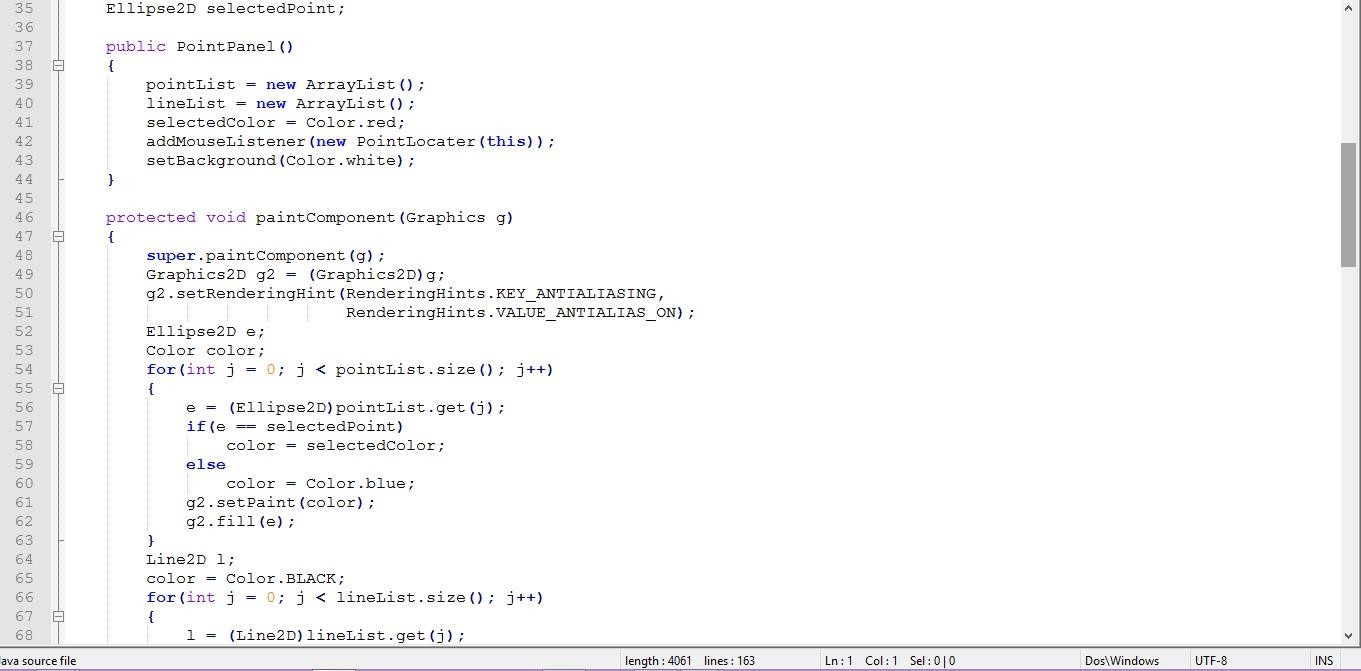
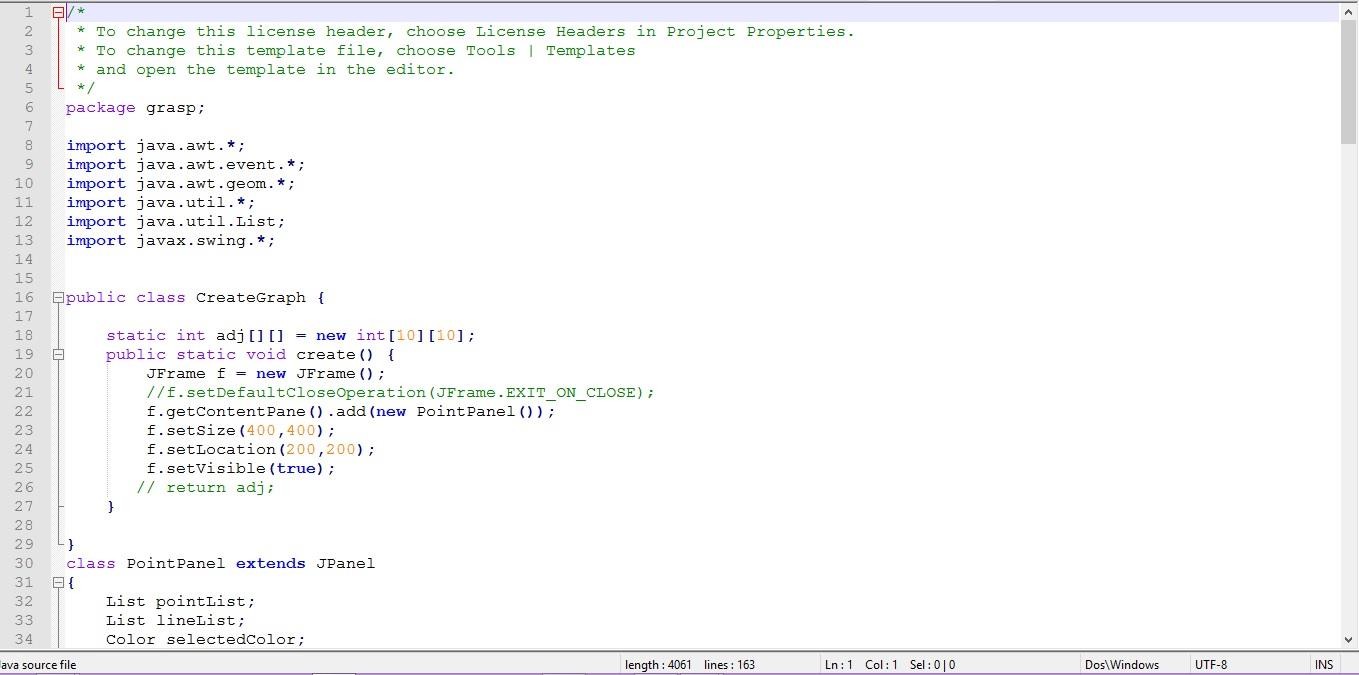
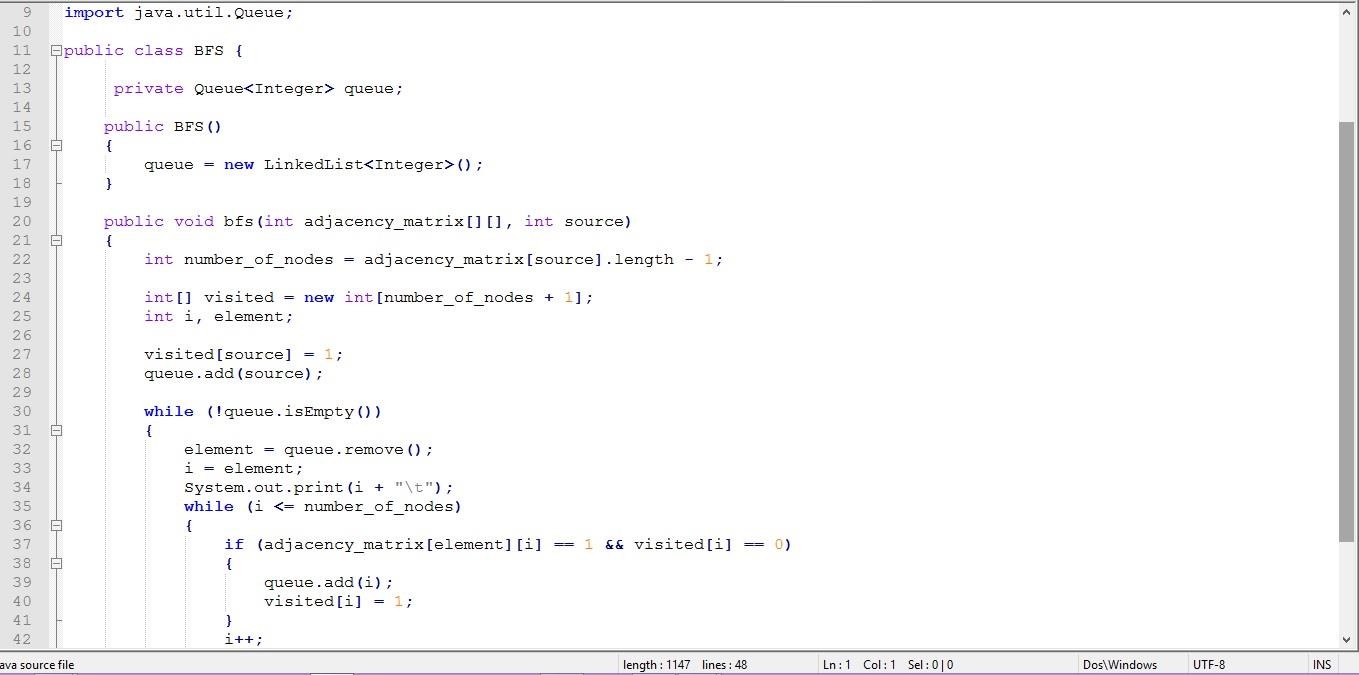
**Choice for algorithm selection:**



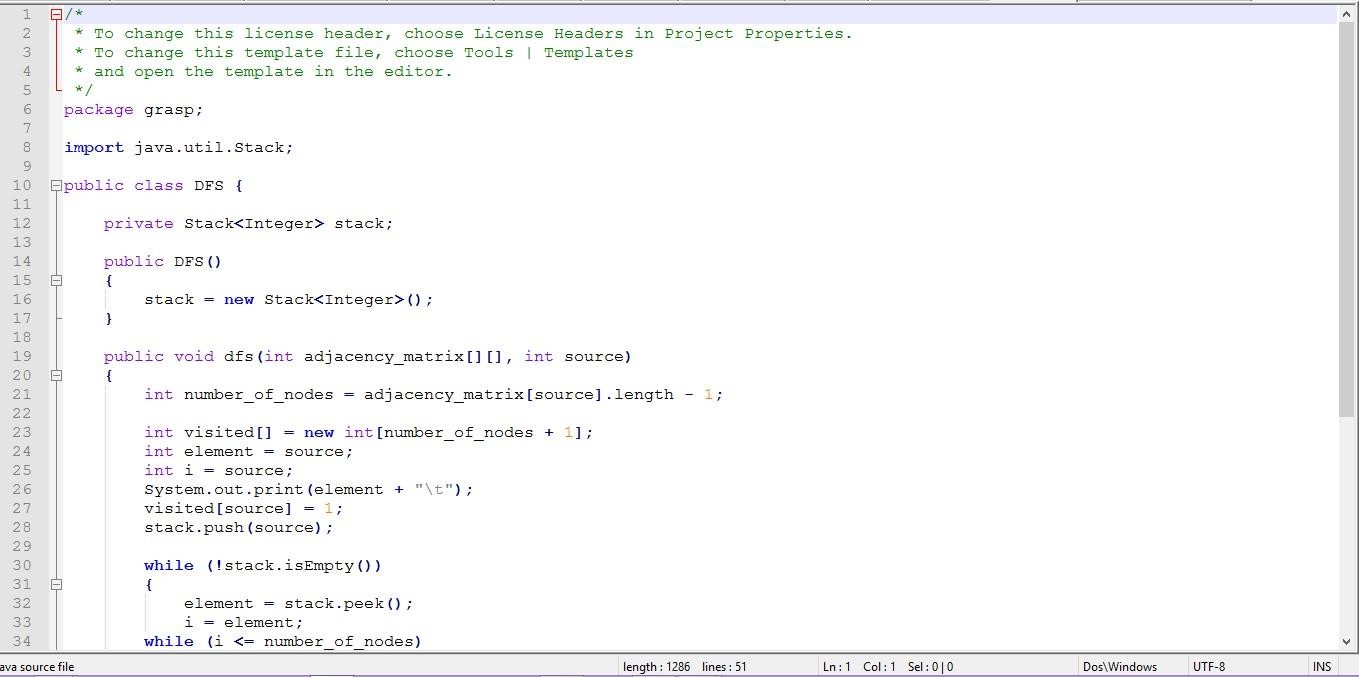
**Output in the form of summary:**



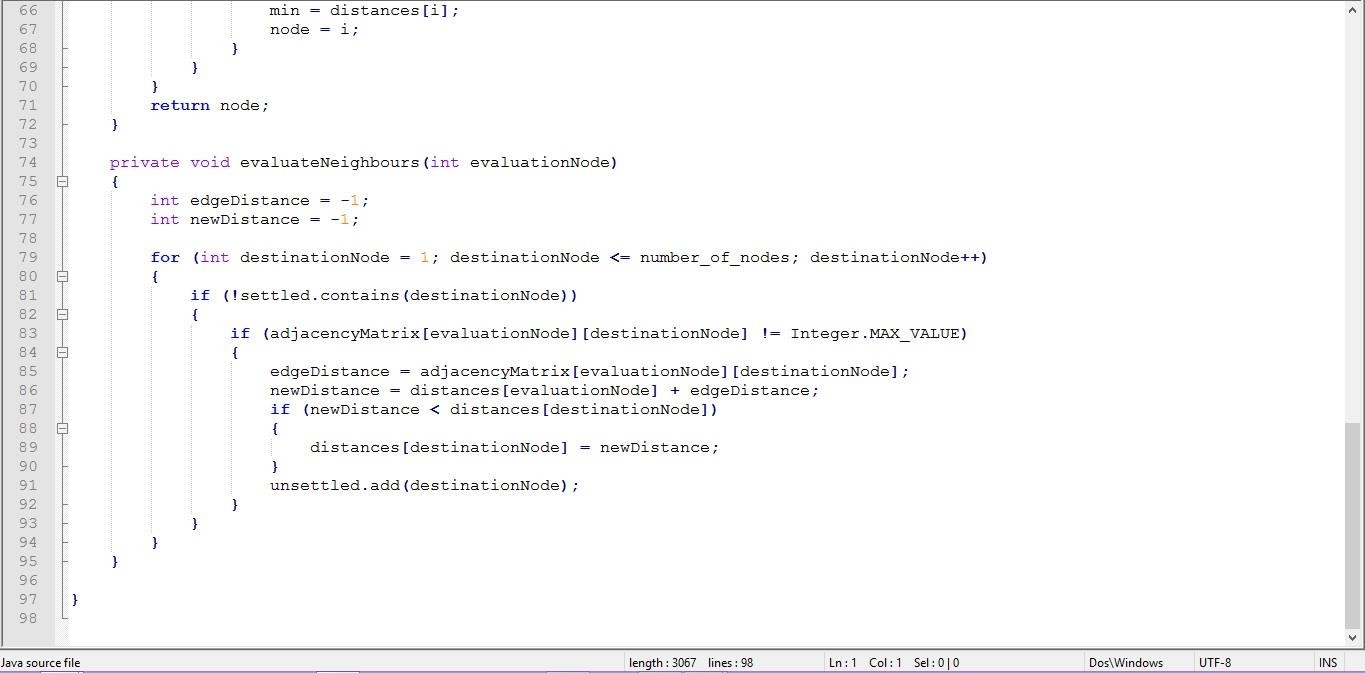
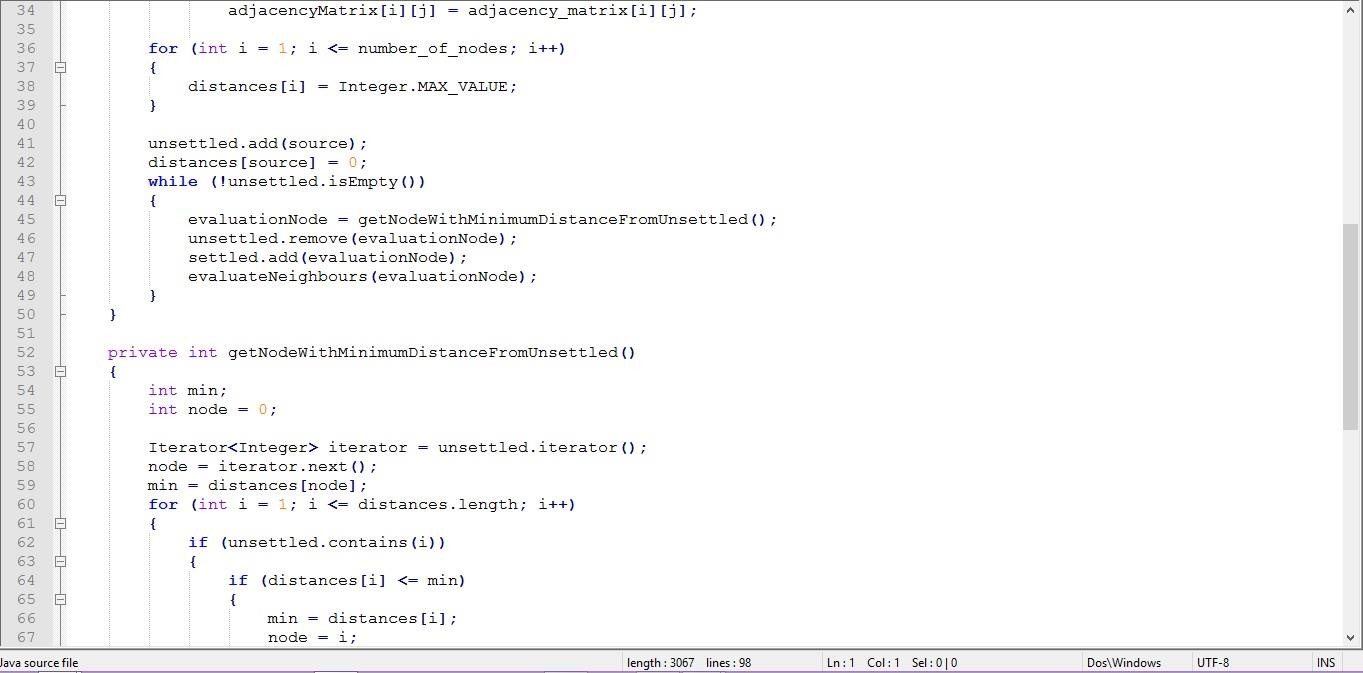
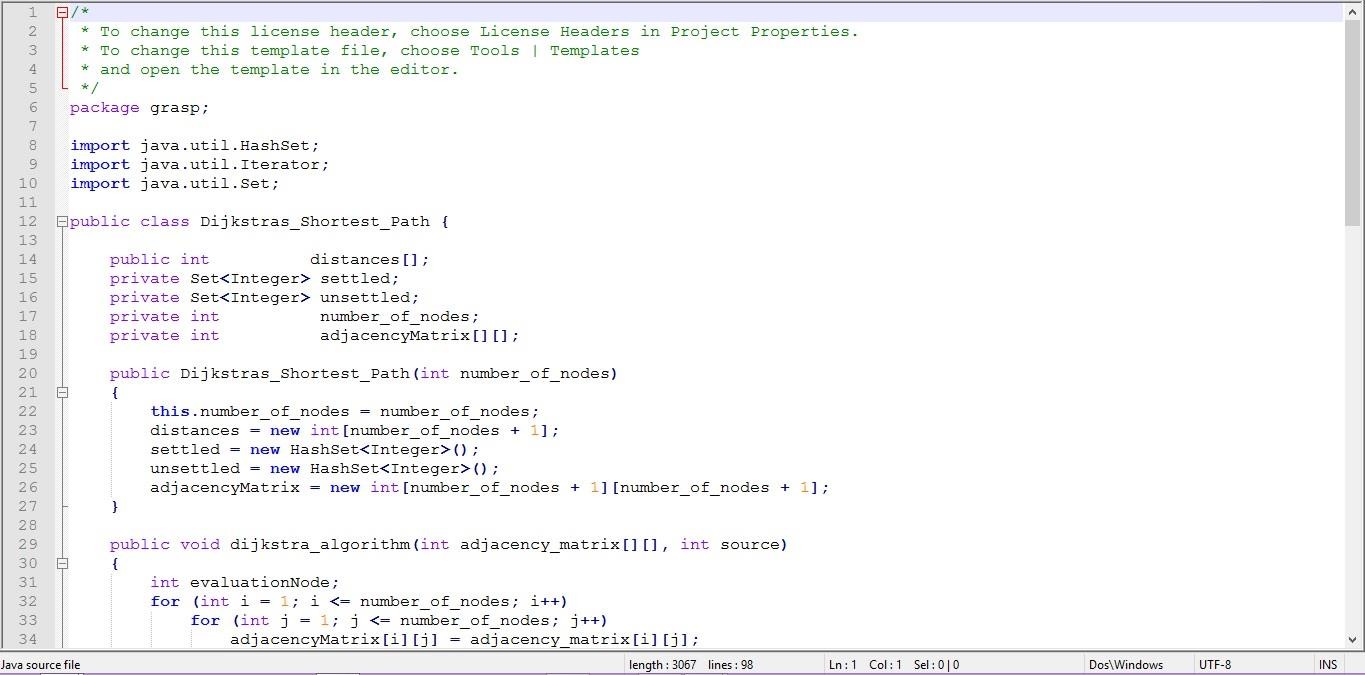
**Code for creating a graph:-**



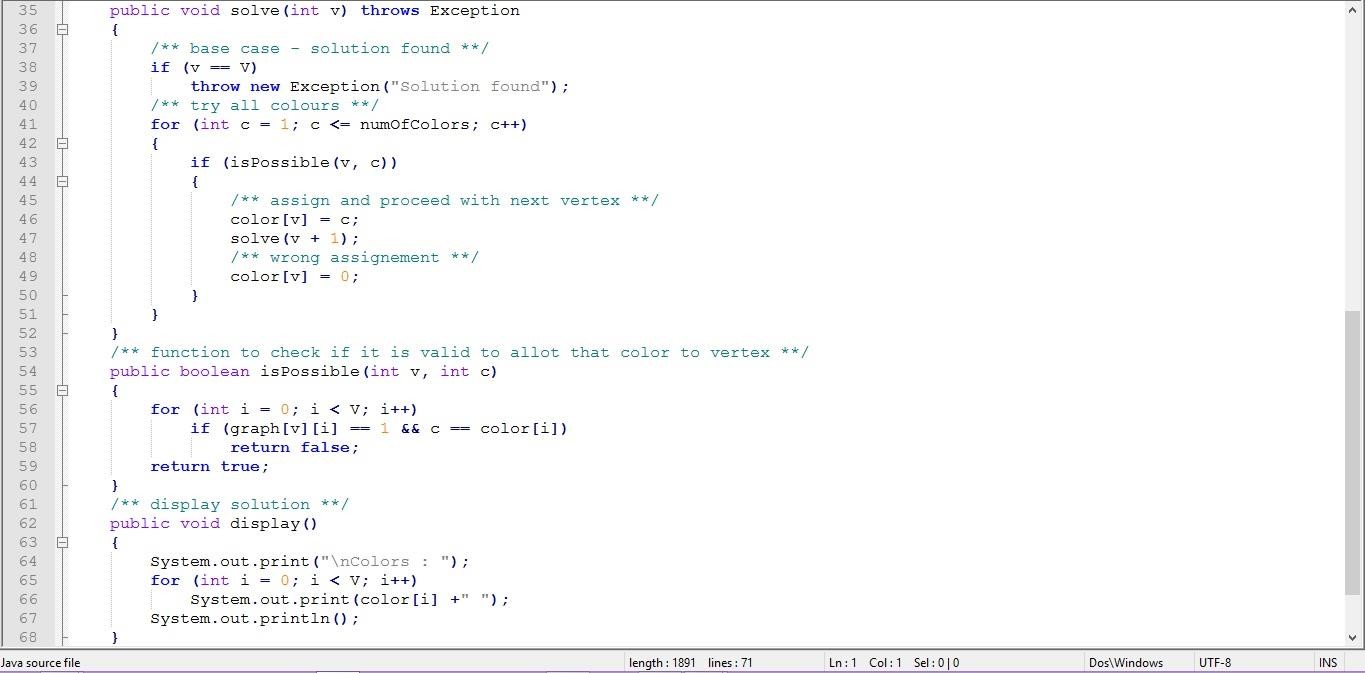
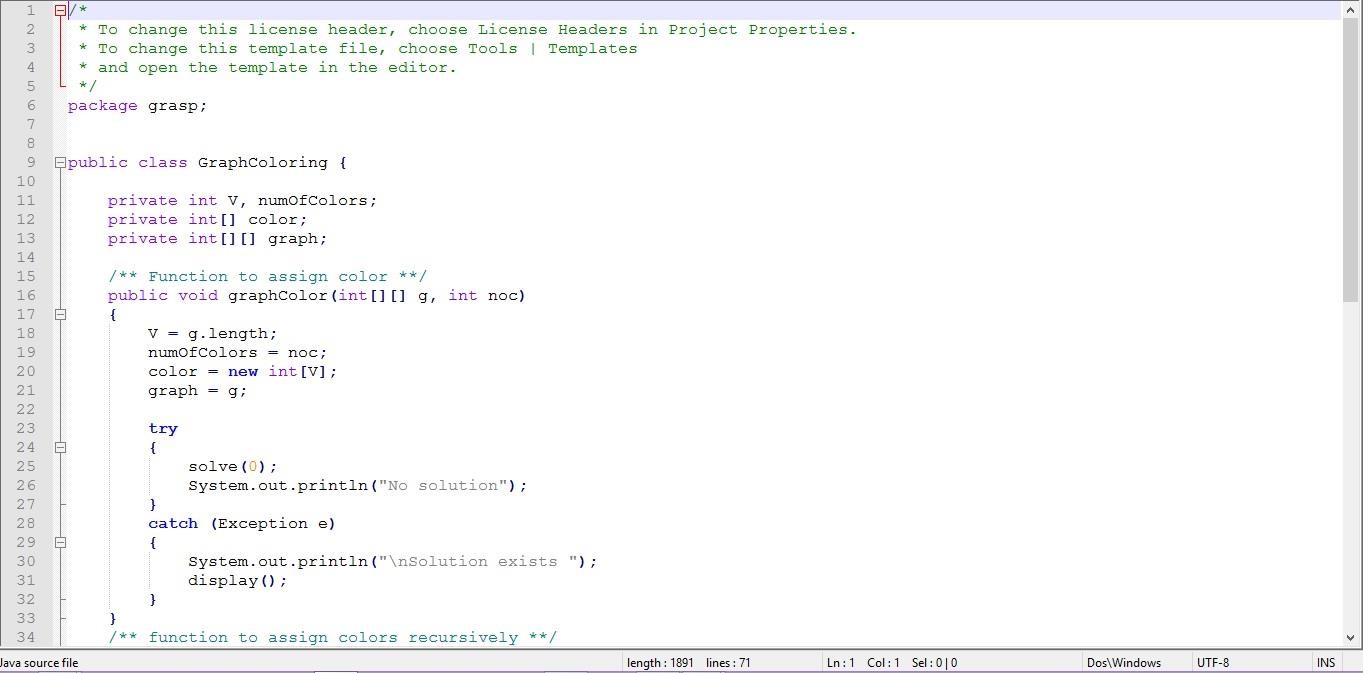
**Code for DFS:-**



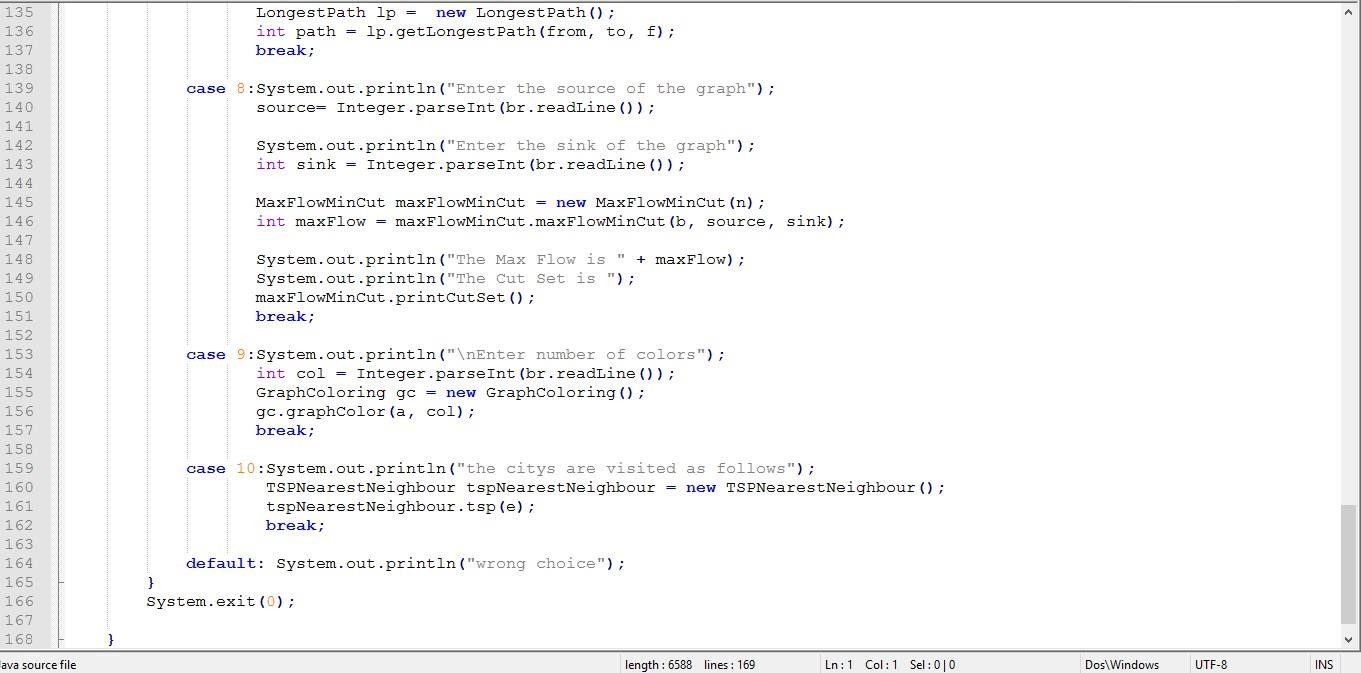
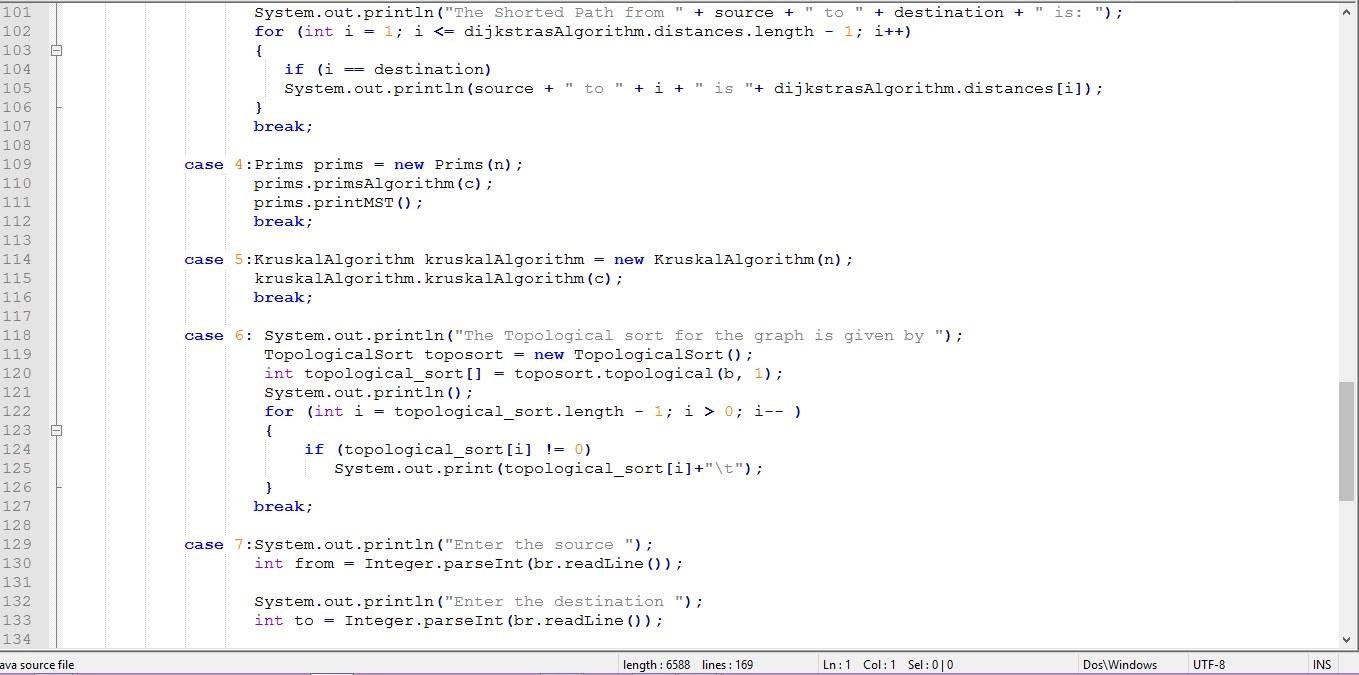
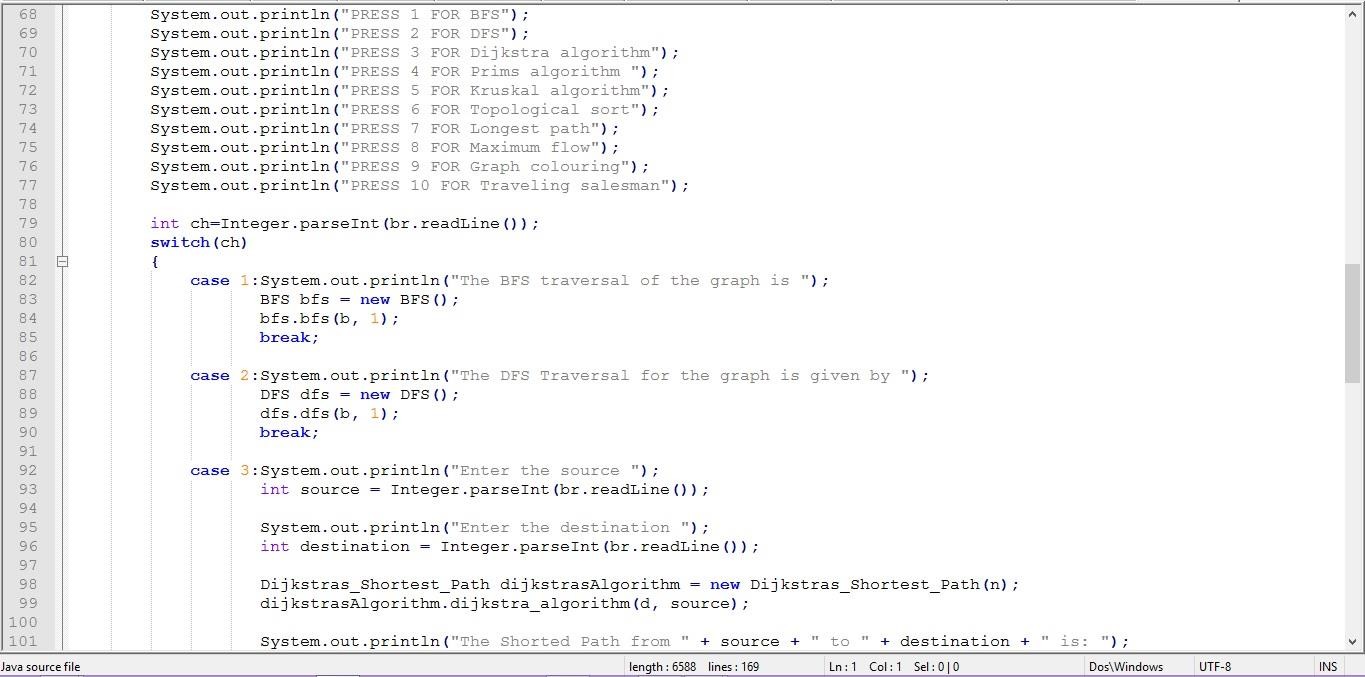
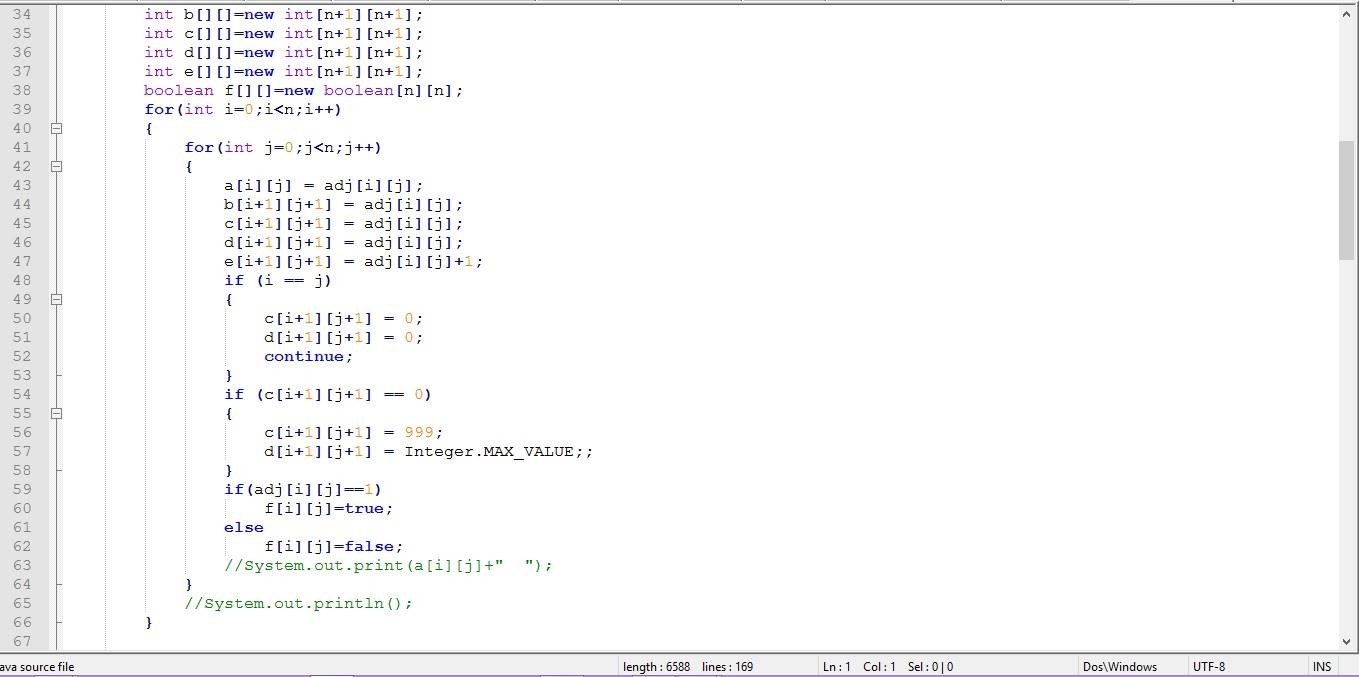
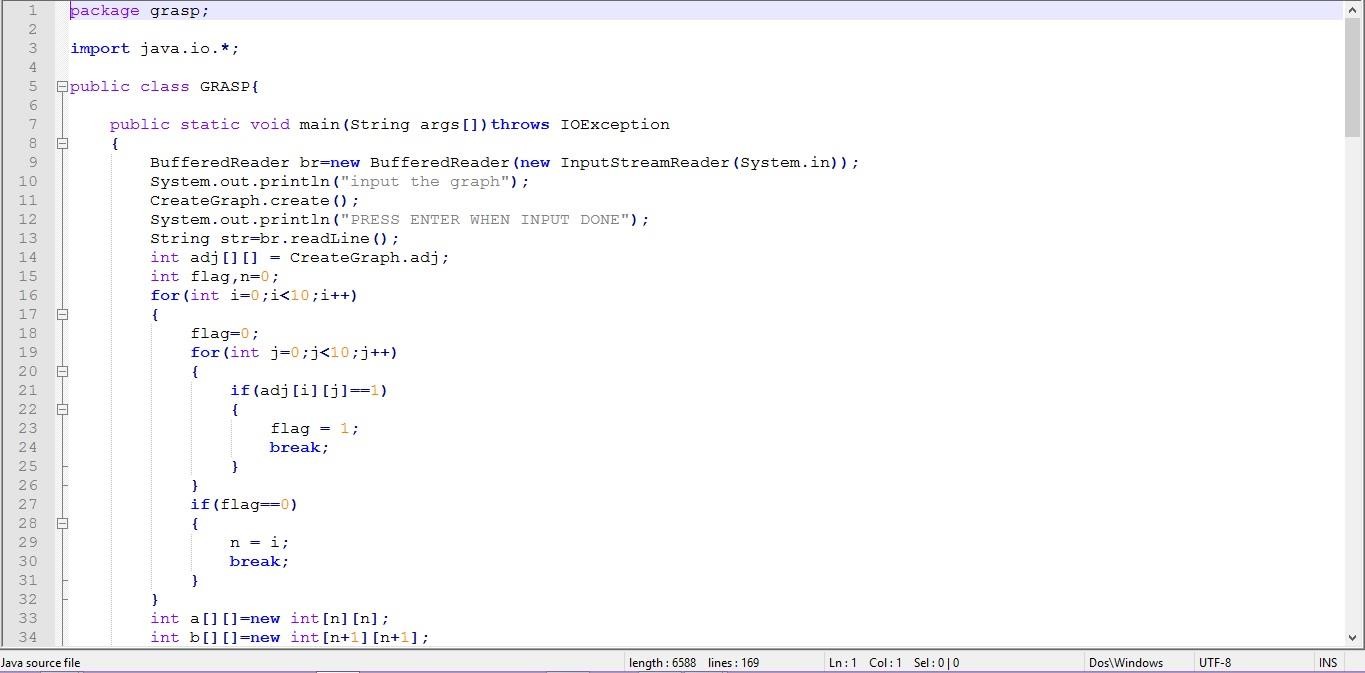
**Code for Diikstra Shortest path: -**



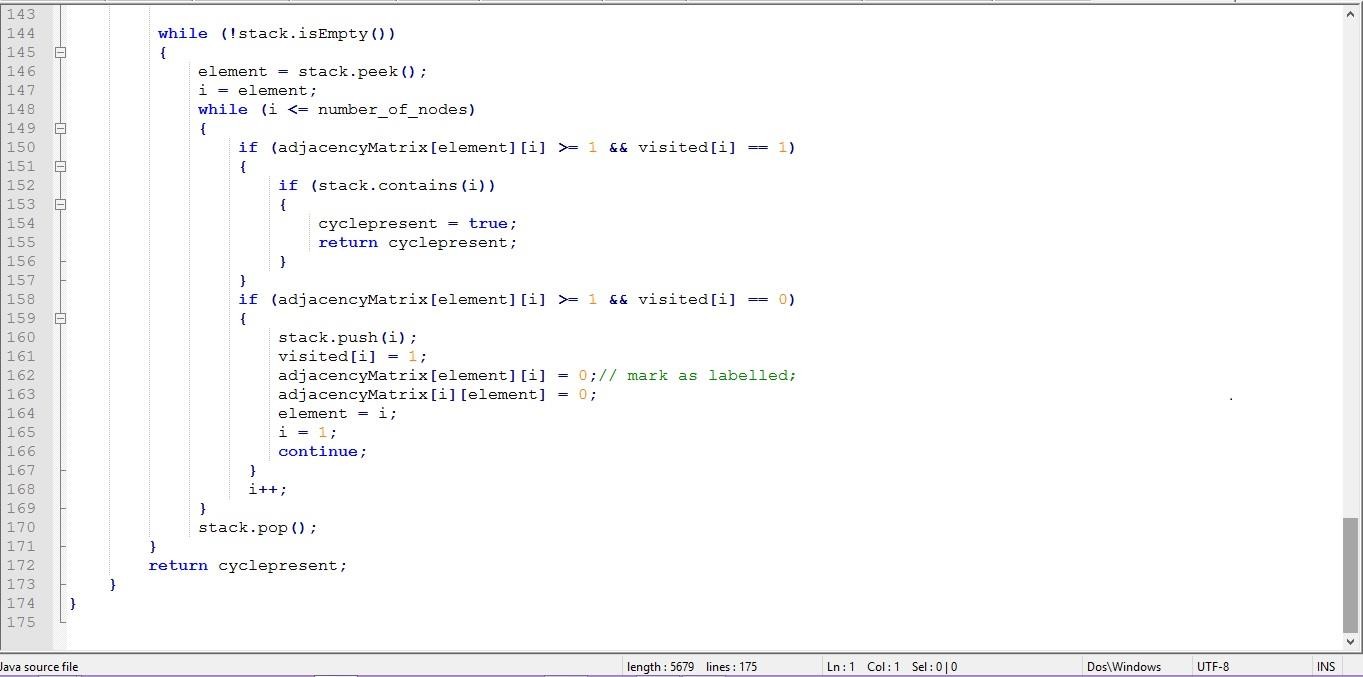
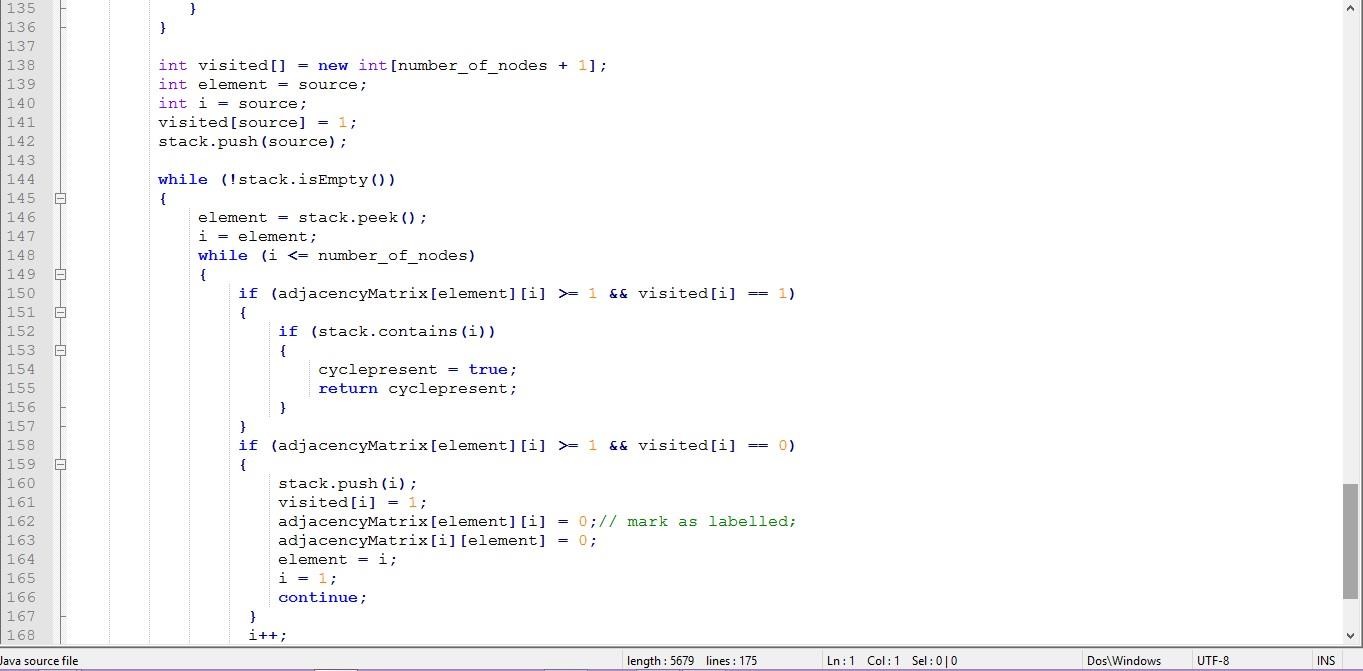
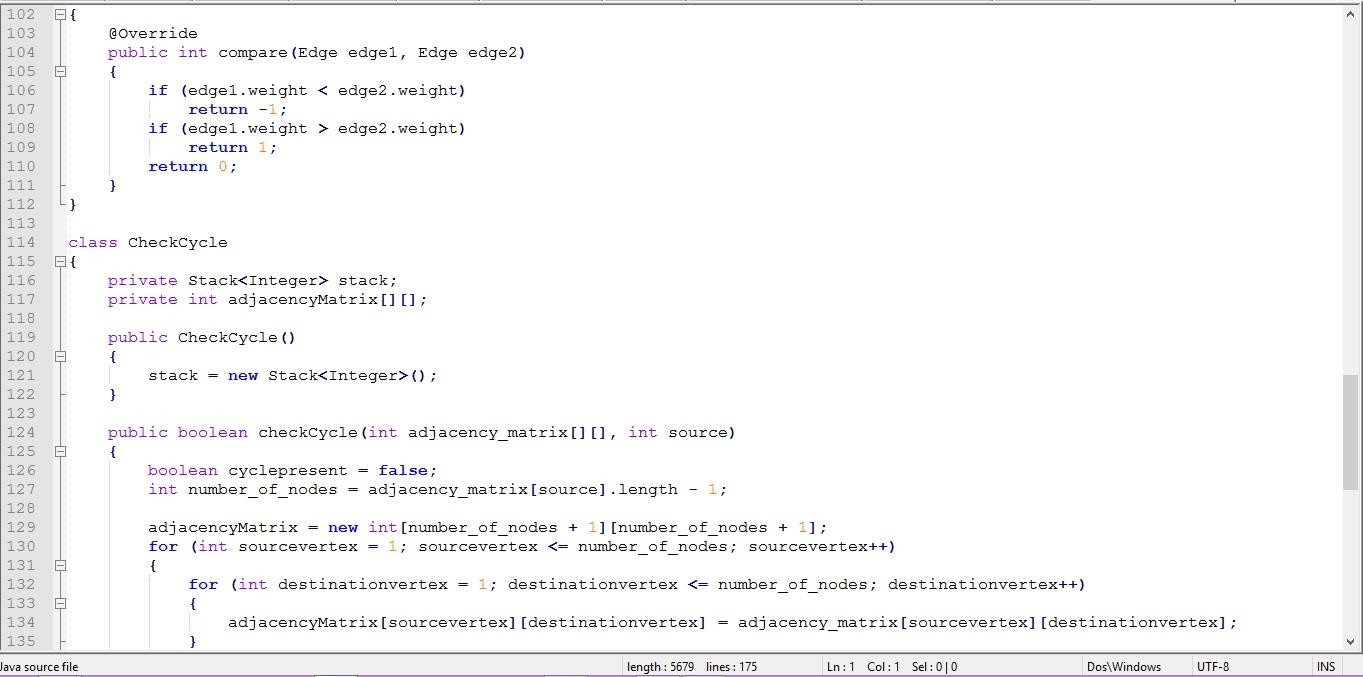
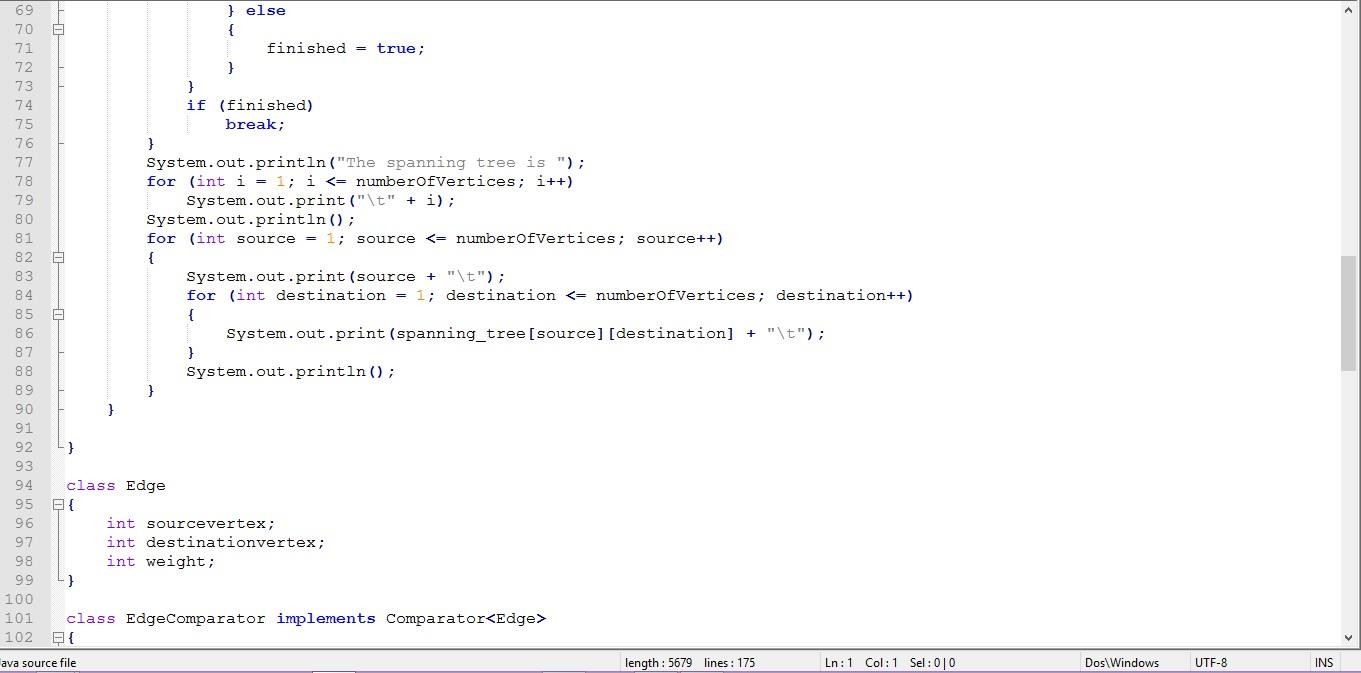
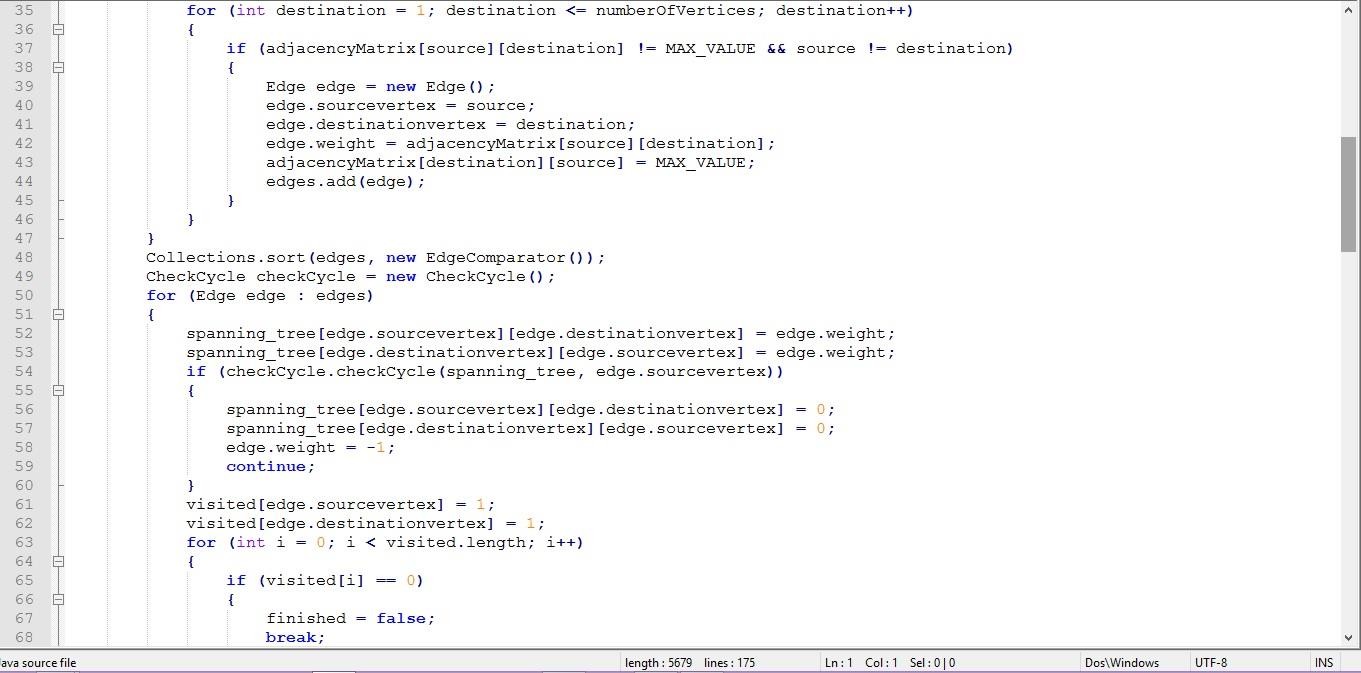
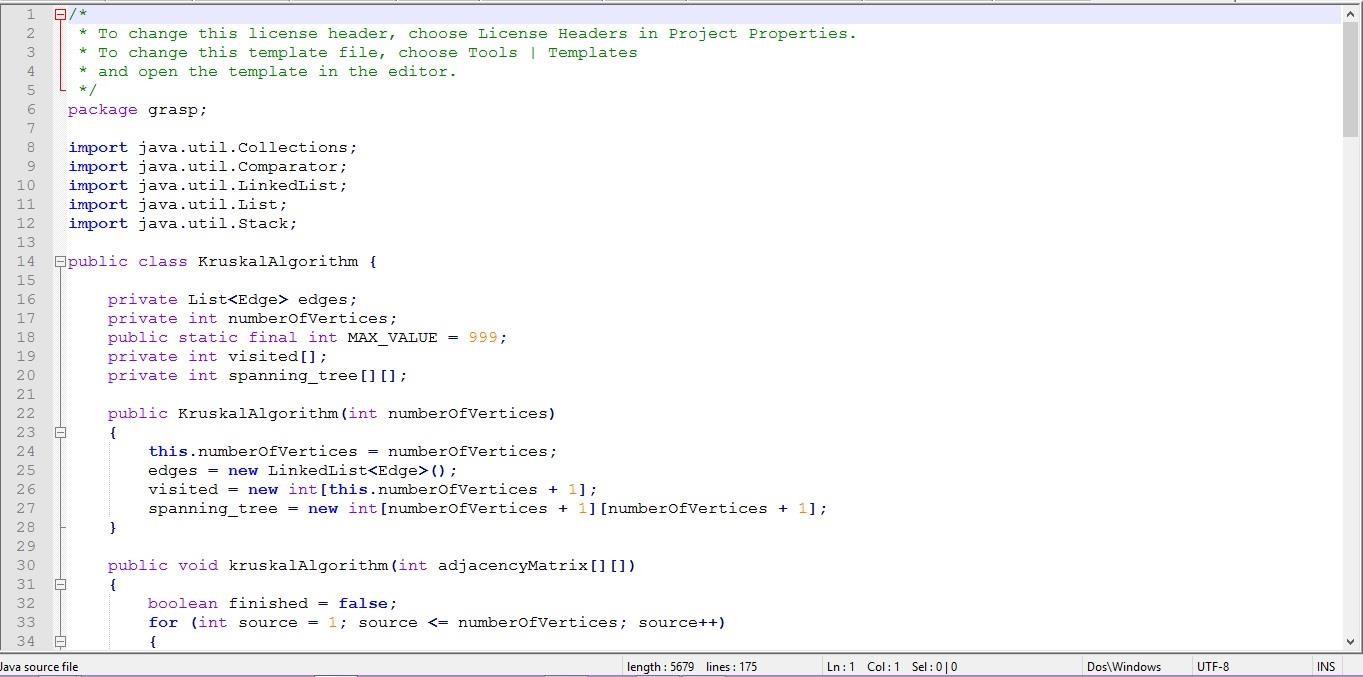
**Code for Graph Colouring: -**



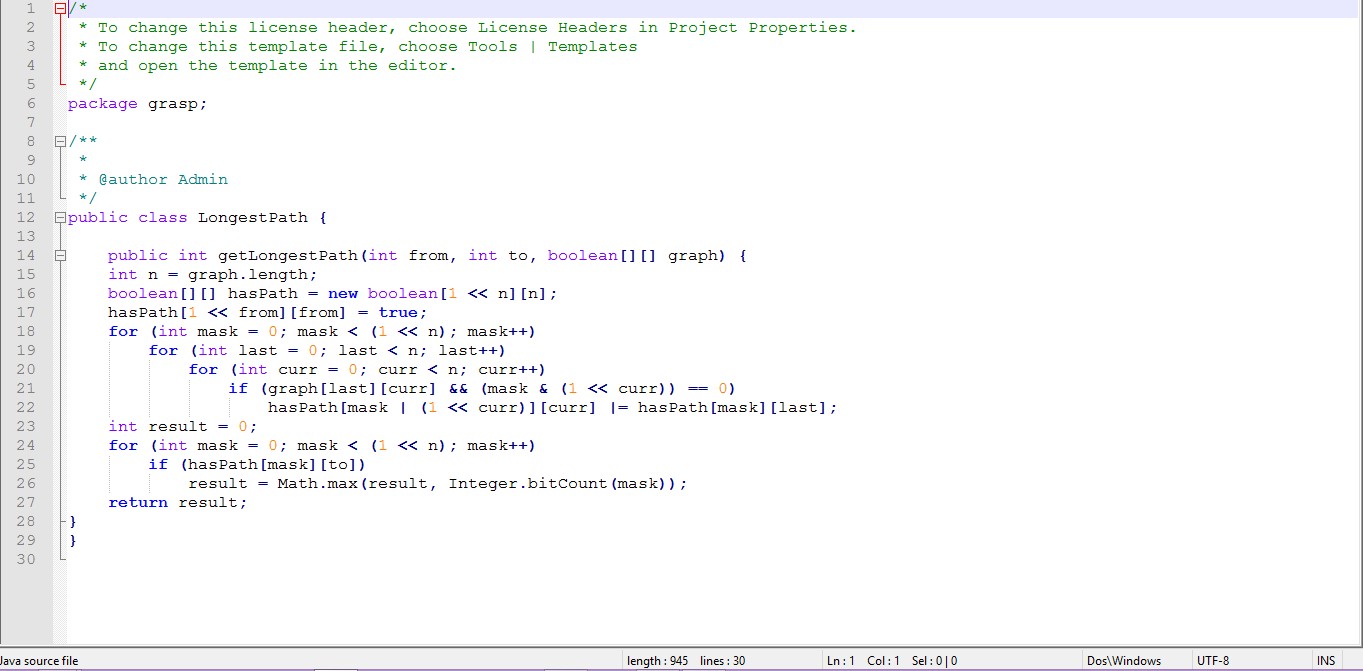
**Code for GRASP file: -**



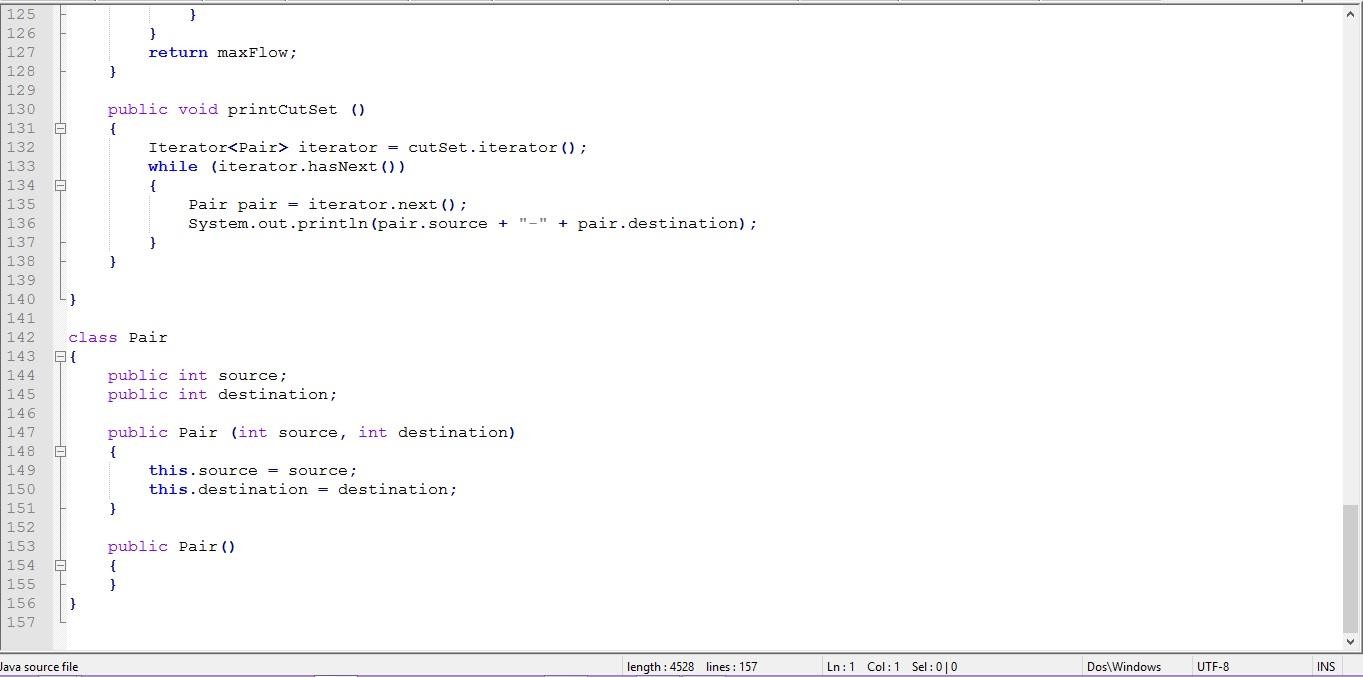
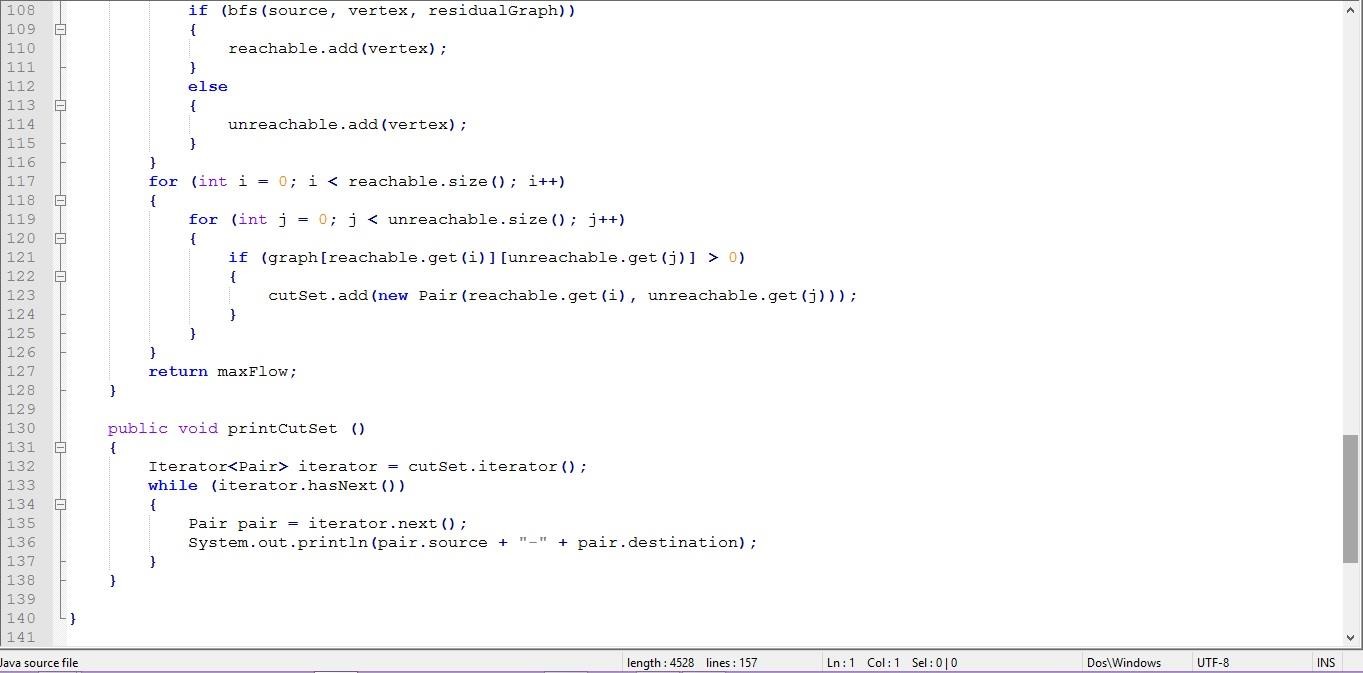
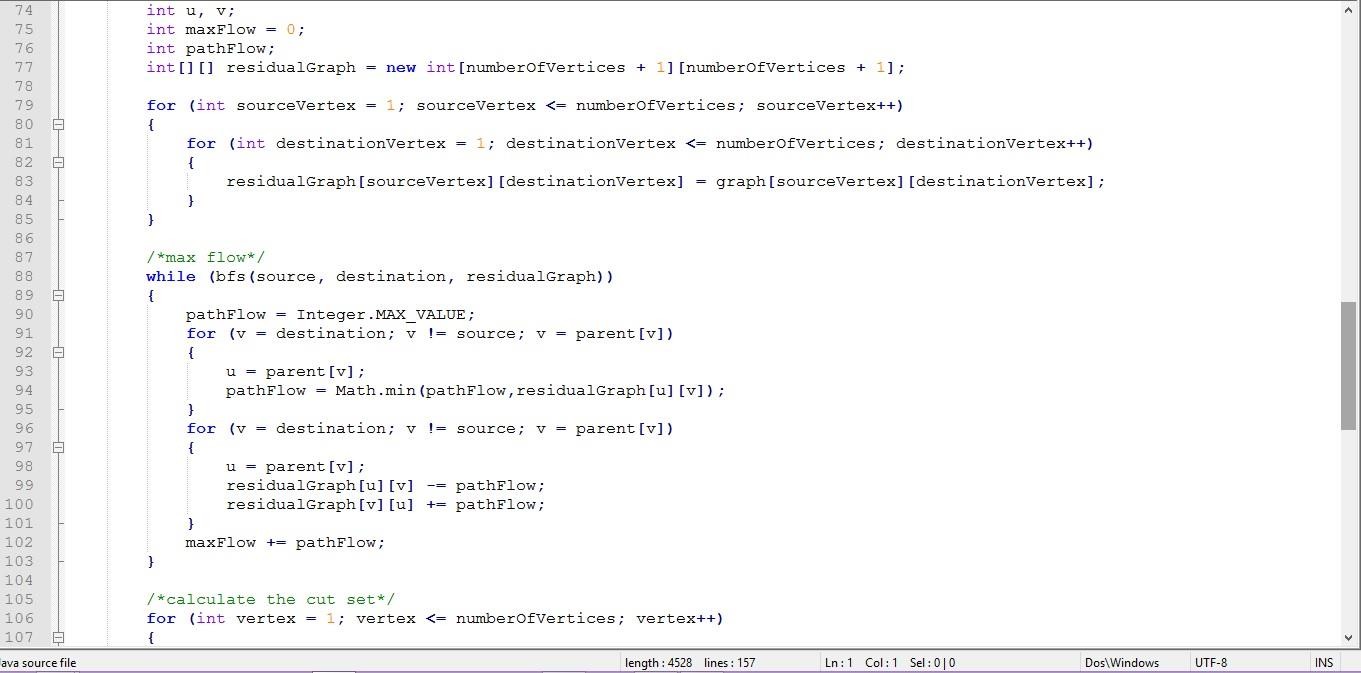
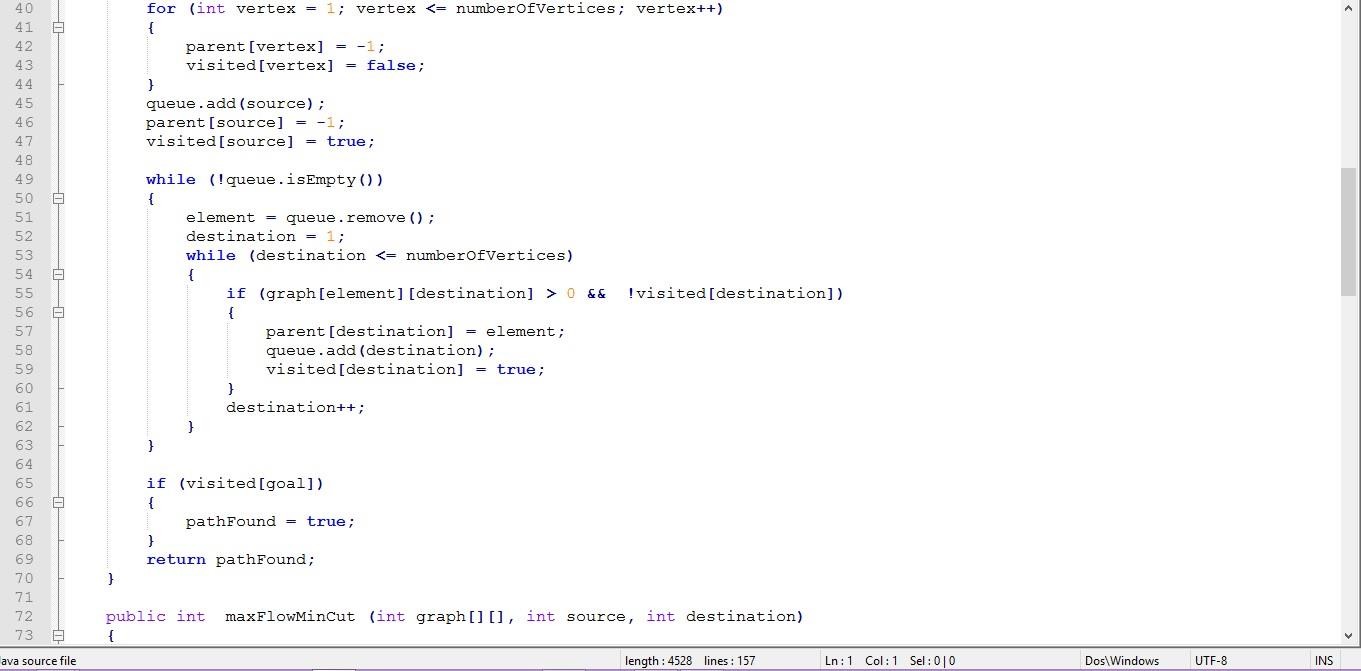
**Code for Kruskal Algorithm: -**



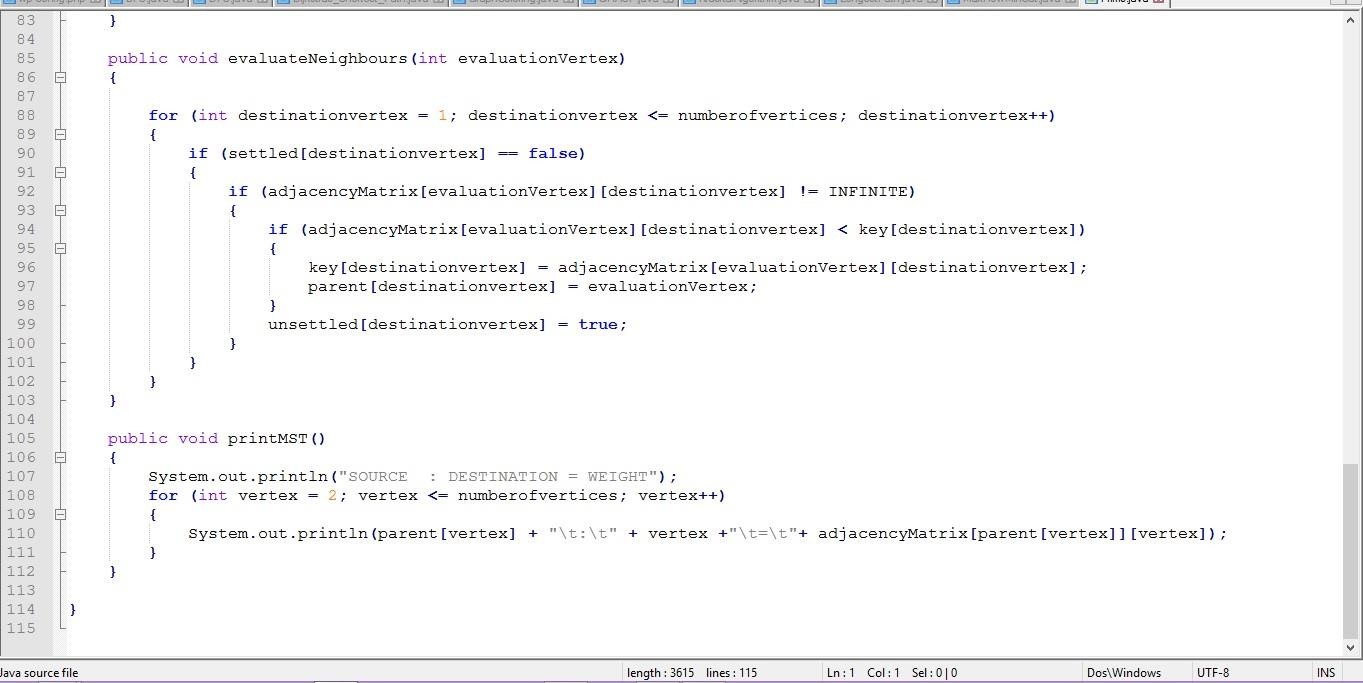
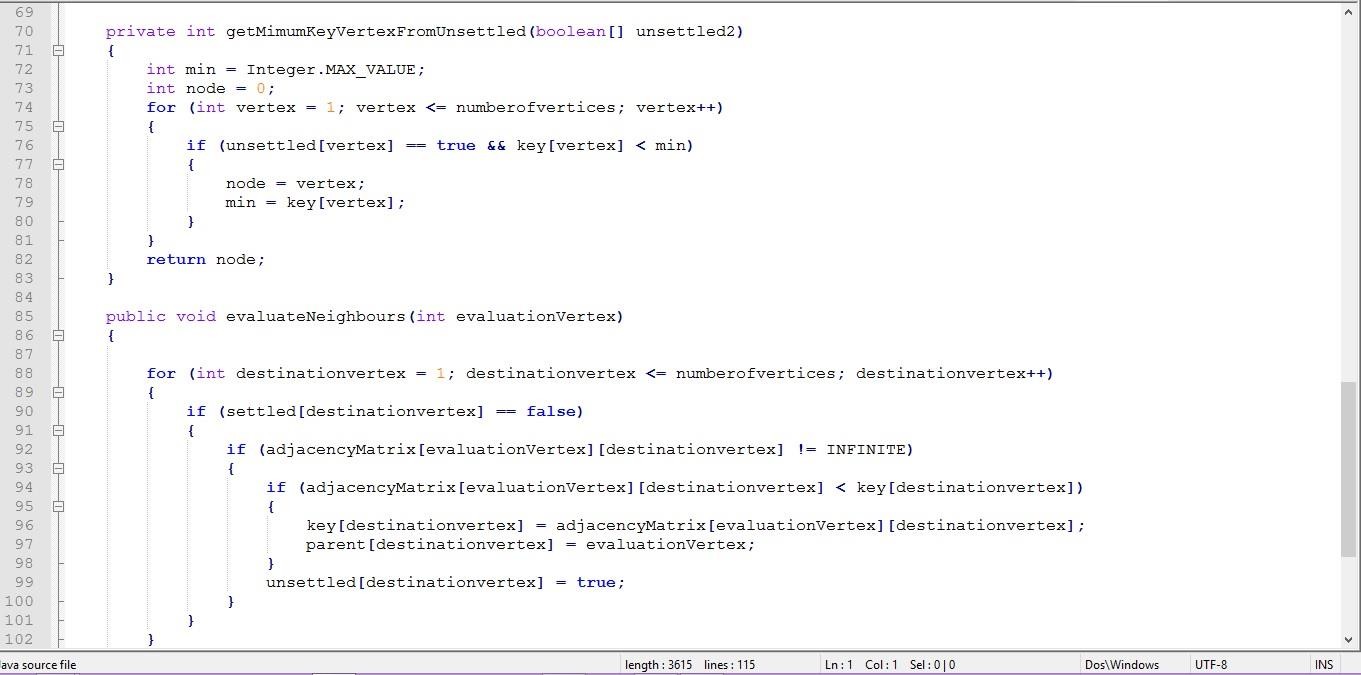
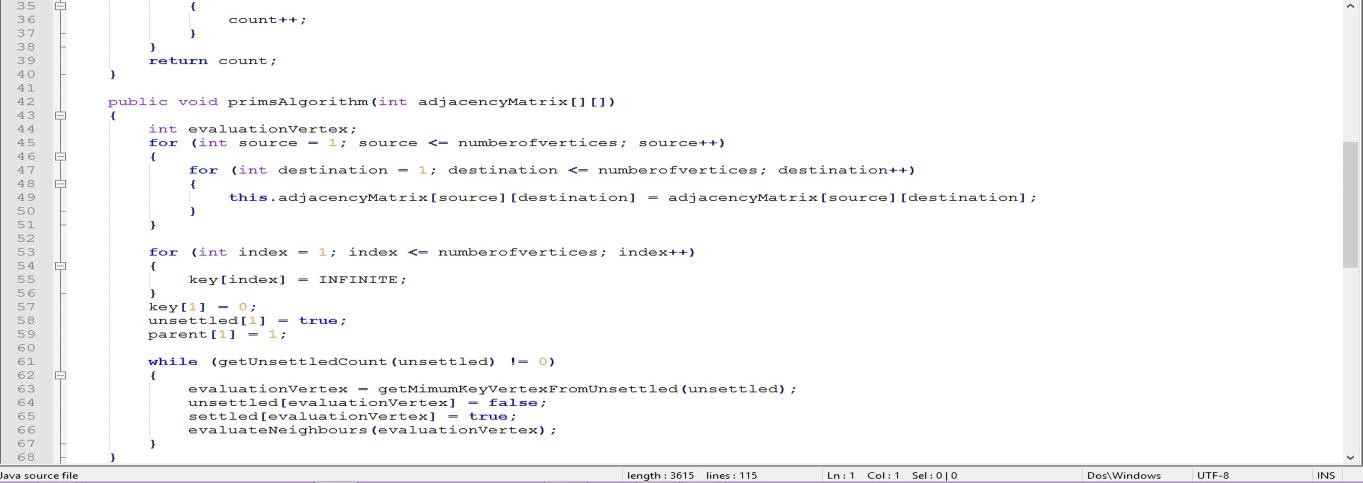
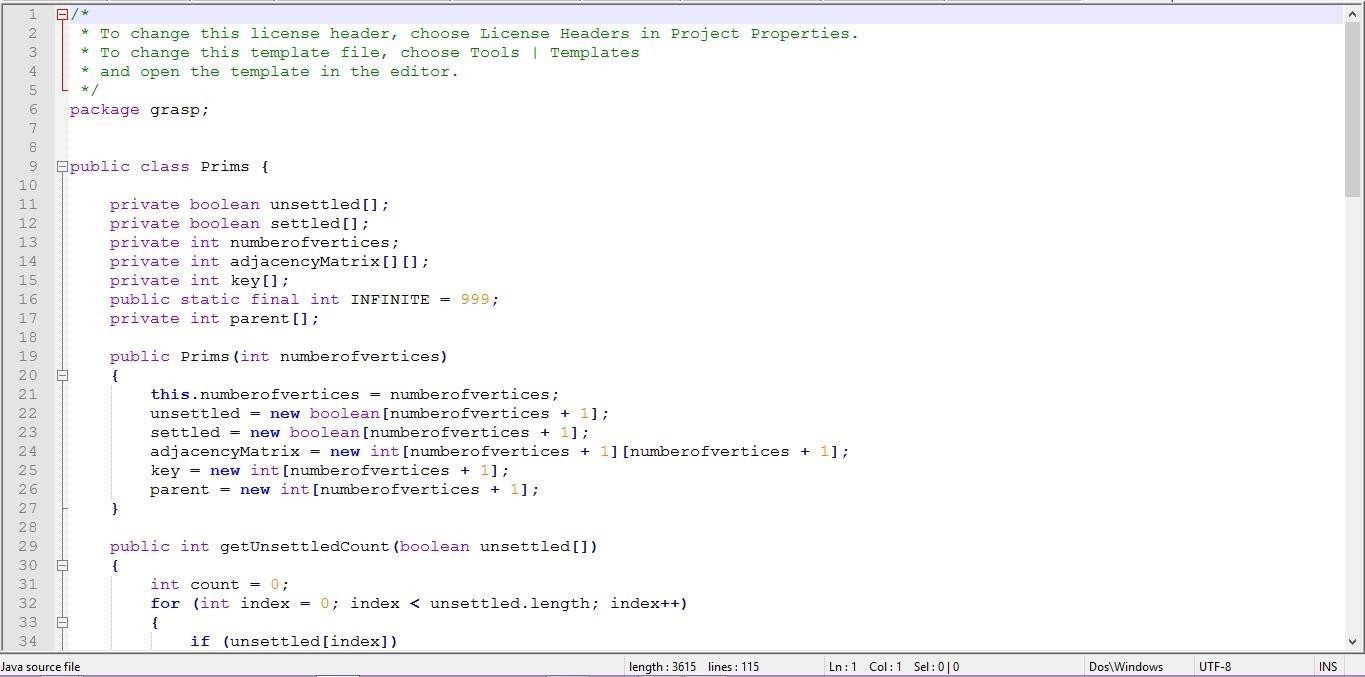
**Code for longest path: -**



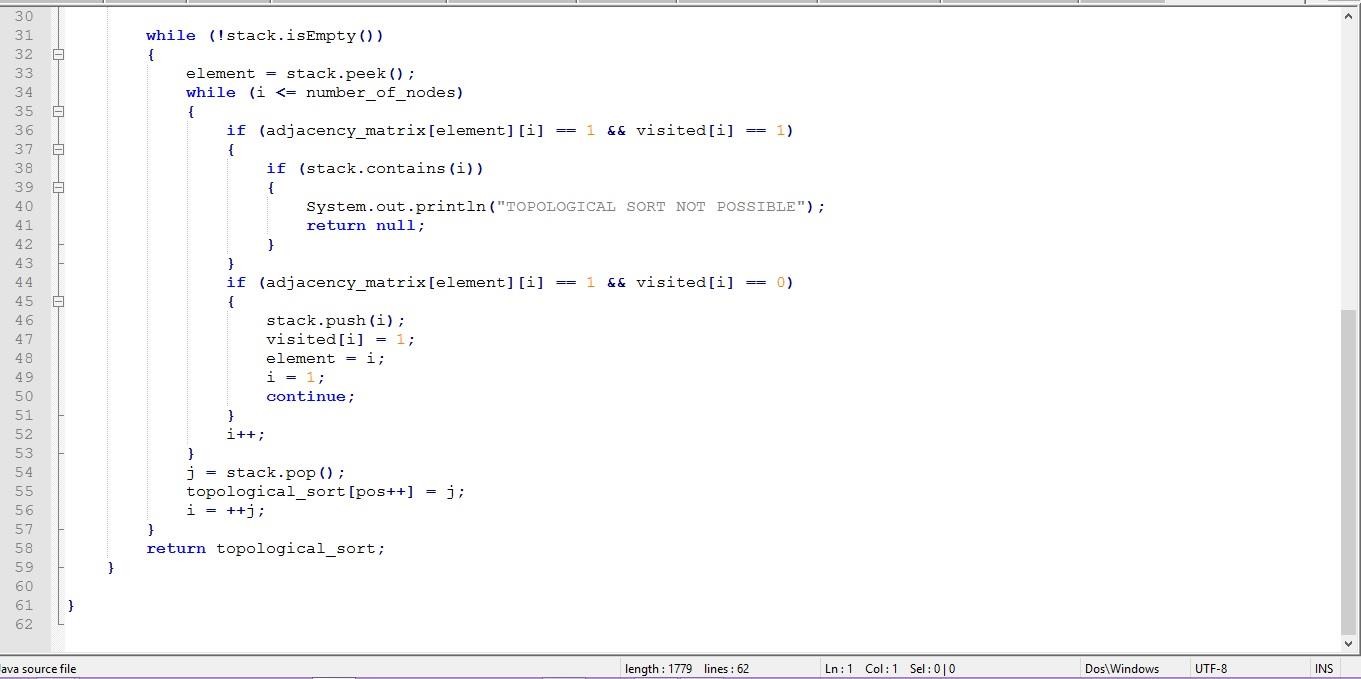
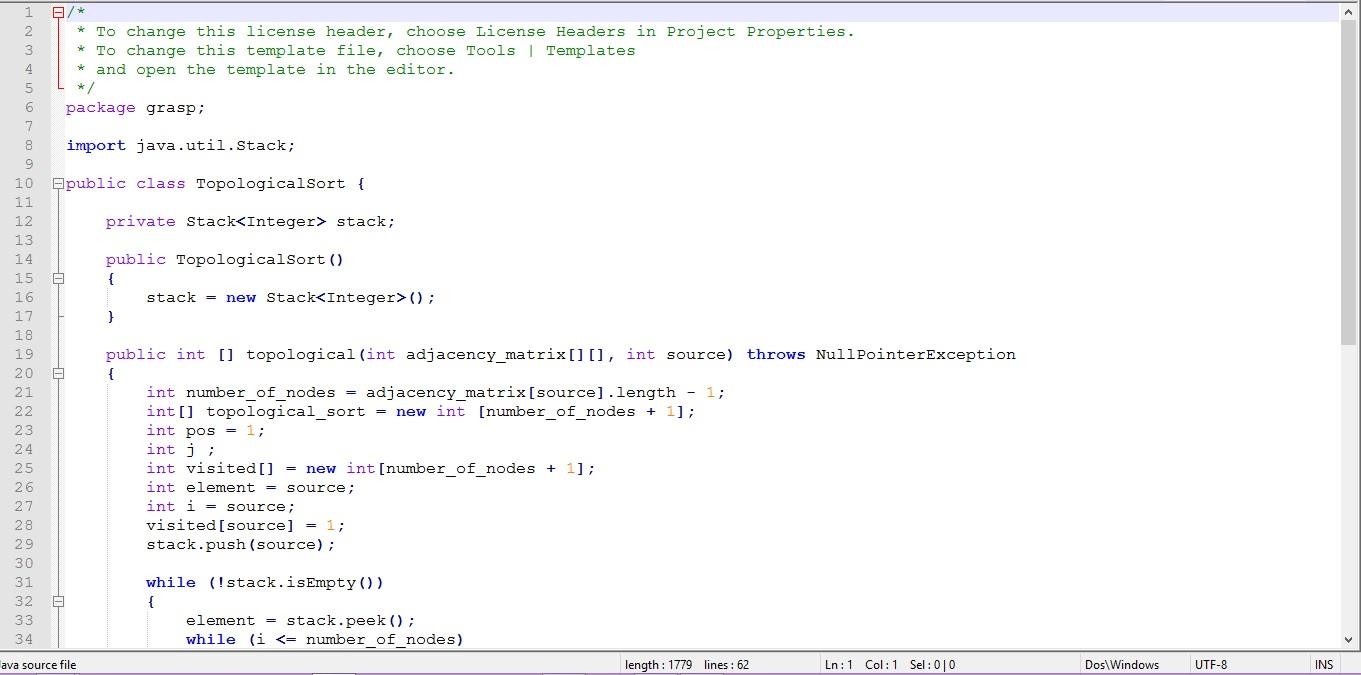
**Code for maximum and minimum flow:-**



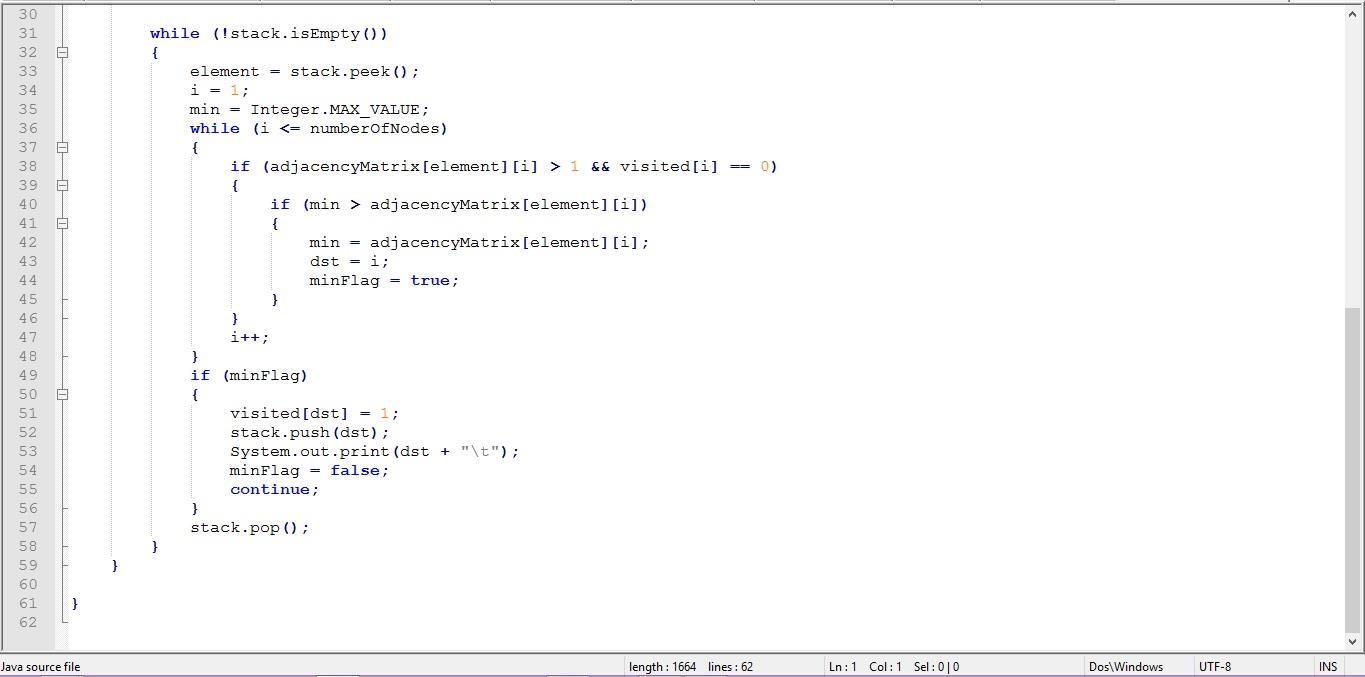
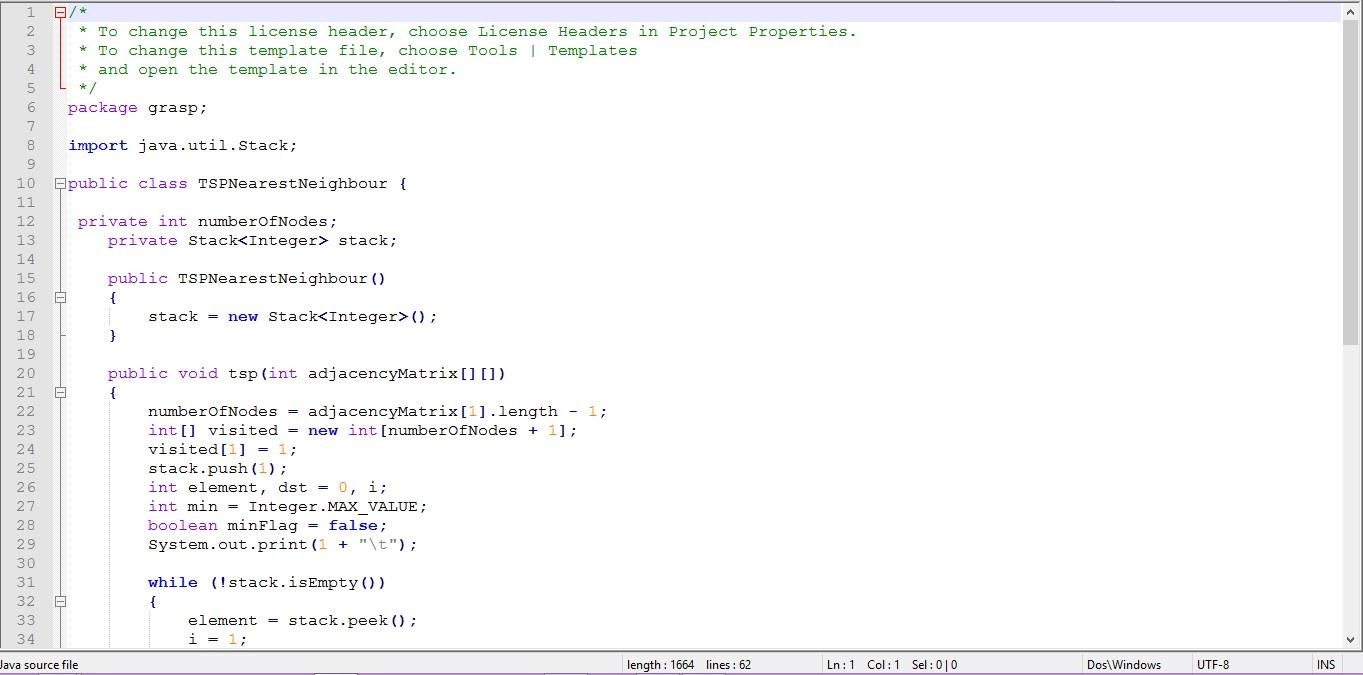
**Code for prims: -**



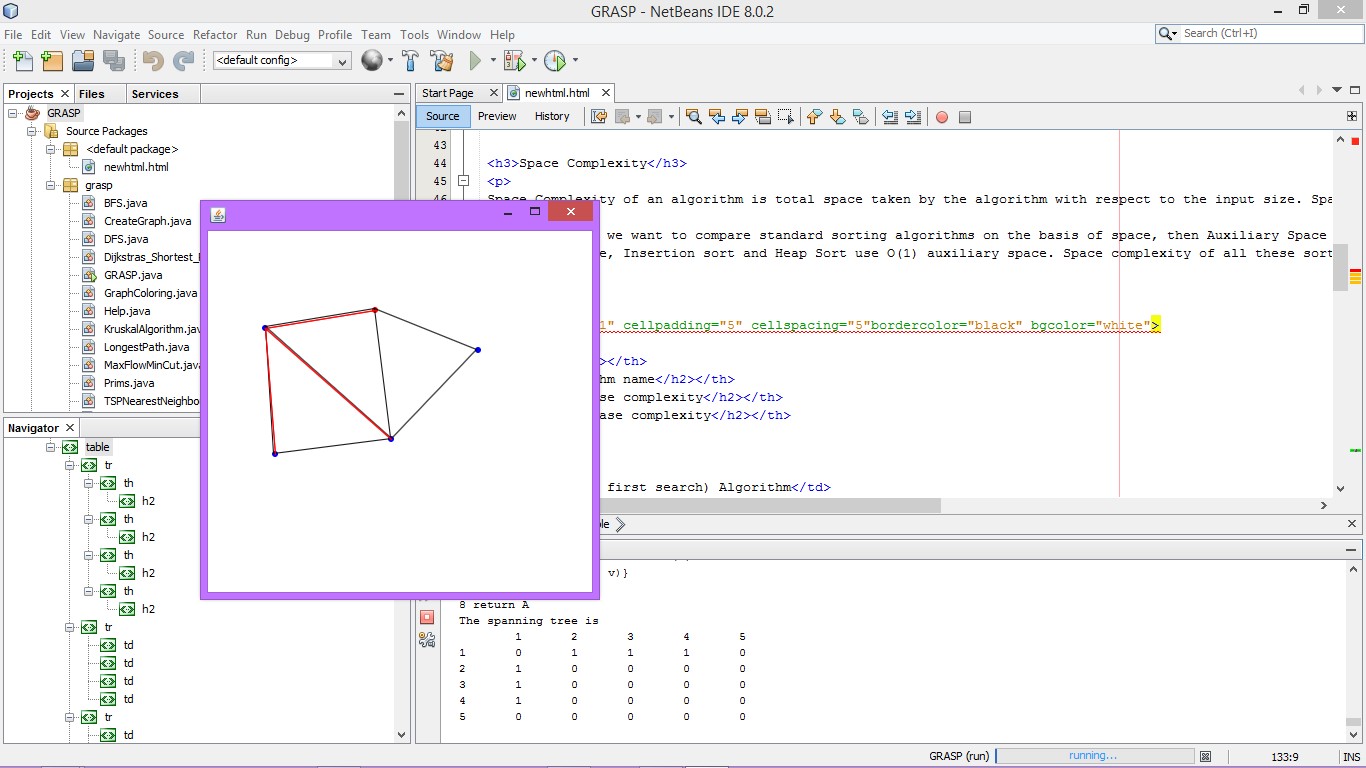
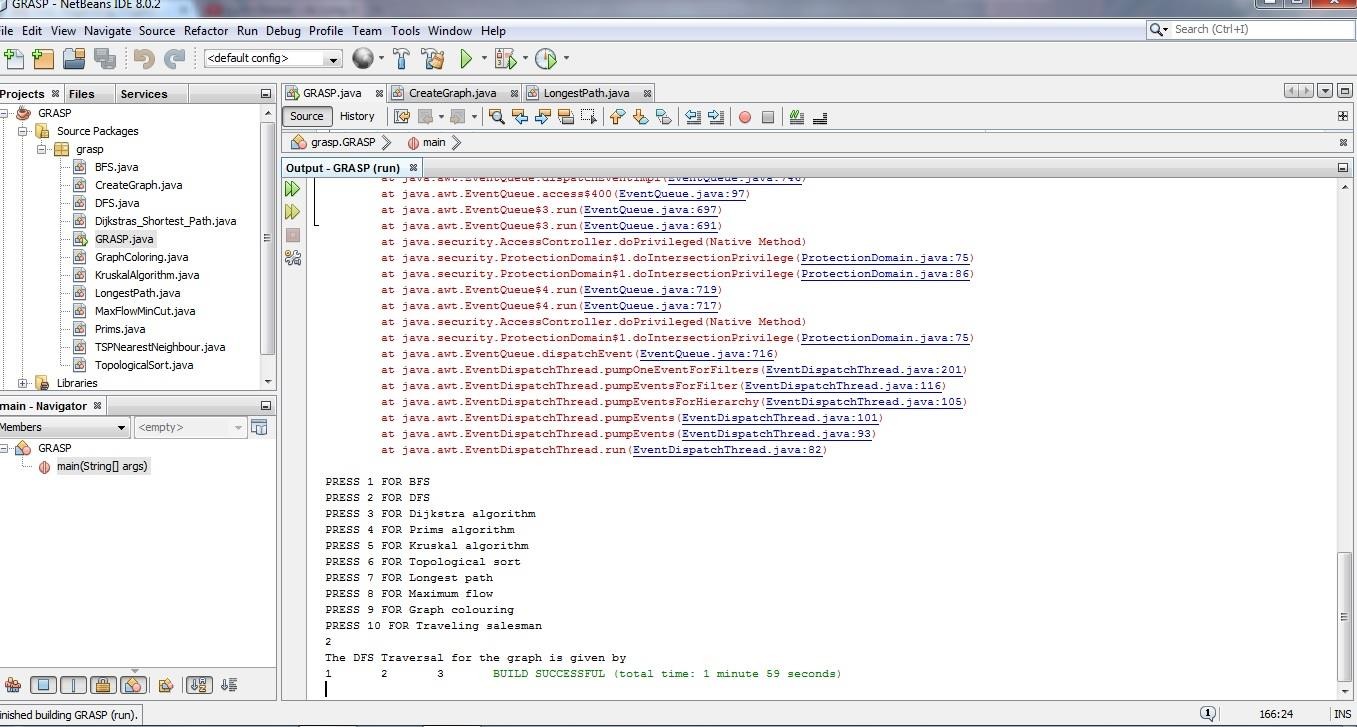
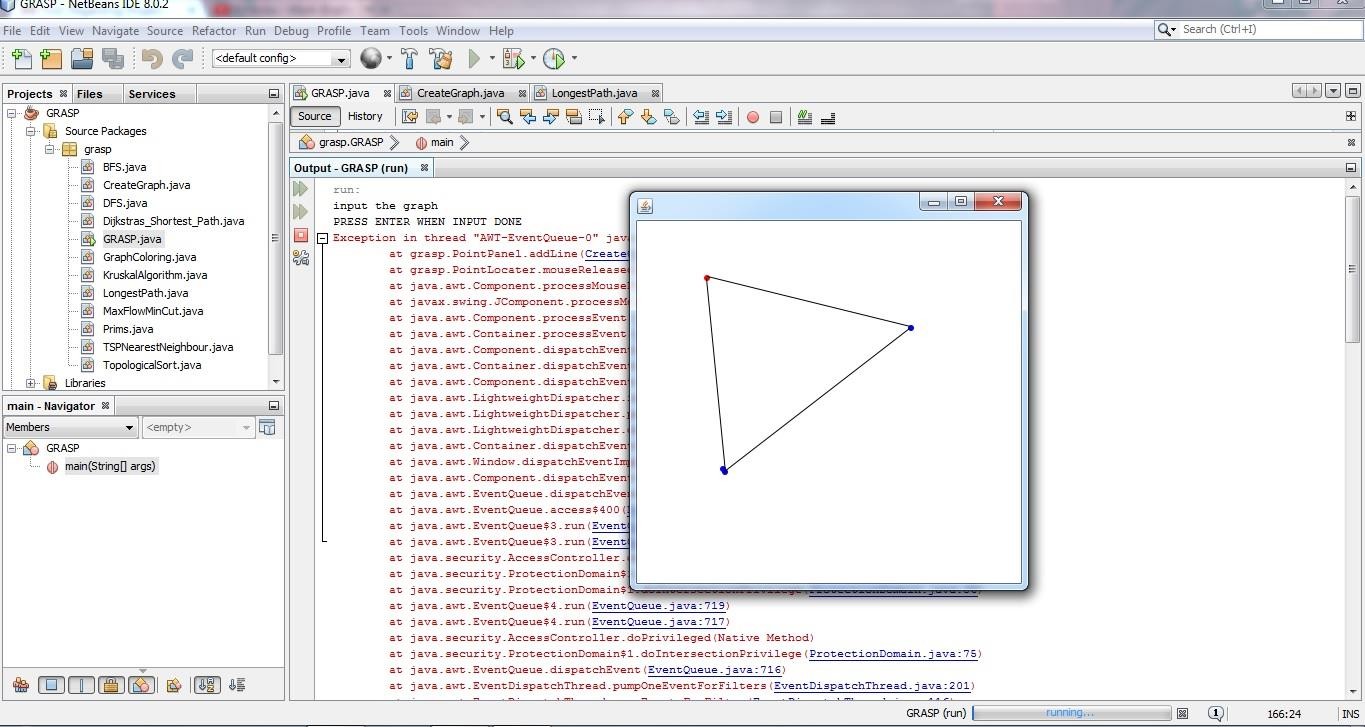
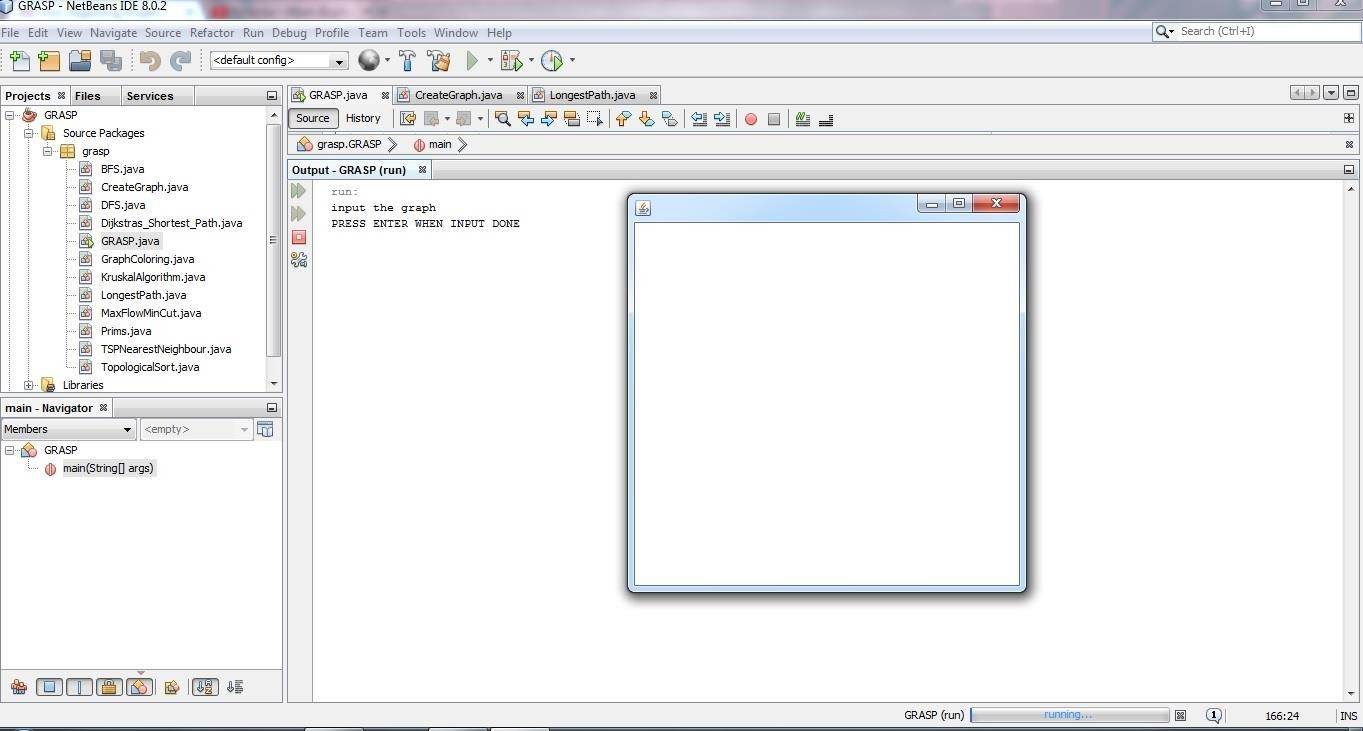
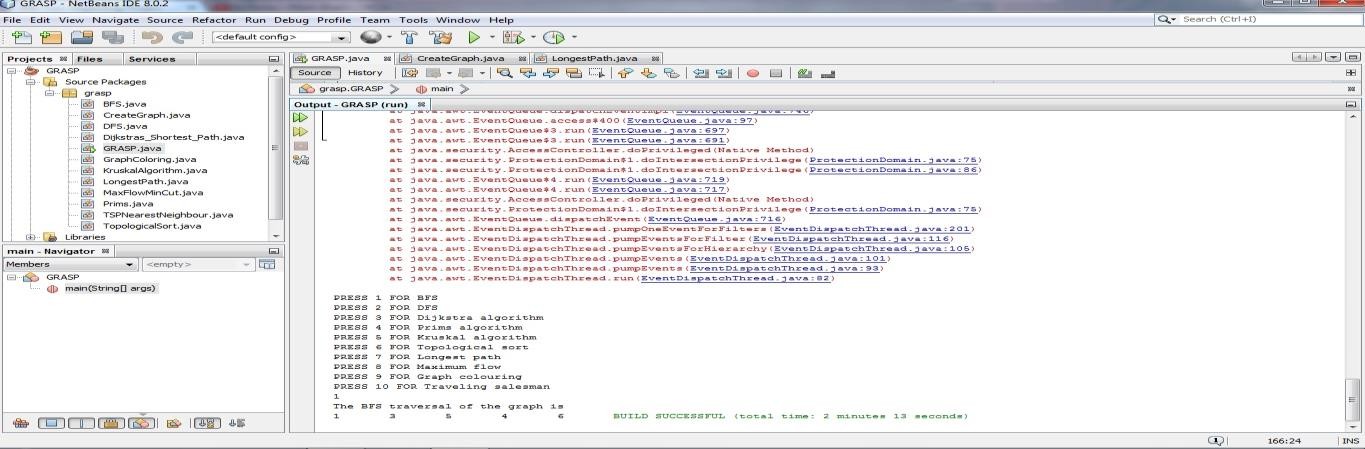
**Code for topological sort: -**



**Code for nearest Neighbour problem: -**



**Snapshots of GRASP Problem Domain: -**



**7.0 APPENDICES**

**A. Bibliography/List of reference: -** The various list of reference is given below. In designing the GRASP, we took help from various internet resources and books and various help source for java API Mouse Listener.

 <https://docs.oracle.com/javase/tutorial/uiswing/events/mouselistener.html>

 <https://www.google.com/>

 <https://www.wikipedia.org/>

 Various books and internet resources.

[ http://www.developer.com/net/vb/article.php/874351/A-Simple-Java-](http://www.developer.com/net/vb/article.php/874351/A-Simple-Java-Drawing-Tool.htm) [Drawing-Tool.htm](http://www.developer.com/net/vb/article.php/874351/A-Simple-Java-Drawing-Tool.htm)

 Various open source project from GitHub and open source libraries.

 https:/[/www.youtube.com/](http://www.youtube.com/)

 Software requirements Specification book by R. S. Pressman.

**B. List of Figures: -**

Input screen…………………………………………………………………....9

Complexity Table……………………………………………………………19

Summarisation (2.4.5) ……………………………………………………….20

Graphical output (2.4.6) ………………………………………………….….20

Use case Diagram …………………………………………………………....21

Class Diagram…………………………………………………………….….21

Sequence Diagram…….……………………………………………………...22

Level-0 DFD………………………………………………………………….23

Level-1 DFD………………………………………………………………….23

Level-2 DFD………………………………………………………………….24

ER-Diagram…………………………………………………………………..25

Snapshots………………………………………………………………….37-55

**C. List of Tables: -**

Table no 1 : Software interface……………………………………………….10

Table no 2 : Unit Testing……………………………………………………..28