

BHANDARKARS ARTS AND SCIENCE COLLEGE, KUNDAPURA



Department of Computer Science

Bachelor of Computer Application

VI SEMESTER BCA MINI PROJECT PROPOSAL

2023-24

VISUALIZATION AND ANALYSIS OF ALGORITHMS

Final Year Project Proposal

SN	Student Name	Reg. No.
01	Swasthik Devadiga	U05BA21S0030
02	Abhishek S Kotari	U05BA21S0028
03	Chethan Kumar	U05BA21S0041

Guide name: Mr. Ganesh K

1. Abstract

In our Project, We plan to implement several algorithms including Ada concept of Knapsack ,Tower of Hanoi, Job Scheduling, and Travelling Salesman. These algorithms are important in the field of computer science and have practical application in various industries. By studying and implementing these algorithms, we aim to deepen my understanding of algorithmic problem solving and optimize their performance.

2. Background and introduction of sponsor/client and problem statement

In this project we aim to explore and analyse the Knapsack problem, Tower of Hanoi, Travelling Salesman problem. Our goal is to understand the complexities of these problems, develop efficient algorithms to solve them, and evaluate their performance in various scenarios. By addressing these problems, we aim to contribute to the field of algorithm design and optimization.

3. Aims and Objectives

- To compare and analysis performance of all algorithm.
- To present the analysis result graphically.
- To analyse and compare the time complexity of the implemented algorithms.
- To provide insights into the practical implications of algorithmic choices on real-world applications.

4. Scope of the Project

- Sole focus is to compare various ADA programming algorithms and its complexity.
- This is mini project with very limited time.
- The project will adhere to ethical guidelines for research and academic integrity.
- Key features such as strong typing, concurrency, and exception handling will be considered.

6. Proposed Methodology

Modules:

- **Knapsack**

The Knapsack Problem is a classic optimization problem in computer science and combinatorial optimization. It can be stated as follows:

Given a set of items, each with a weight and a value, determine the number of each item to include in a knapsack of a fixed capacity to maximize the total value while keeping the total weight within the capacity limit.

- **Travelling Salesman**

The Traveling Salesman Problem (TSP) is another classic problem in the field of optimization. It can be stated as follows:

Given a list of cities and the distances between each pair of cities, the task is to find the shortest possible route that visits each city exactly once and returns to the origin city (the salesman's home city).

- **Tower Of Hanoi**

The Tower of Hanoi is a classic problem in the field of computer science and mathematics, often used to illustrate principles of recursion and problem-solving strategies. It is a simple but intriguing puzzle that consists of three rods and a number of disks of different sizes which can slide onto any rod. The objective is to move the entire stack of disks from one rod to another

- **Job Assignment**

It is required to perform all jobs by assigning exactly one worker to each job and exactly one job to each agent in such a way that the total cost of the assignment is minimized.

- **Dijkstra's Algorithm**

The Single Source Shortest Path (SSSP) algorithm is a method used to find the shortest paths from a single source vertex to all other vertices in a weighted graph. In other words, it calculates the minimum distance from one vertex (the source) to all other vertices in the graph, considering the weights assigned to the edges.

7. Proposed solution and anticipated results

- **Knapsack:-**

Input:-

Enter No. of Items : 4

Enter Weights: 3 5 4 6

Enter Values: 36 25 41 34

Enter Maximum Capacity: 15

Output:-

Optimal Solution is : 111

Selected Packs:

Package 4 with w=6 and values=34

Package 3 with w=4 and values=41

Package 1 with w=3 and values=36

- **Travelling Salesman:-**

Input:-

Enter no. of cities: 4

Enter cost matrix:

0	10	15	20
5	0	9	10
6	13	0	12

8 8 9 0

Enter Source City: 1

Output:-

Path is : 1->2->4->3->1

Minimum Cost : 35

● **Tower Of Hanoi:-**

Input:-

Enter no. of disk: 3

Output:-

Move disk 1 from rod A to rod C

Move disk 2 from rod A to rod B

Move disk 1 from rod C to rod B

Move disk 3 from rod A to rod C

Move disk 1 from rod B to rod A

Move disk 2 from rod B to rod C

Move disk 1 from rod A to rod C

● **Job Assignment:-**

Input:-

Jobs: J1, J2, J3, J4

Persons: P1, P2, P3, P4

Cost matrix:

9 2 7 8

6 4 3 7

5 8 1 8

7 6 9 4

Output:-

J1->p2=6

J2->p1=2

J3->p3=1

J4->p4=4

13

● **Dijkstra's Algorithm:-**

Input:-

Enter The Number of nodes : 5

Enter the Edge Matrix :

0 3 0 7 0

3 0 4 2 0

0 4 0 5 6

7 2 5 0 4

0 0 6 4 0

Output:-

Distance from Source to Node 1: Distance=0
Distance from Source to Node 2: Distance=3
Distance from Source to Node 3: Distance=7
Distance from Source to Node 4: Distance=5
Distance from Source to Node 5: Distance=9

8 Schedule of activities and Gantt chart

Activity	Tentative Date
Completing the proposal for the Project	Week 1
Completing the First Module(Knapsack)	Week 2
Completing the Second Module (Travelling Salesman)	Week 3
Completing the Third (Tower Of Hanoi)	Week 4
Completing the Fourth Module (Job Assignment)	Week 5
Completing the Fifth ()	Week 6
Testing the Modules	Week 7-8
Documentation	Week 9
Finalizing the Project	Week 10

9 Requirements

9.1 Hardware Interface:

- Processor - i3, i5 or equivalent.
- RAM - 4GB+
- Hard Disk Drive (SSD)– 512 GB, 1 TB

9.2 Software Interface:

- Browser - Chrome, Firefox or equivalent.
- Operating System – Mac, Windows(latest), Linux
- Front-end (HTML, CSS)
- Back-end (Javascript)
- Editor – Notepad, Visual Studio Code, MS Word.

9.3 Communication Interface:

- Internet, Ethernet interface

10 References

- i. "Introduction to Algorithms" by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein
- ii. **VisuAlgo**: A website with visualizations for various algorithms, including those you are interested in.
- iii. **GeeksforGeeks**: Provides explanations and visualizations for many algorithms.

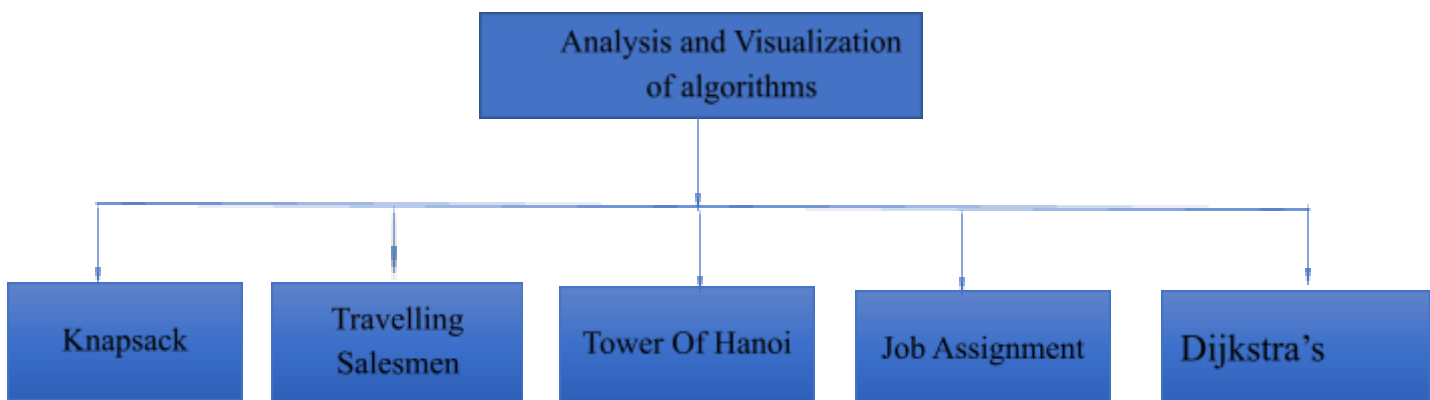
3. SYSTEM DESIGN

3.1 Introduction

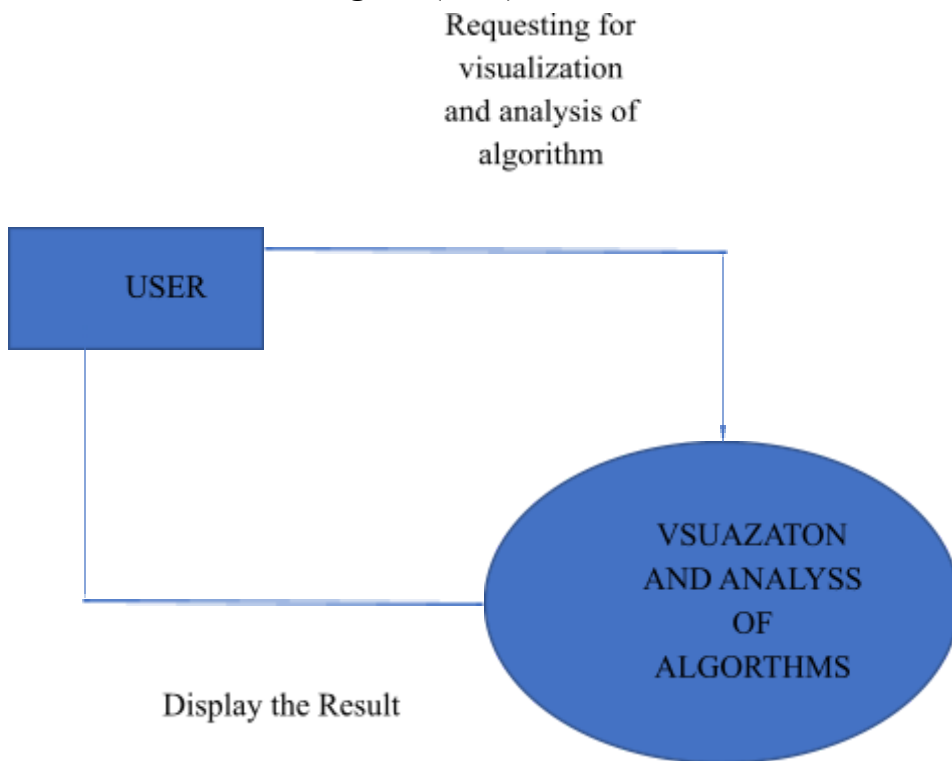
- System design is the process of defining the architecture, module interfaces and data for a system to satisfy the specified requirements.
- The purpose of the design phase is to plan the solution of the problem specified by the requirement documents.
- This is the first step that moving from problem domain to the solution domain.
- The design of the system is essentially a blueprint or a plan for a solution for the system.

3.2 Functional decomposition

Functional decomposition is the process of taking a complex process and breaking it down into its smaller, simpler parts. Using Functional decomposition larger or complex functionalities are more easily understood. It is mainly used during project analysis phase, so each phase can be viewed as software. So, this has modular with some sub modules.



3.3 Context Flow Diagram (CFD)



3.4 Data Flow Diagram (DFDs Level 0, Level 1, Level 2)

Data flow diagram shows the flow of data through system. Data flow diagrams also called the data flow graphs. It views a system as a function that transforms the inputs into desired outputs. It aims to

capture the transformation that taken place within a system to the input data so that eventually the output data is produced.

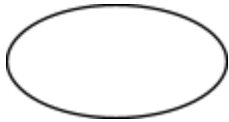



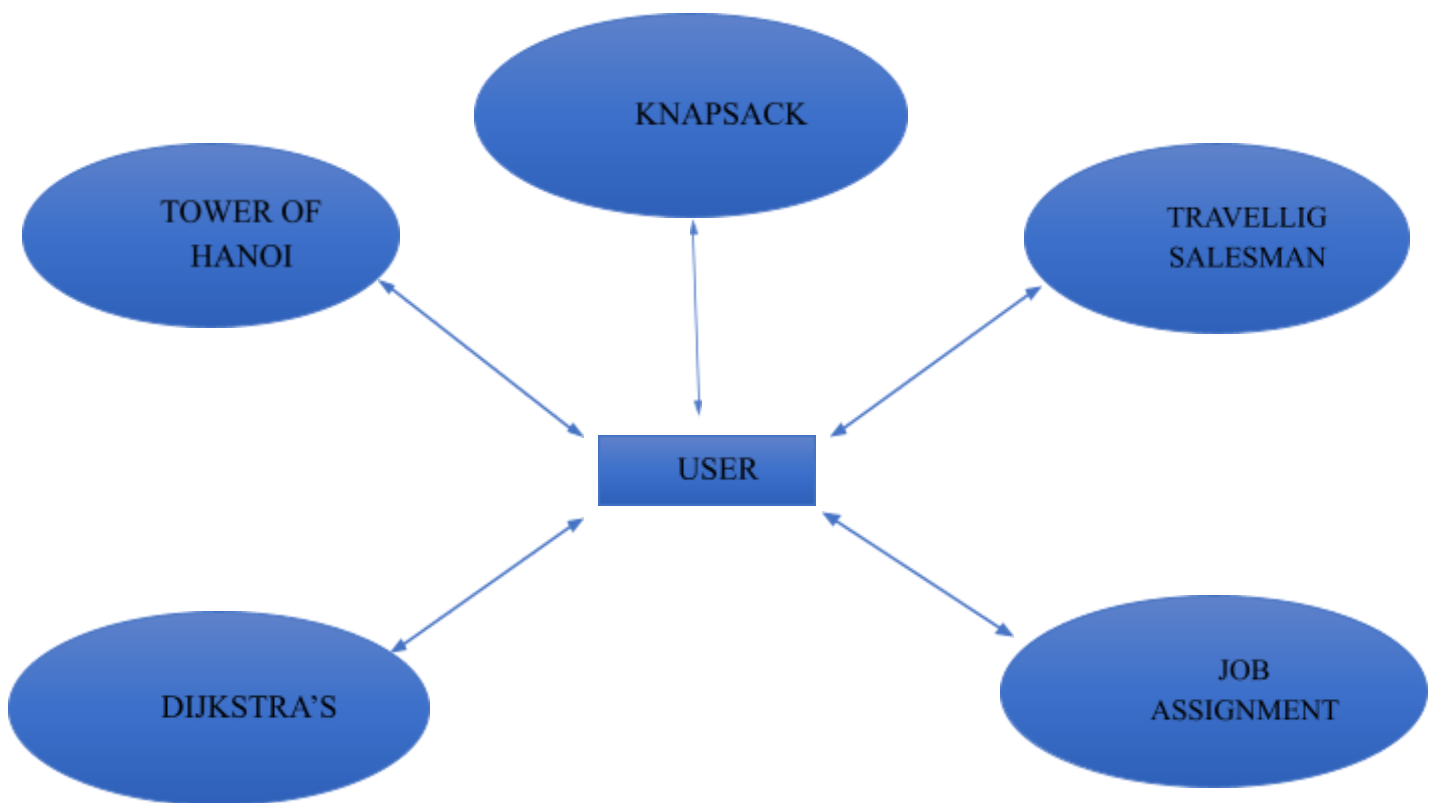
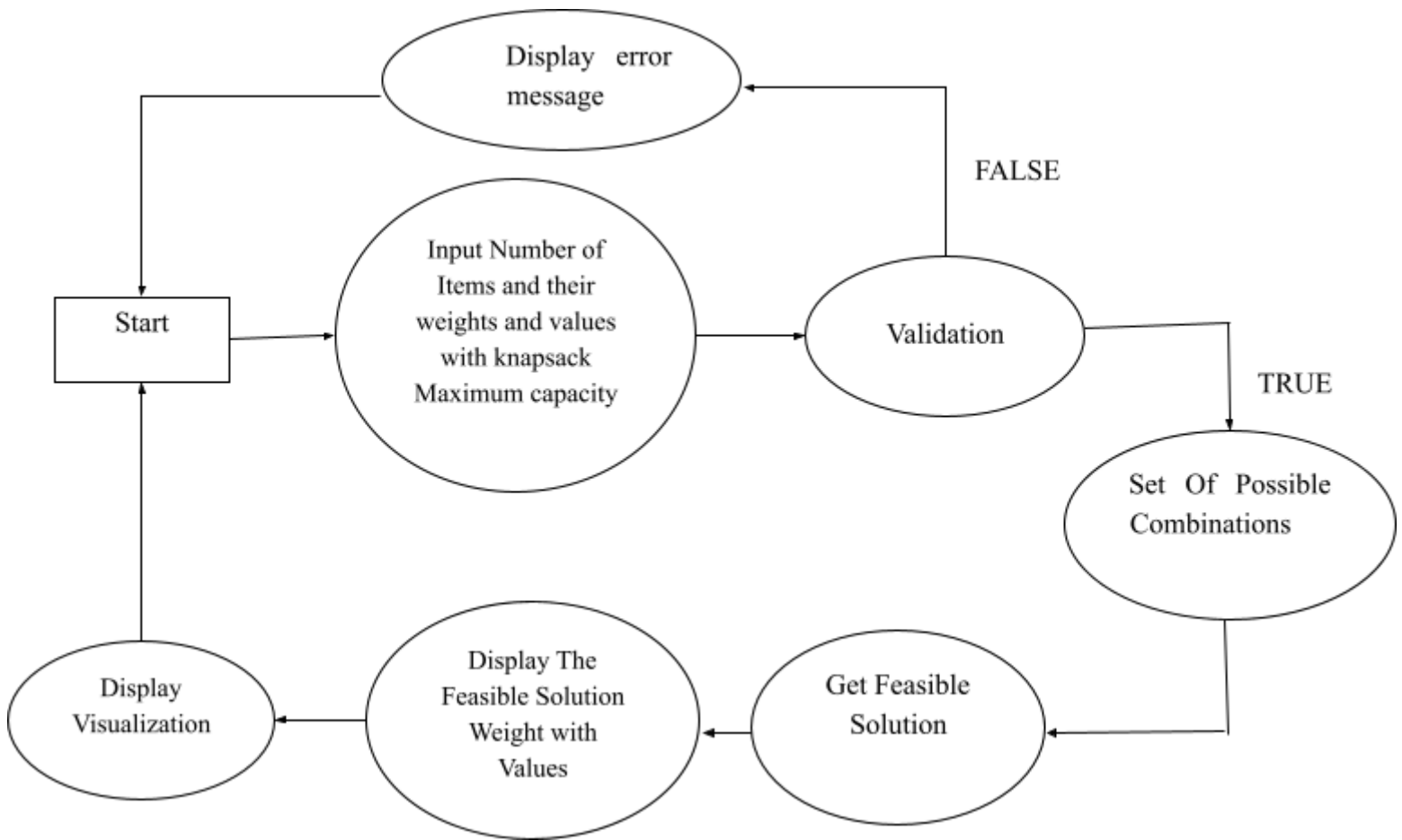
Symbols	Name	Description
	Process	It performs transformation of data from one state to another.
	Source /Sink	It represents the external entity that may be either source or Sink.
	Flow of data	It represents the flow of data from source to destination
	Data Source/Data storage	It is the place where data is stored.

Table 3.1 Data Flow Diagram Symbols.



3.4.1 Knapsack



3.4.1.1 Input

Input Number of Items and their weights and values with knapsack Maximum capacity

3.4.1.2 Process

Finding Set Of Possible Combinations and Get Feasible Solution

3.4.1.3 Output

Display The Feasible Solution Weight with Values

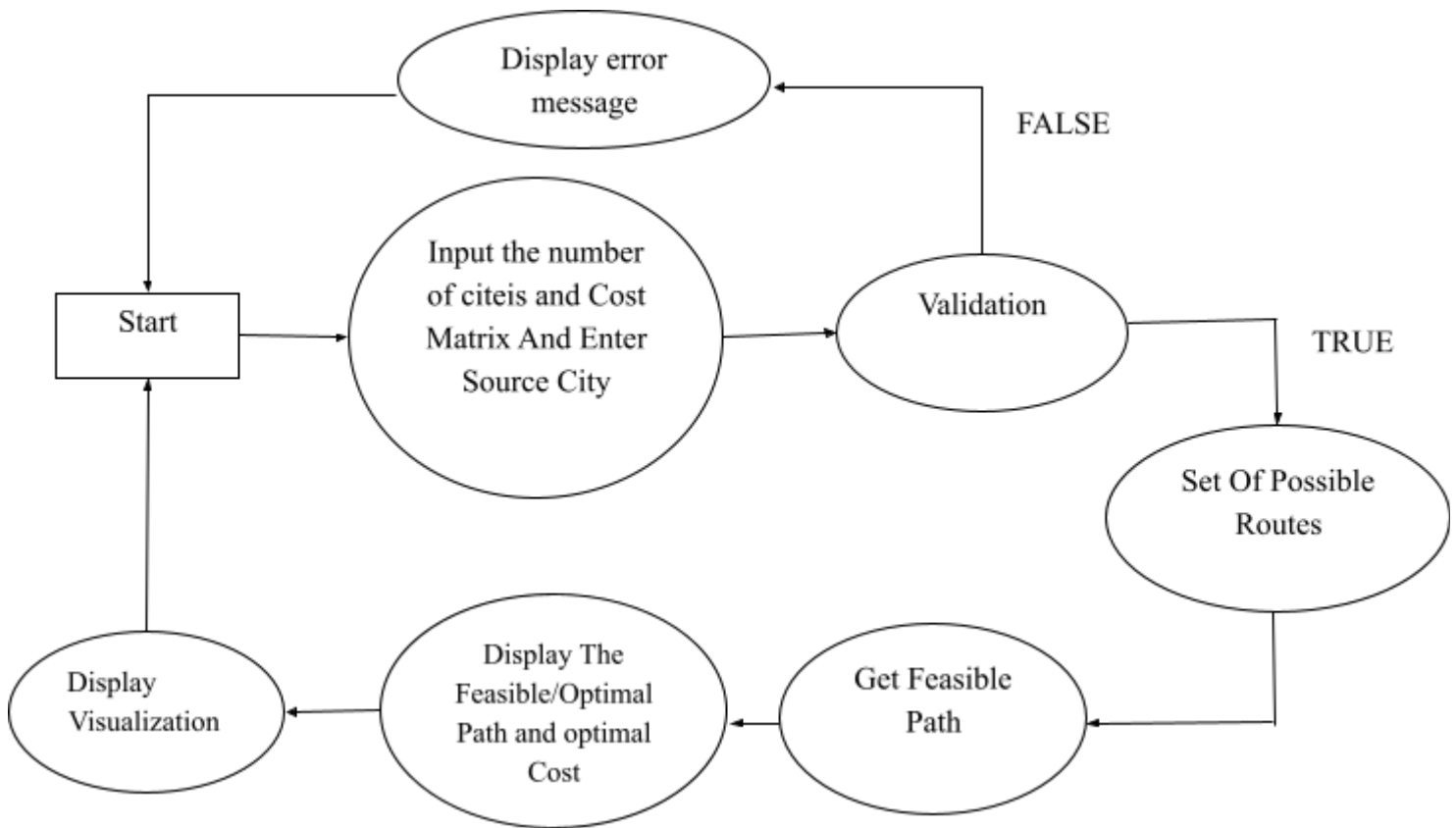
3.4.1.4 Resource allocation

style.css file.

3.4.1.5 User interface

Buttons, labels, textbox, message box, alert box.

3.4.2 Travelling Salesman



3.4.2.1 Input

Input the number of cities and Cost Matrix And Enter Source City

3.4.2.2 Process

Finding Set Of Possible Routes to travel around all cities and come back to source city

3.4.2.3 Output

Display The Feasible/Optimal Path and optimal Cost and Display Visualization

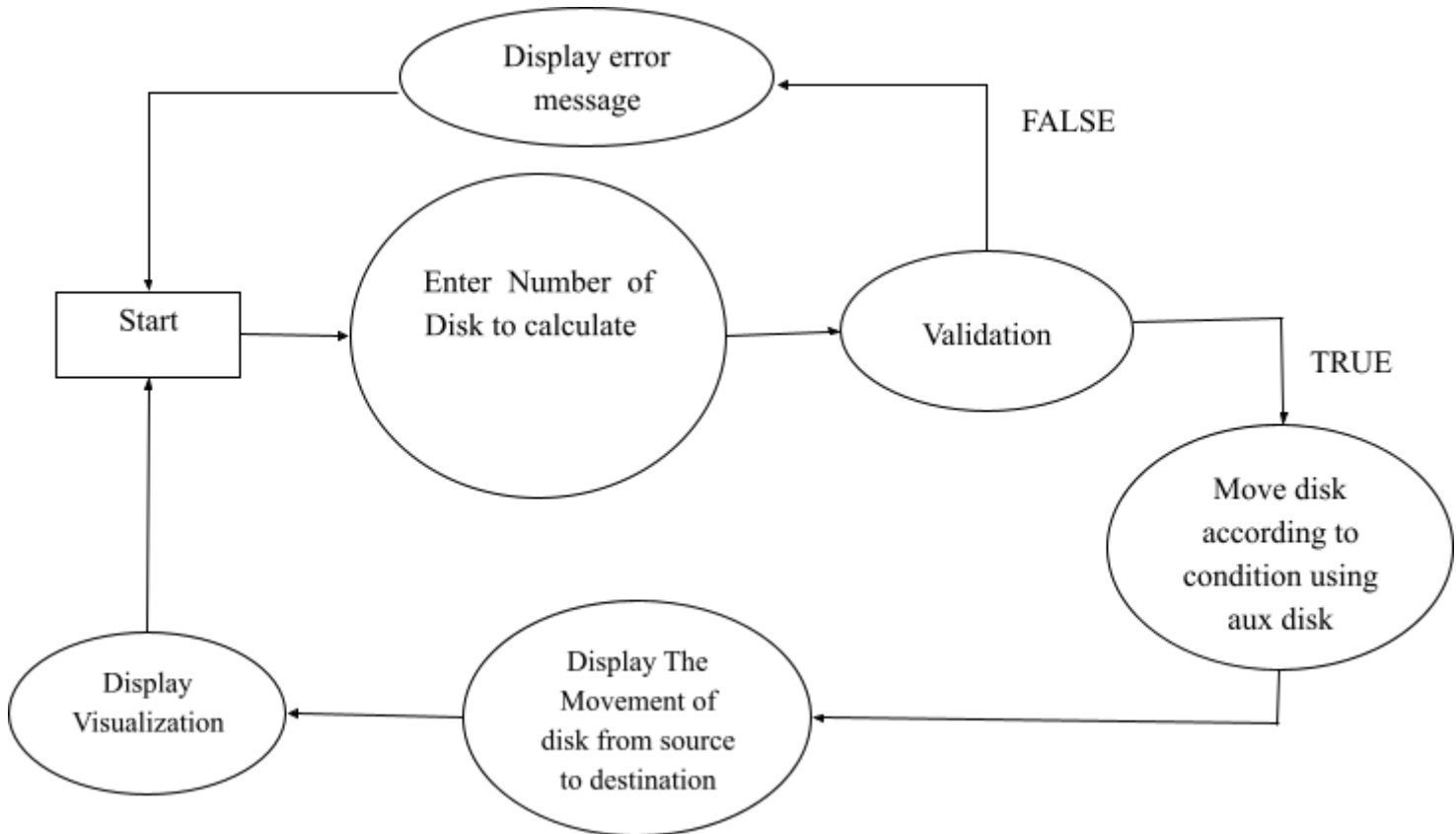
3.4.2.4 Resource allocation

style.css file.

3.4.2.5 User interface

Buttons, labels, textbox, message box, alert box.

3.4.3 Tower Of Hanoi



3.4.3.1 Input

Enter Number of Disk to calculate

3.4.3.2 Process

Move disk according to condition using aux disk with help of recursion to move disks from source to destination

3.4.3.3 Output

Display The Movement of disk from source to destination

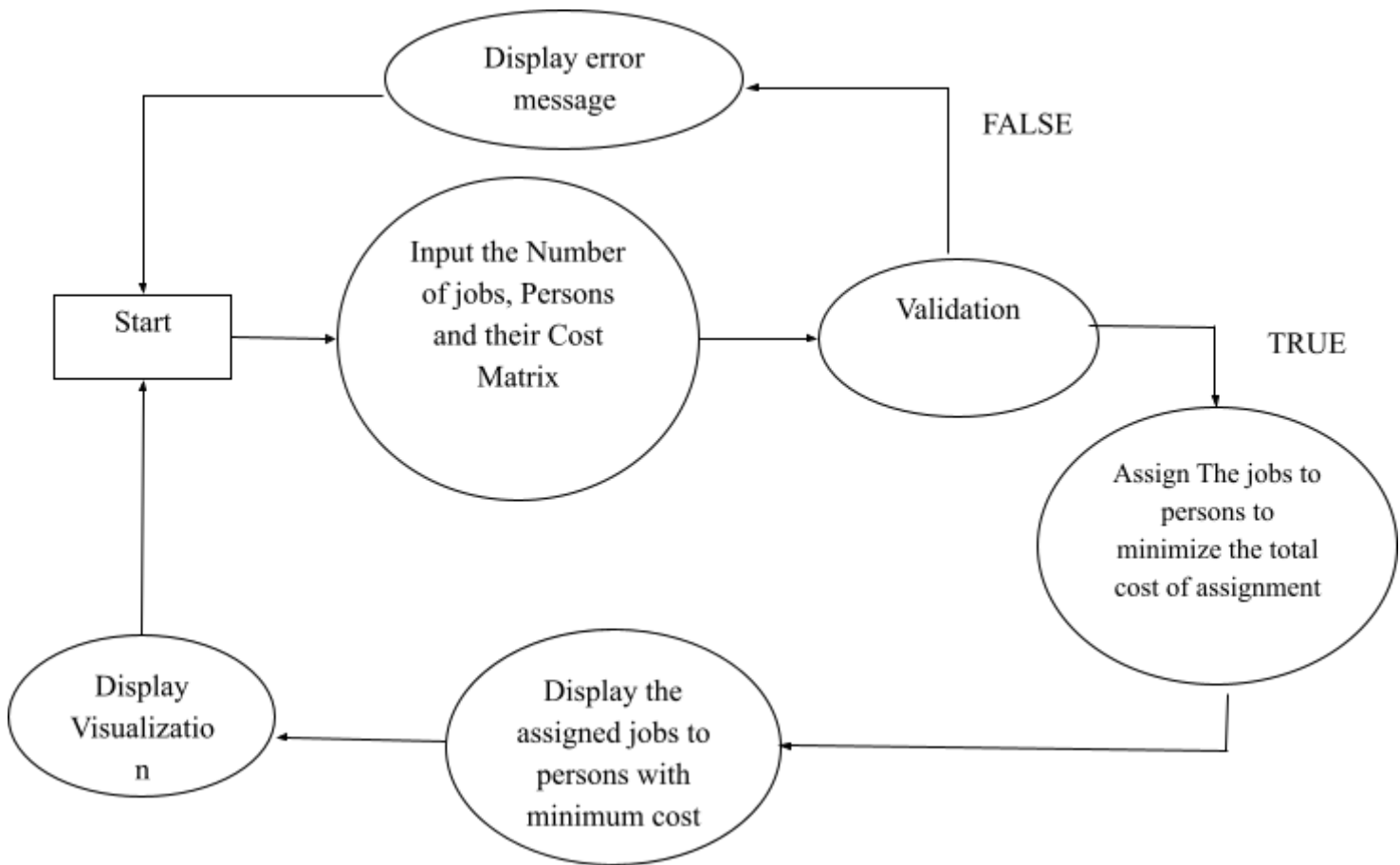
3.4.3.4 Resource allocation

style.css file.

3.4.3.5 User interface

Buttons, labels, textbox, message box, alert box

3.4.4 Job Assignment



3.4.4.1 Input

Input the Number of jobs, Persons and their Cost Matrix

3.4.4.2 Process

Assign The jobs to persons to minimize the total cost of assignment.

3.4.4.3 Output

Display the assigned jobs to persons with minimum cost and display visualization

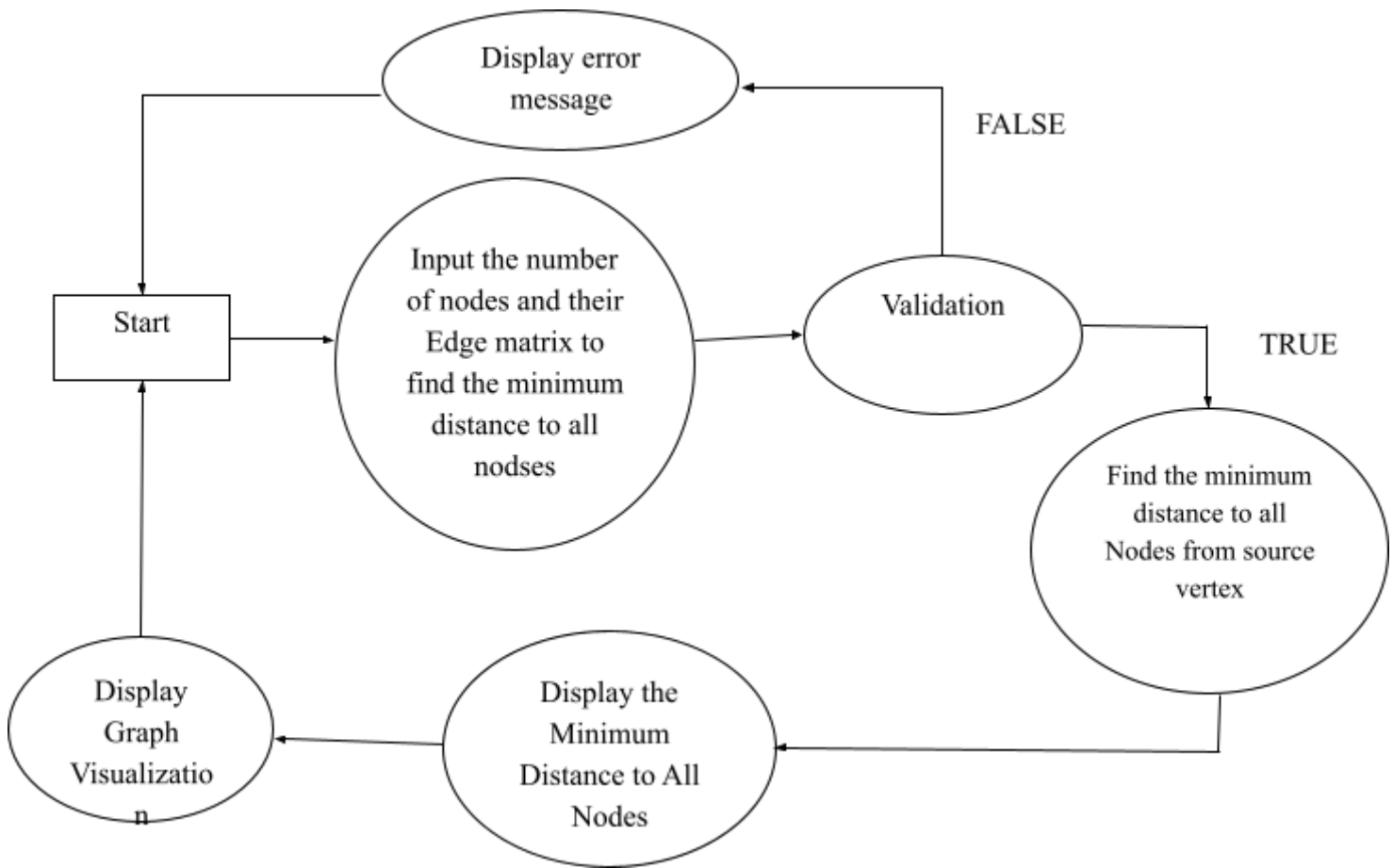
3.4.4.4 Resource allocation

style.css file.

3.4.4.5 User interface

Buttons, labels, textbox, message box, alert box.

3.4.5 Dijkstra's



3.4.5.1 Input

Input the number of nodes
Input the edge matrix

3.4.5.2 Process

Find the minimum distance from source to all nodes

3.4.5.3 Output

Display the Minimum distance to all nodes from source nodes

3.4.5.4 Resource allocation

style.css file.

3.4.5.5 User interface

Buttons, labels, textbox, message box, alert box.

4. DETAILED DESIGN



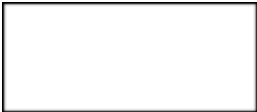

4.1 Introduction

During detailed design, the internal logic of each module specified in system design is decided. During this phase further details of the modules are decided. Design of each of the modules usually specified in a high-level description language which is independent of the language in which software eventually be implemented.

4.2. Module decomposition of software

Structure chart:

Structure chart is a top-down modular design, consist of squares representing different modules in a system and lines. Structure chart shows how program has been partitioned into manageable modules hierarchy and organization of those modules and communicational interface.

Symbol	Name	Process
	Data flow	Show the direction flow of data.
	Control flow	Shows the direction of flow control.
	Processing	Shows manipulation, calculation and processing.
	Module Invocation	It represents subordinate module being invoked by

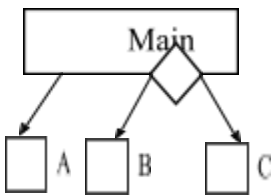
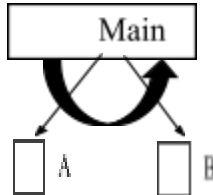


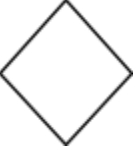








		superior ordinate module.
	Condition invocation	It indicates that the invocation of subordinate modules depends on the evaluation of a condition.
	Iteration	It represents the iteration.

Table 4.1 Structure chart

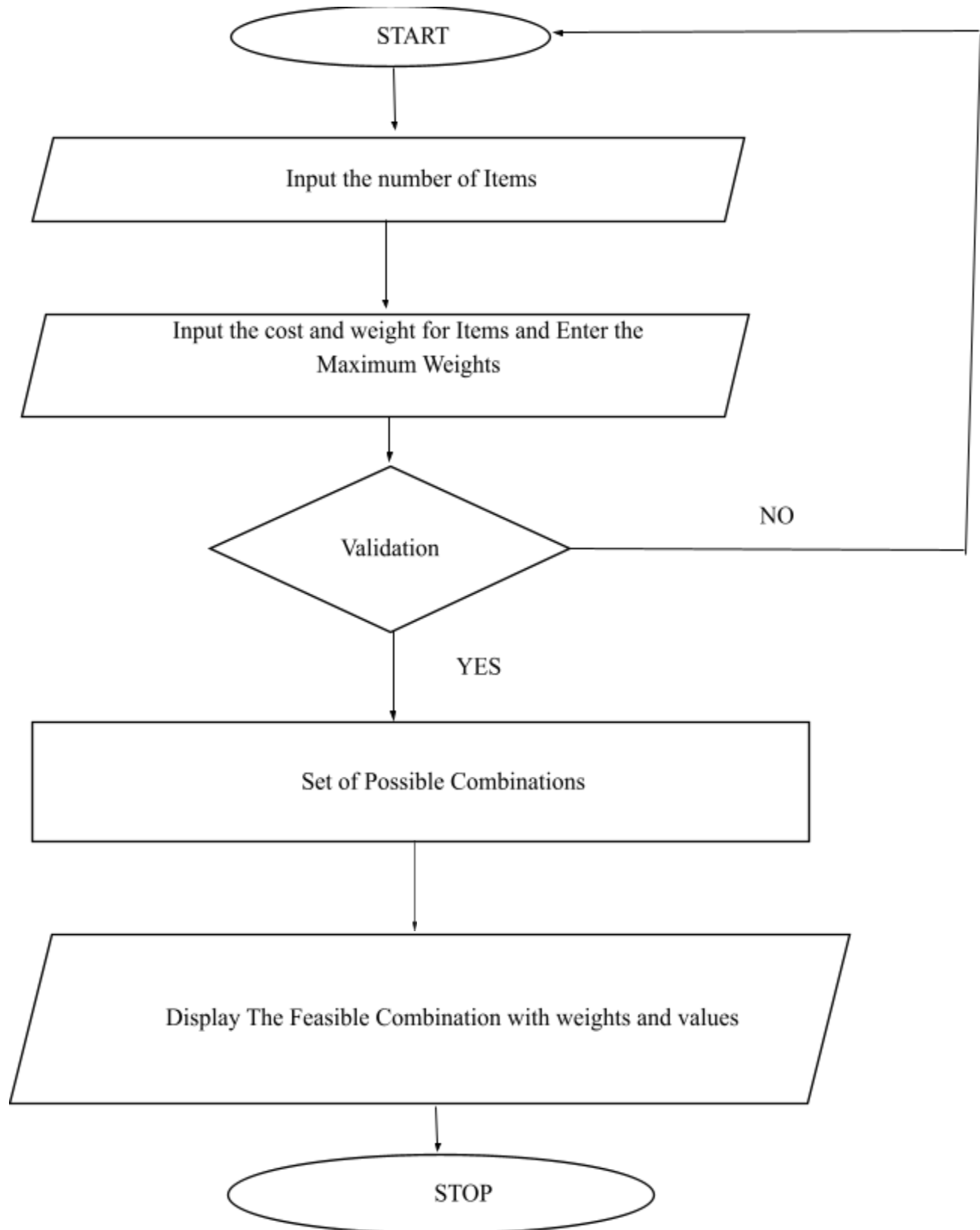
Flow chart:

Symbol	Name	Purpose
	Terminator	It indicates the start and end process.
	Input/output	Input/output data.

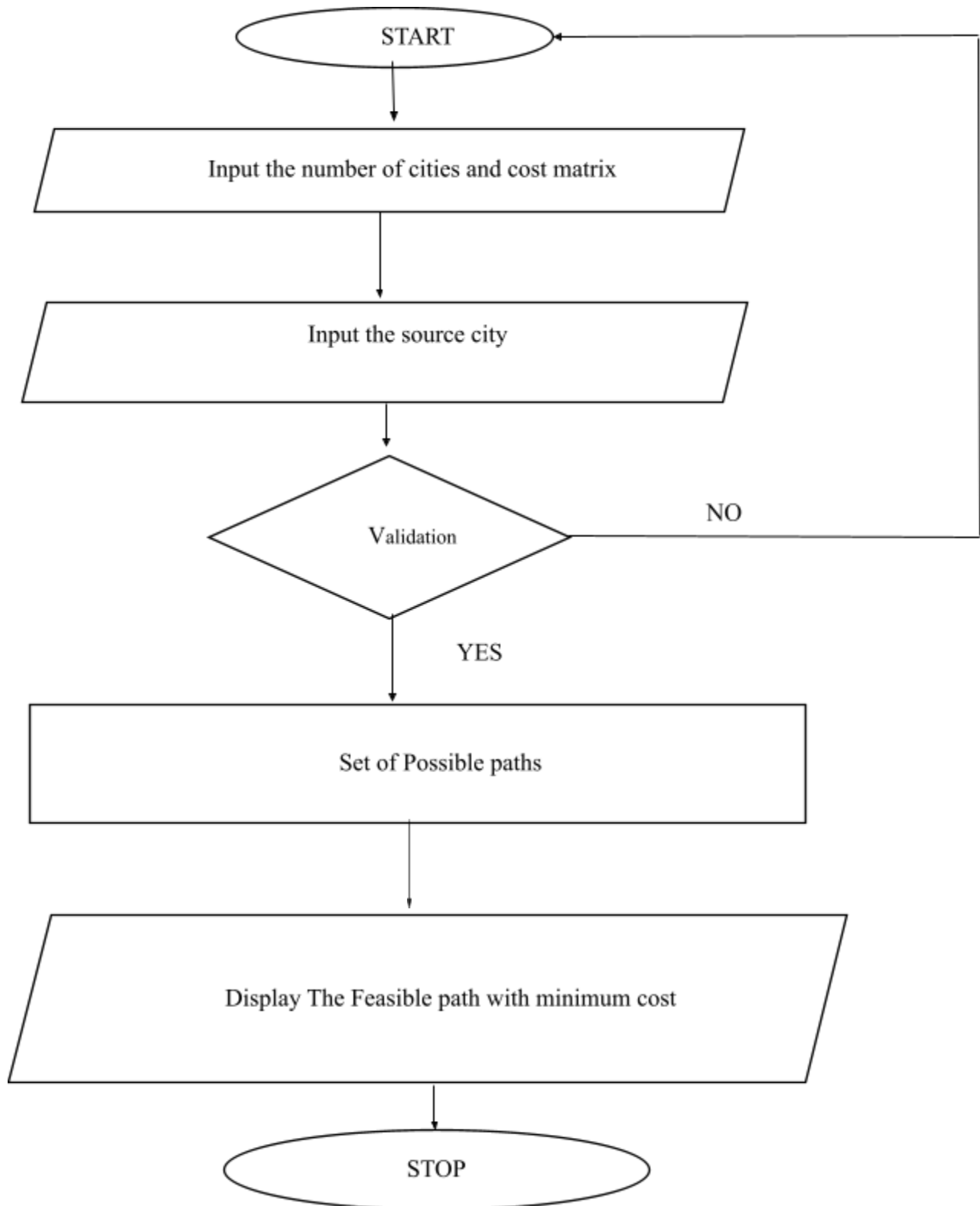
	Decision	It represents a comparison or question that determines an alternate path to be followed.
	Flow direction	Shows the direction of the data flow.
	Processing	It represents manipulation, calculation or information processing
	Direction access storage.	File storage.
	Preparation (Looping)	An instruction or group of instruction.
	In-page	
	Off-page	
	Delay	
	Pre-defined Process	

Flow chart is a graphical representation of solution to the given problems. A flow chart is pictorial representation of an algorithm, workflow or process. The diagrammatic representation illustrates a solution model to given problem. It uses the following symbols.

Knapsack – Flowchart:-



Travelling Salesman:-



Algorithm :

Tower of Hanoi :

Function recursiveHanoi(n,s,a,d)

If n==1 then

 print(s+ "to" +d);

 return;

 end

 recursiveHanoi(n-1,s,d,a);

 print(s+ "to" +d);

 print(s+ "to" +d);

 recursiveHanoi(n-1,a,s,d);

end

Algorithm :

Dijkstra:-

Input: a graph G, a source vertex and destination vertex t

Output: a path from s to t with a maximum load

For each vertex v do

{

Status[v]=0;wt[v]=-1;dad[v]=-1;

}

Status[s]=2;wt[s]= $+\infty$;

For each edge[s,w] do

{

Status[w]=1;wt[w]=weight(s,w);dad[w]=s;

}

While there are fringes do

v=the fringe with max wt-value;

status[v]=2;

for each edge[v,w] do

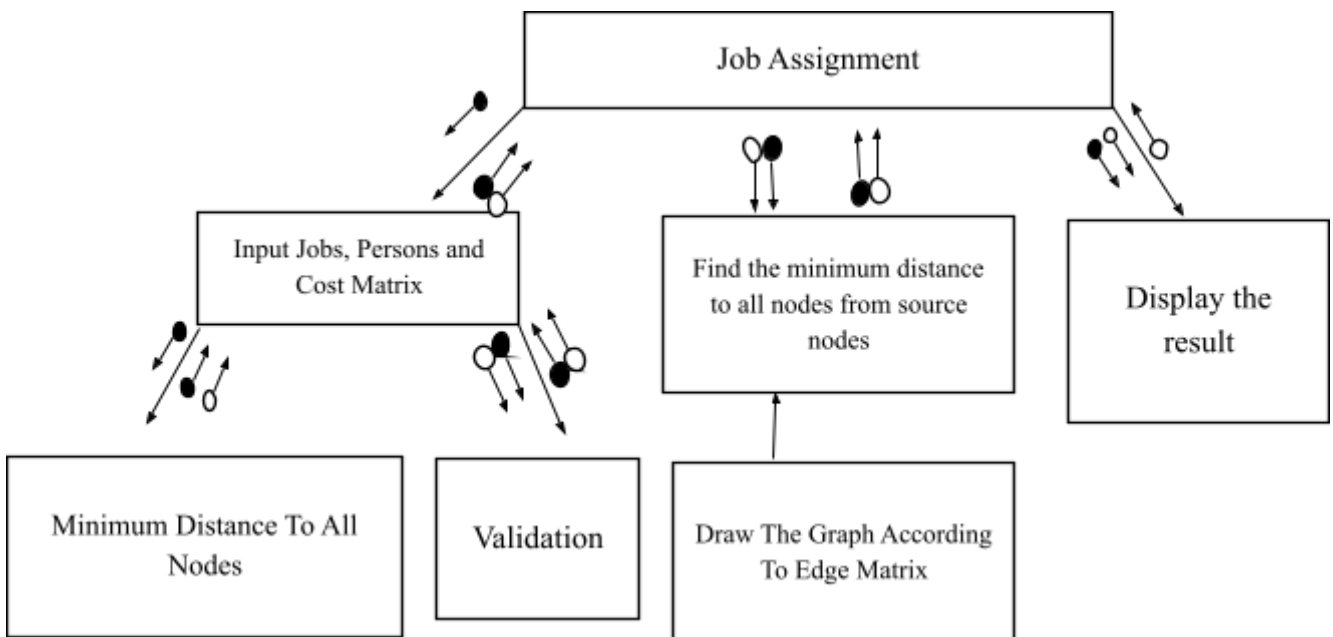
case 1:status[w]==0;

{Status[w]=1;wt[w]=min{wt[v],weight(v,w)};dad[w]=v;}

Case 2: (status[w]==1) and (wt[w]<min{wt[v],weight(v,w)}):

{ wt[w]=min{wt[v], weight(v,w)};dad[w]=v;}

Job Assignment:-



6. TESTING

6.1 Introduction

Testing is the major quality control measures and during the software development it is used to detect errors that could have occurred during any of the phase like requirement analysis, design, coding. The goal of the testing is to uncover errors in the program.

6.2 Levels of Testing

Testing is done in different levels which includes the following.

- Unit Testing
- Integration Testing
- System testing
- Acceptance testing

- **Unit Testing**

In Unit testing each module gets tested during the coding phase itself. The purpose is to exercise the different parts of the module code to detect the coding errors.

- **Integration Testing**

After new testing the modules are gradually integrated into sub systems. It is performed to detect design errors by focusing on testing the interconnection between modules.

- **System Testing**

System is tested against the system requirement if all the requirements are met and if the system performs as specified by the requirement.

- **Acceptance Testing**

It is performed to demonstrate to the client on real life data of the client, the operation of the system.

6.3 Test Case

It is the input that tests the genuineness of the program and successful execution Of the test case reveals. that there are no errors in the program that are under testing. It is a set of conditions or variables under which tester will determine whether an application or software is working currently

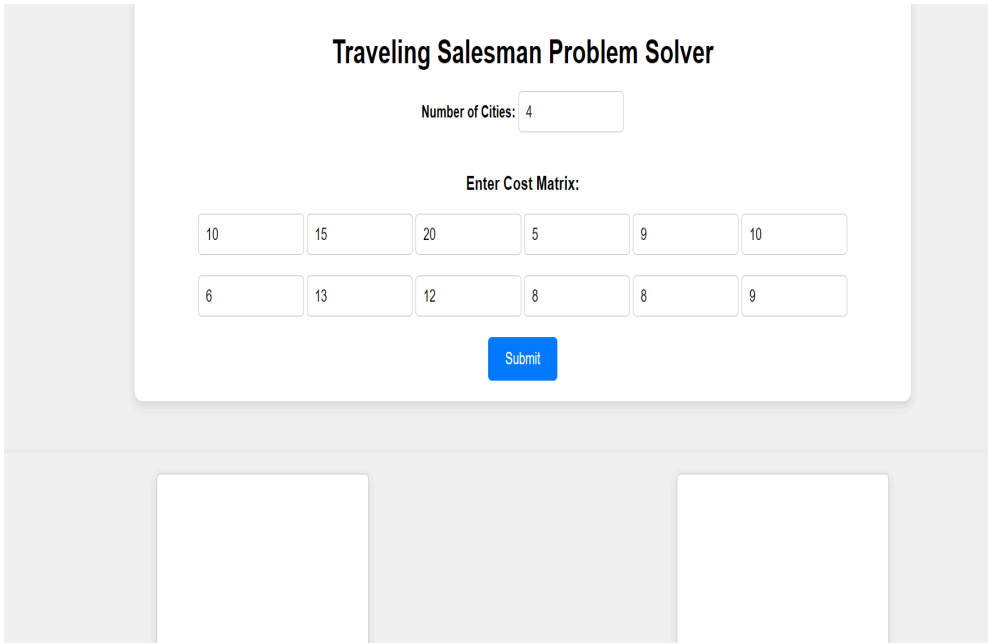
KNAPSACK

Test case ID	01
Title	Knapsack
Purpose	Determine the most valuable combination of items to fit in knapsack.
Test data	Number of items, their weights and values and maximum capacity.
Steps	<ol style="list-style-type: none">1. Enter number of items2. Enter the Items Weight with Value3. Enter Maximum Capacity4. Display the optimal solution.
Expected output	<div>The input:</div> <div><div><div>Knapsack Problem Solver</div><div>Number of Items: <input type="text"/></div><div>Knapsack Capacity: <input type="text"/></div><div>Solve</div></div><div>Visualization</div></div>

The output:



TRAVELLING SALESMAN

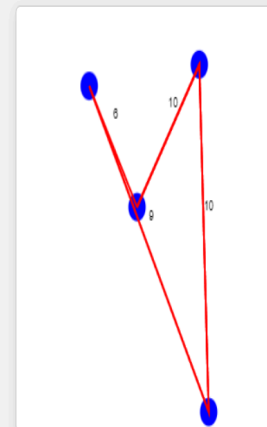
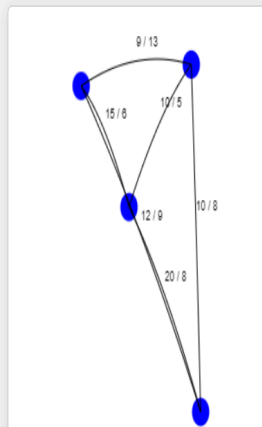
Test case ID	02
Title	Travelling Salesman
Purpose	Finding most efficient route that salesman can visit all cities and return to source city.
Test data	Enter Number of Cities and their cost matrix.
Steps	<ol style="list-style-type: none"> 1. Enter number of cities 2. Consider 0/1 as source city 3. Enter Cost Matrix 4. Display the optimal Paths.
Expected output	<p>The input:</p> 

The output:

Result:

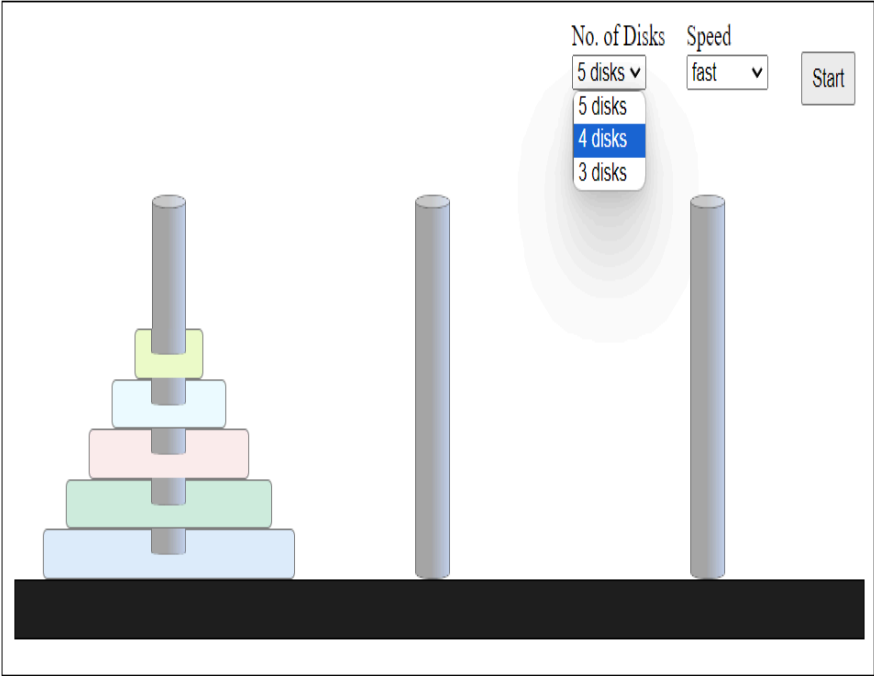

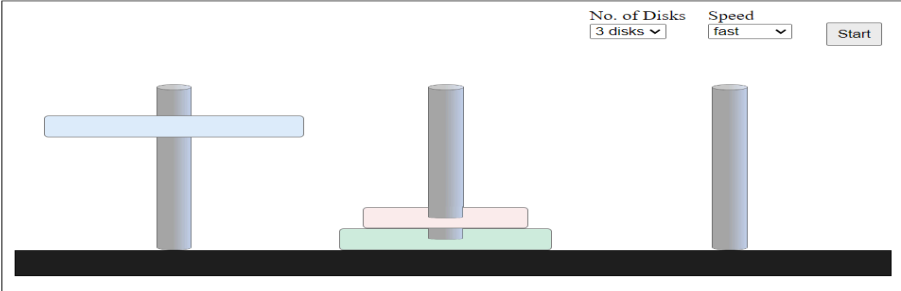

Minimum Cost: 35

Optimal Path: 1 -> 2 -> 4 -> 3 -> 1



TOWER OF HANOI

Test case ID	03
Title	Tower Of Hanoi

Purpose	Move a stack of disks from one rod to another rod with certain rules.
Test data	Enter number of disks.
Steps	<div>1. Enter number of disks.</div> <div>2. Click start button.</div> <div>3. Visualise disk movement.</div> <div>4. Display the steps.</div>
Expected output	<div>The input:</div> <div></div> <div></div> <div>The output:</div> <div></div> <div></div>

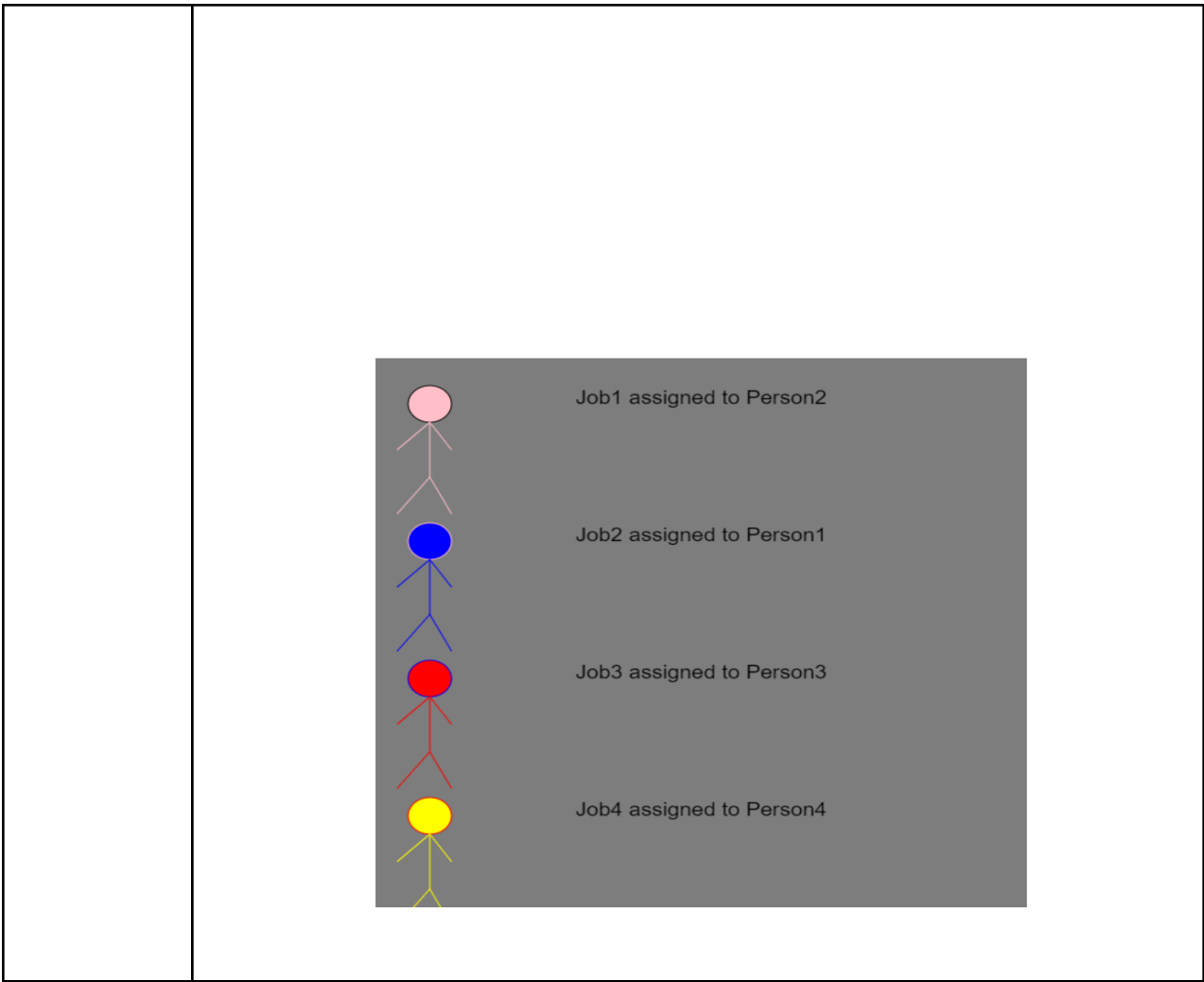
	<div> Steps: Move disk from Bar 1 to Bar 3 Move disk from Bar 1 to Bar 2 Move disk from Bar 3 to Bar 2 Move disk from Bar 1 to Bar 3 Move disk from Bar 1 to Bar 3 Move disk from Bar 1 to Bar 2 Move disk from Bar 3 to Bar 2 Move disk from Bar 1 to Bar 3 Move disk from Bar 2 to Bar 1 Move disk from Bar 2 to Bar 3 Move disk from Bar 1 to Bar 3 </div>
--	---

JOB ASSIGNMENT

Test case ID	04
Title	Job Assignment
Purpose	Efficiently allocate tasks to individuals to maximize productivity.
Test data	Enter Jobs, Persons and their Cost Matrix.

Steps	<div>1. Enter the Jobs.</div> <div>2. Enter the Persons.</div> <div>3. Enter Cost Matrix.</div> <div>4. Display the optimal solution.</div>																												
Expected output	<div>The input:</div> <div><div><div><div>Job Assignment</div><div><div>Number of Jobs: 4</div><div>Number of Persons: 4</div><div>Generate Matrix</div><table><tr><td>9</td><td>2</td><td>7</td><td>8</td></tr><tr><td>6</td><td>4</td><td>3</td><td>7</td></tr><tr><td>5</td><td>8</td><td>1</td><td>8</td></tr><tr><td>7</td><td>6</td><td>9</td><td>4</td></tr></table><div>Find Optimal Assignment</div></div></div></div></div> <div>The output:</div> <div><div><div><div>Job Assignment</div><div><div>Number of Jobs: 4</div><div>Number of Persons: 4</div><div>Generate Matrix</div><table><tr><td>9</td><td>2</td><td>7</td><td>8</td></tr><tr><td>6</td><td>4</td><td>3</td><td>7</td></tr><tr><td>5</td><td>8</td><td>1</td><td>8</td></tr></table></div></div></div></div>	9	2	7	8	6	4	3	7	5	8	1	8	7	6	9	4	9	2	7	8	6	4	3	7	5	8	1	8
9	2	7	8																										
6	4	3	7																										
5	8	1	8																										
7	6	9	4																										
9	2	7	8																										
6	4	3	7																										
5	8	1	8																										

Optimal Assignment:
Job 1 assigned to Person 2 with cost 2



DIJKSTRA’S

Test case ID	05
Title	Dijkstra’s
Purpose	Finding shortest path in graph from source to each node.
Test data	Enter number of nodes and edge matrix.
Steps	1. Enter number of nodes 2. Click on start button. 3. Enter edge matrix. 4. Display the distance to each node from source.

<div>Expected output</div>	<div>The input:</div> <div data-bbox="326 205 1305 1113"><div><h3>Dijkstra's Algorithm</h3><p>Enter the number of nodes:</p><div><div>5</div></div><div>Submit</div><p>Enter the edge matrix:</p><table><tr><td>0</td><td>3</td><td>0</td><td>7</td><td>0</td></tr><tr><td>3</td><td>0</td><td>4</td><td>2</td><td>0</td></tr><tr><td>0</td><td>4</td><td>0</td><td>5</td><td>6</td></tr><tr><td>7</td><td>2</td><td>5</td><td>0</td><td>4</td></tr><tr><td>0</td><td>0</td><td>6</td><td>4</td><td>q</td></tr></table><div>Calculate Shortest Paths</div></div></div> <div>The output:</div> <div data-bbox="342 1425 1326 1927"><div>Calculate Shortest Paths</div><div><h3>Shortest Paths</h3><p>Distance from Source Node To Node 1: Distance = 0 Distance from Source Node To Node 2: Distance = 3 Distance from Source Node To Node 3: Distance = 7 Distance from Source Node To Node 4: Distance = 5 Distance from Source Node To Node 5: Distance = 9</p></div></div>	0	3	0	7	0	3	0	4	2	0	0	4	0	5	6	7	2	5	0	4	0	0	6	4	q
0	3	0	7	0																						
3	0	4	2	0																						
0	4	0	5	6																						
7	2	5	0	4																						
0	0	6	4	q																						

--	--

USER INTERFACE

KNAPSACK

Knapsack Problem Solver

Number of Items:

Knapsack Capacity:

Solve

Visualization

TRAVELLING SALESMAN

Traveling Salesman Problem Solver

Number of Cities:

Enter Cost Matrix:

<input type="text" value="10"/>	<input type="text" value="15"/>	<input type="text" value="20"/>	<input type="text" value="5"/>	<input type="text" value="9"/>	<input type="text" value="10"/>
<input type="text" value="6"/>	<input type="text" value="13"/>	<input type="text" value="12"/>	<input type="text" value="8"/>	<input type="text" value="8"/>	<input type="text" value="9"/>

Result:

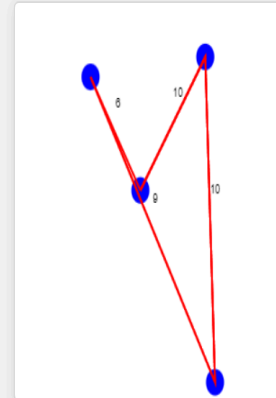
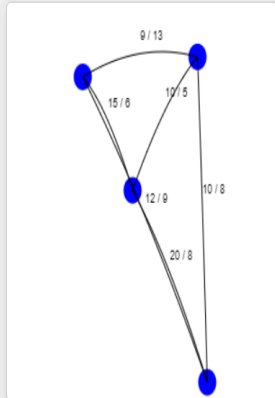
Minimum Cost: 35

Optimal Path: 1 -> 2 -> 4 -> 3 -> 1

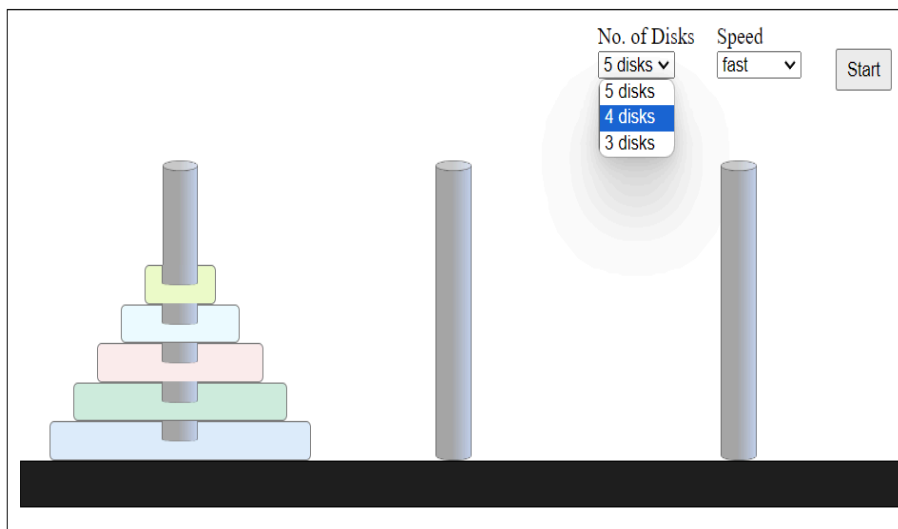
Result:

Minimum Cost: 35

Optimal Path: 1 -> 2 -> 4 -> 3 -> 1



TOWER OF HANOI



Steps:

Move disk from Bar 1 to Bar 3
Move disk from Bar 1 to Bar 2
Move disk from Bar 3 to Bar 2
Move disk from Bar 1 to Bar 3
Move disk from Bar 1 to Bar 3
Move disk from Bar 1 to Bar 2
Move disk from Bar 3 to Bar 2
Move disk from Bar 1 to Bar 3
Move disk from Bar 2 to Bar 1
Move disk from Bar 2 to Bar 3
Move disk from Bar 1 to Bar 3

JOB ASSIGNMENT

Job Assignment

Number of Jobs:

Number of Persons:

Generate Matrix

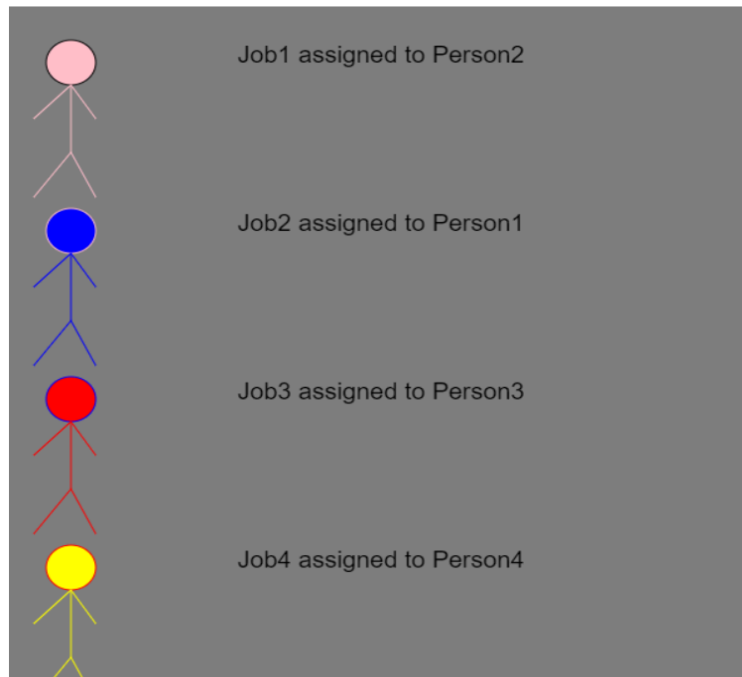
<input type="text" value="9"/>	<input type="text" value="2"/>	<input type="text" value="7"/>	<input type="text" value="8"/>
<input type="text" value="6"/>	<input type="text" value="4"/>	<input type="text" value="3"/>	<input type="text" value="7"/>
<input type="text" value="5"/>	<input type="text" value="8"/>	<input type="text" value="1"/>	<input type="text" value="8"/>
<input type="text" value="7"/>	<input type="text" value="6"/>	<input type="text" value="9"/>	<input type="text" value="4"/>

Find Optimal Assignment

Optimal Assignment:

Job 1 assigned to Person 2 with cost 2
Job 2 assigned to Person 1 with cost 6
Job 3 assigned to Person 3 with cost 1
Job 4 assigned to Person 4 with cost 4

Total Cost: 13



DIJKASTRA'S

Dijkstra's Algorithm

Enter the number of nodes:

Enter the edge matrix:

0	3	0	7	0
3	0	4	2	0
0	4	0	5	6
7	2	5	0	4
0	0	6	4	0

Calculate Shortest Paths

Shortest Paths

Distance from Source Node To Node 1: Distance = 0
Distance from Source Node To Node 2: Distance = 3
Distance from Source Node To Node 3: Distance = 7
Distance from Source Node To Node 4: Distance = 5
Distance from Source Node To Node 5: Distance = 9

■

