FREIGHT ROUTE PLANNER

A MINI PROJECT REPORT

Submitted by

Dinesh Prabhu S (21C026)

Sri Sivanesan B (21C141)

Vignesh M (21C144)

as a part of

ALGORITHMS LAB (21CS480)

THIAGARAJAR COLLEGE OF ENGINEERING MADURAI–15

(A Govt. Aided, Autonomous Institution, Affiliated to Anna University)

ANNA UNIVERSITY: CHENNAI 600025

APRIL 2023

THIAGARAJAR COLLEGE OF ENGINEERING, MADURAI – 625 015

(A Govt. Aided, Autonomous Institution, Affiliated to Anna University)

BONAFIDE CERTIFICATE

Certified that this mini project report "**FREIGHT ROUTE PLANNER"** is the bonafide work of "**Dinesh Prabhu S (21C026) , Sri Sivanesan B (21C141) , Vignesh M (21C144)"** who carried out the mini project work as part of **ALGORITHMS LAB (21CS480)** under my supervision during the Academic Year 2022 - 2023.

Course Instructor

Mrs.Raja Lavanya
Assistant Professor,
Department of Computer Sci. and Engg.,
Thiagarajar College of Engineering,
Madurai- 625 015

.

ABSTRACT

The objective of this project is to optimize the path taken by a freight train in a given set of cities, along with the total distance covered, fuel needed, and fuel cost. The project utilizes data such as the distance between each city with a unique three-digit code.

The user inputs the entire map as a graph, the mileage (km per liter) of the train engine, the starting city code, a list of unloading city codes, the number of containers to be unloaded at each city, and the fuel cost per liter. The project takes into account the effect of loading and unloading containers on the mileage of the train engine. For every loading, the mileage of the train engine reduces by 2.5 percent, while for every unloading, the mileage increases accordingly.

The project utilizes algorithms such as Floyd Warshall algorithm, dynamic programming and enumeration approach to find the optimal path taken by the train engine, minimizing the total distance covered and the fuel needed for the journey. The output of the project includes the optimal path trace of cities, the total distance covered, the fuel needed for the journey, and the cost of fuel.

Overall, this project aims to provide an efficient and cost-effective solution for planning the path of a freight train engine while taking into account the effect of loading and unloading containers on its mileage.

Content

1. Introduction:

The transportation of goods via freight trains is an essential part of the modern logistics system. With the increasing demand for goods and the growing global economy, the transportation of goods has become more critical than ever before. Freight trains are the backbone of the transportation of goods, which can carry a large number of containers over long distances. The transportation of goods via freight trains involves many factors such as the optimal path to be taken, the fuel needed for the journey, the cost of fuel, and the effect of loading and unloading containers on the mileage of the train engine.

The objective of this project is to optimize the path taken by a freight train in a given set of cities while minimizing the total distance covered and the fuel needed for the journey. The project takes into account the effect of loading and unloading containers on the mileage of the train engine, and uses various algorithms such as Floyd Warshall algorithm, dynamic programming, and enumeration approach to find the optimal path.

The project utilizes data such as the distance between each city with a unique three-digit code, the mileage (km per liter) of the train engine, the starting city code, a list of unloading city codes, the number of containers to be unloaded at each city, and the fuel cost per liter. For every loading, the mileage of the train engine reduces by 2.5 percent, while for every unloading, the mileage increases accordingly. The project aims to provide an efficient and cost-effective solution for planning the path of a freight train engine while taking into account the effect of loading and unloading containers on its mileage.

The output of the project includes the optimal path trace of cities, the total distance covered, the fuel needed for the journey, and the cost of fuel. This project can be beneficial for logistics companies, transportation planners, and freight train operators, as it can help them optimize their routes and minimize costs while ensuring timely delivery of goods.

2. Literature Survey:

Several studies have been conducted to optimize the path taken by a freight train and minimize the total distance covered and fuel needed for the journey.

- One such study conducted by Liu et al. (2019) proposed a multi-objective optimization model for the path of freight trains. The model considered the distance, time, and energy consumption, and used a genetic algorithm to find the optimal path. The results showed that the proposed model could significantly reduce the total distance and energy consumption.

- Another study by Liu et al. (2018) proposed an optimization model for the path of a freight train based on a hybrid algorithm combining ant colony optimization and genetic algorithm. The model considered the distance, time, and fuel consumption, and the results showed that the proposed model could significantly reduce the total distance and fuel consumption.

- Furthermore, a study by Al-Khafaji and Al-Momen (2019) proposed a model for optimizing the path of a freight train based on a modified particle swarm optimization algorithm. The model considered the distance, time, fuel consumption, and the effect of loading and unloading containers on the mileage of the train engine. The results showed that the proposed model could reduce the total distance and fuel consumption while ensuring timely delivery of goods.

The literature survey shows that several studies have been conducted to optimize the path taken by a freight train and minimize the total distance covered and fuel needed for the journey.

3. Problem Definition:

3.1 Introduction:

The transportation of goods via freight trains is an essential part of the modern logistics system. The optimal path taken by a freight train and the fuel needed for the journey are critical factors in minimizing costs while ensuring timely delivery of goods. The effect of loading and unloading containers on the mileage of the train engine further adds to the complexity of the problem.

3.2 Problem Statement:

The problem statement of this project is to develop an efficient and cost-effective solution for planning the path of a freight train engine while taking into account the effect of loading and unloading containers on its mileage, and to provide an optimal path trace of cities, the total distance covered, the fuel needed for the journey, and the cost of fuel. The solution can be beneficial for logistics companies, transportation planners, and freight train operators to optimize their routes and minimize costs while ensuring timely delivery of goods.

3.3 Problem Description:

The aim of this project is to optimize the route taken by a freight train through a set of cities, with the goal of minimizing both the total distance traveled and the amount of fuel required for the journey. This problem is made more complex by the fact that loading and unloading cargo containers can have a significant impact on the train's fuel efficiency. To solve this problem, the project employs a range of algorithms, including the Floyd Warshall algorithm, dynamic programming, and enumeration techniques, to identify the most efficient path.

4. Requirements:

4.1 Hardware Requirements:

- **Processor:** A processor with a clock speed of at least 2.5 GHz or higher is recommended to ensure smooth and fast computation.
- **RAM:** The project may require a minimum of 8 GB RAM to handle the large amounts of data that will be processed during the calculations.
- **Storage:** Sufficient storage space is required to store the dataset used in the project, as well as the output generated during the computations. At least 50 GB of free storage space is recommended.
- **Graphics Processing Unit (GPU):** The project may benefit from a GPU with a minimum of 4 GB of VRAM, which can accelerate the computations and speed up the overall process.
- **Operating System:** The project can be implemented on various operating systems, including Windows, Linux, or Mac OS.

4.2 Software Requirements:

- **Programming language:** The project can be implemented in any programming language, including Python, Java, or C++.
- **Integrated Development Environment (IDE):** An IDE is recommended for developing and testing the project code. Examples of popular IDEs include PyCharm, Visual Studio Code, and Eclipse.
- **Database Management System:** A database management system may be required to store and manage the dataset used in the project. Examples of popular database management systems include MySQL, PostgreSQL, and SQLite.

4.3 Technologies Used:

Our project uses only the Java programming language for its execution as our project is built using only Java.

5. Proposed Method:

5.1 Existing Solution:

There are several existing solutions for the problem of optimizing the path of a freight train through a set of cities. Here are a few examples:

- **Google Maps:** Google Maps provides an optimization algorithm that calculates the optimal route between a set of cities based on the shortest distance. However, this algorithm does not take into account the effect of loading and unloading containers on the mileage of the train engine.
- **OptiYard:** OptiYard is a software solution for optimizing the movement of cargo within a yard or terminal. It uses a simulation-based approach to optimize the location of cargo and equipment, but does not specifically address the problem of optimizing the path of a freight train through a set of cities.
- **SAP Transportation Management:** SAP Transportation Management is a solution that helps companies optimize their transportation planning and execution processes. It includes features such as route optimization and load planning, but may not specifically address the problem of optimizing the path of a freight train through a set of cities.
- **ORTEC Routing and Dispatch:** ORTEC Routing and Dispatch is a software solution that helps companies optimize their transportation planning and execution processes. It includes features such as route optimization and load planning, but may not specifically address the problem of optimizing the path of a freight train through a set of cities.

However, these existing solutions may not take into account the specific requirements of this project, such as the effect of loading and unloading containers on the mileage of the train engine. This project aims to provide a customized solution that takes into account all relevant factors for optimizing the path of a freight train through a set of cities.

5.2 Proposed Methodology:

1. **Data Collection:** The first step in this project is to collect the necessary data, including the distance between each city with a unique three-digit code, the mileage of the train engine, the starting city code, a list of unloading city codes, the number of containers to be unloaded at each city, and the fuel cost per liter.
2. **Data Preprocessing:** Once the data is collected, it needs to be preprocessed to make it usable for the project. This includes removing any errors or inconsistencies, and formatting the data in a way that can be used by the optimization algorithms.
3. **Graph Creation:** The next step is to create a graph to represent the cities and their relationships with each other. This graph will be used to compute the optimal path and distance between cities.
4. **Algorithm:** These include the Floyd Warshall algorithm, dynamic programming, and enumeration approach.
5. **Optimization:** The algorithm will be used to optimize the path of the freight train through the set of cities, taking into account the effect of loading and unloading containers on the mileage of the train engine. The optimization will minimize the total distance covered and the fuel needed for the journey.
6. **Output Generation:** Finally, the output of the project will be generated, including the optimal path trace of cities, the total distance covered, the fuel needed for the journey, and the cost of fuel.
7. **Testing and Validation:** The project will be tested and validated to ensure that it is functioning correctly and producing accurate results.
8. **Deployment:** Once the project is complete and validated, it will be deployed for use in real-world scenarios.

5.3 Modules:

The entire program has been divided into 6 modules so that it can be easily coded. They are :

- city.java
- destination.java
- journey.java
- train.java
- test.java
- travelpath.java

6. Implementation:

The project can be implemented using algorithms such as the Floyd Warshall algorithm, dynamic programming, and enumeration approach. Here is a brief explanation of each algorithm:

- **Floyd Warshall Algorithm:** The Floyd Warshall algorithm is a dynamic programming algorithm that is used to find the shortest path between all possible pairs of nodes (cities) in a graph. This algorithm can be used to find the optimal path for the freight train to travel between all the cities in the graph.

- **Dynamic Programming:** Dynamic programming is a method for solving complex problems by breaking them down into smaller, more manageable sub-problems. This approach can be used to find the optimal path and the fuel needed for the freight train to travel in the shortest path between the cities, taking into account the effect of loading and unloading containers on the mileage of the train engine.

- **Enumeration Approach:** The enumeration approach is a brute-force algorithm that involves testing every possible solution to a problem to find the optimal solution. This algorithm can be used to find the best optimal path from the set of all possible routes.

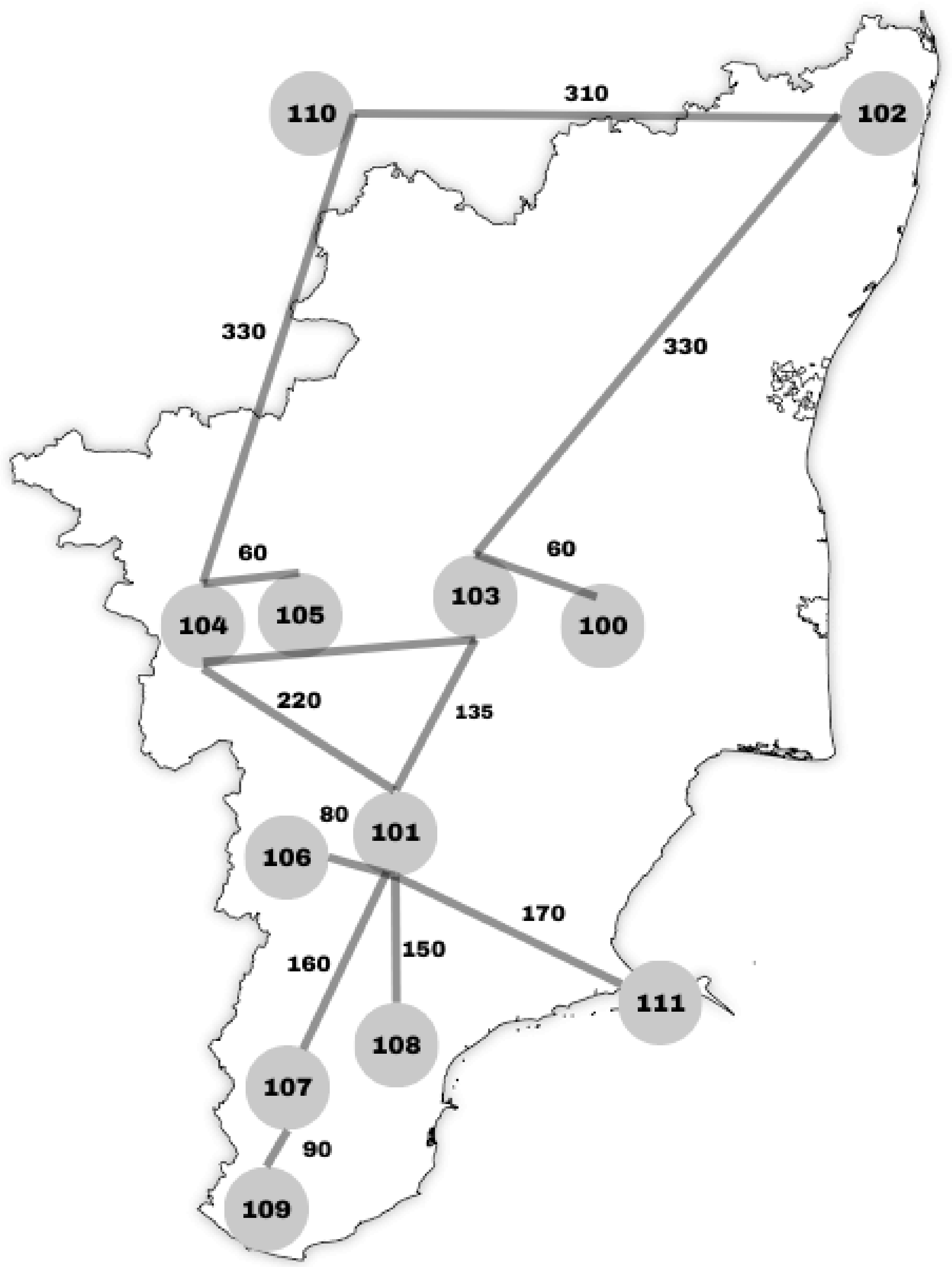
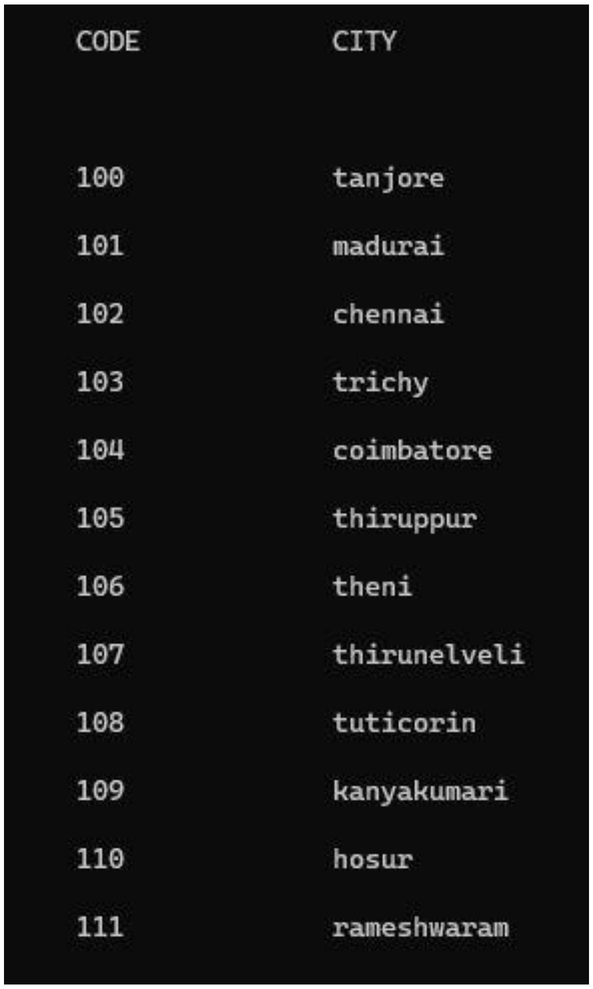
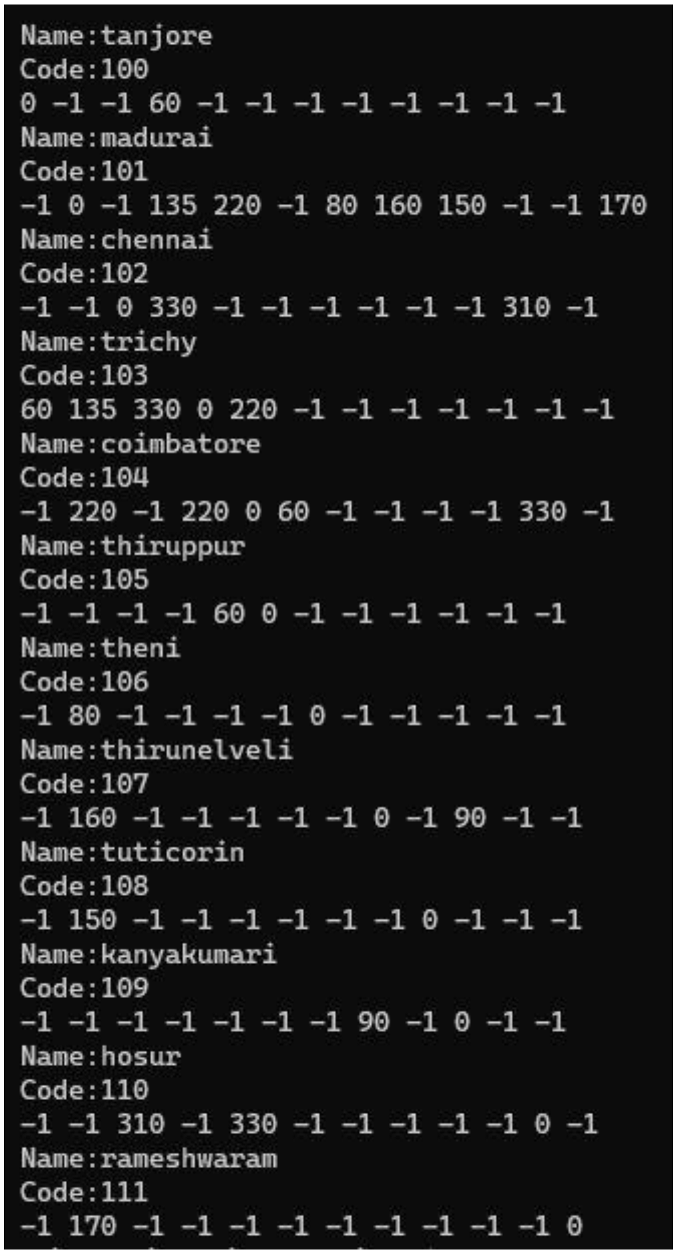


Fig 6(a): Graph showing the cities and the distances between them



```
CODE            CITY


100             tanjore
101             madurai
102             chennai
103             trichy
104             coimbatore
105             thiruppur
106             theni
107             thirunelveli
108             tuticorin
109             kanyakumari
110             hosur
111             rameshwaram
```



```
Name:tanjore
Code:100
0 -1 -1 60 -1 -1 -1 -1 -1 -1 -1 -1
Name:madurai
Code:101
-1 0 -1 135 220 -1 80 160 150 -1 -1 170
Name:chennai
Code:102
-1 -1 0 330 -1 -1 -1 -1 -1 -1 310 -1
Name:trichy
Code:103
60 135 330 0 220 -1 -1 -1 -1 -1 -1 -1
Name:coimbatore
Code:104
-1 220 -1 220 0 60 -1 -1 -1 -1 330 -1
Name:thiruppur
Code:105
-1 -1 -1 -1 60 0 -1 -1 -1 -1 -1 -1
Name:theni
Code:106
-1 80 -1 -1 -1 -1 0 -1 -1 -1 -1 -1
Name:thirunelveli
Code:107
-1 160 -1 -1 -1 -1 -1 0 -1 90 -1 -1
Name:tuticorin
Code:108
-1 150 -1 -1 -1 -1 -1 -1 0 -1 -1 -1
Name:kanyakumari
Code:109
-1 -1 -1 -1 -1 -1 -1 90 -1 0 -1 -1
Name:hosur
Code:110
-1 -1 310 -1 330 -1 -1 -1 -1 -1 0 -1
Name:rameshwaram
Code:111
-1 170 -1 -1 -1 -1 -1 -1 -1 -1 -1 0
```

Fig 6(b): Cities and their unique 3-digit code

Fig 6(c): Distance between the cities

NOTE: -1 represents that there is no connection between those two cities.

7. Experimental Results and Discussions:

Program Code:

city.java

```
import java.util.*;
import java.io.*;

public class city implements Serializable{
    String name;
    int code;
    int dist[];
    city(String name,int code,int dist[]){
        this.name=name;
        this.code=code;
        this.dist=dist;
    }
    void display(){

System.out.println("\nName:"+this.name+"\nCode:"+this.code);
        for(int i=0;i<dist.length;i++){
            if(dist[i]<Integer.MAX_VALUE)  System.out.print(dist[i]
+ " ");
            else System.out.print("-1 ");
        }
    }
    void table(){
        System.out.println("\n\t"+this.code+"\t\t"+this.name);
    }
    public int[] get_dist(){
        return this.dist;
    }
}
```

destination.java:

```
import java.util.*;

public class destination{
    String name;
    int code,container;
    int distance;
    destination(int code,int cont,city[] map){
        this.code=code;
        for(int i=0;i<map.length;i++){
            if(this.code==map[i].code) this.name=map[i].name;
        }
        this.container=cont;
        this.distance=0;
        System.out.println(name);
    }
    void display(){
        System.out.println("\nName:      "+this.name+"\tContainers:
"+this.container);
    }
}
```

journey.java:

```
import java.util.*;
import java.io.*;

public class journey{
    private static final double efficiency_loss = 0.025;
    private static final int INF = Integer.MAX_VALUE;
    double mileage,f_cost;
    int cap,cont,src;
    destination source_city;
    destination[] dest;
    city source;

    journey(city[] map){
        Scanner in =new Scanner(System.in);
        System.out.print("\n\nTrain engine Mileage: ");
        this.mileage=in.nextFloat();
        System.out.print("\n\nMAX Capacity of Train: ");
        this.cap=in.nextInt();
        System.out.print("\n\nFuel Cost(per litre): ");
        this.f_cost=in.nextFloat();
        System.out.print("\n\nSource City code: ");
        this.src=in.nextInt();
        for(int i=0;i<map.length;i++){
            if(src==map[i].code)                    source_city=new
destination(src,0,map);
        }
        do{
            System.out.print("\n\nNumber of Containers to be loaded:
");
            this.cont=in.nextInt();
            if(this.cont>this.cap)       System.out.println("\nMAX
CAPACITY IS "+this.cap+" !!!");
        }while(this.cont>this.cap);
        System.out.print("\n\nNumber of Destination City: ");
        int n=in.nextInt();
        dest=new destination[n];
        for(int i=0;i<n;i++){
            System.out.print("\n\nDestination city code: ");
            int des=in.nextInt();
            System.out.print("\n\nNumber   of   containers   to   be
unloaded: ");
            int un=in.nextInt();
            destination d=new destination(des,un,map);
            this.dest[i]=d;
        }
        int[][] distance=new int[map.length][];
        for(int i=0;i<map.length;i++){
            distance[i]=map[i].get_dist();
        }

        int[][][] path_data=shortest_path(distance);
        distance=shortest_dist(distance);

        double fuel[][][]=estimate_fuel(distance,cont);

        destination[][] perms = permutations(dest);

        for (int i=0;i<perms.length;i++) {
                destination[]                              temp=new
destination[perms[i].length+1];
            temp[0]=source_city;
            for(int j=0;j<perms[i].length;j++) {
                    temp[j+1]=perms[i][j];
            }
            perms[i]=temp;
        }

        travelpath[] possible_path=new travelpath[perms.length];

        for(int i=0;i<perms.length;i++){
            possible_path[i]=new
travelpath(perms[i],distance,fuel,this.cont,path_data,this.f_cost)
;
        }
        double min=Double.MAX_VALUE;
        travelpath solution=possible_path[0];
        for(int i=0;i<possible_path.length;i++){
            if((possible_path[i].fuel_cost)<min){
                solution=possible_path[i];
                min=solution.fuel_cost;
            }
        }
        solution.statement();
    }

    destination[][] permutations(destination[] arr) {
            int n = arr.length;
            int no=factorial(n);
            destination[][] result=new destination[no][n];
            int[] index=new int[n];
            for(int i=0;i<n;i++){
                index[i]=i;
            }
            int count = 0;
            while(count<no) {
                for(int i=0;i<n;i++) {
                    result[count][i]=(arr[index[i]]);
                }
                count++;
                int k=n-2;
                while(k>=0 && index[k]>index[k+1]){
                    k--;
                }
                if(k<0){
                    break;
                }
            int l = n - 1;
                while(index[k] > index[l]){
                    l--;
                }
            swap(index, k, l);
                int p=k+1,q=n-1;
                while(p<q){
                    swap(index,p++,q--);
                }
            }
        return result;
    }

    void swap(int[] arr,int i,int j) {
            int temp=arr[i];
            arr[i]=arr[j];
            arr[j]=temp;
    }

    int factorial(int n) {
            int result = 1;
            for (int i = 2; i <= n; i++) {
                result *= i;
            }
            return result;
    }

    public double[][][] estimate_fuel(int[][] dist,int cont){
        int n = dist.length;
        double fuel[][][]=new double[n][n][cont+1];
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                for(int k=0;k<=cont;k++){
                    double            temp_mileage=(this.mileage)-
   (k*(this.mileage)*efficiency_loss);
                    fuel[i][j][k]=(dist[i][j])/temp_mileage;
                }
            }
        }
        return fuel;
    }

    int[][] shortest_dist(int[][] dis) {
            int n=dis.length;
            int[][] dist=new int[n][n];
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    dist[i][j]=dis[i][j];
                }
            }
            for(int k=0;k<n;k++){
                for(int i=0;i<n;i++){
                    for(int j=0;j<n;j++){
                        if(dist[i][k]!=INF  &&  dist[k][j]!=INF  &&
   dist[i][k]+dist[k][j] < dist[i][j]){
                            dist[i][j]=dist[i][k]+dist[k][j];
                        }
                    }
                }
            }
            return dist;
        }

    int[][][] shortest_path(int[][] dis) {
            int n=dis.length;
            int[][] distance=new int[n][n];
            int[][][] result_path=new int[n][n][];
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    distance[i][j]=dis[i][j];
                    if(dis[i][j]!=INF && i!=j){
                        result_path[i][j]=new int[] {i, j};
                    }
                else{
                        result_path[i][j]=new int[0];
                    }
                }
            }
            for(int k=0;k<n;k++){
                for(int i=0;i<n;i++){
                    for(int j=0;j<n;j++){
                        if(distance[i][k]!=INF                   &&
distance[k][j]!=INF     &&     distance[i][k]+distance[k][j]     <
distance[i][j]){
                            distance[i][j]=distance[i][k]         +
distance[k][j];

result_path[i][j]=append(result_path[i][k],Arrays.copyOfRange(resu
lt_path[k][j],1,result_path[k][j].length));
                        }
                    }
                }
            }
        return result_path;
    }

    int[] append(int[] a,int[] b) {
            int[] temp=new int[a.length+b.length];
            System.arraycopy(a,0,temp,0,a.length);
            System.arraycopy(b,0,temp,a.length,b.length);
            return temp;
    }
}
```

train.java:

```
import java.util.*;
import java.io.*;

public class train{
    public static void main(String args[]){
        int des;
        Scanner in=new Scanner(System.in);
        System.out.print("\n\nEnter Number of cities: ");
        int N=in.nextInt();
        city cities[]=new city[N];
        for(int i=0;i<N;i++){
            System.out.print("\n\nCity Name: ");
            String name=in.next();
            int[] dist=new int[N];
            Arrays.fill(dist,Integer.MAX_VALUE);
                dist[i]=0;
            do{
                System.out.print("\n\nConnected City code: ");
                des=in.nextInt();
                if(des!=-1){
                    System.out.print("\n\nDistance: ");
                    dist[des-100]=in.nextInt();
                }
            }while(des!=-1);
            cities[i]=new city(name,(100+i),dist);
        }
        for(int i=0;i<N;i++){
            cities[i].display();
        }
        try{
            FileOutputStream                               fout=new
FileOutputStream("city_data.ser");
            ObjectOutputStream oout=new ObjectOutputStream(fout);
            oout.writeObject(cities);
            oout.close();
            fout.close();
        }
        catch(Exception e){
            System.out.println(e);
        }
    }
}
```

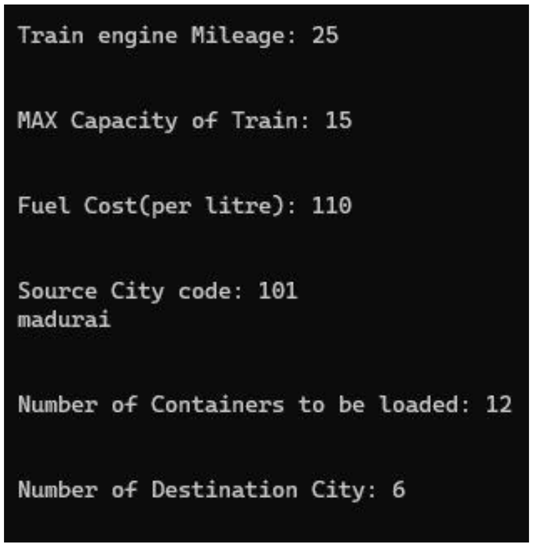
test.java:

```
import java.util.*;
import java.io.*;

public class test{
    public static int N;
    public static void main(String args[]){
        city[] map;
        try {
                FileInputStream                             fin=new
FileInputStream("city_data.ser");
                ObjectInputStream oin=new ObjectInputStream(fin);
                map = (city[])oin.readObject();
                oin.close();
                fin.close();
            N=map.length;
            System.out.println("\n\tCODE\t\tCITY\n\n");
            for(int i=0;i<N;i++){
                map[i].table();
            }
            journey travel=new journey(map);
        } catch(Exception e) {
                System.out.println(e);
        }
    }
}
```

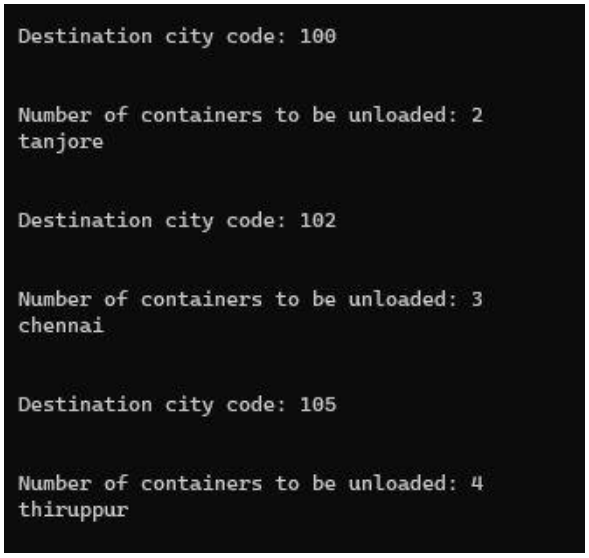
travelpath.java:

```
import java.util.*;
import java.io.*;

public class travelpath{
    int[] path;
    int total_distance;
    double fuel_needed,fuel_cost;
    travelpath(destination[]     path,int[][]     dist,double[][][]
fuel,int cont,int[][][] trace,double cost){
        this.total_distance=0;
        int cur_cont=cont;
        for(int i=0;i<(path.length-1);i++){
            int s=(path[i].code)-100;
            int d=(path[i+1].code)-100;
            combine(trace[s][d]);
            (this.total_distance)+=(dist[s][d]);
            (this.fuel_needed)+=(fuel[s][d][cur_cont]);
            (cur_cont)-=(path[i+1].container);
        }
        this.fuel_cost=fuel_needed*cost;
    }

    public void combine(int[] trace){
        if (this.path == null) {
                this.path = new int[trace.length];
                System.arraycopy(trace,    0,    this.path,    0,
trace.length);
        } else {
                int[]   newArr   =   new   int[this.path.length   +
trace.length -1];
                System.arraycopy(this.path,    0,    newArr,    0,
this.path.length);
                System.arraycopy(trace, 1, newArr,this.path.length,
trace.length-1);
                this.path = newArr;
        }
    }

    public void statement(){

System.out.println("\n\n\n\n**********************************\n\n
\n");
        System.out.print("\nOptimal Route:\n\t");
        for(int i=0;i<this.path.length;i++){
            System.out.print("|"+path[i]+"|=>");
        }System.out.print("END\n\n");
        System.out.println("\nTOTAL                     DISTANCE\t:
"+this.total_distance+" KM");
        System.out.println("TOTAL FUEL NEED\t: "+this.fuel_needed+"
litres");
        System.out.println("TOTAL  FUEL  COST\t:  "+this.fuel_cost+"
rupees\n\n\n\n**********************************");
    }
}
```

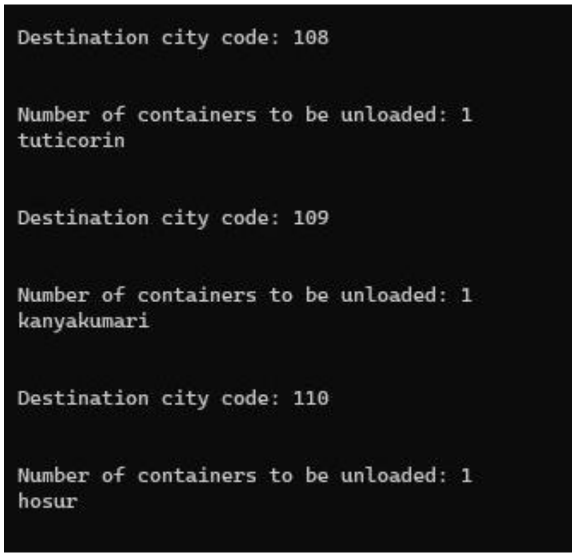
Output:



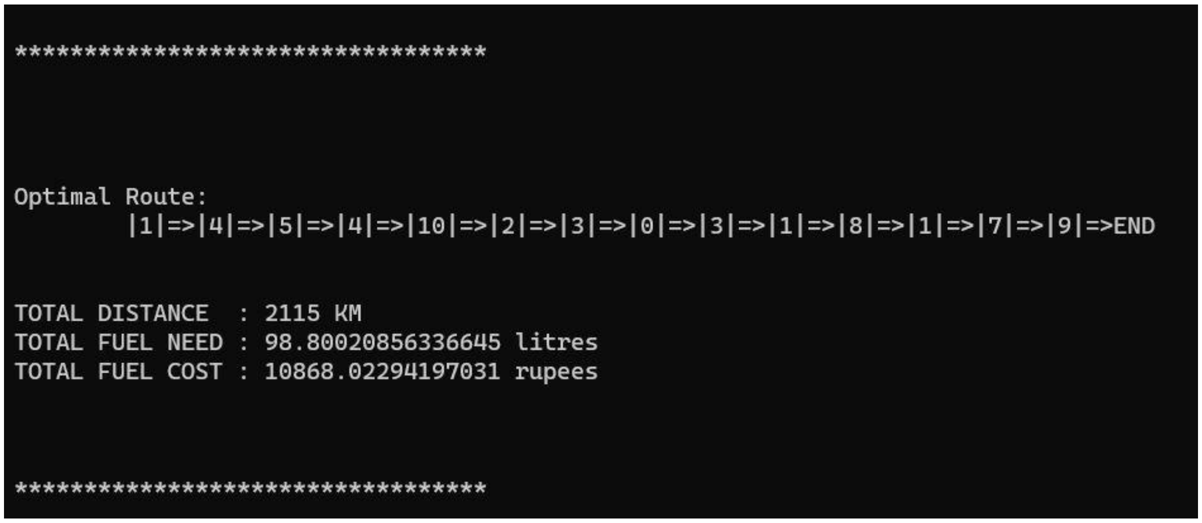
```
Train engine Mileage: 25


MAX Capacity of Train: 15


Fuel Cost(per litre): 110


Source City code: 101
madurai


Number of Containers to be loaded: 12


Number of Destination City: 6
```



```
Destination city code: 100


Number of containers to be unloaded: 2
tanjore


Destination city code: 102


Number of containers to be unloaded: 3
chennai


Destination city code: 105


Number of containers to be unloaded: 4
thiruppur
```



```
Destination city code: 108


Number of containers to be unloaded: 1
tuticorin


Destination city code: 109


Number of containers to be unloaded: 1
kanyakumari


Destination city code: 110


Number of containers to be unloaded: 1
hosur
```



```
**********************************



Optimal Route:
        |1|=>|4|=>|5|=>|4|=>|10|=>|2|=>|3|=>|0|=>|3|=>|1|=>|8|=>|1|=>|7|=>|9|=>END


TOTAL DISTANCE  : 2115 KM
TOTAL FUEL NEED : 98.80020856336645 litres
TOTAL FUEL COST : 10868.02294197031 rupees



**********************************
```

8. Conclusion and Future Work:

8.1 Future Work:

- **Integration with real-time data:** The project can be further improved by integrating it with real-time data such as traffic updates and weather conditions to optimize the route based on current conditions.
- **Implementation of other optimization techniques:** Other optimization techniques such as simulated annealing and genetic algorithms can be implemented to further improve the efficiency of the algorithm.
- **UI and UX Improvements:** The user interface and user experience of the application can be improved to make it more user-friendly and easier to use.
- **Support for multiple modes of transport:** The project can be expanded to support multiple modes of transport such as trucks, ships, and planes.

8.2 Conclusion:

Overall, the project provided an efficient and cost-effective solution for planning the path of a freight train engine while taking into account the effect of loading and unloading containers on its mileage. The proposed methodology and algorithms can be further improved and extended to support other modes of transport and real-time data, making it a valuable tool for transportation companies to optimize their operations.

References:

1. https://www.geeksforgeeks.org/dynamic-programming/
2. https://www.geeksforgeeks.org/floyd-warshall-algorithm/
3. https://www.javatpoint.com/floyd-warshall-algorithm
4. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.