FETCHING PATH WITH MINIMUM COST

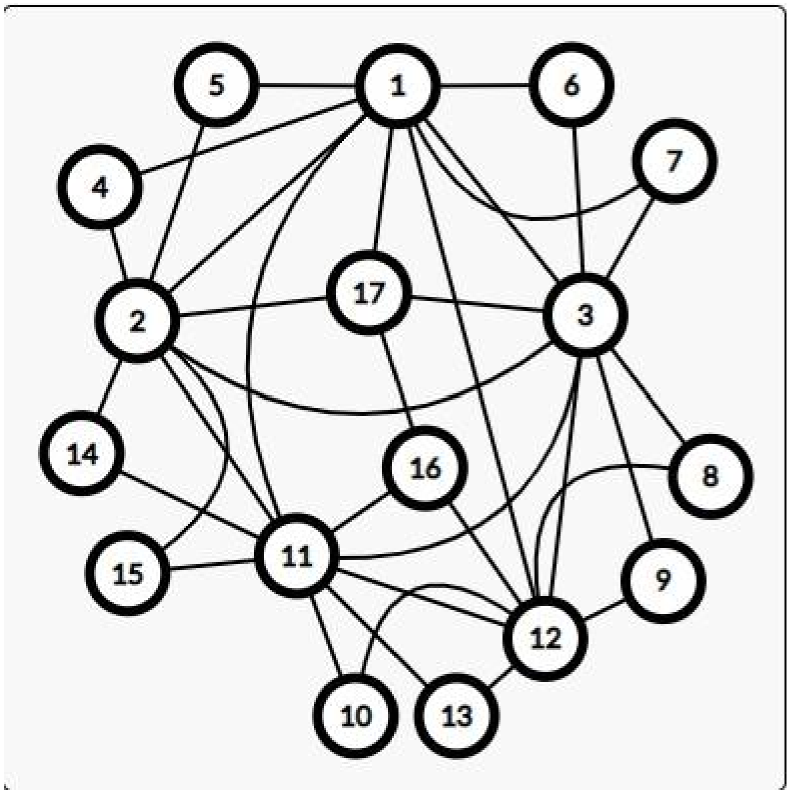
Team Members:-

SAURAV KUKADIYA - CE056
YASH KAMANI - CE049

Subject :-	Data Structures and Algorithms
Semester :-	3
Batch :-	C3
Date :-	19th October , 2019

1. Problem Description :-

There are 17 different cities. This problem takes source and destination as input in form of indexes and fetch shortest route between source and destination for passengers' required seats. We are giving the unique city name to the each of the node numbers of graph as follows :



1) Bengaluru
2) Chennai
3) Hydrabad
4) Kochi
5) Madurai
6) Pondicherry
7) Mysore
8) Coimbatore
9) Ooty
10) Tirumala
11) Wazhi
12) Vizag
13) Hampi
14) Port-Blair
15) Vellore
16) Kovalam
17) Erode

- **Application :**

This program fetches shortest routes between source and destination . It also allocates number of seats given by users . As there is limited seats between two cities , it also takes care that is there that number of seats are available or not . If seats are available then it will be allocates to that user and then program asks user for their details . After that program retrieve the details and store it in one file as backup.If required seat will not be available then it will print appropriate message.

2. Solution :-

- First of all the program will start execution by creating object of Operation class and uses this object to initialize node[] to make adjecency list. After that it will create edges by calling Create_adg method. This method takes two nodes and insert that into each others adjecency list sorted by weight of the edge from that adjecency list.

- Then after control goes to Dijkstra methods .It performs dijkstra algorithm as given below and finds minimum cost of all nodes from source , it also set reverse pointers that leads all nodes to source through the path that has minimum cost.

- At the end execution of Dijkstra it calls ShortestPath which create a path between source and destination using recursive call and it also store that path in array list in terms of indexes.

3. Design and Algorithm :

Class Diagram :

class Adjecents
Node_No , Available_Seats , Weight : int
Adj_Ptr : Adjecents

class Node_Table_Directory
Node_No , dest_frm_src : int
City_Name : String
List_Ptr : Adjecents
Reverse : Node_Table_Directory

class Operation
node[] : Node_Table_Directory
path : ArrayList<Integer>
sum_of_Distance : int
Intialize() : void
Create_Adg(int , int , int) : void
Dijkstra(int , int , int) : int
ShortestPath(int , int) : void

class Contains_main
public static void main(String args[])

Object Diagram :

Adjecents
Adj_Ptr , List_Ptr , adj1 , adj2 , temp , temp1 , pre_temp , pre_temp1 : Adjecents

Node_Table_Directory
node[] : Node_Table_Directory

Operation
obj : Operation

Implementation :

- **Initialize() :** It initialize array of Node_Table_Directory objects .
- **Create_Adg(int , int , int)** : It takes two node number indexes for which you want to create edge and it insert that node number into each others adjecency list sorted by weight.
- **Dijkstra(int , int , int)** : It traverse from source to all nodes of the graph as per dijkstra algorithm.
- **ShortestPath(int , int)** : It retrieves the shortest path from source to destination using reverse pointer created in Dijkstra method.

Algorithms :

- ALGORITHM INITIALIZE()

Let node[] be the array of objects of class Node_Table_Directory and I is as counter.

```
1.  [Initialize all the objects]
    REPEAT FOR I =0,1,2,…,16
        node[I]   <-   Node_Table_Directory
        Node_No(node[I])   <-   I + 1
        City_Name(node[I])   <-   APPROPRIATE_CITY_NAME
        dst_frm_src(node[I])   <-   MAX_VALUE
        List_Ptr(node[I])   <-   Adjecents
        Adj_ptr(List_Ptr(node[I]))   <-   NULL
        reverse(node[I])   <-   NULL
2.  [Finish]
    Return
```

- ALGORITHM CREATE_ADG(NodeNum1 , NodeNum2 , dst)

Here adj , temp and pre_temp are the object references of Adjecents class.

```
Node_No(adj)   <-   NodeNum2
Available_Seats(adj)   <-   7
Weight(adj)   <-   dst
Adj_ptr(adj)   <-   NULL
pre_temp   <-   List_Ptr(node[NodeNum1 - 1])
temp   <-   Adj_ptr(List_Ptr(node[NodeNum1 - 1]))
IF (temp = NULL)
    Adj_Ptr(pre_temp)   <- adj
ELSE
    REPEAT WHILE temp != NULL AND dst > Weight(temp)
        pre_temp   <-   temp
        temp   <-   Adj_Ptr(temp)
    Adj_ptr(adj)   <-   Adj_ptr(pre_temp)
    Adj_ptr(pre_temp)   <-   adj
Repeat same algorithm by swapping NodeNum1 and NodeNum2
```

➢ ALGORITHM SHORTESTPATH (source , dest)

Here tp is temporary object reference to one of the node of graph and will be used to make recursive call.path is arraylist which is used to store node numbers of shortest path between source and destination.

```
tp    <-    node[ dest - 1 ]
IF ( tp != node[ source - 1 ] )
     SHORTESTPATH( source , Node_No(reverse(node[dest - 1])))
ADD Node_No(tp) TO path
```

➢ ALGORITHM DIJKSTRA(source , dest , no_Of_seats)

Here S , V , and Q are LinkedList of type integer.I will be used as counter.The value retrieved by EXTRACT_MIN function is stored in u.temp is object reference of Adjecents.sum_of_Distance is used to get path with minimum cost.

```
dst_frm_src(node[ source - 1])    <-    0
reverse(node[ source - 1 ])    <-    node[ source - 1 ]
REPEAT FOR I = 0 , 1 , 2 , …. 16
     ADD    (I+1)    TO V
     ADD    (I+1)    TO Q
REPEAT WHILE Q IS NOT EMPTY
       u <- 0
      min <- MAX_VALUE
      REPEAT FOR I = 0,1,2,…,SIZE(Q) - 1
           IF ( min > dst_frm_src(node[ Q.get(I) - 1 ] ) )
                min    <-    dst_frm_src(node[ Q.get(I)    - 1 ])
                u    <-    Q.get(I)
      REMOVE THE ELEMENT AT INDEX OF u FROM Q
      ADD    u    TO S
      temp    <-    Adj_Ptr(List_Ptr(node[ u-1 ] ) )
      REPEAT WHILE    temp != null
           IF ( dst_frm_src(node[ Node_No(temp) - 1 ])    > dst_frm_src(node[ u-1 ] ) +
                                                               Weight(temp) )
                reverse(node[ Node_No(temp) - 1 ])    <-    node[ u-1 ]
                dst_frm_src(node[ Node_No(temp) - 1 ])    <- dst_frm_src(node[ u-1 ] ) +
                                                                    Weight(temp)
           temp    <-    Adj_Ptr(temp)
  path    <-    ArrayList<Integer>
  SHORTESTPATH( source , dest )
  sum_of_Distance    <-    dst_frm_src(node[ dest - 1 ])
  REPEAT FOR I = 0,1,..,SIZE(path) - 1
      temp    <-    Adj_Ptr(List_Ptr(node[ path.get(I) - 1 ] ) )
      REPEAT WHILE    Node_No(temp) != path.get(I+1)
           temp    <-    Adj_Ptr(temp)
      Available_Seats(temp)    <-    Available_Seats(temp) - no_Of_seats
      IF (Available_Seats(temp)    <    0 )
           Available_Seats(temp)    <-    Available_Seats(temp) + no_Of_seats
           RETURN(-1)
  RETURN(1)
```

4. Source Code :-

```
import java.util.ArrayList;                          // import all packages for given
program

import java.util.Collections;

import java.util.LinkedHashSet;

import java.util.LinkedList;

import java.util.Scanner;

import java.util.Stack;

import java.io.FileWriter;

import java.io.IOException;

// creating adjecents class structure for adjecents lists

class Adjecents

{

        int Node_No ;

        int Available_Seats ;

        int Weight ;

        Adjecents Adj_Ptr ;

}

// creating node table directory to point each adjecents lists

class Node_Table_Directory

{

        int Node_No ;

        String City_Name ;

        Adjecents List_Ptr ;

        Node_Table_Directory reverse ;

        int dst_frm_src ;

}
```

```
class Operation
{
        Node_Table_Directory node[] = new Node_Table_Directory[17] ;
        ArrayList<Integer> path ;
        int sum_of_Distance = 0 ;
        // in this method it intialize all nodes having object of Node_Table_Directory
        public void Initialize()
        {
                for( int i=0 ; i<17 ; i++ )
                {
                        node[i] = new Node_Table_Directory() ;
                        node[i].Node_No = i+1 ;
                        node[i].List_Ptr = new Adjecents() ;
                        node[i].dst_frm_src = Integer.MAX_VALUE ;
                        node[i].List_Ptr.Adj_Ptr = null ;
                        node[i].reverse = null ;
                }
                node[0].City_Name = "Bengaluru" ;
                node[1].City_Name = "Chennai" ;
                node[2].City_Name = "Hydrabad" ;
                node[3].City_Name = "Kochi" ;
                node[4].City_Name = "Madurai" ;
                node[5].City_Name = "Pondicherry" ;
                node[6].City_Name = "Mysore" ;
                node[7].City_Name = "Coimbatore" ;
                node[8].City_Name = "Ooty" ;
```

```
          node[9].City_Name = "Tirumala" ;

          node[10].City_Name = "Wazhi" ;

          node[11].City_Name = "Vizag" ;

          node[12].City_Name = "Hampi" ;

          node[13].City_Name = "Port-Blair" ;

          node[14].City_Name = "Vellore" ;

          node[15].City_Name = "Kovalam" ;

          node[16].City_Name = "Erode" ;

     }

/* it will take two nodes and insert it into each others adjecents list sorted by weight
of edge*/

     void Create_Adg( int NodeNum1 , int NodeNum2 , int dst )

     {

          Adjecents adj1 = new Adjecents() ;

          adj1.Node_No = NodeNum2 ;

          adj1.Available_Seats = 7 ;

          adj1.Weight = dst ;

          adj1.Adj_Ptr = null ;

          Adjecents pre_temp = node[ NodeNum1 - 1 ].List_Ptr ;

          Adjecents temp = node[ NodeNum1 - 1 ].List_Ptr.Adj_Ptr ;

          if( temp == null )

          {

               pre_temp.Adj_Ptr = adj1 ;

          }

          else

          {

               while( temp != null && dst > temp.Weight )
```

```
            {
                  pre_temp = temp ;
                  temp = temp.Adj_Ptr ;
            }
            adj1.Adj_Ptr = pre_temp.Adj_Ptr ;
            pre_temp.Adj_Ptr = adj1 ;
      }
      Adjecents adj2 = new Adjecents() ;
      adj2.Node_No = NodeNum1 ;
      adj2.Available_Seats = 7 ;
      adj2.Weight = dst ;
      adj2.Adj_Ptr = null ;
      Adjecents pre_temp1 = node[ NodeNum2 - 1 ].List_Ptr ;
      Adjecents temp1 = node[ NodeNum2 - 1 ].List_Ptr.Adj_Ptr ;
      if ( temp1 == null )
            pre_temp1.Adj_Ptr = adj2 ;
      else
      {
            while ( temp1 != null && dst > temp1.Weight )
            {
                  pre_temp1 = temp1 ;
                  temp1 = temp1.Adj_Ptr ;
            }
            adj2.Adj_Ptr = pre_temp1.Adj_Ptr ;
            pre_temp1.Adj_Ptr = adj2 ;
      }
}
```

```
/*this method takes source and destination ..and performs dijkstra algorythm
  it also sets reverse pointers towards the source whenever needs*/
      int Dijkstra( int source , int dest , int no_Of_seats )
      {
            node[ source - 1 ].dst_frm_src = 0 ;
            node[ source - 1 ].reverse = node[ source - 1 ] ;
            LinkedList<Integer> S = new LinkedList<Integer>() ;
            LinkedList<Integer> V = new LinkedList<Integer>() ;
            LinkedList<Integer> Q = new LinkedList<Integer>() ;
            for ( int i=0 ; i<17 ; i++ )
            {
                  V.add(i+1) ;
                  Q.add(i+1) ;
            }
            while( !Q.isEmpty() )
            {
                  int u = 0 ;
                  int min = Integer.MAX_VALUE ;
                  for ( int i=0 ; i<Q.size() ; i++ )
                  {
                        if ( min>node[ Q.get(i) - 1 ].dst_frm_src )
                        {
                              min = node[ Q.get(i) - 1 ].dst_frm_src ;
                              u = Q.get(i) ;
                        }
                  }
                  Q.remove( Q.indexOf(u) ) ;
```

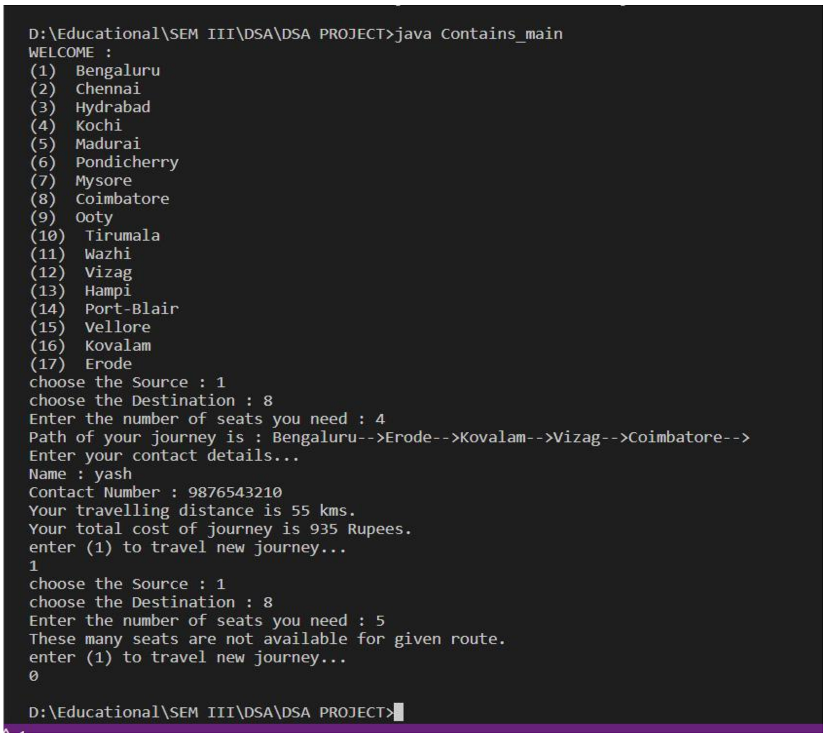
```
        S.add(u) ;

        Adjecents temp = node[ u-1 ].List_Ptr.Adj_Ptr ;

        while( temp != null )

        {

              if ( node[ temp.Node_No - 1 ].dst_frm_src >
                 node[ u-1 ].dst_frm_src + temp.Weight )

              {

                    node[ temp.Node_No - 1 ].reverse = node[ u-1 ] ;

                    node[ temp.Node_No - 1 ].dst_frm_src =
                            node[ u-1 ].dst_frm_src + temp.Weight ;

              }

              temp = temp.Adj_Ptr ;

        }

  }

  path = new ArrayList<Integer>() ;

  ShortestPath( source , dest ) ;

  sum_of_Distance = node[ dest - 1 ].dst_frm_src ;

  for ( int i=0 ; i<path.size()-1 ; i++ )

  {

        Adjecents temp = node[ path.get(i) - 1 ].List_Ptr.Adj_Ptr ;

        while( temp.Node_No != path.get(i+1) )

        {

              temp = temp.Adj_Ptr ;

        }

        temp.Available_Seats = temp.Available_Seats - no_Of_seats ;

        if ( temp.Available_Seats < 0 )

        {

              temp.Available_Seats += no_Of_seats ;
```

```
                        return(-1) ;
                    }
                }
                return(1) ;
        }
/* this method is for retrive path between souce and destination with minimum cost
using reverse pointer*/
        void ShortestPath( int source , int dest )
        {
                Node_Table_Directory tp = node[ dest - 1 ] ;
                if( tp != node[ source - 1 ] )
                        ShortestPath( source , node[dest-1].reverse.Node_No ) ;
                path.add(tp.Node_No) ;
        }
}


class Contains_main
{
        int source ;
        int dest ;
        public static void main( String args[] ) throws IOException
        {
                Operation obj = new Operation() ;
                obj.Initialize() ;
// creating adges by calling Create_adg function


                obj.Create_Adg( 1 , 2 , 13 ) ;
obj.Create_Adg( 1 , 3 , 89 ) ;

obj.Create_Adg( 1 , 4 , 8 ) ;

obj.Create_Adg( 1, 5 , 9 ) ;

obj.Create_Adg( 1 , 6 , 83 ) ;

obj.Create_Adg( 1 , 7 , 47 ) ;

obj.Create_Adg( 1 , 11 , 100 ) ;

obj.Create_Adg( 1 , 12 , 33 ) ;

obj.Create_Adg( 2 , 3 , 17 ) ;

obj.Create_Adg( 2 ,4 , 49 ) ;

obj.Create_Adg( 2 , 5 , 37 ) ;

obj.Create_Adg( 2 , 11 , 117 ) ;

obj.Create_Adg( 2 , 14 , 131 ) ;

obj.Create_Adg( 2 , 15 , 129 ) ;

obj.Create_Adg( 3 , 6 , 12 ) ;

obj.Create_Adg( 3 , 7 , 2 ) ;

obj.Create_Adg( 3 , 11 , 11 ) ;

obj.Create_Adg( 3 , 12 , 44 ) ;

obj.Create_Adg( 3 , 8 , 51 ) ;

obj.Create_Adg( 3 , 9 , 49 ) ;

obj.Create_Adg( 8 , 12 , 39 ) ;

obj.Create_Adg( 9 ,12 , 46 ) ;

obj.Create_Adg( 10 , 11 , 30 ) ;

obj.Create_Adg( 10 , 12 , 13 ) ;

obj.Create_Adg( 11 , 12 , 33 ) ;

obj.Create_Adg( 11 , 13 , 67 ) ;

obj.Create_Adg( 11 , 14 , 12 ) ;

obj.Create_Adg( 1 , 17 , 3 ) ;
obj.Create_Adg( 2 , 17 , 45 ) ;
obj.Create_Adg( 3 , 17 , 44 ) ;
obj.Create_Adg( 11 , 15 , 13 ) ;
obj.Create_Adg( 12 , 13 , 6 ) ;
obj.Create_Adg( 11 , 16 , 111 ) ;
obj.Create_Adg( 12 , 16 , 8 ) ;
obj.Create_Adg( 16 , 17 , 5 ) ;
System.out.println("WELCOME : ") ;
int cont = 0 ;
int code = 0 ;
for ( int i=0 ; i<17 ; i++ )
{
        System.out.println( "(" + (i+1) + ")    " + obj.node[i].City_Name ) ;
}
do
{
        System.out.print("choose the Source : ") ;
        Scanner sc = new Scanner( System.in ) ;
        int source = sc.nextInt() ;
        System.out.print("choose the Destination : ") ;
        int dest = sc.nextInt() ;
        System.out.print("Enter the number of seats you need : ") ;
        int no_of_seats = sc.nextInt() ;
        code = obj.Dijkstra( source , dest , no_of_seats ) ;
        if( code == 1 )
        {
                System.out.print("Path of your journey is : ") ;
                for ( int i=0 ; i<obj.path.size() ; i++ )

                {

                        System.out.print( obj.node[ obj.path.get(i) - 1 ].City_Name
                                                + "-->" ) ;

                }

                System.out.println() ;

                System.out.println("Enter your contact details...") ;

                System.out.print("Name : ") ;

                String name = sc.next() ;

                System.out.print("Contact Number : ") ;

                String no = sc.next() ;

                FileWriter fl = new FileWriter( "Passenger_Info.txt" , true ) ;
                                  // insert passenger details in files

                fl.write( name + ":" + no + "        Source : " +
                        obj.node[source - 1].City_Name + "    " +
                        "   Destination : " + obj.node[dest - 1].City_Name +
                        "       Number of seats :- " + no_of_seats);

                fl.write(13) ;

                fl.close();

                System.out.println("Your travelling distance is " +
                                      obj.sum_of_Distance + " kms.") ;

                System.out.println("Your total cost of journey is " +
                                       obj.sum_of_Distance*17 + " Rupees.") ;

        }

        else if ( code == -1 )

        {

                System.out.println("These many seats are not available for given
                                        route.") ;

        }

        System.out.println("enter (1) to travel new journey...");
```

```
                cont = sc.nextInt() ;
            }
            while( cont == 1 ) ;
        }
    }
```

5. Testing :-

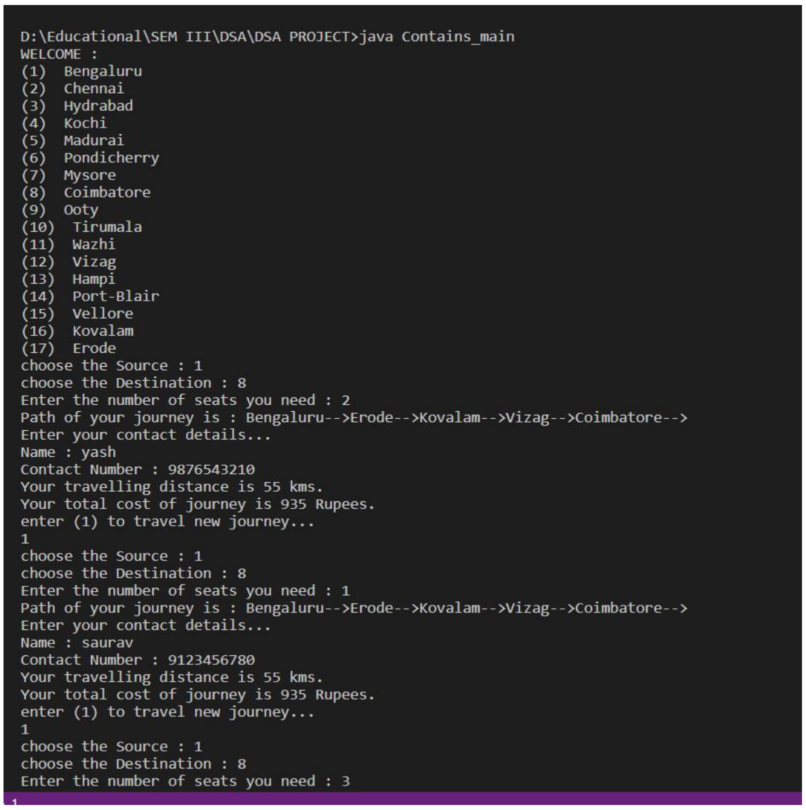
Testing 1 :



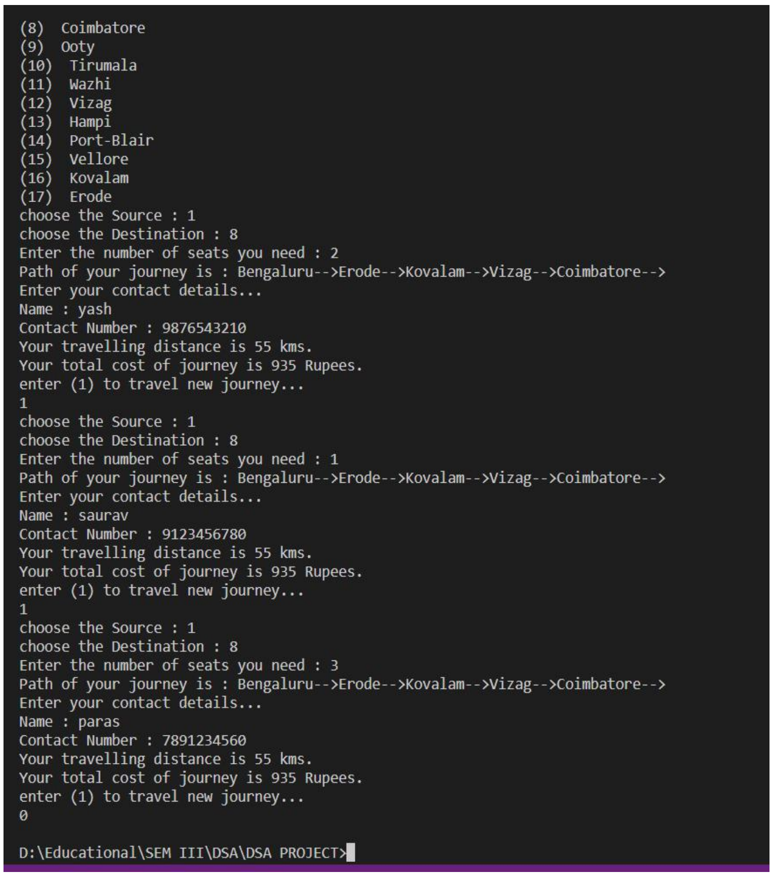
```
D:\Educational\SEM III\DSA\DSA PROJECT>java Contains_main
WELCOME :
(1)  Bengaluru
(2)  Chennai
(3)  Hydrabad
(4)  Kochi
(5)  Madurai
(6)  Pondicherry
(7)  Mysore
(8)  Coimbatore
(9)  Ooty
(10)  Tirumala
(11)  Wazhi
(12)  Vizag
(13)  Hampi
(14)  Port-Blair
(15)  Vellore
(16)  Kovalam
(17)  Erode
choose the Source : 1
choose the Destination : 8
Enter the number of seats you need : 4
Path of your journey is : Bengaluru-->Erode-->Kovalam-->Vizag-->Coimbatore-->
Enter your contact details...
Name : yash
Contact Number : 9876543210
Your travelling distance is 55 kms.
Your total cost of journey is 935 Rupees.
enter (1) to travel new journey...
1
choose the Source : 1
choose the Destination : 8
Enter the number of seats you need : 5
These many seats are not available for given route.
enter (1) to travel new journey...
0

D:\Educational\SEM III\DSA\DSA PROJECT>
```

Here in second iteration it shows that seats are not available because in this program we take 7 as maximum available seats between two cities.That is more than the sum of required seats which is 9(=(4+5)).

Testing 2 :

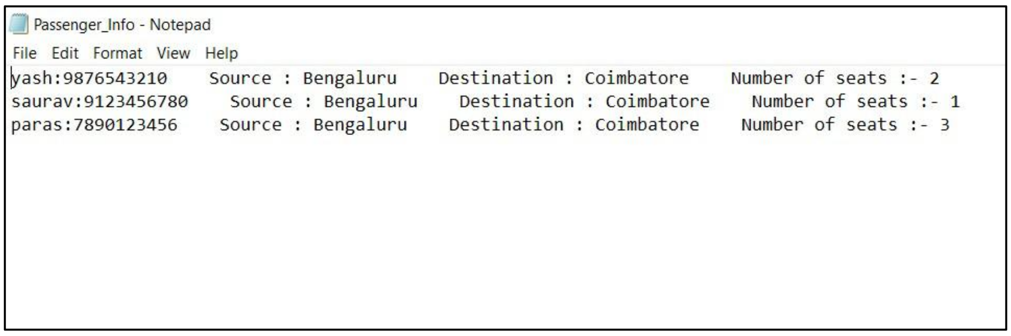


```
D:\Educational\SEM III\DSA\DSA PROJECT>java Contains_main
WELCOME :
(1)  Bengaluru
(2)  Chennai
(3)  Hydrabad
(4)  Kochi
(5)  Madurai
(6)  Pondicherry
(7)  Mysore
(8)  Coimbatore
(9)  Ooty
(10)  Tirumala
(11)  Wazhi
(12)  Vizag
(13)  Hampi
(14)  Port-Blair
(15)  Vellore
(16)  Kovalam
(17)  Erode
choose the Source : 1
choose the Destination : 8
Enter the number of seats you need : 2
Path of your journey is : Bengaluru-->Erode-->Kovalam-->Vizag-->Coimbatore-->
Enter your contact details...
Name : yash
Contact Number : 9876543210
Your travelling distance is 55 kms.
Your total cost of journey is 935 Rupees.
enter (1) to travel new journey...
1
choose the Source : 1
choose the Destination : 8
Enter the number of seats you need : 1
Path of your journey is : Bengaluru-->Erode-->Kovalam-->Vizag-->Coimbatore-->
Enter your contact details...
Name : saurav
Contact Number : 9123456780
Your travelling distance is 55 kms.
Your total cost of journey is 935 Rupees.
enter (1) to travel new journey...
1
choose the Source : 1
choose the Destination : 8
Enter the number of seats you need : 3
```



```
(8)  Coimbatore
(9)  Ooty
(10)  Tirumala
(11)  Wazhi
(12)  Vizag
(13)  Hampi
(14)  Port-Blair
(15)  Vellore
(16)  Kovalam
(17)  Erode
choose the Source : 1
choose the Destination : 8
Enter the number of seats you need : 2
Path of your journey is : Bengaluru-->Erode-->Kovalam-->Vizag-->Coimbatore-->
Enter your contact details...
Name : yash
Contact Number : 9876543210
Your travelling distance is 55 kms.
Your total cost of journey is 935 Rupees.
enter (1) to travel new journey...
1
choose the Source : 1
choose the Destination : 8
Enter the number of seats you need : 1
Path of your journey is : Bengaluru-->Erode-->Kovalam-->Vizag-->Coimbatore-->
Enter your contact details...
Name : saurav
Contact Number : 9123456780
Your travelling distance is 55 kms.
Your total cost of journey is 935 Rupees.
enter (1) to travel new journey...
1
choose the Source : 1
choose the Destination : 8
Enter the number of seats you need : 3
Path of your journey is : Bengaluru-->Erode-->Kovalam-->Vizag-->Coimbatore-->
Enter your contact details...
Name : paras
Contact Number : 7891234560
Your travelling distance is 55 kms.
Your total cost of journey is 935 Rupees.
enter (1) to travel new journey...
0

D:\Educational\SEM III\DSA\DSA PROJECT>
```

Here we exists from program until we asks for more than available seats(7). So it will simply allocate all the three users to their required seats and stores their details.



```
yash:9876543210     Source : Bengaluru    Destination : Coimbatore    Number of seats :- 2
saurav:9123456780     Source : Bengaluru    Destination : Coimbatore    Number of seats :- 1
paras:7890123456     Source : Bengaluru    Destination : Coimbatore    Number of seats :- 3
```

Above files stores details of all the users to whom the seats are allocated successfully.