I. "Trains" Problem

The local commuter railroad services a number of towns in Kiwiland. Because of monetary concerns, all of the tracks are 'one-way.' That is, a route from Kaitaia to Invercargill does not imply the existence of a route from Invercargill to Kaitaia. In fact, even if both of these routes do happen to exist, they are distinct and are not necessarily the same distance!

The purpose of this problem is to help the railroad provide its customers with information about the routes. In particular, you will compute the distance along a certain route, the number of different routes between two towns, and the shortest route between two towns.

Input: A directed graph where a node represents a town and an edge represents a route between two towns. The weighting of the edge represents the distance between the two towns. A given route will never appear more than once, and for a given route, the starting and ending town will not be the same town.

Output: For test input 1 through 5, if no such route exists, output 'NO SUCH ROUTE'. Otherwise, follow the route as given; do not make any extra stops! For example, the first problem means to start at city A, then travel directly to city B (a distance of 5), then directly to city C (a distance of 4).

1. The distance of the route A-B-C.
2. The distance of the route A-D.
3. The distance of the route A-D-C.
4. The distance of the route A-E-B-C-D.
5. The distance of the route A-E-D.
6. The number of trips starting at C and ending at C with a maximum of 3 stops. In the sample data below, there are two such trips: C-D-C (2 stops). and C-E-B-C (3 stops).
7. The number of trips starting at A and ending at C with exactly 4 stops. In the sample data below, there are three such trips: A to C (via B,C,D); A to C (via D,C,D); and A to C (via D,E,B).
8. The length of the shortest route (in terms of distance to travel) from A to C.
9. The length of the shortest route (in terms of distance to travel) from B to B.
10. The number of different routes from C to C with a distance of less than 30. In the sample data, the trips are: CDC, CEBC, CEBCDC, CDCEBC, CDEBC, CEBCEBC, CEBCEBCEBC.

Test Input:
For the test input, the towns are named using the first few letters of the alphabet from A to D. A route between two towns (A to B) with a distance of 5 is represented as AB5.
Graph: AB5, BC4, CD8, DC8, DE6, AD5, CE2, EB3, AE7
Expected Output:
Output #1: 9
Output #2: 5
Output #3: 13
Output #4: 22
Output #5: NO SUCH ROUTE
Output #6: 2
Output #7: 3
Output #8: 9
Output #9: 9
Output #10: 7

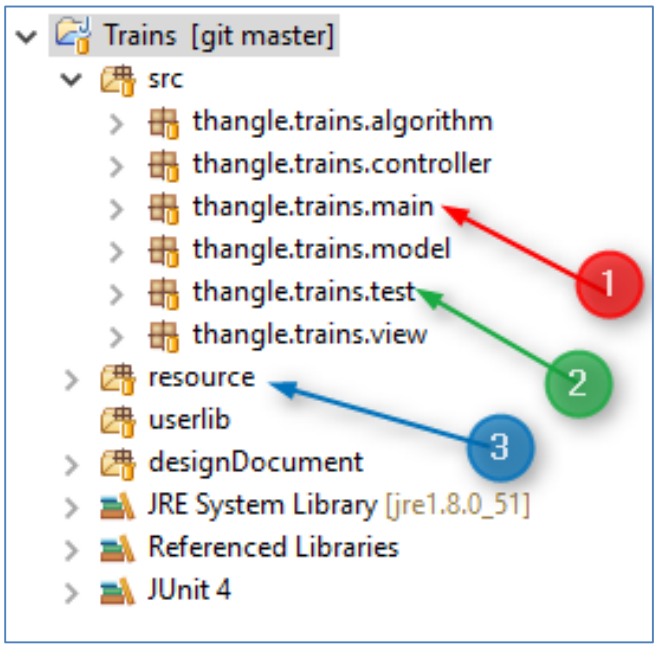
II. How to run the "Trains" application

1. Setup environment

- You need the Eclipse Java IDE to import the project.
- Java JDK 1.7 or 1.8 to build and run the project.

2. How to run

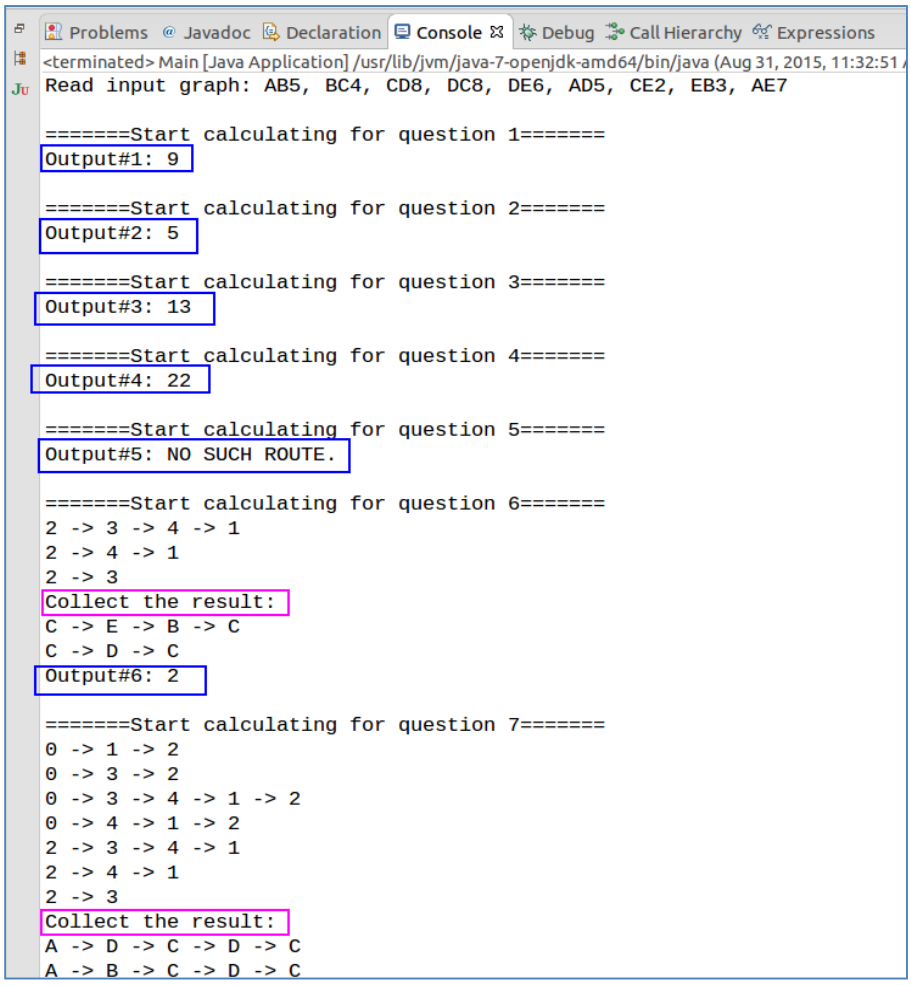
- Import project **Trains** to Eclipse and select "*Package Explorer*" as the figure below.



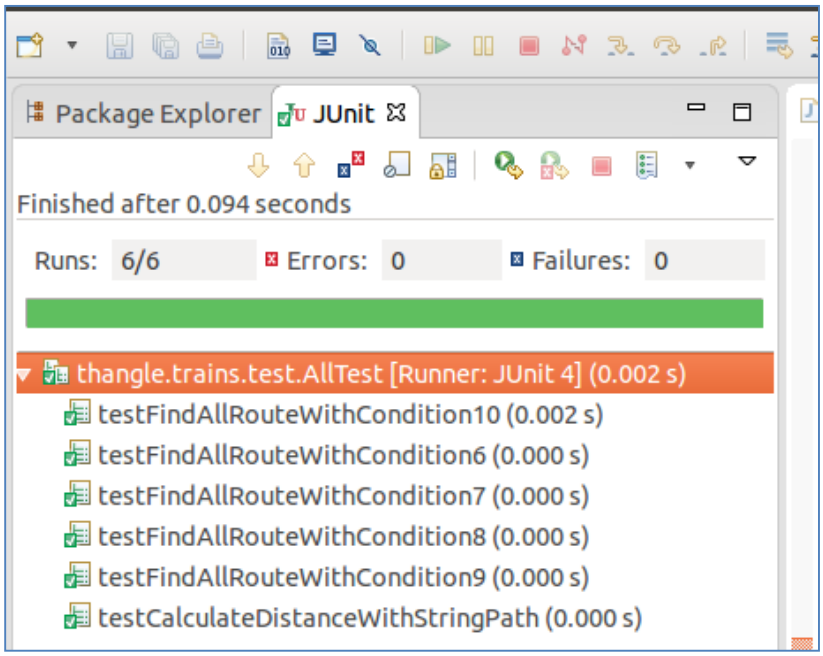
- In **src** folder, select class **Main.java** (No.1) and run as "*Java Application*" to execute the program or go to class **AllTest.java** to execute unit test (Junit) as above image (No.2).
- In **resource** folder, open file **GraphData.txt** to change input data as above image (No.3).
- The format in file **GraphData.txt** needs follow as in example. To add a new node, the new node name is only letters from "A" to "Z" and to be only *increased one by one*.
  For example: current graph is: **AB5, BC4, CD8, DC8, DE6, AD5, CE2, EB3, AE7**
  To add new node, then the new letter should be used is "**F**". Eg. **AB5, BC4, CD8, DC8, DE6, AD5, CE2, EB3, AE7, EA7, AF6, FD15** (check file **GraphData2.txt** for reference).
- To change the question, go to method **calculateUserRequests** of class **GraphUserView.java** and change arguments of methods as image below.

```
public void calculateUserRequests() {

    /*
     * The distance of the route A-B-C.
     * Note: A=0, B=1, C=2, D=3, E=4.
     */
    graphController.FindAllRouteWithCondition1("012"); // A-B-C. (= 9)

    /*
     * The distance of the route A-D.
     * Note: A=0, B=1, C=2, D=3, E=4.
     */
    graphController.FindAllRouteWithCondition2("03"); // A-D. (=5)

    /*
     * The distance of the route A-D-C.
     * Note: A=0, B=1, C=2, D=3, E=4.
     */
    graphController.FindAllRouteWithCondition3("032"); // A-D-C.(=13)

    /*
```

- The results are printed out to Eclipse console (as image below).



- Unit test result (just do some test cases)



III. Design overview

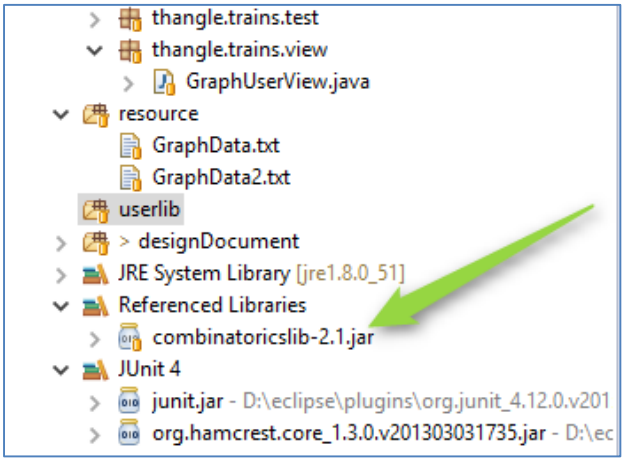
1. Software structure

a. I try to design the program using MVC design pattern as possible. The software consists 6 source code packages.

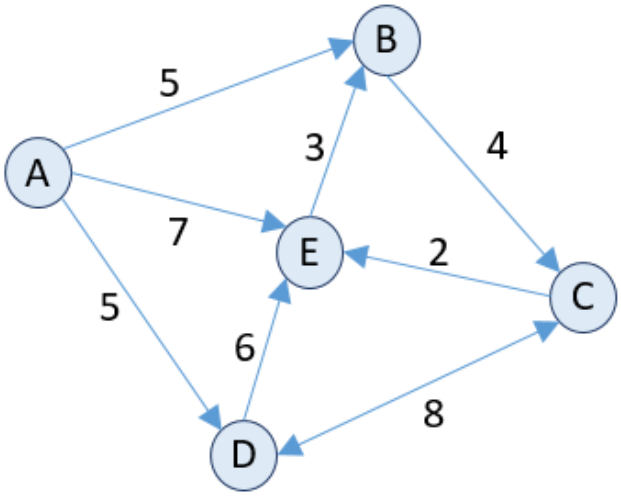
- **thangle.trains.algorithm**: implementation of algorithms as Depth First Search, Dijkstra. For combinatorial algorithm, I used a library at: https://code.google.com/p/combinatoricslib/
- **thangle.trains. controller:** The controller that to process requests from View.
- **thangle.trains. model:** the models which hole data.
- **thangle.trains.view**: the view, a simple class to display data to user.
- **thangle.trains.main:** implementation of main class. Program starts from here.
- **thangle.trains.test:** implementation of unit test using Junit.

b. The **resoure** folder to store resource files (text files as input data).

c. A user library (*combinatoricslib-2.1.jar*) for combinatorial algorithm is put into the folder */Trains/ userlib* as image below.



2. The solution for the problem



- For question 1 to 5: just simple implement a function to calculate the distance of a route with input is nodes of the route. For example, the distance from A-B-C = distance(A-B) + distance(B-C).

- To find all routes from node N to N (N is any node in graph). We use *depth first search* algorithm to find all routes from N to Ka. Here Ka is set of all adjacent nodes of node N.
- To find all routes from node N to N (N is any node in graph) that the routes are allowed repeatedly, firstly we find all routes from node N to N without repeatedly as the above step and then do a combination of these found routes by using the *combinatorial* algorithm to get the final result. Using same idea for shortest path from node N to N (*Dijkstra* and *combinatorial* algorithm).
- For question 6, 7, 10: to use *depth first search* and *combinatorial* algorithm for calculation.
- For question 8, 9: to use *Dijkstra* and *combinatorial* algorithm for calculation.

3. Class diagram

- Below is the class diagram of the program (draw by IBM Rational Rhapsody).

End.

Thang Le.

http://letrungthang.blogspot.sg