**The introduction of the solution for problem one, trains. (Java)**

There is 5 parts in this project:

1. Vertex
2. Edge

(2) Graph

(3) Action

(4) Output

1. Vertex

This class is to describe the point in graph, it has 2 members, index and value, and the value can be any type of an object.

**Field Members:**

int index; // the index it in graph

V value;

3. Graph

**Field Members:**

(1) Vector<Edge<V>> edgeList; //store all the edges

(2) Vector<String> v\_list;

//store the key value of vertex, the index in vertex maps to the index of vertex in v\_list

(3) float[][] edges;

//store the distance between vertex, the row and colum index maps to index of vertex in v\_list

(4) int v\_num; //the number of vertexes

(5) int edge\_num; //the number of edges

4. Action

This class mainly aims at supplying some method to operate graph.

**Field Members:**

(1) Graph<String> graph;

(2) int max=65535; // refers to infinite numbers

(3) class Node{

int index;

float distance;

} //this is mainly for computing number of routes limited by distance

**Method Members:**

(1) void initGraph(String txtPath)

It initiates the graph, including its edge list, vertex list and two-dimension array edges mapping the vertexes and edges.

(2) void initEdges()

After initGraph, set the edges[][] according to the edge list and vertex list.

(3) float computeDistance(int[] indexs)

Computing the distance among the given route, indexes[], containing the index of given vertexes, returning a total distance added by the distance between 2 adjacent vertexes.



Fig. 1 The process of computeDistance.

(4) float shortestRoute(String vertex1, String vertex2)

To calculate the shortest distance between 2 given points, and the principle refers to Dijkstra. The different part compared with Dijkstra is that it will return the result once the destination point has been calculated.



Fig. 2 The process of shortestRoute.

(5) int routeNum(String srcVertex, String destVertex, int level, boolean total)

The third parameter refers to the stops limited in route.



Fig. 3 Paths in tree.



Fig. 4 The process of routeNum.

The fourth parameter refers to whether the number of stops in route is definitely the given number or within the number.

Treat the source point as a tree root node, then all the routes, exclude the end point, are the branches of the tree with the depth of level. If total is true, it needs to get the nodes in each level top-down, except root node, then get the index of each node as the row in edges[][] to search it whether has a connection with the end point, if true then the routes number plus 1. If total is false, only need to obtain all the leaf node of the tree. When it traverses the tree, it only adds the current level nodes and the last level nodes into 2 queues, by the way, the node here is not exactly a node, but an integer refers to vertex index.

(6) int routeNumLimitedByDistance(String srcVertex, String destVertex, float distance)

This method is similar to the method above. The different it is limited by distance when traversing the tree and the last one is limited by the given level. It employs the class Node, which is added into queue here, as the tree node, conveniently to compute the sum distance from the root.



Fig. 5 The process of routeNumLimitedByDistance.

5.Output

It offers the 10 output of the problem.