Shortest and Least Time-Spent Routing in Subway System

EL7363 Communications Networks II:

0486634 Yaolong Li

Outline

- 1 Search backgroud
- 2 Two ways to find the paths (A:find all the paths first. B:using a modified Dijkstra Algorithm)
- 3 The algorithm how to find all the paths, and the algorithms to pick the path which take the least nodes or least time.
- 4 The reason why the Dijkstra is not suitable in subway model.
- 5 The modified Dijkstra Algorithm.
- 6 The comparison between method A and method B.

BACKGROUND

- **In big cities like New York and Chicago, a great amount of traffic is in divided into subway system, since the roads in these cities cannot hold so many vehicles and not every one can afford a private car.**
- **People may have destinations and need a path to follow in a subway system. A single train cannot take the whole trip and people need to change their trains in subway station.**

BACKGROUND

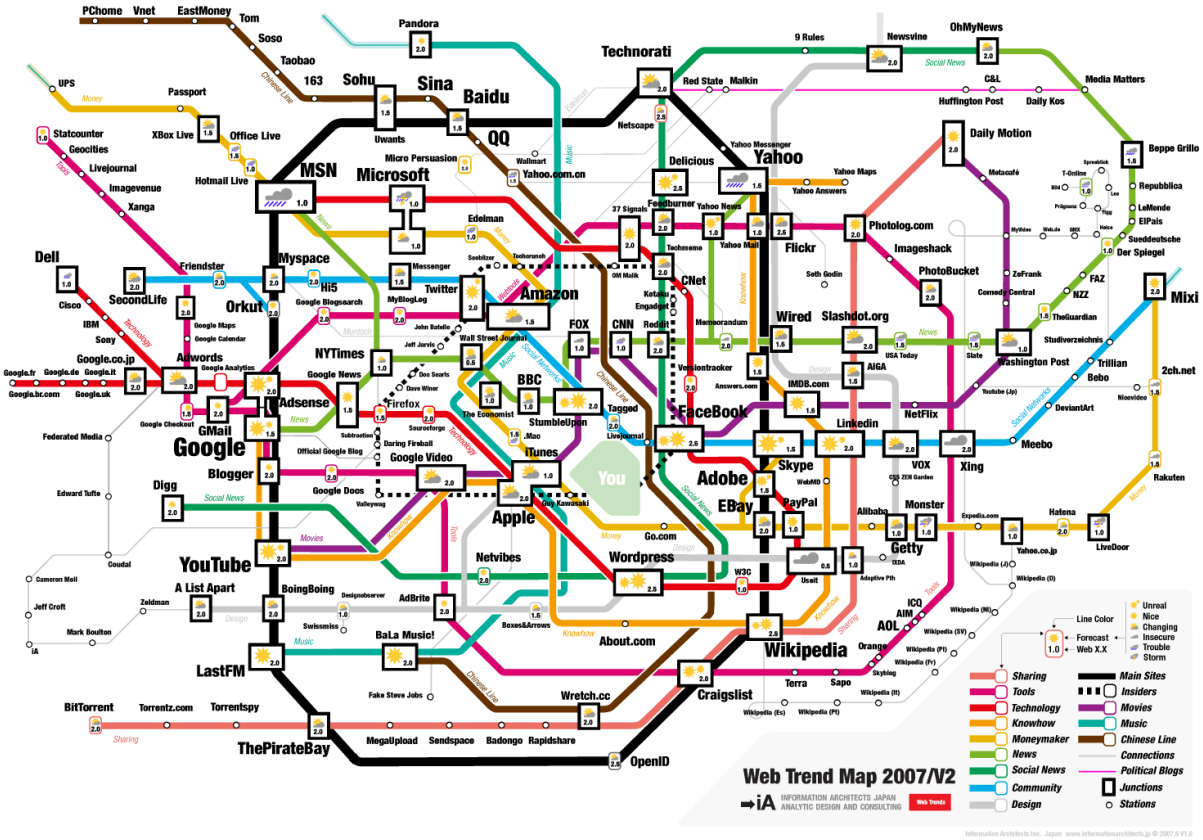
- **For one day, the timetable for a train can be changed to fit the traffic.**

  In the rush hour, more train will help people get to work.

- **In the case of the unestimatable factor, such as rainstorm, train damages, railway blocking, the links between station can change greatly.**
- **The railway company need to update these change to the public and then citizens may change their path to save time or money.**

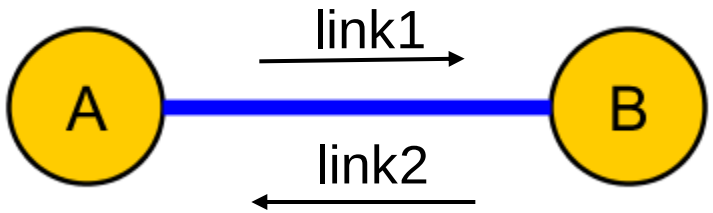
Find the Best Path in The System

- In this way, people should find the most suitable path. Since the complexity of the subway system, additional technologies needed.



Model The System

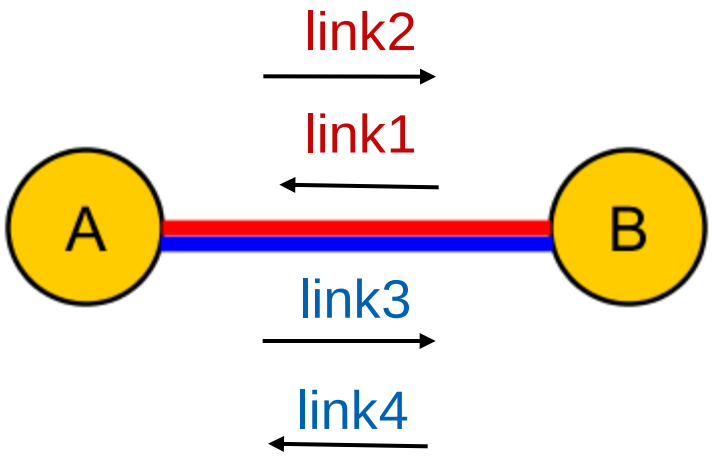
- In this model, we have N Stations, linked by L links. Stations are refered as Nodes in this system.
- Links are directed, which means for the same line between two stations (nodes), there are two links. And these two links have opposite direction.



- For example,Node A and Node B is linked by a railway, however, there are two directed links on this road, they are link1 and link2.

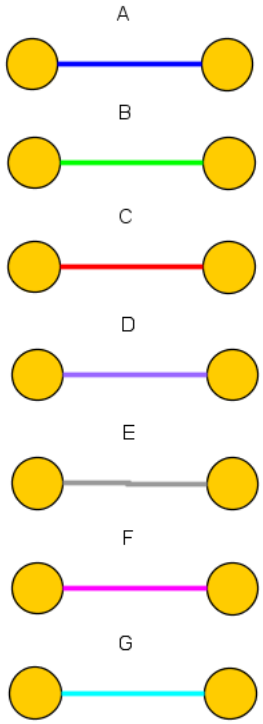
Model The System

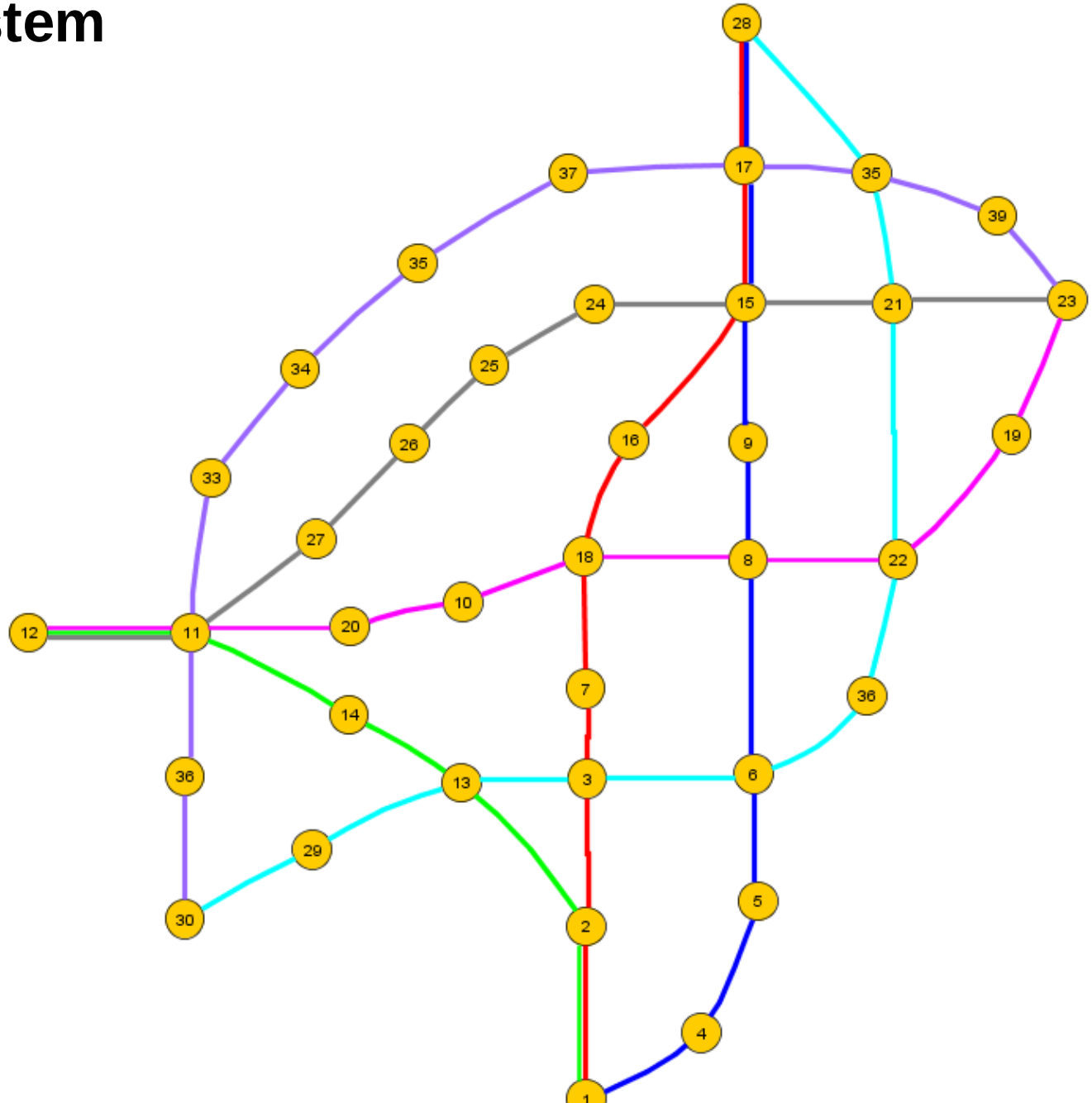
- For two station, there may more than one subway lines. Thus, more links are put there.
- For each link, there only one subway line.



Model The System

- The virtual map



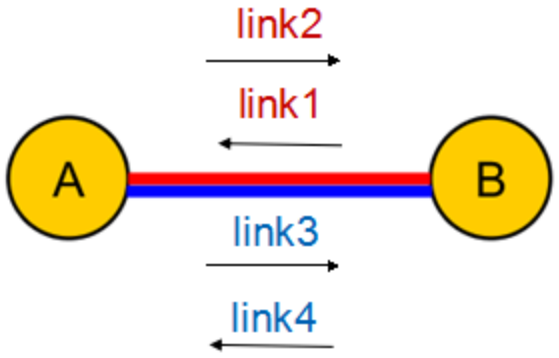


Model Setting

- **In this virtual map, 38 nodes are contained. 112 links are used to link whole nodes together.**
- **There are 7 different subway lines in this system. Different subway have different timetable.**
- **Node: For each node, has two list : a list collects the links come into the node, and a list collects the links come from this node.**
- **Link: For each link, has a original node and terminal node.**

Node A has a "into" list (link1,link4), and a "from" list(link2,link3).

Link1 has a orginal node B and a terminal node A



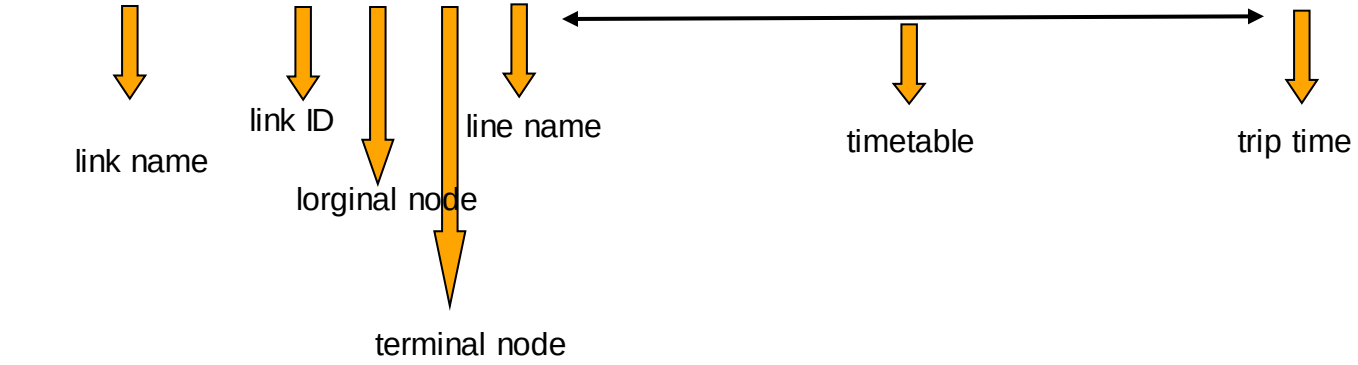
Links in this Model.

- For a link, it has a list of time table, which stands for the time this link begins. (like in real world, for a subway, there are many train come to the same sation as the same line.)

- Also, the link has a trip time, means how long this link lasts.

in java coding

```
/////////  A line links //////
Link link1=new Link(1,node1,node4,"A",0,30,100,200,340,400,515,620,700,820,915,1025,30);
Link link2=new Link(2,node4,node1,"A",310,340,410,510,700,710,825,930,1010,1130,1225,1335,30);
```



Link All The Nodes With Links

- **Then, after initialize the nodes and link, we can link them together.**

```
node1.addIntoLink(link34);
node1.addIntoLink(link18);
node1.addIntoLink(link2);
node1.addOutList(link33);
node1.addOutList(link17);
node1.addOutList(link1);
```

- **The method *Node B.addIntoLink(Link A)*, is used to add Link A to the "into" list of this Node B. Then, use *Node B.addOutLink(Link C)* to add Link C to the "out" list of this Node B.**

Different Algorithms to Find Path

- **1 Find all the pathes, then, pick the path which takes the least time or nodes.**

- **2 Using Dijkstra Algorithm to find the shortest path.**

However, the original version of Dijkstra is not suitable in this model, since there are more lines can lead to the same node! This will be discuss later.

1 Algorithm to Find All The Pathes

- 1. Suppose that the user is in station A and wants to go to station B.
- 2. Step 1: Firstly, get the node(station) A's out list. In the list, get the first link: link1. Then, we can go to station X1 with link1. (since for one link there is only one terminal and one original node)
- 3 Step 2: Check X1,

  (1)if X1 is node B, then stop , we get the terminal. Remember this path.

  (2)If X1 is A or other stations which we have gone through, stop using this link1, use the second link(link2) in the out list of A, then went to step 2 and check the link2's terminal node X2.

  (3)else, it's the first time to go through X1, add X1 to a node list, called used-node list. Then, start from X1 to follow step 1 again.

- 4 Try all the nodes and links, then we get all the path available.

1 Algorithm to Find All The Pathes

- codes in java

```
public  void search(Node here,Node aim, PassedNodes nodes, Path history){
       for (int i=0;i<here.getOutList().size();i++){
           if (nodes.getNodes().contains(here.getOutList().get(i).getTo().getNodeID()+"")){
           } else if (here.getOutList().get(i).getTo()==aim)
           {
               pathList.add(pathAddLink(history,here.getOutList().get(i)));
           } else {
               search(here.getOutList().get(i).getTo(),aim, updateNodes(nodes,here.getOutList().get(i).getTo()),pathAddLink(history,here.getOutList().get(i)));
               continue;
           }
       }
   }
```

- Then, we get all the paths to the same destination and save them into a path list.

Find the Shortest Node-Count Path

- **Since we have all the paths, we can count the number of nodes needed for each of the paths.**
- **1. Get the first path in the path list, get the node needed number N. Set N as the smallest number. Add this path into my shortest path list.**
- **2. Then go to the next path, get node number M.**

f M<N, set M as the smallest number. Then clear the shortest path list and add this path into it.

If M=N, add this path into the shortest path list(since it is possible two paths have the same nodes number.)

If M>N, go to 2. to check again.

Finally, we can get the path enjoys the shortest node-count path.

This can get more than one path which take the least nodes

Find the Least Time-Spent Path

Find the needed time for one path

- In java code, model time like this :

  100=1:00 ; 240=2:40 ; 30=0:30

- 1 First, we have a time now, which means the time right now.
- 2 Then, we compare now with the timetable of the link (Since we have get all the path, and for each path we have the links in order).
- 3 Set begin-time as the time when a link begins in the timetable.

If now > begin-time, go to next one in the timetable, then compare again.

If now< begin-time, it means this begin-time is fine, and user can take it, so, update now=begin-time.

then, now=now + (trip-time)

- For the next node, go through 3 again.

Then we have the end-time of this path

Find the Least Time-Spent Path

- We can compare all the needed time with all the path.
- Then we can get the path which needs the shortest time.
- 1. Get the first path in the path list, get the node needed time T. Set T as the shortest time. Add this path into my shortest path list.
- 2. Then go to the next path, get needed time T1.

f T1<T, set T1 as the smallest number. Then clear the shortest path list and add this path into it.

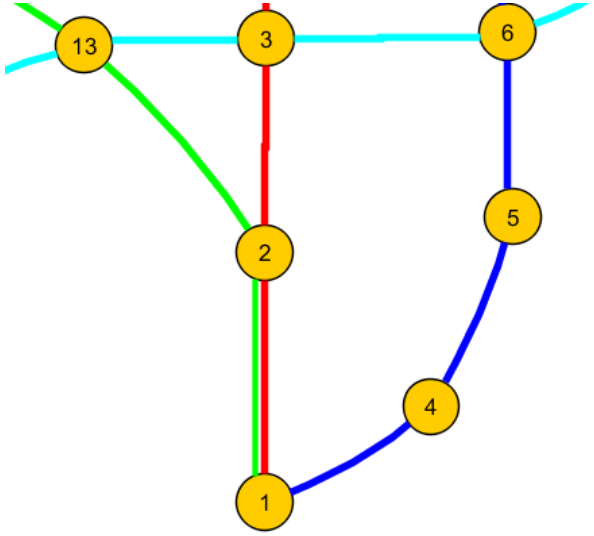
If T1=T, add this path into the shortest path list(since it is possible two paths have the same nodes number.)

If T1>T, go to 2. to check again.

Finally, we can get the path enjoys the shortest-time path.

Why the Dijkstra is not Fittable in Subway Model

- **In the original model of Dijkstra, there is only one link between nodes.**
- **In subway, there is more than one links between nodes.**
- **In subway, , when try to find the time spent on the path, the station waiting-time should be take into consideration.The waiting time can be changed with time, while the on-train time is the same.**



A Modified Dijkstra Algorithm with the concept of Heuristic

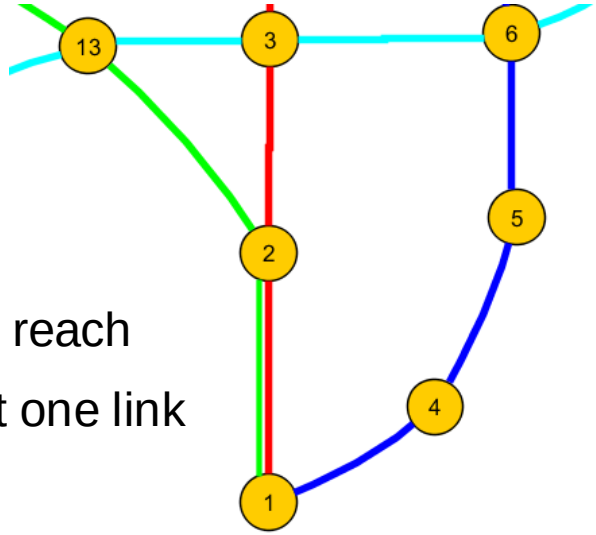
- **Here, we consider a modified Dijkstra. Since to find the least-time spent path is more complex than to find the shortest node path, we will only take consider the least-time case here.**

  (to convert a least-time problem into a shortest node problem,just need to change the time weight to 1)

- **User wants come from Node 1 to Node 6**

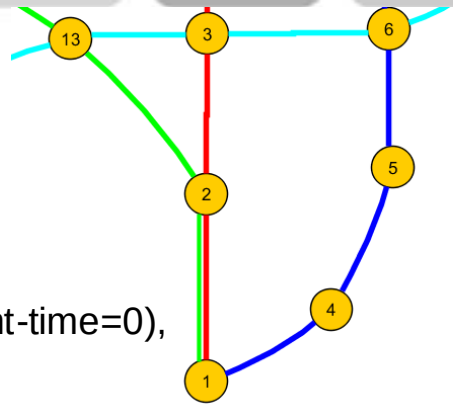
  here, we introduce a concept of label

  label : (node, time-weight,root link)



node: for a single node. time-weight: the time when we reach this node. root link: in order to reach this node, the last one link we need

A Modified Dijkstra Algorithm



1 In iteration 0, we get node 1, and take node label as (node 1, 0, null)

since node 1 is the original node, and we need not wait any time to get to it(weight-time=0), and no root link. Save this label in a list.

2 In iteration 1, we scan the list, we get label about node 1. With these links which come from node 1, we can reach node 2 and node 4. Then, we get labels as below

(node 2, weight-time1=0+wait-time(green)travel-time(green), green link);

(node 2, weight-time2=0+wait-time(red)+travel-time(red), red link);

(node 4, weight-time3=0+wait-time(blue)+travel-time(blue),blue link)

Find the label which owns the smallest weight-time, for example, if weight-time1<weighttime2<weight-time3, we will only keep label about node2 by green link into the list.

3 In iteration 2 Scan the list again, get labels about node 1 and node 2. From these nodes, get their nearby modes (node 3, node 13, node 4) and calculate the new label :

new label : (node X, weight-time=rootnode's weight-time+wait-time+travel-time, a color link root-node and node X.

Save the label which has the smallest weight-time to the list.

4 n iteration 3 Get to iteration 2 again, until X is the node user wants. Then, the terminal node's weight-time is the time to reach the terminal.

Then, from the terminal node, we can get the root, and the root-node, and the rootnode's root....

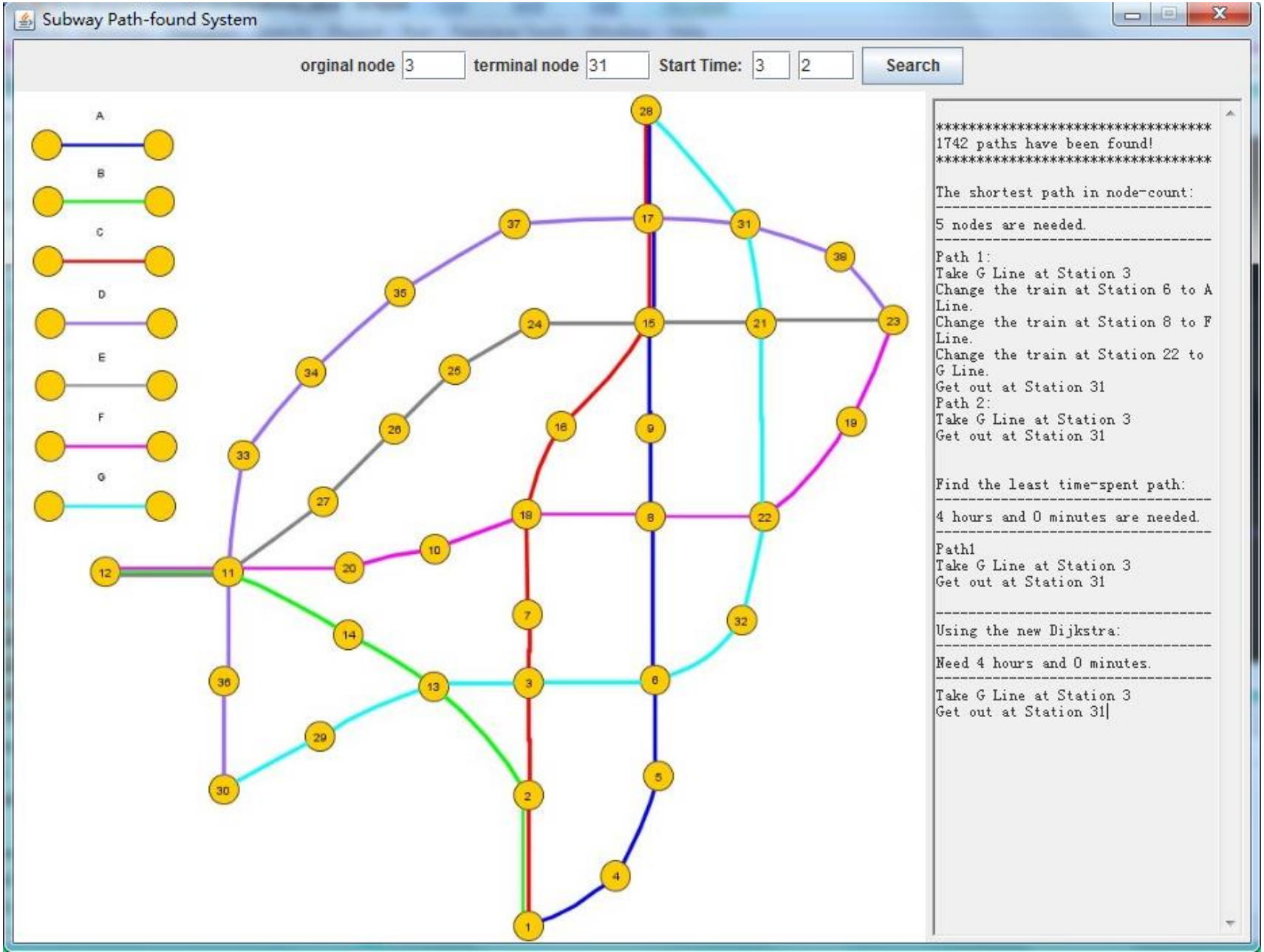
We get all the root links, therefore, we get the path!

The Comparisons

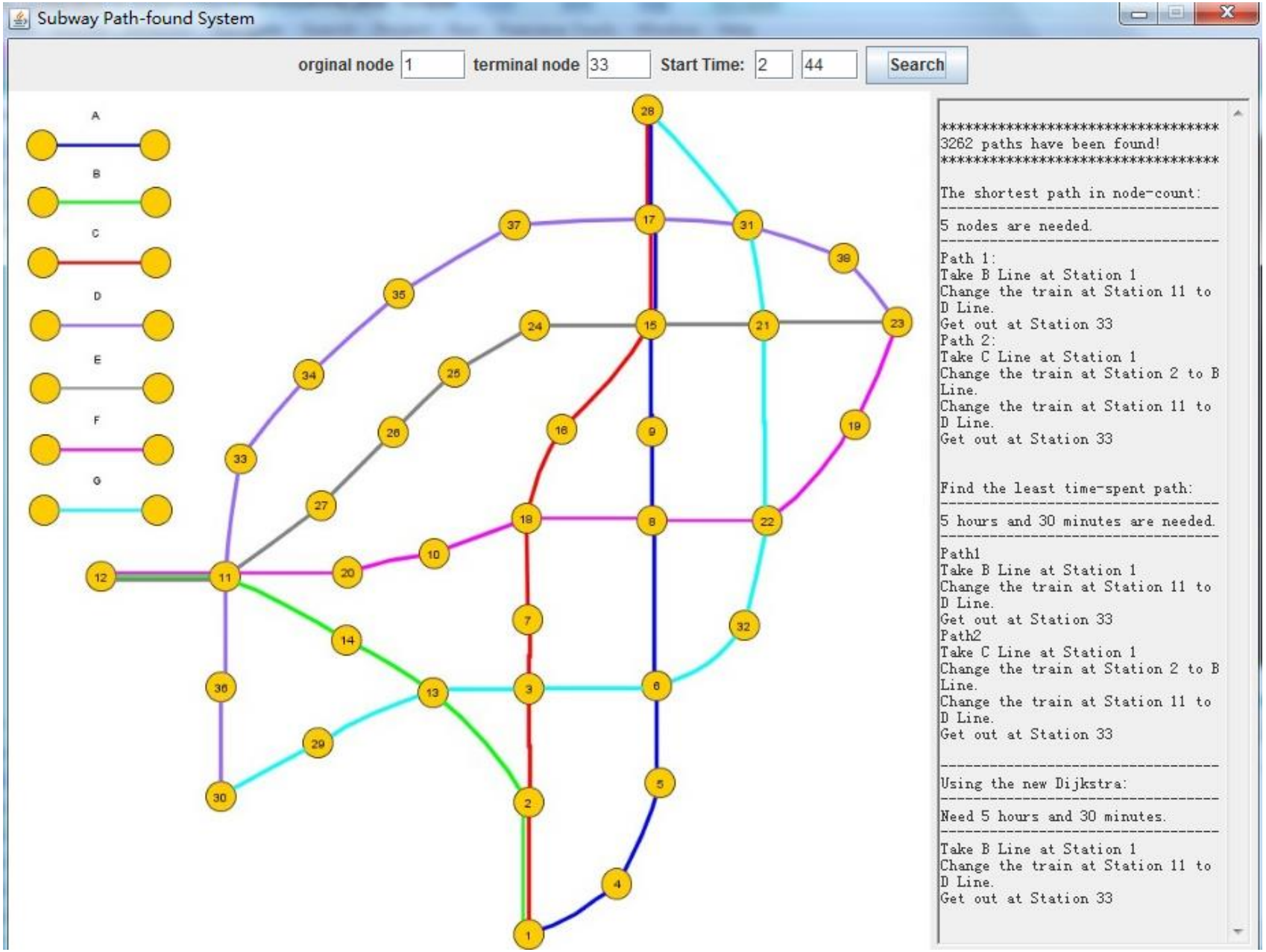
- **The modified Dijkstra Algorithm can save time since not all the path has been found.**

- **However, since for one node, there is only one label, one root link in the modified Dijkstra Algorithm, which means only one path can be found. In the subway system, there may be more than one paths have the least spent time. in the case of the first algorithm in this report, every path can be found!**

Java Code Result 1



Java Code Result 2



Java Code Result

- In the second photo, we can find the modified Dijkstra Algorithm only find one path as has been discussed before.

- However, it does't affect the efficiency. In this Java model, many links and the timetables use the same numbers and structure to simplify the modeling work. In real life, it is difficulty for used to get more than one least-time paths.