[B,H]

[R,H]

[R,T]

[R,B]

[R,B]

[R,H]

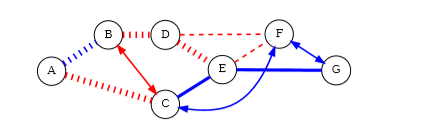
[B,C]

[B,C]

[R,H]

[B,T]

[B,T]



We can solve this problem using graph.

Here we can model that each transfer is vertex.

For example, we can think the link between A and B.

Since we consider the route A->B and B->A in our search, we should model this problem as directed graph. We should note that A->B->A cannot be existed in a route. However B->D->E->F->D->B can be existed. So when we generate the graph, we should consider this.

The link between vertices can have value 0 or 1.

If the transfer between two vertices is free, then the edge has 1. Otherwise it will have 0.

Here we should add the start vertex manually.

For above example, we can set the nodes of the graph as following.

Nodes

0 🡪 [0 A, 0, 0]

1 🡪[A B B H]

2 🡪[B A B H]

3 🡪[B D R H]

4 🡪[D B R H]

5🡪 [A C R H]

6 🡪 [C A R H]

7🡪 [D E R H]

8🡪[E D R H]

9🡪[E F R B]

10🡪[F E R B]

11🡪[F G B C]

12🡪[G F B C]

13🡪[E G B T]

14🡪[G E B T]

15🡪[D F R B]

16🡪[F D R B]

17🡪[B C R C]

18🡪[C B R C]

19🡪[C F B C]

20🡪[F C B C]

21🡪[C E B T]

22🡪[E C B T]

As we can see, we add the node 0 as a start node. We use the adjacent list to represent graph.

For example, for node 3, there are two adjacent nodes, i.e, node 7 and 15.

For node 4, there is only one adjacent node, node 17.

According to the rule the resulting can be given as following.