Computer Networks
Computer Assignment 3

Amirmohammad Khosravi (810198386)
Adib Rezaei (810198401)

In this project, we implemented routing algorithms LSRP and DVRP using c++.
Below we explain how the program works and what classes we have.

Structs:

We have the structs below. The Route struct is used in the DVRP algorithm and contains destination, next-hop, cost, and path.
The LSP and Spt structs are used in the LSRP algorithm. LSP saves a node and its adjacent nodes. The spt struct saves a node and its distance from a source node.

```
struct Route{
    Router* destination;
    Router* nextHop;
    int cost;
    vector<Router*> path;
};
typedef struct Route Route;
typedef vector<Route> Routes;

struct LSP{
    Router* source;
    map<Router*, int> adjacents;
    vector<Router*> path;
};
typedef struct LSP LSP;
typedef vector<LSP> LSPs;

struct Spt{
    Router* destination;
    int distance;
};
typedef struct Spt Spt;
typedef vector<Spt> SptSet;
```

Classes:

We have a class named Router which is equal to the node class in a graph. It saves a vector of LSPs, a vector of Routes, and an id. The id is given when a Router is created.

```
class Router
{
public:

    Router(int _id) {id = _id;}
    int getId() {return id;}
    void addRoute(Route route) {routes.push_back(route);}
    Routes getRoutes() {return routes;}
    bool exchangeRoutes(Router* router, int weight, Routes _routes);
    void deleteRoutes() {routes.clear();}
    void deleteLsps() {lsps.clear();}
    bool exchangeLsps(LSPs _lsps);
    LSPs getLsps() {return lsps;}
    void addLsp(LSP lsp) {lsps.push_back(lsp);}

private:

    LSPs lsps;
    Routes routes;
    int id;
};
```

Next, we have a class named Link which is equal to the class edge in a graph. It has three methods: router1, router2, and weight which is the weight of the edge (link) between router1 and router2.

```
class Link
{
public:

    Link(Router* r1, Router* r2, int w) {router1 = r1; router2 = r2; weight = w;}
    Router* getFirstRouter() {return router1;}
    Router* getSecondRouter() {return router2;}
    int getWeight() {return weight;}
    void setNewWeight(int _weight) {weight = _weight;}

private:

    Router* router1;
    Router* router2;
    int weight;
};
```

A top class is called network which saves a vector of Routers and a vector of Links, altogether defines a network.

```
class Network
{
public:

    Network(){}
    Router* getRouter(int routerId);
    void addLink(Router* r1, Router* r2, int weight);
    void sortRoutersById();
    vector<int> getRoutersIds();
    int getLinkWeight(int routerId1, int routerId2);
    void doDvrpAlgorithm();
    Routes getDvrpResults(int src);
    void doLsrpAlgorithm();
    Routes getLsrpResults(int src);
    SptSet getSptSet(int src);
    void modifyLink(int routerId1, int routerId2, int weight);
    void removeLink(int routerId1, int routerId2);

private:

    Routers routers;
    Links links;
};
```

We don't have access to network class directly. Instead, we have an Interface class that gets inputs and orders from the command line and decrypt them and then asks the network class to answer the command. After that Interface class shows the results on the command line.

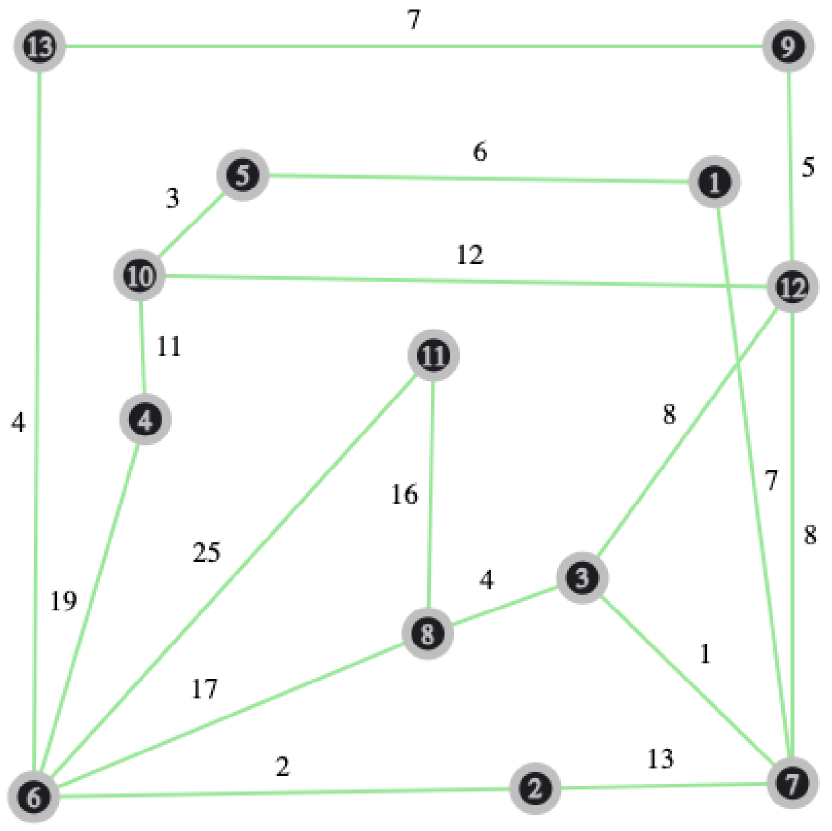
```
class Interface
{
public:

    Interface(Network* n) {network = n; dvrpDone = false; lsrpDone = false;}
    void getOrder();

private:

    void addTopology();
    void showTopology();
    void doLsrpAlgorithm();
    vector<vector<int> > makeGraph();
    void doDvrpAlgorithm();
    void printDvrpResults(int src);
    void printLsrpResults(int src);
    void printLsrpItrResulat(SptSet sptSet);
    void modifyLink();
    void removeLink();

    bool dvrpDone;
    bool lsrpDone;
    Network* network;
};
```

Implementation:

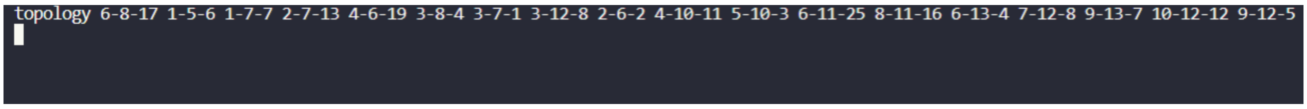
Algorithms are implemented just like what is said in the reference. The DVRP is similar to the Bellman-ford algorithm and the LSRP is close to the Dijkstra algorithm. In DVRP we have a list of Routes and we have Routers to exchange their Routes which each other. For LSRP we first make LSP for each Router and then we make them exchange their LSPs with each other. Then for a single Router, we apply the Dijkstra algorithm to find the smallest path from that Router to each Router (we already have the whole network data in the LSP of each Router).

Outputs:

We get the following network to our program and see what the results will be.



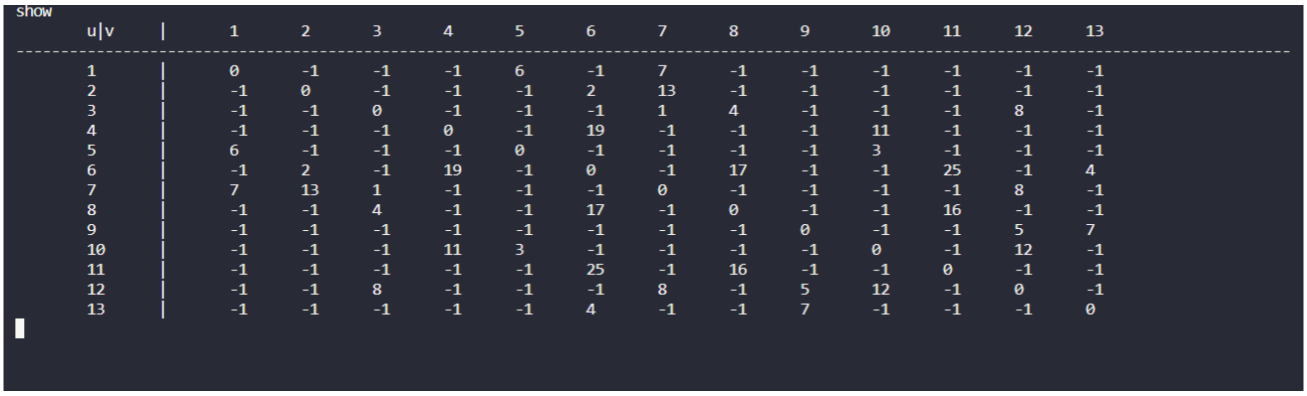
the command to make this network is like below:



```
topology 6-8-17 1-5-6 1-7-7 2-7-13 4-6-19 3-8-4 3-7-1 3-12-8 2-6-2 4-10-11 5-10-3 6-11-25 8-11-16 6-13-4 7-12-8 9-13-7 10-12-12 9-12-5
```

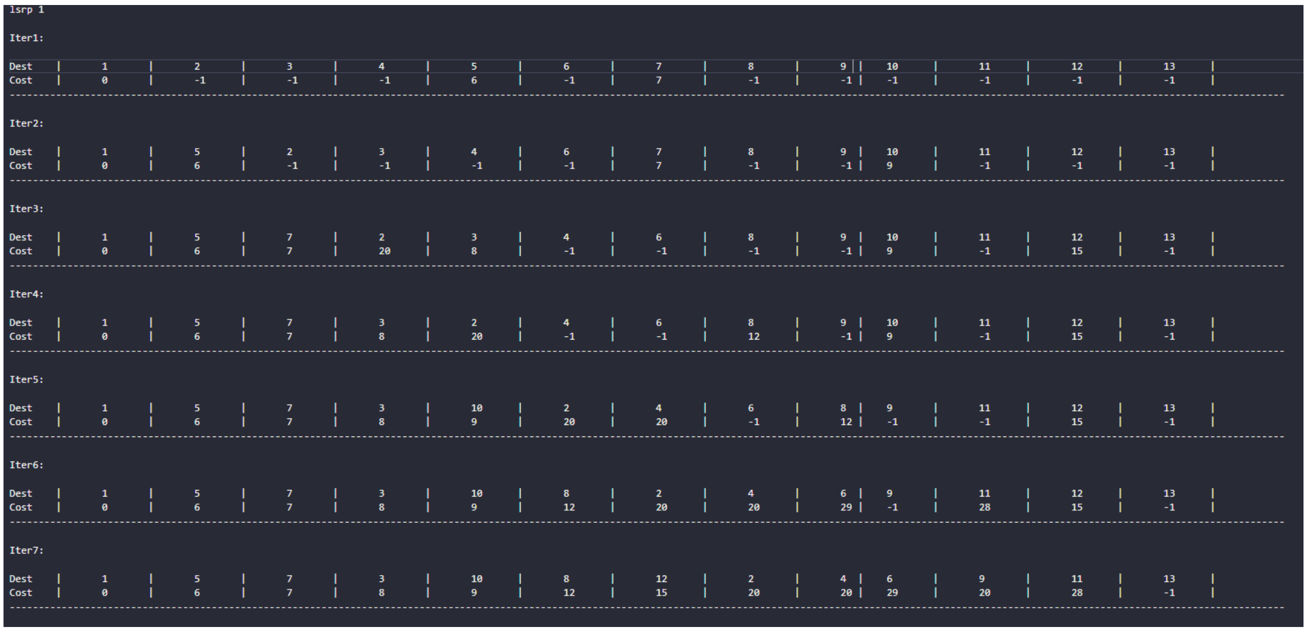
topology 6-8-17 1-5-6 1-7-7 2-7-13 4-6-19 3-8-4 3-7-1 3-12-8 2-6-2 4-10-11 5-10-3 6-11-25 8-11-16 6-13-4 7-12-8 9-13-7 10-12-12 9-12-5

Now we use the show command to show the network. The result will be like below:

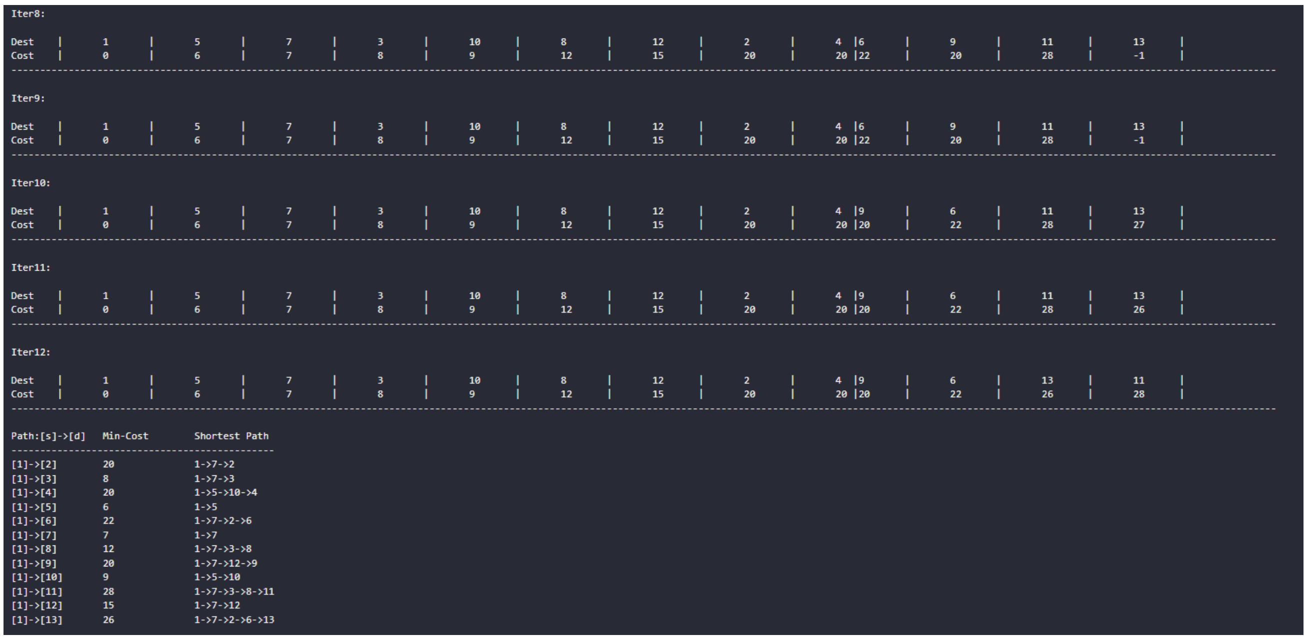


```
show
        u|v    |      1      2      3      4      5      6      7      8      9      10     11     12     13
--------------------------------------------------------------------------------------------------------------
        1      |      0      -1     -1     -1     6      -1     7      -1     -1     -1     -1     -1     -1
        2      |      -1     0      -1     -1     -1     2      13     -1     -1     -1     -1     -1     -1
        3      |      -1     -1     0      -1     -1     -1     1      4      -1     -1     -1     8      -1
        4      |      -1     -1     -1     0      -1     19     -1     -1     -1     11     -1     -1     -1
        5      |      6      -1     -1     -1     0      -1     -1     -1     -1     3      -1     -1     -1
        6      |      -1     2      -1     19     -1     0      -1     17     -1     -1     25     -1     4
        7      |      7      13     1      -1     -1     -1     0      -1     -1     -1     -1     8      -1
        8      |      -1     -1     4      -1     -1     17     -1     0      -1     -1     16     -1     -1
        9      |      -1     -1     -1     -1     -1     -1     -1     -1     0      -1     -1     5      7
        10     |      -1     -1     -1     11     3      -1     -1     -1     -1     0      -1     12     -1
        11     |      -1     -1     -1     -1     -1     25     -1     16     -1     -1     0      -1     -1
        12     |      -1     -1     8      -1     -1     -1     8      -1     5      12     -1     0      -1
        13     |      -1     -1     -1     -1     -1     4      -1     -1     7      -1     -1     -1     0
```

Now we use the lsrp command to apply the LSRP routing algorithm (first we use lsrp 1 to see if it works. we can use lsrp without arguments to show the result for all nodes):

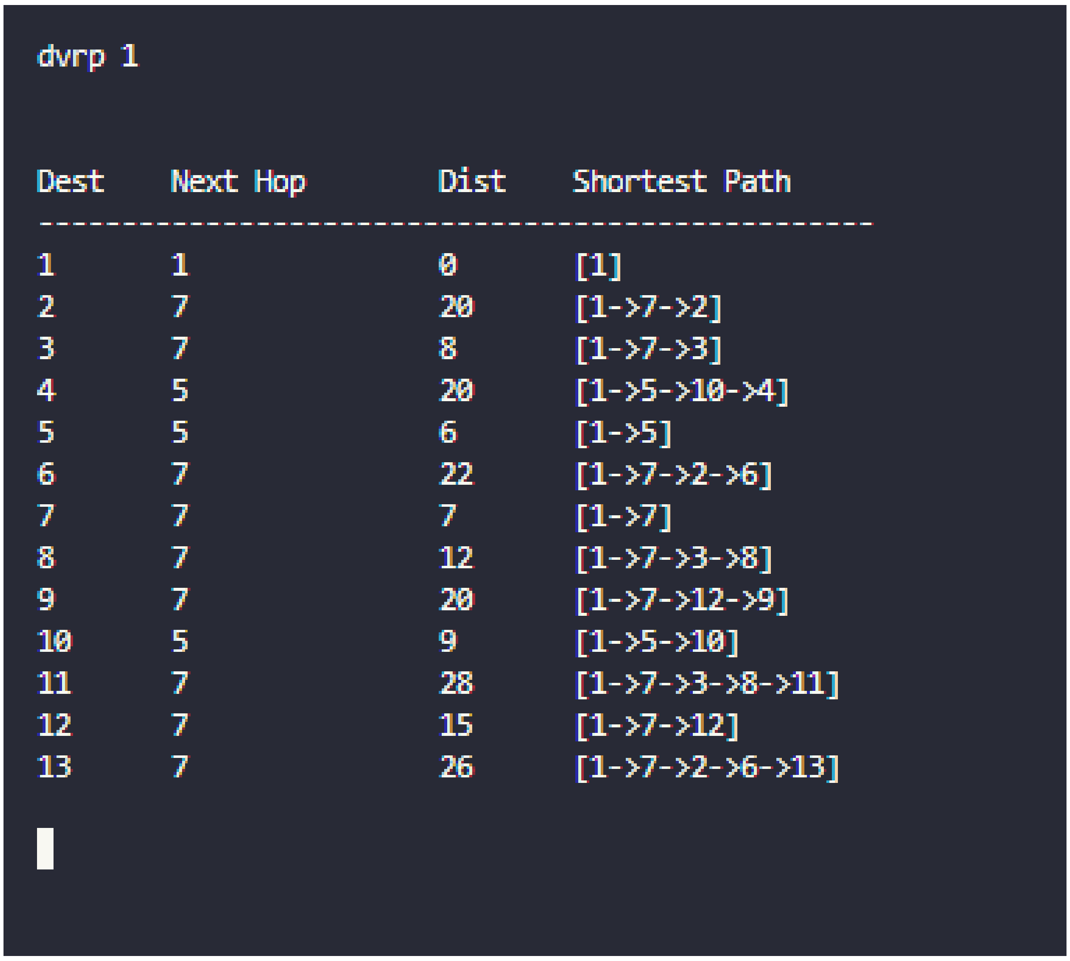


```
lsrp 1

Iter1:

Dest  |    1    |    2    |    3    |    4    |    5    |    6    |    7    |    8    |    9 ||   10    |    11   |    12   |    13   |
Cost  |    0    |   -1    |   -1    |   -1    |    6    |   -1    |    7    |   -1    |   -1 |   -1    |    -1   |    -1   |    -1   |
------------------------------------------------------------------------------------------------------------------------------------------

Iter2:

Dest  |    1    |    5    |    2    |    3    |    4    |    6    |    7    |    8    |    9 |   10    |    11   |    12   |    13   |
Cost  |    0    |    6    |   -1    |   -1    |   -1    |   -1    |    7    |   -1    |   -1 |    9    |    -1   |    -1   |    -1   |
------------------------------------------------------------------------------------------------------------------------------------------

Iter3:

Dest  |    1    |    5    |    7    |    2    |    3    |    4    |    6    |    8    |    9 |   10    |    11   |    12   |    13   |
Cost  |    0    |    6    |    7    |   20    |    8    |   -1    |   -1    |   -1    |   -1 |    9    |    -1   |    15   |    -1   |
------------------------------------------------------------------------------------------------------------------------------------------

Iter4:

Dest  |    1    |    5    |    7    |    3    |    2    |    4    |    6    |    8    |    9 |   10    |    11   |    12   |    13   |
Cost  |    0    |    6    |    7    |    8    |   20    |   -1    |   -1    |   12    |   -1 |    9    |    -1   |    15   |    -1   |
------------------------------------------------------------------------------------------------------------------------------------------

Iter5:

Dest  |    1    |    5    |    7    |    3    |   10    |    2    |    4    |    6    |    8 |    9    |    11   |    12   |    13   |
Cost  |    0    |    6    |    7    |    8    |    9    |   20    |   20    |   -1    |   12 |   -1    |    -1   |    15   |    -1   |
------------------------------------------------------------------------------------------------------------------------------------------

Iter6:

Dest  |    1    |    5    |    7    |    3    |   10    |    8    |    2    |    4    |    6 |    9    |    11   |    12   |    13   |
Cost  |    0    |    6    |    7    |    8    |    9    |   12    |   20    |   20    |   29 |   -1    |    28   |    15   |    -1   |
------------------------------------------------------------------------------------------------------------------------------------------

Iter7:

Dest  |    1    |    5    |    7    |    3    |   10    |    8    |   12    |    2    |    4 |    6    |    9    |    11   |    13   |
Cost  |    0    |    6    |    7    |    8    |    9    |   12    |   15    |   20    |   20 |   29    |    20   |    28   |    -1   |
------------------------------------------------------------------------------------------------------------------------------------------
```



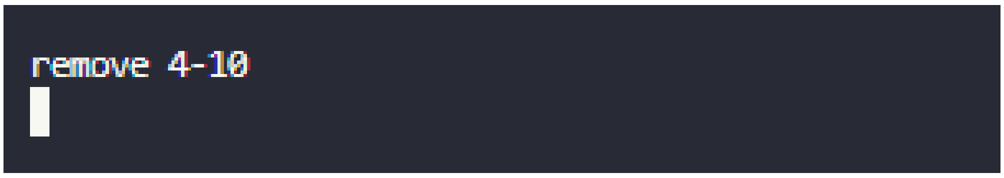
```
Iter8:

Dest    |       1       |       5       |       7       |       3       |       10      |       8       |       12      |       2       |       4  |6       |       9       |       11      |       13      |
Cost    |       0       |       6       |       7       |       8       |       9       |       12      |       15      |       20      |       20 |22      |       20      |       28      |       -1      |
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Iter9:

Dest    |       1       |       5       |       7       |       3       |       10      |       8       |       12      |       2       |       4  |6       |       9       |       11      |       13      |
Cost    |       0       |       6       |       7       |       8       |       9       |       12      |       15      |       20      |       20 |22      |       20      |       28      |       -1      |
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Iter10:

Dest    |       1       |       5       |       7       |       3       |       10      |       8       |       12      |       2       |       4  |9       |       6       |       11      |       13      |
Cost    |       0       |       6       |       7       |       8       |       9       |       12      |       15      |       20      |       20 |20      |       22      |       28      |       27      |
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Iter11:

Dest    |       1       |       5       |       7       |       3       |       10      |       8       |       12      |       2       |       4  |9       |       6       |       11      |       13      |
Cost    |       0       |       6       |       7       |       8       |       9       |       12      |       15      |       20      |       20 |20      |       22      |       28      |       26      |
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Iter12:

Dest    |       1       |       5       |       7       |       3       |       10      |       8       |       12      |       2       |       4  |9       |       6       |       13      |       11      |
Cost    |       0       |       6       |       7       |       8       |       9       |       12      |       15      |       20      |       20 |20      |       22      |       26      |       28      |
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Path:[s]->[d]   Min-Cost        Shortest Path
---------------------------------------------
[1]->[2]        20              1->7->2
[1]->[3]        8               1->7->3
[1]->[4]        20              1->5->10->4
[1]->[5]        6               1->5
[1]->[6]        22              1->7->2->6
[1]->[7]        7               1->7
[1]->[8]        12              1->7->3->8
[1]->[9]        20              1->7->12->9
[1]->[10]       9               1->5->10
[1]->[11]       28              1->7->3->8->11
[1]->[12]       15              1->7->12
[1]->[13]       26              1->7->2->6->13
```

Now we use the dvrp command to apply the DVRP algorithm (we used dvrp 1. We can use dvrp without arguments to show the result for all Routers):



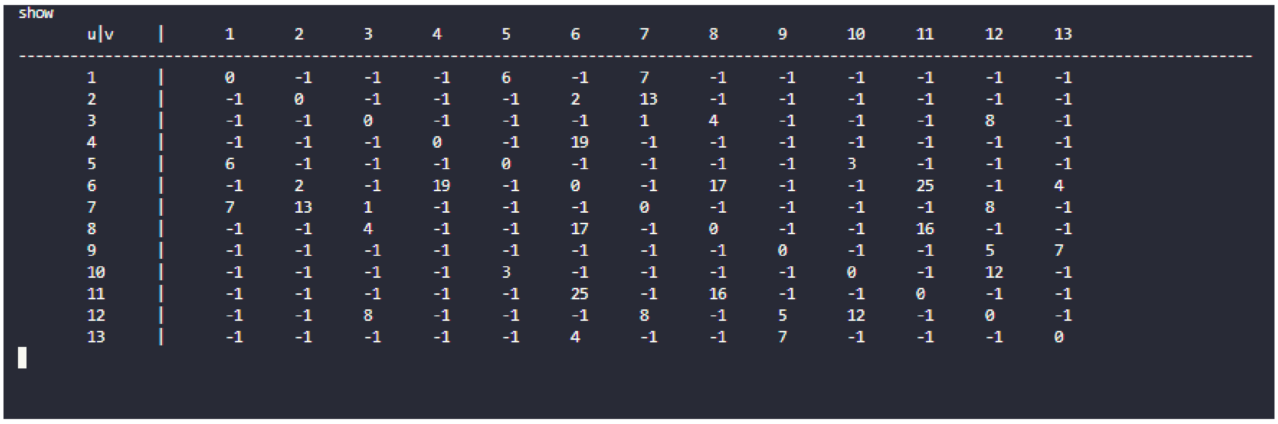
```
dvrp 1


Dest    Next Hop        Dist    Shortest Path
-------------------------------------------------
1       1               0       [1]
2       7               20      [1->7->2]
3       7               8       [1->7->3]
4       5               20      [1->5->10->4]
5       5               6       [1->5]
6       7               22      [1->7->2->6]
7       7               7       [1->7]
8       7               12      [1->7->3->8]
9       7               20      [1->7->12->9]
10      5               9       [1->5->10]
11      7               28      [1->7->3->8->11]
12      7               15      [1->7->12]
13      7               26      [1->7->2->6->13]
```

Now we delete the link between routers 4 and 10:



```
remove 4-10
```

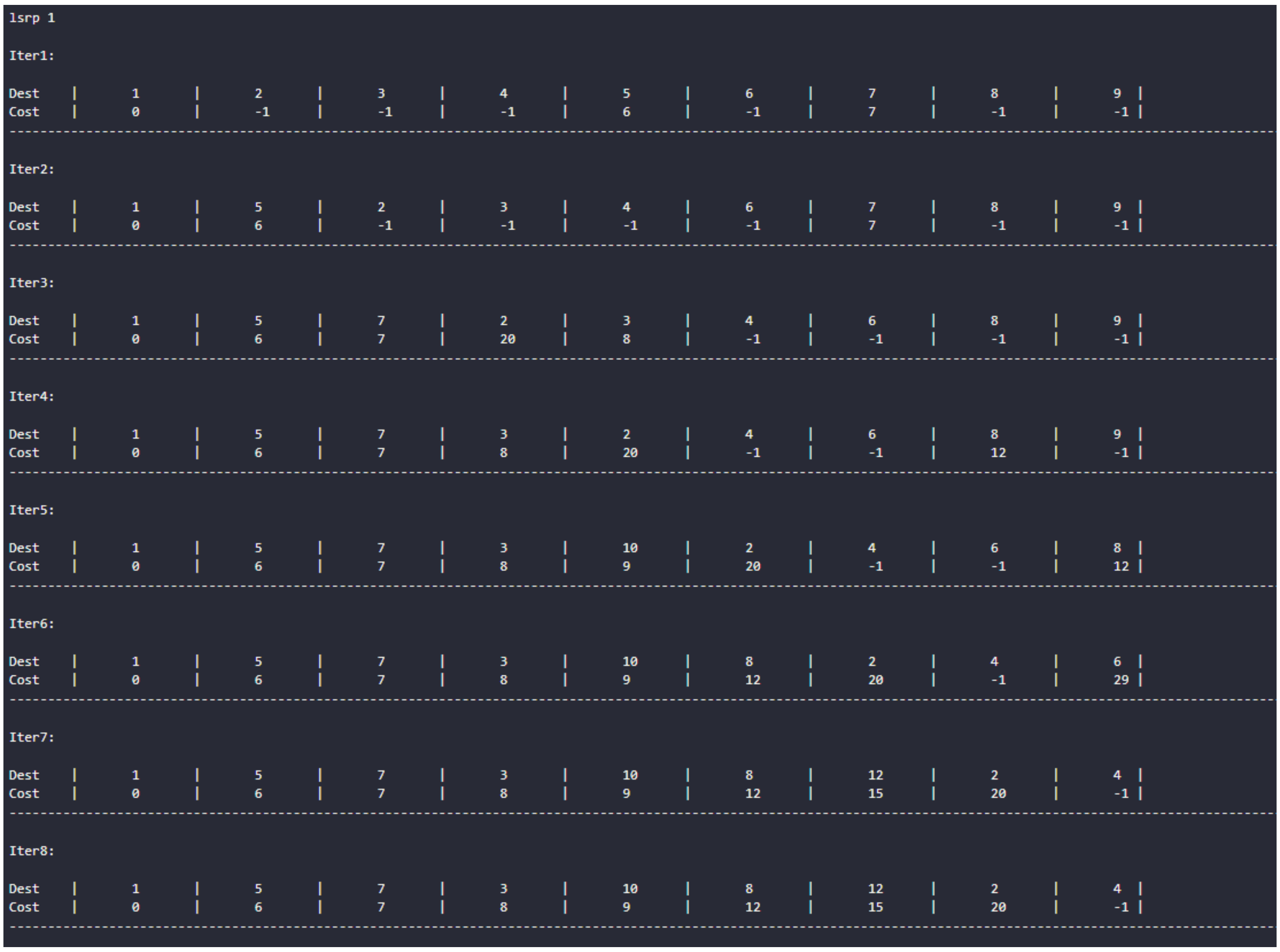
The network will be like:



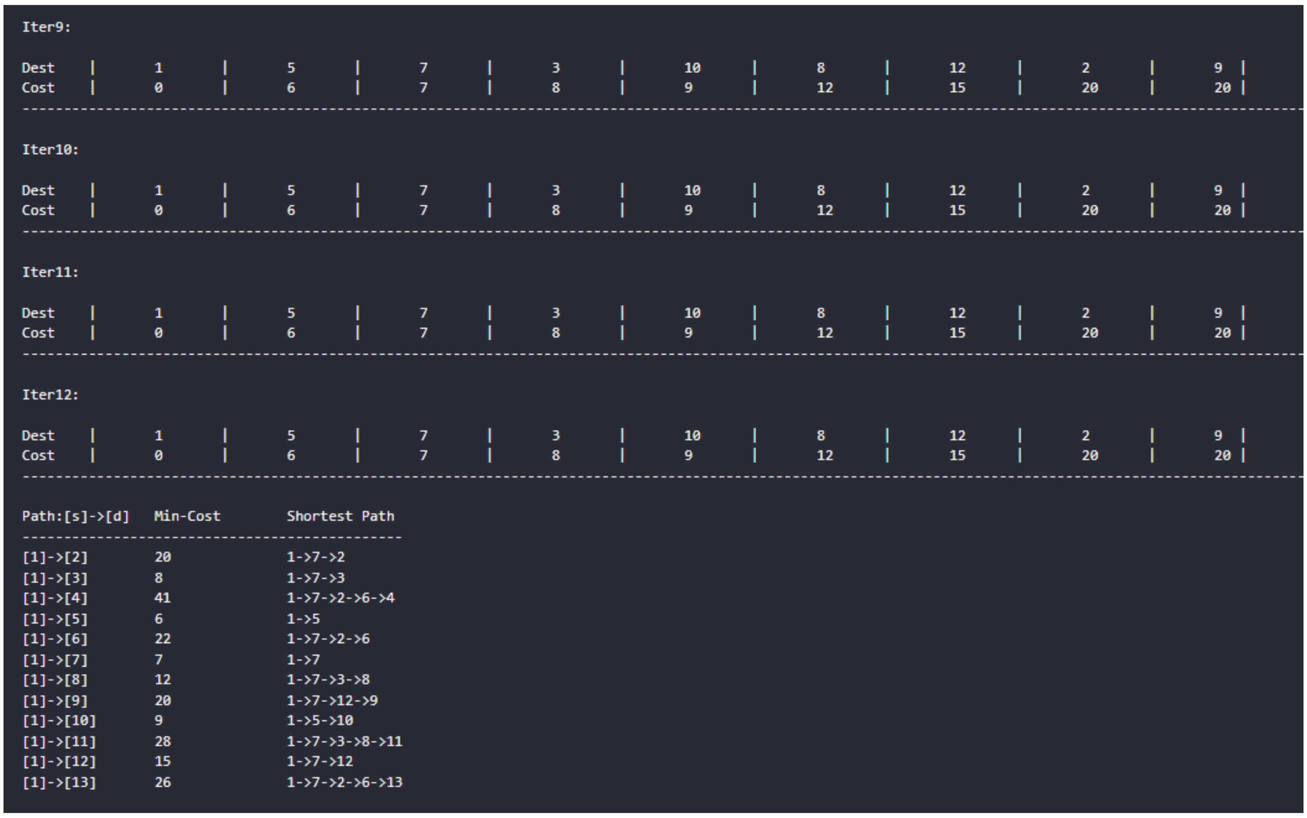
```
show
        u|v     |       1       2       3       4       5       6       7       8       9       10      11      12      13
--------------------------------------------------------------------------------------------------------------------------------
        1       |       0       -1      -1      -1      6       -1      7       -1      -1      -1      -1      -1      -1
        2       |       -1      0       -1      -1      -1      2       13      -1      -1      -1      -1      -1      -1
        3       |       -1      -1      0       -1      -1      -1      1       4       -1      -1      -1      8       -1
        4       |       -1      -1      -1      0       -1      19      -1      -1      -1      -1      -1      -1      -1
        5       |       6       -1      -1      -1      0       -1      -1      -1      -1      3       -1      -1      -1
        6       |       -1      2       -1      19      -1      0       -1      17      -1      -1      25      -1      4
        7       |       7       13      1       -1      -1      -1      0       -1      -1      -1      -1      8       -1
        8       |       -1      -1      4       -1      -1      17      -1      0       -1      -1      16      -1      -1
        9       |       -1      -1      -1      -1      -1      -1      -1      -1      0       -1      -1      5       7
        10      |       -1      -1      -1      -1      3       -1      -1      -1      -1      0       -1      12      -1
        11      |       -1      -1      -1      -1      -1      25      -1      16      -1      -1      0       -1      -1
        12      |       -1      -1      8       -1      -1      -1      8       -1      5       12      -1      0       -1
        13      |       -1      -1      -1      -1      -1      4       -1      -1      7       -1      -1      -1      0
```

Now we run the algorithms again to see the results:

LSRP (lsrp command):

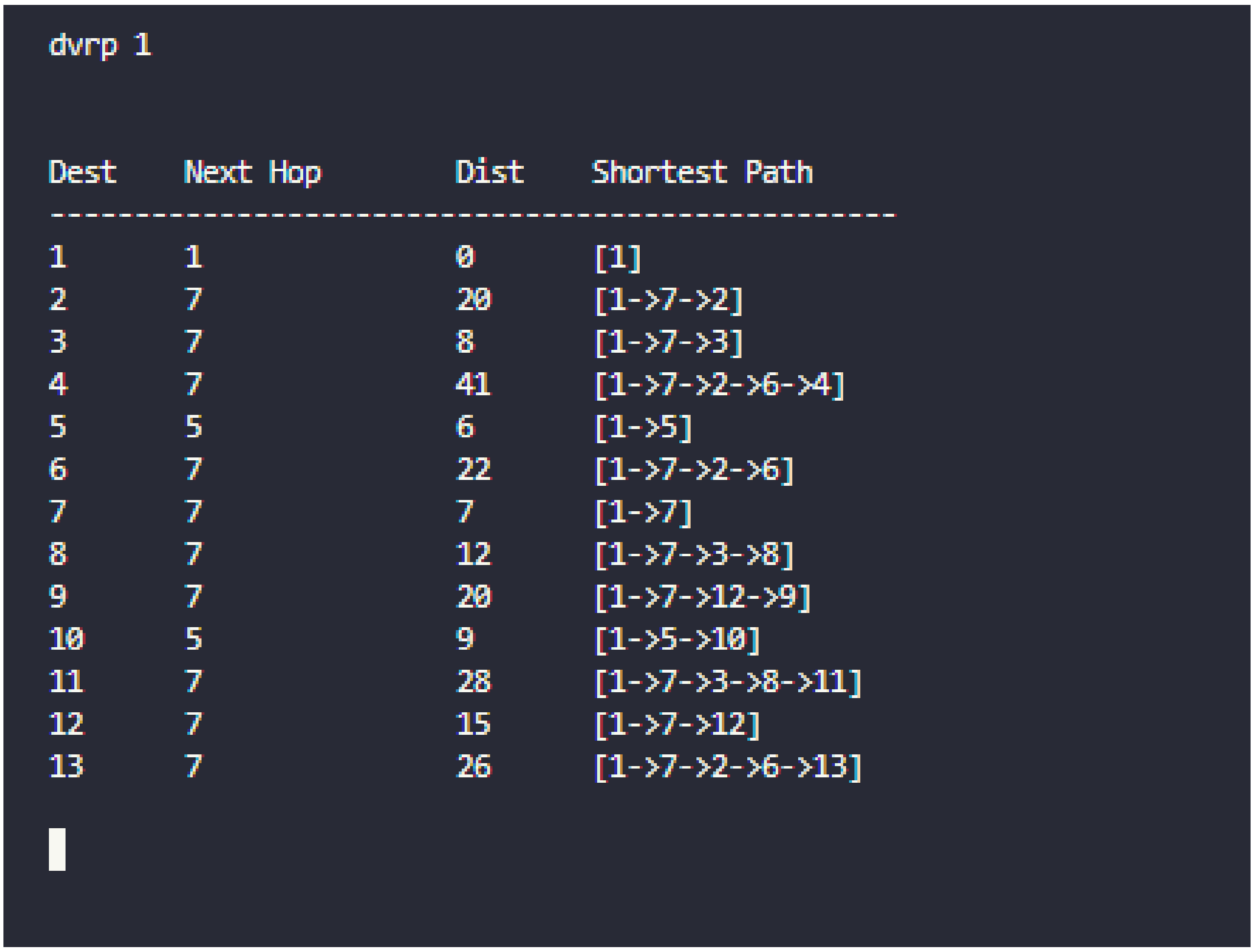


```
lsrp 1

Iter1:

Dest  |    1    |    2    |    3    |    4    |    5    |    6    |    7    |    8    |    9  |
Cost  |    0    |   -1    |   -1    |   -1    |    6    |   -1    |    7    |   -1    |   -1  |
------------------------------------------------------------------------------------------------

Iter2:

Dest  |    1    |    5    |    2    |    3    |    4    |    6    |    7    |    8    |    9  |
Cost  |    0    |    6    |   -1    |   -1    |   -1    |   -1    |    7    |   -1    |   -1  |
------------------------------------------------------------------------------------------------

Iter3:

Dest  |    1    |    5    |    7    |    2    |    3    |    4    |    6    |    8    |    9  |
Cost  |    0    |    6    |    7    |   20    |    8    |   -1    |   -1    |   -1    |   -1  |
------------------------------------------------------------------------------------------------

Iter4:

Dest  |    1    |    5    |    7    |    3    |    2    |    4    |    6    |    8    |    9  |
Cost  |    0    |    6    |    7    |    8    |   20    |   -1    |   -1    |   12    |   -1  |
------------------------------------------------------------------------------------------------

Iter5:

Dest  |    1    |    5    |    7    |    3    |   10    |    2    |    4    |    6    |    8  |
Cost  |    0    |    6    |    7    |    8    |    9    |   20    |   -1    |   -1    |   12  |
------------------------------------------------------------------------------------------------

Iter6:

Dest  |    1    |    5    |    7    |    3    |   10    |    8    |    2    |    4    |    6  |
Cost  |    0    |    6    |    7    |    8    |    9    |   12    |   20    |   -1    |   29  |
------------------------------------------------------------------------------------------------

Iter7:

Dest  |    1    |    5    |    7    |    3    |   10    |    8    |   12    |    2    |    4  |
Cost  |    0    |    6    |    7    |    8    |    9    |   12    |   15    |   20    |   -1  |
------------------------------------------------------------------------------------------------

Iter8:

Dest  |    1    |    5    |    7    |    3    |   10    |    8    |   12    |    2    |    4  |
Cost  |    0    |    6    |    7    |    8    |    9    |   12    |   15    |   20    |   -1  |
------------------------------------------------------------------------------------------------
```



```
Iter9:

Dest  |    1    |    5    |    7    |    3    |   10    |    8    |   12    |    2    |    9  |
Cost  |    0    |    6    |    7    |    8    |    9    |   12    |   15    |   20    |   20  |
------------------------------------------------------------------------------------------------

Iter10:

Dest  |    1    |    5    |    7    |    3    |   10    |    8    |   12    |    2    |    9  |
Cost  |    0    |    6    |    7    |    8    |    9    |   12    |   15    |   20    |   20  |
------------------------------------------------------------------------------------------------

Iter11:

Dest  |    1    |    5    |    7    |    3    |   10    |    8    |   12    |    2    |    9  |
Cost  |    0    |    6    |    7    |    8    |    9    |   12    |   15    |   20    |   20  |
------------------------------------------------------------------------------------------------

Iter12:

Dest  |    1    |    5    |    7    |    3    |   10    |    8    |   12    |    2    |    9  |
Cost  |    0    |    6    |    7    |    8    |    9    |   12    |   15    |   20    |   20  |
------------------------------------------------------------------------------------------------

Path:[s]->[d]   Min-Cost        Shortest Path
------------------------------------------------
[1]->[2]        20              1->7->2
[1]->[3]        8               1->7->3
[1]->[4]        41              1->7->2->6->4
[1]->[5]        6               1->5
[1]->[6]        22              1->7->2->6
[1]->[7]        7               1->7
[1]->[8]        12              1->7->3->8
[1]->[9]        20              1->7->12->9
[1]->[10]       9               1->5->10
[1]->[11]       28              1->7->3->8->11
[1]->[12]       15              1->7->12
[1]->[13]       26              1->7->2->6->13
```

Now DVRP (dvrp command):



```
dvrp 1


Dest    Next Hop        Dist    Shortest Path
--------------------------------------------------
1       1               0       [1]
2       7               20      [1->7->2]
3       7               8       [1->7->3]
4       7               41      [1->7->2->6->4]
5       5               6       [1->5]
6       7               22      [1->7->2->6]
7       7               7       [1->7]
8       7               12      [1->7->3->8]
9       7               20      [1->7->12->9]
10      5               9       [1->5->10]
11      7               28      [1->7->3->8->11]
12      7               15      [1->7->12]
13      7               26      [1->7->2->6->13]
```

According to the results, the LSRP algorithm is much faster.
After removing 4-10, the convergent increase because data needs more time and effort to propagate.