Exercise 3. RIP and OSPF

1. ***Demo.*** Find the routes in Figure 4 from node A to all the other nodes in the network using Dijkstra's algorithm. You may use the animation of Dijkstra's algorithm at http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/dijex1.html

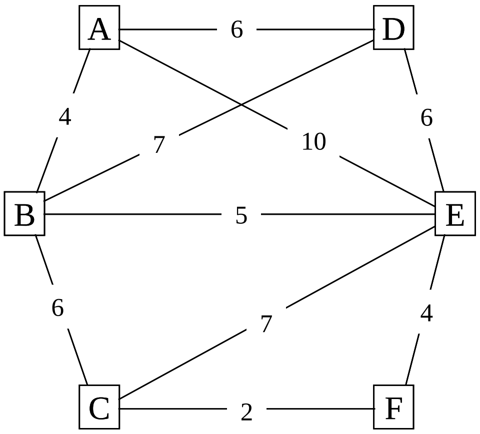

Figure 13. Network 1

Solution

We use the java applet at the given address. The input file of the applet consists of a line indicating the number of nodes and links, a list of nodes and a list of links. The line describing a node contains the name of the node, the x-coordinate and the y-coordinate. The links are described by the name, the start node, the end node and the cost. The above network is defined by the following file:

```
6 11 graph
"a"  80 50
"b"  50 150
"c"  80 250
"d" 220 50
"e" 250 150
"f" 220 250
"" a d 6
"" a b 4
"" a e 10
"" b d 7
"" b e 5
"" b d 7
"" b c 6
"" c f 2
"" e f 4
"" c e 7
"" d e 6
```

By clicking at the applet window, we can follow how the algorithm progresses. The main steps are showed in Figure 14.

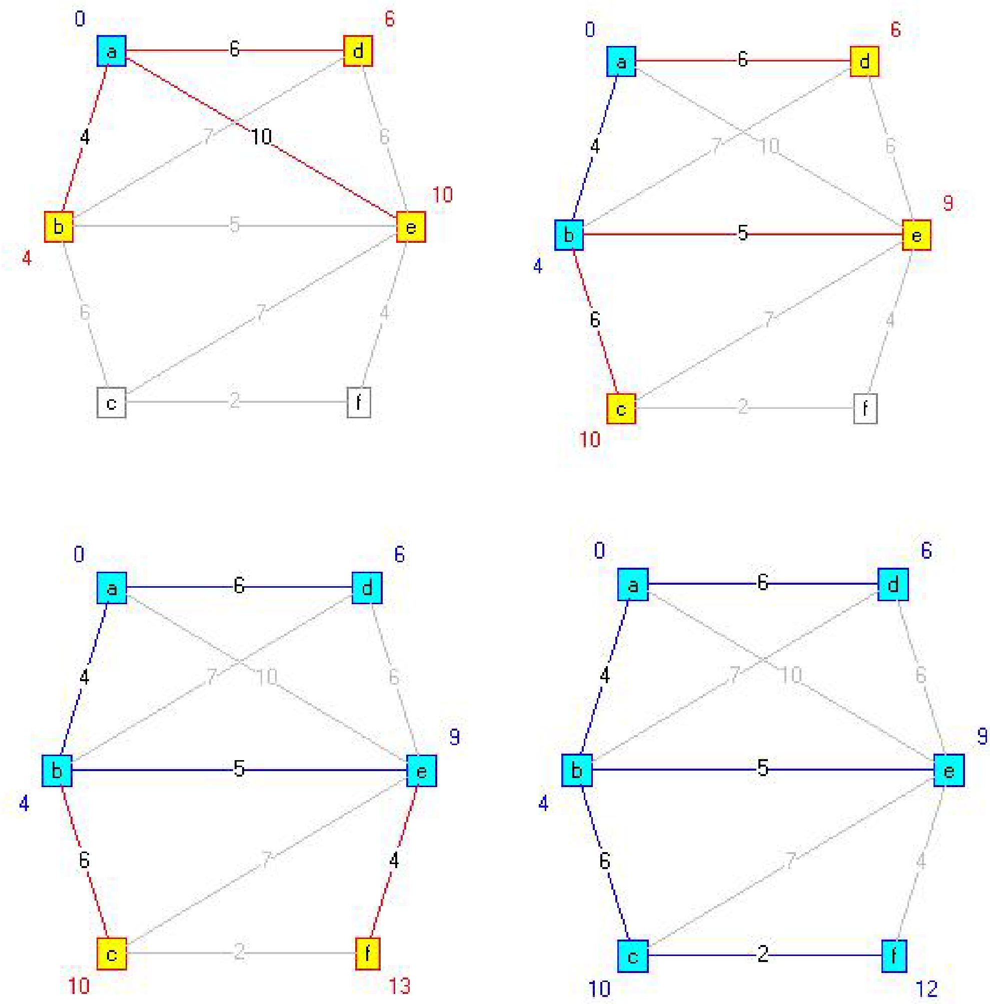

Figure 14. Dijkstra's algorithm for Network 1

2. Describe the differences between RIP and OSPF. Consider such aspects as routing algorithm, functionality, scalability, capability, complexity, stability, loop avoidance and so on.

Solution

Routing algorithms

RIP belongs to the distance vector family in which the routing table information (i.e. distance vector) is advocated; OSPF belongs to the link state family in which only link state information is advocated. To calculate routes, RIP runs Bellman-Ford algorithm based on the routing table information; OSPF runs Dijkstra algorithm based on topology information.

Functionality and capability

On the other hand, both algorithms compute the shortest path in the distributed way. Therefore, both generally produce the same routing results. In addition, RIP-2 supports authentication and multicasting, while OSPF supports multiple metrics, multiple areas, external routes, etc. OSPF consists of the *Hello* protocol, *Exchange* protocol and *Flooding* protocol, which are responsible for maintaining the link state database.

Scalability
RIP is suitable for relatively small networks with simple network topology and rare failures. OSPF is much more scalable for complex and relatively large networks. This is because OSPF uses link state and supports external routes, multiple areas, etc.

Stability
Generally, RIP can keep stable if the network topology is relatively simple and if link failures are rare, but for large and complex networks RIP is quite unstable. It computes new routes after any change in the network topology, but in some cases very slowly by counting to infinity. During the time it takes to perform the computation, the network is left in a transient state where loops may occur and cause temporary congestion. OSPF can keep stable even in relatively large and complex networks.

Complexity
OSPF is more complex than RIP. RIP has only two messages; OSPF needs five different messages and three procedures. RIP needs only one routing table while OSPF needs to maintain both a link state database and a routing table.

Loop avoidance
In RIP, a loop can be detected when counting the distance to infinity. In OSPF, a loop can be found and removed in principle after all link state databases become consistent.

Others
RIP is over UDP while OSPF is over IP.

Table 3. Differences between RIP and OSPF

	RIP	OSPF
Classification	Distance vector	Link state
Routing algorithm	Bellman-Ford	Dijkstra
Route computation	Shortest path	Shortest path
Supports	Authentication, multicasting, etc	Multiple metrics, multiple areas, external routes, etc
Scalability	Small network	Large network
Stability	Stable in small networks	Stable even in large networks
Complexity	Simple	Complex
Loop avoidance	Counting to infinity	Keeping consistent database

3. Describe how the Dijsktra and Bellman-Ford algorithms work. You may use pseudo-code if you wish. How do the two algorithms differ from each other?

Bellman-Ford algorithm

Find the shortest paths from the given node so that the path contains only one link; continue searching with the path containing two links etc. To put the algorithm in a formal way:

Definitions:
- s = source node
- d_{ij} = cost of the link from node i to node j; $d_{ii} = 0$, $d_{ij} = \infty$ if the nodes are not directly connected and $d_{ij} \geq 0$, if they are
- h = maximum number of links in the particular stage of the algorithm
- $D_n^{(h)}$ = the value of the least cost from node s to node n, when there are h links maximum in the path

The actual algorithm contains two steps and step number two is repeated until none of the costs changes:

1. Initialize:
   $D_n^{(0)} = \infty$, for all $n \neq s$
   $D_s^{(h)} = 0$, for all h

2. For all $h \geq 0$:
   $D_n^{(h+1)} = \min_j [D_j^{(h)} + d_{jn}]$
   The path from node s to node i ends with a link from node j to node i.

The Bellman-Ford algorithm is a typical distance vector algorithm. These algorithms call for each neighboring router to send all or some portion of its routing table.

Dijkstra algorithm

Find the shortest paths from the given node to all nodes by developing paths according to the increasing path length. The algorithm advances in steps: by the kth step the least cost paths to k nodes have been found. These nodes form the set M. Taking the step k+1, the node which does not belong to set M, but to which you can find the least cost path is added to M. When nodes are added to M the shortest paths are defined. The algorithm can be described in a formal way as follows:

Definitions:
- N = the set of nodes in the network
- s = source node
- M = the set of nodes that have been included in the path
- d_{ij} = the cost of the link from node i to node j; $d_{ii} = 0$, $d_{ij} = \infty$ if the nodes are not directly connected and $d_{ij} \geq 0$, if they are
- D_n = the value of the least cost path from node s to node n

The algorithm has three steps and step number two and three are repeated until $M = N$. Then the final paths have been defined to all nodes in the network.

1. Initialize:
   $M = \{s\}$, source node $D_n = d_{sn}$, to all $n \neq s$, these are the costs to the neighboring nodes.

2. Find the neighboring node w, which does not belong to M, and which has the least cost path from s and add this node w to M. Or in other words:
   Find $w \in M$, so that $D_w = \min_{j \in M} D_j$, add w to M.

3. Update the least cost paths:
   $D_n = \min[\, D_n, D_w + d_{wn} \,]$ for all $n \notin M$
   If the latter term is the minimum, the path from node s to node n is now the path from s to w added with a path from w to n.

The Dijkstra algorithm is a typical link state algorithm. It floods routing information to all nodes in the network. Each router sends only the portion of the routing table that describes the state of its own links.

Pay particular care to the information that the algorithms need to gather. In essence, link state algorithms (Dijkstra) send small updates everywhere while distance vector algorithms (Bellman-Ford) send larger updates only to neighboring routers.

The step 2 in Bellman-Fords needs information about the cost to all neighboring nodes to determine node n. In addition, the Bellman-Ford algorithm needs to know the total cost from node s to all neighboring nodes. Now every node is able to uphold information about costs and paths to other nodes in the network and it can easily exchange this information with the neighboring nodes. Thus, all nodes can utilize the step 2 in the Bellman-Ford algorithm based on information of the neighboring nodes and the costs to these nodes.

On the other hand, the third step in the Dijkstra algorithm would seem to require knowledge of the topology of the whole network. Thus, every node would have to know the costs of all connection and therefore information would have to be exchanged between all nodes.

4. Use the Dijkstra and Bellman-Ford algorithms and show how they find the routes in Figure 5 from node A to all the other nodes in the network. Return the final routing table and show how it developed there. You may use the animation of Dijkstra's algorithm at http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/dijex8.html

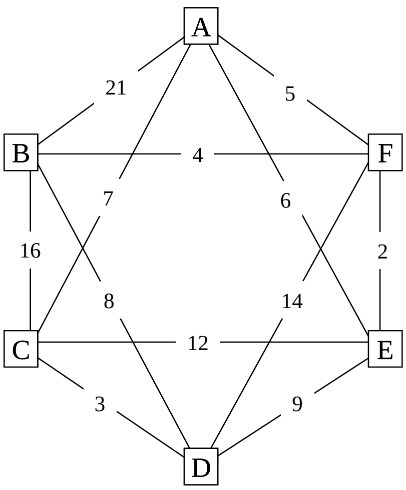

Figure 15. Network 2

Solution

Dijkstra

The most important steps of the animation are shown in Figure 16. These correspond to the first, second, third and final steps in the development of the routing table.

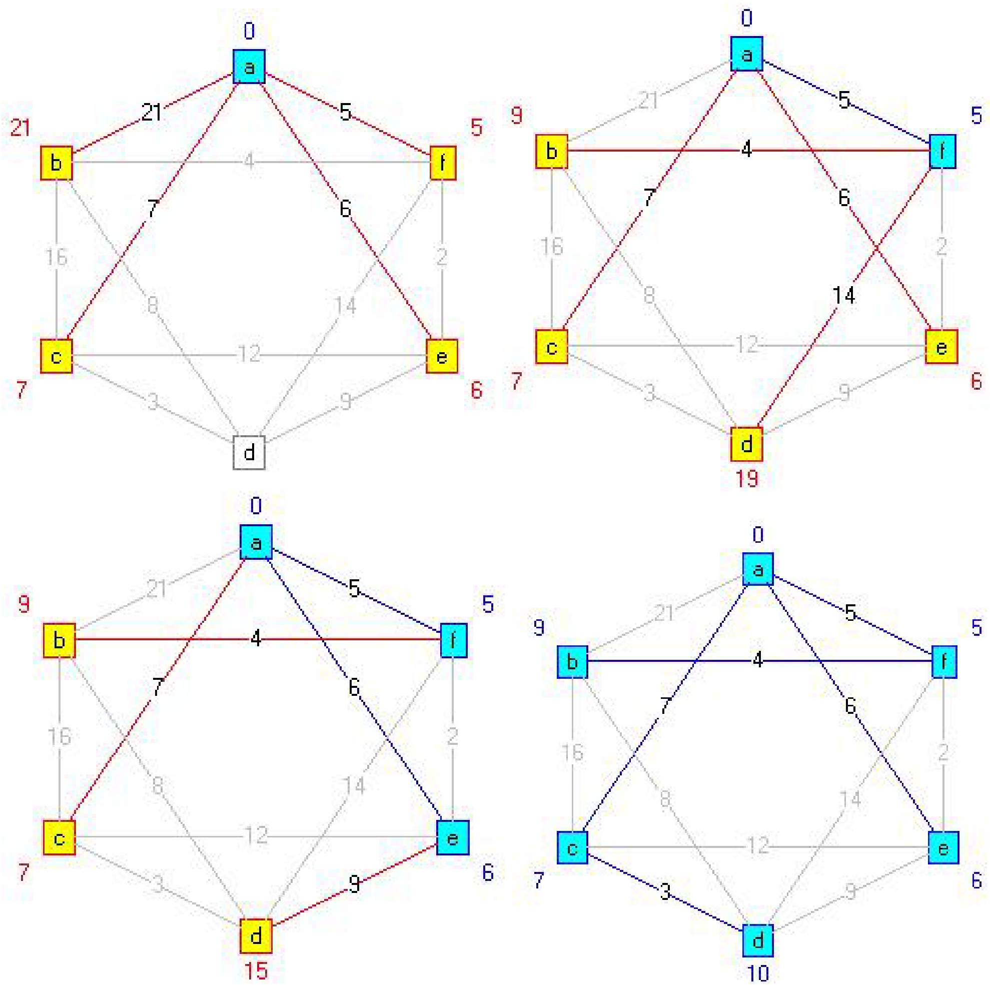

Figure 16. Dijkstra's algorithm for Network 2

The development of the routing table is shown in Table 4. The last row represents the final routing table.

Table 4. The development of the routing table of Network 2 with Dijkstra algorithm

M	D_b *path*	D_c *path*	D_d *path*	D_e *path*	D_f *path*
1 a	21 a-b	7 a-c	∞	6 a-e	5 a-f
2 a,f	9 a-f-b	7 a-c	19 a-f-d	6 a-e	5 a-f
3 a,f,e	9 a-f-b	7 a-c	15 a-e-d	6 a-e	5 a-f
4 a,f,e,c	9 a-f-b	7 a-c	10 a-c-d	6 a-e	5 a-f
5 a,f,e,c,b	9 a-f-b	7 a-c	10 a-c-d	6 a-e	5 a-f
6 a,f,e,c,b,d	9 a-f-b	7 a-c	10 a-c-d	6 a-e	5 a-f

Bellman-Ford

The development of the routing table is shown in Table 5. The last row represents the final routing table.

Table 5. The development of the routing table of Network 2 with Bellman-Ford algorithm

H	D_b^h *path*	D_c^h *path*	D_d^h *path*	D_e^h *path*	D_f^h *path*
0	∞ -	∞ -	∞ -	∞ -	∞ -
1	21 a-b	7 a-c	∞ -	6 a-e	5 a-f
2	9 a-f-b	7 a-c	10 a-c-d	6 a-e	5 a-f