1 Overview

Facility network provides a modeling method for common facilities such as municipal water network, transmission lines, gas pipelines, telecommunications networks, water system, etc. in our real world. All the networks' resources are flowing directionally, which can be modeled and analyzed with the facility network.

The facility network analysis is one of the common functions of network analysis. It mainly processes a variety of connectivity analyses and tracing analyses.

The facility network analysis must follow some rules to build the network models. It must have the flow direction, which is the flow direction of the substance. The network direction depends on the network topology, and the locations of the source and sink.

2 Basic Concepts

A facility network is composed of a group of edges and junctions. It can be used to express and model the real-world network facilities according to the specified connectivity rules. Users can specify the meaning and rules of the basic elements (points/lines) that compose the facility network to specify how the resources flow in the facility network.

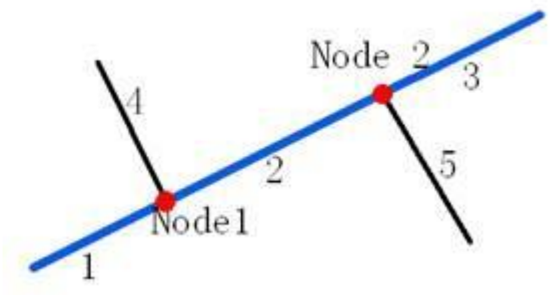

Figure 2-1 Facility network diagram

The figure 2-1 is a simple network diagram. The thick blue line is the main pipeline composed of segments 1, 2, 3; the thin black segments 4 and 5 are the branch pipeline; the red junctions Node 1 and Node 2 are the connected points of the main pipeline and branch pipeline.

Edges: The established facility network generally includes the line objects, which are used to denote the flowing resource pipelines such as water pipelines, electric wires, natural gas pipelines, etc.

Simple Edges: The edge that can only connect two junctions with its two endpoints, such as the branch pipelines 4 and 5 in the figure 2-1 . The simple edges don't have the internal connectivity. If you need to add a new junction on it, then it must be split into two simple edges.

Complex Edges: Besides the two junctions at the end, the complex edge also can be added junctions in its middle. As shown in the figure 2-1, the blue main pipeline connects the two branch pipelines at the junctions Node 1 and Node 22, but it is not really be split, i.e., you can add as many junctions as you can.

Note: Universal GIS Core (UGC) currently doesn't have the complex edges, i.e., all the complex edges will be split physically into simple edges.

Junctions: The facility network generally includes the point objects, which denotes the junctions of two or more flowing resource pipeline, such as the pumping stations and valves of the water network, electric gates of the power grid, and supply points of the natural gas network.

User defined junctions: Junctions defined by users in facility network building. These junctions correspond to single junction point elements of logical network.

Orphan junctions: While adding an edge to facility network, the start node and end node of the edge will be automatically added as orphan junctions to maintain the connectivity of the network. To delete these orphan junctions, they should be included in the user defined junctions; otherwise, the connectivity of the network will be damaged.

Connectivity rules: In most facilities, users will not want all types of edges to connect with all types of junctions or edges logically. For instance, a junction of the fire hydrant type can only be connected to edges of the branch line type and a trunk line with the diameter of 0.8 meters can only be connected with a branch line with the diameter of 0.5 meters through a junction of the compression release valve. Therefore, connectivity rules are needed for specifying these restricts.

Connectivity rules work through defining to which types of and how many objects a certain type of object can connect. Connectivity rules can be categorized into two classes:

Edge-junction rules: Define to which types B of junctions can edges of type A can connect.

Edge-edge rules: Define through which types of junctions edges of type A can connect with edges of type B.

Default junctions: After we specify connectivity rules for a pair of edges of type A and B respectively, we can then add a default type of junction for them. If it is the case, when a user add connection with an edge of type B for an edge of type A, it is possible for him not to explicitly specify the junction type. The specified default junction type will be applied.

Logical network: When users build a facility network according to point and line objects and connectivity rules, a logical network will be automatically created for depicting connectivity and tracing analyses. An edge or junction feature in a facility network can correspond to multiple edges or junction elements in the logical network. Logical network will not store real coordinates but manage and maintain network with a group of tables. When edges and junctions of a facility network has been changed, the corresponding logical network will be maintained automatically. Source, sink, network weight, validity that will be introduced below rely on logical network. In another word, users face geometric network directly, but algorithms for network analyses need logical network built based on geometric network.

Source: The junctions that the resources flow out of, such as the power station and water station in the real world.

Sink The junctions that the resources flow into, such as the access points of the power grid and water network in the real world.

Network weight: Every network can associate with a group of weights. For example, there is a hydraulic weight in the water network, which associates with the edge's length. It represents that the water pressure continues to reduce with the water flowing due to the friction of the pipeline. A type of network weight can associate with one or more than one types of objects. The weight value can be 0 like all the values of the orphaned junctions that do not associate with any fields are 0.

Valid Feature and Invalid Feature: All the edges and junctions of the logical network can be invalid due to some reasons. Invalid edges/junctions turns into network barriers. Valid or invalid feature can be denoted by a field.

3 Facility Analysis Steps

Steps for facility analysis is as follows:

1. Build network dataset. Please refer to the help document of SuperMap iDesktop for details.
2. Build flow or levels for the network dataset. Please refer to the help document of SuperMap iDesktop for details.
3. Set facility analysis environment, that is, to call FacilityAnalyst .setAnalystSetting, (FacilityAnalystSetting facilityAnalystSetting) methods.
4. Load facility network model.
5. Implement facility analysis through facility analysis methods provided by the FacilityAnalyst class.

Details for facility analysis are introduced below.

4. Facility Network Analysis Environment Setting

Facility network analysis environment setting will influence the final analysis results. Parameters for the facility analysis include the network dataset, node ID field, edge ID field, from node ID field, to node ID field, weight information, distance tolerance, barrier nodes, barrier edges, flow direction field, etc.

Facility network analysis environment setting is implemented through the setAnalystSetting() method of the FacilityAnalyst class. The parameters of this method correspondes to the FacilityAnalystSetting class, which has been introduced. Note that the network dataset specified by the setNetworkDataset() method of the FacilityAnalystSetting class must be of a network dataset with flow direction and levels built.

5. Loading Facility Network Model

After facility network environment setting, the load() method of the FacilityAnalyst class must be called to validate the environment setting and load facility network data.

6 Facility Network Analysis Methods

Implement facility analysis through facility analysis methods provided by the FacilityAnalyst class. Next facility network analysis functions will be introduced in detail.

6.1 Check Loops

Methods for loop check:

Syntax:

public int[] FacilityAnalyst.checkLoops()

Returned value:

Returns array of edge IDs for loops in the network.

6.2 Check Connected Loops

Methods for connected loop check:

Method 1: Check the connected loops according to the given array of network node IDs.

Syntax:

public int[] FacilityAnalyst.findLoopsFromNodes(int[] nodeIDs)

Parameters:

nodeIDs: The specified array of the node IDs.

Return value:

Returns the array of network edge IDs for loops connected with given network nodes.

Method 2: Check the connected loops according to the given array of network edge IDs.

Syntax:

public int[] FacilityAnalyst.findLoopsFromEdges(int[] edgeIDs)

Parameters:

edgeIDs: The specified array of the edge IDs.

Return value:

Returns the array of network edge IDs for loops connected with given network edges.

6.3 Check Disconnected Edges

Methods for checking disconnected edges:

Method 1: Check the disconnected loops according to the given array of network node IDs.

Syntax:

public int[] FacilityAnalyst.findUnconnectedEdgesFromNodes(int[] nodeIDs)

Parameters:

nodeIDs: The specified array of the node IDs.

Returned value:

Returns the network edges disconnected with the given network nodes.

Method 2: Check the disconnected loops according to the given array of network edge IDs.

Syntax:

public int[] FacilityAnalyst.findUnconnectedEdgesFromEdges(int[] edgeIDs)

Parameters:

edgeIDs: The specified array of the edge IDs.

Returned value:

Returns the network edges disconnected with the given network edges.

6.4 Check Connected Edges

Methods for checking connected edges:

Method 1: Check the connected edges according to the given array of network node IDs.

Syntax:

public int[] FacilityAnalyst. findConnectedEdgesFromNodes (int[] nodeIDs)

Parameters:

nodeIDs: The specified array of the node IDs.

Returned value:

Returns the network edges connected with the given network nodes.

Method 2: Check the connected loops according to the given array of network edge IDs.

Syntax:

public int[] FacilityAnalyst. findConnectedEdgesFromEdges (int[] edgeIDs)

Parameters:

edgeIDs: The specified array of the edge IDs.

Returned value:

Returns the network edges connected with the given network edges.

6.5 Check Edge for Sink

There are two methods to get edge for sink.

Method 1: Finds all the edges between the specified edge and its sinks. The result is the array of edge IDs.

Syntax:

public int[] FacilityAnalyst. findSinkFromEdge (int[] edgeID, String weightName, boolean isLoopValid)

Parameters:

edgeID: The specified edge ID.

weightName: The name of the specified WeightFieldInfo object.

isLoopValid: Determines whether the loop is valid or not. Return true, if the loop is valid; false, otherwise.

Returned value:

Returns the array of the edge IDs.

Method 2: Finds all the edges between the specified node and its sinks. The result is the array of edge IDs.

Syntax:

public int[] FacilityAnalyst. findSinkFromNode (int[] nodeID, String weightName, boolean isLoopValid)

Parameters:

nodeID: The specified node ID.

weightName: The name of the specified WeightFieldInfo object.

isLoopValid: Determines whether the loop is valid or not. Return true, if the loop is valid; false, otherwise.

Returned value:

Returns the array of the edge IDs.

6.6 Check Edge for Source

There are two methods to get edge for source.

Method 1: Finds all the edges between the specified edge and its sources. The result is the array of edge IDs.

Syntax:

public int[] FacilityAnalyst. findSourceFromEdge (int[] edgeID,String weightName,boolean isLoopValid)

Parameters:

edgeID: The specified edge ID.

weightName: The name of the specified WeightFieldInfo object.

isLoopValid: Determines whether the loop is valid or not. Return true, if the loop is valid; false, otherwise.

Returned value:

Returns the array of the edge IDs.

Method 2: Finds all the edges between the specified node and its sources. The result is the array of edge IDs.

Syntax:

public int[] FacilityAnalyst. findSourceFromNode (int[] nodeID, String weightName, boolean isLoopValid)

Parameters:

nodeID: The specified node ID.

weightName: The name of the specified WeightFieldInfo object.

isLoopValid: Determines whether the loop is valid or not. Return true, if the loop is valid; false, otherwise.

Returned value:

Returns the array of the edge IDs.

6.7 Tracing Upstream

Concept--Upstream:

As figure 6-1 shows, the water represents the flowing substance in the facility network. Q is a sink node. Q receives the water input, and the nodes A, B, C, L, E, F and G are all upstream nodes of Q, which are also called the upstream of Q since the water from those nodes flows into Q finally. The edges AB, BC, CL, LQ, EB, FG and GL are the upstream edges of Q, which are also called the upstream of Q.

In addition, with respect to the edge LQ, node A, B, C, L, E, F and G are the upstream nodes of the edge LQ, which is called the upstream of edge LQ. Edge AB, BC, CL, EB, FG and GL are the upstream edge of LQ, which is also called the upstream of the edge LQ.

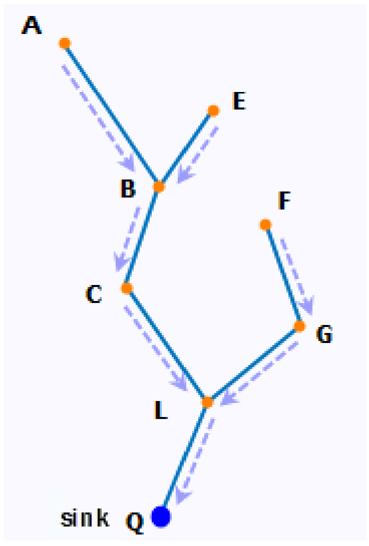

Figure 6-1 Upstream diagram

Method 1: Finds all the edges by tracing the upstreams of the specified node. The result is the array of the edge IDs.

Syntax:

public int[] FacilityAnalyst.traceUpFromNode(int nodeID, String weightName,boolean isLoopValid)

Parameters:

nodeID: The specified network node ID.

weightName: The name of the specified WeightFieldInfo object.

isLoopValid: Determines whether the loop is valid or not. Return true, if the loop is valid; false, otherwise.

Returned value:

Returns the array of upstream edge IDs for the analyzed node.

Example: In figure 6-1, with the node Q as the analysis node, tracking the edges along the upstream will return the ID array of edge AB, BC, CL, LQ, EB, FG and GL.

Method 2: Finds all the edges by tracing the upstreams of the specified edge. The result is the array of the edge IDs.

Syntax:

public int[] FacilityAnalyst.traceUpFromEdge(int edgeID, String weightName,boolean isLoopValid)

Parameters:

edgeID: The specified network edge ID.

weightName: The name of the specified WeightFieldInfo object.

isLoopValid: Determines whether the loop is valid or not. Return true, if the loop is valid; false, otherwise.

Returned value:

Returns the array of upstream edge IDs for the analyzed edge.

Example: In figure 6-1, with the edge LQ as the analysis edge, tracking the edges along the upstream will return the ID array of edge AB, BC, CL, EB, FG and GL.

6.8 Tracing Downstream

Concept--Downstream:

As figure 6-2 shows, suppose the water represents the flowing substance in the facility network. Q is a source node. Water flows out from Q, and A, B, C, L, E, F and G are all downstream nodes to Q, which is called the downstream of Q, as the nodes receive the water from Q. Edge BA, CB, LC, QL, BE, GF and LG are the downstream edges to Q, which is also called the downstream of Q.

In addition, with respect to the edge QL, the node A, B, C, L, E, F and G are the downstream nodes of the edge QL, which is called the downstream of the edge QL. The edge BA, CB, LC, BE, GF and LG are the downstream edges of QL, which is also called the upstream of the edge QL.

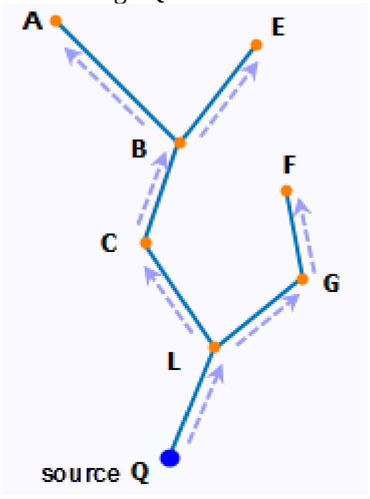

Figure 6-2 Downstream diagram

Method 1: Finds all the edges by tracing the downstreams of the specified node. The result is the array of the edge IDs.

Syntax:

public int[] FacilityAnalyst.traceDownFromNode(int nodeID,String weightName,boolean isLoopValid)

Parameters:

nodeID: The specified network node ID.

weightName: The name of the specified WeightFieldInfo object.

isLoopValid: Determines whether the loop is valid or not. Return true, if the loop is valid; false, otherwise.

Returned value:

Returns the array of downstream edge IDs for the analyzed node.

Example: In figure 6-2, with the node Q as the analysis node, tracking the edges along the downstream will return the ID array of edge BA, CB, LC, QL, BE, GF and LG.

Method 2: Finds all the edges by tracing the downstreams of the specified edge. The result is the array of the edge IDs.

Syntax:

public int[] FacilityAnalyst.traceDownFromEdge(int edgeID, String weightName,boolean isLoopValid)

Parameters:

edgeID: The specified network edge ID.

weightName: The name of the specified WeightFieldInfo object.

isLoopValid: Determines whether the loop is valid or not. Return true, if the loop is valid; false, otherwise.

Returned value:

Returns the array of downstream edge IDs for the analyzed edge.

Example: In figure 6-2, ith the edge QL as the analysis edge, tracking the edges along the upstream will return the ID array of edge BA, CB, LC, BE, GF and LG.

8.9 Common Upstream

Method 1: Finds the common upstream edges of those node according to the given array of the node IDs;

Syntax:

public int[]FacilityAnalyst.findCommonAncestorsFromNodes

(int[] nodeIDs,String weightName,boolean isLoopValid)

Parameters:

nodeIDs: The specified array of the node IDs.

weightName: The name of the specified WeightFieldInfo object.

isLoopValid: Determines whether the loop is valid or not. Return true, if the loop is valid; false, otherwise.

Returned value:

Returns the common upstream edges ID array of the given node.

As figure 6-3 shows, the flow direction is the arrow direction in the picture. If searching the common upstream edges of Q and P, the ID array of edge AB, BC, CL, EB, GL and FG will be returned.

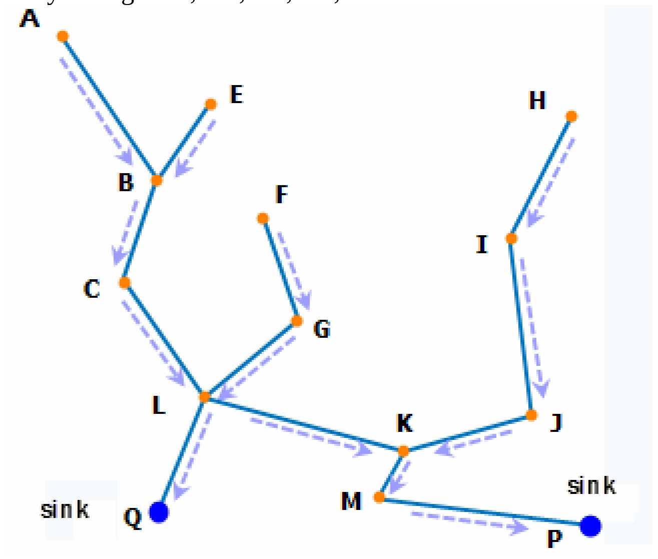

Figure 6-3 Common upstream diagram

Method 2: Finds the common upstream edges of those edges according to the given array of the edge IDs.

Syntax:

public int[]FacilityAnalyst.findCommonAncestorsFromEdges

(int[] edgeIDs,String weightName,boolean isLoopValid)

Parameters:

nodeIDs: The specified array of the node IDs.

weightName: The name of the specified WeightFieldInfo object.

isLoopValid: Determines whether the loop is valid or not. Return true, if the loop is valid; false, otherwise.

Returned value:

Returns the common upstream edges ID array of the given edge.

As figure 6-3 shows, the flow direction is the arrow direction in the picture. If searching the common upstream edges of Q and P, the ID array of edge AB, BC, CL, EB, GL and FG will be returned.

6.10 Common Downstream

Method 1: Finds the common downstream edges of those node according to the given array of the node IDs;

Syntax:

public int[] FacilityAnalyst.findCommonCatchementsFromNodes(int[] nodeIDs,String weightName,boolean isLoopValid)

Parameters:

nodeIDs: The specified array of the node IDs.

weightName: The name of the specified WeightFieldInfo object.

isLoopValid: Determines whether the loop is valid or not. Return true, if the loop is valid; false, otherwise.

Returned value:

Returns the common downstream edges ID array of the given node.

As figure 6-4 shows, the flow direction is the arrow direction in the picture. If searching the common downstream edges of Q and P, the ID array of edge AB, BC, CL, EB, GL and FG will be returned.

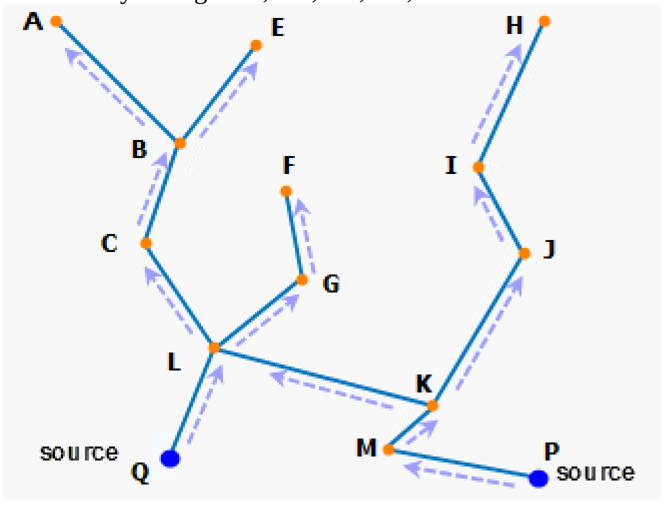

Figure 6-4 Common downstream diagram

Method 2: Finds the common downstream edges of those edges according to the given array of the edge IDs.

Syntax:

public int[] FacilityAnalyst.findCommonCatchmentsFromEdges(int[] edgeIDs,String weightName,boolean isLoopValid)

Parameters:

nodeIDs: The specified array of the node IDs.

weightName: The name of the specified WeightFieldInfo object.

isLoopValid: Determines whether the loop is valid or not. Return true, if the loop is valid; false, otherwise.

Returned value:

Returns the common downstream edges ID array of the given edge.

As figure 6-4 shows, the flow direction is the arrow direction in the picture. If searching the common downstream edges of Q and P, the ID array of edge AB, BC, CL, EB, GL and FG will be returned.

6.11 Upstream Path Analysis

Method 1: Searches the least cost upstream path according to the specified node ID.

Syntax:

public FacilityPathResult FacilityAnalyst.findPathUpFromNode(int nodeID,String weightName)

Parameters:

nodeID: The specified node ID.

weightName: The name of the specified WeightFieldInfo object.

Returned value:

The result is stored in the FacilityPathResult class.

Table 6.1 Methods for the FacilityPathResult class

Method	Description
getNodes	**Gets the array of the node IDs passed by the result path of the facility path analysis.**
Syntax: int[] getNodes	
getEdges	**Gets the array of the edge IDs passed by the result path of the facility path analysis.**
Syntax: int[] getEdges	

As figure 6-5 shows, with the node Q as the given node participating the analysis, tracking the shortest path for the upstream is to find all upstream paths of Q and then pick up the shortest path.

The upstream path of the node Q refers to tracking the upstream nodes of Q until the most upstream nodes, shown as nodes A, E, F in figure 6-5. Therefore, there are three upstream paths, ABCLQ, EBCLQ and FGLQ. It is obvious that FGLQ is the shortest upstream path.

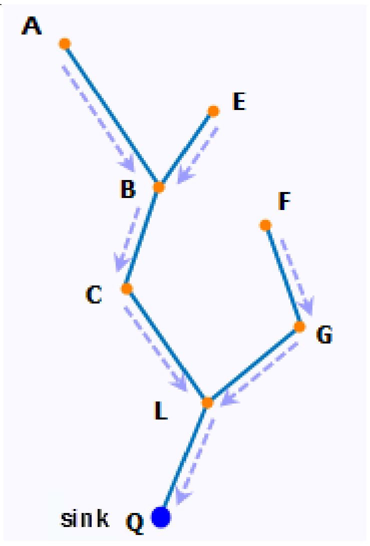

Figure 6-5 Upstream path analysis diagram

Method 2: Searches the shortest path or path with least number of edges upstream according to the specified node ID.

Syntax: public FacilityPathResult FacilityAnalyst.findPathUpFromEdge(int edgeID,String weightName)

Parameters:

edgeID: The specified edge ID.

weightName: The name of the specified WeightFieldInfo object.

Returned value:

The result is stored in the FacilityPathResult class.

As figure 6-5 shows, with the edge LQ as the given edge participating the analysis, tracking the shortest path for the upstream is to find all upstream paths of LQ and then pick up the shortest path.

The upstream path of the edge LQ refers to tracking the upstream nodes of LQ until the most upstream nodes, shown in the above picture of the node A, E, F. Therefore, there are three upstream paths, ABCL, EBCL and FGL. It is obvious that FGL is the shortest upstream path.

6.12 Downstream Path Analysis

Method 1: Searches the least cost downstream path according to the specified node ID.

Syntax:

public FacilityPathResult FacilityAnalyst.findPathDownFromNode(int nodeID,String weightName)

Parameters:

nodeID: The specified node ID.

weightName: The name of the specified WeightFieldInfo object.

Returned value:

The result is stored in the FacilityPathResult class.

As figure 6-6 shows, with the node Q as the given node participating the analysis, tracking the shortest path for the downstream is to find all downstream paths of Q and then pick up the shortest path.

The downstream path of the node Q refers to tracking the downstream nodes of Q until the most upstream nodes, shown as nodes A, E, F in figure 6-6. Therefore, there are three upstream paths, QLCBA, QLCBE and QLGF. It is obvious that QLGF is the shortest upstream path.

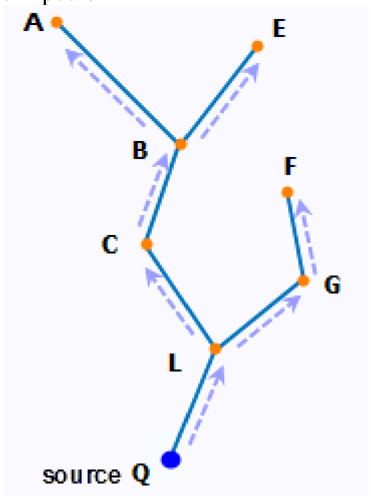

Figure 6-6 Downstream path analysis diagram

Method 2: Searches the shortest path or path with least number of edges downstream according to the specified node ID.

Syntax:

public FacilityPathResult FacilityAnalyst.findPathDownFromEdge(int edgeID,String weightName)

Parameters:

edgeID: The specified edge ID.

weightName: The name of the specified WeightFieldInfo object.

Returned value:

The result is stored in the FacilityPathResult class.

As figure 6-6 shows, with the edge QL as the given edge participating the analysis, tracking the shortest path for the downstream is to find all downstream paths of QL and then pick up the shortest path.

The downstream path of the edge QL refers to tracking the downstream nodes of QL until the most upstream nodes, shown as nodes A, E, F in figure 6-6. Therefore, there are three downstream paths, LCBA, LCBE and LGF. It is obvious that LGF is the shortest upstream path.

6.13 Facility Network Analysis

Method 1: The find path analysis. Find the path which is shortest or has the least edges between the specified start node and the end node.

Syntax:

public FacilityPathResult FacilityAnalyst.findPathFromNodes(int startNodeID,int endNodeID,String weightName)

Parameters:

startNodeID: The specified start node ID.

endNodeID: The specified end node ID.

weightName: The name of the specified WeightFieldInfo object.

Returned value:

The result is stored in the FacilityPathResult class.

Method 2: The find path analysis. Finds the path which is shortest or has the least edges between the specified start edge ID and the end edge ID.

Syntax:

public FacilityPathResult FacilityAnalyst.findPathFromEdges(int startEdgeID,int endEdgeID,String weightName)

Parameters:

startEdgeID: The specified start edge ID.

endEdgeID: The specified end edge ID.

weightName: The name of the specified WeightFieldInfo object.

Returned value:

The result is stored in the FacilityPathResult class.