Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 4235523, 14 pages
https://doi.org/10.1155/2022/4235523

Research Article

Architecture Design and Code Implementation of Road Network Path Search System

Hongmin Shen[1] **and Hongzhou Pan**[2]

[1]*School of Management, Xiamen University Tan Kah Kee College, Zhangzhou 363105, China*
[2]*College of Humanities, Xiamen Huaxia University, Xiamen 361024, China*

Correspondence should be addressed to Hongzhou Pan; panhz@hxxy.edu.cn

Received 23 March 2022; Revised 8 April 2022; Accepted 4 May 2022; Published 26 May 2022

Academic Editor: Kuruva Lakshmanna

Path search is a hot issue in computer science and artificial intelligence science. When the user enters the starting point and ending point to be queried in the road network path search system, the system will return the best path to the user. In this paper, the road network path search system that can run and calculate the optimal navigation path to the test data is developed by designing the software architecture through the comprehensive use of database design, programming language, shortest path algorithm, UML diagram, software development model, GIS system source data, and other methods. Based on the software design principles of scalability, flexibility, and pluggability, the design and code in this paper can be practically applied to various systems such as GIS, GPS, logistics robots, unmanned aerial vehicles, and autonomous vehicles to realize their road network path planning and navigation functions.

1. Introduction

The road network path search system is one of the core functions of various systems such as GIS, GPS, logistics robots, unmanned aerial vehicles, and autonomous vehicles [1]. The shortest path search algorithm includes Dijkstra algorithm, A∗ algorithm, SPFA algorithm, Bellman-Ford algorithm, Floyd-Warshall algorithm, and Johnson algorithm.

Dijkstra algorithm is a typical single-source shortest path algorithm to calculate the shortest path from one point to all other points, and its main feature is that the starting point is the center and the outer layer expands until it reaches the ending point [2]. For example, this algorithm can find the shortest path between two cities if the vertices in the graph represent cities, and the weights on the edges represent the driving distance between cities.

Bellman-Ford algorithm is also a single-source shortest path algorithm, which differs from the Dijkstra algorithm in that the weight of the edge can be negative [3].

SPFA algorithm is a queue optimization of the Bellman-Ford algorithm, with relatively good timeliness. Unlike the Dijkstra algorithm and the Bellman-ford algorithm, the SPFA algorithm has unstable time efficiency, which means that the time required for different graphs varies greatly [4].

Floyd-Warshall algorithm is an algorithm for solving the shortest path between any two points. It can correctly deal with the shortest path problem of directed graphs or negative weights [5].

A∗ algorithm, like the Dijkstra algorithm, can calculate the shortest path and is a heuristic search algorithm like the Bellman-Ford algorithm [6].

Johnson algorithm is a combination of the Dijkstra algorithm and Bellman-Ford algorithm and used to solve the shortest path problem in edge graphs with negative weights [7].

Much work so far is focused on algorithms of the road network path search system, but the design and complete implementation of the complete system are rarely discussed.

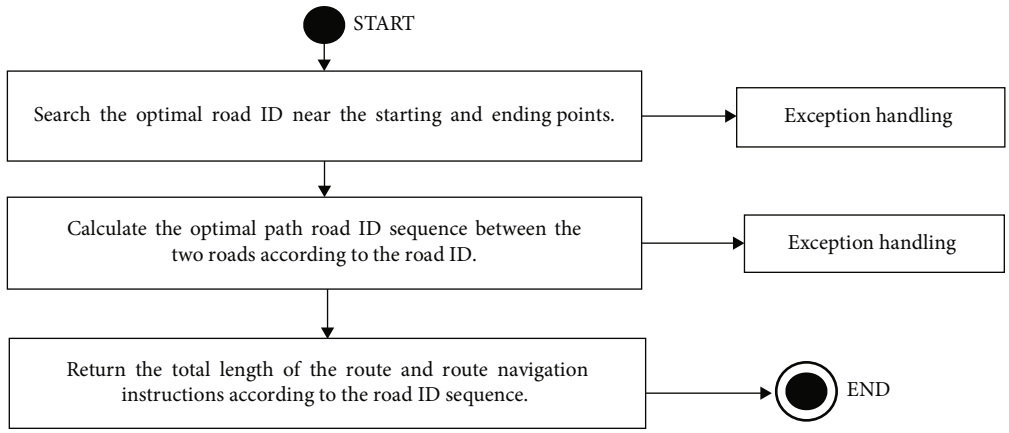

Figure 1: The core processing flow of the path search system.

According to the software design principles, this paper designs and implements a complete road network path search system that can be applied in practice. The system design and code are suitable for practical applications of geographic information, logistics, unmanned aerial vehicles, automobiles, and other industries [8].

This research is conducted based on the study on existing road network structure of various GIS, GPS, and route planning systems and the software design requirements of accuracy, robustness, flexibility, reusability, and efficiency. The research process is as follows. Firstly, the functional requirements of the system are determined. Secondly, a road network database is developed based on the function compatible with various application scenarios [2]. Thirdly, the software architecture of the path search system is designed based on the database. Fourthly, the Microsoft SQL Server is used to implement the database of system. Finally, the Microsoft C# programming language is adopted to realize the software code of each function of the system.

2. Architecture Design of the Road Network Path Search System

2.1. Function Description of Path Search System. The function of path search system is as follows: after the user inputs the starting point and the ending point of the query, the system returns the optimal path information to the user [9]. The information returned to the user includes the total length of the path and the instructions to navigate the path. The core processing flow of the program is shown in Figure 1.

2.2. Database Design of Road Network Path Search System. For different application scenarios, we design a database compatible with various road networks. The database table design of the road network path search system is as follows.

2.2.1. Road Information Table. The structure of the road information table is shown in Table 1 below.

The road information table structure is described as follows:

Table 1: Road table structure.

Column name	Data type	Primary key	Is null
RoadID	int	Yes	No
Name	Varchar(50)_	No	No
Length	Bigint	No	No
FatherRoadID	int	No	No

Table 2: The structure of road network topology table.

Column name	Data type	Primary key	Is null
ConnectionID	int	Yes	No
RoadID	int	No	No
LinkedRoadID	int	No	No
RightOrLeft	Numeric (1)	No	Yes
Direction	Numeric (2)	No	Yes

(i) The data of the road table is obtained by SQL program from the road network data of the GIS system or by field measurement. The road network path search system is generally used together with the GIS system. So, the information including road ID, name, length, and father road ID of the road table can be obtained from the database table of the GIS system by SQL program

(ii) Each road ID requires a road mark that can be mapped to the map metadata

(iii) The names of road tables are distinguished by short codes of location and type. Examples of short code are as follows: Xiamen city's short code is XM, Chengdu city's short code is CD, and Wanda Plaza's short code is WDPZ. The types of the road table include car, foot, logistics, fly, and robot. Based on short codes and types, the following are examples of road tables names: the driving road table name of Xiamen city is XM_CarRoads, the walking road table name of Xiamen city is XM_FootRoads, the logistics road table name of Chengdu city is CD_

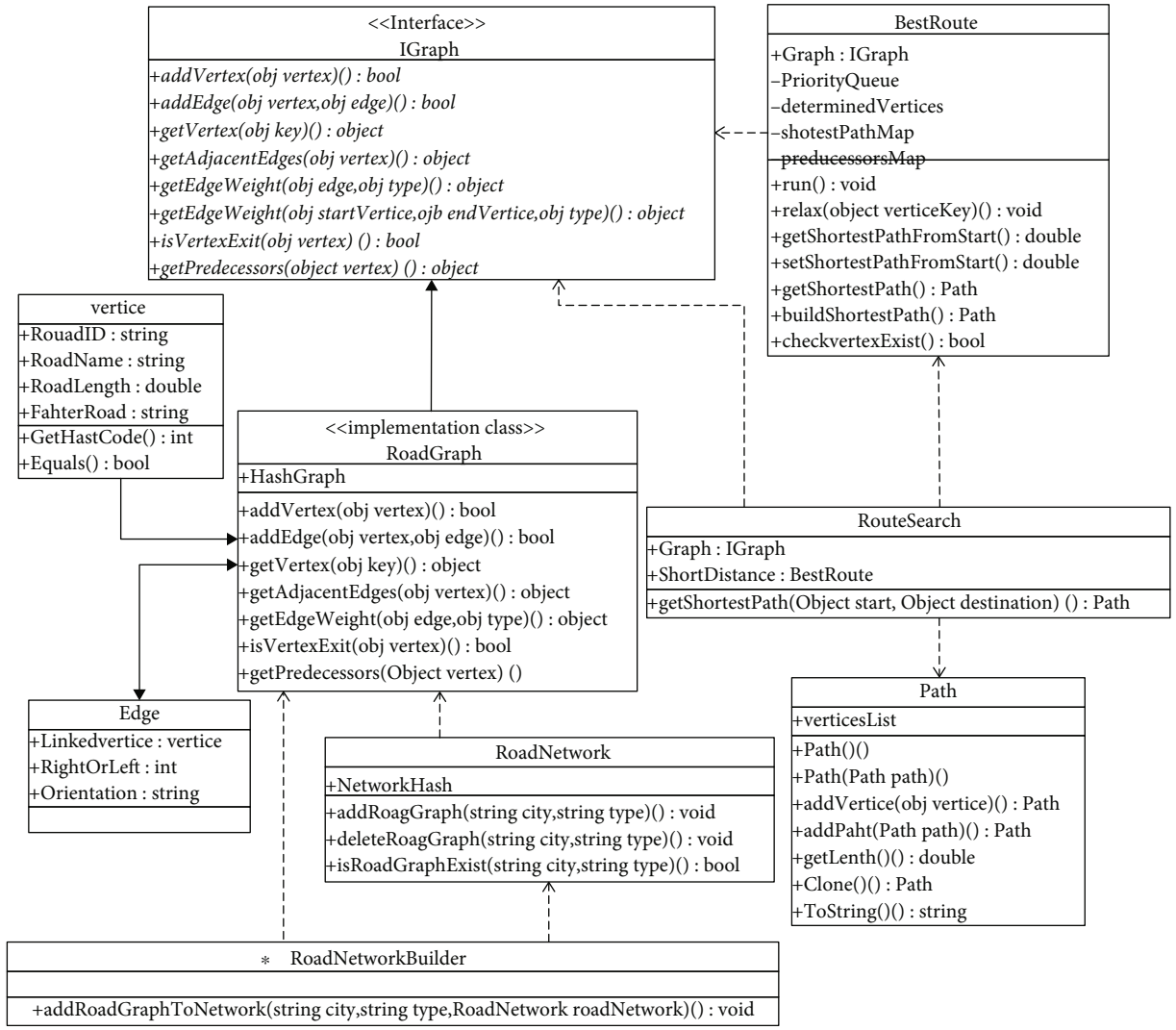

Figure 2: UML class diagram of path search system program.

LogisticsRoads, and the road table name for Wanda Plaza food delivery robot is WDPZ_RobotRoads

(iv) Field description: RoadID indicates the Road ID, Name represents the road name, Length is the road length (or the time or other parameters for calculating the optimal path), FatherRoadID means the parent road ID to which the road belongs

2.2.2. Road Network Topology Table. The road network topology table stores the connection information of the road through a directed graph. The structure of road network topology table is shown in Table 2 below.

The road network topology table structure is described as follows:

(i) The data of road network topology table is obtained by SQL program from the road network data of the GIS system or by field measurement

(ii) The columns of RoadID and LinkedRoadID follow the foreign key constraint by the column RoadID of the road information table

(iii) The road network is a vector diagram that considers one-way lines

(iv) The data description of RightOrLeft column: 0: keep straight, 1: turn right, and 2: turn left

(v) The data description of direction column: 10: east, 20: south, 30: west, and 40: north

(vi) Same as those of the road information table, the names of road network topology tables are distinguished by short codes of location and type. For example: the driving road network topology table name of Xiamen city is XM_CarWayNetwork, the walking road network topology table name of Xiamen city is XM_FootWayNetwork, the logistics road network topology table name of Chengdu city

```
using System.Data.SqlClient;
namespace JGXY.LBS.Route{
public class RoadNetworkBuilder{
public SqlConnection con;
public RoadNetworkBuilder(SqlConnection conn)
{this.con = conn; }
public void addRoadGraphToNetwork(string city,string type,RoadNetwork roadNetwork) {
string roadSql = "select * from "+city+"_"+type+"Roads";
string wayNetworkSql = "select w.RoadID as V_RoadID,r.RoadID as E_RoadID,r.FatherRoadID as E_FatherRoadID ,r.Length as E_length,r.[Name] as E_Name,isnull(RightOrLeft,0) as RightOrLeft,isnull(Orientation,10) as Orientation" +" from " +city+"_"+type+"WayNetwork w left outer join "+city+"_"+type+"Roads r on w.LinkedRoadID = r.RoadID ";
SqlDataAdapter roadAdp = new SqlDataAdapter(roadSql,con);
SqlDataAdapter wayNetworkAdp = new SqlDataAdapter(wayNetworkSql,con);
try {
System.Data.DataSet roadData = new System.Data.DataSet();
roadAdp.Fill(roadData);
System.Data.DataSet edgeData = new System.Data.DataSet();
    wayNetworkAdp.Fill(edgeData);
    RoadGraph graph= new RoadGraph(roadData.Tables[0].Rows.Count);
    //read database initialize route network
    foreach(System.Data.DataRow myRow in roadData.Tables[0].Rows){
        Vertice vertex= new Vertice();
        vertex.RoadID = myRow["RoadID"].ToString();
        vertex.FatherRoad = myRow["FatherRoadID"].ToString();
        vertex.RoadLength = System.Convert.ToDouble(myRow["Length"].ToString());
        vertex.RoadName = myRow["Name"].ToString();
        graph.addVertex(vertex);
}
        foreach(System.Data.DataRow row in edgeData.Tables[0].Rows){
    Vertice vertex = new Vertice(row["V_RoadID"].ToString());
    Edge edge = new Edge(new Vertice(row["E_RoadID"].ToString(),row["E_Name"].ToString(),System.Convert.ToDouble(row["E_Length"].ToString()),row["E_FatherRoadID"].ToString()),(Direction.RightOrLeft)System.Enum.Parse(typeof(Direction.RightOrLeft) ,row["RightOrLeft"].ToString()),(Direction.Orientation)System.Enum.Parse(typeof(Direction.Orientation) ,row["Orientation"].ToString()));
        graph.addEdge(vertex,edge);
    }
        roadNetwork.addRoadGraph("RT","Car",graph);}
        catch(System.Exception e)
        {
        throw new Entity.LBS.Route.RouteException(e.Message);
        }
    }
}// END INTERFACE DEFINITION RoadNetworkBuilder
    }
```

Pseudocode 1

is CD_LogisticsWayNetwork, and the road network topology table name for Wanda Plaza food delivery robot is WDPZ_RobotWayNetwork

2.3. UML Class Diagram Design of Road Network Path Search System. The UML class diagram is a graphical notation used to construct and visualize object-oriented systems. Also, it is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, methods, and the relationships among objects. The principle of single responsibility means that a class does only one specific thing and does not try to do more than it should. According to the principle of single responsibility, opening and closing, Richter substitution, interface, separation, and dependency inversion, the UML class diagram of the road network path search system we designed is shown in Figure 2 below.

3. Code Implementation of Road Network Path Search System

The road path search system is developed using the Microsoft C#.NET programming language, which is a modern, object-oriented, and type-safe programming language with characteristics of being advanced, safe, stable, expandable, and portable. C# enables developers to build many types of secure and robust applications that run in the NET ecosystem. The function description and code of each part of the program are as follows.

```
1.  namespace JGXY.LBS.Route{
2.  public class RoadNetwork{
3.  public System.Collections.Hashtable NetworkHash;
4.      public RoadNetwork(){
5.      NetworkHash = new System.Collections.Hashtable(10);
6.      }
7.      public void addRoadGraph(string city,string type,IGraph graph){
8.      NetworkHash.Add(city+"_"+type,graph);
9.      }
10.       public void deleteRoadGraph(string city,string type){
11.       NetworkHash.Remove(city+"_"+type);
12.       }
13.       public bool isRoadGraphExist(string city,string type){
14.       return NetworkHash.Contains(city+"_"+type);
15.       }
16.       }// END CLASS DEFINITION RoadNetwork
17.   }
```

Pseudocode 2

```
1.  namespace JGXY.LBS.Route
2.  public interface IGraph
3.  {
4.      bool addVertex(object vertex);
5.      bool addEdge(object vertex,object edge);
6.      object getVertex(object key);
7.      object getAdjacentEdges(object vertex);
8.      object getEdgeWeight(object edge,object type);
9.      object getEdgeWeight(object startVertice,object endVertice,object type);
10.   bool isVertexExit(object vertex);
11.   System.Collections.ICollection getPredecessors(object vertex);
12.   int getVerticesNumber();
13.   }// END INTERFACE DEFINITION IGraph
```

Pseudocode 3

3.1. RoadNetworkBuilder Class. The RoadNetworkBuilder class initializes each road network object according to location and traffic type. The road network object at each location is a global object, which only needs to be initialized once.

The main method of this class is described as follows:

(i) addRoadGraphToNetwork (string city, string type, RoadNetwork roadNetwork): initialize a road network object RoadNetwork according to the location and type (Car, Foot, logistics, Fly, Robot)

The code implementation of the RoadNetworkBuilder class is as follows.

3.2. RoadNetwork Class. The function of the RoadNetwork class is to store various locations and all types of road networks in the system. All types of road networks in the system are initialized only once when the system is started.

The code implementation of the RoadNetwork class is as follows.

3.3. IGraph Class. The IGraph class is the network graph interface of the optimal path algorithm. The interface is to realize the scalability and compatibility of the algorithm; that is, the optimal path algorithm can use any topology network graph which conforms to the interface.

The code implementation of the IGraph class is as follows:

3.4. RoadGraph Class. The RoadGraph class implements the interface of IGraph class. It saves a road network of one location and one type. The data of the road network is saved by the following format:

(i) Every road is regarded as a vertex, and the distance between the two roads is expressed by the length of the second road

(ii) All vertices (roads) are stored in the hash table. The key of the hash table is road vertex, and the value of the hash table is the set of linked road vertex of this road

The code implementation of the RoadGraph class is as follows.

```
using System.Collections;
namespace JGXY.LBS.Route
{   public class RoadGraph:IGraph
  { private int VerticeNumber;
    public Hashtable HashGraph;
    public RoadGraph(int totalVerticesNumber)
    {   VerticeNumber = totalVerticesNumber;
      HashGraph = new Hashtable(totalVerticesNumber,1.0f);
    }
     public bool addVertex(object vertex)
     {
     if(!HashGraph.ContainsKey(vertex))
     {   ArrayList linkedEdges = new ArrayList(6);
       HashGraph.Add(vertex,linkedEdges);
       return true;
       }
     else
       { throw new JGXY.LBS.Route.RouteException("Vertice exits"); }
     }
     public bool addEdge(object vertex,object edge)
     {   if(HashGraph.ContainsKey(vertex))
       {   ArrayList linkedEdges = HashGraph[vertex] as ArrayList;
         linkedEdges.Add(edge);
         return true;
       }
       else
       {throw new JGXY.LBS.Route.RouteException("Vertex does not exit");      }
     }
     public object getVertex(object roadID)
     {   return null;
     }
     public object getAdjacentEdges(object vertex)
   {   if(HashGraph.ContainsKey(vertex))
     {ArrayList linkedEdges = HashGraph[vertex] as ArrayList;
     ArrayList adjacentEdges = new ArrayList(6);
     adjacentEdges.AddRange(linkedEdges);
     return adjacentEdges;
     }
     else
     {   throw new JGXY.LBS.Route.RouteException("Vertex does not exit");}
   }
     public object getEdgeWeight(object edge,object type)
     {return ((Edge)edge).LinkedVertice.RoadLength;}
     public object getEdgeWeight(object startVertice,object endVertice,object type)
     {return ((Vertice)endVertice).RoadLength;}
     public bool isVertexExit(object vertex)
     {   if(HashGraph.ContainsKey(vertex))
     return true;
     else
       return false;
       }
     public System.Collections.ICollection getPredecessors(object vertex)
     {return null;}
     public int getVerticesNumber()
     {return this.VerticeNumber; }
   }
}// END CLASS DEFINITION RoadGraph
}
```

Pseudocode 4

```
1.  namespace JGXY.LBS.Route
2.  {
3.  public class Vertice
4.  { public string RoadID;
5.  public string RoadName;
6.  public double RoadLength;
7.  public string FatherRoad;
8.  public Vertice()
9.  { }
10.  public Vertice(string roadID)
11.  {   RoadID = roadID; }
12.  public Vertice(string roadID,string roadName,double roadLength,string fatherRoad)
13.  { RoadID = roadID;
14.  RoadName = roadName;
15.  RoadLength = roadLength;
16.  FatherRoad = fatherRoad;
17.  }
18.  public override int GetHashCode()
19.  {return RoadID.GetHashCode();}
20.  public override bool Equals(object obj)
21.  { if(this.RoadID == ((Vertice)obj).RoadID)
22.  return true;
23.  else
24.     return false;
25.  }
26.  public Vertice Clone()
27.  { return new Vertice(RoadID,RoadName,RoadLength,FatherRoad); }
28.  }// END CLASS DEFINITION vertice
29.}
```

Pseudocode 5

```
1.  namespace JGXY.LBS.Route
2.  {
3.  public class Edge
4.  {
5.  public Vertice LinkedVertice;
6.  public Direction.RightOrLeft RightOrLeft;
7.  public Direction.Orientation Orientation;
8.  public Edge(Vertice vertice,Direction.RightOrLeft rightOrLeft,Direction.Orientation orientation)
9.  { LinkedVertice = vertice;
10.  RightOrLeft = rightOrLeft;
11.  Orientation = orientation;
12.  }
13.  public override int GetHashCode()
14.  { return LinkedVertice.RoadID.GetHashCode(); }
15.  public override bool Equals(object obj)
16.  { if(this.LinkedVertice.RoadID == ((Edge)obj).LinkedVertice.RoadID)
17.     return true;
18.  else
19.     return false;
20.  }
21.  public Edge Clone()
22.  {return new Edge(LinkedVertice.Clone(),RightOrLeft,Orientation); }
23.  }// END CLASS DEFINITION Edge
24.}
```

Pseudocode 6

```
1. using System.Collections;
2. namespace JGXY.LBS.Route
3. { public class BestRoute
4.   { public IGraph Graph;
5.   private PriorityQueue priorityQueue;
6.   private Hashtable determinedEdges;
7.   private Hashtable shortestPathMap;
8.   private Hashtable preducessorsMap;
9.   public BestRoute(IGraph graph)
10.   { this.Graph = graph;
11.   int verticeNumber = graph.getVerticesNumber();
12.   priorityQueue = new PriorityQueue(verticeNumber);
13.   determinedEdges = new Hashtable(verticeNumber);
14.   shortestPathMap = new Hashtable(verticeNumber);
15.   preducessorsMap = new Hashtable(verticeNumber);
16.   }
17.   public void run(object sourceVertex,object destinationVertex)
18.   { priorityQueue.Clear();
19.   determinedEdges.Clear();
20.   determinedEdges.Add(new Edge((Vertice)sourceVertex,Direction.RightOrLeft.Straight,Direction.Orientation.North),0);
21.   ArrayList adjacentEdges = (ArrayList)Graph.getAdjacentEdges(sourceVertex);
22.   for(int i =0; i <adjacentEdges.Count;i++)
23.   { priorityQueue.Enqueue(adjacentEdges[i],((Edge)adjacentEdges[i]).LinkedVertice.RoadLength);
24.   shortestPathMap.Add((Edge)adjacentEdges[i],((Edge)adjacentEdges[i]).LinkedVertice.RoadLength);
25.   preducessorsMap.Add((Edge)adjacentEdges[i], new
26.            Edge((Vertice)sourceVertex,Direction.RightOrLeft.Straight,Direction.Orientation.North));
27.   }
28.   while(!priorityQueue.IsEmpty())
29.   { Edge minEdge = (Edge)priorityQueue.Dequeue();
30.   if(minEdge.LinkedVertice.Equals(destinationVertex))
31.   { preducessorsMap.Add("END",minEdge);
32.   break;
33.      }
34.         determinedEdges.Add(minEdge,null);
35.      relax(minEdge);
36.      }
37.   }
38.   public void relax(object edge)
39.   { ArrayList adjacentEdges = (ArrayList)Graph.getAdjacentEdges(((Edge)edge).LinkedVertice);
40.   for(int i =0; i <adjacentEdges.Count;i++)
41.   { if(!determinedEdges.Contains(adjacentEdges[i]))
42.      {  double  distance  =  System.Convert.ToDouble(shortestPathMap[edge])  +
((Edge)adjacentEdges[i]).LinkedVertice.RoadLength;
43.      if(distance < priorityQueue.getLenght(adjacentEdges[i]))
44.      { priorityQueue.Enqueue(adjacentEdges[i],distance);
45.      shortestPathMap.Add((Edge)adjacentEdges[i],distance);
46.      preducessorsMap.Add((Edge)adjacentEdges[i],edge);
47.      }
48.   }
49.   }
50. }
51.public Path getShortestPath(Vertice start ,Vertice end)
52. { checkGraph();
53.   checkStartVerticeExits(start);
54.   checkEndVerticeExits(end);
55.   Path path = new Path();
56.   path.SetStartVertice(start);
57.   path.SetEndVertice(end);
58.   Edge preEdge = preducessorsMap["END"] as Edge;
59.   ArrayList pathArray = new ArrayList(10);
```

Pseudocode 7: Continued.

```
60.      pathArray.Add(preEdge);
61.   do
62.   { preEdge = preducessorsMap[preEdge] as Edge ;
63.   pathArray.Add(preEdge);
64.   }
65.   while(preEdge !=null && preEdge.LinkedVertice != start);
66.   pathArray.RemoveAt(pathArray.Count -1);
67.   pathArray.Reverse();
68.   path.viaEdges = pathArray;
69.   return path;
70. }
71. public void checkGraph()
72. { if(Graph == null)
73.   { throw new JGXY.LBS.Route.RouteException("Can't find a Graph"); }
74. }
75. public void checkStartVerticeExits(Vertice start)
76. { if(!Graph.isVertexExit(start))
77.   throw new JGXY.LBS.Route.RouteException("Can't find start vertex");
78. }
79. public void checkEndVerticeExits(Vertice end)
80. { if(!Graph.isVertexExit(end))
81.   throw new JGXY.LBS.Route.RouteException("Can't find end vertex");
82. }
83.
84. }// END CLASS DEFINITION
85. public class PriorityQueue
86. { Hashtable priorityQueue;
87. public PriorityQueue(int verticeNumber)
88. {priorityQueue = new Hashtable(verticeNumber); }
89. public void Enqueue(object obj,double lenght)
90. {priorityQueue[obj] = lenght;}
91. public object Dequeue()
92. { IDictionaryEnumerator myEnumerator = priorityQueue.GetEnumerator();
93.   Edge minEdge = null;
94.   double length = double.MaxValue;
95.   while(myEnumerator.MoveNext())
96.   {
97.            if(System.Convert.ToDouble(myEnumerator.Value) < length)
98.   {
99.   length =System.Convert.ToDouble(myEnumerator.Value) ;
100.   minEdge = (Edge)myEnumerator.Key;
101.   }
102.   }
103.   priorityQueue.Remove(minEdge);
104.   return minEdge;
105. }
106. public void Clear()
107. { priorityQueue.Clear(); }
108. public bool IsEmpty()
109. {
110.   if(priorityQueue.Count > 0)
111.   return false;
112.   else
113.   return true;
114. }
115. public bool Exits(object obj)
116. {return priorityQueue.Contains(obj); }
117. public void Reset(object obj,double lenght)
118. { priorityQueue.Add(obj,lenght); }
119. public double getLenght(object obj)
```

Pseudocode 7: Continued.

```
120. {
121.   if(priorityQueue.ContainsKey(obj))
122.       return System.Convert.ToDouble(priorityQueue[obj]);
123.   else
124.       return double.MaxValue;
125. }
126. }
127.}
```

Pseudocode 7

```
1.   namespace JGXY.LBS.Route
2.     { public class RouteSearch
3.         { private IGraph Graph;
4.         private BestRoute bestroute;
5.         public RouteSearch(RoadGraph graph)
6.         { Graph = graph;
7.         bestroute = new BestRoute(Graph);
8.         }
9.         public Path getShortestPath(Vertice start,Vertice end)
10.          {   return bestroute.getShortestPath(start,end); }
11.          }// END CLASS DEFINITION RouteSearch
12.      }
```

Pseudocode 8

3.5. Vertice Class. The function of the vertex class is to store the information of the road. Note: the Vertice class overrides the GetHashCode methods and Equals methods of the base class.

(i) Equals(object obj): the Equals method, defined by the Object class of the C# language, determines whether the specified object is equal to the current one by comparing the memory address of the two objects. So the Vertice class should override the Equals method to determine whether two Vertice objects are equal by comparing the values of the road ID

(ii) GetHashCode(): the Vertice class also overrides the GetHashCode method to generate the hash code by the value of the road ID

The code implementation of the Vertice class is as follows.

3.6. Edges Class. Edges class is a linked edge class, used to store the linked way of each road.

The code implementation of the Edges class is as follows:

3.7. BestRoute Class. The BestRoute class that can implement the shortest path algorithm is used to compute the shortest path between two vertices.

The code implementation of the BestRoute class is as follows.

3.8. RouteSearch Class. The RouteSearch class is used to calculate the shortest path between two vertices and then store the result in the Path object.

The code implementation of the RouteSearch class is as follows.

3.9. Path Class. The Path class is used to store the results of the best path algorithm. The code implementation of the Path class is as follows:

4. Test and Results of the Road Path Search System

4.1. Test Road Network of the Road Path Search System. In this paper, the test road network as shown in Figure 3 is used to test the system. The location of the road network is set to XM city, and the road network is a car drive network.

4.2. Test Database of the Road Path Search System. According to the test road network diagram in Figure 3, we initialize the data in the following test database tables in XM city. The name of the database is Route, the database login user is "sa2," and the database login password is "sa123[]."

4.2.1. The Test Data Initialization of the Road Network Topology Table. We initialize the test data of the road network topology table XM_CarWayNetwork in XM city as shown in Table 3, which realizes the road network topology of Figure 3.

4.2.2. The Test Data Initialization of the Road Information Table. According to the test road network diagram in Figure 3, we initialize the test data of the road information table named XM_CarNetwork that realizes the road network topology in Figure 3. The table XM_CarNetwork is used for car driving of XM city, as shown in Table 4 below.

```
1.  namespace JGXY.LBS.Route
2.  { public class Path
3.     {
4.  private Vertice startVertice;
5.  private Vertice endVertice;
6.  public System.Collections.ArrayList viaEdges;
7.  public Path() { }
8.  public void SetStartVertice(object vertex)
9.  { startVertice = (Vertice)vertex; }
10.  public void SetEndVertice(object vertex)
11.  { endVertice = (Vertice)vertex; }
12.  public void AddEdge(Edge edge)
13.  { viaEdges.Add(edge); }
14.  public double GetLength()
15.  { double length =0;
16.  System.Collections.IEnumerator viaEdgesEnumerator = viaEdges.GetEnumerator();
17.  while(viaEdgesEnumerator.MoveNext())
18.  {
19.  length += ((Edge)viaEdgesEnumerator.Current).LinkedVertice.RoadLength;
20.  }
21.  length = length+startVertice.RoadLength-endVertice.RoadLength/2;
22.  return length;
23.  }
24.  public string getPathInstruction()
25.  { string instruction = null ;
26.  System.Collections.ArrayList viaEdgesReformed = new System.Collections.ArrayList(10);
27.  if(startVertice != null)
28.  viaEdgesReformed.Add(new Edge(startVertice.Clone(),Direction.RightOrLeft.Right,Direction.Orientation.North));
29.  else
30.     return "Can't find the best Path.";
31.
32.  if(viaEdges.Count>0)
33.  { for(int i =0 ;i< viaEdges.Count ;i++)
34.     {if(((Edge)viaEdgesReformed[viaEdgesReformed.Count -1]).LinkedVertice.FatherRoad ==
((Edge)viaEdges[i]).LinkedVertice.FatherRoad)
35.     {((Edge)viaEdgesReformed[viaEdgesReformed.Count -1]).LinkedVertice.RoadLength +=
((Edge)viaEdges[i]).LinkedVertice.RoadLength;
36.     }
37.     else
38.     {viaEdgesReformed.Add(((Edge)viaEdges[i]).Clone());}
39.     }
40.          instruction += "Via " + ((Edge)viaEdgesReformed[0]).LinkedVertice.RoadName + ", drive " + ((Edge)viaEdgesReforme-
d[0]).LinkedVertice.RoadLength + " miles, ";
41.     for(int i =1 ;i < viaEdgesReformed.Count ; i++)
42.     {   string rol;
43.     switch(((Edge)viaEdgesReformed[i]).RightOrLeft.ToString())
44.        {
45.
46.           case "Straight": rol = " Keep straight,\r\n"; break;
47.        case "Right": rol = " Turn right,\r\n"; break;
48.        case "Left": rol = " Turn left,\r\n"; break;
49.        default: rol = " Keep straight,\r\n"; break;
50.     }
51.     if( i != viaEdgesReformed.Count -1)
52.     {instruction += rol + "Via " + ((Edge)viaEdgesReformed[i]).LinkedVertice.RoadName + " drive " + ((Edge)viaEdgesReforme-
d[i]).LinkedVertice.RoadLength + " miles,"; }
53.     else
54.     {instruction += rol + "Via " + ((Edge)viaEdgesReformed[i]).LinkedVertice.RoadName + " drive " + ((Edge)viaEdgesReforme-
d[i]).LinkedVertice.RoadLength / 2 + " miles,Arrive at destination.";}
55.  else { instruction = "Can't find the best Path.";}
```

Pseudocode 9: Continued.

```
56.            return instruction;
57.   }
58. }// END CLASS DEFINITION Path
59.}
```

Pseudocode 9

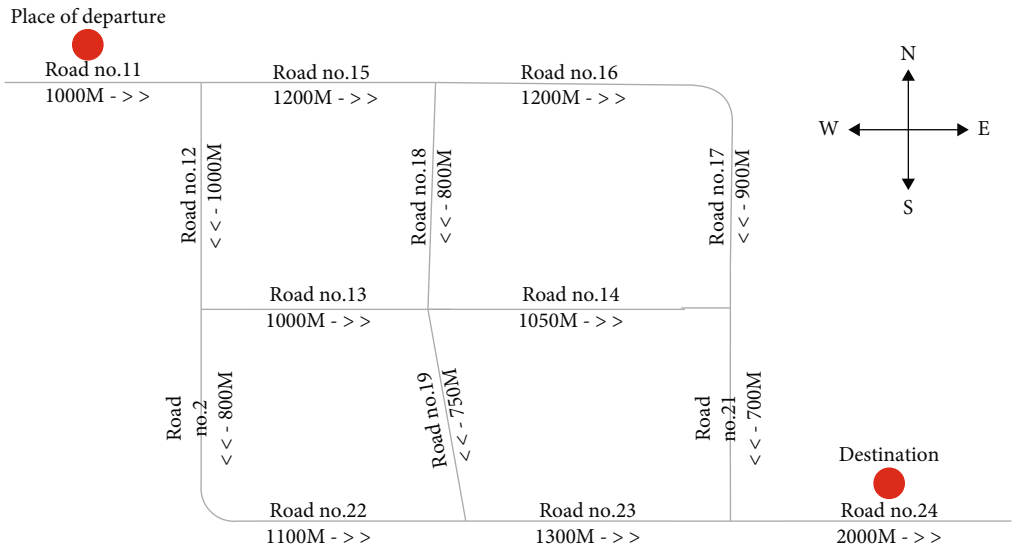

Figure 3: Test road network diagram.

Table 3: Test data of the road network topology table.

WINDOWS-2S8TAV..._CarWayNetwork				
ConnectionID	RoadID	LinkedRoadID	RightOrLeft	Orientation
9	18	14	2	11
10	12	20	0	12
11	13	19	1	12
12	20	22	2	11
13	22	23	0	11
14	19	23	2	11
15	21	24	2	11
16	23	24	0	11
17	18	19	0	12
1	11	12	1	12
2	12	13	2	11
3	11	15	0	11
4	13	14	0	11
5	14	21	1	12
6	15	16	0	13
7	16	17	1	12
8	15	18	1	12

Table 4: Test data of the road information table.

WINDOWS-2S8TAV...dbo.XM_CarRoads			
RoadID	Length	Name	FatherRoadID
20	800	Road No. 20	20
21	700	Road No. 21	21
22	1100	Road No. 22	22
23	1300	Road No. 23	23
24	2000	Road No. 24	24
11	1000	Road No. 11	11
12	1000	Road No. 12	12
13	1000	Road No. 13	13
14	1050	Road No. 14	14
15	1050	Road No. 15	15
16	1200	Road No. 16	16
17	900	Road No. 17	17
18	800	Road No. 18	18
19	750	Road No. 19	19

4.3. Test Program and Running Results

4.3.1. Test Program. Under the principle of system design and implementation, test program is written based on the test data to test the road network path search system. The test program sets the starting point at the midpoint of Road No. 11 and the ending point at the midpoint of Road No. 24. The code of test program is as follows.

4.3.2. Running Results of the Test Program. The result of running the test program is shown in Figure 4. The system calculates the total length of the best path, returns the correct car navigation instructions, and concludes that the total length of the shortest path is 5600 meters, and the road

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.  using System.Data.SqlClient;
6.  using JGXY.LBS.Route;
7.  namespace TestRouteConsole
8.  {       class Program
9.      {
10.         static void Main(string[] args)
11.         { try
12.             { RoadNetwork rn = new RoadNetwork();
13.                 SqlConnection conn = new SqlConnection("database=Route;server=localhost;user = sa2; password =sa123[];Connect
Timeout=30");
14.                 RoadNetworkBuilder rnb = new RoadNetworkBuilder(conn);
15.                 rnb.addRoadGraphToNetwork("XM", "Car", rn);
16.                 Vertice start = new Vertice("11", "Road No.11", 500, "11");
17.                 Vertice end = new Vertice("24", "Road No.24", 1000, "24");
18.                 RouteSearch routeSearch = new RouteSearch((RoadGraph)rn.NetworkHash["RT_Car"]);
19.                 Path path = routeSearch.getShortestPath(start, end);
20.                 Console.WriteLine("Total path length:" + path.GetLength().ToString() + "Meters ");
21.                 Console.WriteLine("Best path navigation:" + path.getPathInstruction());
22.                 Console.Read();
23.             }
24.             catch (Exception ex)
25.             { Console.WriteLine(ex.Message);       }
26.         }
27.     }
28.  }
```

Pseudocode 10

Figure 4: The running results of the test program.

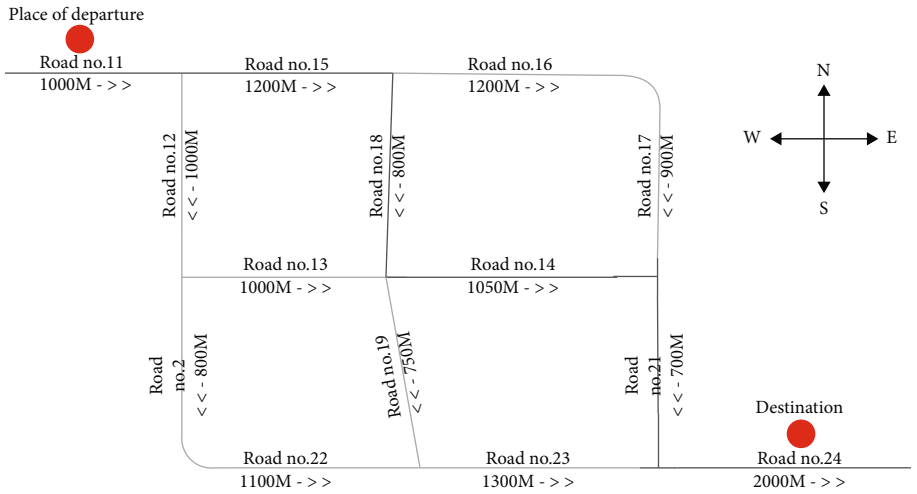

Figure 5: The optimal path diagram.

sequence is No. 11, No. 15, No. 18, No. 14, No. 21, and No. 24.

According to the car navigation instructions of the test program, we blacken the related roads to get the best path diagram shown in Figure 5, which proves that the optimal path computed by the road network path search system is correct.

5. Discussion

The road network path search system is widely used today. This paper designs a complete software architecture of the road network path search system through the comprehensive use of database, programming language, shortest path algorithm, UML language, software design pattern, and other methods and realizes the complete function of the system by Microsoft C#.NET programming language.

The system architecture designed in this paper has good scalability, flexibility, and pluggability. The code of the system uses Dijkstra as the shortest path search algorithm. If you want to use other shortest algorithms, you can replace the code of BestRoute class without affecting the design and function of the entire system.

According to the test program result, we can search the road network by the system designed in this paper to get the best route navigation information.

The system can be deployed on Microsoft Windows operation systems or a cloud server. Based on the map display and GPS functions of the GIS system, our system can realize the function of real-time map navigation.

The system can also be used as a path search module for UAVs, self-driving cars, logistics robots, and other applications to realize the path finding function. Furthermore, cooperating with the GPS, infrared equipment, sensors, cameras, and other devices, our system can be used to realize the automatic navigation function of the machine.

The future work is to enable this system to support more platforms and application scenarios, compatible with more computer languages, so as to improve the efficiency of the algorithm, and support more shortest path algorithms.

6. Conclusion

This paper investigates the architecture design and code implementation of the road network path search system, including system function design, database design and implementation, UML design, code implementation, application method, and system test. The system completely and reliably implements the function of road network path search. Moreover, the system can be studied as a software prototype and its architecture design and code can be applied to various GIS, GPS, robots, unmanned aerial vehicles, logistics robots, and other systems that require path search function.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] K. Seo, J. Ahn, and D. H. Im, "Optimization of shortest-path search on RDBMS-based graphs," *ISPRS International Journal of Geo-Information*, vol. 8, no. 12, p. 550, 2019.

[2] X. Fu, Z. M. Cheng, and J. X. Wang, "Research on online scheduling and charging strategy of robots based on shortest path algorithm," *Computers & Industrial Engineering*, vol. 153, article 107097, 2021.

[3] M. Asaduzzaman, T. K. Geok, F. Hossain et al., "An efficient shortest path algorithm: multi-destinations in an indoor environment," *Symmetry*, vol. 13, no. 3, p. 421, 2021.

[4] D. Garg, "Dynamizing Dijkstra: a solution to dynamic shortest path problem through retroactive priority queue," *Journal of King Saud University-Computer and Information Sciences*, vol. 33, no. 3, pp. 364–373, 2021.

[5] A. Ebrahimnejad, "An acceptability index based approach for solving shortest path problem on a network with interval weights," *RAIRO: Recherche Opérationnelle*, vol. 55, p. 1767, 2021.

[6] Q. Tu, L. Cheng, T. F. Yuan, Y. Cheng, and M. M. Li, "The constrained reliable shortest path problem for electric vehicles in the urban transportation network," *Journal of Cleaner Production*, vol. 261, article 121130, 2020.

[7] L. S. Liu, J. F. Lin, J. X. Yao et al., "Path planning for smart car based on Dijkstra algorithm and dynamic window approach," *Wireless Communications & Mobile Computing*, vol. 2021, article 8881684, 12 pages, 2021.

[8] T. Song, X. Huo, and X. Wu, "A two-stage method for target searching in the path planning for mobile robots," *Sensors*, vol. 20, no. 23, p. 6919, 2020.

[9] O. S. Oubbati, M. Atiquzzaman, P. Lorenz, A. Baz, and H. Alhakami, "SEARCH: an SDN-enabled approach for vehicle path-planning," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 14523–14536, 2020.