

XML 数据公交信息查询优化算法及实现

刘明珠, 丁亦楠, 郑云非

(哈尔滨理工大学 测控技术与通信工程学院, 黑龙江 哈尔滨 150080)

摘 要: 针对公交线路规划的问题, 必须提供一个准确快捷的公交查询系统以满足人们日常出行的需求. 研究了基于 XML 数据的公交查询系统, 该系统采用 B/S 模式, 利用 ASP.NET 框架和 C# 语言, 实现了公交运行查询功能. 在换乘查询算法部分及在 Dijkstra 算法的基础上, 分析了路径寻优的原理, 实现了路径距离计算的具体方法. 并通过减少临时节点排序及数量的方式, 改进了 Dijkstra 算法, 最终减少了寻找路径的时间并简化了路径计算. 最后, 以天津市公交数据为例, 用改进的 Dijkstra 算法对公交查询系统进行了分析验证, 结果表明, 利用改进的 Dijkstra 算法可以实现高效的公交信息查询, 节约查询时间, 节省内存资源.

关键词: 公交查询; Dijkstra 算法; XML 数据; 路径寻优; 最短路径

DOI: 10.15938/j.jhust.2015.02.016

中图分类号: TP319 **文献标志码:** A **文章编号:** 1007-2683(2015)02-0085-06

Algorithm Optimization and Realization on Public Transportation Information Inquiry Based on XML Data

LIU Ming-zhu, DING Yi-nan, ZHENG Yun-fei

(School of Measure-control Technology and Communication Engineering, Harbin University of Science and Technology, Harbin 150080, China)

Abstract: For the complex and cumbersome problems of bus lines, an accurate and efficient public transportation information inquiry system must be provided to meet people's daily travel needs. This public transportation information inquiry system is based on XML data. By using B/S mode, ASP.NET framework and the C# language, a practical public transportation information inquiry system has been built. Firstly, this paper analyzed the path optimization principles and achieved a specific method to calculate the path distance based on Dijkstra's algorithm. And then, a improving Dijkstra algorithm has been provided by reducing the number and order of temporary nodes to reduce the time occupation for finding the paths and simplified the calculated amount. Finally, as an example, the improved Dijkstra algorithm has been used in Tianjin public transportation information inquiry system, the result indicates that using the new algorithm can not only shorten time occupation for query but also save the internal memory.

Keywords: public transportation inquiry; Dijkstra algorithm; XML data; path optimization; shortest path

收稿日期: 2014-08-28

作者简介: 刘明珠(1973—), 女, 副教授, 硕士研究生导师, E-mail: lmz@hrbust.edu.cn.

丁亦楠(1989—), 男, 硕士研究生;

郑云非(1991—), 男, 硕士研究生.

0 引言

随着现代城市规模发展,城市的公交线路也越来越复杂,准确又方便地查阅公交线路,已经成为与人们生活息息相关的问题.传统的公交查询系统大多使用 Sybase SQL Server^[1]、Oracle^[2] 等传统数据库系统保存公交信息数据,并运用 (structured query language SQL) 语言实现对这些数据库的操作^[3]. SQL 语法简单,语言语句可嵌套,这使它具有极大的灵活性,但是,对于管理者的要求较高.目前,利用 XML 文件的结构化标记数据和可移植性的特点,构建公交查询系统后台数据结构,以减轻系统管理者负担,并提供准确、快捷的公交线路查询系统成为智能公交系统的研究热点之一^[4]. XML 即可扩展标记语言,它使用表示起始和结束标签的语法来描述和交换独立于应用程序的结构化数据^[5].使用 XML 存储数据使得管理者不再需要使用传统的 SQL 语言来更新管理数据,减轻了管理人员的负担^[6].本文首先对 Dijkstra 算法进行改进,并在 Visual Studio 开发环境下,利用 ASP. net 框架,采用 B/S 模式,构建 XML 数据,并将线路与站点数据相结合,同时融合拐点数据,使得线路查询算法更加简单,为公交查询系统的数据管理提供了一种新方法.

1 系统需求描述

1.1 性能需求

一个好的公交查询系统应具有准确性,快速性,易扩展性和易使用性^[7].它保证系统输出结果的准确性,避免出现重复或者错误的查询结果给用户造成误导.并且在数据的导入、导出和查询过程中数据的处理时间响应上不超过 1s.系统利用的 XML 数据非常便于管理者在后台随时进行更新和修改,即可直接打开 XML 数据文件进行手动修改.同时查询者无需进行登录即可进行公交查询,系统配备合理的人机交互界面,方便用户查询和使用,节约了用户查询时间.

1.2 功能需求

公交查询系统功能需求结构如图 1 所示.

1) 线路查询: 包含了线路的上、下行信息,并涵盖了线路经过的所有站点信息.当用户输入的信息不明确或不详细时,系统自动进行模糊查询,为用户匹配需要的线路信息.

2) 站点查询: 此功能在用户进行查询时,将包

含此站点的所有公交线路反馈给用户,若用户输入的站名不明确时,系统自动进行模糊查询,为用户反馈合适的公交线路信息.

3) 乘车路线查询: 本系统以较少换乘次数作为设计依据设计了零换乘站点查询和一次换乘查询.首先输出零换乘的查询结果,其次输出一次换乘的查询结果,给查询者带来方便.

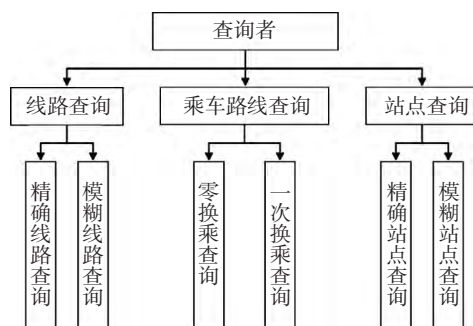


图1 系统功能需求框架结构

2 Dijkstra 换乘算法描述及其优化

Dijkstra 算法^[8]以广度优先算法为基础,主要解决从固定的单个点源,经过若干点后,到其他固定顶点的最短路径问题^[9-11].本系统换乘查询部分采用了改进的 Dijkstra 换乘算法.

2.1 基于传统 Dijkstra 算法的换乘查询算法

在本文设计的公交查询系统中,一次换乘查询采用了 Dijkstra 算法,具体过程如下:

Step1. 设起点为站点 A,设终点为站点 C,其它的站点为站点 B. 设从站点 A 到站点 B 的集合为 $U(A, B)$ 距离为 ΔA , 若站点 B 为中转站,则设 $\Delta A = 0$; 若站点 B 非中转站则 $\Delta A = \infty$.

Step2. 计算出 ΔA 的具体数值,对 ΔA 的值排序,并从 ΔA 最小值开始暂定其为 ΔA 的值,暂定该 ΔA 下的站点 B 为最佳中转站点.

Step3. 设从站点 B 到站点 C 的集合为 $U(B, C)$, 距离为 ΔB , 则从站点 A 到站点 C 的距离为 $\Delta C = \Delta A + \Delta B$. 对 ΔC 的值和对应的换乘数据进行保存.

Step4. 重复 step2 和 step3 直到所有 ΔA 为最大值.

Step5. 对所有已保存的 ΔC 的值进行排序找到从站点 A 到站点 C 的最短距离和相对应的换乘数据.

该过程基于传统 Dijkstra 算法可以实现一次换乘查询,提供准确结果^[12].但由于遍历计算节点较

多,并产生大量中间数据,导致其效率不高^[3],算法时间得不到优化。

2.2 传统 Dijkstra 换乘算法优化原理

基于传统的 Dijkstra 换乘算法,对于一个公交查询系统,一般来说有以下两种常规优化方案:

1) 传统优化方案一: 该 Dijkstra 算法优化方案根据在查询中转站时,对中转站设置一定的权值来实现,这样做的好处是可以对换乘的次数进行排序,优先选择少换乘的路线,之后再对换乘路线的距离进行计算并排序。但是,这种方案当换乘次数增多时,会产生大量的查询线路无用结果,并且由于权值的加入,最终导致运算结点的增加,使得运算次数增多,不具备一定的实用性,虽然针对于传统的 Dijkstra 算法进行了一定优化,但并不是最佳优化方案。

2) 传统优化方案二: 该 Dijkstra 算法优化方案在计算起点 A 到中转站 B 的距离 ΔA 时只计算站点 B 为中转站的距离,然后直接计算站点 B 到站点 C 的距离 ΔB ,最后依据 $\Delta C = \Delta A + \Delta B$ 直接计算全程线路的距离 ΔC 的值后再对 ΔC 排序。这样做减少了中途排序的过程,节约了大量计算机内存。但是这种方案在查找同一组线路下的中转站会出现中转站重复的情况,进而增加中途结点,没有实现算法完全优化。

2.3 本系统 Dijkstra 换乘算法优化原理

在本系统中,传统的 Dijkstra 算法在计算过程中产生大量的 0 元素, ∞ 元素以及中间数据,这样做会占用较多计算资源^[4],且该算法的时间复杂度为 $O(n^2)$ ^[5-17],当数据量过大时,中途的结点数据量会占据大量的计算机内存单元,算法运行的时间将会不可接受,因此,需要对传统的 Dijkstra 换乘算法进行优化,使之减少资源占用,减轻计算复杂度。优化实现的具体过程为:

Step1. 设起点为站点 A,终点为站点 C。找出包含 A 的线路 $L(A)$ 和包含站点 C 的线路 $L(C)$ 。

Step2. 分别求出线路 $L(A)$ 和线路 $L(C)$ 的所有站点,找出 $L(A)$ 与 $L(C)$ 线路站点中包含的相同站点,即为中转站点 B。若查找出的相同站点包含重复的 $L(A)$ 与 $L(C)$ 线路,则停止重复查找,直接查询下一条线路的相同站点。

Step3. 直接计算站点 B 到站点 C 的距离 ΔB 。

Step4. 依据 $\Delta C = \Delta A + \Delta B$ 直接计算 ΔC 的值,再对 ΔC 排序。

这样做与传统的 Dijkstra 算法过程相比,避免了中途积累大量的结点数据,并减少了中途排序的过程,当算法在运算过程中产生 n 个结点数时,优化算

法的时间复杂度减少为 $O(n)$,这样节约了大量计算机内存并节省了大量运算时间,并提高了查询速度。

2.4 Dijkstra 换乘算法优化实验验证结果

为了验证本系统的 Dijkstra 换乘算法优化的搜索效率,正确性以及优化效率,本文对传统 Dijkstra 算法,传统 Dijkstra 算法优化方案一,优化方案二和本系统的 Dijkstra 优化算法分别进行了实验,并进行了算法的对比验证。

在进行实验验证时,首先选取 10 组不同的起点和终点数据,然后使用这 10 组数据进行线路查询,在换乘线路查找过程中,分别记录使用这 4 种算法方案所产生的平均节点数及算法的搜索效率,并验证输出结果的正确性,最终分析说明优化 Dijkstra 换乘算法相比其它 3 种算法方案的提升效率,各项比较结果如表 1 所示。

表 1 传统 Dijkstra 换乘算法与优化 Dijkstra 换乘算法在换乘线路查询下的优化结果对比

使用算法	平均节点数	搜索效率%
传统 Dijkstra 换乘算法	103	67
传统 Dijkstra 优化方案(1)	47	78
传统 Dijkstra 优化方案(2)	22	89
系统 Dijkstra 优化换乘算法	15	92

从表 1 中可以看出,使用优化后的 Dijkstra 换乘算法过程中产生的节点数明显减少,并且优化提升明显,相比其他 3 种算法方案,本系统使用的 Dijkstra 优化换乘算法很好的节省了内存占用单元和查询时间,满足了用户需求。

3 基于 XML 数据的公交信息查询优化算法的实现

3.1 XML 数据模块设计说明及程序实现

XML 数据设计直接影响系统的性能以及管理员的维护^[8,19]。XML 数据将线路与站点数据相结合,并且融合了线路拐点数据,所谓拐点即公交按其线路行走时需要拐弯的点。融合拐点数据后,使站点距离计算更加精确,本系统的 XML 数据模块如下所示:

```
<? xml version = "1.0"? >
< lines >
< line name = " " >
< stations >
< station name = " " longitude = " " latitude
= " " / >
< /stations >
```

</line>
</lines>

由于 XML 文档可以被表示为标签在节点上的树^[2], 则通过图 2 表示出该 XML 文档的数据模式图. 通过这些数据可以查询乘车线路, 并计算出两站点之间的距离, 并且非常便于管理员在后台随时更新和维护该 XML 数据.

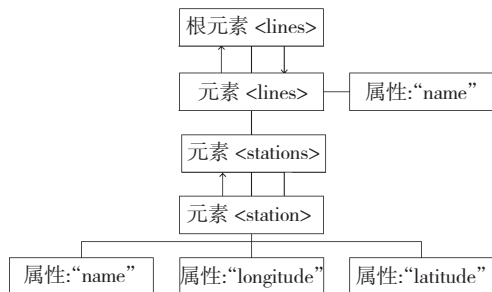


图 2 XML 文档数据模式图

在此模块中:

主要包含 3 个元素, 分别为: station 元素、line 元素及 lines 元素.

1) 标签 <stations> </stations> 中存储了该条线路的所有站点和拐点. 其中, 元素 name 表示站点名称, 若该点为拐点, 则 name 为空, 元素 longitude 表示该站点(拐点)的经度, 元素 latitude 表示该站点(拐点)的纬度. 用 C# 进行数据传入时, 首先建立 XSpot 类, 类中定义了用来存储线路名称的字符串变量和用来存储坐标的坐标变量, 用 XSpot 类定义一个 spot 自定义类型, 将每条线路的站点数据分别存入其中.

2) 标签 <line> </line> 中存储了一条线路和该线路所包含的所有站点和拐点. 其中, 元素 name 表示线路名称.

3) 标签 <lines> </lines> 中存储着所有线路, 在用 C# 进行数据传入时, 首先建立 XLine 类, 类中定义了用来存储线路名称的字符串变量和用来存储线路包含的站名名称及站点信息的 XSpot 变量. 用 XSpot 类定义一个 spots 自定义类型, 将 spot 内的每条线路站点信息全部存入其中, 整合为全部线路的全部站点名称, 并用 XLine 类定义一个 line 自定义类型, 将标签内的数据全部存入 line 中, 其中包含每条线路名称 name 和对应的站点及站点坐标 spots.

3.2 计算站点距离的数学模型描述

1) 当用户从站点 A_1 到站点 A_m 时, 中途要经过 $m-2$ 个站点及拐点, 分别设站点为 $A_2, A_3, \dots, A_{m-2}, A_{m-1}$.

2) 设站点 A_1 的经度为 x_1 , 站点 A_1 纬度为 y_1 ; 站点 A_2 经度为 x_2 , 站点 A_2 纬度为 y_2 , 则站点 A_1 与站点 A_2 的经度差为

$$X_1 = x_2 - x_1, \quad (1)$$

它们的纬度差为

$$Y_1 = y_2 - y_1, \quad (2)$$

3) 由于地面上经度或纬度相差 1 度以内对应的地表弧长大约是 $S = 111\text{km}$, 且站点与站点间的距离都小于 1 经度或 1 纬度. 因此, 站点 A_1 与站点 A_2 之间的距离为

$$L_1 = S \cdot \sqrt{X_1^2 + Y_1^2}, \quad (3)$$

同理, 站点 A_2 与站点 A_3 之间的距离为

$$L_2 = S \cdot \sqrt{X_2^2 + Y_2^2}, \quad (4)$$

站点 A_{m-1} 与站点 A_m 之间的距离为

$$L_{m-1} = S \cdot \sqrt{X_{m-2}^2 + Y_{m-2}^2}, \quad (5)$$

4) 由此可知, 从站点 A_1 到站点 A_m , 即用户的乘车所经过的距离为

$$L = \sum_{i=1}^{m-1} L_i = \sum_{i=1}^{m-1} S \cdot \sqrt{(x_{i+2} - x_i)^2 + (y_{i+2} - y_i)^2} = \sum_{i=1}^{m-1} S \cdot \sqrt{X_i^2 + Y_i^2}. \quad (6)$$

3.3 基于优化 Dijkstra 乘车查询算法的实现

本系统乘车查询实现分为零换乘与一次换乘两部分.

3.3.1 零换乘

首先应用 C# 的 List 容器类以及 System.Xml 控件将 XML 后台数据中的线路信息与站点拐点的信息传入程序中, 并使用 List 类构建 lines1 和 lines2 两个新容器. 设用户查询的起点站为 station1, 终点站为 station2, 查找包含 station1 的所有线路并将其存入容器 lines1 中, 同时将包含 station2 的所有线路存入 lines2 中. 之后, 使用 List 类构建 line 容器来保存共同包含 station1 与 station2 的线路, 此线路即为用户零换乘所需要的乘车线路, 之后利用站点距离数学模型及 C# 语言实现可计算出 station1 到 station2 的距离. 在 C# 中, 求两个 List 的交集, 可以通过调用 List 类中的 Intersect 方法 "line = lines1.Intersect(lines2).ToList();" ; 利用 C# 容器类的这个特点可简单有效的求出用户所需的乘车线路, 再通过索引找出 station1 及 station2 分别在 line 中的位置, 即可求出用户乘坐该线路所要经过的站点及距离. 最后, 利用 C# 语言提供的接口 IComparer, 建立内部继

承类,并利用该接口中提供的 CompareTo() 函数对类中的成员进行比较,通过这个方法,对 line 中的线路路径长度进行比较后排序并将结果存入新容器 lines 中,可得出零换乘从路径最短到路径最长的查询结果(流程图如图3左边所示)。

3.3.2 一次换乘

用 C# 语言实现时,首先设起点站为 station1,终点站为 station2,利用 Find() 函数,找出所有包含 station1 的线路 line1 中以及所有包含 station2 的线路 line2,并用索引标记出 station1, station2 分别在 line1 和 line2 中的位置。其次,利用判断中转站的方法 "line1.spots[i].GetName() == line2.spots[j].GetName()" 来找出 line1 和 line2 中的共同站点,即换乘站点,其中,spots 代表线路的站点,GetName() 用来获取站点中的线路名称。由于在相同 line1 与 line2 间一般会有多个换乘站点,因此,先通过对 station1 与 station2 到换乘站点距离分别进行计算,求出 station1 到 station2 的距离,再取出 station1 到 station2 在相同 line1 与 line2 下的最短距离,最终只保留相同 line1, line2 下 station1 到 station2 的最短路径结果,并保存在 List 容器 path 中,再进行下一组 line1, line2 的判断。最终,将保留的换乘查询结果排序并利用 lines.Add(path); 全部保存在 List 容器 lines 中,即为一次换乘的乘车线路(流程如图3右边所示)。

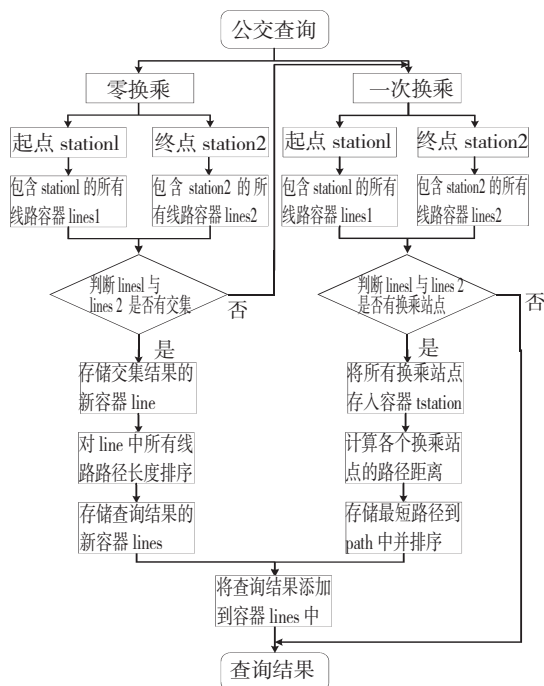


图3 公交查询系统流程图

3.4 查询结果的输出

本系统通过使用天津市公交信息数据,最终实现了线路查询、站点查询、乘车线路查询三大功能。

1) 线路查询: 在该方面本系统以“101路”线路作为实验数据,在进行线路查询时,输入“101”,通过模糊查询来找到101路公交的上下行线路所有信息,查询结果如图4所示。



图4 线路查询输出结果

2) 站点查询: 在站点查询的方面,本系统以“天津站”为例,查询了经过天津火车站的所有公交线路,查询结果如图5所示。

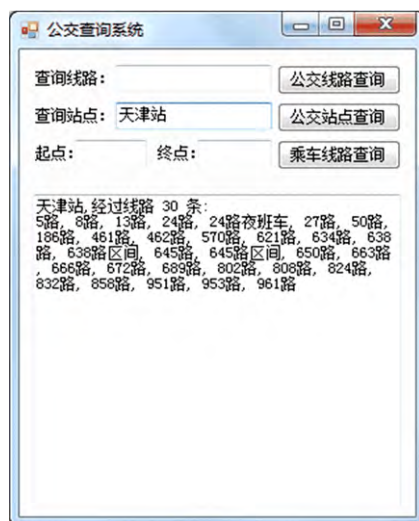


图5 站点查询输出结果

3) 乘车路线查询: 在乘车路线查询结果输出方面,系统根据人们出行时的乘车心理,首先选择了输出零换乘的查询结果,当零换乘的查询结果较少或

没有零换乘的乘车路线时,系统将继续自动进行一次换乘查询,并将结果输出.同时,为了保证系统结果简明扼要,在进行线路查找时,添加了限制语句,令输出的乘车线路结果保持在5条以内,既当容器lines中存储的线路个数达到5条时,系统自动停止线路查询,这样既减少了搜索乘车线路所花费的时间,又给用户提供了快捷方便的结果.系统以起点为“天津站”,终点为“天津西站”作为实验依据,对该系统进行了实验验证,乘车线路的查询结果如图6所示.

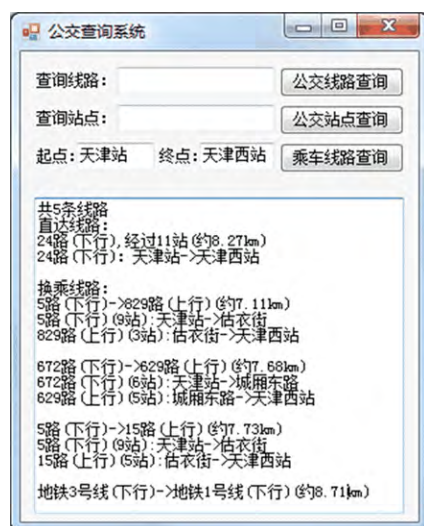


图6 乘车路线查询输出结果

4 结 语

本系统首先通过使用XML数据,进而实现对传统Dijkstra换乘算法的改进,并将结果与传统Dijkstra换乘算法和传统Dijkstra换乘优化方案进行比较,验证了本系统算法的优越性,最终构建了一个对于用户方便使用,对于管理员方便管理的简便实用的公交查询系统.实验表明了利用改进的Dijkstra算法可以实现高效的公交信息查询,节约了查询时间,节省了内存资源.该系统对于传统Dijkstra换乘算法的优化为公交查询系统提供了一种新的解决思路.

参 考 文 献:

- [1] MCGOVERAN D. Guide to SYBASE and SQL Server [M]. Addison-Wesley Longman Publishing Co., Inc., 1999.
- [2] ABRAMSON I, ABBEY M, COREY M J, et al. Oracle Database 11g [M]. McGraw-Hill, 2009.
- [3] 李洪涛. 基于数据库的公交换乘算法的实现与优化 [J]. 软件

导刊, 2008(2): 133-135.

- [4] 任杰, 柳春华. 基于XML的公交线路数据系统查询设计 [J]. 计算机光盘软件与应用, 2012, 12: 181-183.
- [5] ZHOU Chuan-Sheng. Research on XML Based Flexible web Application Design [J]. Applied Mechanics and Materials, 2013 (1): 1661-1666.
- [6] 曹毅, 吴玲达, 崔亮. 基于关系数据库的半结构化XML数据组织技术 [J]. 计算机工程与设计, 2014(10): 3472-3479.
- [7] 吴永军, 蔡永香, 郭庆胜. 城市公交查询系统的设计与实现 [J]. 测绘信息与工程, 2006(5): 40-42.
- [8] Wikipedia, Dijkstra's algorithm. http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm [EB/OL], 2014.
- [9] 王海帅, 冀振燕, 王森. 公交线路查询算法 [J]. 计算机系统应用, 2013(2): 88-91.
- [10] 李臣波, 刘润涛. 一种基于Dijkstra的最短路径算法 [J]. 哈尔滨理工大学学报, 2008, 13(1): 35-37.
- [11] 鄢章华, 滕春贤. 基于数据库的Dijkstra算法的改进与实现 [J]. 哈尔滨理工大学学报, 2008, 13(2): 32-34.
- [12] WANG Shu-Xi. The Improved Dijkstra's Shortest Path Algorithm and Its Application [J]. Procedia Engineering, 2012, 29: 1186-1190.
- [13] 刘建美, 马寿峰, 马帅奇. 基于改进的Dijkstra算法的动态最短路径计算方法 [J]. 系统工程理论与实践, 2011(6): 1153-1157.
- [14] YOUSEFI Saleh, ABBASI Tofigh, ANVARI Zahra. 基于车载通信实时交通信息的城市道路路径选择方法 [J]. 交通运输系统工程与信息, 2014(5): 23-36.
- [15] 张钟, 吕敏, 孙广中. 一种求解点到点的最短路径的高效下界算法(英文) [J]. 中国科学技术大学学报, 2014(10): 874-880.
- [16] RIGDON Edward E. Rethinking Partial Least Squares Path Modeling: Breaking Chains and Forging Ahead [J]. Long Range Planning, 2014(3): 161-167.
- [17] GUO Jinchao, GAO Yu, CUI Guangzhao. The navigation of mobile robot based on hybrid Dijkstra algorithm [J]. Journal of Computational Information Systems, 2014(9): 3879-3886.
- [18] CHEN Liang Jeff. Mapping XML to a wide sparse table [J]. IEEE Transactions on Knowledge and Data Engineering, 2014 (6): 1400-1414.
- [19] 李洪安. XML设计与规范化研究 [J]. 西北大学, 2009.
- [20] 刘春阳, 宋雷雄, 郑雪峰, 等. 基于XML树-表结构的多元巡检数据存储方法 [J]. 计算机仿真, 2014: 133-136.

(编辑: 王 萍)