基于 OpenStreetMap 最短路径算法的分析与实现

张英辉,张水平,张凤琴,王 蓉
(空军工程大学 信息与导航学院,陕西 西安 710077)

摘 要: 随着计算机网络技术和地理信息科学的发展,最短路径问题无论是在交通运输,还是在城市规划、物流管理、网络通讯等方面,都发挥了重要的作用。文中旨在阐述如何基于 OSM 运用 Dijkstra 算法计算两联通节点之间的最短路径。首先介绍了开放式 OSM 的特点以及地图数据文件中道路图像元素的数据结构;然后运用正则表达式算法从 OSM 数据中提取出交通道路信息,并选择合适的结构进行存储;最后通过将道路信息抽象成路径拓扑图,并以道路的地理距离作为路径权值,运用 Dijkstra 最短路径算法求解出两连通节点之间的最短路径。

关键词: 最短路径算法;开放街道地图;地理信息系统;正则表达式
中图分类号: TP301.6 **文献标识码:** A **文章编号:** 1673 - 629X(2013)11 - 0037 - 05
doi:10.3969/j.issn.1673 - 629X.2013.11.010

Analysis and Implementation of Shortest Path Algorithm Based on OpenStreetMap

ZHANG Ying - hui,ZHANG Shui - ping,ZHANG Feng - qin,WANG Rong
(College of Information and Navigation,University of Air Force Engineering,Xi'an 710077,China)

Abstract: With the rapid development of computer network and GIS technology,the shortest path problem plays an important role in transportation,city planning,logistics management,network communication,etc. Focus on describing how to use Dijkstra algorithm to calculate the shortest path between two connected nodes based on OSM. Firstly introduce the characteristics of OSM and data structure of road image feature. Secondly,extract road information by regular expression from OSM data file and store them in appropriate construction. Finally,form road topological diagram and takes the geographical distance as road weight,calculating the shortest path between two connected nodes by Dijkstra algorithm.

Key words: shortest path algorithm; OSM; GIS; regular expression

0 引 言

随着计算机的普及以及地理信息科学的发展,地理信息系统(Geographic Information System,GIS)因其强大的功能得到日益广泛和深入的应用,它在电子导航、交通旅游、城市规划以及电力、通讯等各种管网、管线的布局设计中发挥了重要的作用。在 GIS 中经常遇到的一个问题就是最短路径的搜索,最短路径可以指地理位之间的最短距离,也可以指时间的最短、费用最少等,它们的核心算法都是最短路径算法。要实现最短路径的搜索,首先应从地理信息数据中提取出交通道路和交汇点[1]的信息,将其抽象成图模型,然后运用最短路径算法求解最短路径,最后将求解结果绘制到 GIS 地图上。目前,最经典的最短路径算法是 Dijkstra[2]算法,它是多数系统解决最短路径问题采用的理论基础。

1 OSM 简介

Open Street Map(OSM)数据库是一套依靠全球用户的参与形成的公共地图数据服务[3],它如今已经是全球范围内最精确和完善的矢量地理数据集[4]之一。与传统的 GIS 和地图数据提供商相比,通过用户自发提供 GPS 设备记录的轨迹数据、基于影像数据的栅格矢量化数据及机构或公司捐助的矢量数据等作为数据来源的 OSM,在数据和地图服务[5]的获取方式上更加便捷(完全免费)、使用更加方便、具有更高的实效性。

作为系统的核心,OSM 后台数据可以被任何人所

收稿日期: 2013 - 02 - 06 **修回日期:** 2013 - 05 - 12 **网络出版时间:** 2013 - 08 - 28
基金项目: 空装计划项目(KJ2012197)
作者简介: 张英辉(1988 -),男,河北唐山人,硕士研究生,研究方向为分布式数据库、云计算;张水平,教授,研究方向为分布式数据库、云计算。
网络出版地址: http://www.cnki.net/kcms/detail/61.1450.TP.20130828.0829.016.html

使用和编辑。用户也可提取数据渲染[6]自定义样式的地图。所有的OSM服务都建立在用户输入的后台数据的基础之上,而信息丰富的地图产品是其自然结果。

1.1 OSM数据来源

OSM地图数据是对外公开的,任何人或组织均可自由下载使用。OSM提供了多种方式为用户提供地图数据下载功能。基于XML的.osm文件是Open Street Map的标准地图数据格式文件。可以通过以下几种方式获得OSM数据信息:

(1)利用JOSM进行下载。

JOSM是一个采用Java实现的OpenStreetMap(OSM)编辑器。它提供了一个良好的接口方便用户对指定区域进行数据下载并对数据进行渲染及可视化处理[7]。但该方法由于需要直接将数据加载到内存,因而数据是否下载成功与指定区域的地图数据量大小和本地计算机内存容量有关。

(2)通过HTTP请求。

使用OSM API及指定区域经纬度构成的包络矩形[8]构建URL,直接从浏览器通过连接地址下载指定范围的OSM数据,基本格式为:http://api.openstreetmap.org/api/0.6/map?bbox=11.54,48.14,11.543,48.145,其中bbox中经纬度顺序为最小的经度、最小的纬度、最大的经度、最大的纬度。这种方法简单实用,只要知道数据区域的经纬度范围,便可构建URL格式连接串下载OSM数据。

(3)通过OSM官网下载。

OSM完整数据集可在OpenStreetMap官方网站下载,也可以根据国家及区域下载,此外还可以使用Osmosis控制行工具从planet,oasm中裁剪数据。

(4)通过第三方OSM服务商网站下载。

有许多第三方的独立商业公司为OpenStreetMap提供各种商业服务。Cloudmad就是其中一家,它的网站上提供了多种OSM数据的下载。

1.2 OSM数据格式

OSM地理数据文件的存储方式是xml结构,后缀名可以是xml或osm,该数据文件既包含了地理数据,又包括了对元数据[8]的描述。每一地理元素都拥有对应的元数据,用于记录数据修改的时间、修改者账户号和作者名称等。OSM地理数据的描述采用的是一种包含拓扑[9]性质的数据结构,其中与道路相关的地理元素主要包括节点(Node)和道路(Way)。

(1)Node节点。

Node节点包括经纬坐标或高度信息。其他一些可选信息,如name等,在tag子数据中表示。以下就是Node的XML描述。

```
<osm version="0.6" generator="OpenStreetMap server">
    <node id="483034256" lat="55.9458449" lon="-3.2035477" version="1"
        changeset="2369219" user="spytfyre" uid="166957" visible="true"
        timestamp="2009-09-04T13:35:42Z">
        <tag k="name" v="The Blue Blazer" />
        <tag k="amenity" v="pub" />
    </node>
</osm>
```

(2)Way线或区域。

Way道路由2~2 000个点(nodes)所构成,Way可表示如下3种图形事物(非闭合线、闭合线、区域)。对于超过2 000 nodes的Way,可以通过分割来处理。

```
<osm version="0.6" generator="OpenStreetMap server">
    <way id="43157302" visible="true" timestamp="2009-10-26T10:45:09Z"
        version="1" changeset="2954960" user="Ed Avis" uid="31257">
        <nd ref="540653724" />
        <nd ref="25507043" />
        <nd ref="107762" />
        <nd ref="25507038" />
        <nd ref="107759" />
        <tag k="highway" v="primary" />
        <tag k="lcn_ref" v="6a" />
        <tag k="name" v="Parliament Street" />
    </way>
</osm>
```

以上示例区域由四个Node节点构成,ref中指明了四个Node节点的ID。

1.3 OSM道路数据的提取与处理

在OSM中,道路数据主要存储于Way要素中并通过(tag)中的“highway”键(Key)来标识,而道路的等级分类则通过“highway”键的值(Value)来反映,包括“primary”、“motorway”、“trunk”、“track”等。道路数据中还包括道路的物理结构(如路面材质、车道数、宽度)和法律限制(如最大限速、通行高度)等相关的属性信息,这些内容是通过标签来添加的。其中“nd”用于描述组成该Way的有序节点串(Nodes),“ref”指的是Nodes对应的Node ID,可以通过Node ID找到对应的节点元素,并获得其坐标系信息。

首先利用正则表达式[10]快速匹配OSM文件中的节点信息,以Node ID为索引建立节点的哈希表;然后提取OSM文件中的道路信息,构建道路信息列表,道路中若包含多个路段,则将每个路段拆分成独立的道路对象,道路的起始和终止端点从节点哈希表中获取;最后形成道路元数据对象列表[11]信息。关键代码如下:

```
class PointGeo //节点对象信息
{    public double lat;
     public double lng;
}
class LineInfor //道路元对象信息
{    public PointGeo     Begin;
     public PointGeo     End;
}
//提取节点信息
private Hashtable DecodeNodes( string xmlMapData)
{
    MatchCollection Matches, MatchItem;
    Hashtable hsNodes = new Hashtable( );
    Matches = Regex. Matches( xmlMapData, " < node [\\s|\\S|.] * ? > [\\s|\\S|.] * ? < /node > ", RegexOptions. IgnoreCase | RegexOptions. ExplicitCapture);
    foreach ( Match item in Matches) {
        PointGeo node = new PointGeo( );
        MatchItem = Regex. Matches( item. ToString( ), " id = \"( ? < NodeId > \\d* ) \" * lat = \"( ? < lat > [\\d|\\.] * ) \" * lon = \"( ? < lon > [- |\\d|.] * ) ", RegexOptions. IgnoreCase | RegexOptions. ExplicitCapture);
        if ( MatchItem = = null || MatchItem. Count < = 0)
            continue;
        string NodeId = MatchItem [0] . Groups ["NodeId"] . ToString( );
        node. lat = double. Parse( MatchItem [0] . Groups ["lat"] . ToString( ) );
        node. lng = double. Parse( MatchItem [0] . Groups ["lon"] . ToString( ) );
        hsNodes. Add( NodeId, node);
    }
    return hsNodes;
}
//提取道路信息
private List < LineInfor > DecodeLines( string xmlMapData, Hashtable hsNodes)
{
    MatchCollection Matches, NodesList;
    List < LineInfor > lstLines = new List < LineInfor > ( );
    Matches = Regex. Matches( xmlMapData, " < way [\\s|\\S|.] * ? > [\\s|\\S|.] * ? < /way > ", RegexOptions. IgnoreCase | RegexOptions. ExplicitCapture);
    foreach ( Match item in Matches) {
        NodesList = Regex. Matches( item. ToString( ), " < nd ? ref = \"( ? < NodeId > \\d* ) \" ? / > ", RegexOptions. IgnoreCase | RegexOptions. ExplicitCapture);
        foreach ( Match node in NodesList) {
            bool Begin = true;
            LineInfor objLine = new LineInfor( );
            objLine. Begin = new PointGeo( );
            objLine. End = new PointGeo( );
            string NodeId = node. Groups ["NodeId"] . ToString( );
            if ( Begin = = true) {
                objLine. Begin. lng = ( ( PointGeo) hsNodes [NodeId] ) . lng / 100000;
                objLine. Begin. lat = ( ( PointGeo) hsNodes [NodeId] ) . lat / 100000;
                Begin = false;
            } else {
                objLine. End. lng = ( ( PointGeo) hsNodes [NodeId] ) . lng / 100000;
                objLine. End. lat = ( ( PointGeo) hsNodes [NodeId] ) . lat / 100000;
                lstLines. Add( objLine);
                objLine. Begin. lng = ( ( PointGeo) hsNodes [NodeId] ) . lng / 100000;
                objLine. Begin. lat = ( ( PointGeo) hsNodes [NodeId] ) . lat / 100000;
            }
        }
    }
    return lstLines;
}
```

2　Dijkstra 算法思想

Dijkstra 算法是典型的最短路径算法，用于计算一个节点到其他所有节点的最短路径，其主要特点是以起始点为中心向外层扩展，直到扩展到终点为止。Dijkstra 算法的基本思想是以源点为圆心，按最短路径长度递增的顺序通过对路径长度迭代得到从源点到其他各目标节点的最短路径[12]。

根据 Dijkstra 算法的基本思想，引入如下状态变量[13]：

(1) 辅助向量 D，可用数组实现，其每一个分量 $D[i]$ 用来存放源点到其他节点的最短路径长度；

(2) 集合 V 和 S，其中 V 集合用来存放未确定最短路径的节点，S 集合用来存放已确定最短路径的节点。S 的初始状态为空集，V 则包含带权图中的所有节点；

(3) 辅助变量 path_matrix，可用二维数组实现，用来记录源点到其他各个节点的最短路径所经过的顶点。

算法描述如下:

(1) 初始化集合 V、S，同时利用带权图的邻接矩阵 arcs 初始化数组 D，即若源点到相应节点有弧，对应的分量取值为弧上的权值，否则对应的分量取值为 ∞；

(2) 选择 D 中最小的数组分量，假设为 $D[i]$，则 i

就是已求得源点到其最短路径的终点，故将 $S = S \cup \{i\}$，即将已确定最短路径的节点 i 加入 S 集合；

(3) 根据节点 i 修改更新数组 D 中源点到集合 $V - S$ 中的节点 k 所对应的分量，若 $D[i] + arcs[i][k] < D[k]$，则 $D[k] = D[i] + arcs[i][k]$；

(4) 重复(2)、(3)操作，直至所有的节点确定最短路径，即集合 V 为空。

3 Dijkstra 算法实现

(1) 构造权值矩阵数组。

把从 OSM 中提取出来的道路信息构造成端点之间的距离权值矩阵数组[14]。每条道路元数据包含起始和终止两个节点，权值为两点间的地理距离[15]。初始设置所有的节点间距离为无限大(近似为双精度类型的最大值)。将所有端点放到未确定最短路径的节点。

```
private void PrepareNodeMatrix()
{
    double RightValue, radLat1, radLat2, b;
    _NodesMatrix = new double[_lstLines.Count, _lstLines.Count];
    for (int i = 0; i < _NodesMatrix.GetLength(0); i++)
        for (int j = 0; j < _NodesMatrix.GetLength(1); j++)
            _NodesMatrix[i, j] = double.MaxValue;
    for (int i = 0; i < _lstLines.Count; i++)
    {
        radLat1 = _lstLines[i].Begin.lat * Math.PI / 180.0;
        radLat2 = _lstLines[i].End.lat * Math.PI / 180.0;
        b = _lstLines[i].Begin.lng * Math.PI / 180.0 - _lstLines[i].End.lng * Math.PI / 180.0;
        RightValue = 2 * Math.Asin(Math.Sqrt(Math.Pow(Math.Sin((radLat1 - radLat2) / 2), 2) + Math.Cos(radLat1) * Math.Cos(radLat2) * Math.Pow(Math.Sin(b / 2), 2)));
        RightValue = Math.Round(RightValue * 63781370 * 10000) / 10000;
        _NodesMatrix[i, i+1] = RightValue;
        _NodesMatrix[i+1, i] = RightValue;
    _UnVisitedPath.Add(i)
    }
}
```

(2) 在未确定节点中查找与指定节点之间的最短距离。

```
int GetMinWayIndex(int _Ps)
{
    double min = double.MaxValue;
    int MinIdx = -1, nodeIndex;
    for (int i = 0; i < _UnVisitedPath.Count; i++) {
        nodeIndex = (int)_UnVisitedPath[i];
        if (min > _NodesMatrix[_Ps, nodeIndex]) {
            min = _NodesMatrix[_Ps, nodeIndex];
            MinIdx = i;
    }
    }
    return MinIdx;
}
```

(3) 遍历未确定节点，直至所有的节点确定最短路径。

```
public string FindWay()
{
    int idx, nodeIdx, tmp;
    while (_UnVisitedPath.Count > 0) {
        idx = GetMinWayIndex();
    if (idx < 0)  break;
        nodeIdx = (int)_UnVisitedPath[idx];
    if (_Pd == nodeIdx) break;
    for (int i = 0; i < _UnVisitedPath.Count; i++) {
    if (i == idx) continue;
            tmp = (int)_UnVisitedPath[i];
    if (_NodesMatrix[_Ps, tmp] - _NodesMatrix[nodeIdx, tmp] > _NodesMatrix[_Ps, nodeIdx]) {
                _NodesMatrix[_Ps, tmp] = _NodesMatrix[_Ps, nodeIdx] + _NodesMatrix[nodeIdx, tmp];
                _NodesMatrix[tmp, _Ps] = _NodesMatrix[_Ps, tmp];
    path[tmp] = path[nodeIdx] + "," + nodeIdx.ToString() + ",";
            }
        }
        _UnVisitedPath.RemoveAt(idx);
    }
    return path[Pd];
}
```

4 结束语

OpenStreetMap 是一个网上地图协作计划[16]，目标是创造一个内容自由且能让所有人编辑的世界地图。作为免费开放的地图服务，个人、网站和企业都可以使用这些地图信息。

目前包括苹果和微软在内的多家大公司都在使用 OpenStreetMap 的服务。

文中基于 OpenStreetMap 数据，介绍了如何从地图数据中提取出交通道路信息，建立合适的存储模型，形成道路图[17]，运用 Dijkstra 最短路径算法求解最短路径。文中旨在阐述基于 OSM 数据实现道路信息的提取和最短路径算法的实现，并未优化 Dijkstra 算法的运算效率[18]，带权路径仅仅以距离作为唯一的权值。

在实际城市道路选择中,还应考虑交通拥堵、费用、单双向行驶等方面的问题。

参考文献:

[1] 马　超. 遗传算法和 Dijkstra 算法在动态权值系统中的比较[J]. 计算机技术与发展,2012,22(9):21 - 24.

[2] 苏　莹,王英杰,余卓渊. 一种建立公交网络的最短路径改进算法[J]. 地球信息科学,2005,7(2):99 - 104.

[3] 杜　莹,刘建忠. 基于 WebGIS 最优路径分析的设计与实现[J]. 测绘学院学报,2002,19(1):56 - 58.

[4] 刘文宝,邓　敏,易　彤. 矢量 GIS 中模糊地理边界的分析[J]. 山东科技大学学报: 自然科学版,2000,19(1):28 - 32.

[5] 白　尘. 交通路网中最优路径算法的道路权重选择[J]. 中国管理信息化,2009,12(15):54 - 56.

[6] 赵春燕,王国华,周　军. 支持城市多种交通方式的最佳路径分析[J]. 测绘信息与工程,2009,8(4):8 - 10.

[7] 何　卫. Web 地理信息系统的设计与实现[D]. 西安: 西安电子科技大学,2004.

[8] Sun Yushan. Embedded vehicle navigation system based on VxWorks[C]//Proceedings of the 5th international symposium on test and measurement (volume 1). [s. l.]: [s. n.], 2003.

[9] Zhang Yi,Zhang Meng,Liang Yanchun. Application of an improved dijkstra algorithm in multicast routing problem[J]. Computer science,2009,36(8):205 - 207.

[10] Wang Yawen,Wang Xili,Cao Ham,et al. Shortest route - planning algorithm within dynamic restricted searching area[J]. Application research of computers,2007,24(7):89 - 91.

[11] 常志明,毛新军,齐治昌. 基于 Agent 的网构软件构件模型及其实现[J]. 软件学报,2008,19(5):1113 - 1124.

[12] Chen Wenjie. The research and application of reasonable route set of route choice[D]. Guangzhou: Guangzhou Sun Yat - sen University,2008.

[13] Azevedo J A,Madeira J J E R S,Martins E Q V,et al. A shortest paths ranking algorithm[C]//Proc of the annual conf AIRO. [s. l.]: [s. n.],1990:1001 - 1011.

[14] 乐　阳,龚健雅. Dijkstra 最短路径算法的一种高效率实现[J]. 武汉测绘科技大学学报,1999,24(3):209 - 212.

[15] 曾　松,杨佩昆,方棣波. 城市道路网结构的可达性评价[J]. 同济大学学报,2001,29(6):666 - 671.

[16] 王行风,贾　凌. GIS 支持下的城市交通网络最短路径研究[J]. 计算机与现代化,2005,13(3):9 - 12.

[17] Martins E Q V,Santos J L E. A new shortest paths ranking algorithm[J]. Investigacao operacional,1999,20(1):47 - 62.

[18] Sim K M,Sun W H. Ant colony optimization for routing and load - balancing: survey and new directions[J]. IEEE transactions on system,2003,33(5):560 - 572.

(上接第 36 页)

一步在嵌入式平台移植更为复杂的视频分析算法奠定了良好的基础。当然,因为 DM6446 是在 2005 年推出的,性能有限,因此在实时性方面,还不能达到十分理想的状态,下一步将进一步探索如何优化算法以及更合理的分配系统地资源。

参考文献:

[1] 叶　林,陈岳林. 基于达芬奇平台的智能视频汽车安全驾驶系统[J]. 计算机系统应用,2009(9):34 - 37.

[2] 周金模. 基于达芬奇技术的嵌入式实时视频系统研究[D]. 武汉: 华中师范大学,2008.

[3] 宋　磊,方向忠. 达芬奇技术的视频应用分析[J]. 电视技术,2006(9):31 - 33.

[4] 鲁　达. 基于 DM6446 平台的智能视频监控关键算法研究与实现[D]. 上海: 上海交通大学,2009.

[5] 赵　勇,袁誉乐. DAVINCI 技术: 原理与应用指南[M]. 南京: 东南大学出版社,2007.

[6] Barnich O,Broeck M V D. ViBe: a universal background subtraction algorithm for video sequences[J]. IEEE Transaction on Image Processing,2011,20(6):1709 - 1723.

[7] 徐红仙,张　桦,孙志海. 基于 DM6446 的车牌定位快速算法实现与优化[J]. 杭州电子科技大学学报,2011,31(1):54 - 57.

[8] 郑　红,刘振强,王　鹏. 嵌入式实时系统的 DSP 软件开发技术[M]. 北京: 北京航空航天大学出版社,2010.

[9] 延瑾瑜. 基于 TMS320DM6437 的 H.264 视频编码器的研究与优化[D]. 北京: 北京化工大学,2009.

[10] 陈延利,施永豪. 运动目标检测与跟踪的 DSP 实现[J]. 计算机技术与发展,2012,22(8):82 - 84.

[11] Wu Y T. Brief analysis of the DM6467 HD - VICP subsystem functional simulator[C]//Proc of 2010 2nd International Conference on Networking and Digital Society (ICNDS). [s. l.]: [s. n.],2010:609 - 612.

[12] Wu L,Li C. Performance modeling of a reconfigurable shared buffer for high - speed switchirouter[C]//Proc of IEEE International Conference on Communications. [s. l.]: [s. n.], 2008:5674 - 5679.