基于 XML-Tree 的单源最短路径改进算法及快速仿真①

戴莉萍, 黄龙军

(江西师范大学 软件学院, 南昌 330022)

摘 要: 单源最短路径问题是图论中的一个基础课题. 结合图与树在数据结构表示上的相似性及易转换性, 基于 XML 技术提出了一种改进的单源最短路径算法. 该算法利用 XML 结构, 按照任意树的生成顺序组织成一棵树; 而后对树中的每条边做判断, 不断调整当前各个节点到源点之间的最短距离. 使用基本控件快速实现该算法的仿真过程, 实验结果表明, 该算法具有较好的时间效率, 灵活性较强、简单易懂及较好的应用价值.

关键词: 单源最短路径; XML 树型结构; treeview 控件

Improvement and Quick Simulation of Single-Source Shortest Path Algorithm Based on XML-Tree

DAI Li-Ping, HUANG Long-Jun

(Software School, Jiangxi Normal University, Nanchang 330022, China)

Abstract: Single-source shortest path problem is one of basic researches in graphic theory. Considering the similarity and easy transferability of data structure between graphic and tree, an improved algorithm solving this problem is put forward based on the technology of XML. Using XML structure, this algorithm transfers a graphic to some tree, and analyzes each edge in this tree to adjust the best distance between other vertices and the source vertex. With basic controls used in the execution of this algorithm to simulated the process, and experimental results show that this new algorithm is much more efficient, and more flexible, and easy to understand and good to practical applications.

Key words: single-source shortest path; XML-tree structure; treeview

在图论中, 最短路径问题就是寻找图中连接两个顶点的一条路径, 使其权值和最小. 单源最短路径问题则是给定一个源点 v, 求取从 v 点到图中其它点的最短路径及其权值和. 解决单源最短路径问题的经典算法是基于贪心思想的 Dijkstra 算法[1,2], 它设置顶点集合 S, 每次从 V-S 中取出具有最短特殊路径长度的顶点 u, 将 u 添加到 S 中, 不断地做贪心选择来扩充这个集合, 其算法效率较低. 还有一些算法是基于传统的 Dijkstra 算法之上进行改进的 1. 其中有些算法是使用二维数组, 需要较多的比较次数; 有些算法使用了链表, 需要较多的遍历时间; 有些使用了最小堆, 需要较多的构造时间等等; 不同的算法有着不同的优势所在[3-6]. 本文中提出的基于 XML-tree 单源最短路径算法是以图中的边元素为基础, 利用 XML 格式将图转换为树形结构, 之后利用节点生成的顺序逐一判断路径长短, 进行相应的取舍后生成结果值.

1 算法的输入输出结构

在基于 XML-Tree 的单源最短路径改进算法中, 输入数据结构主要为存放图信息的 XML 文件、可由图转化而来的树结构及对应不同树结构的自定义结构体.

图是由点、边及权值所体现. 例如<a, b, 10>表示为连接点 a 和点 b 的一条权值为 10 的边. XML 元素的基本结构由开始标记、数据内容和结束标记组成. 因此 XML 中一个结点可以作为一条边的描述, 例如: <edge onepoint="a" anotherpoint="b">10</edge>.

此 edge 元素的描述中, 有两个属性 onepoint 和

① 基金项目:江西省高校教改课题(JXJG-14-2-20)

收稿时间:2015-03-13;收到修改稿时间:2015-05-09

© 中国科学院软件研究所 http://www.c-s-a.org.cn

anotherpoint, 对应的就是边的两个顶点. 开始标记和结束标记之间的文本内容则是这条边的权重. 对于有向图, onepoint 可以是起点, anoterpoint 是终点. 对于无向图, onepoint 可以是 a 或者 b, 有向图的处理比无向图要简单很多, 因此以下讨论无向图, 以图 1 为例.

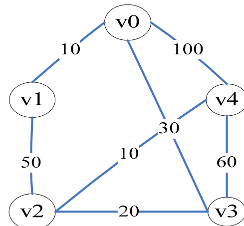


图 1　算法示例图

图的边、点和权值信息放置在一个 xml 文件中, 内容如下所示:

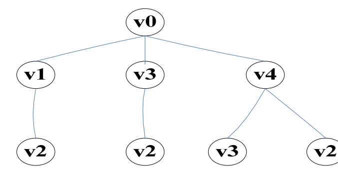
```
<?xml version="1.0"?>
<edges>
 <edge onepoint="v0" anotherpoint="v1">10</edge>
 <edge onepoint="v0" anotherpoint="v3">30</edge>
 <edge onepoint="v0" anotherpoint="v4">100</edge>
 <edge onepoint="v1" anotherpoint="v2">50</edge>
 <edge onepoint="v3" anotherpoint="v2">20</edge>
 <edge onepoint="v3" anotherpoint="v4">60</edge>
 <edge onepoint="v4" anotherpoint="v2">10</edge>
</edges>
```

而后将图转换为树状结构, 可以按照层次生成或者左子女优先顺序生成, 如图 2 所示.

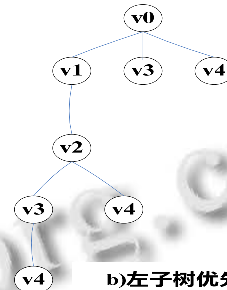
在生成树的同时, 将这些边数据顺序地赋值给自定义结构体数组, 结构体定义如下:

```
Structure edge
    Dim father As String
    Dim child As String
    Dim weight As Integer
End Structure
```

其中 father 和 child 对应于 XML 中 edge 元素的 onepoint 或 anotherpoint 属性, weight 赋值为元素的文本内容. 例如根据图 2(b)可以看出边的产生顺序为 <v0,v1>, <v0,v3>, <v0,v4>, <v1,v2>, <v2,v3>, <v2,v4>, <v3,v4>. 顶点和权重也按照这样的顺序依次存入结构体数组 edge 中.



a)层次型树状



b)左子树优先型树状

图 2　转换为树的图

算法的结果存入一个 xml 文件中, 结果如下显示:

```
<?xml version="1.0"?>
<nodes>
   <node id="v0"> 0</node>
   <node id="v1" fatherid="v0"> 10</node>
   <node id="v3" fatherid="v0"> 30</node>
   <node id="v4" fatherid="v2"> 60</node>
<node id="v2" fatherid="v3"> 50</node>
</nodes>
```

从这个 xml 文件中可以看出源点为 v0; id 为 v1 的节点到 v0 的最短距离为 10, 其直接父节点 fatherid 为 v0. 结果只保留了某个节点的前趋节点和最短距离; 如果需要知道具体路径, 则可以通过递归或循环来获取.

2　算法的处理过程

算法最主要的部分则是求取源点至其它各个节点之间的最短距离. 由于图结构已经转换为 edge 数组的线性结构, 因此算法的操作对象为 edge 数组. 算法描述如下.

算法功能描述: 计算各个点至源点的最短路径

输入: 结构体 edge 数组

输出: resultdata.xml 文件

处理过程:

在 xml 中添加一个 node, id 为源点 v0

for i=0 to 边的数量-1

{

www.c-s-a.org.cn
www.c-s-a.org.cn
© 中国科学院软件研究所 http://www.c-s-a.org.cn

```
  查找 id=edge[i].father 的 node(fathernode);
  if 顶点 edge[i].child 不存在于 xml 中
  {
    创建一个新的 node;
    新 node 的 id=edge[i].child;
    新 node 的 fatherid=edge[i].father;
    新 node 的距离=fahternode 的当前距离+edge[i].weight;
  }
  else  //该节点 childnode 已经存在在 xml 中
  {
    weight1=childnode 的当前距离
    weight2=fathernode 的当前距离+edge[i].weight
    if weight1>weight2
    {
    1)向下修改 childnode 的所有子节点距离值;
    2)向上判断修改 childnode 的祖先节点是否会受此边的影响而变成 childnode 的子节点;
    3)修改 childnode 的 fatherid 和距离值;
    }
    else  //weight1<=weight2
    {
     if fathernode 的当前距离>childnode 的当前距离+edge[i].weight
     {
     4)向下修改 fathernode 的所有子节点距离值;
     5)向上判断修改 fathernode 的祖先节点是否会受此边的影响而变成 fathernode 的子节点;
     6)修改 fathernode 的 fatherid 和距离值;
     }
    }
  }
}
```

从算法描述中可以看出, 一条边的加入可能会引起两个端点各自的子孙节点、祖先节点的距离值发生变化. 例如图 2(b)中当处理到边<v2,v3>时, v3 已经在 xml 结果中存在, 即图中出现回路. 假如由于<v2,v3>边的出现使得 v2 距离 v0 的长度更短, 那么 v2 的父节点应该由 v1 变为 v3, 修改 v2 其下所有子节点的最新距离, 同时判断 v2 的父节点 v1 是否会成为 v2 的子节点, 即 v2 的所有父辈节点是否会改变相应的父子关系. v3 也需要做同样的判断处理.

对应算法中 1)4)步骤的过程 modifyson 描述如下, 其中参数 father 表示节点名称, weight1 是该节点原有的较长距离, weight2 表示该节点新的较短距离.

```
Procedure modifyson (father,weight1,weight2)
 {
   在 xml 中找到 father 的直接子节点 varnodelist
   For j = 0 To 子节点个数 – 1
   {
     varnode=varnodelist.item(j)//取子节点
     if varnode=源点 then 退出
     oldweight=varnode 的当前距离
     newweight=weight2+(oldweight-weight1)
     modifyson(varnode,oldweight,newweight)
     在 xml 中修改 varnode 的距离值
   }
 }
```

以下的 changeancestor 过程对应了算法描述中的 2)5)处理过程. 其中参数 sonname 是当前子节点, newweight 是子节点新的最短距离值.

```
Procedure changeancestor( sonname, newweight)
 {
   If sonname=源点 then 退出
   在 xml 中定位到 id=sonname 的点信息;
   在 xml 中定位到 sonname 的父节点 fathernode;
   此边权值 edgeweight=sonname 的当前距离-fathernode 的当前距离;
   weight1=fathernode 的当前距离
   weight2=newweight+edgeweight/ /新距离
   if weight1>weight2
   {
     modifyson(fathernode,weight1,weight2);
     changeancestor(fathernode,weight2);
     修改 fathernode 的 fatherid 父节点为 sonname;
     修改 fathernode 的最短距离为 weight2;
    }
  }
```

算法描述中并没有规定图需要转化为怎样的树型结构, 它是按照树的生成顺序来处理的, 例如同一条边, 按照深度优先生成树, 它在数组中的下标是 3, 但按照广度优先生成树, 它在数组中的下标是 5, 位置不

© 中国科学院软件研究所 http://www.c-s-a.org.cn

同使得该点在执行过程中影响到的祖先节点和子孙节点个数也会不同. 因此该算法的时间效率需要考虑有三个因素: 图的顶点、边和树的生成顺序, 这也使得增加算法灵活性的同时不能保证该算法时间效率的稳定性. 设图有 n 个顶点和 e 条边, 通过整个算法实现过程可以看出算法的最好时间效率是 O(e), 也就是说只执行了循环中的 if 语句, 即增加新的节点; 或者后面需要处理的边全部被舍弃了, 即不需要遍历父节点和子节点进行判断修改. 在最坏情况下, 每 1 次循环中上下遍历的节点个数为 n-1, 此时的时间复杂度为 O(en), 根据连通图的性质, 时间复杂度为 $O(n^2)$—$O(n^3)$. 算法中可以通过调整树的生成顺序来很好地降低该时间复杂度.

该算法的程序代码量小, 数据结构简单, 操作易于理解. 分析代码可以发现该算法如果去掉步骤 2)5), 则可以完成有向图的单源最短路径问题求解. 同样基于 XML-tree 算法也可以很方便地处理正负权值, 只需要加入相应的判断即可, 例如:

```
if weight1<0 and varedge[i].weight<0.
```

3 算法的快速仿真

在算法的快速仿真过程中, 除去算法本身执行需要的代码外, 可视化的部分应尽量简单, 包括代码量较短、控件使用较少等等. 本文中算法的仿真就是采用了 VS2010 的 VB.Net 开发环境来模拟算法的执行过程. 图 3 是初始数据的输入界面执行效果.

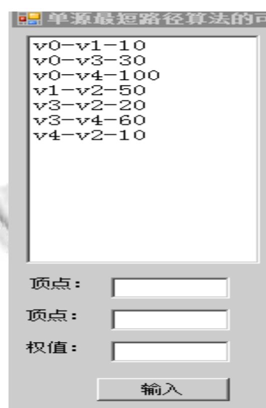


图 3 算法输入界面

输入处理的关键代码如下:

```
……
varelement.SetAttribute("onepoint", onepoint)
varelement.SetAttribute("anotherpoint", anotherpoint)
varnode.InnerText = weight
rootnode.AppendChild(varnode)
```

对于每条输入的边, 首先都创建一个新的 element 元素, 对该元素的 onepoint 和 anotherpoint 属性分别赋值为边的两个顶点. 元素的 innertext 赋值为权值, 然后将此新元素使用 appenchild 方法加入.

如果不进行仿真, 则选定源点后, 该 XML 文件中的每条边可以按照特定的顺序依次进行处理, 得到最终结果. 对于快速仿真, 则合适的控件可以选择 treeview, 而且图的无向性使得树的生成是多样的, 图 4 生成了其中的两种结构: 宽度优先和左子树优先.

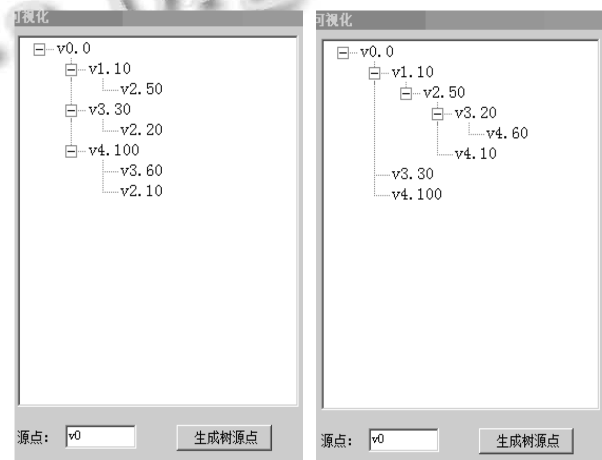


图 4 图的树型结构

图中节点旁的数字表示该节点与其父节点的权重值, 各个节点的新增在 treeveiew 的节点双击 nodedoubleclick 事件中, 在产生新节点的同时将边信息写入到自定义的结构体 edge 数组中.

edge 数组保存了新算法所需要的数据及处理顺序. 增加新节点的关键代码如下:

```
strarray =
Split(tv_original.SelectedNode.Text.ToString, ".")
querystr = "./edge[@onepoint='" & strarray(0) & "' or
@anotherpoint='" & strarray(0) & "']"
varnodelist = rootnode.SelectNodes(querystr)
For j = 0 To varnodelist.Count - 1
    ……
    varedge(edgenum).father = strarray(0)
    varedge(edgenum).weight = varnode.InnerText
If varelement.GetAttribute("onepoint") = strarray(0)
Then
    varedge(edgenum).child =
```


© 中国科学院软件研究所 http://www.c-s-a.org.cn

```
                varelement.GetAttribute("anotherpoint")
        nodestr =varelement.GetAttribute("anotherpoint") &
                "." & varnode.InnerText
    Else
        …… // 与上段类似, 将 anotherpoint 改为 onepoint 即可
    End If
edgenum = edgenum + 1
tv_original.SelectedNode.Nodes.Add(nodestr)
……
Next j
```

首先获取了双击的节点名称, 在 XML 文件中查找 onepoint 或 anotherpoint 等于节点名称的边; 而后将这些边加入到 treeview 和 edge 数组中. 算法的动态执行使用了 Timer 控件的 Tick 事件, 运行效果如图 5 所示.

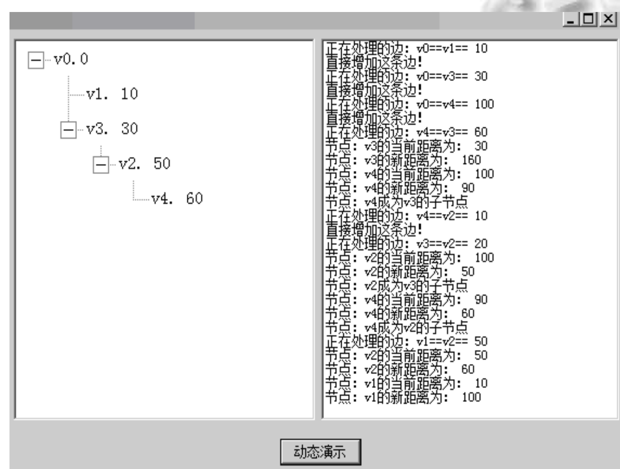


图 5　算法动态执行效果图

左半部分仍旧是一个 treeview 控件, 随着程序每处理完一边, 该 tree 就会相应的发生变化. Treeview 中每个节点旁边是距离源点 v0 的当前最短记录. 右边则是每条边处理的文字说明. 以下算法的主体代码.

```
……
If i <= edgenum - 1 Then
  querystr = "./node[@id='" & varedge(i).child & "']"
  childnode = rootnode.SelectSingleNode(querystr)
  querystr = "./node[@id='" & varedge(i).father & "']"
  fathernode = rootnode.SelectSingleNode(querystr)
  If childnode Is Nothing Then
    …… // 增加一个新节点
  Else
    ……
    weight1 = Int(childnode.InnerText)
    weight2 = Int(fathernode.InnerText) +
                varedge(i).weight
    If weight1 > weight2 Then
      Call modifyson(varedge(i).child, weight1, weight2)
      Call changeancestor(varedge(i).child, weight2)
      childelement.SetAttribute("fatherid",
varedge(i).father)
      childelement.InnerText = Str(weight2)
    Else
      ……// 与上段的 weight1 和 weight2 等代码类似
 Call rebuildtree(tbx_source.Text) //重构树
 ……
```

算法中的 rebuildtree 过程实现了每一条边处理之后的 tree 结构变化, 也就是将当前 resultdata 的 xml 文件中的数据以递归的方式组织起来.

4　结语

基于 XML-tree 单源最短路径算法在开始和结束部分使用了树遍历的简单递归方法, 中间的处理过程以循环、判断和递归为主, 时间复杂度较小, 数据结构为 XML 和自定义结构体. 该算法效率高、易于理解、适用性较好, 例如可以应用在有向图、无向图或甚至稍加修改来适应混合图; 且具有较强的灵活性, 例如修改判断语句即可很好地实现每对顶点间最短距离. 本算法从另一个角度揭示了图、树及线性结构之间的关联性和转换性, 对于最短路径问题的进一步研究与应用提供了一定的参考思路和方法.

参考文献

1 王晓东.算法设计与分析.第 2 版.北京:清华大学出版社,2010.
2 严蔚敏,陈文博.数据结构及应用算法教程.北京:清华大学出版社,2011.
3 周玉林.单源最短路径问题的改进算法.上饶师范学院学报,2001,21(3):18–22.
4 王防修,周康.基于回溯法的 Dijkstra 算法改进及仿真.计算机仿真,2013,30(11):352–355.
5 蒲在毅,任建军.用标号法实现单源最短路径问题的迪杰斯特(dijkstra)算法.四川师范学院学报(自然科学版),2003,24(1):122–126,131.
6 王强.最短路径问题的若干算法的编程.计算机科学, 2004,31(B07): 94–95,100.
7 胡英.Visual Basic 程序设计.北京:清华大学出版社,2014.
8 孙更新,李伟超等.XML 编程与应用教程.第 2 版.北京:清华大学出版社,2014.

© 中国科学院软件研究所 http://www.c-s-a.org.cn