基于配对堆改进的 Dijkstra 算法

张林广 方金云 申排伟
(中国科学院计算技术研究所空间信息处理技术实验室, 北京 100080)

摘 要 在 GIS 网络分析系统中,Dijkstra 算法是求解最短路径的经典算法。为了进一步提高求解最短路径的效率和节省系统的内存空间,提出了使用一种新式的数据结构——配对堆,以便通过实现可降级的优先队列来改进 Dijkstra 算法,然后通过研究配对堆的基本操作,给出了使用配对堆结构实现 Dijkstra 算法的方法和流程,并分析了其算法复杂度。该算法在 VegaGIS 系统中实现,取得到了较好的效果。

关键词 Dijkstra 最短路径 优先队列 配对堆 织女星地理信息系统
中图法分类号: P208 **文献标识码**: A **文章编号**: 1006-8961(2007)05-0922-05

An Improved Dijkstra Algorithm Based on Pairing Heap

ZHANG Lin-guang, FANG Jin-yun, SHEN Pai-wei
(*Laboratory of Spatial Information Technology, Institute of Computing Technology, CAS, Beijing* 100080)

Abstract Dijkstra is a classic algorithm to compute the shortest-path in GIS network analysis system. To improve the algorithm efficiency and save memory usage, this paper proposes a new data structure, called "pairing heap", to implement the priority queue. So that the Dijkstra algorithm can use pairing heap. This paper gives out the implementation details and algorithm's flow via studying the basic function of paring heap, and then analyses the algorithm's complexity. This new algorithm has applied in VegaGIS and obtains good results.

Keywords Dijkstra, shortest-path, priority queue, paring heap, VegaGIS

1 引 言

随着城市建设的高速发展,城市的交通网络变得越来越复杂,规模也越来越大,给人们出行带来极大不便。为此,如何快速选择一条既省时间又经济合理的路线是地理信息系统(geography information system, GIS)在交通领域研究的主要问题。GIS 网络分析功能是解决这些问题的主要方法,其核心思想就是求解最短路径和最优路径,而利用 GIS 网络分析功能则可以解决公交换乘、行使导引、最近设施查找、车辆送货(TSP 问题)等交通中存在的许多问题。

GIS 网络分析中最主要的算法就是 Dijkstra 最短路径算法。由于 GIS 空间要素具备空间属性,一般可使用空间信息来优化 Dijkstra 算法,例如:①用椭圆限制搜索区域[1]来减少遍历的节点;②利用道路与起始地到目标地的直线之间的夹角关系[2]使搜索趋向于目标地;③直接采用当前节点到目标地节点的直线距离的直线优化 Dijkstra 算法[3];④A* 算法[4~6]是使用评估函数来有目的地选择接近目标地的节点,以提高算法效率。虽然基于空间信息的 Dijkstra 优化算法已经取得了较好的效果,但在求解非距离权值的最优路径时,作用却不大。例如,如果按时间最短计算,从北京的一端到另一端,一个好的选择可能是直接走环路,而不是穿过城市中心地区的距离最短路径。因此,路径优化时,一方面可通过特定的信息对 Dijkstra 算法进行优化,另一方面也需要不断

基金项目:国家"863"高技术研究发展计划项目(2001AA1135210,2002AA114020)
收稿日期:2006-02-10;改回日期:2006-05-08
第一作者简介:张林广(1981~),男。2003 年毕业于吉林大学,现为中国科学院计算技术研究所硕士研究生。主要研究方向为海量空间信息处理关键技术。E-mail:zhanglg@ict.ac.cn

改进实现 Dijkstra 算法的数据结构和方法。

2 Dijkstra 算法思想

Dijkstra 算法是求最短路径的最基本和使用最广泛的算法。使用 Dijkstra 算法可以求出网络图中一个节点到另一个可连通节点的最短路径。主要计算过程步骤如下:

(1) 以一个给定的起始节点开始,依次遍历该节点的所有邻接节点,并计算起始节点到邻接节点的距离,也就是连接节点之间的弧段长度;

(2) 从所有未进行邻接节点遍历,但仅从计算过距离的节点中取出距离最小的节点进行邻接节点遍历,并累加距离;

(3) 如果遍历到的节点已经计算过距离,就比较当前新计算的距离与已计算过的距离大小,然后取其中最小值,并更新节点的信息;

(4) 反复进行步骤(2)、(3),直到取出的节点是目标节点,其得到的距离就是起始节点到该节点的最短路径距离。

在 Dijkstra 算法中,为了获得已经计算过的距离中距离值最小的节点,一般采用一个可更新、可排序的优先队列来实现。因此,如何实现 Dijkstra 算法的优先队列结构,就成为提高 Dijkstra 算法性能的关键。标准的优先队列仅支持"插入(insert)"、"删除最小(delete-min)"两种元素操作,但使用 Dijkstra 算法还需要优先队列具备"降级(decrease-priority)"操作,以便能动态调整已排序队列中的元素,而选择合适的数据结构实现可降级的优先队列,则可以有效地降低算法的复杂度和减少算法的执行时间。

为了降低优先队列的操作时间复杂度,一般可采用堆结构来实现优先队列。设 V 为节点总数,E 为弧段总数。采用二叉堆(binary heap)的优先队列的所有操作时间复杂度都是 $O(\log N)$,Dijkstra 算法的时间复杂度为 $O(E\log V)$[7]。d-堆(d-ary heap)类似于二叉堆,只是每个节点有 d 个儿子,它将插入和降级的时间复杂度改进为 $O(\log_d N)$,但删除最小的时间复杂度只提高到 $O(d\log_d N)$[8],d 的一个好的选择是 $d=\max(2,[E/V])$,因此它把 Dijkstra 算法的时间复杂度改进为 $O(E\log_{(2+[E/V])}V)$。左式堆(leftist heap)类似于二叉堆,其和二叉堆具有相同的操作时间复杂度,但是它却能提供时间复杂度为 $O(\log N)$ 的"合并(merge)"操作。二项堆(binomial heaps)的插入操作能够达到的摊还时间复杂度为 $O(1)$,其他操作的摊还时间复杂度为 $O(\log N)$[9]。如果采用懒惰合并(lazy merging)方式,则插入操作摊还时间复杂度能够提高到 $O(1)$。Fredman 和 Tarjan 提出的斐波那契堆(Fibonacci heap),其插入时间复杂度为 $O(1)$,降级的摊还时间复杂度为 $O(1)$,删除最小的摊还时间复杂度为 $O(\log N)$,Dijkstra 算法的时间复杂度为 $O(E+V\log V)$[10],是实现优先队列的常用选择。

3 基于配对堆改进的 Dijkstra 算法

3.1 基本概念

本文描述的 Dijkstra 算法是采用一种新式的堆结构来实现优先队列,叫做"配对堆(pairing heap)"[11],这是一种比斐波那契堆简单,但性能却不低于斐波那契堆的数据结构(如图 1 所示)。

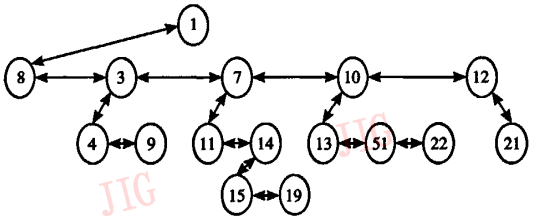

图 1　一个配对堆的例子

Fig. 1　An example of pairing heap

配对堆是由配对堆节点组成。每个堆节点除了用于排序的元素外,还包括前驱节点、左儿子节点、右兄弟节点 3 个指针。为了支持"降级"操作,作为最左儿子节点的前驱节点指针将指向其父亲节点;否则这个节点就是一个右兄弟节点,而其前驱节点指针将指向这个节点的左兄弟。配对堆节点的数据结构如图 2 所示。

以图 3 为例,A 为 B 的前驱节点,B 为 C 的前驱节点,C 为 D 的前驱节点;B 是 A 的左孩子节点,D 是 C 的左孩子节点;C 是 B 的右兄弟节点。

配对堆的性质是:(1)根节点具有整个堆最小值元素(针对最小值配对堆而言);(2)任何一个堆节点的元素值均小于等于其左儿子节点;(3)同一层次串联的兄弟虽没有大小顺序,但是均大于最左兄弟的父节点(此性质是由配对堆的"合并"操作决定的)。以上 3 个性质决定了配对堆结构能够实现可降级的优先队列。

```
template < class T >
class pnode
{
  public:
    pnode(const T& Item = T()):
      Element(Item), Prev(NULL), LeftChild(NULL),
      NextSibling(NULL){}
  T Element;                          //堆节点保存的元素
  class pnode * Prev;                 //前驱节点指针
  class pnode * LeftChild;            //左孩子节点指针
  class pnode * NextSibling;          //右兄弟节点指针
};
```

图 2 配对堆节点的数据结构

Fig. 2 The data structure of paring heap node

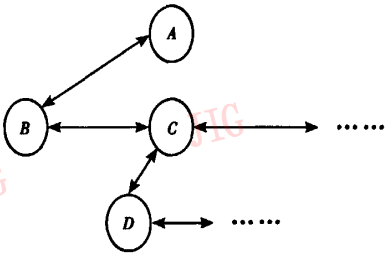

图 3 配对堆节点的指针关联

Fig. 3 The relation between pairing-heap nodes' pointer

3.2 配对堆构造过程

配对堆的基本操作包括“合并”、“插入”、“降级”和“删除最小”4 种。

其中“合并”就是把两个子堆合并成一个堆,它是后 3 种操作实现的基础。为了不失一般性,合并的第 2 个子堆可以是一个节点,也可以是具有右兄弟节点的子堆。合并的过程就是让具有较大根节点的子堆成为另一个子堆的最左的儿子(如图 4 所示)。

所谓“插入”就是为配对堆增加一个新元素。构造配对堆时,先把新元素作为一个新的子节点,然后通过与堆根节点进行合并操作来完成构造。“插入”是“合并”的特殊操作。

所谓“降级”就是改变一个堆节点的元素值,其过程是先把需要改变值的节点连同儿子节点从堆中移出组成为一个新子堆,然后把修改节点值后的子堆和旧子堆合并,其过程如图 5 所示。

“删除最小”是个复杂的操作,它是从堆中移出最小的根节点。当把堆的根节点删去后,则需要从原根节点的所有儿子节点中选取最小的节点成为新

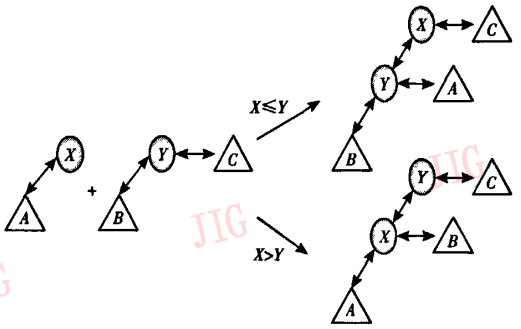

图 4 两个子配对堆的“合并”操作

Fig. 4 Merge operation of two sub pairing-heap

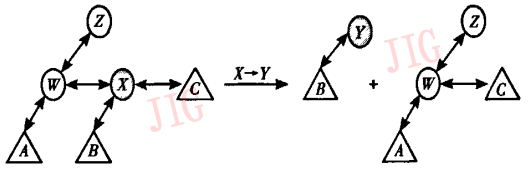

图 5 配对堆的“降级”操作

Fig. 5 Decrease-priority operation of pairing heap

堆的根节点。“删除最小”最简单的方法就是把所有儿子节点的关联打开成为若干子堆,再依次调用合并操作生成一个新的配对堆;另一种有效的方法称为“两趟合并法(two-pass merging)”[12],其过程是:首先从左到右,将打开生成的子堆每两两进行合并,然后再从右到左,依次合并剩余的子堆。例如,如果有 X_1 到 $X_6$6 个子堆,那么在第 1 次合并的过程中,把 X_1 和 X_2 合并成 Y_1,把 X_3 和 X_4 合并成 Y_2,把 X_5 和 X_6 合并成 Y_3;第 2 次合并,把 Y_3 和 Y_2 合并,再跟 Y_1 合并。“删除最小”的操作方法有很多,主要目的就是为了使配对堆每个节点的儿子数尽可能的少,以提高算法效率。

有关配对堆 4 种操作算法的实现过程可以参考文献[13]。

3.3 基于配对堆的 Dijkstra 算法

使用堆结构实现优先队列的最大问题是需要知道与已访问的网络节点对应的堆节点指针,由于更改堆的操作都是直接操作堆节点指针完成,因此,需要一种网络节点与堆节点之间的快速访问方式。如果采用关联表的存储方式,那么每次访问节点的查询时间复杂度就为 $O(\log V)$,这样使 Dijkstra 算法的时间复杂度就增加为 $O((E+V\log V)\log V)$;另一种方式是建立网络节点的哈希表,一般采用每个节点具有的唯一整型 ID 值作为哈希值。虽然这样可以以常量

时间复杂度查询到堆节点,但是由于使用哈希表需要预先申请大量的表空间,因而内存的利用率低。

本文借鉴哈希表的思想,只为每一个网络节点定义一个堆节点指针(如图 6 所示),但不预先申请内存。而是当算法遍历到对应节点时,再动态生成堆节点。这样既保证了算法的执行效率,又减少了内存的浪费。其中,堆节点中保存了对应的网络节点以及到网络节点花费等相关信息。另外,再以一个变量保存配对堆的根节点,以便于每次取出权值最小的节点。在算法求出最终结果后,再统一遍历网络节点,并释放生成的堆节点内存空间。

新的算法在原始 Dijkstra 算法的基础上,为每个网络节点增加了一个指向堆节点的指针,这样在算法遍历到新节点时,不仅需要构造配对堆节点,还要动态更新配对堆。新算法流程如图 7 所示。

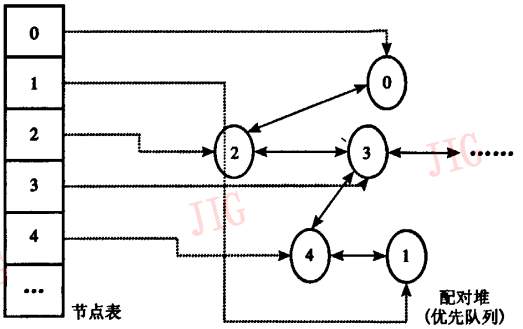

图 6　网络节点与配对堆节点的对应关系

Fig. 6　The relation between network node and pairing heap node

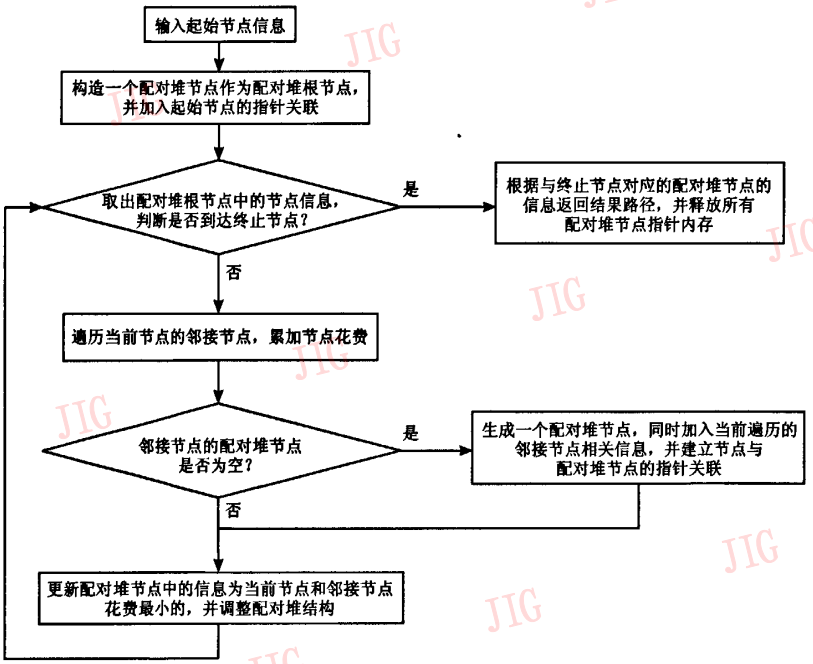

图 7　改进的 Dijkstra 算法流程

Fig. 7　The flow of improved Dijkstra algorithm

3.4　算法复杂度分析

使用配对堆实现 Dijkstra 算法的时间复杂度主要是由配对堆的 4 种操作决定的。其中“合并”、“插入”、“降级”均可在常量时间复杂度完成,其复杂度是 $O(1)$。由于“删除最小”操作的最坏情况是根节点具有 $N-1$ 个儿子节点,因此“删除最小”操作的最坏时间复杂度是 $O(N)$。采用“两趟合并法”需要对所有儿子节点进行“合并”操作,由于其能够有效减少同一层次的兄弟节点数,因此可降低最坏情况的发生机率。文献[12]指出了它能达到与斐波那契堆相同的摊还时间复杂度 $O(\log N)$,因此使用配对堆实现 Dijkstra 算法的时间复杂度为 $O(E+V\log V)$。

在空间使用上,本文配对堆节点仅包括前驱节点、左儿子节点和右兄弟节点 3 个指针。而斐波那契堆,其堆节点在实现上则需要包括父节点、孩子节点、左节点、右节点 4 个指针,以及孩子节点个数和

一个标记信息。可见本文配对堆的节点要比斐波那契堆节点节省一个指针、一个整型值和一个布尔值的内存空间。如果用指针记录堆节点的元素,那么在32bit机器下,使用配对堆的构造的优先队列比使用斐波那契堆要节省了36%的内存空间。

4 应用实例

织女星地理信息系统(VegaGIS)是在国家"863"项目"面向网络海量空间信息大型GIS"课题资助下,由中国科学院计算技术研究所开发的自主知识产权的大型分布式网络地理信息系统。其网络分析模块就使用了本文改进后的Dijkstra算法,实现了其网络分析功能(如图8所示)。在一个具有27 102个节点和34 661条弧段的城市交通网络中,查询一条对角贯穿的最短路径仅需要0.03s(测试环境:奔腾2.4G,2G内存)。

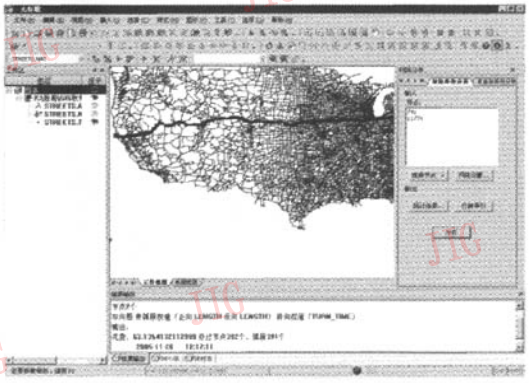

图8 最短路径查询实例

Fig. 8 An example of shortest-path search

5 结 论

本文主要提出了一种基于配对堆结构改进的Dijkstra算法,即通过分析配对堆结构的特点,给出了其用于实现可降级优先队列的4种基本操作。对比其他的堆结构,采用配对堆实现的优先队列,不仅算法的时间复杂度低,而且要比具有同等时间复杂度的其他堆结构节省内存空间。这样通过为GIS网络节点增加堆节点指针的方式,就能够实现复杂度为$O(E+V\log V)$的Dijkstra算法。在一个实例系统中,采用本文算法实现了GIS网络分析的路径分析功能,并取得了较好的效果,这就说明了基于配对堆实现Dijkstra算法的可行性。"删除最小"是配对堆的重要操作,采用何种合并算法是提高基于配对堆的Dijkstra算法的效率关键。有关配对堆的理论分析和实践检验,也是今后进一步的研究目标。

参考文献(References)

1 LU Feng, LU Dong-mei, CUI Wei-hong. Time shortest path algorithm for restricted searching area in transportation networks[J]. Journal of Image and Graphics, 1999, 4(10): 849~853. [陆锋, 卢冬梅, 崔伟宏. 交通网络限制搜索区域时间最短路径算法[J]. 中国图象图形学报, 1999, 4(10): 849~853.]

2 YAN Han-bing, LIU Ying-chun. A new algorithm for finding Shortcut in a city's road net based on GIS technology[J]. Chinese Journal of Computers, 2000, 23(2): 210~215. [严寒冰, 刘迎春. 基于GIS的城市道路网最短路径算法探讨[J]. 计算机学报, 2000, 23(2): 210~215.]

3 WANG Kai-yi, ZHAO Chun-jiang, XU Gui-xian, *et al.* A high-efficiency realization way of the shortest path search problem in GIS field[J]. Journal of Image and Graphics, 2003, 8(8): 951~956. [王开义, 赵春江, 胥桂仙等. GIS领域最短路径搜索问题的一种高效实现[J]. 中国图象图形学报, 2003, 8(8): 951~956.]

4 Sedgewick R, Vitter J S. Shortest paths in Euclidean graphs[J]. Algorithmica, 1986, 1(1): 31~48.

5 Ikeda T, Hsu M Y, Imai H, *et al.* A fast algorithm for finding better routes by AI search techniques[A]. In: Proceedings of IEEE Vehicle Navigation and Information Systems Conference[C], Yokohama, Japan, 1994: 291~296.

6 Goldberg A V, Harrelson C. Computing the shortest path: A* search meets graph theory[A]. In: 16th Annual ACM-SIAM Symposium on Discrete Algorithms(SODA'05)[C], Vancouver, Canada, 2005: 156~165.

7 Philip Klein, Satish Rao, Monika Rauch, *et al.* Faster shortest-path algorithms for planar graphs[A]. In: Proceedings of Annual ACM Symposium on Theory of Computing[C], Montreal, Quebec, Canada, 1994: 27~37.

8 Johnson D B. Priority queues with update and finding minimum spanning trees[J]. Information Processing Letters, 1975, 4(3): 53~57.

9 Brown M R. Implementation and analysis of binomial queue algorithms[J]. SIAM Journal on Computing, 1978, 7(3): 298~319.

10 Fredman M L, Tarjan R E. Fibonacci heaps and their uses in improved network optimization algorithms[J]. Journal of the ACM, 1987, 34(3): 596~615.

11 Fredman M L, Sedgewick R, Sleator D D, *et al.* The pairing heap: A new form of self-adjusting heap[J]. Algorithmica, 1986, 1(1): 111~129.

12 Stasko J T, Vitter J S. Pairing heaps: experiments and analysis[J]. Communications of the ACM, 1987, 30(3): 234~249.

13 Mark Allen Weiss. Data Structures and Algorithm Analysis in C (Second Edition, Chinese version)[M]. Beijing: China Machine Press, 2005: 372~376. [(美)Mark Allen Weiss著. 冯舜玺译. 数据结构与算法分析——C语言描述(第二版)[M]. 北京: 机械工业出版社, 2005: 372~376.]