

# 求图中受顶点数限制的所有最短路径的算法

王卫强, 孙 强

(华东师范大学 计算机科学技术系, 上海 200062)

**摘 要:** 提出了图中从一个顶点到另一个顶点的求受顶点数限制的所有最短路径的一个算法, 算法基于逆邻接表、最短路径生成树和叶子指针链表等几种特殊的数据结构。对算法进行了详细的理论分析, 分析结果表明该算法实现简单、效率较高, 且易于描述、实现和理解, 并用 C 语言设计了相应的程序验证了该算法。

**关键词:** 逆邻接表; 限制; 最短路径; 生成树; 时间复杂度

**中图法分类号:** TP311 **文献标识码:** A **文章编号:** 1000-7024 (2008) 07-1754-04

## Algorithm of all vertices-constrained shortest path

WANG Wei-qiang, SUN Qiang

(Department of Computer Science and Technology, East China Normal University, Shanghai 200062, China)

**Abstract:** An algorithm for all vertices-constrained shortest paths is put forward bases on some special data structures, such as inverse adjacency list and uses a minimum spanning tree and the pointer list that marks leaves of the trees. The theoretical analysis show that the efficiency of this algorithm is high, and is that it is very simple and very easy to be described, fulfilled and understood. And using a C program testifies its quality.

**Key words:** inverse adjacency list; constrain; shortest path; spanning tree; time complexity

## 0 引 言

最短路径算法有着广泛应用<sup>[1]</sup>, 在许多研究和应用领域起着核心作用<sup>[2]</sup>, 如在交通运输和通信网络中, 最短路径问题有着重要地位<sup>[3,4]</sup>。其中受顶点数限制的最短路径问题有着实用价值, 如: 在交通运输中, 某汽车队从甲地到乙地, 假设选择路线的指标有两个: 其一是距离短; 其二是沿途中尽可能少的经过一些城市点(这是从时间上考虑到由于城市交通拥挤, 经过一个城市需较多的时间)。即: 汽车队往往要求所选择的行车路线在沿途经过城市的个数不超过  $k$  个的限制条件下, 总路程最短<sup>[5]</sup>。在计算机网络系统中, 数据包转发也尽可能少的经过一些站点并使经过的路程最短, 因为数据包在内存中存储转发需要耗费大量时间。

文献[5]针对周算法时间复杂度较大, 且数据存储结构较复杂等不足, 提出自己的算法, 对比周算法有很大的改进, 文献[6]在文献[5]的基础上提出了自己的改进算法。但他们都有一个不足之处: 即只能求出一个顶点到另一个顶点的受顶点数限制的一条最短路径, 而不能求出这两个顶点的所有受顶点数限制的最短路径。本算法解决了求受顶点数限制的所有最短路径的问题, 解决此问题在网络、交通等领域有重要的实用价值。

## 1 存储结构

设图  $G$  共有  $n$  个顶点, 它们的编号分别为  $0, 1, \dots, n-1$ , 源点为  $s$ , 终点为  $t$ ,  $k$  为受限制的顶点数 ( $k \leq n$ )。

(1) 长度为  $n$  的指针数组  $\text{adjlist}[i] (1 \leq i \leq n)$ , 它是一个逆邻接表, 其中的每个元素  $\text{adjlist}[i]$  指向一个链表, 链表中的每个元素到  $v_i$  都有直接通路, 并且所有到  $v_i$  有直接通路的顶点都在这个链表内。

(2) 一维数组  $\text{dist}[j] (0 \leq j \leq n-1)$ ,  $\text{dist}[j]$  中存放从源点  $s$  到顶点  $j$  的当前最短距离, 初始化为源点  $s$  到顶点  $j$  的直接距离。

(3) 一维数组  $\text{pdist}[j] (0 \leq j \leq n-1)$ ,  $\text{pdist}[j]$  的值等于上一轮循环中  $\text{dist}[j]$  的值, 即  $\text{pdist}$  数组记录了上一轮  $\text{dist}$  数组的值, 以便比较  $\text{dist}$  数组中对应元素在新一轮循环后是否被修改。初始化为源点  $s$  到顶点  $j$  的直接距离。

(4) 一维数组  $\text{flag}[m] (0 \leq m \leq k-2)$ , 标记终结点  $t$  的直接前驱结点是否有效。设图中顶点  $V_j$  有  $a (a > 1)$  个直接前驱结点  $V_p$ ,  $V_p$  满足下面两个条件:  $\text{cost}(V_p, V_j) \neq \infty$  (即  $\langle V_p, V_j \rangle$  为一条边) 且  $\text{dist}(s, V_p) + \text{cost}(V_p, V_j) = \text{dist}(s, V_j)$ , 其中  $\text{cost}$  为边上的权值 ( $\text{cost}$  的值为  $\max$  表示两顶点之间没有直接通路),  $\text{dist}$  为最短路径长度。满足上述两个条件的结点  $V_p$  称为  $V_j$  的有效直接前驱结点。 $\text{dist}$  的值是在程序中不断优化计算所得, 因此数

收稿日期: 2007-05-18 E-mail: wwq\_1982@126.com

基金项目: 国家自然科学基金项目 (60673048)。

作者简介: 王卫强 (1982—), 男, 山东威海人, 硕士研究生, 研究方向为软件理论与算法; 孙强 (1963—), 男, 安徽合肥人, 副教授, 研究方向为软件理论与算法。

组 flag 的值也是随程序的执行动态变化的。flag[m] = 1 表示终结点 *t* 的直接前驱结点有效, 反之, flag[m] = 0 为无效。初始化 flag[m] = 0。

(5) 有效直接前驱结点链表, 是由表结点组成。表结点由 data 域和 next 域构成。其中, data 域存放结点的值, next 指向链表中的下一个结点。其链表中首结点由二维数组 pre 中的指针标记。结点类型定义如下:

```
typedef struct Arcnode {
    int data;
    struct Arcnode *next;
}ArcNode;
```

(6) 二维指针数组 pre[m][j] ( $0 \leq m \leq k-2, 1 \leq j \leq n$ ), pre[m][j] 指向从源点 *s* 出发经过最多 *m* 个中间顶点(不包括源点和终点)最后到达顶点 *j* 的当前最短路径上顶点 *j* 的所有有效直接前驱结点的链表的头结点。

(7) 二维标记数组 seq[m][i] ( $0 \leq m \leq k-2, 0 \leq i < n$ ), 用于标记当前顶点的 pre 数组是否发生变化, 防止在下次循环中 pre 链表中产生重复的结点, 若 dist 的值更改(优化)或 dist 的值不变且上轮循环 seq[m-1][i] = 1, 则令 seq[m][i] = 1。初始化 seq[0][0] = 0, seq[0][i] = 1 ( $0 < i < n$ ), seq[m][i] = 0 ( $0 < m \leq k-2, 0 \leq i < n$ )。

(8) 树的扩充标准形式存储结构, 树中结点形式如下:

data	child0, child1, ..., childM-1	parent
------	-------------------------------	--------

data 域存放结点的值; child 域指向最短路径树中此结点的孩子结点, 在图 G 中即为此结点的有效直接前驱结点, 最多有 *M* 个(*M* 为图 G 的入度); parent 指向最短路径树中此结点的父亲结点。结点类型定义如下:

```
typedef struct node {
    int data;
    struct node *child[M];
    struct node *parent;
}NODE;
```

(9) 叶指针链表, 用于标记各生成树的叶子结点。包含两个域: leaf 和 link, leaf 指向生成的最短路径树中的叶子结点, link 指向叶指针链表中下一个结点。结点类型定义如下:

```
typedef struct Leaf {
    struct node *leaf;
    struct Leaf *link;
}LEAF;
```

## 2 算法思想

本算法是基于下面两个等式: 等式(1)和等式(2)。设  $dist[j]^m$  ( $0 \leq m \leq k-2$ ) 表示从源点 *s* 出发经过最多 *m* 个中间顶点(不包括源点和终点)最后到达顶点 *j* 的最短路径的长度(即: 顶点数限制为 *m*+2 时从源点 *s* 到顶点 *j* 的最短路径长度,  $1 \leq j \leq n$ )。

显然有式(1)如下, 即初始化为源点 *s* 到 *j* 的直接距离

$$dist[j]^0 = cost[s][j] \quad (1)$$

而

$$dist[j]^m = \min\{dist[j]^{m-1}, dist[i]^{m-1} + adjlist[i] \rightarrow cost\} \quad (2)$$

式中:  $1 \leq i, j \leq n, 0 \leq m \leq k-2$ 。关于式(2)证明可参见文献[3]。

基于上面的两个等式, 我们就可以求出从源点 *s* 到终点 *t* 的最多经过 *m* 个中间顶点的最短路径长度, 且可得到二维指针数组 pre 和标志数组 flag, 通过这两个数组, 就可以生成以终结点为根, 叶子结点为源的多棵树, 每棵树的高度分别对应满足顶点数限制条件的一个值; 并用指针链表中的 leaf 指针标记各个叶子结点, 那么从每一个叶子结点到根结点的路径就是满足顶点数限制的一条最短路径。遍历完叶子指针链表就可以求出受顶点数限制的所有最短路径。

## 3 算法实现

### 3.1 算法的核心程序

(1) 初始化数组: pdist[i], dist[i], pre[0][i], flag[m], seq[m][i] ( $0 \leq i < n, 0 \leq m \leq k-2$ )。

(2)  $m = m + 1$ , 如果  $m > k-2$ , 则转(8);  $i = 0$ 。

(3)  $i = i + 1$ , 如果  $i > n$ , 则转(6)。

(4) 如果  $(dist[i] > pdist[adjlist[i] \rightarrow vertex] + adjlist[i] \rightarrow cost)$ , 则将上式右边的值赋给左边, 即修改 dist[i]; 并将值为 *i* 的表结点赋给 pre[m][i], 即 pre[m][i] 指向此表结点; seq[m][i] = 1。

如果  $(dist[i] = pdist[adjlist[i] \rightarrow vertex] + adjlist[i] \rightarrow cost)$  且 seq[m-1][i] = 1, 则将值为 *i* 的表结点插入到 pre[m][i] 所指向的链表, seq[m][i] = 1; 若 *i* 为终结点, 则 flag[m] = 1。

(5) 如果 adjlist[i] → next 非空, 则 adjlist[i] 指向下一结点; 转(4)。

(6) 如果  $(dist[i] = pdist[i] \text{ 且 } seq[m][i] = 0)$ , 则 pre[m][i] = pre[m-1][i]。

(7) 如果 *i* 为终结点 *t*, 且  $dist[i] < pdist[i]$  的值, 则 flag[m] = 1 且 flag[x] = 0 ( $0 \leq x < m$ )。

(8)  $pdist[u] = dist[u]$  ( $1 \leq u \leq n$ ); 转(2)。

(9) 打印 dist[i] 的值, 即为受顶点数限制的最短路径长度。

(10) 置初值:  $m = 0, i = 0, j = t$ , 空 Leaf 结点。

(11)  $m = m + 1$ , 如果  $m > k-1$ , 则转(13)。

(12) 如果 flag[m] = 1, 表明此时终点 *t* 的直接前驱结点有效, 生成树根结点, 并根据此有效直接前驱结点, 调用递归函数 Tree(*m*, *q*, *j*)。生成最短路径树, 此树的高度为 *m*, 并将树叶结点插入叶链表中, 转(11)。

(13) 遍历叶链表 *h*, 即从各生成树的叶子结点, 求出从 *s* 到 *t* 受顶点数限制的所有最短路径。

### 3.2 根据 pre[] 生成树的递归程序

Tree(int m, struct NODE \*q, int j)

(1) 初始化:  $a = g = 0, v = pre[m][j]$ 。

(2) 当 *v* 不为空, 转(3)。

(3) 根据 *v* 的值在树中插入结点 *r*; 若 *r* 为树叶结点, 则在叶链表中插入, 返回。  $a = m - 1, g \rightarrow r \rightarrow data$ , 若  $a \geq 0$ , 调用 Tree(*a*, *r*, *g*)。 *v* 指向下一个结点。转(2)。

## 4 算法分析

第(1)步赋初值, 时间为  $O(n)$ 。

第(2)步到第(8)步是算法的主要步骤, 是一个三重循环, 求出了从源点到终点受顶点数限制的最短路径长度, 并求出了各个顶点的有效直接前驱顶点, 即二维指针数组 pre。第(2)

步为最外层循环,时间为 $O(k-2)$ ;第(3)步为中间层循环,时间为 $O(n)$ ;第(4)(5)步为内层循环,循环次数与各个顶点 $v_i$ 有关(若图中到有直接通路的顶点数为 $e_i$ ,则循环 $e_i$ 次),有向图中弧和顶点的关系为 $\sum_{i=1}^n e_i = w$ ,则由(3)(4)(5)构成的循环,时间最坏为 $O(w)$ ( $w$ 为有向图中弧的总数)。

第(6)步时间为 $O(1)$ 。第(7)步时间为 $O(m)$ 。第(8)步时间为 $O(n)$ 。

所讨论的有向图不包括退化为树或者单个顶点的情况,即有 $w>n$ ,因此第(2)到(8)步的时间复杂度为 $O((k-2)*w)$ 。

第(11)(12)步是根据pre数组生成最短路径树和叶指针链表。第(11)步时间为 $O(k-2)$ 。第(12)步的时间取决于递归函数Tree( $m, g, j$ )。

递归算法的复杂性分析依赖于相应的循环关系的可解性,求解循环关系的一般技术是令 $T(n)$ 表示算法在问题规模为 $n$ 时所执行的步数,算法的递归体部分转变为 $T(n)$ 上的循环关系,该循环关系的解则为算法的复杂性函数<sup>[7]</sup>。对于递归函数Tree,其算法复杂性可以转化为生成的最短路径树的结点数,每生成最短路径树的一个结点就调用一次Tree函数,因此递归函数Tree的时间为最短路径树的结点数(根结点除外)。假设图 $G$ 中受顶点数限制的最短路径实际上共有 $sh(G, i, j)$ 条,对于每条路径上的顶点数都不大于 $k-1$ ,因此递归函数Tree的最坏情况的复杂度为 $O((k-1)*sh(G, i, j))$ ,即为循环生成最短路径树Tree函数的一个上限值。

第(13)步是根据叶子指针链表输出所有的最短路径,有 $sh(G, i, j)$ 条最短路径,且每条最短路径所包含的顶点数都不会超过 $k$ ,其时间最坏情况下为 $O(k*sh(G, i, j))$ 。

所以,本算法的时间复杂度为

$$\max\{O((k-2)*w), O(k*sh(G, i, j))\}$$

从实际情况来分析,因为大多数实际使用的带权有向图 $G$ 中,从一个顶点到另一个顶点的所有受限制的最短路径的条数不会太多(有4条至5条最短路径的情况就比较少见了),即在大多数实际问题中算法的时间复杂度为 $O((k-2)*w)$ 。

也会有极少数实际问题中出现 $sh(G, i, j) \geq w$ 的情况,此时算法的时间复杂度为 $O((k-1)*sh(G, i, j))$ 。可见本算法是一个高效的算法。

## 5 实例

设 $s=0, t=6, k=4$ ,用本算法求图1中顶点数限制值为4的情况下从顶点0到顶点6的所有最短路径。图的逆邻接表如图2所示。求解最短路径长度、二维指针数组pre和标志flag过程如表1( $\infty$ 表示max)和图3所示,根据pre和flag生成的最短路径树如图4所示,根据生成树和叶指针链表求出的受顶点数限制的所有最短路径为(0,2,6),(0,4,5,6),(0,2,3,6),(0,4,2,6),

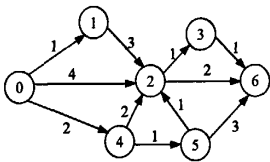


图1 顶点数限制值为4的图

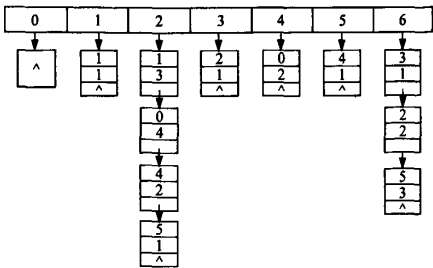


图2 有向图的逆邻接表存储结构

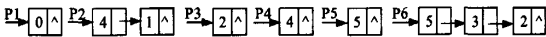


图3 pre数组中各个指针所指向的ArcNode结点

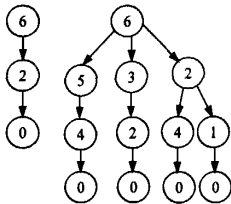


图4 根据pre和flag生成的树

(0,1,2,6)。表1中的pre数组中的指针指向图3中的ArcNode结点链表。

表1 最短路径长度和二维数组pre求解过程

m	dist							pre						
	0	1	2	3	4	5	6	0	1	2	3	4	5	6
0	0	1	4	$\infty$	2	$\infty$	$\infty$	P1	P1	P1	P1	P1	P1	P1
1	0	1	4	5	2	3	6	P1	P1	P2	P3	P1	P4	P3
2	0	1	4	5	2	3	6	P1	P1	P5	P3	P1	P4	P6

## 6 结束语

该算法在实际应用中有重要的使用价值,可广泛应用于交通、网络寻优等领域<sup>[8]</sup>,如:在交通中,由于某个城市或某条线路出现问题,或者在网络通信中,由于某个路由器等硬件或某条通信线路发生故障,我们就可以从所有最短路径中选择完好的路径,使得延迟和代价等达到最小,且本算法比较简单,易于描述、实现和理解,但是本算法在生成最短路径树的过程中,孩子结点定义为图的最大入度,浪费了一定的存储空间,这也是本算法将要继续考虑得问题,如何做到时间和空间的一个最佳结合点。但在算法的复杂性分析中,对时间复杂性考虑的更多<sup>[9]</sup>,所以这是对此类问题的一种有意义的探索和研究。

## 参考文献:

- [1] 孙强,沈建华,顾君忠. 求图中顶点之间所有最短路径的一种实用算法[J]. 计算机工程,2002,28(2):134-136.
- [2] 谭国真,高文. 时间依赖的网络中最小时间路径算法[J]. 计算机学报,2002,25(2):165-172.

[3] 戴树贵,潘荫荣,胡幼华,等. 求带多个限制条件的单源多权最短路算法[J]. 计算机应用与软件,2004,21(12):78-81.

[4] 谭国真,隋春丽. PC 机群环境下最短路并行算法的研究[J]. 小型微型计算机系统,2001,22(11):1302-1304.

[5] 孙强,杨宗源. 求受顶点数限制的最短路径问题的一个算法[J]. 计算机工程,2002,28(9):73-74.

[6] 钟子飞,黄水松,伍磊. 一种求受顶点数限制的最短路径的新算

法[J]. 计算机工程与设计,2004,25(7):1114-1115.

[7] John R Hubbard. Data structures with Java[M]. America: The McGraw-Hill Companies Inc, 2001.

[8] 徐凤生. 最短路径的求解算法 [J]. 计算机应用, 2004,24 (5): 88-89.

[9] 郑宗汉,郑晓明. 算法设计与分析[M]. 北京:清华大学出版社, 2005.

(上接第 1700 页)

表 2 结合 P2P 技术的分层缓存服务器集群中映射表的结构

Object-ID	Location	$T_{live}$	$T_{average}$	Live
-----------	----------	------------	---------------	------

结合了 P2P 技术的分层缓存服务器集群的适应性缓存策略如下(如图 2 所示):

(1)用户 Client A 向缓存服务器 Cache A 请求某网页内容,如果 Cache A 中缓存了该对象,则直接将内容返回给 A,同时根据 2.1 中映射表的内容替换策略对映射表的内容进行修改;

(2)如果 Cache A 中没有缓存用户请求的对象,Cache A 从其映射表中发现 Cache B 缓存了用户请求的对象,并且该对象 Object-ID 对应的 Live 项为 1,则 Cache A 将该请求直接转发给 Cache B,Cache B 将该对象内容返回给 Cache A,Cache A 缓存该对象,同时从映射表中删除对应对象的记录,并根据 2.1 中映射表的内容替换策略对映射表的内容进行修改;

(3)如果 Cache A 从其映射表中发现 Cache B 缓存了用户请求的对象,但是该对象 Object-ID 对应的 Live 项为 0,则 Cache A 直接将该请求转发给原始服务器 Server A,同时删除映射表中对应对象的记录,原始服务器 Server A 返回对象内容,Cache A 缓存该对象的内容,并将其转发给用户,同时根据 2.1 中映射表的内容替换策略对映射表的内容进行修改;

(4)如果 Cache A 没有在其本地缓存以及映射表中发现用户请求对象,则 Cache A 将用户请求直接转发给原始服务器 Server A,原始服务器 Server A 返回该对象的内容,Cache A 缓存该内容,并转发给用户,同时根据 2.1 中映射表的内容替换策略对映射表的内容进行修改。

Live 选项的内容由定时更新策略不断更新,在一定的周期内,每个缓存服务器根据映射表中 Object-ID 的 Location 对目的缓存服务器发送一定数量的 ICMP 包(该包数量很少,类似于 Windows 中的一次 Ping 命令所发送的 4 个数据包),对方缓存服务器在规定的 TTL 时间内没有响应,则说明目的缓存服务器已经关闭或者退出该群,此时映射表中的对应的该对象的 Live 项要被设置为 0,为防止 Live 项为 0 的缓存服务器“暂时休眠”,暂不从映射表中删除该项,而是等待下次 ICMP 包的检查结果,如果确实没有回应信息,则该项被删除出映射表,这样就节省了宝贵的内存空间。在等待 ICMP 包的检查结果的时候,如果有用户要求访问 Live 项为 0 的对象,则拥有该对象 Object-ID 的缓存服务器首先删除映射表中该项的内容,并直接将用户的请求转发至原始服务器,然后缓存从原始服务器返回的该对象的内容,与此同时,将该内容转发给用户,

这样,有下一个用户访问该对象时,缓存服务器就可以从自身的缓存中找到该对象,从而提高了集群边缘缓存字节命中率。

## 5 结束语

一般的缓存服务器集群的适应性缓存策略<sup>[1,9-10]</sup>采用的是 LRU (least-recently-used) 算法、LFU (least-frequently-user) 算法、LLF(lowest-latency-first)算法和 SIZE 算法等,以上算法实现较为简单,但是没有充分考虑到用户访问特性,无法有效的提高缓存服务器集群的边缘缓存字节命中率。结合了 P2P 技术的缓存服务器集群的适应性缓存策略,极大的节省了网络带宽、减少了网络延迟、提高了文件和文件字节命中率,尤其极大的提高了集群边缘缓存字节命中率,并且该系统具有可扩充性,可以随时加入一台或者多台缓存服务器。在负载方面,集群内的各个缓存服务器相互协作,有效的保持了负载均衡。文件共享方面实现了快取分享,同时很好的避免了网络拥塞,最大限度的改善了网络带宽的使用。

## 参考文献:

[1] 贺琛,陈肇雄,黄河燕. Web 缓存技术综述[J].小型微型计算机系统,2004,25(5):836-842.

[2] 金志刚,张钢,舒炎泰. 基于网络性能的智能 Web 加速技术-缓存与预取[J].计算机研究与发展,2001,38(8):1000-1004.

[3] 李文中,顾铁成,周俊,等. 适应性 Web 缓存的研究[J].计算机科学,2005,32(4):11-15.

[4] 周文莉,吴晓非. P2P 技术综述[J].计算机工程与设计,2006,27(1):76-79.

[5] 黄立德,谢文雄. 具可扩充性的协同合作式 Web Caching 共享架构[DB/OL]. <http://www.nsysu.edu.tw/TANET99/Download/TANET009/TANET009.DOC>, 1999-10-23/2007-03-28.

[6] 高勇,李子木,吴建平. CERNET 上 CDN 性能的研究[J]. 计算机工程,2002,28(S1):211-215.


[7] Sahasrabudhe L H, Mukherjee B. Multicast routing algorithms and protocols a tutorial[J]. IEEE Network, 2000,14(1):90-102.

[8] 陈姝,方滨兴,周勇林. p2p 技术的研究与应用[J]. 计算机工程与应用,2002,26(3):20-23.

[9] 林永旺,张大江,钱华林. 一个基于集中管理的协作式 Web 缓存系统[J].计算机研究与发展, 2001,38(1):68-73.

[10] 张震波,杨鹤标,马振华. 基于 LRU 算法的 Web 系统缓存机制[J].计算机工程,2006,32(19):68-70.

# 求图中受顶点数限制的所有最短路径的算法

作者: 王卫强, 孙强, WANG Wei-qiang, SUN Qiang  
作者单位: 华东师范大学计算机科学技术系, 上海, 200062  
刊名: 计算机工程与设计   
英文刊名: COMPUTER ENGINEERING AND DESIGN  
年, 卷(期): 2008, 29(7)  
被引用次数: 9次

## 参考文献(9条)

1. 孙强, 沈建华, 顾君忠. 求图中顶点之间所有最短路径的一种实用算法[期刊论文]-[计算机工程](#) 2002(2)
2. 谭国真, 高文. 时间依赖的网络中最小时间路径算法[期刊论文]-[计算机学报](#) 2002(2)
3. 戴树贵, 潘荫荣, 胡幼华, 孙强. 求带多个限制条件的单源多权最短路径算法[期刊论文]-[计算机应用与软件](#) 2004(12)
4. 谭国真, 隋春丽. PC机群环境下最短路径并行算法的研究[期刊论文]-[小型微型计算机系统](#) 2001(11)
5. 孙强, 杨宗源. 求受顶点数限制的最短路径问题的一个算法[期刊论文]-[计算机工程](#) 2002(9)
6. 钟子飞, 黄水松, 伍磊. 一种求受顶点数限制的最短路径的新算法[期刊论文]-[计算机工程与设计](#) 2004(7)
7. John R Hubbard. [Data structures with Java](#) 2001
8. 徐凤生. 最短路径的求解算法[期刊论文]-[计算机应用](#) 2004(5)
9. 郑宗汉; 郑晓明. [算法设计与分析](#) 2005

## 本文读者也读过(10条)

1. 张权范. ZHANG Quan-fan. 求解PERT两点间最短路径的Floyd算法分析与程序实现[期刊论文]-[中国制造业信息化](#) 2008, 37(11)
2. 钟子飞, 黄水松, 伍磊. 一种求受顶点数限制的最短路径的新算法[期刊论文]-[计算机工程与设计](#) 2004, 25(7)
3. 贺鹏, 殷亚君. 最短路径算法浅析[期刊论文]-[甘肃科技](#) 2010, 26(2)
4. 宋丽敏. Song Limin. 最短路径的编程实现[期刊论文]-[华北航天工业学院学报](#) 2001, 11(4)
5. 何清林. Floyd算法的景区最短路径查询系统的设计与实现[期刊论文]-[电脑编程技巧与维护](#) 2010(1)
6. 湛文红. Zhan Wenhong. 网络图形最短路径算法分析与研究[期刊论文]-[电脑与电信](#) 2010(7)
7. 王卫强. 求图中受顶点数限制的所有最短路径的算法分析研究[学位论文] 2007
8. 余伟辉, 陈闳中. 时间依赖有向无环网最小时间路径算法[会议论文]-2008
9. 余伟辉, 陈闳中, YU Wei-hui, CHEN Hong-zhong. 时间依赖有向无环网最小时间路径算法[期刊论文]-[计算机工程与科学](#) 2008, 30(11)
10. 柳亚玲, 邹伟松. 随机时间依赖网络的最短路径[会议论文]-2001

## 引证文献(8条)

1. 董俊, 黄传河. 改进Dijkstra算法在GIS导航应用中最短路径搜索研究[期刊论文]-[计算机科学](#) 2012(10)
2. 孙强, 徐远涛. 地理信息系统中Dijkstra算法的改进与研究[期刊论文]-[硅谷](#) 2009(19)
3. 况超. 求图中每对顶点间的所有最短路径算法的分析与研究[学位论文] 硕士 2010
4. 赵福生. 一种源顶点到其他各顶点所有路径的算法及其Web服务设计[期刊论文]-[长江大学学报\(自然版\)](#) 理工卷 2013(03)
5. 孙刚明. 求受顶点数限制的所有最短路径的一个算法[期刊论文]-[广西教育学院学报](#) 2012(03)
6. 冯震, 刘佳, 李靖, 曹延飞. 复杂网络中最短路径问题的求解算法研究[期刊论文]-[自动化技术与应用](#) 2010(03)

7. [沈海澜, 王玉斌, 陈再良, 曹子文](#) [一种基于分层图的改进SPFA算法](#)[期刊论文]-[计算机工程](#) 2012(13)
8. [黄书力, 胡大裘, 蒋玉明](#) [经过指定的中间节点集的最短路径算法](#)[期刊论文]-[计算机工程与应用](#) 2015(11)

引用本文格式: [王卫强, 孙强](#). [WANG Wei-qiang. SUN Qiang](#) [求图中受顶点数限制的所有最短路径的算法](#)[期刊论文]-[计算机工程与设计](#) 2008(7)