文章编号:1001 - 9081(2010)11 - 2965 - 02

有效的哈希冲突解决办法

张朝霞,刘耀军
(太原师范学院 计算机系, 太原 030012)
(zzxcn@ sina. com)

摘　要:为了提高解决哈希冲突的效率,在冲突解决机制和数据元素被查找的先验概率的基础上,结合堆排序的优点,提出了一种更有效的处理哈希冲突的方法,称其为以先验概率为基础的哈希大顶堆查找。该方法首先依据关键字被查的先验概率的大小建立相应的哈希大顶堆,然后利用哈希大顶堆进行查找。最后通过严密的效率分析可看出:该方法在最坏的情况下的时间复杂度才为 $O(n \log n)$,不但降低了冲突时执行查询的查找长度,从而降低查询响应的时间复杂度,而且该方法对于记录数越大的文件越适用。

关键词:链地址法;哈希冲突;先验概率;哈希查找;哈希平衡树
中图分类号: TP311.13　　**文献标志码**:A

Effective solution to Hash collision

ZHANG Zhao-xia, LIU Yao-jun
(*Department of Computer, Taiyuan Normal University, Taiyuan Shanxi* 030012, *China*)

Abstract: To improve the efficiency of Hash conflict-solving, a more effective method to deal with Hash collision was proposed based on a conflict-solving mechanism and the prior probability of the searched elements and combined with the advantages of heap sort. This method was defined as a prior probability-based Hash heap big top, which established a corresponding Hash heap big top based on the prior probability of the searched elements and then began the research with it. The time complexity of searching is $O(n \log n)$ at worst. Hence, this algorithm can not only reduce the searching length when conflicts occur so as to shorten the time complexity of searching, but also be applicable to huge sets of keys.

Key words: chain addressing; Hash collision; prior probability; Hash search; Hash AVL tree

0　引言

数据查找主要有顺序匹配、二分查找和哈希查找几种。顺序匹配需要一个一个地匹配,所以效率低下,不适合大规模查找;二分查找需要数据有序,条件比较苛刻;而哈希查找一次就可以定位到数据,查找速度很快,该方法由于其查找速度快,查询、插入、删除容易操作等原因获得了广泛的应用[1-3]。

虽然哈希表查找得到了很广泛的应用,但是哈希表对哈希函数的设计有很高要求,若哈希函数设计得不好,就会产生哈希冲突,理论上查找的最坏复杂度为 $O(n)$。目前,有不少研究人员提出了完美哈希函数[4-6],就是没有冲突的哈希函数,即:函数 H 将 n 个 K 值映射到 m 个整数上,这里 $m \geqslant n$,而且,对于任意的关键字 K_1、K_2,$H(K_1)$!= $H(K_2)$,并且,如果 $m = n$,则 H 是最小完美哈希函数,但是这些函数都是针对小词典有效的方法,同时,不能保证对任意词典构造完美哈希函数。也就是说哈希函数其实是非常难以找到的,好的哈希函数更是很少[4]。因此,高效地解决哈希冲突,降低冲突时执行查询的查找长度,从而使查询响应时间更短成了本文关注的焦点。

哈希是一种重要的存储方法,也是一种重要的查找方法。它的基本思想是:以关键字 K 为自变量,通过构造一个确定的函数 f,计算出对应的函数值 $f(k)$,$f(k)$ 即为关键字等于 K 的节点的存储地址。查找时,再根据要查找的关键字用同样的函数 $f(k)$ 计算地址,然后到相应的存储单元取出要查找的节点。按这个思想建立的表,称为哈希表,称函数 f 为哈希函数,称 $f(k)$ 的值为哈希地址。但事实上,很难构造一个哈希函数使得所有关键字的哈希地址都不一样,因而便会出现对于关键字 k_i 和 k_j,虽然 $k_i \neq k_j$ 而 $f(k_i) = f(k_j)$,这种现象称为哈希冲突。

处理哈希冲突的方法有很多[7],如链地址法、线性探测再散列、二次再散列法,以及建立一个公共溢出区等方法。但是,在建立哈希表处理这些冲突时,往往不会综合考虑发生冲突时提高效率的解决冲突方法。比如,文献[7]采用开放定址法和链地址法解决哈希冲突。文献[8]基于关键字被查找的先验概率提出了一种改进的链地址法,该方法考虑到若字符串 S_1 和 S_2 的哈希值相同,但查找 S_1 的频率远远高于 S_2 时,在建立哈希表时应将其作为一个考虑因素,使得 S_2 连接在 S_1 之后,查找效率就会提高,尤其是在处理大批相同哈希值的情况下。文献[9]认为对这些具有相同哈希值的数据采用顺序查找方式,效率非常低。如果对哈希表中具有相同哈希值的数据不使用链式方法解决冲突,而采用 AVL(哈希平衡)树来解决冲突,对具有相同哈希值的数据的查找将变成 AVL 树的查找,比链式的顺序查找效率将提高很多。但仍存在没有考虑关键字被查找的先验概率的缺点。

1　传统的解决哈希冲突方法的优缺点

文献[7]中的拉链法与开放定址法相比,有如下几个优点:1)拉链法处理冲突简单,且无堆积现象,即非同义词决不会发生冲突,因此平均查找长度较短。2)由于拉链法中各链

收稿日期:2010 - 04 - 23;**修回日期**:2010 - 07 - 27。

作者简介:张朝霞(1975 -),女,山西运城人,讲师,硕士研究生,主要研究方向:算法设计、模式识别;　刘耀军(1963 -),男,河北阳原人,教授,博士研究生,主要研究方向:半群代数理论、形式语言、自动机。

《计算机应用》Journal of Computer Applications

表上的节点空间是动态申请的,故它更适合于造表前无法确定长的情况。3) 开放定址法为减少冲突,要求装填因子 α 较小,故当节点规模较大时会浪费很多空间。而拉链法中可取 $\alpha \geqslant 1$,且节点较大时,拉链法中增加的指针域可忽略不计,因此节省空间。4) 在用拉链法构造的散列表中,删除节点的操作易于实现,只要简单地删去链表上相应的节点即可。而对开放地址法构造的散列表,删除节点不能简单地将被删节点的空间置为空,否则将截断在它之后填入散列表的同义词节点的查找路径。拉链法的缺点是:1) 指针需要额外的空间,故当节点规模较小时,开放定址法较为节省空间,而若将节省的指针空间用来扩大散列表的规模,可使装填因子变小,这又减少了开放定址法中的冲突,从而提高平均查找速度;2) 传统的链地址法没有考虑被查关键字的先验概率。

文献[8] 考虑了关键字被查找的先验概率,提出了一种改进的链地址法,改进的哈希表建立过程是通过首先对关键字 K 进行排序,排序规则是按该关键字期望被查询的概率从大到小排序,后续步骤同原来的哈希表建立过程。新方法的理论依据就是每个关键字 K 出现的概率有差别,按出现的先验概率大小来决定其冲突处理时的位置,将出现概率大的放在所需比较次数少的位置,从而提高出现冲突情况下的查找效率,亦即提高了哈希表的整体查找效率。但是,该方法没有突破链表存储,仍然采用顺序查找。

文献[9] 的 AVL 树与文献[7 - 8] 的区别在于逻辑结构不同。文献[7 - 8] 将所有关键字为同义词的节点链接在同一个单链表中。若选定的散列表长度为 m,则可将散列表定义为一个由 m 个头指针组成的指针数组 $T[0..m-1]$。凡是散列地址为 i 的节点,均插入到以 $T[i]$ 为头指针的单链表中。T 中各分量的初值均应为空指针。而 AVL 树就是每一个表头指针不再链接一个单链表,而是作为平衡二叉树的根节点,若选定的散列表长度为 m,则可将散列表定义为一个由 m 个头指针组成的指针数组 $T[0..m-1]$。凡是散列地址为 i 的节点,均插入到以 $T[i]$ 为根节点的 AVL 树中。该方法突破了原有的存储结构,实现了新的突破,但是该方法在建立 AVL 树时没有考虑关键字的先验概率,具有相同哈希地址的关键字按关键字本身的大小构造 AVL 树。如关键字序列 k 为{9,11,20,14,1,17,22,25},查找关键字 17 的过程如图 1 中的粗线所示。

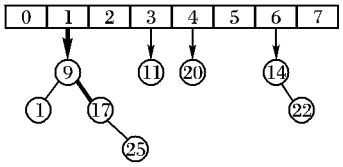

图 1 在 AVL 树中查找关键字 17 的过程

2 方法改进

2.1 收集数据元素被查询的先验概率

数据元素期望被查询的概率可以通过一些先验概率统计而获得。比如,搜索引擎中某个元素的先验概率可以通过统计前一段时间(前半年) 内该数据元素被访问次数除以哈希表中所有元素被访问的总次数,来计算得到该数据元素被查询的先验概率,即为该关键字期望被查询的概率。比如关键字序列 k 为{9,11,20,14,1,17,22,25},且每个关键字所对应的期望被查询的概率序列 p 为{900,800,950,700,960,940,600,200},则可以得到一个新的关键字序列 xk{(9,900),(11,800),(20,950),(14,700),(1,960),(17,940),(22,600),(25,200)},具体实现方法,在大顶堆的元素结构中添加关键字的字频项 count,即 xk 序列中元素结构定义为(key,count)。

2.2 改进方法

结合文献[8]提出的改进的链地址法和文献[9]的 AVL 树查找方法的优点,本文提出一种更有效的处理哈希冲突的办法,该方法是基于关键字被查找的先验概率建立的哈希大顶堆。

哈希大顶堆的存储步骤如下:1)用给定的哈希函数构造哈希表;2)对哈希地址相同的关键字建立各自的大顶堆,排序规则是按该关键字期望被查询的概率实行堆排序,该方法对记录数较少的文件并不值得提倡,但对 n 较大的文件还是很有效的。堆排序在最坏的情况下的时间复杂度为 $O(n \log n)$,而文献[8]是按直接插入法进行排序,其时间复杂度为 $O(n^2)$。

哈希大顶堆的查找步骤如下:1)用给定的哈希函数求得哈希地址;2)依据关键字的哈希地址和先验概率在相应的哈希大顶堆中查找该关键字。如关键字序列 xk{(9,900),(11,800),(20,950),(14,700),(1,960),(17,940),(22,600),(25,200)}。哈希大顶堆见图 2。

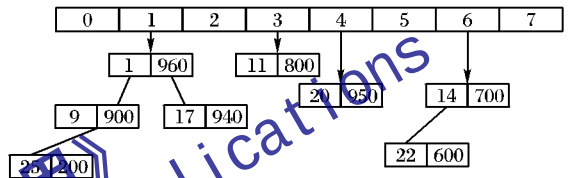

图 2 哈希大顶堆

新方法的理论依据既借了鉴文献[8],就是每个关键字出现的概率有差别,按出现的先验概率大小来决定其冲突处理时的位置,将出现概率大的放在所需比较次数少的位置,从而提高出现冲突情况下的查找效率,亦即提高了哈希表的整体查找效率。同时,又受文献[9]的启发,在逻辑结构上突破了链表结构,采用哈希大顶堆结构来实现,从而改善了查找方法,提高了效率。

2.3 效率分析

查找运算的主要操作是关键字的比较,所以通常把查找过程中对关键字需要执行的平均比较次数(也称为平均查找长度)作为衡量一个查找算法优劣的标准。文献[7]传统的链地址法处理冲突的哈希表查找成功时的平均查找长度 $ASL_{成功}$ 定义为:

$$ASL_{成功} = \sum_{i=1}^{n} p_i B_i$$

其中:n 是节点的个数;p_i 是查找第 i 个节点的概率,若不特别声明,认为每个节点的查找概率相等,即 $p_1 = p_2 = \cdots = p_n = 1/n$;$B_i$ 是找到第 i 个节点所需进行的比较次数。

文献[8]改进后的方法考虑了查询关键字为不等概率事件,使得查找关键字先验概率 p_i 大的节点排在前面,从而找到第 i 个节点所需进行的比较次数 B_i 就减小了。可以证明,相对于假设等概率查找情况 $ASL_{成功}$ 减小了。因此,新方法得到的平均查找长度小于等于原来的方法[8]。

文献[9]和文献[7]一样,都没有考虑被查关键字的先验概率,但文献[9]在查找时利用平衡二叉树的查找效率高的优点,类似于折半查找,查找的时间复杂度为 $O(n \log n)$。

本文的哈希大顶堆查找方法既考虑了被查关键字的先验概率,又采用大顶堆(完全二叉树)结构,减少了比较次数。该方法的运行时间主要耗费在建初始堆和调整建新堆时进行

(下转第 3004 页)

Journal of 《计算机应用》 Computer Applications

是一个临时存放顶点坐标数据和方便查找定位的数组,在使用完之后可以释放其所占的内存。

3 实例测试

本文在 1 台 P4 2.80 GHz CPU,512 MB 内存的 PC 上对改进算法[2]和本文算法进行了实现,并测试了几个 SMF 文件的实例,测试结果在表 1 中列出。

表 1 测试实例

测试例子名称	三角面片数	改进算法用时/s	本文算法用时/s
Cow	5 804	0.098	0.051
阀体	23 470	0.458	0.237
轮毂	50 940	1.017	0.511
Bonny	69 451	1.375	0.687
Horse	96 966	2.079	0.951
dragon	108 588	2.154	0.987

4 结语

从测试数据来看,由于改进了辅助查找索引方式,即从 AVL 树改为数组,所以拓扑重建的时间复杂度也从 $O(n \log n)$ 直接降为 $O(n)$,时间大为缩短,较大模型的效果尤其明显。

模型有了拓扑关系之后,对模型的后续处理工作变得相对容易,在缺陷修复和网格简化中利用拓扑关系可以直接找到邻接三角面片,无需遍历或借助辅助结构查找。现在可以说拓扑重建的时间复杂度不能再降低了,因为从数据文件中读取三角面片本身就需要 $O(n)$ 时间。

参考文献:

[1] BECHET E, CUILLIERE J-C, TROCHU F. Generation of a finite element MESH from stereolithography (STL) files[J]. Computer Aided Design, 2002, 34(1): 1 - 17.

[2] 刘金义, 侯宝明. STL 格式实体的快速拓扑重建[J]. 工程图学学报, 2003, 24(4): 34 - 39.

[3] 张必强, 邢渊, 阮雪榆. 面向网格简化的 STL 拓扑信息快速重建算法[J]. 上海交通大学学报, 2004, 38(1): 39 - 42.

[4] 戴宁, 廖文和, 陈春美. STL 数据快速拓扑重建关键算法[J]. 计算机辅助设计与图形学学报, 2005, 11(11): 2447 - 245.

[5] 张翔, 廖文和, 程筱胜, 等. STL 格式文件的拓扑重建方法研究[J]. 机械科学与技术, 2004, 24(9): 1093 - 1096.

[6] 侯宝明, 刘雪娜. STL 实体模型的拓扑重建及其缺陷修复[J]. 计算机工程, 2005, 31(3): 213 - 214, 217.

[7] DAI CHUNXIANG, JIANG YING, HU QINGXI, *et al*. Efficient topological reconstruction for medical model based on mesh simplification [M]. Berlin: Springer-Verlag, 2007: 526 - 535.

[8] 邱元庆, 周惠群, 朱珊珊, 等. 利用散列对 STL 文件进行拓扑重建和修复[J]. 机械科学与技术, 2009, 28(6): 795 - 802.

[9] BURKARDT J. SMF format description [EB/OL]. [2010 - 05 - 20]. http://people.sc.fsu.edu/~burkardt/data/smf/smf.txt.

[10] MANTYLA M. An introduction to solid modeling [M]. New York: W H Freeman & Co, 1988: 110 - 132.

[11] WEILER K. Edge-based data structures for solid modeling in curved-surface environments [J]. IEEE Computer Graphics and Applications, 1985, 5(1): 21 - 40.

[12] 严蔚敏, 吴伟民. 数据结构: C 语言版[M]. 北京: 清华大学出版社, 1998: 227 - 238.

(上接第 2966 页)

的反复“筛选”上。对深度为 h 的堆,筛选算法中进行关键字比较次数至多为 $2(h-1)$ 次,则在建含 n 个元素、深度为 h 的堆时,由于第 k 层上的节点数至多为 2^{k-1},以它们为根的二叉树的深度为 $h-k+1$,则调用 $\lfloor \frac{n}{2} \rfloor$ 次 HeapAdjust(堆排序的筛选算法,见文献[7]282 页的算法 10.10),该算法执行时进行的关键字比较次数为:

$$\sum_{i=h-1}^{1} 2^{i-1}(h-i) = \sum_{i=h-1}^{1} 2^{i}(h-i) = \sum_{j=1}^{h-1} 2^{h-j} \cdot j \leqslant 2n \sum_{j=1}^{h-1} j/2^{j} \leqslant 4n$$

又因为含有 n 个节点的完全二叉树的深度为:

$\lfloor \mathrm{lb}\, n \rfloor + 1$

则调整建新堆时调用 HeapAdjust 过程 $n-1$ 次,总共进行比较的次数不超过 $2n(\lfloor \mathrm{lb}\, n \rfloor)$,即:

$$2(\lfloor \mathrm{lb}\,(n-1) \rfloor + \lfloor \mathrm{lb}\,(n-2) \rfloor + \cdots + \lfloor \mathrm{lb}\, 2 \rfloor) < 2n(\lfloor \mathrm{lb}\, n \rfloor)$$

因此,堆排序在最坏的情况下,其算法执行的时间复杂度才为 $O(n \log n)$,显然比前几种方法的查找效率都高。

3 结语

综上所述,将数据元素期望被查找的先验概率作为建立哈希大顶堆时插入节点的顺序依据,查找过程与原哈希查找方法相比在不增加其他消耗的同时降低了冲突时执行查询的查找长度,从而使查询响应时间更短。但该方法对记录数 n 较小的文件并不值得提倡,仅对 n 较大的文件还是很有效的。因此,对于记录数 n 较大的场合该方法有很好的应用价值。

参考文献:

[1] KUMAR S, CROWLEY P. Segmented Hash: An efficient Hash table implementation for high performance networking subsystems [C] // Proceedings of the 2005 ACM Symposium on Architecture for Networking and Communications Systems. New York: ACM Press, 2005: 91 - 103.

[2] 王果, 徐仁佐. 结合哈希过滤的一种改进多连接查询优化算法[J]. 计算机工程, 2004, 30(7): 57 - 59.

[3] 张科. 多次 Hash 快速分词算法[J]. 计算机工程与设计, 2007, 28(7): 1716 - 1718.

[4] 完美哈希函数[EB/OL]. [2010 - 06 - 27]. http://blog.csdn.net/chixinmuzi/archive/2007/08/05/1727195.aspx.

[5] BOTELHO F C, KOHAYAKAWA Y, ZIVIANI N. A practical minimal perfect Hashing method [EB/OL]. [2010 - 06 - 28]. http://homepages.dcc.ufmg.br/~nivio/papers/wea05.pdf.

[6] BOTELHO F C, PAGH R, ZIVIANI N. Simple and space-efficient minimal perfect Hash functions [EB/OL]. [2010 - 06 - 28]. http://homepages.dcc.ufmg.br/~nivio/papers/wads07.pdf.

[7] 严蔚敏. 吴伟. 民数据结构: C 语言版[M]. 北京: 清华大学出版社, 2006: 251 - 262.

[8] 马如林. 蒋华. 张庆霞. 一种哈希表快速查找的改进方法[J]. 计算机工程与科学, 2008, 30(9): 66 - 68.

[9] 周伟明. 多任务下的数据结构与算法[M]. 武汉: 华中科技大学出版社, 2006.

《计算机应用》Journal of Computer Applications