第16卷 第1期 郑州工学院学报 Vol.16 No.1
1995年 3 月 Journal of Zhengzhou Institute of Technology Mar. 1995

用Hash表实现树的压缩存贮*

任军员

(郑州工学院土木建筑工程系)

摘 要: 本文探讨了如何使用杂凑表以一种非常压缩的形式实现树的存贮。使用这种方法，任意树中的指针均能以每个结点 $6+\log_2 n$ 比特表示（这里 n 是一个结点所能有的最大孩子数目），从而在存贮大型的树时，使所需的存贮容量能显著地减少。

关键词: 压缩杂凑，杂凑树，树结构

中图分类号: TP31

树是一种在计算机科学和软件工程中应用十分广泛的数据结构。随着计算机应用领域的不断扩展，需要在内存中存贮的树的规模也越来越大，因此必须研究树的压缩存贮形式。树的压缩存贮，指的是压缩树的结构信息，而不涉及与树相关的数据。在树的结构信息中，指针往往消耗了大多数的空间。本文探讨一种用Hash表实现树的方法，可使树的指针具有非常简短的形式，从而达到树的压缩存贮的目的。

1 树的Hash表实现

1.1 Hash表

Hash（杂凑）是一种处理符号表的常用技术。对于一个给定的关键字集合K，设计一个函数h，将关键字映射到M个连续的整数集合{0, 1, …, M−1}中去,且把这些整数解释为有M个单元的Hash表的标号。函数h称为杂凑函数。这样，任何一个关键字k∈K都有一个初始杂凑地址i＝h（k）。如果有若干个关键字的初始杂凑地址是相同的，就会发生冲突，必须使用某种算法予以解决。一组冲突的关键字称为一个冲突组。

解决冲突的算法有很多种，我们现在采用一种简单有效的算法——双向线性试探法。当一个关键字要存放到Hash表中时，双向线性试探法首先查询其初始杂凑地址，如果它是空的，则关键字被存放到这里；如果已经存放了另一个关键字，则双向线性试探法首先根据新关键字在其冲突组中的次序，确定它的存贮位置，如果新计算出来的位置是空的，则插入完成，否则依次试探此位置上下两边的一系列相邻单元，直至找到一个空位置，然后向上或向下移动其间的所有表项，为新来的关键字腾出位置。双向线性试探法按插入的

* 收稿日期: 1994-04-29

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

顺序存贮同一冲突组的关键字，并使它们被存贮在一组连续的单元中。在同一冲突组中用插入的序号 j 来区分不同关键字。j 为 4 比特时可识别最大为 16 的冲突组，j 为 5 比特时可识别最大为 32 的冲突组。可用概率的方法来确定可能杂凑到同一位置的关键字的最大数目，用以确定 j 域的大小。

另外，为了区分不同的冲突组和从初始杂凑地址能搜索到已被移动过的对应冲突组，每个杂凑表项至少还需要两个额外的比特。一个叫"未用"比特，用来描述一个特定的位置是否已经被杂凑到，"未用"比特不随冲突组移动。另一个叫"改变"比特，用来标注冲突组的边界。每一冲突组的最低单元，该比特置"1"，其余单元置"0"。未用和改变比特接合在一起，就可定位冲突组和将它们相互区分开来[1]。

1.2 树的表示

为了用 Hash 表存贮树，树的每一个结点都应被一个结点关键字所唯一地确定。结点关键字必须隐含该结点在 Hash 表中的位置信息和与其它结点的关联信息。因为一个结点在 Hash 表中的位置可由二元组<i, j>（i 表示初始杂凑地址，j 表示在其冲突组中的序号）所确定，所以从逻辑上我们可按以下办法来构造一棵树:

①如果一个结点被存贮在 T[<i, j>]，则其第 m 个孩子的结点关键字是三元组<m, i, j>。

②一个特殊的结点关键字<root>提供给根结点。

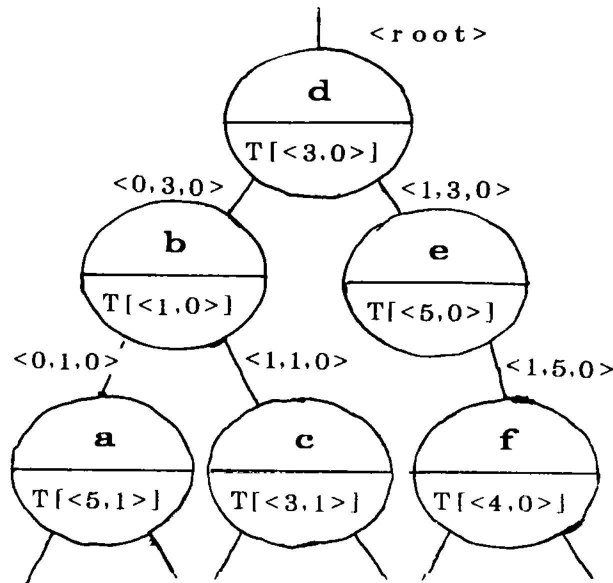

图 1

T[ ]表示 Hash 表的存贮数组，也称 Hash 向量。图 1 表示了一个用这种方法构造的二叉树，为了便于说明，每个结点给定一个单字符的标识符。例中的杂凑值也是随机选择的。

关键字 k	d,<root>	b,<0,3,0>	c,<1,3,0>	a,<0,1,0>	c,<1,1,0>	f,<1,5,0>
杂凑值 h(k)	3	1	5	5	3	4

树通过结点关键字<root>来访问。在例中，h(<root>)=3，因为根结点 d 存贮在 T[<3,0>]。它的两个孩子 b 和 c，分别具有各自的结点关键字<0,3,0>和<1,3,0>。

从树的上述表示方法中可以看出，如果给定了结点在 Hash 表中的位置，也就是说，确定了二元组<i, j>，它的任何孩子均能通过构造适当的结点关键字<m,i,j>，计算其杂凑值，找到它在 Hash 表中的存贮位置。寻找结点的双亲甚至更容易，从它的结点关键字<m,i,j>中可直接得出双亲结点在 T[<i,j>]。这种表示法使得从双亲结点到孩子和从孩子结点到双亲的访问都变得非常容易。

1.3 结点关键字的实现

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

从上节的讨论中我们可知，结点关键字（根结点除外）的逻辑结构是3元组<m,i,j>，在实际实现时，这个3元组被随机化成一个数，然后再划分成2部分：初始杂凑地址i和唯一性数（等于m域和j域的结合）。另外，为了实现从孩子结点到双亲的访问，还必须使3元组<m,i,j>能从这两个数中重构。下面详细讨论具体的实现步骤。

第一步，将整数m,i,j合并成一个数C

$$C=(m\times \mathrm{rangeof}(j)+j)\times \mathrm{rangeof}(i)+i$$
$$=(m\times 2^4+j)\times M+i \quad (\text{当 j 域取 4 比特时})$$

这里，rangeof()函数返回自变量的最大取值范围。如果j域取4比特，则$C_{max}=16nM$,（n为一个结点所能有的最大孩子数目），C的取值范围为$0\sim C_{max}^{-1}$.

第二步，将C随机化

$C'=(c\times a) \mod p$

如果p是大于C_{max}的一个素数，a是在1和p之间的一个整数，并且对于所有能除尽p−1的素数q，$a^{(p-1)/a}\neq 0 \mod p$，那么c′将是随机的，步骤也是可逆的[4]。a的合适值可通过试探得到，首先选取一个初值，然后对所有形式为(p−1)／a的x，测试是否有某个q使得$a^x \mod p$不等于1，如果需要，增加这个初值直到条件满足。实际上，2p／3是一个好的起点，一般不需要增加几次就可使条件满足。

由于c′现在是随机的，杂凑函数可选c′ mod M，对应的唯一性数就是[C′／M]。如果M是2的乘幂，则这些值就是可用的比特域。

重构<m,i,j>时，这些步骤的每一步都能被反转。给定初始杂凑地址i和唯一性数b，则

$$C'=b\times M+i$$
$$C=(a^{-1}\times C') \mod p \quad (\text{这里 } a^{-1}=a^{p-2} \mod p)$$
$$m=c \ \mathrm{diV}\ 16M$$
$$i=c \mod M$$
$$j=(c \mod 16M)\ \mathrm{diV}\ M$$

在具体实现时，初始杂凑地址i是在从树的一个结点访问另一个与它直接相连的结点时临时计算的，并不需要存贮在杂凑表中。在杂凑表中每个结点的m域需要$\log_2 n$比特，j域一般选4比特，另外，为了“未用”和“改变”标志需要2比特，总共只需占用$6+\log_2 n$比特。如果是二叉树，n=2，唯一性数给定为5个比特，则压缩杂凑法使每个结点总共只需占用7个比特的存贮量，并且对于任意大小的树都是一样的。

2 分析

2.1 用该方法存贮树时，每个结点在Hash表中仅占$6+\log_2 n$个比特，注意，这是n个指针一起的总长度，而不是单个指针的长度，并且对任意大小的树都是一样的。因此用这种方法能大大压缩树所需要的存贮空间。

2.2 由于一个结点的结点关键字在它的双亲结点在表中的存放位置被确定以前不能被构造，因此，必须从根往下形成树的结构。由于同样的原因，一棵子树一旦被存贮便不能移动；除非树被重新建立，也不能被嫁接到另一个结点上去。

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

2.3　冲突组 j 的大小实际上是没有限制的．对于大多数应用来说，4 比特便可满足需要。当然也存在着出错的可能性，出错的概率与杂凑表的存贮密度 λ 有关，$\lambda = N/M$ 称为负载因子，式中 M 是 Hash 表的有效存贮单元数，N 是实际存贮的项数．γ 个关键字杂凑到同一个初始杂凑地址的概率是

$$e^{-\lambda}\frac{\lambda^{\gamma}}{\gamma!}$$

λ 总是比 1 小，我们假定它至多是 0.8，那么任何位置有多于 K 个关键字杂凑到它的概率 e 被限制在

$$e \leqslant Me^{-0.8}\sum_{\gamma=K+1}^{\infty}\frac{0.8^{\gamma}}{\gamma!}$$

令 K＝16(j 域 4 比特),$e < 7\times 10^{-17}M$,在大多数应用中这是可以接受的．如果令 K＝32,则产生一个错误的概率被缩小到 $e < 3\times 10^{-41}M$．虽然这种方法的成功运行依赖于概率的假设,但它可使出错的可能性变得如此之小,以致我们可以完全把这个问题忽略．

3　结论

用 Hash 表实现树的压缩存贮，可使树在内存中具有十分紧凑的存贮形式，其每个结点的指针仅占用 $6+\log_2 n$ 个比特的存贮空间，而且与树的大小无关．但是，它不允许对树进行剪接，也不能回收无用的结点，插入和搜索操作也比用更简单的方法来表示树要慢一些．因此，这种表示法适合于那些需要在有限的内存空间中构造单调增长的大型树的应用领域，并且在这些应用领域中，一个偶然的故障是可以接受的．

参 考 文 献

1 J.G.Cleary, Compact hash tables using bidirectional linear probing, IEEE TRANS. COMPUTERS, C-33, (9),1984

2 D.E.Knuth, The Art of Computer Programming Vol.3: Sorting and Searching,1973

3 JOHN J.Darragh, Bonsai: A compact Reprosentation of Trees, Software-Paractice and Experience, Vol 23(3),1993.

Trees can be stored in Very Compact Form Using Hash Tables

Ren Junyuan

(Zhengzhou Institute of Technology)

Abstract: This paper shows how trees can be stored in a very compact form using hash tables. Using the method, pointers in any tree can be represented with in $6+\log_2 n$ bits per node where n is the maximun number of children a node can have. The total storage requirment can be reduced significantly when storing large trees.

Keywords: Compact hashing, Hash trees, Tree structures

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net