基于改进跳跃表的数据检索系统应用①

Application of Data Retrieval System Based on Improvement of Skip List

陈庆全　黄文明　崔亚楠　（桂林电子科技大学 计算机与控制学院 广西 桂林 541004）

摘　要：具有双向指针的跳跃表结构是对简单跳跃表的改进，其优点是在数据检索过程中能够避免指针回退和减少某些结点值的比较次数，使得数据检索效率进一步提高。本文将使用 C++语言对该算法进行描述，并且由该算法实现的检索功能模块已经成功应用在柳州城市节水系统中，从节水系统的使用效果验证了该算法的可行性。

关键词：线性表 跳跃表 双向跳跃表 数据检索 算法分析

对于具有大量数据存储的系统来说，数据检索一直是一个难题。在大多数应用程序，一般都是使用简单的线性表（Linear list）来处理。线性表作为一种最常见的数据结构，具有结构简单，操作方便等优点。但是在具有大量数据的应用系统中，采用线性结构经行数据检索，检索的效率往往不尽如人意。如何能有效提高海量数据的检索，一直是业界人士研究的重点内容。

基于对海量数据内容检索性能提高的要求，本文将介绍一种新的数据结构算法，该算法是在跳跃表（Skip list）结构的基础上改进得到的，通过这些改进能有效弥补跳跃表结构在检索中存在的不足。从而进一步提高数据检索的速度与稳定性。

1　线性链表概念

线性结构由于结构简单而被广泛应用。线性结构的基本特征为：除起始结点没有直接前驱外，其他结点有且只有一个直接前驱；除终端结点没有直接后继外，其他结点有且只有一个直接后继。线性表是目前最常用的一种线性结构，线性表中包括了顺序表和线性链表。

线性链表使用指针表示结点间的逻辑关系。故存储结点包含两个部分：其中一个为数据域 data，它存储数据元素；另外一个是指针域 next，它存放一个指针，指向后继结点。

线性链表的数据检索过程是从头结点（head 指针所指向的结点，头结点指针域指向第一个结点）开始找到第一个结点，再由它的指针域往下找，通过比较结点的数据域 data 从而找到所需结点。链表的结点是按结点值顺序排列的，因此检索过程中，链表需要沿着结点一个一个移动，平均需要访问 Θ(n) 个结点[1]。

2　跳跃表策略

跳跃表是对简单的链表的改进，它是用于实现字典 ADT 的概率数据结构。设计跳跃表是为了解决基于数组的线性表和链表的基本问题：一次检索或一次更新需要的时间是线性的。因为跳跃表的一些决定是随机做出的，所以它是一种概率数据结构（probabilistic data structure）[1]。

跳跃表的优点就是在检索过程能跳过其中一些结点，减少关键码的比较次数，从而有效的把数据检索工作减少。在具有二级指针的跳跃表中，进行检索时，先沿着 1 级指针走，直到找到一个比检索关键码大的结点值。然后回到 0 级指针。对于一个有 n 个结点的跳跃表，第一个结点和中间结点最终会有 logn 个指针。进行检索时，从最高级别指针开始，走得尽可能远，一次跳过多条记录。然后，根据需要使步幅越来越短。

① 基金项目：2007 年广西区研究生教育创新项目（2007105950812m18）

通过这种安排,在最差情况下的访问数是 Θ(logn)。

在跳跃表的检索过程中,无论是检索成功或失败,都会跳过一些结点值的比较过程。如果在具有多级指针的跳跃表中,检索过程将会跳过更多的结点值的比较。如下图1为一个具有三级指针的跳跃表。

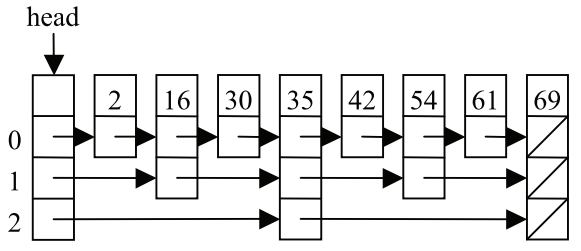

图1　具有三级指针的跳跃表

具有三级指针的跳跃表是对具有二级级指针的跳跃表的扩展,在检索过程,从头结点开始,首先沿着2级指针向前走,如果遇到的下一个结点的值比需要检索的关键码值大,则退回一个结点,并且指针级别数由原来的2级减为到1级,1级指针向前检索,依次类推,遇到结点值比需要检索的关键码值大的时候,则退回到一个结点,指针级别减为0级指针向前检索,直到检索成功或检索失败。

例如在图1中检索关键码为69的过程中,指针的移动跳过了结点值为2、16、30、42、54和61的结点。检索效率将进一步提高。

在关键码的检索中,我们发现了在检索失败的过程中必然遇到的一个问题,就是指针回退问题。如果在一个具有n级指针的跳跃表中,在关键码的检索过程中,最差的情况需要n-1次指针回退,并且某些具有n级指针的结点值会被重复比较n次。如在图1的跳跃表中检索关键码62,2级指针从头结点移动到达结点值为69的结点,比较69大于62,2级指针回退到结点值为35的结点,指针级别数减1。然后沿着值为35的结点的1级指针到达结点值为69的结点,比较69大于62,1级指针退回到结点值为54的结点,指针级别数减1。最后沿着0级指针到达结点值为69的结点,比较69大于62,从而可以判断检索失败。在此检索过程中,指针必须做出两次回退,而结点值为69的结点与关键码进行了三次比较。

由于出现了指针回退和某些结点值出现多次重复比较,这必然给检索过程增加额外的负担,因此我们可以在跳跃表的基础上改进结点的指针结构,从而实现一种具有双向指针结构的跳跃表。

3　改进的跳跃表算法

考虑到在跳跃表中当指针到达的结点值比需要检索的关键码大的情况,指针必须退回到当前级别指针的上一个结点,并从原来的n级指针减少到n-1级指针再继续向前移动进行关键码比较。在单向的跳跃表中,每个结点只有一个指针域数组,该指针域数组分别指向了当前结点的各个级别的后继结点。在单向跳跃表中,结点指针存储在一个名为forward的数组中,其中forward[0]存储一个0级指针,forward[1]存储一个1级指针,依此类推。为了实现当前结点的指针既能指向后继结点,也能指向前驱结点,现在每个结点里增加一个指针数组backward,其中位置backward[0]存储一个0级指针,backward[1]存储一个1级指针,依次类推。该指针数组里的每一个元素都是指向当前结点各个指针级别的前驱结点。如下图2所示。

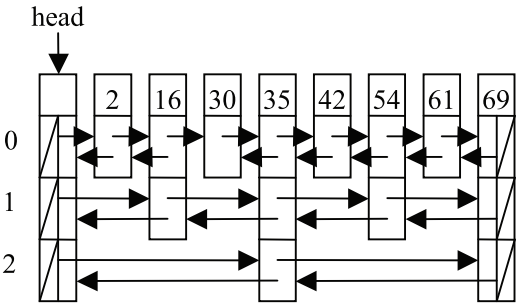

图2　具有三级指针的双向跳跃表

在具有双向指针的三级跳跃表中,检索关键码为69的过程中,首先是从头结点的forward[2]指针向前移动,一直到达值为69的结点,比较结点值与关键码的大小,判断检索成功。该检索过程跳过了值为2、16、30、42、54、61的结点比较过程。因此它跟单向跳跃表具有同样的检索效率。

例如检索关键码为54的过程,首先从头结点开始沿着forward[2]指针向前移动,一直到达结点值为69的结点,比较69大于54,此时不必做指针回退,只需将指针级别数减1。比较backward[1]->data(值为54)与关键码大小,54等于54,从而可以判断判断检索成功。此检索过程不必要从结点值为69的forward[2]指针退回到结点值为35的forward[2]指针,并且跳过了值为2、16、30、42的结点的比较过程。

检索关键码为62的过程,从头结点开始沿着for-

ward[2]指针向前移动，一直到达结点值为 69 的结点，比较 69 大于 62，指针级别数减 1。比较 backward[1]->data(值为 54)与关键码大小，54 小于 62，指针级别数再减 1。比较 backward[0]->data(值为 61)与关键码大小，61 小于 62。由于当前指针级别已经是 0 级，故可以判断关键码不在跳跃表中，检索失败。此检索失败的过程中，除了避免了 forward 指针的回退，还有效避免了结点值为 69 的结点与关键码的多次比较，因此可以大大的提高检索的效率。

在单向跳跃表中，跳跃表类定义一个 level 数据成员，它存储当前跳跃表中结点的最大级数。在具有双向指针的跳跃表中，同样在类中定义一个数据成员 level，它表示双向跳跃表中结点的最大级数。假定指针 head 指向双向跳跃表的头结点，那么查找关键码的函数 find(使用 C++语言实现)如下图 3 所示。

```
template <class Key, class Elem, class KEComp, class EEComp >
bool DuSkipList < Key, Elem, KEComp, EEComp >::
find(const Key&K, Elem&e) const{
    DuSkipNode <Elem > *x = head;
    int i = level;
    while(x->forward[i] != NULL){
        if(KEComp::gt(K, x->value)) x = x->forward[i]
        else if(KEComp::eq(x->value,K)){
            e = x->value;
            return true;
            }
        else break;
        }//while;
    if(i>0){
        i--;
        for(;i>=0;i--)
            while(x->backward[i] != NULL) && KEComp::ge(x->backward[i]->value,K) x = x->backward[i];
        if ((x != NULL) &&KEComp::eq(x->value,K)){
            e = x->value;
            return true;
            }
        }//if(i>0);
    return false;
    }
```

图 3　具有双向指针的跳跃表的查找函数 find()

4　应用实例和性能分析

针对本文前面所介绍的具有双向指针结构的跳跃表算法，将其应用于柳州城市节水系统的数据检索中，能有效地提高数据检索的速度。在柳州城市节水系统中，存储了大量的广西自治区柳州市节约用水计划用水办公室的用户数据，其中包括了柳州市自来水公司的用水客户的各个月的用水情况，用户的实际用水和节约用水情况。在这些用户数据中，按每个客户的水表户号(从 1 开始，最大为一个 7 位的整数，目前最大的水表户号为:210486)顺序排列。在柳州市节水系统中，处理用户数据的时候，需要检索大量的用户信息或各个用户的节约用水或超计划用水情况等。如果能有效的提高检索的速度，那么将大大的提高整个城市节水系统的运行性能。图 4 和图 5 为使用五级指针的双向跳跃表结构的水表户号检索结果。图 4 为水表户号“8000”检索成功所需的时间(单位/秒)，而图 5 为水表户号“200000”检索失败所需时间(单位/秒)。

图 4　水表户号检索成功耗时

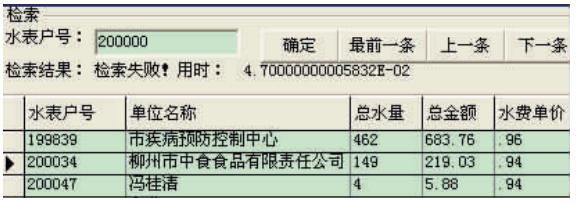

图 5　水表户号检索失败耗时

为了验证双向跳跃结构比普通的线性结构具有更好的检索效率，在节水系统中通过检索不同的水表户号所需的时间得出如下表 1 所示的数据。其中跳跃表结构和双向跳跃表结构均采用五级指针结构。

表 1　使用不同结构算法的检索耗时(单位/ms)

检索内容	线性表耗时	跳跃表耗时	五级双向跳跃表耗时	检索成功与否
100	187	131	131	是
200	234	181	116	否
400	547	181	116	是
800	969	174	123	是
1000	1203	525	193	否
2000	2023	263	116	是
4000	4969	409	116	是
8000	10656	47	16	是
10000	13031	141	54	否
20000	26969	47	47	是
40000	55422	1203	189	否
80000	96797	178	131	否
100000	108735	1109	131	否
110000	115531	148	178	是
120000	120453	1114	126	否
140000	135734	1419	135	否
180000	178000	219	47	是
200000	202672	572	47	否

从上表 1 的数据显示可知道,采用线性结构进行检索,检索所需的时间会随着水表户号的增加而线性递增。如果检索的客户号进一步增大,检索成功或失败所需的时间将达数十分钟。而采用跳跃表的结构来进行数据检索,可以有效的提高了检索速度。从表 1 中可以看出,采用五级指针的跳跃表结构进行检索耗时最大的不超过 2 秒。因为在检索过程结点数据的移动中,一开始就可以跳过大量的比关键字小的数据。而线性结构检索中,大部时间就是使用在这些被跳过的数据的移动过程中。因此跳跃表能有效的避免了这些数据的逐个移动,从而达到了提高检索速度的目的。在双向跳跃表的检索中,检索成功或失败所需的时间又比简单跳跃表少,最大的不超过 0.2 秒。因为双向跳跃表能避免多次相同数据的比较和指针来回移动。在表 1 中也存在双向跳跃表检索时间比简单跳跃表检索时间长结果,但是这只是少数的。通过以上的数据可以看出,双向跳跃表的整体性能要比简单跳跃表的高。从而可以验证本文提出的算法在顺序存储的数据检索中更有效。

5　结束语

本文所提出的算法是根据跳跃表的特点改进而来的,同时它又可以有效避免在单向跳跃表检索过程中的指针多次回退,在实际的应用实例结果可以证明,双向跳跃表在数据检索的效率上明显优于单向的跳跃表。通过将该算法应用于现有的软件系统从而能有效改进系统的运行性能,因此具有很好的实际应用价值。在信息时代的今天,许多软件系统均要处理大量的数据,因此,一种高效的检索算法必然会提高整个软件的性能。并且对于目前许多的数据库管理工具存储和检索机制有一定的参考价值。

参考文献

1 Clifford A S. 张铭,刘晓丹译. 数据结构与算法分析(C + +版). 第二版,北京:电子工业出版社,2001:257 – 261.

2 严蔚敏,吴伟民. 数据结构(C 语言版). 北京:清华大学出版社,2001:18 – 39.

3 Herlihy M, Lev Y, Luchanqco V, Shavit N. A simple optimistic skiplist algorithm // Structural Information and Communication Complexity – 14th International Colloquium, SIROCCO 2007, Proceedings. Berlin: Springer – Verlag, 2007:124 – 138.

4 Lamoureux MG, Nickerson BG. A deterministic skip list for k – dimensional range search. Acta Informatica, 2005, 41(4):221 – 255.

5 Patricio VP, Munro JI, Thomas P. The binomial transform and the analysis of skip lists. Theoretical Computer Science, 2006, 352(1):136 – 158.

6 Wang D, Liu JC. Peer – to – peer asynchronous video streaming using skip list // 2006 IEEE International Conference on Multimedia and Expo, ICME 2006 – Proceedings. Piscataway NJ: Institute of Electrical and Electronics Engineers Computer Society, 2006:1397 – 1400.

7 Sundell H, Tsigas P. Fast and lock – free concurrent priority queues for multi – thread systems. Journal of Parallel and Distributed Computing, 2005, 65(5): 609 – 627.

8 马越,张大勇,金一丞. 一种基于跳表的 DDM 相交区域快速查询算法. 计算机仿真,2005,22(7):46 – 50.

9 Shavit N, Lotan I. Skiplist – based concurrent priority queues. // Proceedings 14th International Parallel and Distributed Processing Symposium, IPDPS 2000. Los Alamitos CA: Institute of Electrical and Electronics Engineers Computer Society, 2000:263 – 268.