K-th Number Query 问题的改进算法研究

陈 鑫
CHEN Xin

南京大学 计算机科学与技术系,南京 210093
Department of Computer Science and Technology,Nanjing University,Nanjing 210093,China
E-mail:jsntcx@gmail.com

CHEN Xin.Improvement of *K*-th number query problem algorithm.Computer Engineering and Applications,2009,45(21):150-152.

Abstract: The *K*-th number query is a fundamental problem in computer algorithm,which is a subroutine of numerous problems. Researchers have done a lot of further work including the linear time algorithm for single *K*-th number query.The time com plexity O(lb *n*) for each query solution has already been found for the multi-queries *K*-th number query problem,with the help of balance search tree structure.But the BST-based algorithm is not very easy to implement as well as a big constant factor hidden in the Big-O representation.This paper introduces an algorithm based on Bit Indexed Tree to tackle *K*-th number query with easy implementation and small constant factor.Finally,the experiment shows that the new algorithm is remarkably faster than previous algorithms with nearly optimal memory usage.
Key words: *K*-th number query;bit indexed tree;random-select

摘 要:*K*-th number query 是计算机算法中的一个基础问题,被广泛作为很多算法实现的重要步骤。对该问题进行了深入研究,并找到了单询问渐近时间复杂度最优的算法。目前一般对于多询问的 *K*-th number query 问题使用平衡二叉树解决,询问的时间复杂度为 O(lb *n*)。但该算法实现比较复杂,并且常系数较大,提出了基于 Bit Indexed Tree 数据结构的算法解决,在同等时间复杂度的前提下,实现简单,隐含的常系数很小。最后进行了实验测试,分析显示该新算法不论在时间上还是空间上都优于现有的算法。
关键词:第 *K* 大数查询;位索引树;随机化选择
DOI:10.3778/j.issn.1002-8331.2009.21.044 **文章编号:**1002-8331(2009)21-0150-03 **文献标识码:**A **中图分类号:**TP311.12

1 引言

K-th number query 问题是一个计算机科学中很重要的一个问题,由于其基础性,广泛作为很多算法实现的重要步骤之一,包括实时调度、文本压缩等领域[1],并且在 C++中作为 STL 扩展函数被支持[2]。具体定义如下:

定义:$L(v)=|A\cap(-\infty,v)|$为集合 A 中小于 v 的元素个数。类似的,$LE(v)=|A\cap(-\infty,v)|$为集合 A 中不大于 v 的元素个数。

输入:n 个元素的集合 $A[1..n]$,整数 $k(1\leqslant k\leqslant n)$。

输出:x,满足 $x\in A$,$L(x)<k$ 且 $LE(x)\geqslant k$。

对于一个基于比较的排序算法,由信息论分析可知,其比较次数的下界为 $\Omega(n\text{lb}\,n)$,对于 *K*-th number query 问题,由于不需要集合 A 元素的全序,其下界要小于排序。Donald E. Knuth 给出了一个该问题所需要比较次数的非紧下界[3]:

$$n-k+\sum_{n+1-k<j\leqslant n}\lceil \lg j\rceil$$

对于多个询问的情况下,Donald E.Knuth 设计出一个二进制数的冗余表示系统(redundant representation system),使得对于一个二进制计数器每次操作都从原有的 O(1)的均摊时间复杂度降到了 O(1)最坏复杂度。将这样一个二进制数的设计规约到数据结构的设计,得到了两种不同的结构,对一棵平衡二叉树的维护,或者维护一组 2~3 树,都可以达到实现 *K*-th number query O(lb k)的时间复杂度[4],对于询问值 k 远小于 n 的情况下有所改进。但由于维护复杂,隐含的常系数过大。

2 单询问 *K*-th number query 算法

对于单询问的 *K*-th number query 问题,一般用排序直接实现。将集合 A 的元素排序以后,处于 $A[k]$位置的数就是该集合第 i 小的数。这个直观的算法如果用 Quicksort 实现期望时间复杂度为 O($n$lb n),用 Heapsort 实现最坏情况时间复杂度为 O($n$lb n)。

上文分析已经指出,由于 *K*-th number query 不需得到集合全序,所以排序实际上多做了许多不必要的比较,使得时间复杂度过高。另一个基于 Quicksort 改进的算法,Random-Select,很好地解决了冗余计算的问题,使得该问题能在 O(n)的期望复杂度下解决[5]。算法如下:

Random-Select(P,R);
W=Random Number In $A[P]\cdots A[R]$;
Divide $A[P]\cdots A[R]$ into two parts

作者简介:陈鑫(1987-),男,主要研究方向为计算机理论,人工智能,计算机网络等。
收稿日期:2009-05-04 **修回日期:**2009-06-09

```
(A[P]···A[K]<=W,A[K+1]···A[R]>=W)
If k<=K Random-Select(P,K)
Else Random-Select(K+1,R);
```

由于加入了随机化，更难构造特定的数据使得该算法退化,并且由于该算法具有较小的常系数,所以被应用相当广泛。

3 多询问 *K*-th number query 算法

多询问 *K*-th number query 是在上述单询问 *K*-th number query 上的一个简单扩展。对于一个数集 *A*,需要顺序输出的是输入给出的一系列的 k_i 所对应的 $x_i(1\leqslant i\leqslant m)$,并且在询问中可能对集合 *A* 有所修改。

多询问的 *K*-th number query 可以直接使用 *m* 个单询问 Random-Select 算法解决,得到一个时间复杂度为 O(*mn*)的算法。由于没有利用同一个数集的条件,使得该算法做了许多冗余的计算,下述算法对此进行了优化。

3.1 平衡二叉树算法

平衡二叉树是一个常用的数据结构，在解决动态的插入、删除、询问、找最邻近数等问题方面有着广泛的应用。由于平衡二叉树的高度保证为 O(lb *n*)级别,所以对于任意的上述操作的时间复杂度都可以达到 O(lb *n*)。而对于多询问的 *K*-th number query 问题只需要在普通平衡二叉树上额外保存一些信息并动态维护，即可将每个询问和修改的时间复杂度降低为 O(lb *n*)。

在平衡二叉树的每一个节点上增加两个整数 *Count* 和 *Sum*,其中 *Count* 保存同当前节点值相同的值的个数,*Sum* 保存以该节点为根的子树的 *Count* 值总和。则 *Sum* 的计算方法如下:

```
Update-Sum(Node);
Note.Sum=Note.Count;
If LeftChild<>NULL Sum=Sum+LeftChild.Sum;
If RightChild<>NULL Sum=Sum+RightChild.Sum;
```

平衡树中插入元素进行相应的修改,算法如下:

```
Insert(Position,Node);
If Position=NULL Add(Note),Note.Count=1
Else
  If Node.Key<Position.Node.Key
    Insert(LChild,Node)
    If Not Balance Then Rotate;
  Else If Node.Key>Position.Node.Key
  Insert(RChild,Node);
  If Not Balance Then Rotate;
Else (Node.Key=Position.Node.Key)
  Position.Node.Count+1;
  Update-Sum(Position);
```

其中平衡二叉树维护平衡的旋转工作与删除过程做类似修改。

有了上述的两个额外参数以后,对于第 *k* 大数的求解就可以在 O(lb *n*)的时间复杂度完成。每次都可以判断第 *k* 大的数是属于该节点还是左子树或者是右子树，并递归地转化判断，具体算法如下:

```
Get-KthElement(k,Node);
If k<=LeftChild.Sum return Get-KthElement(k,LeftChild);
If k<=LeftChild.Sum+Node.Count return Node.Key;
return Get-KthElement(k-LeftChild.Sum-Node.Count);
```

3.2 Bit Indexed Tree 介绍

平衡二叉树的维护算法虽然理论复杂度较低,但使用了额外 4*n* 的空间进行指针和区间信息的记录,且实现较复杂。下文介绍一种更为紧凑的数据组织方法:Bit Indexed Tree。

Bit Indexed Tree 数据结构由 Peter m.fenwick 于 1994 年提出[6]。该作者以及后来的很多研究者如文献[7]等都将该数据结构应用于文本压缩的频度统计,将其推广,使其支持 *K*-th number query 查询。

Bit Indexed Tree 的基本思想为任意一个正整数都可以表示为二进制,继而可以拆分成多个连续区间和的形式。例如一个二进制数:101110100,依次将其末尾的 1 改变成 0 可以得到数列:

101110100
101110000
101100000
101000000
100000000
000000000

该数列严格单调递减,通过这种方式 101110100 被自然地分成了多个区间,而求和工作就可以将拆分出的几个区间的部分和相加得到。即:设二进制数 *x* 的最低位的 1 以及后面的 0 构成数为 *LowBit*(*x*),则对于任意整数 *v*,*v* 可以将[*v*-*LowBit*(*x*)+1,*v*]这段区间和表示出来。如当 *v*=12 的时候从 1 到 12 的和就可以用 *C*[12]+*C*[8]完成计算。所建立的部分和图示如下:

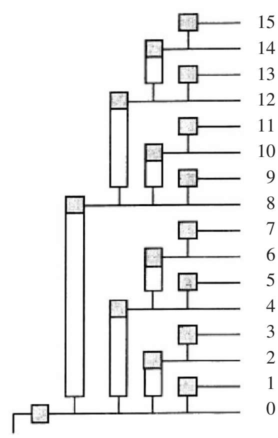

图 1 Bit Indexed Tree **部分和**

对元素的修改可以用类似的方法实现,求和的时候需要递减将末尾的 1 改写成 0,而修改的时候需要将末尾的 1 不断进位,直到达到上界。统计区间[1,*X*]和的算法实现如下:

```
Get-Sum(X);
Sum=0;
While X>0 Do
    Sum=Sum+C[X];
    X=X-LowBit(X);
Return Sum;
```

由于不超过 *n* 的二进制表示至多有 lb *n*+1 个 1,所以上述过程的时间复杂度为 O(lb *n*)。修改某元素值的算法类似,如下所示:

```
Modify(X,Delta);
While X<=UppderBound Do
```

$C[X]=C[X]+\mathrm{Delta}$;

$X=X+\mathrm{LowBit}(X)$;

LowBit(*X*)函数的计算可以通过位运算在 O(1)的时间复杂度内实现,其写成 C 语言位运算的形式为 *LowBit*(*X*)=*X*&(~(*X*-1))。证明很容易,可以通过归纳 *X* 最后一位 1 的位置得到,*X*-1 未退位的部分去反以后与 *X* 进行逻辑与运算为 0,将最低位的 1 分离出。

3.3 优化算法

K-th number query 在 Bit Indexed Tree 结构上可以被高效解决。首先假设数集 *A* 的所有数字都属于区间[1,*n*],如果数集 *A* 本身并不满足这个条件,这一步的工作可以用离散化来实现。即将 *A* 中的所有数排序,排序后的序列映射为[1,*n*]区间。

在 Bit Indexed Tree 中取得第 *K* 大的元素类似于一个二分查找的过程,询问从二进制形式的最高位到最低位,每次确定一个比特的取值。以一个 *Base* 变量记录目前选择的位的和,若检测到当前位选择以后小于 *K*,那么该位一定为 1,反之不然。与平衡二叉树类似,在每次确定某为 1 了以后(相当于平衡二叉树查找其右子树),调整当前的 *K* 值,并进行下一步查找,直到所有位被确定。算法的代码如下所示:

```
Select-Kth(K);
I:=UpperBound;Base=0;
While I>0 Do
      If C[I+Base]<K Then
            K=K-C[I+Base];
            Base=Base+I;
      I=I/2;
Return Base+1;
```

同样由于对于任意数集 *A* 的元素的二进制表示为 lb *n*+1 位,所以扫描的总次数为 O(lb *n*),所以关于该 *K*-th number query 问题的任意操作的时间复杂度都可以在 O(lb *n*)时间复杂度内完成,与平衡二叉树的算法时间复杂度相同。

4 测试及结论

分别采用三个不同的算法(Random-Select、BST、Bit Indexed Tree)实现了 *K*-th number query 问题的解答。其中 $N=|A|$,*M* 为询问和修改的总和。数据全部随机生成。其中 *N/A* 表示运行时间过长,被强行终止。RND-SEL 为 Random-Select 算法,BIT 为 Bit Indexed Tree 算法。其中时间以毫秒为单位,测试结果如图 2。

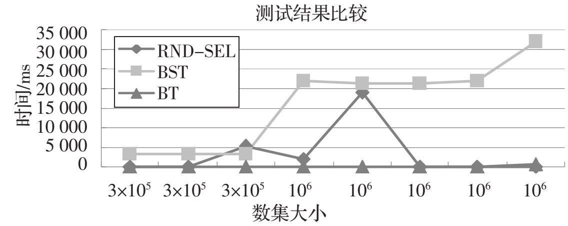

图 2 算法运行时间比较

测试显示,RND-SEL 在询问很少的情况下(如表中 *M*=1)查询效率要优于 BIT 和 BST 数据结构。但一旦询问规模达到一定的程度以后,RND-SEL 算法就难以在短时间内得到结果了。可以看出 RND-SEL 算法的耗时与 *M* 是成正比关系的,符合其理论分析。对于 BST 和 BIT 的数据结构,其渐近时间复杂度虽然相同,但因为平衡二叉树所涉及的更新操作复杂,每次

表 1 测试结果 ms

N	M	RND-SEL	BST	BIT
3×10^5	1	16	3 432	31
3×10^5	10	109	3 375	31
3×10^5	10^3	5 343	3 421	32
10^6	10^2	1 922	21 922	94
10^6	10^3	18 860	21 187	94
10^6	10^4	N/A	21 266	94
10^6	10^5	N/A	22 015	157
10^6	10^6	N/A	32 063	671

需要递归更新其节点信息和旋转以保持平衡,比 BIT 结构复杂很多,大 O 表示隐含 BST 的常系数过大,所以实际测试的效率差异明显。

关于内存使用情况的比较:

表 2 内存使用比较表

算法	额外内存使用
RND-SEL	O(lb*n*)
BST	4*n*
BIT	O(1)

RND-SEL 与 BIT 算法都可以利用单个数组完成更新而求值的操作,只有 O(1)额外内存开销。而 BST 算法由于维护平衡二叉树的左右儿子和平衡因子等信息,需要大量的冗余储存。

分析了对 Bit Indexed Tree 数据结构进行扩展,使其可以高效地完成多询问 *K*-th number query,并与之前的数据结构进行了比较。在新结构下算法在保证渐近时间复杂度的前提下具有较小的常系数,并且冗余保存的数据为 O(1),具有较高实用价值。

参考文献:

[1] Musser D R.Introspective sorting and *K*-th number query algorithms[J].Software:Practice and Experience,1997,27(8):983-993.

[2] ISO/IEC 14882:Specifies requirements for implementations of the C++ programming language and standard library[S].2003.

[3] Knuth D.The art of computer programming[M].3rd ed.NY:Addison-Wesley,1997:207-219.

[4] Clancy M J,Knuth D E.A programming and problem-solving seminar,Technical Report CS-TR-77-606[R].Stanford University,1977.

[5] Cormen T H,Leiserson C E,Rivest R L,et al.Introduction to algorithms[M].America:MIT Press,2002.

[6] Fenwick P M.A new data structure for cumulative frequency tables[J].Software—Practice and Experience,1994,24(3):327-336.

[7] Vines P,Zobel J.Compression techniques for chinese text[J].Software-Practice & Experience,1995.

[8] Andersson A.Balanced search trees made simple[C]//Workshop on Algorithms and Data Structures,1993:60-71.

[9] 曼博.算法引论[M].北京:电子工业出版社,2005.

[10] Fenwick P M.A new data structure for cumulative probability tables:An improved frequency-to-symbol algorithm[J].Software-Practice & Experience,1995.

[11] 高静,杨炳儒,徐章艳,等.一种改进的基于正区域的决策树算法[J].计算机科学,2008(5):138-142.

[12] 张师超,张继连,陈峰,等.负增量式关联规则更新算法[J].计算机科学,2005(9).

[13] 孙沛涛,孙俊清.最大频繁项目集的增量式更新算法[J].计算机工程与设计,2005(12).