文章编号: 1000-2022(2005)06-0862-05

基于位的二分查找算法

成亚萍, 傅德胜

(南京信息工程大学 计算机科学与技术系, 江苏 南京 210044)

摘要: 根据数据在内存中的存储形式, 提出了一种直接利用二进制信息确定待查找数据查找范围的新的二分查找算法。试验结果表明, 该算法是可行的, 且可用于一些特殊场合。

关键词: 算法; 二分查找; 位

中图分类号: TP391 **文献标识码:** A

查找算法是计算机应用中运用较为频繁的一类算法, 也是智能系统经常用到的操作。实现查找的方法有多种, 如顺序查找、二分查找、分块查找等。著名计算机科学家 Kunth 在《计算机程序设计技巧 第三卷 排序和查找》中——分析了已有的比较式查找方法, 认为二分查找的平均工作量最小^[1]。文献[2]提出的主要是基于十进制数的解决方案, 文献[3]提出的主要是基于硬件的解决方案。本文提出一种基于二进制位的二分查找软件方法, 它无需比较数据大小, 适用于所有整数。

1 算法描述

1.1 基于位的二分查找算法的基本思想

数据在内存中是以二进制形式存放的。对有符号数而言, 最高位为 0 表示正数, 最高位为 1 表示负数。由于二分查找的基础是在有序的数据中查找, 所以不妨设数据是从小到大排列的。首先通过判断符号位将数组 (本文以 C 语言描述) 分为前后两段: 负数段和非负数段。判断待查找的数据 z 的符号位, 确定所属数据段。在相应的数据段中对 z 采用基于位的二分法查找。设这段数据的起始位置、终止位置用 i, j 指定, 则中间位置为 $(i+j)/2$ 。由于数据在内存中是补码存放的, 所以若 z 为负数, 先对 z 的各位 (除符号位) 取反, 对数组中的负数查找前进行同样的操作^[4], 采用由高位到低位的比较顺序。若遇到 z 的某一位 p 与数据段中参加比较的数的第 p 位不同时, 即可停止与该数的比较。根据 z 的正负性及终止位的 0、1 性确定 i 与 j 的变化, 进而确定新的 k ; 否则, 继续比较下一位。如此下去, 直到第 0 位也相同, 则得出找到的结论, 或因 $i > j$ 得出该数不存在的结论。

收稿日期: 2004-06-30 改回日期: 2005-01-21

基金项目: 南京信息工程大学科研基金资助项目 (Y505)

作者简介: 成亚萍 (1969-), 女, 山西平遥人, 讲师, 硕士生, 研究方向: 算法设计与分析、图像处理。

E-mail: xiaobaoc@163.com

1.2 基于位的二分查找算法

设整数以 L 位二进制存储, 并有整型数组 $a[N]$, 其中数据以从小到大的顺序排列, 并对负数作如上所述的改变, 此时负数段的顺序变为由大到小。基于位的二分查找算法过程如下:

(1) 判断符号位, 统计负数个数 n_1 。

(2) 初始化 p (用来指定被比较的二进制位的位置, 初始值为 $L-2$)。判断待查找数 z 的符号位, 若为 1 则设 $i=0, j=n_1-1$; 否则, 设 $i=n_1, j=N-1$ 。转 (3)。

(3) 若 $i \neq j, k=(i+j)/2$ 转 (4); 否则转 (6)。

(4) 比较 z 与 $a[k]$ 的第 p 位。若相同, $p=p-1$ 直到 p 为 -1 找到, 查找结束; 否则, 转 (5)。

(5) 若 z 的符号位为 1 且第 p 位为 1 则 $i=k+1, p=L-2$

否则若 z 的符号位为 1 且第 p 位为 0 则 $j=k-1, p=L-2$

否则若 z 的符号位为 0 且第 p 位为 1 则 $i=k+1, p=L-2$

否则 $j=k-1, p=L-2$

转 (3)。

(6) 没有找到, 查找结束。

2 算法的正确性与复杂性分析

2.1 算法的正确性分析

该算法的主要问题是当 z 与 $a[k]$ 的第 p 位出现不同时, 如何确定 i, j 的变化。对于负数先进行本文中定义的操作, 再确定数字大小。因此, 可根据 z 的正负性及 b 的 0-1 性将 i, j 的变化分为 4 种情形:

z 的符号位为 1 z 的第 p 位为 1 表示 z 为负数, 且 $z < a[k]$;

z 的符号位为 1 z 的第 p 位为 0 表示 z 为负数, 且 $z > a[k]$;

z 的符号位为 0 z 的第 p 位为 1 表示 z 为正数, 且 $z > a[k]$;

z 的符号位为 0 z 的第 p 位为 0 表示 z 为正数, 且 $z < a[k]$;

如上所述, 不用比较 z 与 $a[k]$ 的大小, 通过检测相应的二进制位即可得出 z 比 $a[k]$ 大或小的结论, 然后按照二分查找的算法确定 i, j 的变化。可见该算法是正确的。

2.2 算法的复杂性分析

统计负数个数 n_1 时, 最坏的情况为 $n_1=N$, 最好的情况为 $n_1=0$ 其次是比较的过程, 最好的情况是一次找到, 最坏的情况为 $\log_2^N + 1$ 次^[5]。但在每次比较的过程中又分为两种情况: 比较 1 位就得出数据不同的结论和比较 $L-1$ 位才得出数据不同的结论。因此, 该算法的最大循环次数为 $N + (L-1)(\log_2^N + 1)$ 。

3 实验结果

以下数据是采用 TC3.0 运行得到的。实验方法: 用随机函数产生 12 000 个数值, 排序后, 用二分查找、基于位的二分查找两种方法在序列中查找 M 个数值, 实验结果如表 1 所示。

实验结果表明, 基于位的二分查找速度非常接近比较式查找算法中平均工作量最小^[1]的二分查找的速度。分析其略慢的原因, 在于二进制位的频繁比较损失了直接利用二进制数据带来的时间效率。

表 1 查找速度比较试验结果 (含随机数产生时间和排序时间)

Table 1 Test results of search speed (time of producing random number and sorting is included)		
数据个数	二分查找	基于位的二分查找
100	9.560 440	9.560 440
500	10.989 011	11.263 736
1 000	12.857 143	13.241 758
5 000	27.527 473	29.725 275
10 000	45.824 176	50.329 670
12 000	53.186 813	58.571 429

4 小 结

基于位的二分查找算法适用于大规模整数构成的集合,是直接利用数据的二进制位信息进行比较,无需对数据进行二进制与十进制间的转换。无疑,它丰富了数据查找算法。笔者经过多次上机实践,实验结果稳定,所以具有一定的通用性,可用于直接使用二进制数据进行操作的场合。另外,从实验结果看出,直接使用二进制不总是比使用十进制速度快,什么条件下使用二进制更快还有待于研究。

参考文献:

- [1] 夏晓明,禹宏涛.混合查找算法[J].计算机应用研究,1995(3):23-24
- [2] 邝继顺,颜运昌.一种基于区间约束的折半查找算法[J].小型微型计算机系统,2001,22(11):1396-1398.
- [3] 邓勇,刘琪.二进制折半查找算法在DSP上的实现[J].国外电子元器件,2001,8:62-64.
- [4] 陈清华,朱红,杨静宇.一种B快速排序算法[J].计算机工程,2002,28(2):96-98.
- [5] 卢开澄.计算机算法导引 设计与分析[M].北京:清华大学出版社,1996:45

A Binary Search Algorithm Based on Bit

CHENG Ya-ping FU De-sheng

(Department of Computer Science and Technology, NUIST, Nanjing 210044, China)

Abstract This paper presents a new binary search algorithm which makes use of bit message to delimit the searching scope. The feasibility of the algorithm is proved by experiment and the algorithm can be used in some special cases.

Key words algorithm; binary search; bit

附录:基于位的二分查找算法的源程序主要部分

```
#include<stdio.h>
#include<math.h>
```

```

#include <time.h>
#define N 12000
#define CONST1 0x7fff
#define CONST2 0x8000

int a[N]
void main()
{ int n1, i, j, z, k, p, s;
  unsigned h;
  randnum();
  bubblesort();
  for( i= 0; i< N; i+ + )
    printf( "% 7d", a[ i] );
  n1= 0;
  while( ( unsigned) ( a[ n1]& CONST2) == CONST2 && ( n1< N) )
    { a[ n1] = ~ ( ( unsigned) a[ n1]& CONST1);
      n1+ +; }
  for( s= 0; s< M; s+ + )
  { z= temp[ s];      /* temp中含待查找数据*/
  p= L- 2;
  b= ( unsigned) ( z& CONST2);
  if( b== CONST2)
    { z= ~ ( ( unsigned) z& CONST1);
      j= 0;
      i= n1- 1;
      while( i>= j)
        { k= ( i+ j) / 2;
          while( ( getbit( a[ k], p) == getbit( z, p) )&& ( p>= 0) ) /* 函数 getbit可取出指定
的二进制位*/
            p- -;
          if( p== - 1)
            { printf( "% 15d\n", k);
              break; }
          else if ( getbit( z, p) == 1)
            { i= k- 1;
              p= L- 2; }
          else
            { j= k+ 1;
              p= L- 2; }
        }
      if( i< j)

```

```
printf( "There is no the number \n");
}
else
{ i= n1;
j= N - 1;
while( i < = j)
{ k= ( i+ j) /2
while( ( getbit( a[ k], p) == getbit( z p) )&& ( p > = 0) )
p- -;
if( p == - 1)
{ printf( "% 15d\n", k);
break;
}
if( getbit( z p) == 1)
{ i= k+ 1;
p= L- 2 }
else { j= k- 1;
p= L- 2 }
}
if( i > j)
printf( "There is no the number \n");
}
}
}
```