k 分搜索的时间复杂度分析

傅晓航,郑欢欢
(南京理工大学 计算机科学与工程学院,江苏 南京 210094)

摘 要:分治策略的思想是将一个规模较大的问题分解为多个形式相同的子问题来解决。搜索是指在一个排好序的数组中寻找与给定数值 x 相等的元素,传统的搜索算法是遍历,而二分搜索是一种基于分治策略的搜索算法。二分搜索是将数组每次分为相等的两部分,将待查元素 x 与数组中间的元素比较,若相等则搜索成功;否则将搜索范围缩小为原来的一半,之后以此类推,直到找到待查元素,与遍历相比,二分搜索复杂度明显降低。以二分搜索为基础,每次可以将数组分为更多部分,即 k 分搜索,探寻 k 为何值时 k 分搜索算法的时间复杂度最低,能够对搜索算法进一步优化。通过分析、归纳与证明,得出 k 分搜索的时间复杂度为 $O(k\log_k n)$,由于该函数是递增的,因此二分搜索是效率最高的搜索算法,复杂度为 $O(\log_2 n)$;此外,当 $k=n$ 时,k 分搜索退化为遍历,复杂度退化为 $O(n)$。

关键词:分治算法;二分搜索;k 分搜索;最优算法;归纳法
中图分类号:TP301.5 **文献标识码**:A **文章编号**:1673-629X(2021)02-0175-05
doi:10.3969/j.issn.1673-629X.2021.02.032

Time Complexity Analysis of k-Bisection Search

FU Xiao-hang, ZHENG Huan-huan
(School of Computer Science & Engineering, Nanjing University of Science & Technology, Nanjing 210094, China)

Abstract: The idea of a divide-and-conquer strategy is to solve a large problem by breaking it down into subproblems of the same form. Search refers to finding the elements equal to the given value x in an ordered array. The traditional search algorithm is traversal, while the binary search is a search algorithm based on divide-and-conquer strategy. The binary search is to divide the array into two equal parts each time, compare the elements to be searched with the elements in the middle of the array, if equal, the search is successful; otherwise, the search scope will be reduced to half of the original, and so on, until the elements to be checked are found. Compared with traversal, the complexity of binary search is significantly reduced. On the basis of binary search, the array can be divided into more parts each time, that is, k-bisection search. The time complexity of k-bisection search algorithm is the lowest when searching for the value of k, which can further optimize the search algorithm. Through analysis, induction and proof, the time complexity of k-score search is $O(k\log_k n)$. Because the function is increasing, the binary search is the most efficient search algorithm, of which the complexity is $O(\log_2 n)$. In addition, when $k=n$, k-bisection search degenerates to ergodic, and the complexity degenerates to $O(n)$.

Key words: divide-and-conquer; binary search; k-bisection search; optimal algorithm; inductive method

0 引 言

分治算法是一个将规模较大问题分解为多个规模相同子问题的算法[1]。许多算法都是基于分治策略实现的,例如搜索、快速排序、归并排序、TOP-K 算法等等;此外分治策略能够应用于许多算法的优化,例如防火墙规则匹配算法、蚁群算法、并行碰撞检测算法等等,相关研究如文献[2-5]。分治策略的核心思想是将一个问题等分为多个形式相同的子问题,图 1 所示为每次将问题划分为两个相等子问题时的分治算法。为提高算法的性能,该文将研究将原问题分解为多少个子问题时分治算法的效率最高。

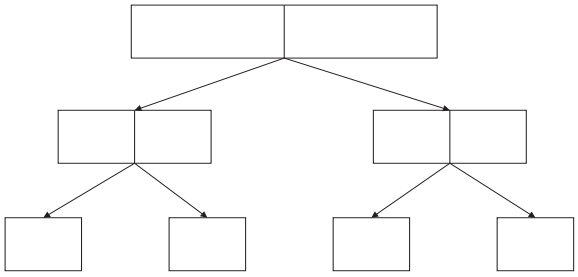

图 1 每次划分两子问题的分治算法

该文以搜索为例,搜索是指在一个排好序的数组

收稿日期:2020-03-27 **修回日期**:2020-07-28
基金项目:国家级大学生创新创业训练计划项目(201910288048Z)
作者简介:傅晓航(1999-),男,CCF 会员(B8366G),研究方向为分布式计算。

中寻找与给定数值 x 相等的元素,很容易想到的方法是将数组遍历,但是遍历的复杂度为 $O(n)$,当数组的规模 n 很大时,查找次数将会很大。

二分搜索是将数组每次分为相等的两部分,将待查元素 x 与数组中点处的元素 $a[n/2]$ 比较,若 $a[n/2]=x$ 则搜索成功;若 $a[n/2]>x$,则待查元素 x 在数组左半区;若 $a[n/2]<x$,则待查元素 x 在数组右半区。将待查元素反复地与数组中点比较,就可以不断地把解的范围缩小到原来的一半,这样时间复杂度会由 $O(n)$ 减小到 $O(\log_2 n)$,性能有显著提升。搜索算法能够应用于一些数据结构如二叉搜索树,优化搜索算法能够进一步提高这些数据结构的性能。

根据分治策略的研究可以想到,如果每次将数组分为相等的更多部分,比如三部分甚至是 k 部分,算法性能还会不会有所提升。每次将数组分为更多的部分,则每趟查找能够将数组缩小到更小的范围,然而每趟需要进行额外的查找,从定性分析的角度,总的查找次数增加或减少均是有可能的。随着 k 的增加,平均查找次数可能先减少,达到最低后再增加;也有可能一直增加,后续将从定量分析的角度,对 k 分搜索的时间复杂度进行证明。

该文将对二分、三分、四分搜索所需的平均查找次数进行复杂度分析,并进行比较,寻找规律,最后分析出 k 分搜索的时间复杂度。通过将每种情况下的查找次数求和并平均,得到算法的平均查找次数,进而得到其时间复杂度。可以证明, k 分搜索的时间复杂度为 $O(k\log_k n)$,易知 $k\log_k n$ 是单调递增函数,因此二分搜索是效率最高的算法。此外,当 $k=n$ 时, k 分搜索退化为遍历,时间复杂度退化为 $O(n)$ 。

1 二分搜索与三分搜索

在本节中,首先通过分析平均查找次数的方式分析二分搜索的时间复杂度,然后按照相同的方式分析三分搜索的时间复杂度,最后对比两种算法在实际数据下所需要的查找次数,并分析两者复杂度差异的原因,为寻找 k 分搜索的规律做铺垫。

1.1 二分搜索

二分搜索法,是通过不断缩小解可能存在的范围,从而求得问题最优解的方法。二分搜索可应用于二叉搜索树、密度峰聚类、网络拓扑探测等算法,相关研究如文献[6-9]。二叉搜索树的左子树若不为空,则其上的所有节点的值均小于根节点,同样,若右子树不为空,则其上的所有节点的值均大于根节点。二叉搜索树便是利用了二分搜索的性质,既能够快速地查找,又能方便地添加与删除节点,是一种高效的数据结构。

二分搜索能够应用于两种常见的问题中:一是应用于最优解问题,求最小的 x 使得 x 满足条件 $M(x)$,可以用二分搜索来求得最小的 x[10];二是在有序数列中查找值,即在一个排好序的数列中寻找与给定值相等的元素。该文主要通过后者来分析二分搜索的时间复杂度以推出 k 分搜索的时间复杂度。

二分搜索的方法如图 2 所示,先将待查元素 x 与二分之一处的元素比较,若相等,则查找成功;若该元素大于 x ,则继续在数组左半区查找;若该元素小于 x ,则继续在数组右半区查找。这样每次都可以将搜索范围缩小一半。

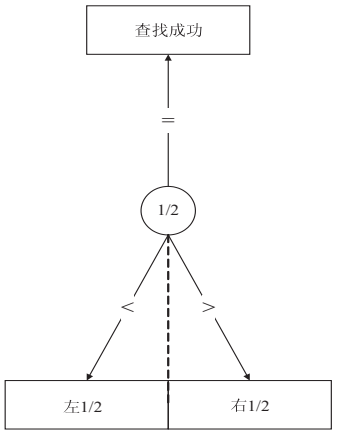

图 2 二分搜索的过程

二分搜索的伪代码如下:

```
1.  low ← 1;high ← n;j ← 0;
2.  while(low ≤ high)and(j = 0);
3.     mid ← (low + high)/2;
4.     if x = A[mid] then j ← mid;
5.     else if x < A[mid] then high ← mid - 1;
6.     else low ← mid + 1;
7.  end while;
8.  return j。
```

图 3 所示为二分查找过程的判定树,要在一个有序数列[2 4 6 9 11 15 18 22 24 31 35]中寻找值为 22 的元素,首先将 22 与数组中点处的元素 15 比较,22 大于 15,则待查元素在数列右半区;再将其与数列右半区中点处的元素 24 比较,发现 22 小于 24,则继续在左半区查找;而 22 大于左半区中点处的元素 18,则继续在右半区查找,最终找到 22。

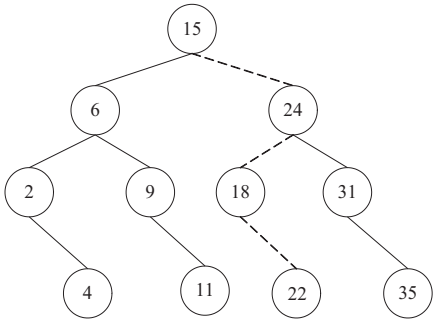

图 3 二分搜索查找过程的判定树

时间复杂度的分析以分析每个元素被查找到的概率的方式进行。经过 1 次查找就能被找到的元素个数为 2^0,经过 2 次查找就能被找到的元素个数为 2^1,以此类推,经过 i 次查找就能被找到的元素个数为 2^{i-1}。设 S 为所有元素被查找到所需要的总次数,则有:

$$S = 1 \times 2^0 + 2 \times 2^1 + 3 \times 2^2 + \cdots + (i-1) \times 2^{i-2} + i \times 2^{i-1} \quad (1)$$

通过错位相减法可以求解式(1),得到:

$$S = (i-1) \times 2^i + 1 \quad (2)$$

由 $1 + 2 + 4 + \cdots + 2^{i-1} = 2^i - 1 \geqslant n$,可以得到 $i \leqslant \log_2(n+1)$,代入式(2)可得:

$$S \leqslant (\log_2(n+1) - 1)(n+1) + 1 \quad (3)$$

每个元素被查找到的概率为 $1/n$,则时间复杂度 $T(n) = S/n$,代入得:

$$T(n) = \frac{1}{n}[(\log_2(n+1) - 1)(n+1) + 1] \approx \log_2(n+1) - 1 \quad (4)$$

即二分搜索平均查找次数的时间复杂度为 $\log_2(n+1) - 1$。通过图 4 可以看出,二分搜索与线性查找相比,时间复杂度由 $O(n)$ 降到了 $O(\log_2 n)$,效率得到了明显的提升。

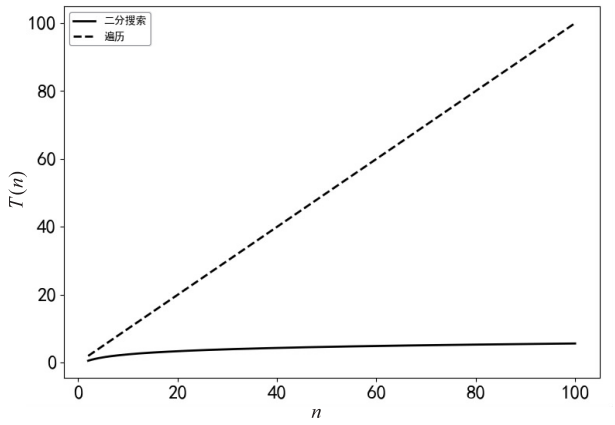

图 4　二分搜索与遍历的时间复杂度的比较

例如,对一个规模为 1e6 的问题,遍历需要的平均查找次数为:

$$T(n) = \frac{1}{n}[1 + 2 + \cdots + n] = \frac{1+n}{2} = 5e5 \quad (5)$$

而二分搜索平均仅需 20 次即可查找成功,极大地提高了算法的性能。

利用二分搜索高效查找的性质,能够将二分搜索应用于一种数据结构即二叉搜索树,又称二分搜索树、二叉排序树,如图 5 所示。二叉搜索树是一种二叉树,对于树中的每一个节点,若其左子树存在,则根节点的值不小于其左子树中每一个节点的值;若其右子树存在,则根节点的值不大于其右子树中每一个节点的值,二叉搜索树的相关研究如文献[11-14]。若二叉搜索树为满二叉树,则其搜索的时间复杂度为 $O(\log_2 n)$,若每层只有一个节点,则复杂度为 $O(n)$,因此二叉搜索树的时间复杂度为 $O(\log_2 n) \sim O(n)$。

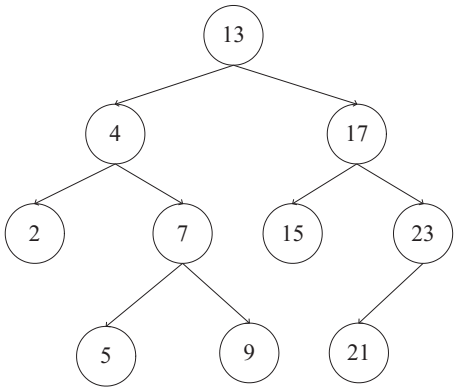

图 5　二叉搜索树

二叉搜索树能够做到与线性二分查找一样的时间复杂度 $O(\log_2 n)$。线性二分查找需要保证待查序列是单调的,因此增加或者删除节点时,需要较多的操作;而二分搜索树能够高效地对节点进行增加或者删除操作,是对二分搜索进行优化的一种数据结构。

后续将继续分析,若将问题划分为更多的子问题,即二分搜索、k 分搜索,能否更好地提升搜索算法的性能。

1.2　三分搜索

三分搜索的方法如图 6 所示,先将待查元素 x 与三分之一处的元素比较,然后将 x 与三分之二处的元素进行比较[15]。若相等,则搜索成功;若 x 小于第一个元素,则继续在左三分之一部分查找;若 x 大于第一个元素小于第二个元素,则继续在中间三分之一部分查找;若 x 大于第二个元素,则继续在右三分之一部分查找。

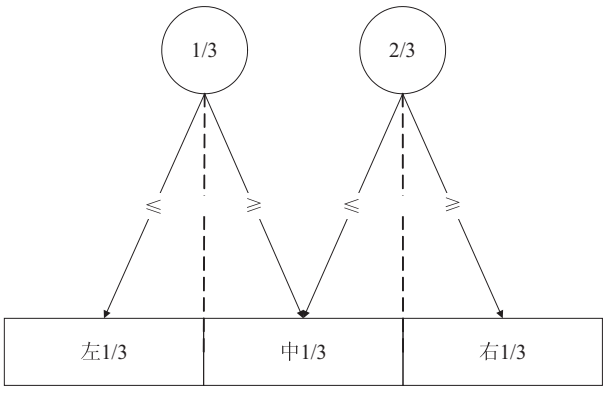

图 6　三分搜索的过程

三分搜索每次能将查找范围缩小三分之一,继续查找,直到找到待查元素 x。

其伪代码如下:

```
1.  low ← 1;high ← n;j ← 0;
2.  while(low ≤ high)and(j = 0);
3.      midl ← low + (high - low)/3;
4.      midr ← low + 2(high - low)/3;
```

```
5.    if x < A[midl] then high ← midl - 1;
6.    else if x = A[midl] then j ← midl;
7.    else;
8.      if x < A[midl] then low ← midl + 1;
9.      else if x = A[midl] then j ← midr;
10.      else low ← midr + 1;
11.  end while
12.  return j。
```

例如对于一个有序数列[2 4 6 9 11 15 18 22 24 31 35 39],寻找与值 22 相等的元素,首先将 22 与三分之一处的元素 9、三分之二处的元素 24 比较,易知 9<22<24,则继续在中间三分之一部分[11 15 18 22]查找;再分别与 11 和 18 比较,发现大于 18,则在右三分之一部分查找,最终找到元素 22。

时间复杂度的分析方法与二分搜索类似,在查找过程中,第 1 趟需要找 2 次,能够被找到的元素有 2×3^0 个,第 2 趟需要找 2×2 次,能够被找到的元素有 2×3^1 个,第 $i-1$ 趟需要找 $2\times(i-1)$ 次,能够被找到的元素有 $(i-1)\times3^{i-2}$ 个,第 i 趟需要找 $2\times(i-2)$ 次,能够被找到的元素有 $i\times3^{i-1}$ 个。每个元素被查找到的概率为 $1/n$,则时间复杂度 $T(n)=4S/n$,其中:

$$S=1\times3^0+2\times3^1+3\times3^2+\cdots+(i-1)\times3^{i-2}+i\times3^{i-1} \tag{6}$$

通过与二分搜索相同的方法可以得到:

$$T(n)=2\frac{n+1}{n}\log_3(n+1)-1\approx 2\log_3(n+1)-1 \tag{7}$$

即三分搜索平均查找次数的时间复杂度为 $2\log_3(n+1)-1$。

1.3 二分搜索与三分搜索的比较

由 1.1 节和 1.2 节可知二分搜索的时间复杂度为 $\log_2(n+1)-1$,三分搜索的时间复杂度为 $2\log_3(n+1)-1$,易知 $\log_2(n+1)-1<2\log_3(n+1)-1$,故二分搜索的效率更高。图 7 为随着数组规模 n 的增长,两者时间复杂度的比较。

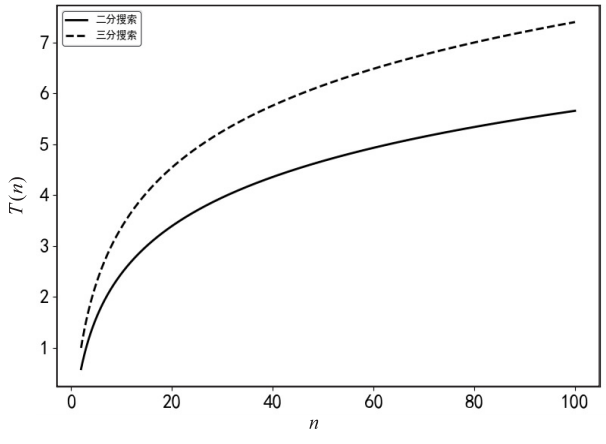

图 7 二分与三分搜索时间复杂度的比较

表 1 为在实际数据下,分别使用规模为 100、1 000、10 000、100 000 的数组,二分搜索与三分搜索所需的查找次数。虽然有个别情况三分所需次数更少,这是因为被查找的位置恰好在三分点上,但从总体上看,还是二分所需次数更好。

表 1 二分与三分搜索在实际数据下的比较

数组规模	100		1 000		10 000		100 000	
二分搜索	6	7	8	8	9	13	12	18
三分搜索	8	5	11	10	13	16	17	23

无论是从平均查找次数的复杂度分析还是通过实例分析都可以看出二分搜索比三分搜索的效率更高。与二分搜索相比,三分搜索虽然每次将待查数组缩小到更小的范围,但是二分搜索每趟只需要查找一次,即中点处的元素,而三分搜索需要两次,即三分之一与三分之二处的元素,每趟搜索就增加了一次查找,随着搜索次数的增加,三分搜索所需要的额外查找次数越来越多。因此,三分搜索与二分搜索相比,平均查找总次数更多,时间复杂度更高。

2 k 分搜索

为了找出 k 分搜索的复杂度,首先分析四分搜索以找出规律,分析方式同上,可以得出:

$$3S=(i-\frac{1}{3})4^i+\frac{1}{3} \tag{8}$$

则时间复杂度为:

$$T(n)=3\frac{n+1}{n}\log_4(n+1)-1\approx 3\log_4(n+1)-1 \tag{9}$$

可以看出,四分搜索与三分搜索相比,时间复杂度再一次增加,据此推测,k 分搜索的时间复杂度是随着 k 的增加而增加的,以下过程证明了这一点。

对于 k 分搜索有:

$$(k-1)S=\frac{i-1}{k-1}k^i+\frac{1}{k-1} \tag{10}$$

时间复杂度为:

$$T(n)=(k-1)^2\frac{1}{n}S=(k-1)\frac{n+1}{n}\log_k(n+1)-1 \tag{11}$$

上式可近似为 $(k-1)\log_k(n+1)-1$,随着 k 的变化,时间复杂度的变化如图 8 所示。

可以看出时间复杂度 T 是随着 k 的增加而增加的,因此二分搜索的效率是最高的。虽然 k 分搜索每次能将搜索范围减小到原来的 $1/k$,即公式中的 $\log_k$,但是每趟搜索需要查找 k 次,即公式中的系数 k,因此,随着 k 的增加,需要的查找次数也越来越多,因此

时间复杂度也是增加的。

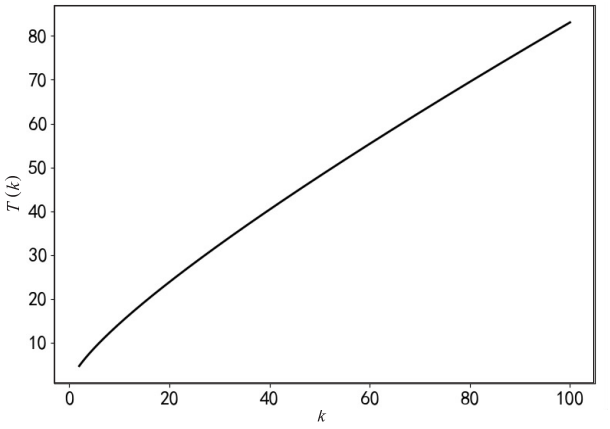

图 8　随着 k 的升高 k 分搜索时间复杂度的变化

因此,各种算法如快速排序、极限情况,当 $k = n$ 时即为遍历。此时每次将搜索范围缩小为 $1/n$,只需一趟搜索,但是需要搜索 $n - 1$ 次,即与前 $n - 1$ 个元素均要进行比较,如图 9 所示,与遍历的方式是相同的。

图 9　n 分搜索

时间复杂度为:

$$T(n) = (n - 1)\frac{n + 1}{n}\log_n(n + 1) - 1 \qquad (12)$$

如图 10 所示, n 分搜索与遍历的复杂度 n 是十分接近或者说近似相同的。

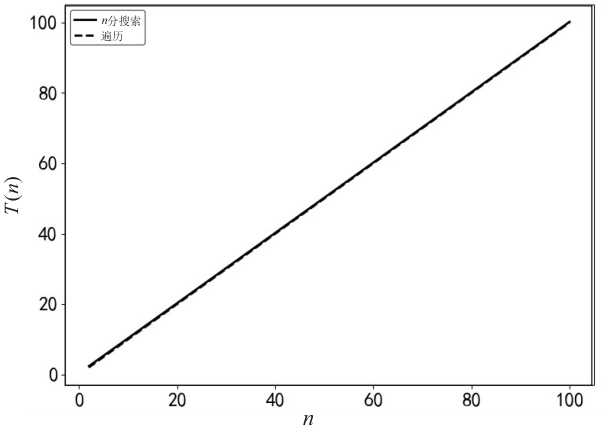

图 10　遍历与 n 分搜索的时间复杂度的比较

3　结束语

通过一系列的分析与证明,研究了二分、三分、k 分搜索的时间复杂度,得出如下结论:(1)证明了 k 分搜索的时间复杂度是 $O(k\log_k n)$;(2)二分搜索是效率最高的搜索算法,对于 k 分搜索来说, k 越大,算法效率越低。虽然三分、k 分搜索与二分搜索相比每次查找能将待查找数组缩小到更小的范围,但是二分搜索每趟只需要查找一次,三分搜索需要两次, k 分搜索需要 $k - 1$ 次,这就增加了查找次数,使得时间复杂度升高;(3)当 $k = n$ 时, k 分搜索退化为遍历,时间复杂度退化为 $O(n)$ 。

参考文献:

[1] CORMAN T H. 算法导论[M]. 潘金贵,译. 第 3 版. 北京:机械工业出版社,2013.

[2] 孙名松,张立新,杜春燕. 增量支持向量机算法研究[J]. 计算机技术与发展,2011,21(5):40-43.

[3] 李　中,李　晓. 一种性能优化的防火墙规则匹配算法[J]. 计算机应用研究,2013,30(4):1205-1207.

[4] 陈　娟,陈　峻. 求解多重序列比对问题的蚁群算法[J]. 计算机应用研究,2007,24(1):25-30.

[5] 赵　伟,谭睿璞,杨秋娜,等. 基于着色算法的并行碰撞检测算法[J]. 计算机应用研究,2009,26(5):1695-1699.

[6] 陈　希,李玲娟. 基于降维和聚类的协同过滤推荐算法[J]. 计算机技术与发展,2020,30(2):138-142.

[7] 孙启航,杨鹤标. 基于编辑距离的序列聚类算法的优化[J]. 计算机技术与发展,2018,28(3):109-113.

[8] 王　尚,卢泽新,彭　伟,等. 一种基于二分搜索的网络拓扑探测方法[J]. 计算机应用研究,2011,28(11):4296-4298.

[9] 黄　谭,苏一丹. 基于混合用户模型的二分图推荐算法[J]. 计算机技术与发展,2014,24(6):145-148.

[10] 秋叶拓哉. 挑战程序设计竞赛[M]. 北京:人民邮电出版社,2013.

[11] 邢海花,胡　丹,贺　辉,等. 一种基于二分查找的快速降型算法[J]. 北京师范大学学报:自然科学版,2018,54(2):179-185.

[12] LINSANGAN N B. Automation of quantification activities of semiconductor-backend process using binary search algorithm[C]//Proceedings of 2017 international conference on advances in image processing (ICAIP 2017). Bangkok, Thailand:ACM,2017:118-122.

[13] 杨智明,夏春梅. 几种常用查找算法的对比分析[J]. 保山学院学报,2017,36(2):57-59.

[14] 王文霞. 基于贪心算法构建最优二叉查找树[J]. 山西师范大学学报:自然科学版,2015,29(1):40-44.

[15] 陈新一. 三分搜索法在数组排序中的应用[J]. 科学技术与工程,2008,8(24):6612-6613.