基于结点群的高效的动态二分查找器

张朝霞 韩素青 亓 慧
(太原师范学院计算机系 太原 030012)

摘 要 通过对几种改进的二分查找算法的分析和总结,提出了一种基于结点群的更为高效的动态二分查找器。该二分查找器不仅使查找效率得以提高,而且使存储结构得以改进,既实现了动态的实时查找,又便于灵活地进行元素尤其是元素群的插入、删除等操作。另外,实验表明,当在大量数据中查找时,该算法明显优于以前改进的所有二分查找算法。

关键词 二分查找,Fibonacci查找,改进的类Fibonacci查找,动态二分查找器
中图法分类号 TP311 **文献标识码** A

Highly Efficient and Dynamic Binary Searcher Based on Node Group

ZHANG Zhao-xia HAN Su-qing QI Hui
(Department of Computer, Taiyuan Normal University, Taiyuan 030012, China)

Abstract Based on the analysis and summary of several improved binary search algorithm, we put forward a more highly efficient and dynamic binary searcher based on node group, which is not only improved in the search efficiency, but also in the storage structure. This algorithm has achieved the dynamic real-time search, and it especially allows to insert and delete a group of elements. In addition, we discovered that the scheme is obviously better than the previous binary search algorithm in searching a large amount of data.

Keywords Binary search, Fibonacci search, Improved Fibonacci search, Dynamic binary searcher

随着现代网络的广泛应用,搜索也日益频繁,因此,有效的搜索技术是当前研究的热点之一。目前已有的基本搜索方法有顺序查找算法、二分算法和索引查找算法等。

当数列有序时,最常采用的搜索方法是二分查找(Binary Search)[1]算法。二分查找算法适用于静态查找表是有序表、并采用顺序存储结构的情况。由于查找表的前置条件是一个已经排好的序列(为方便起见,本文假设该序列以升序排列),因此,查元素时,可以首先与位于序列中间的某个元素进行比较,如果所查找的元素位于该中间元素之后,就在当前序列的后半部分继续查找,如果位于该元素之前,就在当前序列的前半部分继续查找,直到找到相同的元素,或者所查找的序列范围为空为止。当有序表的长度为 n 时,其判定树的深度为 $\lfloor \log_2 n \rfloor + 1$,平均查找长度为 $ASL = (n+1)/n \cdot \log_2(n+1) - 1 \approx \log_2(n+1) - 1 (n > 50$ 时)[2]。

综上可知,二分查找算法充分利用了元素之间的次序关系,在最坏的情况下,采用分治策略,仅以 $O(\log_2 n)$ 的时间复杂度即可完成搜索任务[3]。但在有些情况下,二分查找算法的效率很低,比如当数据量很大而要查找的关键字位于序列的最开始位置时,算法会浪费很多不必要的系统资源来执行冗余的二分操作;这意味着,它不能有效地实现动态实时查找,也不适合链式存储的数据查找。因此,很多学者提出针对二分查找算法的改进。

1 二分查找算法的两个典型的改进算法

本节首先介绍二分查找算法的两个典型的改进算法:Fibonacci查找(Fibonacci Search)算法和分档定位查找算法。

1.1 Fibonacci查找(Fibonacci Search)算法

首先,文献[2,4,7]提出的Fibonacci查找算法是二分查找算法的变形。本质上,这些变形算法均为准二分查找,只是"二分"的概念有所不同罢了。它们和二分查找的共性是将数据查找范围切成两半,直至确定具体的位置或查找不成功。不过,它们在效率和应用范围上与二分查找有一定的区别。二分查找算法利用除法运算来缩小查找范围;而Fibonacci查找算法利用Fibonacci数列建立相应的树状结构来缩小查找范围,本质上是用加减运算来缩小查找范围。理论上,在计算机处理运算指令中,加减运算的效率要高于乘除运算的效率,因此,Fibonacci查找算法在效率上优于二分查找算法。然而,Fibonacci查找算法与二分查找算法一样,采用的仍然是顺序表存储,所以元素的移动(插入和删除等操作)并不方便,动态存储的效率也不高。

1.2 分档定位查找算法

文献[5]提出的分档定位查找算法有许多优势:第一,对

到稿日期:2012-09-11 返修日期:2012-12-09 本文受国家自然科学基金项目(61273294),山西省教育厅高校科技开发项目(20121108),山西省科技厅基础条件平台项目(2012091003-0104)资助。

张朝霞(1975-),女,硕士生,讲师,主要研究方向为算法设计、模式识别和数据挖掘,E-mail:zzxcn@sina.com;韩素青(1964-),女,博士,副教授,主要研究方向为机器学习和数据挖掘;亓 慧(1981-),女,硕士,讲师,主要研究方向为数据结构、算法设计。

于待查找的某个数，无需“比较”，只要“计算”，就可以直接在该查找表中确定一个数据“档”作为查找目标；第二，可以在该“档”范围内使用二分查找；第三，适用于数据量很大的任意数据查找表；第四，在避免全程查找的同时可以避免“冲突”现象[6]。总体来说，分档定位查找算法虽然在动态存储或动态查找表方面确实有很大改进，但是对于档内排序还有进一步改进的空间。本文将 Fibonacci 查找算法与分档定位查找算法的优点结合在一起，提出了新的二分查找算法的改进。

2 改进的二分查找算法及其算法分析

二分查找算法用有序表的中间记录对有序表进行分割，而 Fibonacci 查找算法通过构造斐波那契数列对有序表进行分割，并且查找区间的两个端点和分割点都与斐波那契数列有关。结合 Fibonacci 查找算法的优点，在采用原算法二叉排序树作为查找判定树的基础上，本文提出了一种基于结点群的更为高效的动态二分查找器，与前面两个算法相比，该算法具有以下两方面的优势。第一个优势是借助了一个新的“类斐波那契”[8-10]数列 $H(i)$。该数列与 Fibonacci 数列相比，发散速度要快很多，并且计算只用加减运算、不用除运算。因此，基于该数列设计的二分查找算法可以降低 Fibonacci 查找算法的判定树的高度，并且缩短平均查找长度，这意味着改进后的算法拥有比 Fibonacci 查找算法更快的查询速度。第二个优势是本文给出的二分查找器在存储结构上也有较大的改进。采用有序静态链表结构，结合逆序算法[6]的优点，在整个“结点群”插入或删除结点后重新收集一次静态链表，实现了“结点群”动态查找的可靠性和高效性的统一。

2.1 基于结点群的高效的动态二分查找器

本部分要介绍我们提出的一种基于结点群的更为高效的动态二分查找器，它利用新的类“类斐波那契”数列建立相应的树状结构来缩小查找范围。2.1.1 节首先给出新的“类斐波那契”数列 $H(i)$ 的构造及其优势；2.1.2 节基于新的“类斐波那契”数列给出了改进的“类斐波那契”查找算法。

2.1.1 新的“类斐波那契”数列 $H(i)$[8-10]

“类斐波那契”数列 $H(i)$ 如下：$H(0)=0$，$H(1)=1$，$H(i)=2H(i-1)+1(i\geqslant 2)$。

根据数学归纳法很容易证明，$H(i)$ 的通项公式为 2^i-1。而斐波那契数列 $f(i)$ 的构造为：$f(0)=1$，$f(1)=1$，$f(i)=f(i-1)+f(i-2)(i\geqslant 2)$。又因为，$H(i)$ 还可以表示为 $H(i)=H(i-1)+H(i-1)+1$，所以不难看出，与 Fibonacci 数列相比，其发散速度要快很多，并且计算不包含除法运算。具体分析如下。

斐波那契查找算法虽然把折半查找中分割有序表时用的除法运算变成了加法运算，使其平均性能 ASL（平均查找长度）比折半查找要好，但是在最坏情况下的性能却要比折半查找差[2]，以 $n=7$ 为例，折半查找和斐波那契查找所对应的查找树分别如图 1 和图 2 所示。当 $n=7$ 时，折半查找的平均 $ASL=(1+2\times 2+3\times 4)/7=17/7$；而斐波那契查找的 $ASL=(1+2\times 2+3\times 3+4\times l)/7=18/7$。

从图 2 中不难发现斐波那契查找树中存在退化的分支结点（如结点 7 ）。产生这种退化分支结点的原因在于斐波那契数列本身的构造。它与深度为 i 的满二叉树的结点数 2^i-1 相差甚远[2]。

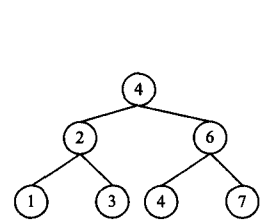

图 1 折半查找树

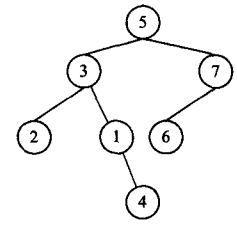

图 2 斐波那契查找树

新的“类斐波那契”数列 $H(i)$ 的通项公式为 2^i-1，它与深度为 i 的满二叉树的结点数 2^i-1 相对应，避免了退化的分支结点。因此，在结点数 n 相同的情况下，新的“类斐波那契”数列与折半查找相比，不仅避免了除法运算，而且降低了查找树的深度；此外，与斐波那契查找相比，避免了退化的分支结点，降低了查找树的深度。

综上可知，利用新的“类斐波那契”数列 $H(i)$，可以在保持斐波那契查找算法优势的基础上，设计出比其查询速度更快的二分查找算法。

2.1.2 改进的“类斐波那契”查找算法

改进的“类斐波那契”查找算法的基本思想是，计算数列 $H(i)$，直至找出第一个不小于表长 n 的项 $H(i)$；依据 $H(i)$ 将有序表 $T[1,\cdots,n]$ 分割为 3 个子表：$T[1,\cdots,H(i-1)]$，$T[H(i-1)+1]$，$T[H(i-1)+2,\cdots,n]$，其中 $T[H(i-1)+1]$ 中只有一个元素，$T[1,\cdots,H(i-1)]$ 的表长为 $H(i-1)$，而 $T[H(i-1)+2,\cdots,n]$ 的表长不大于 $H(i-1)$。

查找时，首先将欲查找的数据元素的关键字与 $T[H(i-1)+1]$ 中唯一元素的关键字进行比较。若它们相等，则查找结束，否则根据比较结果继续用同样的方法在 $T[1\cdots H(i-1)]$ 或 $T[H(i-1)+2,\cdots,n]$ 中进行分割查找。

改进的“类斐波那契”查找算法的类 C 语言程序如下：

```
类斐波那契数的实现，递归方式。
int H(int n) { //斐波那契数的实现
int result=0;
if(n==0)return 0;
else {
result=H(n-1)+H(n-1)+1;
return result;
  }
}
  “类斐波那契”查找算法
int leifib_search(int n,int key,int data[])
{
int low,high,mid;
int index;
low=1,high=n;
while(low<=high)
{
n=high-low+1;
for(index=0;H[index]< n;index++);
mid=low+H[index-1];
if(key>data[mid])
low=mid+1;
else if(key<data[mid])
high=mid-1;
else return mid;
```

}
return -1;

显而易见，改进的“类斐波那契”二分查找算法所对应的查找树具有这样的优点：任何分支结点的左子树均为满二叉树，消除了查找树中的退化分支点，降低了判定树的高度，提高了查询速度[2]。因此，从理论上看，其查找性能明显优于Fibonacci二分查找算法。

2.2 存储结构上的改进——实现了高效的动态查找

存储技术是搜索引擎在提供搜索服务时的关键技术，系统如何存储上百亿的网页数据，如何科学、高效地提供搜索结果，都会影响到用户的“搜索用时”。在常用的搜索算法中，二分查找算法通常利用数组存储元素来提高查找效率，而顺序查找算法则采用链表存储结构来实现动态查找，提高插入和删除操作的灵活性。其实，最好的算法应该是既具有二分查找算法的高效性，又具有顺序查找算法链表存储结构的灵活性。

目前，动态查找表的结构有二叉排序树、B－树、B＋树、键树等，它们都是向下分叉的树结构。这些结构的优点是：每次执行插入和删除操作时，只需在相应结点处重新链接，并保持树的整体不动。缺点是：①所需存储空间过大；②无法采用高效的“分档定位查找[5]”；③对于“结点群（元素群）”的插入或删除效率不高。另外，文档[10]中提出：二叉树的任一结点的左子树结点的值应当小于或等于其父结点的值，而右子树结点的值大于其父结点的值。利用二叉树排序，我们可以将查找范围内的数据组织成二叉树，来取得类似二分查找算法的效果。事实上，将数据中的元素组织成二叉树形式，二叉树排序既具有与二分查找算法相同的查找效率，又具有与链表结构相同的插入、删除操作的灵活性。

如何实现结点群高效的动态查找是数据结构中的重要研究课题。本文给出的基于结点群的更为高效的动态二分查找器采用有序的静态链表结构，并且结合逆序算法的优点，在整个“结点群”插入或删除后重新收集一次静态链表，实现了“结点群”动态查找的可靠性和高效性的统一。

为叙述方便，首先给出如下定义：

定义 1[6] 将一个无序的“主序列（即大群结点）”进行排序（默认为升序），构成一个有序静态链表，称为主表，记作 $d(T)$。

主表是长表，在主表中可以查找、插入和删除结点。

定义 2[6] 对于 $t=0,1,\cdots,n$，$d(t)$ 为主表中的一个结点，定义为（cl，$Link$）（其中，cl 为查找关键字，$Link$ 指向主表中当前结点的下一个结点），称为主结点。

定义 3[6] 将一个无序的“副序列（即小群结点）”进行排序（默认为升序），构成一个有序静态链表，称为副表，记作 $e(Ta)$。

副表比主表短，副表中的结点可以被插入到主表或从主表中删除。

定义 4[6] $e(ta)$ 为副表中的一个结点，定义为（cla，$link$）（其中，cla 为查找关键字，$Link$ 指向副表中当前结点的下一个结点），称为副结点。

定义 5[6] 如果副节点 $e(ta)$ 在主表 $d(T)$ 的结点 $d(t)$ 之后插入，则称 $d(t)$ 为 $e(ta)$ 的“插入点”，记作 $d(t)\nearrow e(ta)$。

插入副节点 $e(ta)$ 的操作可以用 $e(ta).link:=d(t).Link$；$d(t).Link:=e(ta)$ 实现。

定义 6[6] 将副表插入主表或从主表删除后，可以重新收集至一个静态链表中，所形成的新的“主表”称为“汇总表”。

定义 7[6] 当主表为升序时，副表所采用的降序操作称为逆序；反之亦然。

文献[6]提出，一个极佳的动态查找表应具有如下特征：①对于某个副结点（群），只需“计算”，就可以直接在主表中通过动态查找来确定其对应的确切位置，所需时间复杂度仅为 $O(1)$。②动态查找表所需空间复杂度仅为 $O(n)$。③对某个结点（群），在主表中实现其动态查找时，只需在对应结点进行链接，而无需改变其他结点。④查找插入点的次数和链接结点的次数应当最少。即：如果副表中存在一批以同一主结点为插入点的副结点，那么对这些副结点，只需查找一次插入点并做一次链接。

查找副结点的插入点与将副结点插入主表中实际上是算法中相互独立的两个部分。由于主表是有序静态链表构造，因此查找插入点的算法有很多，其中，比较快的算法有二分查找算法和 Fibonacci 查找算法等[2]，其时间复杂度仅为 $O(\log_2 n)$。目前，最快的查找算法应该是分档定位查找，它已经近似地实现了上述极佳的动态查找表具有的4个特征中的特征①和②。本文所给的查找器实现了分档定位查找表由静态表向动态表的跨越，其核心就是要在保持分档定位查找特点的基础上，进一步实现一个极佳的动态查找表应具有特征③和特征④。换句话说，本文给出的算法实现了一个极佳的动态查找表应具备的4种特征。

下面分3小节具体介绍本文给出的基于结点群的更为高效的动态二分查找器实现动态查找时所使用的算法：逆序插入算法、逆序结点群批量插入算法和逆序结点群批量删除算法。其中提到的对主表的查找和对副表元素的查找均采用上面提出的改进的“类斐波那契”查找算法，这些算法的具体过程描述如下。

2.2.1 逆序插入算法

逆序插入算法是将副表 $e(Ta)$ 按表末至表头的降序规则向主表 $d(T)$ 插入。逆序是本文的关键所在，如果采用正序，将会使算法复杂化甚至出错。以下①、②、③和④描述了一个副表逆序插入主表的过程。

①$e[3]$ 插入 $d[T]$：

d[t]	d[0]	d[1]	d[2]	d[3]	d[4]	d[5]	d[6]
cl	2	5	8	15	18	18	26
Link	d[1]	d[2]	d[3]	d[4]	d[5]	e[3]	Λ

e(ta)	e[0]	e[1]	e[2]	e[3]
cla	6	18	23	24
link	e[1]	e[2]	e[3]	d[6]

②$e[2]$ 插入 $d[T]$：

d[t]	d[0]	d[1]	d[2]	d[3]	d[4]	d[5]	d[6]
cl	2	5	8	15	18	18	26
Link	d[1]	d[2]	d[3]	d[4]	d[5]	e[1]	Λ

e(ta)	e[0]	e[1]	e[2]	e[3]
cla	6	18	23	24
link	e[1]	e[2]	e[3]	d[6]

③ $e[1]$插入 $d[T]$：

d[t]	d[0]	d[1]	d[2]	d[3]	d[4]	d[5]	d[6]	d[7]
cl	2	5	8	15	18	18	26	29
Link	d[1]	d[2]	d[3]	d[4]	d[5]	e[1]	d[7]	∧

e(ta)	e[0]	e[1]	e[2]	e[3]
cla	6	18	23	24
link	e[1]	e[2]	e[3]	d[6]

④$e[0]$插入 $d[T]$：

d[t]	d[0]	d[1]	d[2]	d[3]	d[4]	d[5]	d[6]
cl	2	5	8	15	18	18	26
Link	d[1]	e[0]	d[3]	d[4]	d[5]	e[1]	∧

e(ta)	e[0]	e[1]	e[2]	e[3]
cla	6	18	23	24
link	e[1]	e[2]	e[3]	d[6]

主要操作的算法如下：

```
FOR ta:=3 DOWNTO 0 DO//查找 e(ta)的插入点;e(ta)在主表中的插入点为 d(t)。//即 d(t).cl<=e(ta).cla<=d(t+1).cl
e(ta).Link:=d(t).Link;
d(t).Link:=e(ta);//链接副表结点,由于采用副表逆序插入算法,无论哪种结点的插入都只需这两句即可完成,比较简练。
```

2.2.2 逆序结点群批量插入算法

逆序结点群批量插入的过程是：如查得有 $d(t)\nearrow e(ta)$，不是立即将 $e(ta)$插入，而是循环判断 $e(ta-1)$、$e(ta-2)$、$e(ta-3)$是否也应该以 $d(t)$为插入点，若有一批结点都以 $d(t)$为插入点，则将这批结点一次性插入到 $d(t)$之后，这批一次性插入的结点就是结点群。具体过程如⑤所示。

⑤将副表 $e(Ta)$插入主表 $d[T]$中：

d[t]	d[0]	d[1]	d[2]	d[3]	d[4]	d[5]	d[6]	d[7]
cl	2	5	8	15	18	18	26	29
Link	e[0]	d[2]	d[3]	d[4]	d[5]	e[3]	d[7]	∧

e(ta)	e[0]	e[1]	e[2]	e[3]
cla	2.1	3	3.3	4
link	e[1]	e[2]	e[3]	d[1]

2.2.3 逆序结点群批量删除算法

删除一批结点同样有逆序逐点删除算法和逆序结点群删除算法两种。

删除算法看似很简单，只需要在静态链表中将指针跳过被删除结点即可，其实不然。

首先，删除只能采用逆序操作。删除 $d(t)$时，如果采用逆序操作，只需对应删除点 $d(t)$进行{$d(t-1).Link:=d(t).Link$,$d(t).Link:=d(t)$}的操作即可；而如果采用正序删除，删除 $d(t-1)$后，再删除 $d(t)$就会出错，因为 $d(t)$的上一个结点已经不是 $d(t-1)$了，所以{$d(t-1).Link:=d(t).Link$}在此行不通，算法将复杂化。删除 $d(1)$后再插入 $e(0)$的基本操作执行过程如下：

⑥删除 $d(1)$后

d[t]	d[0]	d[1]	d[2]	d[3]	d[4]	d[5]	d[6]	d[7]
cl	2	5	8	15	18	18	26	29
Link	d[2]	d[2]	d[3]	d[4]	d[5]	e[3]	d[7]	∧

⑦形成汇总表后再插入 $e(0)$

d[t]	d[0]	d[2]	d[3]	d[4]	d[5]	d[6]	d[7]
cl	2	8	15	18	18	26	29
Link	e[0]	d[3]	d[4]	d[5]	e[3]	d[7]	∧

e(ta)	e[0]	e[1]	e[2]	e[3]
cla	2.1	3	3.3	4
link	d[2]	e[2]	e[3]	∧

其次，对于插入一批结点并且删除一批结点的复合操作，需要分两步完成。比如先完成插入操作，形成汇总表，然后再完成删除操作。反过来也一样，即首先完成删除操作，形成汇总表，然后再完成插入操作。

无论对副表执行插入还是删除，采用逆序算法都可以实现批量操作，逆序算法包括逆序逐点插入(或删除)算法和逆序结点群插入(或删除)算法两种。为避免出错，在完成副表的一次插入(或删除)操作后，应该马上形成汇总表。逆序算法与分档定位查找算法相结合可以构成一种高效率的动态查找算法[9]。

2.3 改进后的二分查找效率的理论分析

综上所述，本文给出了一类情况下的二分查找算法的改进。即，对于查找存储在外存上的大量数据的情况，我们可以将其视为许多个结点群组成的集合，在结点群之间使用分档定位查找；而在结点群内部可以采用“类斐波那契”查找。传统二分查找运用除法运算来缩小查找范围，而新的“类斐波那契”查找则通过构造新的类 Fibonacci 函数，建立相应的树状结构，同样能保证在缩小范围的过程中仅用到加法的优势，并降低了原 Fibonacci 查找判定树的高度，减少了其 ASL，提高了查询速度。

具体定量评估如下：二分查找中评价一个算法的好坏一般用平均查找长度 ASL 来衡量，但是 ASL 计算起来较复杂。为了更简便地说明算法的效率，我们用最大查找长度 MSL 来表示算法的效率，显然，对于基于分割思想的二分查找算法，其 MSL 的计算应完全取决于查找树的深度。对于原 Fibonacci 查找算法和改进后的查找算法，原斐波那契函数和改进后的类斐波那契函数分别记为 $G(i)$和 $H(i)$，且有

$$G(i)=\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{i-1}-\frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{i-1}$$

$$H(j)=2^j-1$$

式中，i 为原 Fibonacci 查找树的高度，它与深度为 i 的满二叉树的结点数 2^i-1 相差甚远；而 j 为改进后的类 Fibonacci 查找树的高度，它与深度为 j 的满二叉树的结点数 2^j-1 相对应。显然，改进后的 Fibonacci 查找判定树与原 Fibonacci 查找判定树相比，降低了查找判定树的高度，减少了其 ASL，提高了查询速度。

而且，当查找范围内的数据源需要做频繁的变动时，本文采用了有序静态链表结构，结合了对结点群的逆序操作，实现了动态查找，达到了与二分法相同的查找效率的同时又具有链表的插入、删除操作的灵活性。

由上可知，本文结合了动态查找结点群的逆序操作的优点，同时提出在结点群内应用“类斐波那契”查找，在原有的算法基础上都有了进一步改进，既实现了动态查找，又提高了查找效率。

2.4 实验结果分析

为说明笔者改进的算法在效率上高于经典的折半查找算

(下转第 288 页)

Breaking Waves[A]// Proceedings of the 2004 ACM SINGGRAPH/Eurographics Symposium on Computer Animation, 2004[C]. Grenoble: ACM, 2004: 315-324

[2] 林建忠,阮晓东,陈邦国,等. 流体力学[M]. 北京:清华大学出版社,2005:11-13

[3] Muller M, Solenthaler B, Keiser R, et al. Particle-based Fluid-fluid Interaction[A]// Proceedings of EUROGRAPHICS ACM SIGGRAPH Symposium on Computer Animation, 2005[C]. Los Angeles: ACM, 2005: 237-244

[4] 延诃,王章野,廖斌斌,等. 基于物理的海浪场景的真实感建模与绘制[J]. 计算机辅助设计与图形学学报,2008,20(9):1117-1124

[5] 陈曦,何戡,延诃,等. GPU 中的流体场景实时模拟算法[J]. 计算机辅助设计与图形学学报,2010,22(3):396-405

[6] Fourier A. A Simple Model of Ocean Waves[A]// Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, 1986[C]. Dallas: ACM, 1986: 75-84

[7] Yuksel C. Real-time Water Waves with Wave Particle [D]. Brazos County: Texas A&M University, 2010

[8] 苏玉民,付金丽,王卓. 基于三维随机海浪交互式仿真技术研究[J]. 系统仿真学报,2012,24(1):175-179

[9] Tessendorf J. Simulating Ocean Water[A]// Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, 1999 [C]. Los Angeles: ACM, Course Notes 26, 1999

[10] Michel J. Real-Time Synthesis and Rendering of Ocean Water [DB/OL]. http://developer. amd. com/media/gpu _ assets/Mitchell-Real-Time_Synthesis_and_Rendering_of_Ocean_Water (ATITR_Apr05). pdf, 2012-09-14

[11] 任鸿翔. 航海模拟器中基于 GPU 的海洋场景真实感绘制[D]. 大连:大连海事大学,2009

[12] 赵欣,裴炳南. 一种快速的海浪仿真方法[J]. 系统仿真学报,2012,24(1):132-136

[13] Veritas D N. Standard for Certification Maritime Simulator Systems No. 2. 14 January 2011[DB/OL]. http://exchange. dnv. com/publishing/stdcert/Standard2-14. pdf, 2012-09-15

[14] Nouguier F, Guérin C, Chapron B. Choppy Wave Model for Nonlinear Gravity Waves [J]. Journal of Geophysical Research, 2009, 114: C09012

[15] Phillips O. The Equilibrium Range in the Spectrum of Wind-generated Waves[J]. Journal of Fluid Mechanics, 1958, 4: 426-434

[16] 俞聿修. 随机波浪及其工程应用[M]. 大连:大连理工大学出版社,1999:131-174

[17] 李积德. 船舶耐波性[M]. 哈尔滨:哈尔滨理工大学出版社,2007:30-35

[18] Liu Da-gang, Leng Mei. Meteorology and Oceanography for Mariners[M]. Dalian: Dalian Maritime University Press, 2011: 5-7

[19] Ochi M. Ocean Waves: The Stochastic Approach [M]. Cambridge: Cambridge University Press, 1998: 33-50

[20] COST 714 Working Group 3. Measuring and Analyzing the Directional Spectrum of Ocean Waves[M]. Brussels: EU Publications Office, 2005: 3-12

(上接第 247 页)

法,我们利用随机函数给出随机数,采用 vc++语言编程后证实(见表 1),改进后的二分查找器的速度要高于折半查找算法的速度。

表 1 改进的二分查找器与折半查找的比较

查找表中的数据量(个)	折半查找(/ms)	改进后的二分查找(/ms)
1000	134.8	131.8
2000	716.2	684.8
4000	1657.8	1607.8
8000	3661.3	3479.8
160000	8777.5	8721
320000	30577.6	28866.3

结束语 综上所述,与已有的二分查找方案相比,本文提出的基于结点群的动态的二分查找器具有如下优势:首先,通过在结点群之间使用分档定位查找,而在结点群内部使用改进后的"类斐波那契"查找,可以保证在缩小查找范围的过程中体现出加法的优势,因此降低了原 Fibonacci 查找判定树的高度,减少了其 ASL,提高了查询速度,这意味着算法的效率将提高很多。其次,当查找范围内的数据源需要做频繁的变动时,本文采用有序静态链表结构,对结点群使用逆序操作,实现了灵活的动态查找。与折半查找算法相比,其不仅效率高,而且具有链表所特有的插入、删除操作的灵活性。实验表明,我们提出的二分查找器已达到了预期的效果。

总而言之,在查找频繁变动的存储在外存的大量有序数据的情况下,本文提出的基于结点群的更为高效的动态二分查找器具有很好的参考价值,其在实践中的应用价值我们将在随后的研究中进一步探讨。

参 考 文 献

[1] Knuth D E. The Art of Computer Programming, 3: Sorting and Searching [M]. Addison Wesley, 1973

[2] 严蔚敏,吴伟民. 数据结构(C 语言版)[M]. 北京:清华大学出版社,2007:220-222,238-239

[3] iamhaiyang. 二分查找学习札记[OL]. http://blog. chinaunix. net/uid-544794-id-2097252. html, 2012-02-20

[4] 詹炜,等. 一种基于 Fibonacci 数的有序线性表查找算法[J]. 电脑开发与应用,2005,18(12):29

[5] 陈启星,罗启宇. 分档定位排序以及向分档定位查找的发展[J]. 计算机研究与发展,2003,40(5):706

[6] 陈启星,陈彬,陈叶. 用静态链表和逆序插入算法构成的动态查找表[J]. 电脑与信息技术,2007,15(3):2-3

[7] 长春工业大学. 数据结构精品课程[M]. 线性表的查找技术之斐波那契查找,2011

[8] 周德苏,李光明. 对 Fibonacci 静态查找算法的改进[J]. 空军雷达学院学报,2000,14(3):45-47

[9] 罗南超,蹇旭,崔丽. 一种改进的新二分查找算法的研究与实现[J]. 计算机时代,2009,205(7):56

[10] 一道笔试题:用最快速度查出有序数组中的一个数[OL]. http://www. khgl. cn/html/38/n-4z76038. html, 2012