高级数据结构对算法的优化

［摘要］ 使用高级数据结构对算法进行优化的效果有目共睹。本文针对两种基于二权树的高级数据结构：堆和排序二权树展开讨论，说明了两种数据结构的特点、优势和使用方法，并附有实例和数据对比。

［关键字］ 数据结构　　算法优化　　堆　　排序二权树

一、堆

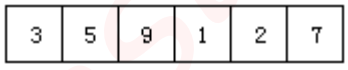
堆是一棵完全二权树，它满足这样的条件：对于每一个非叶节点，它的值都大于它的两个子节点的值（最大堆）或小于它的两个子节点的值（最小堆）。由于堆是一棵完全二权树，所以我们一般采用数组的方法存储堆。如下定义：

Heap:Array[1..n] of Node;

那么，一个最大堆一定满足：$Heap[i] \le Heap[i \text{ Div } 2]\ i \in (2,n)$；同样，对于最小堆有：$Heap[i] \ge Heap[i \text{ Div } 2]\ i \in (2,n)$。不难发现，一个堆的根节点（第一个元素）一定是这个堆中最大的（最大堆中）或是最小的（最小堆中）。这正是堆这种数据结构的价值所在，牢牢抓住这一点，就可以使堆发挥出其最大功效。

首先，我们讨论堆的构造。对于一串无序数列$a_1, a_2 \cdots a_n$，我们将其存入数组中，这样，这一串数实际上已经对应了一个完全二权树。以 3,5,9,1,2,7 为例：

我们可以把数组看作是顺序编号的一棵完全二权树（如图 1）。对于这样一棵完全二权树，要对其进行修改，使其最终形成堆。假定要生成一个最大堆，那么根据最大堆的定义，在这个完全二权树中，没有哪个非叶节点的值要比其子节点还小。所以，要使右边这棵二权树也满足这样的条件，就需要对节点按照从后向前的顺序进行“筛”运算。只不过这里的“筛”不是把节点筛下去，而是把某些节点“筛上去”。从后往前，第一个节点是⑦，但是其父节点⑨的值较大，所以不需要“筛”它。一直搜到节点⑨，我们发现其值要比其父节点③大，所以要把它向上“筛”。我们把⑨和③换个位置，让较大的⑨做父节点，较小的③做子节点。但这还没完，我们发现经过修改后，先前提到的⑦的值比其现在的父节点③的值要大了，所以，我们还要把⑦向上“筛”。于是③又和⑦换了一个位置，变成了⑦的子节点。事实上，为了保证算法的正确性，在每次“筛”的时候，都要“一筛到底”，时刻注意被“筛”下来的节点是否还要继续往下“筛”。注意：如果一个



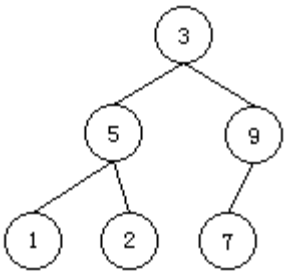


图1

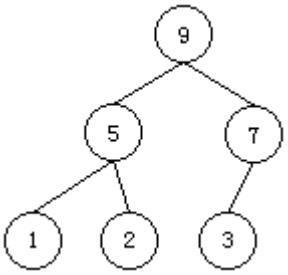


图2

待“筛”的节点的值同时比其两个子节点都要小，就应该用较大的子节点与其对换位置。经过“筛”运算，可以得到这样一个堆：见图 2。它满足最大堆的性质：对于每一个非叶节点，它的值都大于它的两个子节点的值。同时注意：根节点⑨的值是所有元素中最大的。

下面给出构造堆的源程序：

```
For i:=n DownTo 2 Do{从最后一个节点开始从后往前进行“筛”运算}
  If H[i]>H[i Div 2] Then{若此节点的值比其父节点的值要大}
    Begin
     t:=H[i];H[i]:=H[i Div 2];H[i Div 2]:=t;{交换此节点与其父节点}
     j:=2*i;{先将指针指向其左子节点}
     While j<=n Do{下标不越界}
      Begin
       If (j+1<=n) and (H[j]<H[j+1]) Then Inc(j);{若此节点有右子节点并且右子节点的值比左子节点的值要大，则将指针指向其右子节点}
       If H[i]<H[j] Then{若仍需要再往下“筛”，则}
        Begin
         t:=H[i];H[i]:=H[j];H[j]:=t;{交换此节点于其子节点}
         i:=j;j:=2*i{重新定位指针，准备下一次循环}
        End
       Else Break{若不需要再“筛”就退出}
      End
    End;
```

上述算法中“筛”运算的时间复杂度约为$O(\log_2 N)$，是相当高效的。注意到根节点的值是所有节点的值中最大的，抓住这一点，就不难设计出一套时间复杂度约为$O(N\log_2 N)$的算法：把数组分成两个部分，一个部分为已排序区间，在这里的元素都是有序的。另一部分为堆区间，把这里的所有元素理解为一个堆。一开始，已排序区间为空，堆区间为全体元素。首先构造一个堆，把堆的根节点移入已排序区间，然后对剩下来的堆再进行“筛”运算，再次得到一个堆之后，再把堆的根节点移入已排序区间，如此往复直到已排序区间为全体元素，堆区间为空，则排序过程结束。这时，已排序区间中的元素就都是有序的了。这种$O(N\log_2 N)$的算法其实就是所谓的堆排序。（源程序见附录，有关堆排序的效率以及特点参看下面有关排序二权树的内容）

堆在实际解题时有着很重要的应用，使用得好往往能够收到出乎意料的效果。下面用一道例题来说明堆在解题中的应用。

例 1：黑匣子

我们使用黑匣子的一个简单模型。它能存放一个整数序列和一个特别的变量 i。在初始时刻，黑匣子为空且 i 等于 0。这个黑匣子能执行一系列的命令。有两类命令：

ADD(x)：把元素 x 放入黑匣子；

GET：把 i 加 1 的同时，输出黑匣子内所有整数中第 i 小的数。牢记第 i 小的数是当黑匣子中的元素已非降序排序后位于第 i 位的元素。

下面是一个 11 个命令的例子：

编号	命令	i	黑匣子内容	输出
1	ADD(3)	0	3	
2	GET	1	3	3
3	ADD(1)	1	1,3	
4	GET	2	1,3	3
5	ADD(-4)	2	-4,1,3	
6	ADD(2)	2	-4,1,2,3	
7	ADD(8)	2	-4,1,2,3,8	
8	ADD(-1000)	2	-1000,-4,1,2,3,8	
9	GET	3	-1000,-4,1,2,3,8	1
10	GET	4	-1000,-4,1,2,3,8	2
11	ADD(2)	4	-1000,-4,1,2,2,3,8	

现需要一个有效的算法处理给定的一系列命令。ADD 和 GET 命令的总数至多个有 30000 个。

定义 ADD 命令的个数为 M 个，GET 命令的个数为 N 个。

我们用下面得两个整数序列描述命令序列：

1．A(1),A(2),……,A(M)：加入黑匣子的元素序列。所有的数均为绝对值不超过 2000000 的整数。例如在上例中 A=(3,1,-4,2,8,-1000,2)。

2．u(1),u(2),……,u(N)：u(i)表示第 i 个 GET 命令在第 u(i)个 ADD 命令之后，例如在上例中，u=(1,2,6,6)。

你可以假定自然数序列 u(1),u(2),……,u(N)以非降序排列，N≤M，且对于每一个 p

（1≤p≤N）有 p≤u(p)≤M。

输入 M,N, A(1),A(2),……,A(M) ,u(1),u(2),……,u(N)，输出黑匣子的处理结果。

解：显然，此题实际上已经给出了程序基本的算法流程如下：

```
Procedure BlackBox(M,N,A,u);
 Begin
  i:=1;
  For j:=1 To M Do
    Begin
      Add(A[j]);
      While (i<=N) and (u[i]=j) Do
        Begin
          Get(i);
          Inc(i)
        End
    End;
```

当然，此题远没有这么简单。上面程序中的两个过程：Add 和 Get 是要我们自己去编写的，而且，这两个过程的效率直接决定了整个程序的效率。如果这两个过程都能够应付上限为 30000 的测试数据，那么整个算法就是成功的。

事情往往没有这么简单，凭直觉我们就能够发现，上述两个过程的关键在于数据结构的选择。我们先采用朴素的，也是最常用的线性表来解题：

最基本的想法是不考虑黑匣子内数的有序性，每执行一次 Get 命令就把黑匣子内

的数全部排一个序。这种算法实在是太朴素了，不难得出其 Add 过程的时间复杂度为 O(1)，而 Get 过程的时间复杂度为 $O(m\log_2 m)$。这样，整个程序的时间复杂度就为 $O(M+MN\log_2 M)$！我们认为这是一个 $O(n^2\log_2 n)$的算法，在 n 的上限为 30000 时是绝对不可采用的！

显然，上面的算法过于朴素了，我们完全可以时刻保持黑匣子内数的有序性，这样，Get 命令的时间复杂度就被降到了 O(1)。但是，在执行 Add 命令时，我们需要使黑匣子内的数保持有序，这就需要一个插入排序的过程。这样，Add 命令的时间复杂度就应为 O(m)。那么，整个程序的时间复杂度为 O(m+mn)，我们认为这是一个 $O(n^2)$的算法。（源程序见附录。）虽然程序的速度有所提高，但是对于令人恐怖的上限为 30000 的数据，这样的程序的效率依然不令人满意，对于极限数据，一般需要约 11 秒才能出解，这仍然是不能接受的。

因此，我们可以换一种数据结构，采用堆来存储黑匣子内的数并进行运算。堆的重要性质在于堆的根节点的值是堆中全部节点的值中最大（最大堆）或最小（最小堆）的。由此，如果使得堆的根节点就是整个黑匣子中第 i 大的数，那么程序的效率将提高很多。这里有一个很巧的方法：构造两个堆：一个最大堆，一个最小堆，把黑匣子中的数排序，前 i 个数放入最大堆中，之后的数放入最小堆中（没有就不放），这样最大堆的根节点的值就总是黑匣子中第 i 大的数。接下来我们所要做的就是保证在执行 Add 和 Get 命令之后，两个堆依然保持原有性质。这样，Add 过程实际上是一个往堆中插入元素的过程，其时间复杂度为 $O(\log_2 M)$；Get 过程需要调整两个堆的大小（i 发生了变化），也需要进行堆的插入过程，其时间复杂度为 $O(\log_2 N)$。那么，整个程序的时间复杂度为 $O(M\log_2 M+N\log_2 N)$，我们认为这是一个 $O(n\log_2 n)$的算法，其效率是相当令人满意的，对于极限数据，都可以在 0.1 秒左右出解。

下面给出使用上面有关堆的算法中 Add 和 Get 过程的两个过程的源程序（整个程序参见附录）。

```
Procedure Add(num,i:LongInt);
 Var j,k,t:LongInt;
 Begin
  If top1<i Then   {如果最大堆中的数少于 i 个}
   Begin
    Insert1(num);  {向最大堆中添加数据 num}
    Exit   {退出}
   End;
  If num>=b1[1] Then Insert2(num) {如果待添加数据大于第 i 大的数（最大
                 堆的根节点 b1[1]）则将其插入至之后的最小堆中，否则将其插入最
                 大堆中}
                    Else Begin
                         Insert2(b1[1]); {把最大堆中最大的数（根节点）插入最
                                          小堆中}
                         b1[1]:=num;    {把最大堆的根节点替换为待添加数
                                         据，并进行维持最大堆性质的"筛"运
                                         算}
                         j:=1;k:=2;
                         While k<=top1 Do
                          Begin
```

```
                If (k<top1) and (b1[k]<b1[k+1]) Then Inc(k);
                If b1[j]<b1[k] Then
                  Begin
                   t:=b1[j];b1[j]:=b1[k];b1[k]:=t;
                   j:=k;k:=j*2
                  End
                 Else Break
               End
             End
  End;
 Procedure Get;
  Var j,k,t:LongInt;
  Begin
   Writeln(f,b1[1]);  {输出第 i 大的数（最大堆的根节点 b1[1]）}
   If top2<>0 Then  {如果最小堆不为空，则}
    Begin
    Insert1(b2[1]); {把最小堆中最小的数（最小堆的根节点 b2[1]）插入最大堆
中}
    b2[1]:=b2[top2]; {从最小堆中删除根节点}
    Dec(top2);
    j:=1;k:=2;
    While k<=top2 Do
```

```
Begin
 If (k<top2) and (b2[k]<b2[k+1]) Then Inc(k);
 If b2[j]<b2[k] Then
  Begin
   t:=b2[j];b2[j]:=b2[k];b2[k]:=t;
   j:=k;
   k:=j*2
  End
 Else Break
End
End
End;
```

下面给出一个表格，比较在不同数据量下程序 BlackBox1（采用线性表）和 BlackBox2（采用堆）的耗时。（测试机型为 PIII800，128MSD）

表 1

	数据量（N）	BlackBox1 的耗时(s)	BlackBox2 的耗时(s)
1	1000	0.00	0.00
2	2000	0.05	0.00
3	5000	0.33	0.00
4	10000	1.21	0.05
5	20000	5.05	0.05
6	30000	11.32	0.11

不难发现，采用堆的 BlackBox2 在时间复杂度上要明显优于采用线性表的 BlackBox1，足见堆的优越性。

二、排序二权树

同堆一样，排序二权树也是一种拥有特定性质的二权树。其定义为：

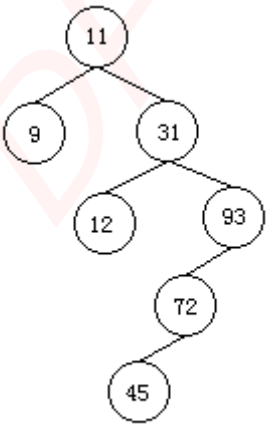


图3 一棵排序二权树

排序此二权树是一颗空树或具有下列性质的树：

(1)若其左子树非空，则左子树上锁有节点的值均小于根节点的值。

(2)若其右子树非空，则右子树上锁有节点的值均大于等于根节点的值。

(3)此二权树的左右子树均是一棵排序二权树。

不难发现，排序二权树是递归定义的，根据其定义，我们可以得到一些有关排序二权树的重要性质：

(1)一个排序二权树的任意一个节点的值都大于其左子节点的值而小于等于其右子节点的值。

(2)对一个排序二权树进行中序遍历，得到的序列是有序（递增）序列。

那么排序二权树到底有什么应用呢？显然，根据上面有关排序二权树的性质，我们可以利用排序二权树的有序性来进行排序：如果我们用一些数据生成了排序二权树，那么只要对其进行一次遍历，就可以得到有序数列。遍历的时间复杂度为 O(n)，由此，排序的效率是诱人的。

但是如何生成一个排序二权树呢？一个简单的方法就是一开始建立一棵空的排序二权树（根据定义，任何一棵空树都是排序二权树），然后不断往其中添加节点，最终构成一棵满足需要的排序二权树。现在，我们把问题转化为如何往一棵排序二权树中添加节点，使得新的树仍是一棵排序二权树。根据定义，在排序二权树中，一个节点的值如果比根节点小，那它就一定在其左子树中；而一个节点的值如果比根节点大，那它就一定在其右子树中。所以，我们要往一棵排序二权树中插入节点，如果此节点的值比根节点的值小，就把此节点插入此排序二权树的左子树中；如果此节点的值比根节点的值大，就把此节点插入此排序二权树的右子树中。由于定义是递归的，所以我们的插入算法也是递归的，以上面的方法不断的向下查找，直到找到一棵空树，就把这棵空树的根节点赋为待插入节点，从而完成了插入过程。因为插入过程的主要耗时在查找上，查找的次数总是不大于此排序二权树的深度，所以

插入过程的时间复杂度可以近似的认为是 $O(\log_2 n)$。下面给出插入过程的源程序：

```
Procedure Insert(Var T:Tree;x:LongInt);
  Begin
   If T=Nil Then  {如果找到一棵空树}
    Begin
     New(T);  {把此空树的根节点赋为待插入节点}
     T^.data:=x;T^.Left:=Nil;T^.Right:=Nil;
     Exit  {退出}
    End;
   If x>T^.data Then Insert(T^.Right,x) Else Insert(T^.Left,x) {如果节点的值比根节点大，
就将其插入至右子树中，否则插入左子树中}
  End;
```

上面这个递归过程实际上是一个末尾递归调用，因此它可以很轻松的改写为非递归程序：

```
Procedure Insert(Var T:Tree;x:LongInt);
  Var p,q:Tree;
  Begin
   If T=Nil Then
    Begin
     New(T);
     T^.Left:=Nil;
     T^.Right:=Nil;
     T^.Data:=x;
     Exit
    End;
   p:=T;q:=Nil; {p 指向当前节点，q 指向 p 的父节点}
   While p<>Nil Do {若未找到一颗空树，则}
    If x<p^.data Then {若待插入节点的值小于当前节点的值}
     Begin q:=p;p:=p^.Left End {就向其左子树查找}
    Else
     Begin q:=p;p:=p^.Right End; {否则向右子树查找}
   If x<q^.data Then {找到叶节点 q，若 x 大于 q 的值，就将其做为 q 的左子节点}
    Begin
     New(q^.Left);
```

```
        q^.Left^.data:=x;
        q^.Left^.Left:=Nil;
        q^.Left^.Right:=Nil
      End
    Else

      Begin  {否则将其做为 q 的右子节点}

        New(q^.Right);
        q^.Right^.data:=x;
        q^.Right^.Left:=Nil;
        q^.Right^.Right:=Nil
      End
  End;
```

这样，由于避免了递归调用，程序在时间和空间上要略优一点。

完成了插入算法之后，我们不难写出生成排序二权树的程序：

```
Procedure Create(Var T:Tree;a:Arr;n:LongInt);
 Var i:LongInt;
 Begin

   For i:=1 To n Do Insert(T,a[i]) {把每一个元数都插入排序二权树中}

 End;
```

这种生成排序二权树的算法的时间复杂度约为 $O(n\log_2 n)$，是相当优的了。

但是，利用排序二权树也存在一个缺点，就是如果插入的数据本身就是有序的，那么生成的将是一棵度为 1 的数，或者说就是一个线性表。那么排序二权树的插入过程的时间复杂度就降为 O(n)，而生成排序二权树的算法就相当于一个插入排序，时间复杂度为 $O(n^2)$。因此，利用排序二权树的排序与快速排序等算法一样，都是不稳定的。

下面给出一份表格，说明利用排序二权树排序(BST1)，经过非递归优化的利用排序二权树排序(BST2)，堆排序(HeapSort)和快速排序(QuickSort)的在随机数据和有序数据两种情况下的耗时(s)（测试环境为 PIII800,128MSD，打叉区域表示耗时大于 100s）：

表 2

序号	数据量(n)		BST1	BST2	HeapSort	QuickSort
1	10000	随机数据	0.11	0.11	0.11	0.11
		有序数据	2.64	0.99	0.11	0.60
2	50000	随机数据	0.49	0.49	0.44	0.44
		有序数据			0.49	14.51
3	100000	随机数据	1.15	1.04	1.04	0.88
		有序数据			0.88	
4	500000	随机数据	5.27	5.21	4.67	4.34
		有序数据			3.74	
5	1000000	随机数据	7.64	7.24	5.71	4.84
		有序数据			4.51	

可见，经过优化的 BST2 在时间复杂度上面要比 BST1 略优一点，但优势有限。虽然同为 $O(n\log_2 n)$级的算法，利用排序二权树的排序，堆排序和快速排序在时间复杂度上还是存在一定区别的。在随机测试数据下，快速排序体现了其“目前速度最快的排序方法”的资本，

展现了其的高效率。相比之下，堆排序差距不大，但利用排序二权树的排序就要逊色的多了。对于本身已经有序的测试数据，利用排序二权树的排序和快速排序都蜕变为 $O(n^2)$级算法，但堆排序则影响不大，甚至还要稍微快一点。这体现了堆排序的稳定性。

由此，我们可以认为：排序二权树是“排过序”的二权树而非“用于排序”的二权树。那么，排序二权树这样一种数据结构到底是用来干什么的呢？怎样才能充分发挥其优势呢？

不难发现，排序二权树这样一种数据结构的优势在于其的有序性，而且其插入和查找算法的时间复杂度为 $O(\log_2 n)$，这是其他任何一种数据结构没有办法达到的。（有序线性表虽然有序，但是插入算法的时间复杂度为 O(n)；堆的插入算法虽然时间复杂度为 $O(\log_2 n)$，但是堆并不具有有序性。）因此，我们要充分发挥排序二权树的优势，就要充分利用其的有序性和插入和查找算法的高效率。

因此，如果要经常对有序数列进行插入或查找工作，就可以采用排序二权树。

例如，在哈希表中，处理冲突元素用得最多的方法一般是链接法，即把发生冲突的所有元素用一个线性表串起来。但是如果冲突过多，这个线性表的长度过长，就会严重降低查找元素的效率。但是，如果把这些冲突的元素用一个排序二权树连接起来，查找元素的效率就会变得高得多。（时间复杂度由 O(n)降到 $O(\log_2 n)$，在 n 较大时相当可观。）

事实上，对于一些难以确定哈希函数的数列，可以直接用树表查找来代替散列表查找。即利用排序二权树来进行查找工作。我们看下面一个例子：

例 2：鸟语字典

Robin 是一只极其聪明的鸟，他着迷于人类丰富多彩的语言。经过长时间的摸索，Robin 模仿人类的英语创造了鸟类的语言。与英语类似，这种鸟语的基本单位（我们不妨也称其为字母）也是由 26 个小写英文字母 a 至 z 组成的。同时，若干个字母组成一个单词，用来表达一定的意思（和英语一样？！），相邻两个单词由一个空格隔开。Robin 为他新发明的鸟语创造了丰富的词汇，并花费大量精力写成一本鸟语字典。正如你所想的那样，Robin 想把一些英文的书籍（如《时间简史》、《物种起源》等）翻译成鸟语。但是，这项工作实在是太浩大了，以至于 Robin 无法完成。聪明的 Robin 想到使用计算机，他编写了一个自动翻译的程序来翻译这些书籍。但是很快，他发现，有很多词汇是他原先所没有想到的。（例如，《时间简史》中的“夸克”，厚厚的鸟语字典里并没有这个词。）对于这种情况，他的自动翻译程序将会不对其做翻译，而是直接放入译文中。下面是一个例子，表示自动翻译程序的执行过程：

字典：

序号	英文	鸟语
1	I	op
2	am	dg
3	a	a
4	bird	myself

英文：I am a clever bird.

译文：op dg a clever myself.

对于没有在字典中出现的单词 clever，自动翻译程序直接将其放入译文中。

现在，Robin 已经翻译了一些著作，他希望补充他的鸟语字典。因此，他想知道有多少单词在他的鸟语字典里是没有的，并且，他想统计出在这些没有出现在他的鸟语字典中的单词中，出现次数最多的单词是哪一个。

输入：

鸟语字典来自当前目录下的 Dic.txt。其中，第一行为一个正整数 n(1≤n≤10000)表示字典内的单词的数目。接下来的 n 行，每行有一个单词（没有多余空格）。字典中没有重复的单词。

输入数据来自当前目录下的 Input.txt。输入文件中是一段文本，由若干鸟语（英语）单词组成，相邻两个单词之间用至少一个空格隔开，文本中可能存在某些标点及其他符号。（文本中的单词数目不超过 10000。）

输出：

输出至当前目录下的 Output.txt。输出文件的第一行是一个整数 m，表示有 m 个单词没有在鸟语字典中出现。接下来的若干行每行一个单词，表示那个没有出现在鸟语字典当中，且出现次数最多的单词（有几个输出几个）。

样例：

Dic.txt:

3

ae

jd

opq

Input.txt

ae　jd . jda opq jda ae. ld jd opq!

Output.txt

2

jda

解：本题的实质是：给定两组字符串 A 和 B（字典和要翻译的文章），统计出在字符串组 B 中出现而在字符串组 A 中未出现的字符串的个数，并输出这些字符串中在字符串组 B 中出现次数最多的一个（或多个）。

了解了本题的实质之后，可以轻松得到本题的大致算法：在字符串组 A 中查找每一个出现在字符串组 B 的字符串，若没有找到，就记录下这个字符串及其出现次数。最后，就可以统计出满足上述条件的字符串的个数及出现次数最多的字符串了。

注意到程序的主要耗时花费在字符串组 A 中查找字符串，所以，提高在字符串组 A 中查找字符串的效率尤为重要。可惜的是，似乎没有什么好的哈希函数可以应付字符串，所以最有效的散列表不幸的不能被使用。因此，解决效率问题的关键在于数据结构的选择。

我们仍然可以朴素的采用线性表来解决：时刻保持字符串组 A 中的字符串的有序

性（对字符串进行比较大小还是比较容易办到的），使用二分查找，从而把查找字符串的时间效率限制在 $O(\log_2 n)$的等级上。但是，值得注意的是，我们还是经常要对字符串组 A 进行插入操作的：一开始生成有序的字符串组不说，为了正确的统计没有在字符串组 A 中出现的字符串，也需要在适当的时候将这些字符串添加进入字符串组 A（当然，我们要对这些新的字符串坐上记号，以区别于其他的字符串）。对有序表的插入算法的时间复杂度为 O(n)。由于 n 比较大，这样的数量级就已经令人头疼了。我们还需要采用更为有效的算法。

更为有效的算法基于更为有效的数据结构：排序二权树。我们完全可以把字符串组 A 中的字符串插入一个排序二权树中，下面所需要做的只是对这个排序二权树进行查找和添加工作。排序二权树的查找效率为 $O(\log_2 n)$，而其插入效率也为 $O(\log_2 n)$。由于 n 比较大，n 和 $\log_2 n$ 的差距很大，这种算法的效率较前一种相比有了较大的提高。

还是通过列表来比较两种算法的效率（测试环境为 PIII800,128MSD）：

表 3

序号	数据量(n)	基于线性表的算法(BDic1)	基于排序二权树的算法(BDic2)
1	1000	1.15s	0.27s
2	2000	4.12s	0.44s
3	5000	18.85s	0.88s
4	10000		1.98s

可以看出，虽然只是由 O(n)降到 $O(\log_2 n)$，效果却是明显的。

其实，本题还有再改进的余地：排序二权树的缺点就是对数据没有选择性，它的效率在很大程度上依赖于插入的顺序，如果输入的数据本身就是有序的，排序二权树将不幸的蜕变为线性表。因此，如果在对排序二权树进行插入的过程中，时刻注意保持二权

树的平衡性，也可以有效的提高算法的效率。至于具体如何保持二权树的平衡性，由于篇幅限制，这里不多加讨论，有兴趣的同学可以翻翻《数据结构简明教程》有关树表查找和平衡树的内容。

参考书目

1．徐孝凯　《数据结构简明教程》　北京：清华大学出版社，1994
2．王建德　《高中计算机竞赛解题指导》　西安：陕西师范大学出版社，2001

附录（源程序）

1. 堆排序

```
Program PHeap(Input,Output);
 Const maxn=1000000;
 Type Heap=Array[1..maxn] of LongInt;
 Var H:Heap;
     t,i,n:LongInt;
     f:Text;
 Procedure Sift(n,i:LongInt);
  Var j,x:LongInt;
  Begin
   x:=H[i];
   j:=2*i;
   While j<=n Do
    Begin
     If (j<n) and (H[j]<H[j+1]) Then Inc(j);
     If x<H[j] Then
      Begin
       H[i]:=H[j];
       i:=j;j:=i*2
      End
     Else Break
    End;
   H[i]:=x
  End;
 Begin
  Assign(f,'Data.in');
  Reset(f);
  Readln(f,n);
  For i:=1 To n Do Read(f,H[i]);
  Close(f);
  For i:=n Div 2 DownTo 1 Do Sift(n,i);
  For i:=n DownTo 2 Do
   Begin
    t:=H[1];H[1]:=H[i];H[i]:=t;
    Sift(i-1,1)
   End;
  Assign(f,'Data.out');
  ReWrite(f);
  For i:=1 To n Do Write(f,H[i],' ');
  Writeln(f);
  Close(f)
 End.
```

返回

2. BlackBox1

```
Program BlackBox(Input,Output);
 Const maxn=30000;
 Type Arr=Array[1..maxn] of LongInt;
 Var a,u,b:Arr;
     i,j,M,N,top,Time:LongInt;
     f:Text;
 Procedure Add(num:LongInt);
  Var i:LongInt;
  Begin
   i:=top;
   While (i>=1) and (b[i]>num) Do
    Begin
     b[i+1]:=b[i];
     Dec(i)
    End;
   b[i+1]:=num;
   Inc(top)
  End;
 Procedure Get(i:LongInt);
  Begin
   Writeln(f,b[i])
  End;
 Begin
  Time:=Meml[$40:$6c];
  Assign(f,'BlackBox.in');
  Reset(f);
  Read(f,M,N);
  For i:=1 To M Do Read(f,a[i]);
  For i:=1 To N Do
   Read(f,u[i]);
  Close(f);
  i:=1;top:=0;
  Assign(f,'BlackBox.out');
  ReWrite(f);
  For j:=1 To M Do
   Begin
    Add(a[j]);
    While (i<=N) and (u[i]=j) Do
     Begin
      Get(i);
      i:=i+1
     End
```

```
   End;
  Writeln('Time=',(Meml[$40:$6c]-Time)/18.2:0:10);
  Close(f)
 End.
```

返回

3. BlackBox2

```
Program BlackBox(Input,Output);
 Const maxn=30000;
 Type Arr=Array[1..maxn] of LongInt;
 Var a,u,b1,b2:Arr;
     i,j,M,N,top1,top2,Time:LongInt;
     f:Text;
 Procedure Insert1(num:LongInt);
  Var p,t:LongInt;
  Begin
   Inc(top1);
   b1[top1]:=num;
   p:=top1;
   While (p>1) and (b1[p Div 2]<b1[p]) Do
    Begin
     t:=b1[p Div 2];b1[p Div 2]:=b1[p];b1[p]:=t;
     p:=p Div 2
    End
  End;
 Procedure Insert2(num:LongInt);
  Var p,t:LongInt;
  Begin
   Inc(top2);
   b2[top2]:=num;
   p:=top2;
   While (p>1) and (b2[p Div 2]>b2[p]) Do
    Begin
     t:=b2[p Div 2];b2[p Div 2]:=b2[p];b2[p]:=t;
     p:=p Div 2
    End
  End;
 Procedure Add(num,i:LongInt);
  Var j,k,t:LongInt;
  Begin
   If top1<i Then
    Begin
     Insert1(num);
     Exit
    End;
```

```
  If num>=b1[1] Then Insert2(num)
                Else Begin
                       Insert2(b1[1]);
                       b1[1]:=num;
                       j:=1;k:=2;
                       While k<=top1 Do
                        Begin
                         If (k<top1) and (b1[k]<b1[k+1]) Then Inc(k);
                         If b1[j]<b1[k] Then
                          Begin
                           t:=b1[j];b1[j]:=b1[k];b1[k]:=t;
                           j:=k;
                           k:=j*2
                          End
                         Else Break
                        End
                      End
 End;
Procedure Get;
 Var j,k,t:LongInt;
 Begin
  Writeln(f,b1[1]);
  If top2<>0 Then
   Begin
    Insert1(b2[1]);
    b2[1]:=b2[top2];
    Dec(top2);
    j:=1;k:=2;
    While k<=top2 Do
     Begin
      If (k<top2) and (b2[k]<b2[k+1]) Then Inc(k);
      If b2[j]<b2[k] Then
       Begin
        t:=b2[j];b2[j]:=b2[k];b2[k]:=t;
        j:=k;
        k:=j*2
       End
      Else Break
     End
   End
 End;
Begin
 Time:=Meml[$40:$6c];
 Assign(f,'BlackBox.in');
```

```
  Reset(f);
  Read(f,M,N);
  For i:=1 To M Do Read(f,a[i]);
  For i:=1 To N Do
   Read(f,u[i]);
  Close(f);
  i:=1;top1:=0;top2:=0;
  Assign(f,'BlackBox.out');
  ReWrite(f);
  For j:=1 To M Do
   Begin
    Add(a[j],i);
    While (i<=N) and (u[i]=j) Do
     Begin
      Get;
      Inc(i)
     End
   End;
  Writeln('Time=',(Meml[$40:$6c]-Time)/18.2:0:10);
  Close(f)
 End.
```

返回

4. BST1

```
Program Binary_Sort_Tree(Input,Output);
 Type Tree=^Node;
      Node=Record
             data:LongInt;
             Left,Right:Tree
            End;
 Var T:Tree;
     i,n,x,Time:LongInt;
     f:Text;
 Procedure Insert(Var T:Tree;x:LongInt);
  Begin

   If T=Nil Then
    Begin
     New(T);
     T^.data:=x;
     T^.Left:=Nil;
     T^.Right:=Nil;
     Exit
    End;
   If x>T^.data Then Insert(T^.Right,x) Else Insert(T^.Left,x)
```

```
  End;
 Procedure Print(T:Tree);
  Begin
   If T=Nil Then Exit;
   Print(T^.Left);
   Write(f,T^.data,' ');
   Print(T^.Right);
   Dispose(T)
  End;
 Begin
  Time:=Meml[$40:$6c];
  Assign(f,'Data.in');
  Reset(f);
  Readln(f,n);
  T:=Nil;
  For i:=1 To n Do
   Begin
    Read(f,x);
    Insert(T,x)
   End;
  Close(f);
  Assign(f,'Data.out');
  ReWrite(f);
  Print(T);
  Close(f);
  Writeln('Time=',(Meml[$40:$6c]-Time)/18.2:0:10)
 End.
```

返回

5. BST2

```
Program Binary_Sort_Tree(Input,Output);
 Type Tree=^Node;
      Node=Record
             data:LongInt;
             Left,Right:Tree
           End;
 Var T:Tree;
     i,n,x,Time:LongInt;
     f:Text;
 Procedure Insert(Var T:Tree;x:LongInt);
  Var p,q:Tree;
  Begin
   If T=Nil Then
    Begin
     New(T);
```

```
     T^.Left:=Nil;T^.Right:=Nil;T^.Data:=x;
     Exit
    End;
   p:=T;
   q:=Nil;
   While p<>Nil Do
    If x<p^.data Then
     Begin
      q:=p;p:=p^.Left
     End
    Else
     Begin
      q:=p;p:=p^.Right
     End;
   If x<q^.data Then
    Begin
     New(q^.Left);
     q^.Left^.data:=x;q^.Left^.Left:=Nil;q^.Left^.Right:=Nil
    End
   Else
    Begin
     New(q^.Right);
     q^.Right^.data:=x;q^.Right^.Left:=Nil;q^.Right^.Right:=Nil
    End
 End;
Procedure Print(T:Tree);
 Begin
  If T=Nil Then Exit;
  Print(T^.Left);
  Write(f,T^.data,' ');
  Print(T^.Right);
  Dispose(T)
 End;
Begin
 Time:=Meml[$40:$6c];
 Assign(f,'Data2.in');
 Reset(f);
 Readln(f,n);
 T:=Nil;
 For i:=1 To n Do
  Begin
   Read(f,x);
   Insert(T,x)
  End;
```

```
  Close(f);
  Assign(f,'Data.out');
  ReWrite(f);
  Print(T);
  Close(f);
  Writeln('Time=',(Meml[$40:$6c]-Time)/18.2:0:10)
 End.
```

返回

6. 快速排序

```
Program QuickSort(Input,Output);
 Const maxn=1000000;
 Type Arr=Array[1..maxn] of LongInt;
 Var A:Arr;
     i,n,Time:LongInt;
     f:Text;
 Procedure Sort(s,t:LongInt);
  Var i,j,x:LongInt;
  Begin
   i:=s;j:=t;x:=A[s];
   While i<j Do
    Begin
     While (A[j]>x) and (j>i) Do Dec(j);
     If j>i Then Begin A[i]:=A[j];Inc(i) End;
     While (A[i]<x) and (i<j) Do Inc(i);
     If i<j Then Begin A[j]:=A[i];Dec(j) End
    End;
   A[i]:=x;
   If s<(i-1) Then Sort(s,i-1);
   If (i+1)<t Then Sort(i+1,t)
  End;
 Begin
  Time:=Meml[$40:$6c];
  Assign(f,'Data.in');
  Reset(f);Readln(f,n);
  For i:=1 To n Do Read(f,A[i]);
  Close(f);
  Sort(1,n);
  Assign(f,'Data.out');ReWrite(f);
  For i:=1 To n Do Write(f,A[i],' ');
  Writeln(f);
  Close(f);
  Writeln('Time=',(Meml[$40:$6c]-Time)/18.2:0:10)
 End.
```

返回

7.BDic1

```
Program Bird_Dic(Input,Output);
 Type Tree=^Node;
      Node=Record
             data:String;
             cs:LongInt;
             Left,Right:Tree
            End;
 Var T,p:Tree;
     ch:Char;
     str:String;
     n,i,num,max,Time:LongInt;
     Buf:Array[1..65535] of Byte;
     f:Text;
 Function LessThan(s1,s2:String):Boolean;
  Var i:Byte;
  Begin
   If Length(s1)<Length(s2) Then Exit(True);
   If Length(s1)>Length(s2) Then Exit(False);
   LessThan:=True;
   For i:=1 To Length(s1) Do
    If s1[i]>s2[i] Then Exit(False)
  End;
 Procedure Insert(Var T:Tree;p:Tree);
  Begin
   If T=Nil Then
    Begin
     T:=p;
     Exit
    End;
   If LessThan(p^.data,T^.data) Then Insert(T^.Left,p) Else Insert(T^.Right,p)
  End;
 Function Find(T:Tree;s:String):Boolean;
  Begin
   If T=Nil Then Exit(False);
   If T^.data=s Then
    Begin
     If T^.cs<>0 Then Inc(T^.cs);
     Exit(True)
    End;
   If LessThan(s,T^.data) Then Find:=Find(T^.Left,s) Else Find:=Find(T^.Right,s)
  End;
 Procedure LoadMax(T:Tree);
  Begin
```

```
  If T=Nil Then Exit;
  If T^.cs>max Then max:=T^.cs;
  LoadMax(T^.Left);
  LoadMax(T^.Right)
 End;
Procedure Print(T:Tree);
 Begin
  If T=Nil Then Exit;
  If T^.cs=max Then Writeln(f,T^.data);
  Print(T^.Left);
  Print(T^.Right)
 End;
Begin
 Time:=Meml[$40:$6c];
 T:=Nil;
 Assign(f,'Dic.txt');
 Reset(f);
 SetTextBuf(f,Buf);
 Readln(f,n);
 For i:=1 To n Do
  Begin
   Readln(f,str);
   New(p);
   p^.data:=str;
   p^.cs:=0;
   p^.Left:=Nil;p^.Right:=Nil;
   Insert(T,p)
  End;
 Close(f);
 Assign(f,'Input.txt');
 Reset(f);
 SetTextBuf(f,Buf);
 str:='';num:=0;
 While Not Eof(f) Do
  Begin
   Read(f,ch);
   Case ch of
    'a'..'z':str:=str+ch;
    ' ':If str<>'' Then
          Begin
           If Not Find(T,str) Then
            Begin
             Inc(num);
             New(p);
```

```
              p^.data:=str;
              p^.cs:=1;
              p^.Left:=Nil;p^.Right:=Nil;
              Insert(T,p)
             End;
            str:=''
           End
    End
   End;
  Close(f);
  Assign(f,'Output.txt');
  ReWrite(f);
  Writeln(f,num);
  If num<>0 Then
   Begin
    max:=1;
    LoadMax(T);
    Print(T)
   End;
  Close(f);
  Writeln('Time=',(Meml[$40:$6c]-Time)/18.2:0:10)
 End.
```

返回

8.BDic2

```
Program Bird_Dic(Input,Output);
 Type Node=Record
            data:String;
            cs:LongInt
           End;
      Arr=Array[1..10000] of Node;
 Var T:Arr;
     p:Node;
     ch:Char;
     str:String;
     n,i,num,max,Time,topT:LongInt;
     f:Text;
 Function LessThan(s1,s2:String):Boolean;
  Var i:Byte;
  Begin
   If Length(s1)<Length(s2) Then Exit(True);
   If Length(s1)>Length(s2) Then Exit(False);
   LessThan:=True;
   For i:=1 To Length(s1) Do
    If s1[i]>s2[i] Then Exit(False)
```

```
  End;
 Procedure Insert(Var T:Arr;Var topT:LongInt;p:Node);
  Var i:LongInt;
  Begin
   i:=topT;
   While (i>=1) and LessThan(p.data,T[i].data) Do Begin T[i+1]:=T[i];Dec(i) End;
   T[i+1]:=p;
   Inc(topT)
  End;
 Function Find(foot,top:LongInt;s:String):Boolean;
  Begin
   If s=T[(foot+top) Div 2].data Then
    Begin
     If T[(foot+top) Div 2].cs<>0 Then Inc(T[(foot+top) Div 2].cs);
     Exit(True)
    End;
   If foot>=top Then Exit(False);
   If LessThan(s,T[(foot+top) Div 2].data) Then Find:=Find(foot,(foot+top) Div
2-1,s)
                                        Else   Find:=Find((foot+top)    Div
2+1,top,s)
  End;
 Procedure LoadMax;
  Var i:LongInt;
  Begin
   For i:=1 To topT Do
    If T[i].cs>max Then max:=T[i].cs
  End;
 Procedure Print;
  Var i:LongInt;
  Begin
   For i:=1 To topT Do
    If T[i].cs=max Then Writeln(f,T[i].data)
  End;
 Begin
  Time:=Meml[$40:$6c];
  topT:=0;
  Assign(f,'Dic.txt');
  Reset(f);
  Readln(f,n);
  For i:=1 To n Do
   Begin
    Readln(f,str);
```

```
    p.data:=str;
    p.cs:=0;
    Insert(T,topT,p)
   End;
  Close(f);
  Assign(f,'Input.txt');
  Reset(f);
  str:='';num:=0;
  While Not Eof(f) Do
   Begin
    Read(f,ch);
    Case ch of
     'a'..'z':str:=str+ch;
     ' ':If str<>'' Then
           Begin
            If Not Find(1,topT,str) Then
             Begin
              Inc(num);
              p.data:=str;
              p.cs:=1;
              Insert(T,topT,p)
             End;
            str:=''
           End
    End
   End;
  Close(f);
  Assign(f,'Output.txt');
  ReWrite(f);
  Writeln(f,num);
  If num<>0 Then
   Begin
    max:=1;
    LoadMax;
    Print
   End;
  Close(f);
  Writeln('Time=',(Meml[$40:$6c]-Time)/18.2:0:10)
 End.
```

返回

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用
33. 高可用的磁盘阵列 Cache 的设计和实现
34. RAID 阵列中高速 Cache 管理的优化

35. 一种新的基于 RAID 的 CACHE 技术研究与实现
36. 基于 PCIE-104 总线的高速数据接口设计
37. 基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计
38. 北斗卫星系统在海洋工程中的应用
39. 北斗卫星系统在远洋船舶上应用的研究
40. 基于 CPCI 总线的红外实时信号处理系统
41. 硬件实现 RAID 与软件实现 RAID 的比较
42. 基于 PCI Express 总线系统的热插拔设计
43. 基于 RAID5 的磁盘阵列 Cache 的研究与实现

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发
23. Vxworks 下的冗余 CAN 通讯模块设计
24. WindML 工业平台下开发 S1d13506 驱动及显示功能的实现

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings
25. Android 应用逆向分析方法研究
26. Android 操作系统的课程教学
27. Android 智能手机操作系统的研究
28. Android 英文朗读功能的实现
29. 基于 Yocto 订制嵌入式 Linux 发行版
30. 基于嵌入式 Linux 的网络设备驱动设计与实现
31. 如何高效学习嵌入式
32. 基于 Android 平台的 GPS 定位系统的设计与实现

Windows CE:

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究

4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究
17. Windows CE 环境下无线网卡的自动安装
18. 基于 Windows CE 的可视电话的研究与实现

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植
12. 基于 PowerPC 双核处理器嵌入式系统 UBoot 移植
13. 基于 PowerPC 的雷达通用处理机设计
14. PowerPC 平台引导加载程序的移植

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现
16. 基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计
17. ARM S3C2440 Linux ADC 驱动
18. ARM S3C2440 Linux 触摸屏驱动
19. Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计
20. Nand Flash 启动模式下的 Uboot 移植
21. 基于 ARM 处理器的 UART 设计

Hardware:

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究
14. MIPS 架构计算机平台的支持固件研究
15. 基于 UEFI 固件的攻击验证技术研究

16. 基于 UEFI 的 Application 和 Driver 的分析与开发

Programming:

1. 计算机软件基础数据结构 - 算法