文章编号:1007-1423(2021)14-0096-04 DOI:10.3969/j.issn.1007-1423.2021.14.019

哈夫曼树在排序算法中的案例教学研究

吐尔地·托合提,崔青,刘淑娴

(新疆大学信息科学与工程学院,乌鲁木齐 830046)

摘要:

本文对于哈夫曼树的应用进行拓展,提出一种基于哈夫曼树的内部排序算法。通过一个综合应用哈夫曼树、栈和队列的教学案例,一方面,加深学生对几种重要数据结构在算法设计中作用的认识,另一方面,鼓励学生敢于创新,引导学生从知识中体会和掌握算法设计的思维方式和技巧,从而培养学生创造性思维能力及解决实际问题的能力。

关键词:

数据结构; 哈夫曼树; 队列; 栈; 排序

基金项目:

新疆维吾尔自治区高等学校本科教育教学改革研究综合改革项目(No.2018JG09);新疆大学金课建设项目(No.XJU2020JK34)

0 引言

哈夫曼树(Huffman Tree),又被称为最优二叉树,是一种带权路径长度最小的二叉树,其应用也很广泛,如基于哈夫曼树的编码[1]、压缩[2-3]、加密[4-5]、数据采样[6]等。在教学中,一般我们重点讲解带权路径长度的概念、哈夫曼树的性质、哈夫曼树的构造,以及哈夫曼编码[8]。其实,哈夫曼树也可以用来对于数据元素序列进行排序。

从哈夫曼树的性质可知,权重越大的叶子结点越靠近根结点,而权重越小的叶子结点离根结点越远。因此,将待排序序列中的每一个元素看成一个叶子结点,将元素关键码看成其权重(关键码为整形情况下),构造一棵哈夫曼树,然后层序遍历并依次输出叶子结点,就能得到基本有序(降序)序列。如果,在哈夫曼树的构造、遍历和输出过程中增设一些约束,那么也完全可以做到对数据元素的降序或升序排序。

本文在数据结构教学中,针对内部排序提出一个教学案例,在排序方法中引入哈夫曼树、二叉树的遍历、栈和队列等综合内容,通过讲解一些启发思路的知识点,一方面加深了学生对于哈夫曼树的性质、构造方法、二叉树遍历算法,以及哈夫曼树、栈和队列应用的理解和掌握,另一方面引导学生要认识到解决问题有多种方案可选,鼓励学生从知识中体会和掌握算法设计的思维方式和技巧,这有助于使学生分析问题和解决问题的能力得到提升。

1 基于哈夫曼树的排序算法思路

在待排序序列中数据元素关键码为整形(int 形)情况下,我们可以将每一个数据元素看成一个叶子结点,将数据元素关键码看成其权重构造一棵哈夫曼树。设待排序数据元素关键字序列为{5,9,15,30,18,27,10,13},在构造哈夫曼树的过程中我们给一个约束:将权重大的结点作为左孩子,将权重小的结点作为右孩子,然后按照哈夫曼树的构造方法就能生成得到如图 1 所示的一棵哈夫曼树。

我们仔细观察图 1 哈夫曼树中叶子结点(待排序元素结点)分布,第一层为根,从第二层开始,发现处在第 i 层上元素关键字均小于第 i-1 层元素关键字,而均大于第 i+1 层元素关键字,最底层上的元素关键字是最小。再观察同一个层上的元素关键字大小,从左到右,关键字大小依次变小。因此,我们对此哈夫曼树进

行层序遍历并输出叶子结点关键字，就能得到一个关键字有序（降序）序列{30，27，18，15，13，10，9，5}，这样就可以做到降序排序。如果，需要进行升序排序，我们可以借助一个栈来暂存层序遍历输出序列，最后依次出栈就能得到升序排序结果。

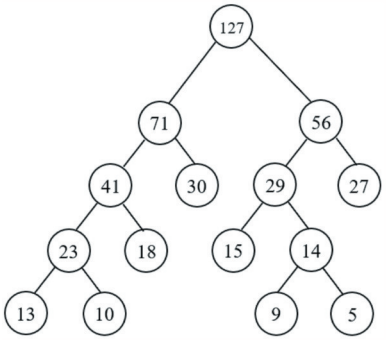

图1 由待排序元素关键字构建的哈夫曼树

2 基于哈夫曼树的排序算法实现

根据以上分析，我们可以设计一套用来排序的哈夫曼树构造算法和排序算法。为了简单讨论，以下假设关键码为整形，而且只考虑关键码，暂不考虑数据元素其他数据项。

构造哈夫曼树时，常用一个结构数组（如 Huff_Nodes）来存储哈夫曼树中每一个结点的信息。我们从二叉树的性质可知，由 n 个叶子结点构建的哈夫曼树中总共有 2n-1 个结点，所以结构数组 Huff_Nodes 大小可设置为 2n-1，其元素结构可定义如图 2 所示形式。

key	parent	l_child	r_child

图2 结构数组元素结构形式

其中，key 域是用来保存待排序数据元素关键字（传统算法中的结点权值 weight），l_child 域和 r_child 域是分别用来保存该结点的左孩子和右孩子结点在数组中的下标（序号），算法中我们通过下标在结点之间建立联系。parent 域保存当前结点在哈夫曼树中的双亲结点在数组中的序号，其初始值为-1。

2.1 哈夫曼树的构造算法

开始构造哈夫曼树之前，首先把 n 个关键字形成的 n 个叶子结点存放在结构数组 Huff_Nodes 的前 n 个分量中，然后根据哈夫曼树构造基本思想和本文给出的约束条件，不断将两个根结点权重最小的子树合并为一个较大的子树，并把被生成的新子树的根结点依次按顺序存放到 Huff_Nodes 数组前 n 个分量后面。具体实现算法描述如下：

```
#define MAXKEYVAL INT_MAX //定义关键字最大值∞
#define MAXLEAFNUM 50 //定义哈夫曼树叶子结点个数
#define MAXNODENUM MAXLEAFNUM*2-1 //定义哈夫曼树结点个数
typedef struct { //定义哈夫曼树结点结构
 int key, parent, l_child, r_child;
} HTNode;
void Huff_Tree(HTNode Huff_Nodes [], int n) { //哈夫曼树的构造函数
 int i,j, min1, min2,index1,index2;
 for (i=0;i<2*n-1;i++) //初始化 Huff_Nodes[]{
  Huff_Nodes [i].key=0;
  Huff_Nodes [i].parent=-1;
  Huff_Nodes [i].l_child=-1;
  Huff_Nodes [i].r_child=-1;
 }
 for (i=0;i<n;i++)
  scanf("%d",&Huff_Nodes[i].key); //输入 n 个叶子结点的关键字
 for (i=0;i<n-1;i++) //开始构造哈夫曼树{
  min1=min2= MAXKEYVAL;
  index1=index2=-1;
  for (j=0;j<n+i;j++){
   if (Huff_Nodes [j].key<min1 && Huff_Nodes [j].parent==-1){// 寻找还未加入哈夫曼树及 key 值最小的两个结点
    min2=min1; //用 min1、min2 来存储两个最小 key 值
    index2=index1; //用 index1、index2 来存储两个最小 key 值对应的结点在数组 Huff_Nodes 中的序号
    min1=Huff_Nodes [j].key;
    index1=j;
   }
   else if (Huff_Nodes [j].key<min2 && Huff_Nodes [j].parent==-1){
    min2=Huff_Nodes [j].key;
    index2=j;
   }
  }
```

```
  //将找出的两棵子树合并为一棵子树,key 值大的作为左子树,小的作为右子树
   Huff_Nodes [n+i].key= Huff_Nodes [index1].key+Huff_Nodes [index2].key;
   if(Huff_Nodes [index1].key>Huff_Nodes [index2].key) {
    Huff_Nodes [n+i].l_child=index1;
    Huff_Nodes [n+i].r_child=index2;
   }
   else{
    Huff_Nodes [n+i].l_child=index2;
    Huff_Nodes [n+i].r_child=index1;
   }
   Huff_Nodes [index1].parent=n+i;
   Huff_Nodes [index2].parent=n+i;
  }
}
```

算法初始化 Huff_Nodes 数组并依次向数组填写新生成的子树信息(子树根结点关键字、根结点左右孩子在数组中的序号),同时,还不断修改与已有子树之间的关联信息。当算法结束时,用来存储图 2 所示哈夫曼树的结构数组 Huff_Nodes 的最终状态如图 3 所示。

	key	parent	l_child	r_child
0	30	13	-1	-1
1	27	12	-1	-1
2	18	11	-1	-1
3	15	10	-1	-1
4	13	9	-1	-1
5	10	9	-1	-1
6	9	8	-1	-1
7	5	8	-1	-1
8	14	10	6	7
9	23	11	4	5
10	29	12	3	8
11	41	13	9	2
12	56	14	10	1
13	71	14	11	0
14	127	-1	13	12

图3 数组 Huff_Nodes 的最终状态

2.2 基于哈夫曼树层序遍历的排序算法

已构建好的哈夫曼树的根结点在数组 Huff_Nodes 的最后一个单元中,我们从根结点出发进行层序遍历(从上到下,从左到右),并把那些叶子结点依次暂存到一个栈中,等遍历结束就出栈所有元素,此时的出栈序列就是一个升序排序序列。如果要进行降序排序,那就不需要经过栈来改变次序,在遍历中直接输出叶子结点即可。升序排序具体实现算法描述如下:

```
typedef struct { //顺序队列结构定义
 HTNode data[MAXNODENUM];
 int front, rear;
} SqQueue;
typedef struct { //顺序栈结构定义
 int data[MAXLEAFNUM];
 int top;
} SqStack;
void HuffSort( ) //基于哈夫曼树的排序算法{
 HTNode Huff_Nodes [MAXNODENUM];
 SqQueue Q; //定义及初始化顺序(循环)队列
 Q.front=Q.rear=-1;
 SqStack S; //定义及初始化顺序栈
 S.top=-1;
 int i,j,c,p,n;
 scanf("%d",&n); //输入叶子结点个数
 Huff_Tree (Huff_Nodes, n); //调用 Huff_Nodes 函数建立哈夫曼树
 Q.rear=(Q.rear+1)% MAXNODENUM;
 Q.data[Q.rear]= Huff_Nodes [2*n-2]; //头结点入队
 while(Q.rear!=Q.front) {
  Q.front=(Q.front+1)% MAXNODENUM; //队头元素出队
  if (Q.data[Q.front].l_chalid==- 1&& Q.data[Q.front].r_chalid==-1) {
   //如出队元素为叶子结点,则其关键字入栈
   S.top++;
   S.data[S.top]= Q.data[Q.front].key;
  }
  else if(Q.data[Q.front].l_chalid!=-1) {
   //出队元素(结点)左孩子入队
   Q.rear=(Q.rear+1)% MAXNODENUM;
   Q.data[Q.rear]= Huff_Nodes [Q.data[Q.front].l_chalid];
  }
  else{
   //出队元素(结点)右孩子入队
   Q.rear=(Q.rear+1)% MAXNODENUM;
   Q.data[Q.rear]= Huff_Nodes [Q.data[Q.front].r_chalid];
  }
 }
```

```
 while (S.top!=-1) //出栈输出关键字升序排序序列 {
   printf("%d",S.data[S.top]);
   S.top--;
 }
}
```

3 结语

排序是数据结构课程最后一章内容，一般我们主要讲解各种典型内部排序算法思路，分析其性能和优缺点，然后要求学生通过运行、调试排序程序去掌握多种排序算法，但很少有拓展内容。本文针对内部排序设计一个教学案例，将栈、队列、哈夫曼树等重要内容综合应用到排序方法中。其目的不仅仅是提出一个排序算法，而更重要的是鼓励学生敢于创新，引导学生从知识中体会和掌握算法设计的思维方式和技巧，从而培养学生创造性思维能力及解决实际问题的能力。

参考文献：

[1]王彩霞.《数据结构》课程中哈夫曼编码教学案例研究[J]. 高教学刊，2020(31)：104-106.
[2]乔雨，嵇浩. 一种基于缓冲窗口的双哈夫曼压缩算法[J]. 物联网技术，2021，11(02)：90-94.
[3]吕姣霖，徐艳. 哈夫曼编码在图像压缩中的应用与分析[J]. 数字通信世界，2021(01)：189-190.
[4]秦飞舟，庄红. 基于 Huffman 算法的数据加密[J]. 电脑知识与技术(学术交流)，2007(19)：180-182.
[5]冯蕾，彭长根，彭延国. 基于哈夫曼树的无证书公钥广播加密方案[J]. 计算机工程与应用，2012，48(24)：85-87+181.
[6]彭永供，邱桃荣，林于渊，等. 基于哈夫曼树的雷电数据采样算法[J]. 计算机工程，2013，39(05)：174-177+182.
[7]朱昌杰，肖建于. 数据结构(C 语言班)[M].2 版 北京：清华大学出版社，2014.

作者简介：

吐尔地·托合提(1975-)男，博士，副教授，硕士生导师，研究方向为自然语言处理及文本挖掘
崔青(1973-)女，硕士，副教授，研究方向为大数据可视化分析
刘淑娴(1978-)女，硕士，副教授，硕士生导师，研究方向为网络空间安全及机器学习
收稿日期：2021-03-11　　修稿日期：2021-03-26

A Case Study of Huffman Tree in Sorting Algorithm

TURDI Tohti, CUI Qing, LIU Shuxian

(College of Information Science and Engineering, Xinjiang University, Urumqi 830046)

Abstract:

This paper expands the application of Huffman tree and proposes an internal sorting algorithm based on Huffman tree. Through a teaching case of comprehensive application of Huffman tree, Stack and Queue, on the one hand, it deepens students´ understanding of the role of several important data structures in algorithm design, on the other hand, it encourages students to dare to innovate, guides students to experience and master the thinking mode and skills of algorithm design from knowledge, so as to cultivate students´ creative thinking ability and ability to solve practical problems.

Keywords:

Data Structure; Huffman Tree; Queue; Stack; Sorting