

《数据结构》实验报告

班 级: _____

姓 名: _____

学 号: _____

E-mail: _____

日 期: _____

◎实验题目:

3.1 哈夫曼编/译码器

Description

写一个哈夫曼码的编/译码系统, 要求能对要传输的报文进行编码和解码。构造哈夫曼树时, 权值小的放左子树, 权值大的放右子树, 编码时右子树编码为1, 左子树编码为0。

◎实验内容:

3.1

Input

输入表示字符集大小为 n ($n \leq 100$) 的正整数, 以及 n 个字符和 n 个权值 (正整数, 值越大表示该字符出现的概率越大); 输入串长小于或等于100的目标报文。

Output

经过编码后的二进制码, 占一行;
以及对应解码后的报文, 占一行;
最后输出一个回车符

一、需求分析

本次实验的主要目标是实现哈夫曼编码和解码的功能。输入的数据是一组字符及其出现的频率, 输出的结果是经过哈夫曼编码的二进制码序列, 以及对该二进制码序列的解码结果。

二、概要设计

为了实现哈夫曼编码和解码的功能, 首先需要构建一个哈夫曼树。哈夫曼树是一种特殊的二叉树, 其特点是树的带权路径长度最短。在构建哈夫曼树的过程中, 需要反复地选取权值最小的两个节点, 然后合并这两个节点, 并将它们的父节点添加到哈夫曼树中。根据构建的哈夫曼树, 就可以生成哈夫曼编码。最后, 根据哈夫曼编码, 可以进行编码和解码的操作。

三、详细设计

数据结构设计:

本次实验的数据结构设计中, 使用了两个关键的结构体: HTNode 和 HCodeNode。

HTNode 是用来表示哈夫曼树节点的结构体, 其中包含五个字段: 权值(weight), 父节点(parent), 左孩子(lchild), 右孩子(rchild) 以及数据(data)。

HCodeNode 是用来表示哈夫曼编码的结构体, 其中包含两个字段: 存放哈夫曼编码的数组 cd 和编码在数组中的起始位置 start。

算法设计:

算法主要包括以下几个部分:

- 1、构造哈夫曼树算法: 首先初始化哈夫曼树, 然后逐个添加节点, 每次选择权值最小的两个节点合并, 生成新的节点。
- 2、生成哈夫曼编码算法: 根据哈夫曼树, 为每个字符生成对应的哈夫曼编码。
- 3、打印哈夫曼编码算法: 根据输入的字符序列, 打印出对应的哈夫曼编码。
- 4、解码哈夫曼编码算法: 根据输入的哈夫曼编码, 解码出对应的字符序列。

下面以伪代码形式描述这几个算法:

//构造哈夫曼树算法

函数 CreateHT(HuffmanTree ht, int n):

 初始化哈夫曼树

 读取输入的字符及其权值

 对于每个新的节点 i:

 使用 select 函数选择权值最小的两个节点 s1 和 s2

 更新 s1 和 s2 的父节点为 i, 更新 i 的左右子节点为 s1 和 s2, 更新 i 的权值为 s1 和 s2 的权值之和

function CreateHT(HuffmanTree ht, int n)

 if n is less than or equal to 1

 return

 initialize the HuffmanTree ht

 input the characters and their weights

 initialize s1, s2

 for i from n+1 to 2n-1

 call select to get s1 and s2

 set ht[s1].parent, ht[s2].parent to i

 set ht[i].lchild, ht[i].rchild to s1 and s2 respectively

 set ht[i].weight to the sum of ht[s1].weight and ht[s2].weight

//生成哈夫曼编码算法

函数 encode(HuffmanTree ht, HuffmanCode hc, int n):

 对于每个字符 i:

 初始化 hc[i].start 为 n

 根据哈夫曼树生成哈夫曼编码

function encode(HuffmanTree ht, HuffmanCode hc, int n)

 for i from 1 to n

 initialize hc[i].start to n

 set child to i, parent to ht[child].parent

 while parent is not 0

 if ht[parent].lchild equals child

 hc[i].cd[hc[i].start--] = 0

 else

 hc[i].cd[hc[i].start--] = 1

 update child and parent

 increment hc[i].start

//打印哈夫曼编码算法

函数 printCode(HuffmanTree ht, HuffmanCode hc, int n):

 读取输入的字符序列 code

 对于 code 中的每个字符:

打印出对应的哈夫曼编码

//解码哈夫曼编码算法

函数 decode(HuffmanTree ht, int n):

对于输入的哈夫曼编码中的每个二进制位:

根据哈夫曼树解码出对应的字符

```
function decode(HuffmanTree ht, int n)
    initialize p
    set i to 0
    while i is less than len
        set p to 2n - 1
        while ht[p].lchild and ht[p].rchild are not 0
            if str[i] equals 0
                set p to ht[p].lchild
            else
                set p to ht[p].rchild
            increment i
        print ht[p].data
```

四 使用说明、测试分析及结果

1、说明如何使用你编写的程序;

第一行输入字符集的大小 n , 以及 n 个字符和 n 个字符的权值;

然后输入目标字符串;

2、测试结果与分析;

输入

```
5 a b c d e 12 40 15 8 25
bbbaddeccbbb
```

输出

```
0001111110111010110000
bbbaddeccbbb
```

3、调试过程中遇到的问题是如何解决以及对设计与实现的回顾讨论和分析

1. 在构建哈夫曼树的过程中, 找到权值最小和次小的两个节点比较困难。

解决方法: 设定两个极大值 $min1$ 和 $min2$, 遍历哈夫曼树的结点, 如果发现权值小于 $min1$ 或 $min2$, 就更新 $min1$ 和 $min2$ 。

2. 在打印哈夫曼编码的过程中, 如何根据字符找到对应的哈夫曼编码?

解决方法: 遍历哈夫曼编码, 发现字符和输入的字符匹配, 就输出对应的哈夫曼编码。

回顾讨论与分析:

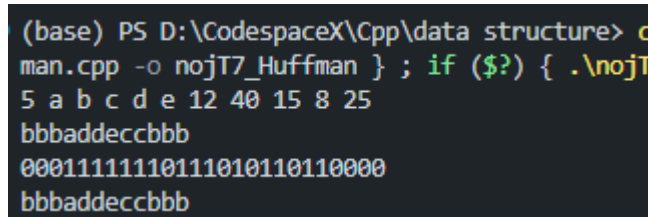
本次实验的设计和实现主要围绕哈夫曼编码和解码的过程进行。在设计哈夫曼树的数据结构时, 选择了结构体数组的方式, 每个节点包含权值、父节点、左子节点、右子节点和字符信息, 这样可以方便地实现哈夫曼编码和解码的算法。

在算法设计中, 构造哈夫曼树是最关键的部分。在构造哈夫曼树的过程中, 需要反复地选取权值最小的两个节点, 然后合并这两个节点, 并将它们的父节点添加到哈夫曼树中。这

这个过程需要对哈夫曼树的节点进行频繁的访问和修改，因此，选择了结构体数组作为哈夫曼树的数据结构，这样可以通过数组索引直接访问和修改节点。

此外，生成哈夫曼编码和解码哈夫曼编码的算法也相当重要。生成哈夫曼编码的过程需要根据哈夫曼树为每个字符生成对应的哈夫曼编码，而解码哈夫曼编码的过程则需要根据哈夫曼编码解码出对应的字符。这两个过程都需要对哈夫曼树和哈夫曼编码的结构体数组进行遍历，因此，对数据结构的设计选择也影响了算法的实现。

运行界面



```
(base) PS D:\CodespaceX\Cpp\data structure> c
man.cpp -o nojT7_Huffman } ; if ($?) { .\nojT
5 a b c d e 12 40 15 8 25
bbbaddeccbbb
0001111110111010110110000
bbbaddeccbbb
```

五、实验总结

本次实验通过编程实现了哈夫曼编码和解码的过程，加深了我对数据结构和算法的理解。在解决问题的过程中，需要运用到诸如循环、条件判断、数组等编程基本技巧，同时也需要对哈夫曼编码的原理有深入的理解。这次实验让我明白，数据结构和算法是紧密相连的，只有当理解了算法的原理，才能更好地设计数据结构，从而更有效地解决问题。

教师评语：

实验成绩：

指导教师签名：

批阅日期：