北京師範大學
BEIJING NORMAL UNIVERSITY

《数据结构》上机实验报告

第 4 次上机

学号：202011140104
姓名：李馨
学院：物理学系
专业：物理学
教师：郑新
日期：2022.10.24

一、实验要求

1. 上机之前应做好充分准备，认真思考所需的上机题目，提高上机效率。

2. 独立上机输入和调试自己所编的程序，切忌抄袭、拷贝他人程序。

3. 上机结束后，整理出实验报告。书写报告时，重点放在实验的方法、思路以及总结反思上，以达到巩固课堂学习、提高动手能力的目的。

二、实验过程

a) 问题描述：

利用最小堆编程实现给定权值集合下构造相应霍夫曼树的算法，并解决以下问题：
有一电文共使用五种字符 a, b, c, d, e, 其出现频率依次为 4, 7, 5, 2, 9。
(1) 构造对应的编码哈夫曼树(要求左子树根结点的权小于等于右子树根结点的权)。
(2) 给出每个字符的哈夫曼编码。
(3) 译出编码系列 11000111000101011 的相应电文。

b) 问题一：利用最小堆构建编码哈夫曼树（Huffman.h, Huffman.cpp）

i. **哈夫曼树定义**（Huffman.h）

```
#define leafNumber 20           //默认权重集合大小
#define totalNumber 39          //树结点最大个数
#include <string>
using std::string;              // 方便处理编码问题

typedef struct {
    char data;                  //结点的值
    int weight;                 //结点的权
    int parent, lchild, rchild; // 双亲、左、右子女
    string HuffmanCode; // Huffman编码字符串
} HTNode;

typedef struct {
    HTNode elem[totalNumber];   //树存储数组
    int num, root;              //外结点数与根
} HFTree;
```

1. 树的储存结构定义为静态三叉链表。
2. 树中外结点有 Huffman 编码，为了方便后续字符串处理，使用 C++标准库中的 string 类类型。

ii. **利用最小堆构造哈夫曼树**（void createHFTreeWithHeap()）

构造哈夫曼树的算法描述如下：
由给定 n 个权值{w0, w1, w2, …, wn-1}，构造具有 n 棵二叉树的森林 F = { T0,

T1, T2, …, Tn-1 }，其中每棵二叉树 Ti 只有一个带权值 wi 的根结点，其左、右子树均为空。重复以下步骤，直到 F 中仅剩一棵树为止：

(1)在 F 中选取两棵根结点权值最小的二叉树，做为左、右子树构造一棵新的二叉树。置新的二叉树的根结点的权值为其左、右子树上根结点的权值之和。

(2) 在 F 中删去这两棵二叉树。

(3) 把新的二叉树加入 F。

1. 首先先由给定的结点数据和权值对最初的森林进行赋值，并进行指针置空：

```
// 由给定char数组和weight数组，构造哈夫曼树，由引用型参数传递
void createHFTreeWithHeap(HFTree& HT, char value[ ], int fr[ ], int n) {
    int i, k, s1, s2;
    for (i = 0; i < n; i++) {                  // 外结点赋值（0~n-1）
        HT.elem[i].data = value[i];
        HT.elem[i].weight = fr[i];
    }
    for (i = 0; i < 2*n-1; i++) {              // 指针置空，2n-1是最后总结点数
        HT.elem[i].parent = -1;
        HT.elem[i].lchild = HT.elem[i].rchild = -1;
    }
```

2. 选取最小权值和次小权值根结点，可以使用最小堆。

a) 最小堆结构定义如下（minHeap.h），其结点数据是结构体，储存了权值、数据和在 Huffman 树中的地址（构造最小堆和构造哈夫曼树时，所用原始 value 数组的索引数）。

```
typedef struct {
    int weight;                 // 结点权值
    char value;                 // 结点数据
    int indexInTree;            // 最初索引，用于构造Huffman树
} heapNode;

typedef struct {
    // 堆结构的定义
    heapNode elem[heapSize];    // 小根堆存储数组
    int curSize;                // 当前元素个数
} minHeap;
```

b) 建立最小堆的函数（minHeap.cpp）：

```
// 小根堆的建立
void creatMinHeap ( minHeap& H, char value[], int fr[], int n ) {
// 将一个数组从局部到整体，自下向上调整为小根堆
    for ( int i = 0; i < n; i++ ) {
        // 复制为完全二叉树
        H.elem[i].weight = fr[i];
        H.elem[i].value = value[i];
        H.elem[i].indexInTree = i;
    }
    H.curSize = n;
    for ( int i = (H.curSize-2)/2; i >= 0; i-- )
            // 自底（最后一个非叶结点，即第一个叶结点的父结点）向上逐步扩大小根堆
        siftDown ( H, i, H.curSize-1 );
            // 局部自上向下筛选
}
```

3. 不断选取最小权值和次小权值根结点的二叉树构造新二叉树，被构造的根节点是内结点，在数组中的地址是 n 到 2n-2：

```
// 利用最小堆构造哈夫曼树
minHeap H;
creatMinHeap(H, value, fr, n);
// 生成堆，堆结点是结构体，储存了在原数组中的index数据
heapNode min1, min2, newHeapNode;
// 堆中最小权值、次小权值的结点，构造的新根结点
for (i = n; i < 2*n-1; i++) {
    // 逐步构造内结点（n~2*n-2）
    Remove(H, min1);     // Remove函数将堆的根节点存至min1中
    Remove(H, min2);
    s1 = min1.indexInTree; s2 = min2.indexInTree;
    // 用这两个结点构造新二叉树
    HT.elem[s1].parent = HT.elem[s2].parent = i;
    HT.elem[i].lchild = s1; HT.elem[i].rchild = s2;
    HT.elem[i].weight = min1.weight + min2.weight;
    HT.elem[i].data = ' ';
        // 内结点data可以不赋值，没有特别意义
```

4. 对堆进行操作，插入新二叉树的根节点：

```
    // 对堆进行操作：插入新的根节点
    newHeapNode = {min1.weight + min2.weight, ' ', i};
                        // 权值、数据置空、在Huffman树中索引
    Insert(H, newHeapNode);
} // end of for(i<2n-1)
```

5. 循环结束后构造完毕，再设置外结点数与根地址：

```
HT.num = n;  HT.root = 2*n-2; // 外结点数与根
```

c）问题一：运行结果（main.cpp）

i. 给定数据 weight 和权值 weight，进行构造，测试代码（所用打印函数在 Huffman.cpp 中）：

```
char value[] = "abcde"; int weight[] = {4, 7, 5, 2, 9};
HFTree HT;
createHFTreeWithHeap(HT, value, weight, 5);
printHFTree(HT);
```

输出结果为：

```
问题    终端    JUPYTER
compilation terminated.
clem@Connor:/mnt/c/Users/12879/Desktop/dataStructures/pr
clem@Connor:/mnt/c/Users/12879/Desktop/dataStructures/pr
clem@Connor:/mnt/c/Users/12879/Desktop/dataStructures/pr
index   data    weight  parent  lchild  rchild
0       a       4       5       -1      -1
1       b       7       7       -1      -1
2       c       5       6       -1      -1
3       d       2       5       -1      -1
4       e       9       7       -1      -1
5               6       6       3       0
6               11      8       2       5
7               16      8       1       4
8               27      -1      6       7
```

d) 问题二：外结点哈夫曼编码（Huffman.cpp：void encodeHuffman()）

i. 算法描述

1. 考察每一个外结点（索引 i=0~n-1，for 循环），向上追溯其双亲，判断（if）是双亲的左孩子还是右孩子。若是左孩子，在初始为空的 temp 字符串的首位插入 0，右孩子则插入 1。
2. 插入后再追溯其双亲的双亲，进行同样的判断和对字符串的操作，直至追溯至根节点（while 循环）。

ii. 代码部分

```
// 对HFTree进行编码，将编码储存进该树中并打印
void encodeHuffman(HFTree& HT) {
    int i, n = HT.num;
    int j, k; string temp;
    // printf("index\tdata\tcode\n");  // 打印表头
    for(i = 0; i < n; i++) {
        temp = ""; j = i;
        while (HT.elem[j].parent != -1) { // 该结点有双亲
            k = HT.elem[j].parent;        // 复制该双亲备用
            if(HT.elem[k].lchild == j) {  // 该结点是左孩子
                temp = "0" + temp;
            }
            else if(HT.elem[k].rchild == j) { // 该结点是右孩子
                temp = "1" + temp;
            }
            j = k; // 取其双亲，进入下一循环
        }
        HT.elem[i].HuffmanCode = temp;
        // 打印该结点编码
        // printf("%d\t%c\t%s\n", i, HT.elem[i].data, temp.c_str());
    }
}
```

1. 编码通过引用型参数储存到被引用的哈夫曼树的每个外结点中，即 HT.elem[i].HuffmanCode。
2. 方便起见，用到了 C++的 string 类的很多操作，比如“在字符串首位插入”这一操作用了 C++特有的加号的运算符。

e) 问题二：运行结果（main.cpp）

i. 沿用上一问的哈夫曼树，对之进行编码，测试代码为：

```
printf("    Huffman编码表\n");
encodeHuffman(HT);
```

ii. 运行结果为：

问题 终端 JUPYTER

index	data	weight	parent	lchild	rchild
0	a	4	5	-1	-1
1	b	7	7	-1	-1
2	c	5	6	-1	-1
3	d	2	5	-1	-1
4	e	9	7	-1	-1
5		6	6	3	0
6		11	8	2	5
7		16	8	1	4
8		27	-1	6	7

Huffman编码表

index	data	code
0	a	011
1	b	10
2	c	00
3	d	010
4	e	11

f) 问题三：翻译编码（Huffman.cpp: void decodeHuffman(HFTree, string)）

i. **算法描述**

1. 首先对给定哈夫曼树进行编码，即调用上一问 encodeHuffman 函数，将编码储存进该树中。
2. 然后对字符串进行从前到后的扫描，与树中哈夫曼编码进行匹配，匹配成功后删除被匹配的字符串区间，再继续扫描，直到字符串为空，结束扫描，返回结果字符串。

ii. **代码部分**

```
// 根据给定HFTree和字符串，进行解码
string decodeHuffman(HFTree& HT, string& codeline) {
    // 编码操作
    encodeHuffman(HT);
    string result; result = "";
    // 11000111000101011
    int n = HT.num; int i, j;
    while (codeline.size() != 0) {
        for (i = 0; i < n; i++) {
            j = codeline.find(HT.elem[i].HuffmanCode);
                // 返回 codeline中 第一个与该字符哈夫曼编码匹配
                // 的子字符串的起始索引，找不到则返回-1
            if (j == 0) {
                // 如果是在开头就匹配了，可以解码此字符
                codeline.erase(0, HT.elem[i].HuffmanCode.size());
                // 在字符串删除位于首位的该编码
                result += HT.elem[i].data;
                break;
            }
            if (i == 4 && j == -1) exit(1); // 无法匹配，解码失败
        }
    }
    std::cout << "解码结果为：" << result << std::endl;
    return result;
}
```

1. “匹配”和“删除”操作使用了 string 类的 find 和 erase 函数。

g) 问题三：运行结果（main.cpp）

i. 沿用上一问的哈夫曼树，对之进行解码，测试代码为：

```
string codeline("11000111000101011");
std::cout << decodeHuffman(HT, codeline) << std::endl;
```

ii. 运行结果为：

```
问题    终端    JUPYTER

index   data    code
0       a       011
1       b       10
2       c       00
3       d       010
4       e       11
解码结果为：ecabcbbe
ecabcbbe
```

三、总结（实验中遇到的问题、取得的经验、感想等）

- 由于哈夫曼树结点和最小堆的结点是两种不同的结构体，在运用时互通会有一些麻烦。比如，找到了最小权值点和次小权值点后突然发现，我们并不知道它们在树中的位置。
  - 这里选择要求在建立哈夫曼树和最小堆时使用相同的数组，根据这一原始数组的索引实现互通。
    - 哈夫曼树的结点中外结点位置索引相较原数组不改变；但在最小堆的结点里会反复重新筛选，位置索引发生改变。
    - 故而在最小堆结点的结构体中储存原始数组索引信息，也即是在哈夫曼树中的索引。
- 实验后两问中为了方便字符串的处理大量运用了 C++的 string 类的方法。
- 在实验中加强了对哈夫曼树构造和编码过程、最小堆的应用的理解和把握。