**《数据结构》上机报告**

**2018 年 11 月 11 日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **姓名：** | **赵得泽** | **学号：** | **1753642** | **班级：** | **电子2班** | **得分：** |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **实验题目** | | 哈夫曼树 | | |
| **问题描述** | | 哈夫曼树，又称最优树，是一类带权路径长度最短的树。 | | |
| **基本要求** | | 1. 练习构建哈夫曼树。 2. 练习用双亲表示法输入哈夫曼树，给出哈夫曼编码。 3. 练习对哈夫曼编码进行译码。 | | |
| **已完成基本内容（序号）：** | 1，2，3 | |
| **选做要求** | |  | | |
| **已完成选做内容（序号）** | |  |
| **数据结构**  **设计** | | **typedef struct {**  **int weight;**  **int parent, lchild, rchild;**  **}HTNode, \*HuffmanTree;**  本实验的数据结构是哈夫曼树，就是一种特殊的二叉树，又叫做**最优二叉树**。  在创建结构体的时候，里面都是int型数据，比如：**权值weight，双亲结点parent，左右孩子lchild、rchild,**没有指针数据。那么这个树实际上就是一个结构体数组构成的，再为结构体数组的各个数据元素进行赋值，就形成了一张表，只要通过给出的权值进行左右孩子及双亲节点的寻找，那么哈夫曼树也就通过这张表建立起来了。 | | |
| **功能(函数)说明** | | **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **函数功能:求最小值**  **说明：求最小值的时候，首先定义一个变量min来存储最小值，并且赋初值**  **为较大的数，然后定义一个标记变量，用来记录最小值的下标，如果这个**  **最小值已经找到，那么就对它的parent变量进行标记，置为1，防止重复**  **寻找。**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **int Min(HuffmanTree &HT, int i)**  **{**  **int min =65535;**  **int j, flag;**  **for (j = 1; j <= i; j++)**  **if (HT[j].weight < min && HT[j].parent == 0)**  **{**  **min = HT[j].weight;**  **flag = j;**  **}**  **HT[flag].parent = 1;**  **return flag;**  **}**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **函数功能：求两个最小值**  **说明：找到最小的两个值，将下标存放在s1,s2中，s1存放最小值下标，s2存放次小值下标，如果s1大于s2，就将数值进行交换，让s1永远都存储最小的下标，s2存放次小的下标。**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **void Select(HuffmanTree &HT, int i, int &s1, int &s2)**  **{**  **s1 = Min(HT, i);**  **s2 = Min(HT, i);**  **int temp=0;**  **if (s1>s2)**  **{**  **temp = s1;**  **s1 = s2;**  **s2 = temp;**  **}**  **}**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **函数功能：创建哈夫曼树**  **说明：首先将n个叶子结点的权值存放在哈夫曼树表的下标从1-n,结构体中别的数据都置零，接下来，每次都从权值中选取最小的两个，权值相加，作为新的权值追加到表的后面，并且将它的左右孩子分别赋值为两个最小权值的下标，也将两个最小权值的双亲结点赋值为新得到的权值的下标，直到最后一个（即2\*n-1),建立完毕。**  **计算总权值的时候，只要把非叶子结点的权值相加即可。**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **void HuffmanCoding(HuffmanTree &HT, HuffmanCode &HC, int w[], int& n, int &sum\_w)**  **{**  **if (n <= 1)**  **return;**  **int m;**  **int s1, s2;**  **m = 2 \* n - 1;**  **HT = new HTNode[m + 1];**  **HuffmanTree p;**  **int i;**  **for (p = HT + 1, i = 1; i <= n; i++, p++, w++)**  **\*p = { \*w,0,0,0 };**  **for (; i <= m; i++)**  **\*p = { 0,0,0,0 };**  **for (i = n + 1; i <= m; i++)**  **{**  **Select(HT,i-1,s1,s2);**  **HT[s1].parent = i;**  **HT[s2].parent = i;**  **HT[i].lchild = s1;**  **HT[i].rchild = s2;**  **HT[i].weight = HT[s1].weight + HT[s2].weight;**  **}**  **HT[m].parent = 0;**  **for (int j = n + 1; j <= m; j++)**  **sum\_w += HT[j].weight;**  **}**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **函数功能:双亲表示法创建哈夫曼树**  **说明：每输入一个结点的信息的时候就进行创建，节点信息包括：结点的权**  **值、双亲、是否是双亲的左孩子，然后分别将其填到原先创建哈夫曼树用的**  **表，但是到最后一个结点的时候，则不必判断否是左右孩子，因为它是根结**  **点。**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **void InputHuffman(HuffmanTree &HT, int &n)**  **{**  **int i, w, pa, lc;**  **int m = 2 \* n - 1;**  **HT = new HTNode[m + 1];**  **HuffmanTree p, q;**  **for (p = HT + 1, i = 1; i <= m; i++, p++)**  **\*p = { 0,0,0,0 };**  **for (q = p = HT + 1, i = 1; i <= m; i++, p++)**  **{**  **cin >> w >> pa >> lc;**  **p->weight = w;**  **p->parent = pa;**  **if (i != m)**  **if (lc == 0)**  **(q + pa - 1)->lchild = i;**  **else**  **(q + pa - 1)->rchild = i;**  **else**  **break;**  **}**  **}**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **函数功能:哈夫曼编码**  **说明：编码的时候应该先申请一个n+1字节大小的二维字符数组，和一个n**  **字节大小的字符数组，后者用来存储对1-n个叶子结点编码的01字符串，前者则将截取的字符串（只有字符和结束标志）对应每个叶子结点存起来。**  **编码的时候，从1-n顺序编码，通过循环，是双亲结点的左孩子，编号0，否则，编号1，直到双亲结点的parent=0,即到了根结点，退出循环，一个叶子结点的编码结束。**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **void HuffmanCoding(HuffmanTree &HT, HuffmanCode &HC, int& n)**  **{**  **int start, c, f, i;**  **HC = (HuffmanCode)malloc((n + 1) \* sizeof(char \*));**  **char\* cd = new char[n];**  **cd[n - 1] = '\0';**  **for (i = 1; i <= n; i++)**  **{**  **start = n - 1;**  **for (c = i, f = HT[c].parent; f != 0; c = f, f = HT[f].parent)**  **if (HT[f].lchild == c)cd[--start] = '0';**  **else cd[--start] = '1';**  **HC[i] = new char[n - start];**  **strcpy(HC[i], &cd[start]);**  **cout << i << " " << HC[i] << endl;**  **}**  **delete[]cd;**  **}**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **函数功能:输入每个字符及编码**  **说明：建立一个结构体，包括字符的ASCII值及编码字符串；然后每次输入**  **的时候，动态建立数组，将编码存进字符串即可。**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **typedef struct Code {**  **int c\_ASCII;**  **char \*str;**  **}\*Cd;**  **void input(Cd &cd, int n)**  **{**  **cd = new Code[n + 1];**  **int i = 1;**  **while (i <= n)**  **{**  **cin >> cd[i].c\_ASCII;**  **cd[i].str = new char[10];**  **cin >> cd[i].str;**  **i++;**  **}**  **}**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **函数功能:译码**  **说明：对输入的待译码字符串用指针p逐个考察，直到末尾。每次考察都将**  **其存在一个字符数组中，然后和输入的字符编码进行比较，若相等，则输出**  **该译码，并将该字符串清空，计数置零，继续接着考察。因为霍夫曼编码都**  **是前缀编码，每个字符都不会有相同的前缀，所以采用此方式。**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **void Decoding(Cd &cd, int n, char s[])**  **{**  **char \*p = s;**  **char s1[10];**  **int i = 0, k;**  **while (\*p != '\0')**  **{**  **s1[i++] = \*p;**  **s1[i] = '\0';**  **for (int k = 1; k <= n; k++)**  **{**  **if (strcmp(s1, cd[k].str) == 0)**  **{**  **cout << char(cd[k].c\_ASCII);**  **memset(s1, '\0', sizeof(s1));**  **i = 0;**  **break;**  **}**  **}**  **p++;**  **}**  **}** | | |
| **开发环境** | | Win10,vs2017,C++高级程序语言设计 | | |
| **调试分析** | | **1.**    **2**.    **3**. | | |
| **心得体会** | 本实验主要是哈夫曼树的创建，哈夫曼编码，译码过程的应用。  **创建哈夫曼树：**主要就是建立一个结构体，申请结构体数组，相当于建立了一张表，然后通过寻找双亲，从叶子结点到根节点进行创建。在创建的过程中，最关键的就是寻找最小的两个叶子结点，在此可以借助结点的parent域，因为parent域已经都初始化为0，那么在寻找最小权值的时候，若它的weight当前得到的最小值小并且parent域为0，即它还没有被当做最小值存储过，若寻找到就让它的parent域标记为1，这样就会避免重复比较这个结点数据。  **哈夫曼编码：**就是从第一个到第n个顺序遍历进行编码，编码时，从叶子结点  到根节点进行编码，每次都要寻找双亲，如果该结点时双亲的左孩子就编码为0，否则，编码为1；如此循环，直到所有的叶子节点都编码完毕。  **译码：**因为每个字符经过哈夫曼编码之后，每个编码不会重复，且不会有相同的前缀，那么对于输入的待译码字符串，只要逐个遍历，然后存进数组和每个字符的编码进行比较即可，若相等，就直接进行译码，清空字符串，计数置零；否则就再进行遍历存到字符串数组，循环直到待译码字符串结尾。  总之，哈夫曼树的创建，编码都是在寻找双亲结点，确定左右孩子结点的基础上进行的，只要掌握好这些关系，就可以很轻松地解决问题。 | | | |