**数据结构大作业**

**二、题目6（哈夫曼树的编/译码器）**

**1.1问题描述**

利用哈夫曼编码进行通讯可以大大提高信道利用率，缩短信息传输时间，减低传输成本。但是，这要求在发送端通过一个编码系统对待传数据进行预先编码；在接受端将传来的数据进行解码（复原）。对于可以双向传输的信道，每端都要有一个完整的编/译码系统。试为这样的信息收发站写一个哈夫曼的编译码系统。

**1.2问题分析**

此题为哈夫曼树的编码问题，包括了哈夫曼树的构造、编码和译码。哈夫曼树编译码的主要用途是实现数据压缩，要对电文中出现的每一个字符进行二进制编码，需遵循两个原则：

1. 发送方传输的二进制编码，到接收方解码后必须具有唯一性，即解码结果与发送方发送的电文完全一致；
2. 发送的二进制编码尽可能短。

根据这两点要求，本题所设计的算法需满足以下要求：

1. 根据输入的字符和字符相应的频度建立哈夫曼树；
2. 能将建立好的哈夫曼树用简单易懂的方式输出；
3. 实现哈夫曼树的编码和译码功能，电文内容与编码内容一一对应；
4. 采用不等长编码，根据不同字符的出现频率基于不等长编码，使用频度较高的字符分配较短的编码，使用频度较低的字符分配较长的编码；
5. 操作界面简洁。

**1.3问题求解**

**1.3.1建立哈夫曼树**

首先，定义哈夫曼树的储存结构，该哈夫曼树的结构体需要一个整型变量储存结点的权值weight，由两个指针分别储存结点的左右孩子，此外，由于输入的权值存在重复，需要额外用一个整型变量id加以区分。

其次，构造哈夫曼树，构造哈夫曼树的哈夫曼算法如下：

1. 由给定的n个权值，构造具有n棵二叉树的森林，其中每一棵二叉树只有一个带有权值的根结点，其左、右结点均为空；
2. 在森林中选取两棵根结点权值最小和次小的二叉树，作为左右子树构造一棵二叉树，其根结点的权值即为其左、右子树的根结点的权值之和；
3. 从森林中删除已构成新二叉树的左右子树的两棵二叉树，并将新构成的二叉树放入森林中；
4. 重复（2）和（3），直到F中仅剩一颗二叉树，即哈夫曼树。

根据算法可知，一个由n个叶子结点组成的初始集合，要生成哈夫曼树需要进行n-1次合并，产生n-1个新结点，最终所得的哈夫曼树共有2n-1个结点。

C语言描述如下：

**// 哈夫曼树结点结构体**

**typedef struct HuffmanTree**

**{**

**int weight;**

**int id; // id用来主要用以区分权值相同的结点**

**struct HuffmanTree\* lchild;**

**struct HuffmanTree\* rchild;**

**}HuffmanNode;**

**// 构建哈夫曼树**

**HuffmanNode\* createHuffmanTree(int\* a, int n)**

**{**

**int i, j;**

**HuffmanNode \*\*temp, \*hufmTree;**

**temp = (HuffmanNode\*\*)malloc(n \* sizeof(HuffmanNode));**

**for (i = 0; i < n; ++i)**

**{**

**temp[i] = (HuffmanNode\*)malloc(sizeof(HuffmanNode));**

**temp[i]->weight = a[i]; // 将数组a中的权值赋给结点中的weight**

**temp[i]->id = i;**

**temp[i]->lchild = temp[i]->rchild = NULL;**

**}**

**for (i = 0; i < n - 1; ++i) // 构建哈夫曼树需要n-1合并**

**{**

**int small1 = -1, small2; // small1、small2分别作为最小和次小权值的下标**

**for (j = 0; j < n; ++j) // 先将最小的两个下标赋给small1、small2**

**{**

**if (temp[j] != NULL && small1 == -1)**

**{**

**small1 = j;**

**continue;**

**}**

**else if (temp[j] != NULL)**

**{**

**small2 = j;**

**break;**

**}**

**}**

**for (j = small2; j < n; ++j)**

**{**

**if (temp[j] != NULL)**

**{**

**if (temp[j]->weight < temp[small1]->weight)**

**{**

**small2 = small1;**

**small1 = j;**

**}**

**else if (temp[j]->weight < temp[small2]->weight)**

**{**

**small2 = j;**

**}**

**// 比较权值，挪动small1和small2使之分别成为最小和次小权值的下标**

**}**

**}**

**hufmTree = (HuffmanNode\*)malloc(sizeof(HuffmanNode));**

**hufmTree->weight = temp[small1]->weight + temp[small2]->weight;**

**hufmTree->lchild = temp[small1];**

**hufmTree->rchild = temp[small2];**

**temp[small1] = hufmTree;**

**temp[small2] = NULL;**

**}**

**free(temp);**

**return hufmTree;**

**}**

**1.3.2哈夫曼树的输出**

关于哈夫曼树的输出，可使用广义表的形式输出，可采用递归的方法。

C语言描述如下：

**// 以广义表的形式打印哈夫曼树**

**void PrintHuffmanTree(HuffmanNode\* hufmTree)**

**{**

**if (hufmTree)**

**{**

**printf("%d", hufmTree->weight);**

**if (hufmTree->lchild != NULL || hufmTree->rchild != NULL)**

**{**

**printf("(");**

**PrintHuffmanTree(hufmTree->lchild);**

**printf(",");**

**PrintHuffmanTree(hufmTree->rchild);**

**printf(")");**

**}**

**}**

**}**

**1.3.3对每个字符进行编码输出**

此哈夫曼树编码方式为从根节点出发，左子树的路径代码设为‘0‘，右子树的路径设为‘1’，查找叶子结点，将每个字符储存在数组string中，将路径上的‘0’和‘1‘储存在数组code中，通过递归的方式逐层查找该哈夫曼树的叶子结点，若该结点的左右子树非空，在code中输入‘0’并查找左子树，在code中输入‘1’并查找右子树，若该结点的左右子树均为空，则该结点为叶子结点，用code输出该结点相应的字符的编码。

C语言描述如下：

**// 递归进行哈夫曼编码**

**void HuffmanCode(HuffmanNode\* hufmTree, int depth,char string[])**

**{**

**static int code[10];**

**if (hufmTree)**

**{**

**if (hufmTree->lchild == NULL && hufmTree->rchild == NULL)**

**{**

**printf("%c的哈夫曼编码为： ",string[hufmTree->id]);**

**int i;**

**for (i = 0; i < depth; ++i)**

**{**

**printf("%d", code[i]);**

**}**

**printf("\n");**

**}**

**else**

**{**

**code[depth] = 0;**

**HuffmanCode(hufmTree->lchild, depth + 1,string);**

**code[depth] = 1;**

**HuffmanCode(hufmTree->rchild, depth + 1,string);**

**}**

**}**

**}**

**1.3.4对输入的字符串进行编码**

先将新输入的字符串储存在string中，调用建立哈夫曼树时所用的字符数组ch，逐个从ch中查找到与string相同的字符，匹配之后，将该字符的权值代入Encode函数中进行编码，逐个输出string中字符的编码。

C语言描述如下：

**//哈夫曼编码**

**void Encode(HuffmanNode\* hufmTree, int depth, int w)**

**{**

**static int num[10];**

**if (hufmTree)**

**{**

**if (hufmTree->lchild == NULL && hufmTree->rchild == NULL && hufmTree->weight == w)**

**{**

**int i;**

**for (i = 0; i < depth; ++i)**

**{**

**printf("%d", num[i]);**

**}**

**}**

**else**

**{**

**num[depth] = 0;**

**Encode(hufmTree->lchild, depth + 1, w);**

**num[depth] = 1;**

**Encode(hufmTree->rchild, depth + 1, w);**

**}**

**}**

**}**

**void HuffmanEncode(HuffmanNode\* hufmTree, char string[], char ch[], int\* a)**

**{**

**int i, j;**

**for (i = 0; i < strlen(string); i++)**

**{**

**for (j = 0; j < strlen(ch); j++)**

**{**

**if (string[i] == ch[j])**

**{**

**Encode(hufmTree, 0, a[j]);**

**}**

**}**

**}**

**}**

**1.3.5对输入的编码进行译码**

取ch为要解码的编码串，string是结点对应的字符串，以ch所给的编码值作为查找路径，从根结点出发，查找叶子结点，并输出该叶子结点所代表的字符，当查找到字符时，返回根结点，进行下一个编码值的查找。

C语言描述如下：

**// 哈夫曼解码**

**void HuffmanDecode(char ch[], HuffmanNode\* hufmTree, char string[])**

**{**

**int i;**

**int num[100];**

**HuffmanNode\* tempTree = NULL;**

**for (i = 0; i < strlen(ch); ++i)**

**{**

**if (ch[i] == '0')**

**num[i] = 0;**

**else**

**num[i] = 1;**

**}**

**if (hufmTree)**

**{**

**i = 0;**

**while (i < strlen(ch))**

**{**

**tempTree = hufmTree;**

**while (tempTree->lchild != NULL && tempTree->rchild != NULL)**

**{**

**if (num[i] == 0)**

**{**

**tempTree = tempTree->lchild;**

**}**

**else**

**{**

**tempTree = tempTree->rchild;**

**}**

**i++;**

**}**

**printf("%c", string[tempTree->id]); // 输出解码后对应结点的字符**

**}**

**}**

**}**

**1.3.6操作界面设置**

本程序的操作界面分为四个部分：建立哈夫曼树的输入部分；输出哈夫曼树部分；输入字符相应编码值部分；编码译码选择部分。

将程序的编码译码选择部分设计成一个选择页面，分为“编码”、“译码”、“退出”三个部分，通过输入‘e‘、’d‘、’q‘进行操作的选择。

C语言描述如下：

**//主程序**

**int main()**

**{**

**printf("创建哈夫曼树:");**

**int i, n;**

**printf("请输入字符的个数：\n");**

**while (1)**

**{**

**scanf("%d", &n);**

**if (n > 1)**

**break;**

**else**

**printf("输入错误，请重新输入！");**

**}**

**int\* arr;**

**arr = (int\*)malloc(n \* sizeof(int));**

**printf("请输入%d个字符出现的频度：\n", n);**

**for (i = 0; i < n; ++i)**

**{**

**scanf("%d", &arr[i]);**

**}**

**char ch[100], string[100],c;**

**printf("请连续输入这%d个频度各自所代表的字符：\n", n);**

**fflush(stdin); // 强行清除缓存中的数据，也就是上面输入权值结束时的回车符**

**gets(string);**

**HuffmanNode\* hufmTree = NULL;**

**hufmTree = createHuffmanTree(arr, n);**

**printf("此哈夫曼树的广义表形式为：\n");**

**PrintHuffmanTree(hufmTree);**

**printf("\n各字符的哈夫曼编码为：\n");**

**HuffmanCode(hufmTree, 0, string);**

**printf("\ne:编码(Encode)\nd:解码(Decode)\nq:退出(Quit))\n");**

**printf("请选择操作：");**

**scanf("%c",&c);**

**do**

**{**

**switch (c)**

**{**

**case'e':**

**{**

**printf("请输入想要编码的文本：");**

**fflush(stdin);**

**gets(ch);**

**printf("编码结果为：\n");**

**HuffmanEncode(hufmTree, ch, string, arr);**

**printf("\n");**

**printf("\ne:编码(Encode)\nd:解码(Decode)\nq:退出(Quit))\n");**

**printf("请选择操作：");**

**break;**

**}**

**case'd':**

**{**

**printf("请输入想要解码的电文：");**

**fflush(stdin);**

**gets(ch);**

**printf("解码结果为：\n");**

**HuffmanDecode(ch, hufmTree, string);**

**printf("\n");**

**printf("\ne:编码(Encode)\nd:解码(Decode)\nq:退出(Quit))\n");**

**printf("请选择操作：");**

**break;**

**}**

**case'q':**

**{**

**exit(0);**

**break;**

**}**

**default:**

**{**

**exit(0);**

**break;**

**}**

**}**

**scanf("%c",&c);**

**} while (c != 'q');**

**free(arr);**

**free(hufmTree);**

**return 0;**

**}**

**1.4程序运行**

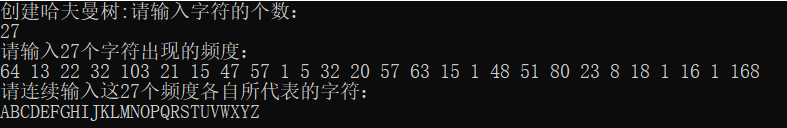
所给数据：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 字符 | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
| 频度 | 64 | 13 | 22 | 32 | 103 | 21 | 15 | 47 | 57 | 1 | 5 | 32 | 20 | 57 |
| 字符 | O | P | Q | R | S | T | U | V | W | X | Y | Z | 空格 |  |
| 频度 | 63 | 15 | 1 | 48 | 51 | 80 | 23 | 8 | 18 | 1 | 16 | 1 | 168 |  |

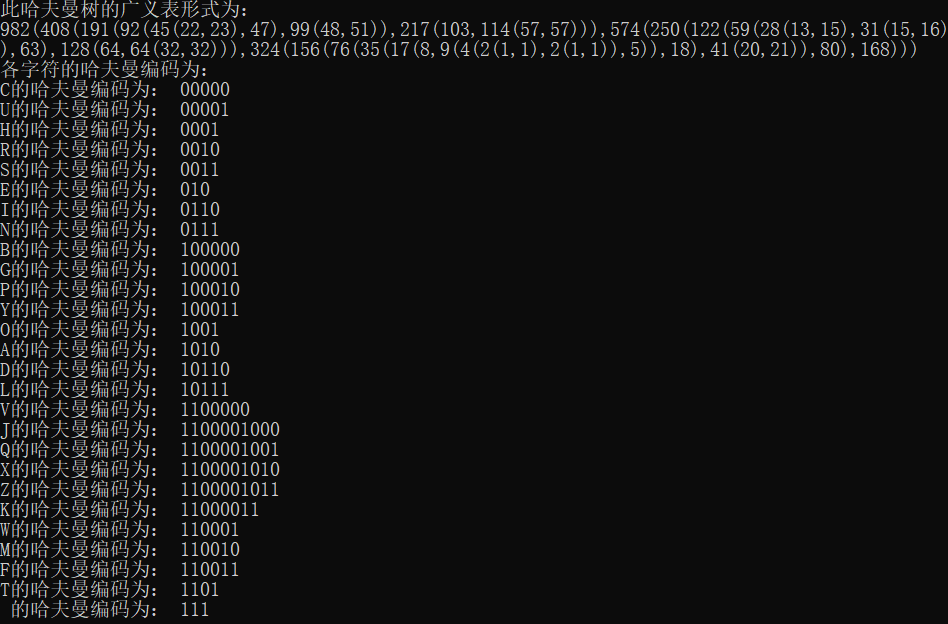
运行结果：

一共有27个字符需要输入

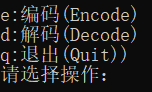
输入操作如下：



哈夫曼树输出和字符编码输入如下：

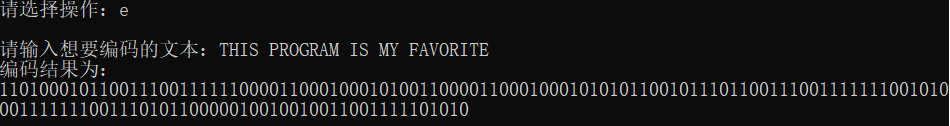


操作选择页面如下：



* 选择“编码”选项，并输入：THIS PROGRAM IS MY FAVORITE

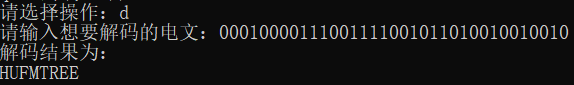
输出结果如下：



输出结果经比较，与字符编码相符，编码正确。

* 选择“解码”选项，并输入：00010000111001111001011010010010010

输出结果如下：



输出结果经比较，我字符相符，解码正确。

* 选择“退出”选项，程序退出。

**1.5总结**

利用哈夫曼树，将字符转换成一组最优前缀编码，从而实现数据的压缩，降低传输的难度，提高传输的效率。

实验心得：通过学习哈夫曼树的定义和原理，基本掌握了构造哈夫曼树的意义以及算法思想，通过实际上机实验，具体构造哈夫曼树，进一步理解了构造哈夫曼树编码的意义。

**附录：**

完整代码如下：

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**// 哈夫曼树结点结构体**

**typedef struct HuffmanTree**

**{**

**int weight;**

**int id; // id用来主要用以区分权值相同的结点**

**struct HuffmanTree\* lchild;**

**struct HuffmanTree\* rchild;**

**}HuffmanNode;**

**// 构建哈夫曼树**

**HuffmanNode\* createHuffmanTree(int\* a, int n)**

**{**

**int i, j;**

**HuffmanNode \*\*temp, \*hufmTree;**

**temp = (HuffmanNode\*\*)malloc(n \* sizeof(HuffmanNode));**

**for (i = 0; i < n; ++i) // 将数组a中的权值赋给结点中的weight**

**{**

**temp[i] = (HuffmanNode\*)malloc(sizeof(HuffmanNode));**

**temp[i]->weight = a[i];**

**temp[i]->id = i;**

**temp[i]->lchild = temp[i]->rchild = NULL;**

**}**

**for (i = 0; i < n - 1; ++i) // 构建哈夫曼树需要n-1合并**

**{**

**int small1 = -1, small2; // small1、small2分别作为最小和次小权值的下标**

**for (j = 0; j < n; ++j) // 先将最小的两个下标赋给small1、small2**

**{**

**if (temp[j] != NULL && small1 == -1)**

**{**

**small1 = j;**

**continue;**

**}**

**else if (temp[j] != NULL)**

**{**

**small2 = j;**

**break;**

**}**

**}**

**for (j = small2; j < n; ++j)**

**{**

**if (temp[j] != NULL)**

**{**

**if (temp[j]->weight < temp[small1]->weight) // 比较权值，挪动small1和small2使之分别成为最小和次小权值的下标**

**{**

**small2 = small1;**

**small1 = j;**

**}**

**else if (temp[j]->weight < temp[small2]->weight) // 比较权值，挪动small1和small2使之分别成为最小和次小权值的下标**

**{**

**small2 = j;**

**}**

**}**

**}**

**hufmTree = (HuffmanNode\*)malloc(sizeof(HuffmanNode));**

**hufmTree->weight = temp[small1]->weight + temp[small2]->weight;**

**hufmTree->lchild = temp[small1];**

**hufmTree->rchild = temp[small2];**

**temp[small1] = hufmTree;**

**temp[small2] = NULL;**

**}**

**free(temp);**

**return hufmTree;**

**}**

**// 以广义表的形式打印哈夫曼树**

**void PrintHuffmanTree(HuffmanNode\* hufmTree)**

**{**

**if (hufmTree)**

**{**

**printf("%d", hufmTree->weight);**

**if (hufmTree->lchild != NULL || hufmTree->rchild != NULL)**

**{**

**printf("(");**

**PrintHuffmanTree(hufmTree->lchild);**

**printf(",");**

**PrintHuffmanTree(hufmTree->rchild);**

**printf(")");**

**}**

**}**

**}**

**// 递归进行哈夫曼编码**

**void HuffmanCode(HuffmanNode\* hufmTree, int depth,char string[]) // depth为哈夫曼树的深度**

**{**

**static int code[10];**

**if (hufmTree)**

**{**

**if (hufmTree->lchild == NULL && hufmTree->rchild == NULL)**

**{**

**printf("%c的哈夫曼编码为： ",string[hufmTree->id]);**

**int i;**

**for (i = 0; i < depth; ++i)**

**{**

**printf("%d", code[i]);**

**}**

**printf("\n");**

**}**

**else**

**{**

**code[depth] = 0;**

**HuffmanCode(hufmTree->lchild, depth + 1,string);**

**code[depth] = 1;**

**HuffmanCode(hufmTree->rchild, depth + 1,string);**

**}**

**}**

**}**

**//哈夫曼编码**

**void Encode(HuffmanNode\* hufmTree, int depth, int w)**

**{**

**static int num[10];**

**if (hufmTree)**

**{**

**if (hufmTree->lchild == NULL && hufmTree->rchild == NULL && hufmTree->weight == w)**

**{**

**int i;**

**for (i = 0; i < depth; ++i)**

**{**

**printf("%d", num[i]);**

**}**

**}**

**else**

**{**

**num[depth] = 0;**

**Encode(hufmTree->lchild, depth + 1, w);**

**num[depth] = 1;**

**Encode(hufmTree->rchild, depth + 1, w);**

**}**

**}**

**}**

**void HuffmanEncode(HuffmanNode\* hufmTree, char string[], char ch[], int\* a)**

**{**

**int i, j;**

**for (i = 0; i < strlen(string); i++)**

**{**

**for (j = 0; j < strlen(ch); j++)**

**{**

**if (string[i] == ch[j])**

**{**

**Encode(hufmTree, 0, a[j]);**

**}**

**}**

**}**

**}**

**// 哈夫曼解码**

**void HuffmanDecode(char ch[], HuffmanNode\* hufmTree, char string[]) // ch是要解码的01串，string是结点对应的字符**

**{**

**int i;**

**int num[100];**

**HuffmanNode\* tempTree = NULL;**

**for (i = 0; i < strlen(ch); ++i)**

**{**

**if (ch[i] == '0')**

**num[i] = 0;**

**else**

**num[i] = 1;**

**}**

**if (hufmTree)**

**{**

**i = 0;**

**while (i < strlen(ch))**

**{**

**tempTree = hufmTree;**

**while (tempTree->lchild != NULL && tempTree->rchild != NULL)**

**{**

**if (num[i] == 0)**

**{**

**tempTree = tempTree->lchild;**

**}**

**else**

**{**

**tempTree = tempTree->rchild;**

**}**

**i++;**

**}**

**printf("%c", string[tempTree->id]); // 输出解码后对应结点的字符**

**}**

**}**

**}**

**//主程序**

**int main()**

**{**

**printf("创建哈夫曼树:");**

**int i, n;**

**printf("请输入字符的个数：\n");**

**while (1)**

**{**

**scanf("%d", &n);**

**if (n > 1)**

**break;**

**else**

**printf("输入错误，请重新输入！");**

**}**

**int\* arr;**

**arr = (int\*)malloc(n \* sizeof(int));**

**printf("请输入%d个字符出现的频度：\n", n);**

**for (i = 0; i < n; ++i)**

**{**

**scanf("%d", &arr[i]);**

**}**

**char ch[100], string[100],c;**

**printf("请连续输入这%d个频度各自所代表的字符：\n", n);**

**fflush(stdin); // 强行清除缓存中的数据，也就是上面输入权值结束时的回车符**

**gets(string);**

**HuffmanNode\* hufmTree = NULL;**

**hufmTree = createHuffmanTree(arr, n);**

**printf("此哈夫曼树的广义表形式为：\n");**

**PrintHuffmanTree(hufmTree);**

**printf("\n各字符的哈夫曼编码为：\n");**

**HuffmanCode(hufmTree, 0, string);**

**printf("\ne:编码(Encode)\nd:解码(Decode)\nq:退出(Quit))\n");**

**printf("请选择操作：");**

**scanf("%c",&c);**

**do**

**{**

**switch (c)**

**{**

**case'e':**

**{**

**printf("请输入想要编码的文本：");**

**fflush(stdin);**

**gets(ch);**

**printf("编码结果为：\n");**

**HuffmanEncode(hufmTree, ch, string, arr);**

**printf("\n");**

**printf("\ne:编码(Encode)\nd:解码(Decode)\nq:退出(Quit))\n");**

**printf("请选择操作：");**

**break;**

**}**

**case'd':**

**{**

**printf("请输入想要解码的电文：");**

**fflush(stdin);**

**gets(ch);**

**printf("解码结果为：\n");**

**HuffmanDecode(ch, hufmTree, string);**

**printf("\n");**

**printf("\ne:编码(Encode)\nd:解码(Decode)\nq:退出(Quit))\n");**

**printf("请选择操作：");**

**break;**

**}**

**case'q':**

**{**

**exit(0);**

**break;**

**}**

**default:**

**{**

**exit(0);**

**break;**

**}**

**}**

**scanf("%c",&c);**

**} while (c != 'q');**

**free(arr);**

**free(hufmTree);**

**return 0;**

**}**