Program2 Report

1. 簡單描述何為 huffman encoding 及其用途

Huffman coding mainly use for compress the data memory occupy, and it would not damage the data. According each data's appear frequency, and the data which appear more would give the shortest code, simply, the data appear less would give the longest code. The way to construct the Huffman code, first, is construct the Huffman tree , another condition for the Huffman tree is it will be the shosrtest weighted path length(WPL) of tree. Notice that the Huffman tree is not unique , however, the weighted path length of Huffman tree is unique.

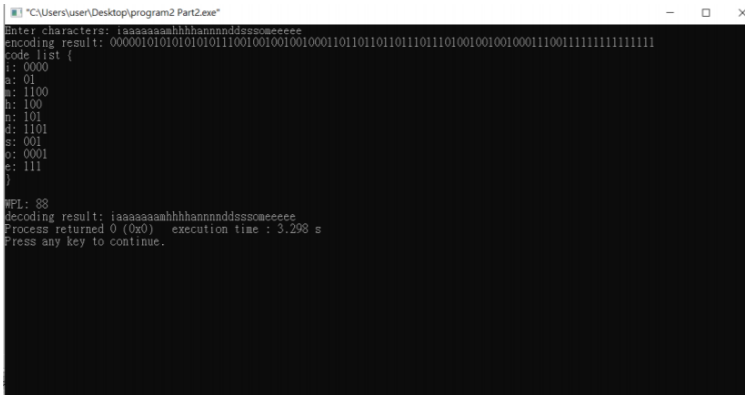
2. Part1 的結果

The code list from A to Z :

A: 1010, B: 111000, C: 111001, D: 10010, E: 010, F: 111010, G: 111011, H: 0011, I: 0110, J: 1111110, K: 1111111, L: 11010, M: 10011, N: 1011, O: 000, P: 111100, Q: 001000, R: 0111, S: 1000, T: 1100, U: 11011, V: 001001, W: 111101, X: 001010, Y: 111110, Z: 001011

The WPL is 436.

3. Part2 的結果



The encoding result:

0000010101010101011100100100100100011011011011011101110100100100

100011100111111111111111

Decording result: iaaaaaaamhhhhannnnddsssomeeeee

Code list:

i: 0000, a: 01, m: 1100, h: 100, n: 101, d: 1101, s: 001, o: 0001, e: 111

The WPL is 88

4. 如何使用程式碼實作 huffman encoding 並得到 Part1 和 Part2 的結果

Part1:

(1) construct a structure for Huffman element, the struct data contain weight, lchild(leftchild), rchild(rightchild) and parent.

```
typedef struct huffNode {
    double weight;                  /*權重*/
    int lchild, rchild, parent;     /*左右子節點和父節點*/
} HTNode, * HuffTree;
```

(2) construct two match array, connect the char(26 English letter) and the

frequency(each letter appear times).

```
N = 26;  //共26字母
//第0個保留不用
ElemType data[N] = {"0", "A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z"};
//第0個保留不用
double w[N] = { 0,7,2,2,3,11,2,2,6,6,1,1,4,3,7,9,2,1,6,6,8,4,1,2,1,2,1 };
```

(3) construct the Huffman tree

```
void createHT(HuffTree& HT, HuffCode& HC, double* w, int n) {
    int s1, s2, m = 2 * n - 1;
    char* code;                                  //暫存
    HT = new HTNode[m + 1];                        //第0個不使用

    for (int i = 1; i <= n; i++) {
        //處理初始化前的第0個節點
        HT[i] = { w[i], 0, 0, 0 };
    }

    for (int i = n + 1; i <= m; i++) {
        //處理初始化後n-1個節點（找出最小兩節點的父節點）
        HT[i] = { 0, 0, 0, 0 };
    }

    //赫夫曼樹建構
    for (int i = n + 1; i <= m; i++) {
        //找出前i-1個節點中權值最小的節點
        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;
    }
```

```

    //赫夫曼編碼
    HC = new char* [n];
    /*這裡以下我真的不知道自己在幹嘛*/
    code = new char[n];


    for (int i = 1; i <= n; i++) {
        //k: 現在的節點，用0和1表示, f: k的父節點, j: 記錄編碼的位置
        int k = i, f = HT[k].parent, j = 0;
        //從葉子到根走一遍
        while (f != 0) {
            if (HT[f].lchild == k) {
                code[j] = '0';
            }
            else if (HT[f].rchild == k) {
                code[j] = '1';
            }
            k = HT[k].parent;
            f = HT[k].parent;
            j++;
        }
        //標記尾巴位置
        code[j] = '\0';
        reverseChars(code, j);
        //站存的編碼移到HC
        HC[i] = new char[n];
        strcpy(HC[i], code);
    }
}
```

(4) list the Huffman code

```
void showHuffCode(ElemType data[], HuffCode HC) {
    cout << "code list{" << endl;
    for (int i = 1; i <= N; i++) {
        cout << data[i] << ": " << HC[i] << "\n";
    }
    cout << "}" << endl;
}
```

(5) calculate the weighted path length(WPL) by DFS and show it.

```
int getWPL(HuffTree& HT, int idx, int depth) {
    //執行dfs直到遇到葉子
    if (HT[idx].lchild == 0 && HT[idx].rchild == 0) {
        return HT[idx].weight * depth;
    }
    return getWPL(HT, HT[idx].lchild, depth + 1) + getWPL(HT, HT[idx].rchild, depth + 1);
}
```

Part2:

(1)Initialize all the frequency=0

(2)Calculate the frequency of each different letter, and sorting them from low to high

```
void frequent(string str)
{
    int length = str.length();
    minnode* node = new minnode[length];     /*宣告最0節點*/
    int i, j;
    for (i = 0; i < length; i++)     /*初始化頻度*/
    {
        node[i].ch_num = 0;
    }
    int char_type_num = 0;  /*初始化為0種字元*/
    for (i = 0; i < length; i++)
    {
        for (j = 0; j < char_type_num; j++)
        {
            if (str[i] == node[j].ch || (node[j].ch >= 'a' && node[j].ch <= 'z' && str[i] + 32 == node[j].ch))
            {
                break;
            }
        }
        if (j < char_type_num)
        {
            node[j].ch_num++;
        }
        else
        {
            if (str[i] >= 'A' && str[i] <= 'Z')
            {
                node[j].ch = str[i] + 32;
```

```
            }
            else
            {
                node[j].ch = str[i];
            }
            node[j].ch_num++;
            char_type_num++;
        }
    }

    /*按照頻度從小到大排列*/
    for (i = 0; i < char_type_num; i++)
    {
        for (j = i; j < char_type_num; j++)
        {
            if (node[j].ch_num < node[j + 1].ch_num)  /*如果前一個小於後面一個 兩者交換*/
            {
                int temp;
                char ch_temp;
                temp = node[j].ch_num;
                ch_temp = node[j].ch;
                node[j].ch_num = node[j + 1].ch;
                node[j].ch = node[j + 1].ch;
                node[j].ch_num = temp;
                node[j].ch = ch_temp;
            }
        }
    }
```

(3)Initialize the nodes

```
    huffmanTree* huff = new huffmanTree[2 * char_type_num - 1];  /*位於確定char_type_num*/
    huffmanTree temp;
    string* code = new string[2 * char_type_num - 1];
    for (i = 0; i < 2 * char_type_num - 1; i++)    /*節點初始化*/
    {
        huff[i].parent = -1;
        huff[i].lchild = -1;
        huff[i].rchild = -1;
        huff[i].flag = -1;
    }
    for (j = 0; j < char_type_num; j++)   /*將排序後的第0個節點權重值賦予樹節點*/
    {
        huff[j].weight = node[j].ch_num;
    }
    int min1, min2;
    for (int k = char_type_num; k < 2 * char_type_num - 1; k++)   /*賦予0級以上的節點值*/
    {
        coding(length, huff, k, min1, min2);
        huff[min1].parent = k;
        huff[min2].parent = k;
        huff[min1].flag = "0";
        huff[min2].flag = "1";
        huff[k].lchild = min1;
        huff[k].rchild = min2;
        huff[k].weight = huff[min1].weight + huff[min2].weight;
    }
```

(4)Assign value to every node

```
for (i = 0; i < char_type_num; i++) {
    temp = huff[i];
    while (1) {
        code[i] = temp.flag + code[i];
        temp = huff[temp.parent];
        if (temp.parent == -1)break;
    }
}
```

(6) calculate the weighted path length(WPL) by DFS and show all the result.

```
int main(void)
{
    int length = 0;     /*字串長度*/
    string str;         /*目標字串*/
    cout << "Enter characters: ";
    cin >> str;
    frequent(str);     /*求各個字串頻度*/
    cout << endl;
    cout << "decoding result: "<<str;
    return 0;
}
```