1.1 编码实现

使用 Huffman 算法实现图片压缩程序，可分为 6 个步骤。

1、创建工程

创建 HuffmanCompressCPro 工程，定义入口函数 int main()；

2、读取原文件

读取文件，统计 256 种字节重复的次数；

3、生成 Huffman 树

根据上一步统计的结果，构建 Huffman 树；

4、生成 Huffman 编码

遍历 Huffman 树，记录 256 个叶子节点的路径，生成 Huffman 编码；

5、压缩编码

使用 Huffman 编码，对原文件中的字节重新编码，获得压缩后的文件数据；

6、保存文件

将编码过的数据，保存到文件“Pic.bmp.huf”中。

1.1.1 创建工程

启动 Microsoft Visual Studio 2010，创建一个空的解决方案，解决方案名为“HuffmanSLN”。在“HuffmanSLN”解决方案下面新建一个空的 Win32 控制台工程，工程名为“HfmCompressCPro”。

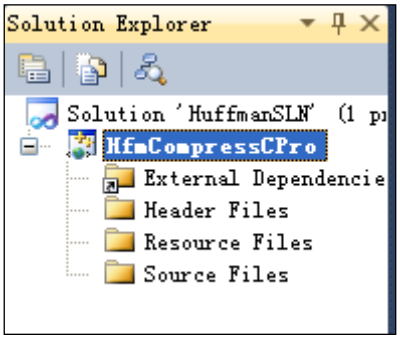

图 Error! No text of specified style in document.**-1 创建工程**

在工程中新建“Main.cpp”文件，创建 int main()函数作为程序运行的入口函数。

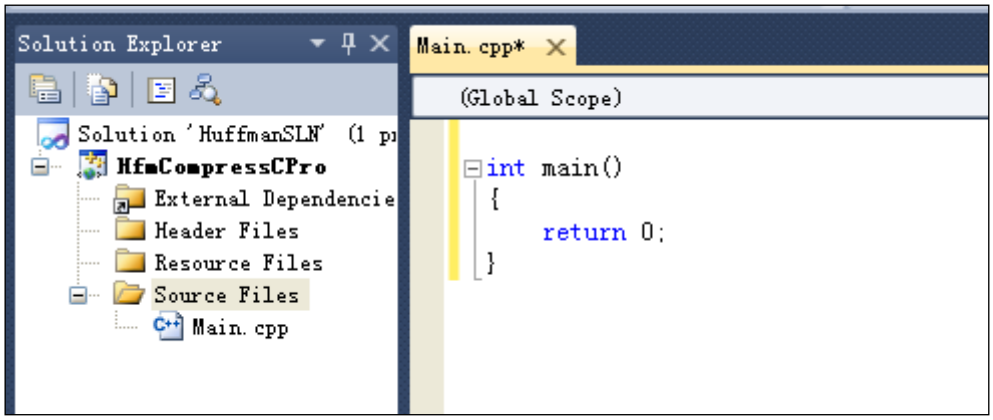

图 Error! No text of specified style in document.**-2 创建入口函数**

在”Main.cpp”文件中导入<iostream>头文件，声明 using namespace std。在 main()函数中使用 cout 输出界面提示信息，同时用 cin 接受用户从键盘输入的信息。

```
#include <iostream>
using namespace std;
int main()
{
    cout<<"========== Huffman 文件压缩 ==========="<<endl;
    cout<<"请输入文件名: ";
    char filename[256];
    cin>>filename;
     return 0;
}
```

运行结果：

图 Error! No text of specified style in document.**-3 运行结果**

1.1.2 读取原文件

以二进制流的方式，只读打开文件。逐字节读取文件数据，统计文件中 256 种字节重复的次数，保存到一个数组中 int weight[256]中。

```
#include <iostream>
#include <stdlib.h>
using namespace std;
int main()
{
```

```
        // ...

        int weight[256] = {0};
        // 以二进制流形式打开文件
        FILE *in = fopen(filename, "rb");
         …
        // 扫描文件，获得权重
        while((ch = getc(in)) != EOF)
        {
              weight[ch]++;
        }
        // 关闭文件
        fclose(in);
         return 0;
    }
```

将扫描的结果在控制台下打印出来。

```
    #include <iostream>
    #include <stdlib.h>
    using namespace std;
    int main()
    {
         // ...

         // 显示 256 种字节的出现的次数
         cout<<"Byte "<<"Weight"<<endl;
         for(int i = 0; i < 256; i++)
         {
               printf("0x%02X %d\n",i,weight[i]);
         }

          return 0;
    }
```

编译运行。

图 Error! No text of specified style in document.-4 运行结果

1.1.3 生成 Huffman 树

根据上一步中统计的结果，构建一棵 Huffman 树。定义一个结构体来记录每个节点的权值、父节点、左孩子和右孩子。使用结构体数组来存储这个 Huffman 树。

1、添加文件

右键单击工程，选择“Add -> New Item...”，新建 Huffman.h 和 Huffman.cpp 文件。我们将在 Huffman.h 文件中定义操作 Huffman 树的相关函数，然后在 Huffman.cpp 文件中实现这些函数。

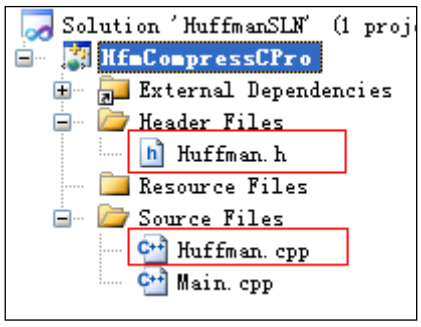

图 Error! No text of specified style in document.-5 添加文件

2、创建 Huffman 树

在 Huffman.h 文件中，定义一个用于存储 Huffman 树节点信息的结构体，该结构体中包含四个信息，分别是节点的权值、父节点、左孩子和右孩子。

这些信息为 Huffman 树的构造和遍历，以及 Huffman 编码提供了依据。

```
// Huffman 树节点
struct HTNode
{
    int weight;// 权值
    int parent;// 父节点
    int lchild;// 左孩子
```

```
        int rchild;// 右孩子
    };
```

在 Huffman.cpp 文件中，定义 HuffmanTree 函数，用于创建 Huffman 树。
具体的代码参见技术分析。

3、查找最小权值节点

在构建 Huffman 树的过程中，涉及到一个选择最小权值节点的问题，可以在 Huffman.h 文件中，定义一个 Select()函数，用来查找 Huffman 树结点数组中权值最小的节点，并在 Huffman.cpp 文件中去实现这个函数。

```
int Select(HuffmanTree pHT, int nSize)
{
    int minValue = 0x7FFFFFFF;// 最小值
    int min = 0;// 元素序号

    // 找到最小权值的元素序号
    for (int i = 1; i <= nSize; i++)
    {
        if (pHT[i].parent == 0 && pHT[i].weight < minValue)
        {
            minValue = pHT[i].weight;
            min = i;
        }
    }
    return min;
}
```

4、编译运行

编写测试函数 TestHufTree()，在控制台输出 Huffman 树每个节点的信息。

```
void TestHufTree(HuffmanTree pHT)
{
    // ..
    for(int i = 1; i < 512; i++)
    {
        printf("pHT[%d]\t\d\t\d\t\d\t\d\n",i, pHT[i].weight, pHT[i].parent,
pHT[i].lchild, pHT[i].rchild,);
    }
}
```

编译运行。

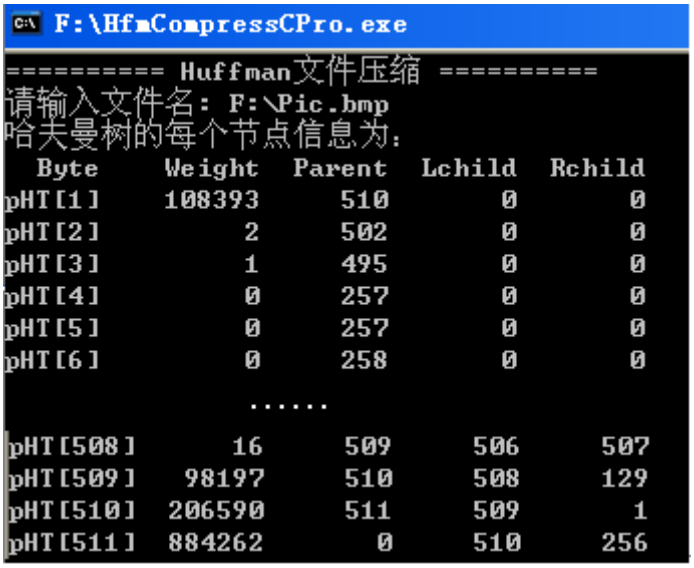

图 Error! No text of specified style in document.**-6 运行结果**

1.1.4 生成 Huffman 编码

遍历生成的 Huffman 树，记录每个叶子节点的路径，生成 Huffman 编码。在 Huffman.h 文件中定义 HuffmanCoding()函数，用来生成 Huffman 编码。定义一个字符串数组，用来保存 Huffman 编码表。

1、先序遍历

编写测试函数 TestHufTreeN()，采用先序遍历的方法，在控制台输出 Huffman 树每个节点的信息。先序遍历 Huffman 树，方法如下：

```
void TestHufTreeN(int root, HuffmanTree pHT)
{
    cout<<pHT[root].weight<<" ";
    if(pHT[root].lchild != 0)
    {
        TestHufTreeN(pHT[root].lchild, pHT);}
    }
    if(pHT[root].rchild != 0)
    {
        TestHufTreeN(pHT[root].rchild, pHT);
    }
}
```

编译运行。

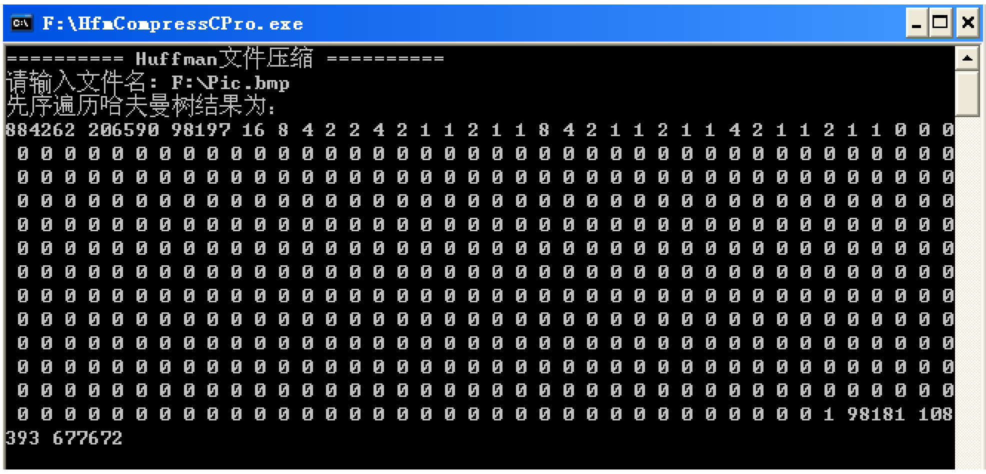

图 Error! No text of specified style in document.**-7** 运行结果

2、生成 Huffman 编码

遍历 Huffman 树时，要有记录访问每个叶子节点的路径，因此需要对先序遍历算法进行一些改进。

在 Huffman.cpp 文件中实现 HuffmanCoding()函数。

3、编译和运行

编写测试函数 TestHufCode()，测试输出每个叶子节点的编码。

```
void TestHufCode(int root, HuffmanTree pHT, HuffmanCode pHC)
{
    if(pHT[root].lchild == 0 && pHT[root].rchild == 0)
    {
        printf("0x%02X %s\n",i,root-1, pHC[root-1]);
    }
    if(pHT[root].lchild)// 访问左孩子
    {
        TestHufCode(pHT[root].lchild, pHT, pHC);
    }
    if(pHT[root].rchild)// 访问右孩子
    {
        TestHufCode(pHT[root].rchild, pHT, pHC);
    }
}
```

编译运行。

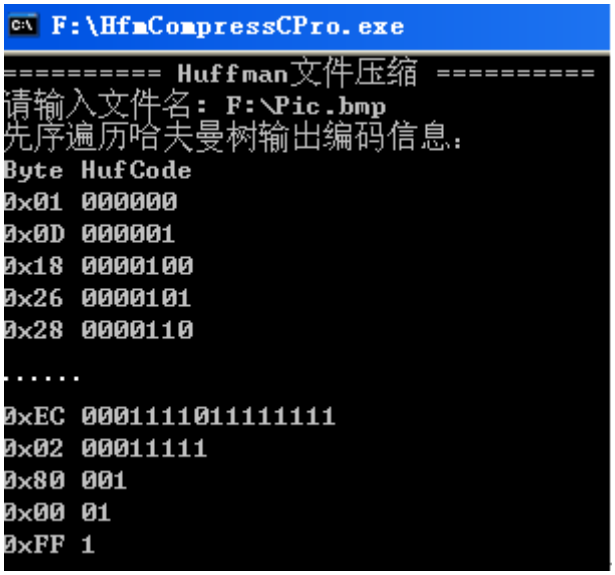

图 Error! No text of specified style in document.**-8 生成 Huffman 码运行结果**

1.1.5 压缩原文件

创建 Compress.h 和 Compress.cpp 文件，定义 Compress()函数用于压缩原文件。

创建 Encode 函数，以二进制流方式只读打开原文件，逐字节读取文件中的数据，使用 Huffman 编码对原文件中的字节重新编码。获得压缩后的文件数据，保存在一个 char 数组中。

1、获得 Huffman 编码

右键单击工程，选择“Add -> New Item...”，新建 Compress.h 和 Compress.cpp 文件。我们将在 Compress.h 文件中定义压缩文件的相关函数，然后在 Compress.cpp 文件中实现这些函数。

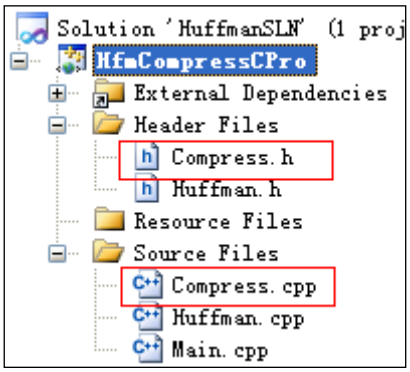

图 Error! No text of specified style in document.**-9 创建 Compress.h 和 Compress.cpp 文件**

在 Compress.h 文件中定义 Compress()函数，用来得到编码文件，然后在 Compress.cpp 文件中实现这个函数。

在 Compress 函数中将会调用 HuffmanTree()和 HuffmanCoding 两个函数来生成 Huffman 编码。

```
int Compress(const char *pFilename)
{
    //…
    HuffmanTree(pHT, weight);
     //…
    HuffmanCoding(pHC, pHT);
}
```

2、字符串转码

由于 Huffman 编码表是以字符数组的形式保存的，重新编码后的数据将是一个很长的字符串。定义 Str2byte 函数，将形如“01010101”字符串转换成字节，才能得到最终的编码。

在 Compress.h 文件中定义 Str2byte()函数用来将八个字符转换成一个字符，在 Compress.cpp 文件中实现这个函数。

```
char Str2byte(const char *pBinStr)
{
    char b = 0x00;
    for(int i = 0; i < 8; i++)
    {
        b = b<<1;// 左移 1 位
        if (pBinStr[i] == '1') {
            b = b | 0x01;
        }
    }
    return b;
}
```

3、编码原文件

创建 Encode 函数，以二进制流方式只读打开原文件，逐字节读取文件中的数据，使用 Huffman 编码对原文件中的字节重新编码。获得压缩后的文件数据，保存在一个 char 数组中。

(1) 计算压缩数据字节数

得到 Huffman 编码之后，需要计算编码完整个文件需要的空间大小，计算的方法为：先计算每个字符的权值乘以该字符编码的长度，这样即为同一个字符在编码文件中所占的大小，将 256 个这样的值相加即为编码整个文件所需的空间大小。除以 8 即为总字节数。

```
int Compress(const char *pFilename)
{
    // ...
    // 计算编码缓冲区大小
    int nSize = 0;
    for (int i = 0; i < 256; i++)
    {
        nSize += weight[i] * strlen(pHC[i]);
    }
    nSize = (nSize % 8) ? nSize / 8 + 1 : nSize / 8;
```

```
        // ...

    }
```

Encode 函数将根据字节数 nSize，创建一个 char 数组作为缓冲区，将压缩后的数据保存到缓冲区中。

(2) 压缩文件

一边读取一边查找 Huffman 编码表，将每个字符用相应的编码替换掉，这样就得到了重新编码后的文件。此时，我们会发现得到的新的文件比原来的文件还要大，所以我们需要对得到的编码进行压缩。

在 Compress.h 文件中定义 Encode()函数来实现压缩编码，在 Compress.cpp 文件中实现这个函数。

```
    int Encode(const char *pFilename, const HuffmanCode pHC, char *pBuffer, const int nSize)
    {
        // ...
        // 开辟缓冲区
        pBuffer = (char *)malloc(nSize * sizeof(char));
        if (!pBuffer)
        {
            cerr<<"开辟缓冲区失败"<<endl;
            return ERROR;
        }

    char cd[SIZE] = {0};// 工作区
    int pos = 0;// 缓冲区指针
    int ch;
        // 扫描文件，根据 Huffman 编码表对其进行压缩，压缩结果暂存到缓冲区中。
        while((ch = fgetc(in)) != EOF)
        {
            strcat(cd, pHC[ch]);// 从 HC 复制编码串到 cd

            // 压缩编码
            while(strlen(cd) >= 8)
            {
                // 截取字符串左边的 8 个字符，编码成字节
                pBuffer[pos++] = str2byte(cd);
                // 字符串整体左移 8 字节
                for (int i = 0; i < SIZE - 8; i++)
                {
                    cd[i] = cd[i+8];
                }
            }
```

```
        }
        if (strlen(cd) > 0)
        {
            pBuffer[pos++] = str2byte(cd);
        }
         ….
    }
```

(3) 编码文件

在 Compress()函数中，调用 Encode()函数，实现文件编码。编码后的数据保存在内存缓冲区中。

```
    int Compress(const char *pFilename)
    {
        // ...
        // 对原文件进行压缩编码
        char *pBuffer = NULL;
        Encode(pFilename, pHC, pBuffer, nSize);
        if (!pBuffer)
        {
            return ERROR;
        }
        // ...

    }
```

4、编译运行

在控制台下显示压缩结果。

图 Error! No text of specified style in document.**-10 压缩文件运行结果**

1.1.6 保存压缩文件

建立一个新的文件，文件名为“原文件名字+ .huf”，将压缩后的数据写入文件。

为了保证压缩后的数据，能够被重新正确解压，我们必须把相关的解压缩规则写入到文件中去。这里的解码规则实际上就是权值数组 weight，解压缩时只要根据权值信息表构建 Huffman 树，然后得到 Huffman 编码，就可以根据编码进行解压缩了。

为了验证解压缩的文件是否正确，还需要将文件的大小保存到文件中去，同时也可以将文件的类型存入文件中，用于判断是否是需要解压的文件，这里默认的是 HUF 类型。

因此，压缩文件中的数据由 2 部分组成：

(1) 文件头

包含文件类型、文件长度和权值数组

(2) 压缩数据

压缩原文件后得到的数据

1、初始化文件头

在 Compress.h 文件中定义一个文件头结构体，包含文件类型、原文件长度和权值数组三个信息。

```
// 文件头
struct HEAD
{
    char type[4];          // 文件类型
    int length;                 // 原文件长度
    int weight[256];     // 权值数值
};
```

在 Compress.h 文件中定义 InitHead()函数，并在 Compress.cpp 文件中实现它，用来扫描文件和初始化文件头的信息。

在扫描文件的时候，可以使用累加的方法，计算出原文件的大小。

```
int InitHead(const char *pFilename, HEAD &sHead)
{
    // 初始化文件头
    strcpy(sHead.type, "HUF");// 文件类型
    sHead.length = 0;// 原文件长度
    for (int i = 0; i < SIZE; i++)
    {
        sHead.weight[i] = 0;// 权值
    }
    // 以二进制流形式打开文件
    FILE *in = fopen(pFilename, "rb");
     …
    // 扫描文件，获得权重
    while((ch = fgetc(in)) != EOF)
    {
        sHead.weight[ch]++;
        sHead.length++;
```

```
        }
        // 关闭文件
        fclose(in);
        in = NULL;

        return OK;
    }
```

2、写文件

在得到了文件头信息，压缩后的编码之后，将这些信息写入到文件中去，即可得到压缩后的新的文件。在 Compress.h 文件中定义 WriteFile()函数，用来生成压缩文件，在 Compress.cpp 文件中实现函数功能。

```
    int WriteFile(const char *pFilename, const HEAD sHead, const BUFFER pBuffer, const int nSize)
    {
        // 生成文件名
        char filename[256] = {0};
        strcpy(filename, pFilename);
        strcat(filename, ".huf");

        // 以二进制流形式打开文件
        FILE *out = fopen(filename, "wb");
        …
        // 写文件头
        fwrite(&sHead, sizeof(HEAD), 1, out);
        // 写压缩后的编码
        fwrite(pBuffer, sizeof(char), nSize, out);
        // 关闭文件，释放文件指针
        fclose(out);
        out = NULL;

        cout<<"生成压缩文件："<<filename<<endl;
        int len = sizeof(HEAD) + strlen(pFilename) + 1 + nSize;
        return len;
    }
```

3、编译运行

当压缩文件成功时，会在控制台显示压缩信息，包括原文件名字和大小，压缩后的文件名和大小，以及压缩比率。

图 Error! No text of specified style in document.**-11 压缩文件显示信息**

1.1.7 扩展功能

1、Huffman 编码

在“6.4 生成 Huffman 编码”中，利用二叉树的特点，遍历二叉树可以得到所有叶子节点的编码。除此之外，还可利用结构体数组的特点，反过来从叶子节点寻找父节点，逆向记录访问路径，生成 Huffman 编码。

试用这种方法，按顺序生成 0x00~0xFF 每个字节所对应的 Huffman 编码。

2、文件解压

经过前面的开发，Pic.bmp 文件已经能够被压缩成 Pic.bmp.huf 文件了。那么这个压缩后的文件，要如何解压呢？

主要思路如下：

(1) 读取 Pic.bmp.huf 文件，扫描文件头。

(2) 获得文件头中的权值数组，生成 Huffman 树和 Huffman 编码表。

(3) 读取压缩包的数据体，查找 Huffman 编码表，将数据还原。

根据上述思路，请大家自行开发解压软件。

1.2 调试和运行

(1) 按 Ctrl+F5，编译运行程序。

(2) 输入文件名

若文件存放在 F 盘根目录下，输入文件完整路径“F:\Pic.bmp”。按回车结束。

图 Error! No text of specified style in document.**-12 输入文件名**

(3) 压缩文件

对文件进行压缩，控制台输出文件长度及压缩比。

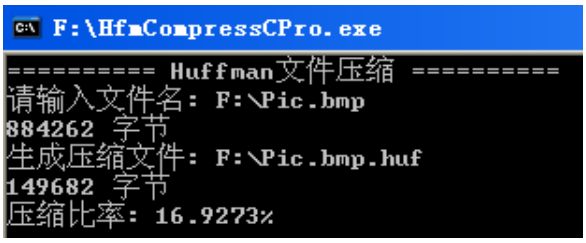

图 Error! No text of specified style in document.**-13 压缩文件**

压缩后生成“Pic.bmp.huf”文件，保存在与 Pic.bmp 相同的目录下。

图 Error! No text of specified style in document.**-14 压缩后生成 Pic.bmp.huf 文件**