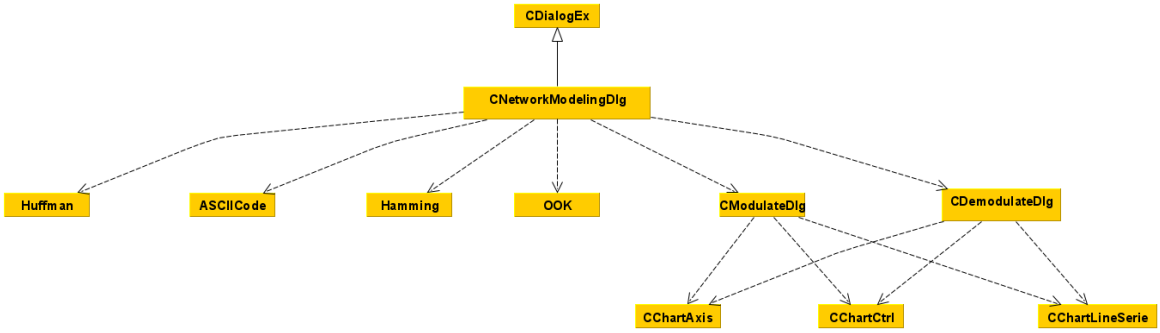
网络建模与分析大作业-代码说明文档

姓名：贾宏宇 学号：2019214512 班级：软硕192班

一、代码架构与运行细节

1.1 代码架构介绍



1. 首先，可视化界面的展示基于`CDialogEx`基类，具体细节由`CNetworkModelingDlg`类实现，该类实现了展示`NetworkModeling`对话框，该对话框是程序与用户交互的基础，用户通过在对话框中点击按钮、输入信息实现自己想要的功能，同时在该类中，保存了必需的变量，如`str_input`、`str_encode`、`m_period`等；命名了对应的点击函数，如`OnBnClickedButtonEncode()`、`OnBnClickedButtonCorrect()`等。
2. `CNetworkModelingDlg`类中包含编码、解码、调制、解调所必须的类函数，共包括三个编码解码类：`Huffman`、`ASCIICode`、`Hamming`和一个调制解调类`OOK`。
3. `CNetworkModelingDlg`类中包含绘图所使用的对话框类`CModulateDlg`、`CDemodulateDlg`这两个类分别负责绘制调制和解调的对话框；这两个类中使用`CChartCtrl`、`CChartAxis`和`CChartLineSerie`三个类绘制信道中传输的信号。
4. 通过上述的架构以及相应功能类的封装，使得代码逻辑性更清晰、鲁棒性更强。

1.2 运行细节介绍

程序的运行界面如图所示，其中运行的步骤和方式如下：

1. 在左上角的输入控件中输入字符串`input`。
2. 选择想要编码的方式：点选按钮`ASCII编码`、`Huffman编码`、`Hamming编码`三种中的任意一个。此时在左下角的输入控件中会出现编码的二进制字符串。
3. 【可选】输入调制的周期（默认16）
4. 【可选】点选`动态绘图`，则绘制出来的OOK调制信号会动态的进行更新；默认为不勾选该条件，此时生成的图像是静态的。
5. 点击`OOK调制`按钮，此时会弹出经过调制后的信号波形，同时中间偏下的输入控件会显示每个点的纵坐标。
6. 【可选】输入噪声方差，点击`添加噪声`按钮，此时会在生成的OOK图像中添加指定方差的高斯白噪声，同时弹出叠加后的图像。
7. 点击`OOK解调`，此时会弹出解调后的图像对话框，同时在右下角的输入控件中会显示解调后的二进制码，此时开发人员可以对比左下与右下角的二进制码，方便定位问题。
8. 点击对应的解码按钮，此时完成解码，解码后的字符串会输出到右上角的输入控件中，用户可以对比左上角与右上角的信息是否一致，从而判断网络通路中是否存在问题。

二、对话框实现细节与变量解释

首先，主对话框的绘制依赖于`CNetworkModelingDlg`类实现，在该类中封装了对应按钮的消息处理函数与需要的变量，当调用`UpdateData(TRUE)`函数时，会从对话框中获取值，更新类中的变量；当调用`UpdateData(FALSE)`函数时，会获取类中变量的值，更新对话框中的值。

其次，相关的消息处理函数与对应变量的命名如下：

```
// 输入原文
CString str_input;
// 进行二进制编码
CString str_encode;
// 调制后的信息
CString str_module;
// 二进制解码信息
CString str_decode;
// 输出解码后原文
CString str_output;
// 噪声方差
double sigma;
// 调制周期
int m_period;
// 封装好的编码、解码、调制、解调类
Huffman huffman;
ASCIICode asciiCode;
Hamming hamming;
OOK ook;
// 绘制调制解调的对话框
CModulateDlg* m_ModulateDlg[100];
int m_ModulateDlg_index;
CDemodulateDlg* m_DemodulateDlg[100];
int m_DemodulateDlg_index;
// 所有按钮的消息处理函数
afx_msg void OnBnClickedButtonEncode();
afx_msg void OnBnClickedButtonDecode();
afx_msg void OnBnClickedButtonAsciiEncode();
afx_msg void OnBnClickedButtonAsciiDecode();
afx_msg void OnBnClickedButtonCorrect();
afx_msg void OnBnClickedButtonDetect();
afx_msg void OnBnClickedButtonModule();
```

```
afx_msg void OnBnClickedButtonDemodule();
afx_msg void OnBnClickedButtonNoise();
```

三、霍夫曼编码实现细节

霍夫曼编码首先需要构建霍夫曼树，其中对树节点的成员变量做如下的定义，其中`ch`表示该节点代表的字符，`freq`保存了该字符出现的频率，`isLeaf`则表示当前节点是否为叶子节点。

```
typedef struct node{
    char ch;
    int freq;
    bool isLeaf;
    struct node* left;
    struct node* right;
}node;
```

其次，需要对霍夫曼树进行初始化，在构建树的过程中，需要从已有节点中找到两个出现频率最低的节点，将这两个节点进行组合，为了降低运行的复杂度，采用优先队列`priority_queue`实现，只要优先队列中节点个数大于2，就不断选出其中两个节点，组合成一个父节点然后再添加进优先队列中，通过此实现达到的时间复杂度为$O(nlgn)$，其中`n`为初始节点个数。

```
priority_queue<node> q;
while (q.size() > 1){
    node left = q.top();
    q.pop();
    node right = q.top();
    q.pop();
    root = new node();
    root->freq = left.freq + right.freq;
    root->left = new node(left); //&left;
    root->right = new node(right); //&right;
    q.push(*root);
}
```

然后，对构建好的霍夫曼树进行dfs遍历，使用dfs从根节点向下遍历时，会有一个字符串`str`保存所经过路线的二进制代码，当遍历到叶子节点时，就可以利用一个map来将编码后的前缀二进制字符串与相应字符对应起来。这样就完成了霍夫曼编码过程。

```
void Huffman::dfs(string str, node* tmp){
    if(tmp->isLeaf){
        code[tmp->ch] = str;
        return;
    }
    if(tmp->left){
        dfs(str + "0", tmp->left);
    }
    if(tmp->right){
        dfs(str + "1", tmp->right);
    }
}
```

最后对霍夫曼进行解码，解码过程，只需要提前保存好霍夫曼树的树结构，然后对于接收到的二进制串不断遍历即可，遇零则转向左子树、遇一则转向右子树。当然由于信道中噪声的存在，在解码过程中可能会报错，还做了相应的容错处理，详情见后文的容错机制。

四、基于正弦信号OOK调制的实现细节

4.1 OOK调制细节

首先，进行OOK调制之前，需要先将传输的信息编码为二进制字符串，然后根据指定的周期（默认为16）生成相应调制的信号，由于采用OOK编码，因此只需要对每一个二进制字符，乘以相应的正弦函数即可，将所有结果保存在一个`vector`中。

```
vector<double> res;
for (char c : str){
    ch = c - '0';
    for (int i = 1;i <= period;i++){
        res.push_back(ch * sin(2 * PI * i / period));
    }
}
```

其次，会将生成的`vector`传入`CModulateDlg`类，在这个类中进行OOK调制图像的绘制，在该类中主要的画图函数为`drawPicture`函数，该函数中使用`ChartCtrl`类绘图，实现了绘制坐标轴、绘制标题、更改外观设置、画图、监听鼠标位置显示坐标等功能。其中设置监听事件，是为了能够在鼠标移动到指定位置时，在对话框左下角显示鼠标的X轴、Y轴坐标。

```
void CModulateDlg::drawPicture(std::vector<double>& vec, bool dynamic){
    // 画坐标轴
    pAxis =
m_ChartCtrl_Modulate.CreateStandardAxis(CChartCtrl::BottomAxis);
    pAxis->SetAutomatic(true);
    // 略去部分代码......
    // 导入标题
    TChartString str1;
    str1 = _T("OOK 调制图像");
    m_ChartCtrl_Modulate.GetTitle()->AddString(str1);
    // 更改外观
    m_ChartCtrl_Modulate.GetTitle()->SetColor(RGB(255, 255, 255));   //标题
字体白色
    // 略去部分代码......
    // 开始绘画折线图
    m_ChartCtrl_Modulate.SetZoomEnabled(true);
    m_ChartCtrl_Modulate.RemoveAllSeries();//先清空
    pLineSerie = m_ChartCtrl_Modulate.CreateLineSerie();
    // 略去部分代码......
    pLineSerie->AddPoints(X1Values, Y1Values, size);
    // 设置鼠标监听事件
    CCustomCursorListener* m_pCursorListener;
    CChartCrossHairCursor* pCrossHair =
        m_ChartCtrl_Modulate.CreateCrossHairCursor();
}
```

然后，在绘图函数的实现过程中，实现了动态绘图机制，该机制使用定时器来实现，当用户点选动态绘图功能后，会建立一个`ID`为1的定时器，该定时器会不断地往图像中添加节点，知道编码数据全部加入到图像为止，此时该定时器会被注销。

```cpp
void CModulateDlg::OnTimer(UINT_PTR nIDEvent){
    pLineSerie->AddPoint(m_index, m_vec[m_index]);
    m_index++;
    if (m_index >= m_vec.size()){
        KillTimer(1);
    }
    CDialogEx::OnTimer(nIDEvent);
}
```

4.2 解调细节

首先，解调的实现封装在`OOK::demodulate`函数中，先对获取的信号进行乘以相同的正弦函数，然后使用滑动平均机制模拟一个简单的滤波机制，值得注意的是，滑动平均所计算的周期是调制解调使用正弦函数周期的一半，对于滑动平均计算后的值，选取每个周期的中点，判断中点处的值是否大于给定阈值，以此作为该周期信号解调后的二进制码。

其次，在绘制解调波形图的时候，实现过程于调制过程类似，就不过多赘述。

最后，在OOK类中添加了对信号`vector`进行编码的`encode`和`decode`函数，该函数主要是将调制后的信号编码为字符串形式，从而方便用户观察每个节点的值，也便于开发人员进行调试。

```cpp
string OOK::encode(vector<double>& vec){
    string res = "";
    stringstream stream;
    stream << fixed << setprecision(4) << vec[0];
    for (int i = 1;i < vec.size();i++){
        stream << "," << vec[i];
    }
    res = stream.str();
    return res;
}
```

五、信道中添加高斯白噪声的实现细节

添加高斯白噪声的细节主要封装在OOK类中的`addNoise`函数，该函数的输入是噪声方差，通过调用`random`库中的`default_random_engine`和`normal_distribution`结构来生成随机噪声。在实现的过程中，为了使得每一次添加的噪声相同，使得实验报告中的实验可重现，为随机数生成器指定了固定的随机数种子，当然，对于不同的编码方式和调制周期，由于生成信号的采样个数不同，所生成的随机数序列与结果并不能保证完全一致。

```cpp
vector<double> OOK::addNoise(vector<double>& vec, double sigma){
    unsigned seed = 1222;
    default_random_engine generator(seed);
    // 第一个参数为高斯分布的平均值，第二个参数为标准差
    std::normal_distribution<double> distribution(0.0, sigma);
    vector<double> res;
    for (int i = 0; i < vec.size();i++){
        res.push_back(vec[i] + distribution(generator));
    }
    return res;
}
```

六、汉明码实现细节

首先，汉明码是在ASCII编码或者霍夫丁编码后的二进制字符串基础上，通过增加冗余位（纠错位）来实现的。该编码首先通过公式$2^r \geq m + r + 1$计算需要添加的纠错位个数`r`，而后对于二的指数次幂的位置，在第$2^k(k = 0, 1, \dots r - 1)$位上进行编码，其余位填补实际的信息位。对于第$2^k$位上的二进制位，首先会对后$2^k$位进行异或操作，而后跳过$2^k$位，再对后$2^k$位异或，以此不断重复，直到编码字符串的末尾，其中实现伪代码如下。

```
string Hamming::encode(string& str){
    int m = str.length();
    int r = encodeLen(m); // 计算冗余位个数
    int n = m + r;
    string res = string(n, '0');
    在res中填充纠错位
    for (int i = 0;i < r;i++){
        异或操作计算纠错位编码值
    }
    return res;
}
```

其次，在对汉明码进行解码时，采用与编码相似的逆过程即可，先计算字符串中的纠错位个数，然后核验纠错位中的每一项是否为0，如果所有纠错位都为0，那么说明该汉明码无误，如果有任意纠错位出现一，则说明汉明码编码有误，此时根据纠错码的位置可以还原出错误编码的位置，将相应位置取反即可。如果还原出来的位置超出字符串长度，则说明该汉明码的出错个数较多，已经超出所能纠错的极限，应该注意，此汉明码最多只能纠正一位错误，具体实现的伪代码如下。

```
string Hamming::decode(string& str){
    int n = str.length();
    int r = decodeLen(n); // 还原冗余位个数
    判断汉明码是否出错;
    if (汉明码出错){
        还原出错位，将其取反;
    }
    return res;
}
```

最后，上述汉明码的实现方式可以为任意长度的二进制编码提供纠错检错能力，但是对于较长的二进制码，其编码的码率较低，因此最终实现时封装了两个(7, 4)汉明码，`encode74`和`decode74`，如果编码二进制串过长，则会将其切分成长度为4的字符串逐个进行编码，同样解码过程类似，会切分成长度为7的字符串逐个进行解码。同时该编码方式还提供了对应的容错机制，详见下一章。

七、容错机制

为了提高程序鲁棒性，我们编码、解码等过程中可能存在的错误进行了`try-catch`处理，并且写了相应的写日志函数。

- 霍夫曼解码过程：根据二进制串遍历子节点的过程中，可能会遇到遍历节点为空的情况，因此在其中加入了`try-catch`机制，检测到遍历节点为空，则会抛出错误，然后输出的结果不是解码后的信息，而是`"Huffman Decode error"`，从而给用户相应错误位置的提示。
- 汉明码解码过程：解码时会分析出错位的序号，如果该位置超过字符串长度，则说明解码时出现两位以上的错误，此时会返回`"Hamming Decode error"`，从而为用户提供相应错误提示。
- ASCII码解码过程：采用ASCII码进行编码时，每一个字符都可以表示成8位二进制符串，因此如果解码的字符串模8不为零，此时有编码的丢失，会返回`"ASCII code Error!"`，从而为用户提供相应错误提示。

  注：该ASCII编码解码过程，由于较为简单，因此没有在上文中单独写一章进行分析，详情可直接查看`ASCIICode.cpp`文件。
- 写日志操作：由于采用`MFC`开发，因此程序不能简单的在命令行中显示输入输出信息，而是需要写在相应的日志中，供开发人员查验，目前该函数实现在`ModulateDlg.cpp`文件中，由于程序已经调试完成，因此该接口没有再次进行调用，而是保留。

```
void WriteLog(CString strLog, CString strType){
    //log路径
    CString strPath =
CString("C:\\Users\\jiahy\\Desktop\\NetworkModeling\\Log.txt"
    CString strTime = CTime::GetCurrentTime().Format(L"[%Y-%m-%d
%H:%M:%S]");
    // 略去部分代码......
    FILE *fp = NULL;
    fopen_s(&fp, sPath, "at+");     //以文本形式追加
    if (NULL == fp)
    {
        return;
    }
    // 略去部分代码......
    fprintf(fp, "%s", Text);
    //除读写缓冲区，需要立即把输出缓冲区的数据进行物理写入时
    fflush(fp);
    fclose(fp);
}
```

日志输出如下所示：

```
[Print][2019-12-21 18:43:30] success

[Print][2019-12-21 18:43:33] success
```