综合编码 1：游程编码+Huffman 编码+加密编码+信道编码及对应的译码

L-D+Huffman 编码+加密编码+信道编码及对应的译码

1. **任务说明**

一个二元无记忆信源，0 符号的出现概率为 1/4, 1 符号的出现概率为 3/4。现要求对该信源连续出现的 n 个符号序列，进行游程编码/对游程编码结果进行 Huffman 编码（可用教材的 MH 编码方案）/使用加密算法进行加密/进行信道编码；然后模拟信道的传输过程，并对收到的信息串进行信道译码/解密译码/Huffman 译码/游程译码。

假定，连续出现的 0 或 1 序列的长度不超过 16，n 不小于 256。

其中加密编码/解密译码可自主选择对称加密算法或非对称算法；

信道编码要求采用(7,4)系统循环码,其中,g(x)= x3+x+1，译码采用简化的译码表；

信道为 BSC 信道，p=10-3

输入：长为 n 的 0/1 串

输出：1. 游程编码结果，2. Huffman 编码结果，3. 加密编码结果 4. 信道编码结果 5. 模拟接收串，6. 信道译码结果，7. 解密编码结果 8. Huffman 译码结果 9. 游程译码结果。

输入文件:in3.txt，一组输入即可

输出文件:out3.txt，对输入的处理结果

1. **问题分析、实现原理、流程图**
2. **问题分析**

游程编码相对简单，编码和译码容易实现。

重点在于霍夫曼树的构建和霍夫曼编码的生成，以及译码问题，构建霍夫曼树要注意设计适当的数据结构，对叶子节点的创建和权重的标记尤为关键。

信道编码采用（7，4）汉明系统循环码，纠错最小码距为3，纠错能力是1，传输模拟p=10-3的二元无记忆对称信道。

1. **实现原理**

**游程编码。**

对于二元相关信源，其输出只有两个符号，即"0"和"1”。在信源输出的一维二元序列中，连续出现"0"符号的这一段称为"0"游程，连续出现"1"符号的这一段称为"1"游程。对应段中的符号个数就是"0"游程长度和"1"游程长度，记为 L（0）和 L (1)。因为信源输出是随机的，所以游程长度是随机变量，其取值可为1、2、3……直至无穷值。又在输出的一维二元序列中，只出现有两种不同的字符也就是只有"0"游程和"1"游程，所以"0"游程和"1"游程总是交替出现。若规定二元序列总是从"0"游程开始，那么第一个为"0"游程，接着第二个必定是"1"游程，然后第三个又是"0”游程，游程交替出现。这样，我们就只需对串的长度（即游程长度）进行标记，这时，只需用一个表示串的长度的标志参量就行。然后就可将信源输出的任意一维二元序列一一对应地映射成交替出现的游程长度的标志序列。当然一般游程长度都用自然数标记，所以就映射成交替出现的表示游程长度的自然数序列，简称游程序列。这种映射是可逆的，是无失真的。例如某二元序列为：00010011111100000001... 可映射成游程序列：31267...

**霍夫曼编码。**

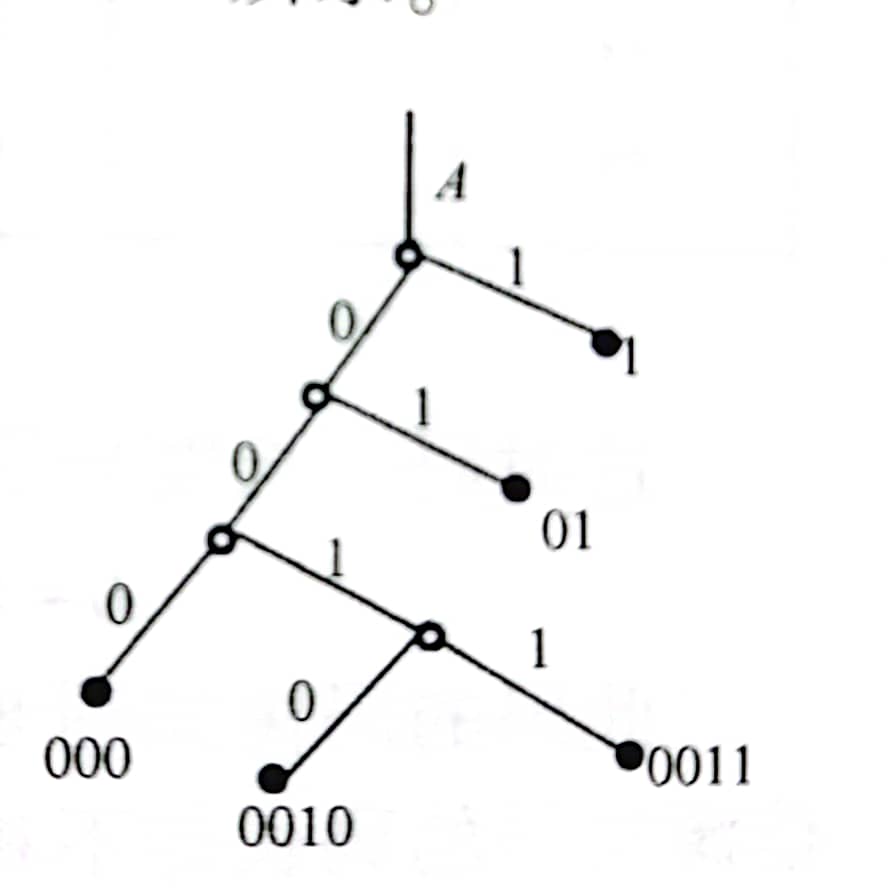
下面首先给出二元霍夫曼码的编码方法，它的编码步骤如下：

(1）将 q 个信源符号按概率分布 P (s)的大小，以递减次序排列起来，设

P1>=p2>=p3>=……p

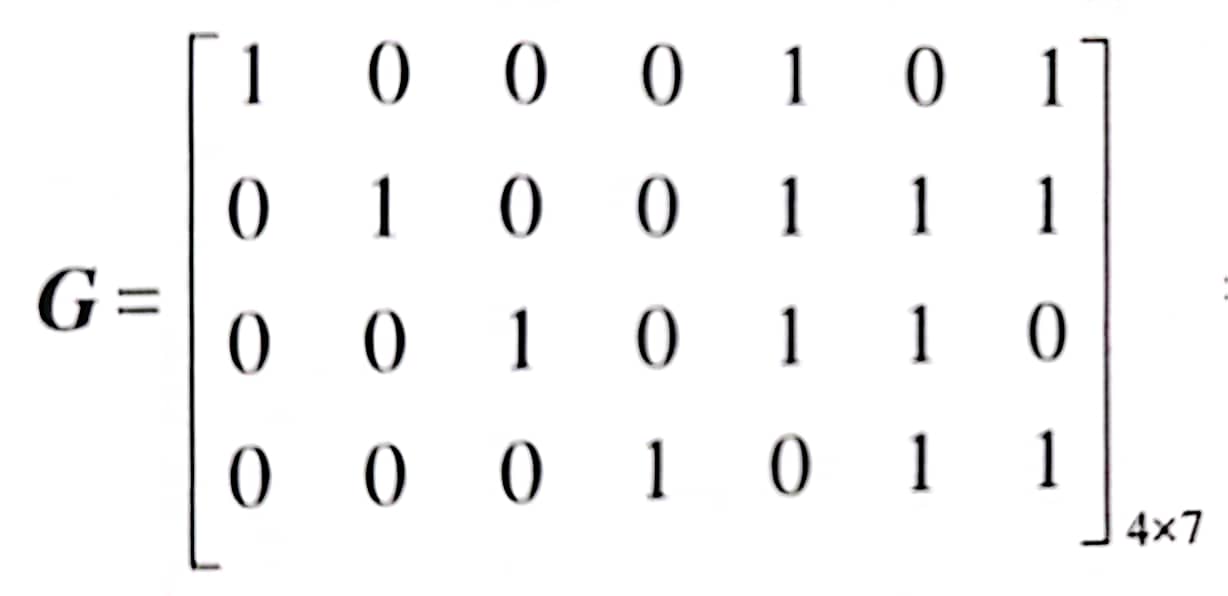
(2）用0和1码符号分别分配给概率最小的两个信源符号，并将这两个概率最小的信源符号合并成一个新符号，并用这两个最小概率之和作为新符号的概率，从而得到只包含q-1个符号的新信源，称为 S 信源的缩减信源 S1。

(3）把缩减信源 S1的符号仍按概率大小以递减次序排列，再将其最后两个概率最小的符号合并成一个新符号，并分别用0和1码符号表示，这样又形成了q-2个符号的缩减信源S2。

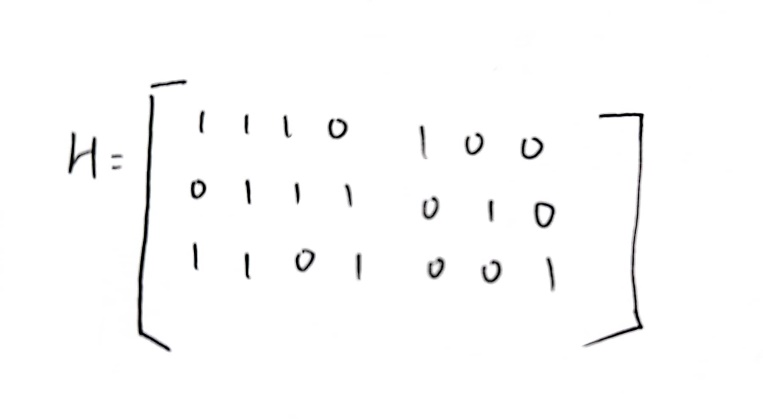
(4）依次继续下去，直至缩减信源最后只剩两个符号为止。将这最后两个符号分别用0和1码符号表示。最后这两个符号的概率之和必为1。然后从最后一级缩减信源开始，依编码路径由后向前返回，就得出各信源符号所对应的码符号序列，即得对应的码字。

一种霍夫曼编码的例子如图所示：

**信道编码：**

信道采用（7，4）系统循环码，又叫（7，4，3）循环汉明码，共有16个许用码字。

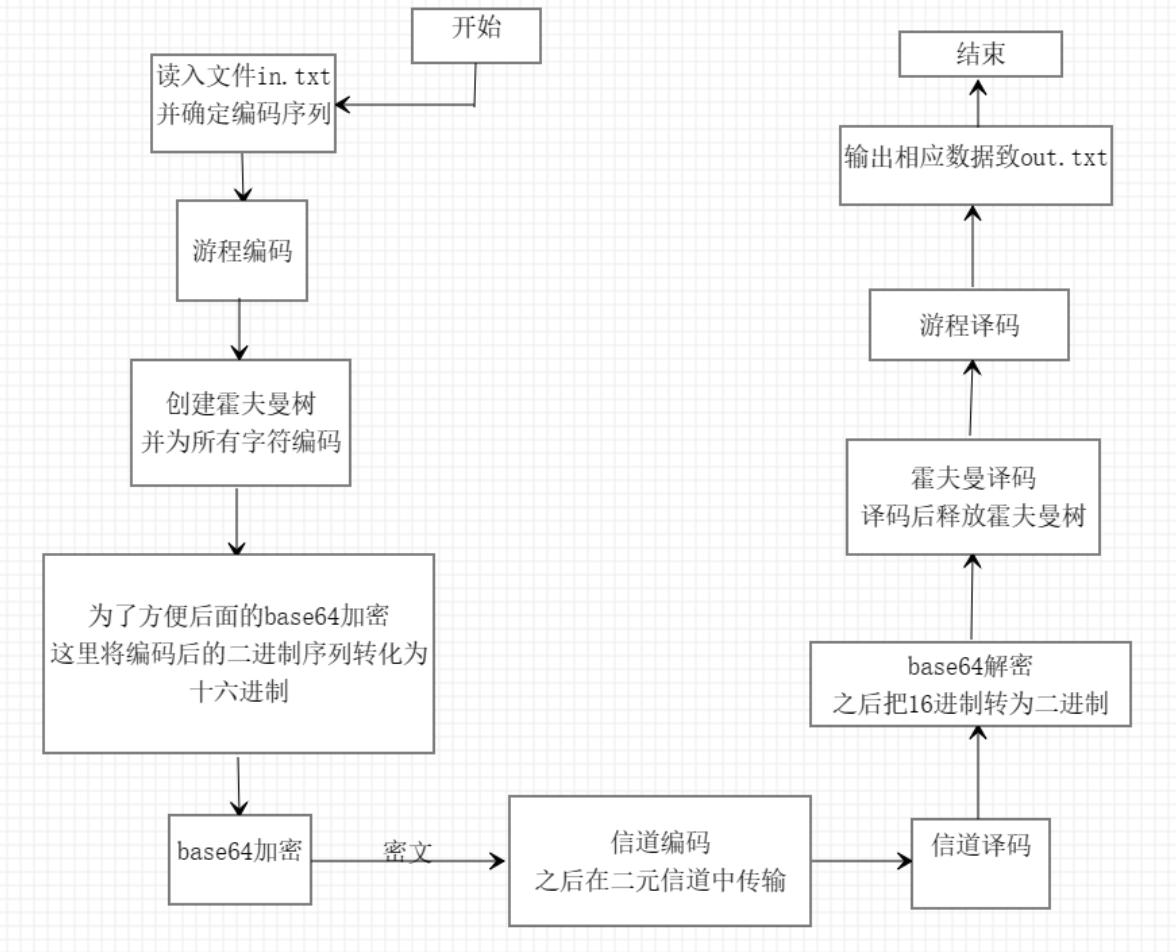
其标准生成矩阵为：

其一致监督矩阵为：

编码时，在算法实现时，采用信息位计算监督位，然后将两者组合的方法，用四个信息位计算出三个监督位之后，将三个监督位拼接在四个信息位的后面，译码时采用查表的方法。

**信道传输：**

使用二元对称无记忆离散信道，较为简单，不多赘述。

1. 流程图
2. **实现源码**

#include<iostream>

#include<string>

#include<stdio.h>

#include<stdlib.h>

#include<fstream>

#include<cctype>

#include<algorithm>

#include<cmath>

#include<vector>

#pragma warning(disable:4996)

#define N 200

#define M 2\*N-1

#define TOLOWER(p) {transform(p.begin(),p.end(),p.begin(),::tolower);}

typedef char\* HuffmanCode[2 \* M];//haffman编码

//typedef long long LL;

using namespace std;

char x[1024];//游程编码

char y[1024];//huff编码

fstream in("in3.txt");

fstream out("out3.txt");

int main()**//将主函数置顶，方便理解整个程序的流程**

{

char s[1024];

string input0;//输入数据

string input1;//二进制序列

string bin;//转换的二进制序列

string hex;//转换的十六进制序列

string deciToBin\_8bit(int);//函数声明，十进制转二进制

string base64\_encode(const string);

string base64\_decode(const string);//base64加密

if (!in.is\_open())

{

cout << "Error opening file"; exit(1);

}

while (!in.eof()) {

in.getline(s, 1024);

cout << s << endl;

cout << endl;

run\_encode(s);

out << "游程编码后：" << x << endl;

cout << endl;

huffman\_encode(x);

out << "Huffman编码后：" << y << endl;

cout << endl;

hex = BinStrToHexStr(y);//二转十六

cout << "Bin\_to\_Hex:" << hex << endl;

out << "Bin\_to\_Hex:" << hex << endl;

cout << endl;

input0 = hex;

for (auto& x : input0)input1 += deciToBin\_8bit(static\_cast<int>(x));//二进制序列

string out0 = base64\_encode(input1);

cout << "base64加密后：" << out0 << endl;

out << "base64加密后：" << out0 << endl;

cout << endl;

create\_G();//生成矩阵

create\_S();//译码表

int lenth = out0.length();

string\* a = new string[lenth];

string\* output = new string[lenth];

string\_to\_ascii(out0, a);//密文转化为ASCII码串

string output1 = "";

//其实这里一个循环就能完成所有工作，但是设计信道函数的时候没有考虑输出的问题，所以为了输出信道编码和译码后的结果

//只能将一个循环拆成三个

/\*for (int i = 0; i < lenth; i++)

{

int f = 0;

output[i] = channel\_encode(a[i]);

output[i] = BSC(output[i]);

output[i] = channel\_decode(output[i]);

f = bin\_to\_deci(output[i]);

output1 += (char)f;

}\*/

cout << "信道编码：";

for (int i = 0; i < lenth; i++)

{

output[i] = channel\_encode(a[i]);//信道编码

cout << output[i];

}

cout << endl; cout << endl; cout << "模拟传递串：";

for (int i = 0; i < lenth; i++)

{

output[i] = BSC(output[i]);//信道传输

cout << output[i];

}

cout << endl; cout << endl; cout << "信道译码：";

for (int i = 0; i < lenth; i++)

{

output[i] = channel\_decode(output[i]);//信道译码

cout << output[i];

int f = 0;

f = bin\_to\_deci(output[i]);//将二进制转成十进制

output1 += (char)f;//变成ASCII码对应的字符，即base64的密文

}cout << endl; cout << endl;

string out1 = base64\_decode(output1);

cout << "base64解密后：" << out1 << endl;

out << "base64解密后：" << out1 << endl;

cout << endl;

bin = HexStrToBinStr(out1);

cout << "Hex\_to\_Bin:" << y << endl;

out << "Hex\_to\_Bin:" << y << endl;

cout << endl;

strcpy(y, bin.c\_str());

out << "huffman译码后：";

huffman\_decode();

cout << endl;

run\_decode(x);

}

in.close();

out.close();

return 0;

}

**//以下是其他函数**

void run\_encode(char a[1024])//游程编码

{

char yc[1024];

int i = 0, j = 0, jishu = 1;

yc[0] = a[0];

for (i = 0; a[i] != '\0'; i++)

{

if (a[i] == a[i + 1])

jishu++;

else

{

yc[j + 1] = jishu + 48;

j = j + 2;

yc[j] = a[i + 1];

jishu = 1;

}

}

yc[j] = '\0';

cout << "游程编码后：" << yc << endl;

strcpy\_s(x, yc);

}

void run\_decode(char a[1024])//游程译码

{

char jieya[1024];

int j, jishu, bz = 0;

for (int i = 0; a[i] != '\0'; i = i + 2)

{

jieya[bz] = a[i];

for (j = bz, jishu = 1; jishu <= a[i + 1] - 48; jishu++, j++)

jieya[j] = a[i];

bz = j;

}

jieya[j] = '\0';

cout << "游程译码后：" << jieya << endl;

out << "游程译码后：" << jieya << endl;

}

typedef struct//霍夫曼树的数据结构定义

{

int weight;//权值

int parent;//父节节点

int LChild;//左子节点

int RChild;//右子节点

}HTNode, Huffman[M + 1];//huffman树

typedef struct Node

{

int weight; //叶子结点的权值

char c; //叶子结点

int num; //叶子结点的二进制码的长度

}WNode, WeightNode[N];

void CreateWeight(char ch[], int\* s, WeightNode CW, int\* p)//产生叶子结点的字符和权值

{

int i, j, k;

int tag;

\*p = 0;//叶子节点个数

//统计字符出现个数,放入CW

for (i = 0; ch[i] != '\0'; i++)

{

tag = 1;

for (j = 0; j < i; j++)

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

{

tag = 0;

break;

}

if (tag)

{

CW[++ \* p].c = ch[i];

CW[\*p].weight = 1;

for (k = i + 1; ch[k] != '\0'; k++)

if (ch[i] == ch[k])

CW[\*p].weight++;//权值累加

}

}

\*s = i;//字符串长度

}

void CreateHuffmanTree(Huffman ht, WeightNode w, int n)//创建霍夫曼树

{

int i, j;

int s1, s2;

//初始化哈夫曼树

for (i = 1; i <= n; i++)

{

ht[i].weight = w[i].weight;

ht[i].parent = 0;

ht[i].LChild = 0;

ht[i].RChild = 0;

}

for (i = n + 1; i <= 2 \* n - 1; i++)

{

ht[i].weight = 0;

ht[i].parent = 0;

ht[i].LChild = 0;

ht[i].RChild = 0;

}

for (i = n + 1; i <= 2 \* n - 1; i++)

{

for (j = 1; j <= i - 1; j++)

if (!ht[j].parent)

break;

s1 = j; //找到第一个双亲为零的结点

for (; j <= i - 1; j++)

if (!ht[j].parent)

s1 = ht[s1].weight > ht[j].weight ? j : s1;

ht[s1].parent = i;

ht[i].LChild = s1;

for (j = 1; j <= i - 1; j++)

if (!ht[j].parent)

break;

s2 = j; //找到第二个双亲为零的结点

for (; j <= i - 1; j++)

if (!ht[j].parent)

s2 = ht[s2].weight > ht[j].weight ? j : s2;

ht[s2].parent = i;

ht[i].RChild = s2;

ht[i].weight = ht[s1].weight + ht[s2].weight;//权值累加

}

}

void CrtHuffmanNodeCode(Huffman ht, char ch[], HuffmanCode h, WeightNode weight, int m, int n)//对叶子结点编码

{

int i, c, p, start;

char\* cd;

cd = (char\*)malloc(n \* sizeof(char));

cd[n - 1] = '\0';//置于末尾

for (i = 1; i <= n; i++)

{

start = n - 1; //cd串每次从末尾开始

c = i;

p = ht[i].parent;//p在n+1至n-1

while (p) //沿父亲方向遍历,直到为

{

start--;//依次向前置值

if (ht[p].LChild == c)//与左子相同,置

cd[start] = '0';

else//否则置

cd[start] = '1';

c = p;

p = ht[p].parent;

}

weight[i].num = n - start; //二进制码的长度(包含末尾)

h[i] = (char\*)malloc((n - start) \* sizeof(char));

strcpy(h[i], &cd[start]);//将二进制字符串拷贝到指针数组h中

}

free(cd);//释放cd内存

//system("pause");

}

void CrtHuffmanCode(char ch[], HuffmanCode h, HuffmanCode hc, WeightNode weight, int n, int m)//对所有字符的编码

{

int i, k;

for (i = 0; i < m; i++)

{

for (k = 1; k <= n; k++) /\*从weight[k].c中查找与ch[i]相等的下标K\*/

if (ch[i] == weight[k].c)

break;

hc[i] = (char\*)malloc((weight[k].num) \* sizeof(char));

strcpy(hc[i], h[k]); //拷贝二进制编码

}

}

void TrsHuffmanTree(Huffman ht, WeightNode w, HuffmanCode hc, int n, int m)//霍夫曼树解码

{

int i = 0, j, p;

while (i < m)

{

p = 2 \* n - 1;//从父亲节点向下遍历直到叶子节点

for (j = 0; hc[i][j] != '\0'; j++)

{

if (hc[i][j] == '0')

p = ht[p].LChild;

else

p = ht[p].RChild;

}

printf("%c", w[p].c); /\*打印原信息\*/

out << w[p].c;

i++;

}

out << endl;

}

void FreeHuffmanCode(HuffmanCode h, HuffmanCode hc, int n, int m)//当霍夫曼树使用完成之后，释放huffman编码内存

{

int i;

for (i = 1; i <= n; i++)//释放叶子结点的编码

free(h[i]);

for (i = 0; i < m; i++) //释放所有结点的编码

free(hc[i]);

}

int n; //n为叶子结点的个数

int m; //m为字符串ch[]的长度

Huffman ht; //Huffman二叉树

HuffmanCode h, hc; //h存放叶子结点的编码，hc 存放所有结点的编码

WeightNode weight; //存放叶子结点的信息

void huffman\_encode(char\* ch)//霍夫曼编码函数

{

n = 0;

int i;

m = 0;

CreateWeight(ch, &m, weight, &n); //\*产生叶子结点信息，m为字符串ch[]的长度

CreateHuffmanTree(ht, weight, n); //产生Huffman树

CrtHuffmanNodeCode(ht, ch, h, weight, m, n); //叶子结点的编码

CrtHuffmanCode(ch, h, hc, weight, n, m); //所有字符的编码

cout<<"Huffman编码后"; //打印字符串的编码

for (i = 0; i < m; i++)

{

cout<< hc[i];

strcpy(&y[i], hc[i]);

}

cout << endl;

//system("pause");

}

void huffman\_decode()

{

cout<<"huffman译码后：";

TrsHuffmanTree(ht, weight, hc, n, m); //解码

FreeHuffmanCode(h, hc, n, m);

cout << endl;

//system("pause");

}

string BinStrToHexStr(string str)//这里是将霍夫曼树译码后的二进制字符串转化为十六进制，方便base64编码

{

string str\_Hex = "";

string temp\_Hex = "";

int iHex = 0;

if (str.size() % 4 != 0)

{

int num = 4 - (str.size() % 4);

for (int i = 0; i < num; i++)

{

str = "0" + str;

}

}

for (int i = 0; i < str.size(); i += 4)

{

iHex = (str[i] - '0') \* 8 + (str[i + 1] - '0') \* 4 + (str[i + 2] - '0') \* 2 + (str[i + 3] - '0') \* 1;

if (iHex >= 10)

{

int left = iHex % 16;

temp\_Hex = 'A' + left - 10;

}

else

{

temp\_Hex = to\_string(iHex);

}

str\_Hex += temp\_Hex;

}

return str\_Hex;

}

string HexStrToBinStr(string str)//base64译码之后的十六进制转成二进制方便霍夫曼译码

{

string str\_Bin = "";

int iDec = 0;

TOLOWER(str);

sscanf\_s(str.c\_str(), "%x", &iDec);

vector<int>vec\_remder;

int remder = 0;

while (iDec / 2 != 0)

{

remder = iDec % 2;

vec\_remder.push\_back(remder);

iDec /= 2;

}

remder = iDec % 2;

vec\_remder.push\_back(remder);

reverse(vec\_remder.begin(), vec\_remder.end());

for (int i = 0; i < vec\_remder.size(); i++)

{

str\_Bin += to\_string(vec\_remder[i]);

}

return str\_Bin;

}

string deciToBin\_8bit(int deci) //十进制转二进制（八位二进制）

{

string hexStr(8, ' ');

int Value = 0;

int i = 0;

for (; deci != 0; ++i, deci /= 2)

{

Value = deci % 2;

hexStr.at(i) = Value + '0';

}

hexStr = hexStr.substr(0, i);

reverse(hexStr.begin(), hexStr.end());

if (hexStr.length() < 8)

{

string add(8 - hexStr.length(), '0');

hexStr = add + hexStr;

}

return hexStr;

}

string deciToBin\_6bit(int deci)//10进制转2进制函数（6位）

{

string hexStr(6, ' ');

int Value = 0;

int i = 0;

for (; deci != 0; ++i, deci /= 2)

{

Value = deci % 2;

hexStr.at(i) = Value + '0';

}

hexStr = hexStr.substr(0, i);

reverse(hexStr.begin(), hexStr.end());

if (hexStr.length() < 6)

{

string add(6 - hexStr.length(), '0');

hexStr = add + hexStr;

}

return hexStr;

}

string base64\_encode(const string in)

{

int i = 0, k = 0;//i总变量 k24位控制变量

const int n = in.length();

string subs;//输出子串

string out;//结果

string value("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/");

while (n - i >= 24)

{

for (k = 0; k < 4; ++k)

{

subs = in.substr(i + k \* 6, 6);

int sum = 0;

int j = 0;

for (auto& x : subs)

{

sum += (x - '0') \* pow(2, 5 - j);

++j;

}

out += value.at(sum);

}

i += 24;

if (i % 76 == 0)cout << endl;

}

if (i != n)//处理剩下的不足24位部分

{

int charlen; //剩下的长度

if ((n - i) % 6 == 0)charlen = (n - i) / 6;

else charlen = (n - i) / 6 + 1;

for (k = 0; k < charlen; ++k)

{

subs = in.substr(i + k \* 6, i + (k + 1) \* 6 - 1); //取出6位

int sum = 0;

int j = 0;

for (auto& x : subs)

{

sum += (x - '0') \* pow(2, 5 - j);

++j;

} //得到下标

out += value.at(sum); //输出对应字符

}

if ((n - i) % 16 == 0)out += "=";

else out += "==";

}

return out;

}

string base64\_decode(const string in)

{

string value("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/");

string out0;

string out1;

string deciToBin\_6bit(int);

for (auto x = in.cbegin(); (\*x) != '=' && x != in.cend(); ++x) //排除等号

out0 += deciToBin\_6bit(value.find(\*x));

const int n = out0.length();

for (int i = 0; i < n; i += 8) //每8位提取一次

{

string subs = out0.substr(i, 8); //提取8位

int j = 0;

int sum = 0;

for (auto& x : subs)

{

sum += (x - '0') \* pow(2, 7 - j);

++j;

} //按权展开，得到ASCII码

out1 += static\_cast<char>(sum); //转换为相应字符

}

return out1;

}

int G[4][3];//这里是标准生成矩阵，所以省略了前四列的单位阵,此矩阵4行3列

string S[8][16];//这里是译码表，由于（7，4）系统循环码最小码距d=3，纠错能力t=1；故我这里只有8行

void create\_G()

{

//初始化生成矩阵，同时，生成阵的转置为监督阵

G[0][0] = 1; G[0][1] = 0; G[0][2] = 1;//第一行 1 0 1

G[1][0] = 1; G[1][1] = 1; G[1][2] = 1;//第二行 1 1 1

G[2][0] = 1; G[2][1] = 1; G[2][2] = 0;//第三行 1 1 0

G[3][0] = 0; G[3][1] = 1; G[3][2] = 1;//第四行 0 1 1

}

void create\_S()

{

//初始化译码表第一行,第一行都为需用码字

S[0][0] = "0000000"; S[0][1] = "1111111"; S[0][2] = "0001011"; S[0][3] = "0010110";

S[0][4] = "0101100"; S[0][5] = "1011000"; S[0][6] = "0110001"; S[0][7] = "1100010";

S[0][8] = "1000101"; S[0][9] = "0011101"; S[0][10] = "0111010"; S[0][11] = "1110100";

S[0][12] = "1101001"; S[0][13] = "1010011"; S[0][14] = "0100111"; S[0][15] = "1001110";

for (int i = 0; i < 7; i++)

{

for (int j = 0; j < 16; j++)//循环生成所有可以纠错的图样

{

int x = 0; string s = "", befor = "", after = "";

befor = S[0][j].substr(0, i);

after = S[0][j].substr(i + 1, (S[0][j].length()) - i - 1);

s = S[0][j].substr(i, 1);

x = stoi(s);//string转int

x = (x + 1) % 2;

s = std::to\_string(x);//int转string

S[i + 1][j] = befor + s + after;

}

}

}

int bin\_to\_deci(string inin)//二进制字符串转10进制，主要是使信道译码之后的字符串变成ASCII码的字符串，方便转换为base64密文

{

int len = inin.length();

int n = 0;

for (int i = 0; i < len; ++i)

{

//其中注意if的判断两个条件相等的符号

if (inin.substr(i,1) == "1")

n += pow(2, len - 1 - i);

}

return n;

}

string channel\_encode(string infomation)//信道编码，这里的infomation是8位二进制

{

string info, e\_info = "", v1, v2, v3, v4;

int n1 = 0, n2 = 0, n3 = 0;

for (int i = 0; i < 2; i++)

{

if (i == 0)info = infomation.substr(0, 4);

if (i == 1)info = infomation.substr(4, 4);

v1 = info.substr(0, 1);

v2 = info.substr(1, 1);

v3 = info.substr(2, 1);

v4 = info.substr(3, 1);

if (v1 == "1") { n1 += G[0][0]; n2 += G[0][1]; n3 += G[0][2]; }

if (v2 == "1") { n1 += G[1][0]; n2 += G[1][1]; n3 += G[1][2]; }

if (v3 == "1") { n1 += G[2][0]; n2 += G[2][1]; n3 += G[2][2]; }

if (v4 == "1") { n1 += G[3][0]; n2 += G[3][1]; n3 += G[3][2]; }

n1 = n1 % 2; n2 = n2 % 2; n3 = n3 % 2;

v1 = std::to\_string(n1); v2 = std::to\_string(n2); v3 = std::to\_string(n3);

e\_info = e\_info + info + v1 + v2 + v3;

n1 = 0; n2 = 0; n3 = 0;

}

return e\_info;//输出14位，也就是两个码字合并

}

string channel\_decode(string infomation)//输入14位，两个码字

{

string info, de\_info = "";

for (int x = 0; x < 2; x++)

{

if (x == 0)info = infomation.substr(0, 7);

if (x == 1)info = infomation.substr(7, 14);

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 16; j++)

{

if (S[i][j] == info)

{

de\_info += info.substr(0, 4);

goto goon;

}

}

}

goon:;

}

return de\_info;//8位，包含两个四位信息源，其余六位校验位已经删去

}

string BSC(string infomation)//模拟BSC信道传输

{

string output = "", x;

for (int i = 0; i < infomation.length(); i++)//一个字符一个字符判断

{

x = infomation.substr(i, 1);

if (x == "0")

{

if (10 == rand() % 1000)//p=10的负3次

x = "1";

goto gon;//防止此处x重新赋值为1后进入到下一个判断

}

if (x == "1")

{

if (100 == rand() % 1000)//p=10的负3次

x = "0";

goto gon;

}

gon:output += x;

}

return output;

}

void string\_to\_ascii(string info, string a[])//把base64密文转化位ASCII码方便信道编码

{

string output = "", x;

char c; int n;

for (int i = 0; i < info.length(); i++)

{

c=info[i];

n = (int)c;

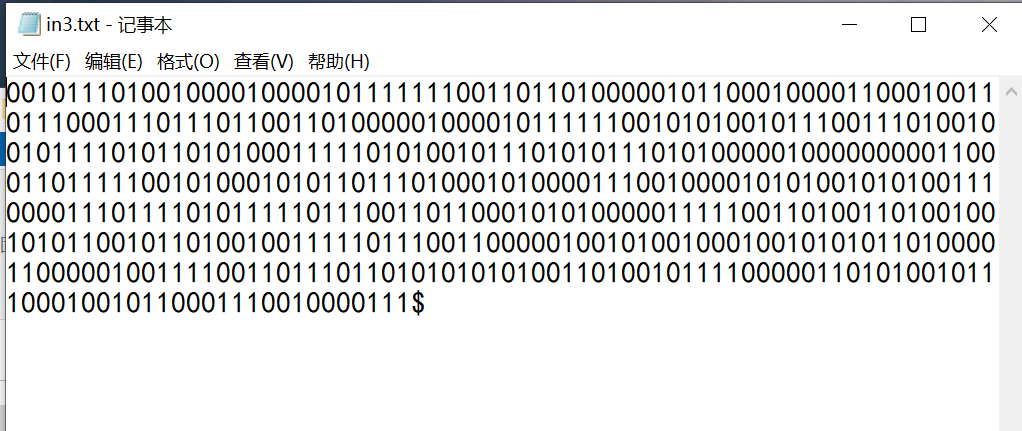
output = deciToBin\_8bit(n);

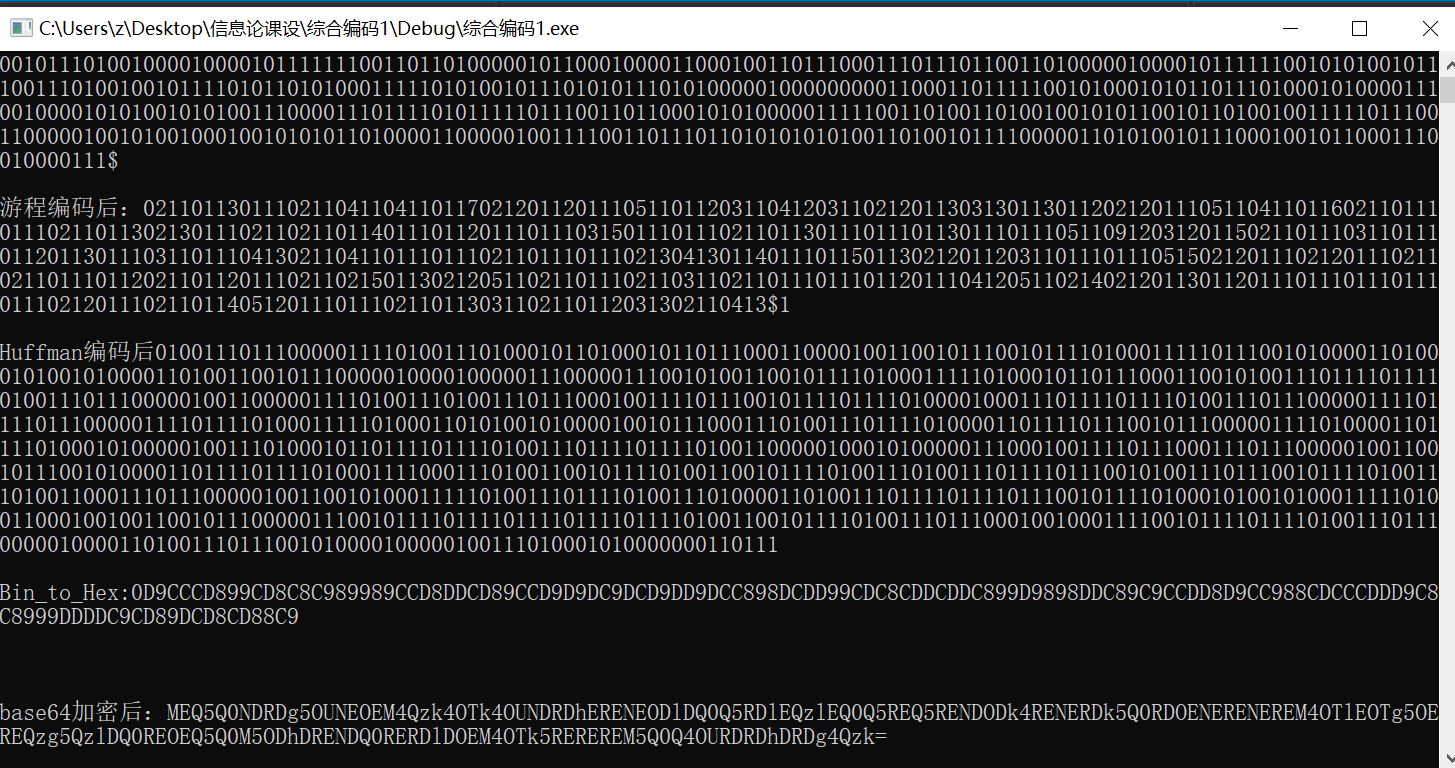
a[i] = output;

}

}

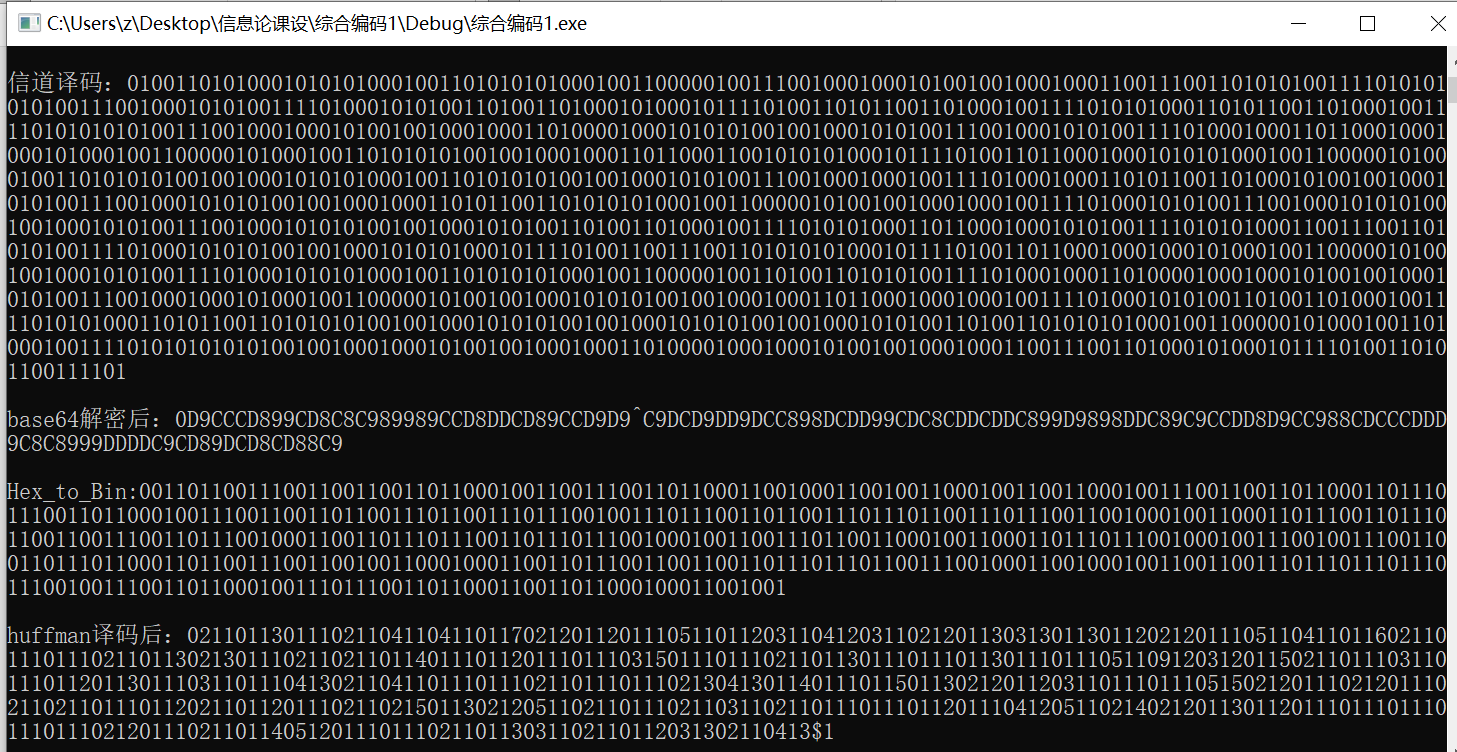
1. **运行结果：输入、输出及结果分析**

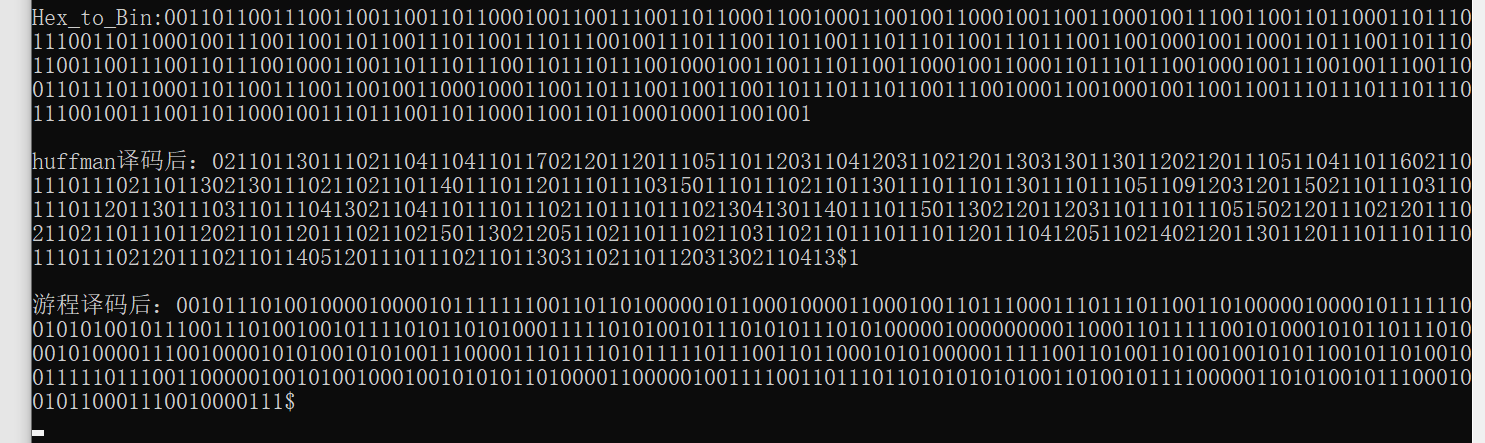
****输入01序列，约为490位。

程序运行截图

****

****

****

****

****

运行结果符合预期，各项功能均正确，编码译码功能正常

1. **设计体会**

通过这次课程设计深入了解了游程编码，霍夫曼编码，信道编码和信道传输的原理，深入了解了霍夫曼树和其叶子节点的生成，以及权值的赋值，编码和译码的原理，都有了更深入的理解。

在这段时间里，极大的提升了我的编程能力和查询资料自学的能力，也建立了综合编码的思想，更加接近实际生产生活，对今后的学习和工作大有裨益，此次课设，收获颇丰。