Huffman 编解码算法实现与压缩效率分析

1）Huffman 编码的数据结构

```
Typedef    struct    huffman_node_tag        //Huffman 节点标签
{
    unsigned char isLeaf            //是否为树叶
    unsigned long count;            //节点代表的符号加权和
    struct huffman_node_tag *parent;       //父节点指针
    union
    {
        struct
        {
           struct huffman_node_tag *zero, *one;     //子节点指针,分别代表 0,1 子节点指针
        };
        unsigned char symbol;   //节点代表的符号
    };
} huffman_node;

typedef    struct    huffman_code_tag        //Huffman 编码标签
{
    unsigned long numbits;          //该码所用的比特数
    unsigned char *bits;            //指向该码比特串的指针
} huffman_code;
```

2）Huffman 编码的流程

从指定文件中读取数据，统计每个符号发生的概率，并建立相应的树叶节点

注意在此处 pSF 代入的是地址

第一次扫描：统计文件中各个字符出现频率

构建 Huffman 树及生成 Huffman 码

按字符概率由小到大将对应节点排序

得到文件出现的字符种类数

构建 Huffman 树

对码树编码

将码表及其他必要信息写入输出文件

第二次扫描：对源文件进行编码并输出

JPEG 原理分析及 JPEG 解码器的调试

1）JPEG 文件格式介绍

SOI start of image 图像开始

标记代码 2 字节 固定值 0xFFD8

APP0 appllication 应用程序保留标记 0

标记代码 2 字节 固定值 0xFFE0

包含 9 个具体字段

①数据长度 2 字节 ①~⑨9 个字段的总长度

②标识符 5 字节 固定值 0x4A46494600，即字符串“JFIF0”

③版本号 2 字节 一般是 0x0102，表示 JFIF 的版本号 1.2

④X 和 Y 的密度单位 1 字节 只有三个值可选

0：无单位；1：点数/英寸；2：点数/厘米

⑤X 方向像素密度 2 字节 取值范围未知

⑥Y 方向像素密度 2 字节 取值范围未知

⑦缩略图水平像素数目 1 字节 取值范围未知

⑧缩略图垂直像素数目 1 字节 取值范围未知

⑨缩略图 RGB 位图 长度可能是 3 的倍数 缩略图 RGB 位图数据

DQT define quantization table 定义量化表

标记代码 2 字节 固定值 0xFFDB

包含 9 个具体字段：

①数据长度 2 字节 字段①和多个字段②的总长度

②量化表 数据长度-2 字节

a) 精度及量化表 ID 1 字节

高 4 位：精度，只有两个可选值 0：8 位；1：16 位

低 4 位：量化表 ID，取值范围为 0～3

b) 表项 (64×(精度+1))字节

例如 8 位精度的量化表,其表项长度为 64×（0+1）=64 字节

本标记段中，字段②可以重复出现，表示多个量化表，但最多只能出现 4 次

SOF0 start of frame 帧图像开始

标记代码 2 字节 固定值 0xFFC0

包含 9 个具体字段：

①数据长度 2 字节 ①~⑥六个字段的总长度

②精度 1 字节 每个数据样本的位数

通常是 8 位，一般软件都不支持 12 位和 16 位

③图像高度 2 字节 图像高度（单位：像素）

④图像宽度 2 字节 图像宽度（单位：像素）

⑤颜色分量数 1 字节 只有 3 个数值可选

1：灰度图；3：YCrCb 或 YIQ；4：CMYK

而 JFIF 中使用 YCrCb，故这里颜色分量数恒为 3

⑥颜色分量信息 颜色分量数×3 字节（通常为 9 字节）

a)颜色分量 ID 1 字节

b)水平/垂直采样因子 1 字节

高 4 位：水平采样因子

低 4 位：垂直采样因子

c)量化表　　1 字节

当前分量使用的量化表的 ID

DHT　define huffman table　定义哈夫曼表

标记代码　2 字节　固定值 0xFFC4

包含 2 个具体字段：

①数据长度　　2 字节

②huffman 表　　数据长度-2 字节

表 ID 和表类型　　1 字节

高 4 位：类型，只有两个值可选

0：DC 直流；1：AC 交流

低 4 位：哈夫曼表 ID，

注意，DC 表和 AC 表分开编码

不同位数的码字数量　16 字节

编码内容　　16 个不同位数的码字数量之和（字节）

本标记段中，字段②可以重复出现（一般 4 次），也可以只出现 1 次。

SOS　start of scan　扫描开始 12 字节

标记代码　2 字节　固定值 0xFFDA

包含 2 个具体字段：

①数据长度　　2 字节　①~④两个字段的总长度

②颜色分量数　　1 字节　应该和 SOF 中的字段⑤的值相同，即：

1：灰度图是；3：　YCrCb 或 YIQ；4：CMYK。

③颜色分量信息

a)颜色分量 ID　　1 字节

b)直流/交流系数表号　　1 字节

高 4 位：直流分量使用的哈夫曼树编号

低 4 位：交流分量使用的哈夫曼树编号

④压缩图像数据

a)谱选择开始　　1 字节　固定值 0x00

b)谱选择结束　　1 字节　固定值 0x3F

c)谱选择　　1 字节　在基本 JPEG 中总为 00

EOI　end of image　图像结束

标记代码　2 字节　固定值 0xFFD9

2）JPEG 文件解码流程

读入文件的相关信息

JPEG 文件解码流程

SOI(0xFFD8)

APP0(0xFFE0)

[APPn(0xFFEn)]可选

DQT(0xFFDB)

SOF0(0xFFC0)

DHT(0xFFC4)

SOS(0xFFDA)

压缩数据

EOI(0xFFD9)

Huffman 表存储方式

在标记段 DHT 内，包含了一个或多个的哈夫曼表。对于单个哈夫曼表，应该包括三部分：

①Huffman 表 ID 和表类型

0x00 表示 DC 直流 0 号表　0x01 表示 DC 直流 1 号表

0x10 表示 AC 交流 0 号表　0x11 表示 AC 交流 1 号表

②不同位数的码字数量

JPEG 的 Huffman 编码只能是 1~16 位。这个字段的 16 个字节分别表示 1~16 位的编码码字在 Huffman 树中的个数。

③编码内容

这个字段记录了 Huffman 树中各个叶子结点的权。所以，上一字段的 16 个数值之和就应该是本字段的长度，也就是 Huffman 树种叶子结点的个数。

初步了解图像数据流的结构

颜色分量单元的内部解码

直流系数的差分编码

反量化&反 Zig-Zag 编码

反离散余弦变化

3）JPEG 文件解码程序实现

对 Huffman 码字解码前不知道码字的长度，解码有两种方法，Huffman 码树遍历和 lookup 查找表

序号	码长	码字	权值
1	2	00	04
2	2	01	05
3	2	10	06
4	3	110	03
5	4	1110	02
6	5	11110	01
7	6	111110	00
8	7	1111110	09
9	8	11111110	07
10	9	111111110	08

第一步　第二步

index	lookup	value	Code_size
1	001100	04	2
2	010001	05	2
3	100101	06	2
4	110010	03	3
5	111011	02	4
6	111101	01	5
7	111110	00	6
8			
9			
10			

见trace及程序　查找表

4）对函数 build_huffman_table 的分析

第一步：得到 code_size.（huffsize）

第二步：得到 code.（huffcode）

第三步：得到 code_size 查找表。

Table->code_size[val] = code_size;

第四步：得到 value 查找表（lookup）。

table->lookup[code++] = val

第五步：当码字长度>9，slowtable 处理。

slowtable[0] = code;　slowtable[1] = val;　slowtable[2] = 0;

彩色空间转换实验

基本原理：
RGB 和 YUV 彩色空间的基础知识；数据类型的分析

1）不带参数的主函数定义 void main（）
带参数的主函数定义 void main（int argc，char*argv[ ]）
调用：命令名 参数 1 参数 2 … 参数 n
参数值：argc=n+1 argv[0]=命令名 argv[1]=参数 1 argv[2]=参数 2 … argv[n]=参数 n

2）文件相关指针
FILE *fp;
FILE *fopen（char *filename，char *mode）;
filename 是要打开的文件路径
mode 是要打开的模式
若成功，返回指向被打开文件的指针；若出错，返回空指针 NULL
举例：

打开模式	描　　述
r	只读，打开已有文件，不能写
w	只写，创建或打开，覆盖已有文件
a	追加，创建或打开，在已有文件末尾追加
r+	读写，打开已有文件
w+	读写，创建或打开，覆盖已有文件
a+	读写，创建或打开，在已有文件末尾追加
t	按文本方式打开 (缺省)
b	按二进制方式打开

```
//指针定义
    char *rgbFileName = NULL;
    FILE *rgbFile = NULL;
//参数获取
    rgbFileName = argv[1];
//打开函数
    rgbFile = fopen(rgbFileName, "rb");
    if (rgbFile == NULL)
    {
        printf("cannot find rgb file\n");
        exit(1);
    }
    else
    {
        printf("The input rgb file is %s\n", rgbFileName);
    }
```

fclose（FILE *fp）;
fp 是要关闭的文件指针
若成功，返回 0；若出错，返回 EOF（-1）
不用的文件应
feof（FILE *fp）;
fp 是文件指针
若文件结束，返回非零值；若文件尚未结束，返回 0
fputc（int c, FILE *fp）;
c 是要输出到文件的字符
fp 是文件指针
若成功，返回输出的字符；若失败，返回 EOF

fgetc（FILE *fp）;

fp 是文件指针

若成功，返回输入的字符；若失败或文件结束，返回 EOF

fwrite/fread（void *buffer, size_t size, size_t count, FILE *fp）;

buffer 是要写/读的数据块地址

size 是要写/读的每个数据项的字节数

count 是要写/读的数据项数量

fp 是文件指针

若成功，返回实际写/读的数据项数量；若失败，一般返回 0

举例：

```
//开辟缓存空间
    rgbBuf = (u_int8_t*)malloc(frameWidth * frameHeight * 3);
    yBuf = (u_int8_t*)malloc(frameWidth * frameHeight);
    uBuf = (u_int8_t*)malloc((frameWidth * frameHeight) / 4);
    vBuf = (u_int8_t*)malloc((frameWidth * frameHeight) / 4);
//读取数据
    fread(rgbBuf, 1, frameWidth * frameHeight * 3, rgbFile);
//写入数据
    fwrite(yBuf, 1, frameWidth * frameHeight, yuvFile);
    fwrite(uBuf, 1, (frameWidth * frameHeight) / 4, yuvFile);
    fwrite(vBuf, 1, (frameWidth * frameHeight) / 4, yuvFile);
```

函数	功能
fputc	输出字符
fgetc	输入字符
putc	输出字符
getc	输入字符
fwrite	输出数据块

函数	功能
fprintf	格式化输出
fscanf	格式化输入
putw	输出一个字
getw	输入一个字
fputs	输出字符串
fgets	输入字符串

fprintf/fscanf（FILE *fp, char *format[,address/argument,...]）;

fputs(char *s, FILE *fp);

若成功，返回输出字符个数；若失败，返回 EOF

fgets（char *s, int n, FILE *fp）;

若成功，返回 s 首地址；若失败，返回 NULL

从 fp 输入字符串到 s 中，输入 n-1 个字符，或遇到换行符或 EOF 为止，读完后自动在字符串末尾添加 '\0'

rewind（FILE *fp）;

fp 是文件指针

使文件位置指针重新返回文件开头

fseek（FILE *fp, long offset, int whence）;

fp 是文件指针

offset 是偏移量

whence 是起始位置

随机改变文件的位置指针

SEEK_SET(0)是文件开始

SEEK_CUR(1)是文件当前位置

SEEK_END(2)是文件末尾

ftell（FILE *fp）;

fp 是文件指针

返回 fp 所指向文件中的读写位置

3）动态数组和指针

①申请空间

```
char *name;
name = (char *)malloc(20);

float *pf;
pf = (float *)malloc(sizeof(float)*20);

double *pd;
pd = (double *)malloc(sizeof(double)*50);
```

②赋值

```
int *pi;
pi = (int *)malloc(sizeof(int)*100);
*pi = 0;
*(pi+1) = 1;
*(pi+2) = 2;

int j;
for(j=0; j<100; j++)
    *(pi+j) = 0;
```

举例：

```
for (i = 0; i < size; i++)
{
    g = b + 1;
    r = b + 2;
    *y = (unsigned char)( RGBYUV02990[*r]    + RGBYUV05870[*g]    + RGBYUV01140[*b]);
    *u = (unsigned char)(- RGBYUV01684[*r]   - RGBYUV03316[*g]    + (*b)/2              + 128);
    *v = (unsigned char)(   (*r)/2           - RGBYUV04187[*g]    - RGBYUV00813[*b] + 128);
    b += 3;
    y ++;
    u ++;
    v ++;
}
```

③空间回收

```
free（filename）;
```

4）RGB to YUV 文件转换

①流程分析

程序初始化（打开两个文件、定义变量和缓冲区等）

读取 RGB 文件，抽取 RGB 数据写入缓冲区

调用 RGB2YUV 的函数实现 RGB 到 YUV 数据的转换

写 YUV 文件

程序收尾工作（关闭文件，释放缓冲区）

②代码分析

定义变量

```
static int init_done = 0;
long i, j, size;
unsigned char *r, *g, *b;
unsigned char *y, *u, *v;
unsigned char *pu1, *pu2, *pv1, *pv2, *psu, *psv;
unsigned char *y_buffer, *u_buffer, *v_buffer;
unsigned char *sub_u_buf, *sub_v_buf;
```

调用快速查找表

```
if (init_done == 0)
{
    InitLookupTable();
    init_done = 1;
}
```

定义指针、开辟缓存空间

```
y_buffer  = (unsigned char *)y_out;
sub_u_buf = (unsigned char *)u_out;
sub_v_buf = (unsigned char *)v_out;
u_buffer  = (unsigned char *)malloc(x_dim * y_dim);
v_buffer  = (unsigned char *)malloc(x_dim * y_dim);
```

```
b = (unsigned char *)rgb;
y = y_buffer;
u = u_buffer;
v = v_buffer;
```

将 RGB 数据转换为 YUV 数据

```
for (i = 0; i < size; i++)
{
    g = b + 1;
    r = b + 2;
//定义指针位置，RGB文件的数据存储方式为一个点依次存储B、G、R信息
    *y = (unsigned char)(  RGBYUV02990[*r] + RGBYUV05870[*g] + RGBYUV01140[*b]);
    *u = (unsigned char)(- RGBYUV01684[*r] - RGBYUV03316[*g] + (*b)/2          + 128);
    *v = (unsigned char)(  (*r)/2          - RGBYUV04187[*g] - RGBYUV00813[*b] + 128);
//利用公式将RGB信息转换为YUV信息，其中RGBYUVxxxxx为快速查找表对应值
    b += 3;
    y ++;
    u ++;
    v ++;
//使指针跳到新的位置
```

将转换为的 U、V 数据转换为 4：2：0 格式

```
for (j = 0; j < y_dim/2; j ++)
{
    psu = sub_u_buf + j * x_dim / 2;
    psv = sub_v_buf + j * x_dim / 2;
//用于输出的U、V数据指针
    pu1 = u_buffer + 2 * j * x_dim;          //加2j行
    pu2 = u_buffer + (2 * j + 1) * x_dim;    //加2j+1行
    pv1 = v_buffer + 2 * j * x_dim;          //加2j行
    pv2 = v_buffer + (2 * j + 1) * x_dim;    //加2j+1行
//1为四格U/V值中左上角值，2为左下角值
    for (i = 0; i < x_dim/2; i ++)
    {
        *psu = (*pu1 + *(pu1+1) + *pu2 + *(pu2+1)) / 4;
        *psv = (*pv1 + *(pv1+1) + *pv2 + *(pv2+1)) / 4;
//对取得的四个值做平均
        psu ++;
        psv ++;
        pu1 += 2;
        pu2 += 2;
        pv1 += 2;
        pv2 += 2;
//使指针跳到新位置
    }
```

附查找表

```
void InitLookupTable()
{
    int i;

    for (i = 0; i < 256; i++) RGBYUV02990[i] = (float)0.2990 * i;
    for (i = 0; i < 256; i++) RGBYUV05870[i] = (float)0.5870 * i;
    for (i = 0; i < 256; i++) RGBYUV01140[i] = (float)0.1140 * i;
    for (i = 0; i < 256; i++) RGBYUV01684[i] = (float)0.1684 * i;
    for (i = 0; i < 256; i++) RGBYUV03316[i] = (float)0.3316 * i;
    for (i = 0; i < 256; i++) RGBYUV04187[i] = (float)0.4187 * i;
    for (i = 0; i < 256; i++) RGBYUV00813[i] = (float)0.0813 * i;
}
```

5）YUV to RGB 文件转换

①流程分析

程序初始化（打开两个文件、定义变量和缓冲区）

读取 YUV 文件，抽取 YUV 数据写入缓冲区

调用 YUV2RGB 的函数实现 YUV 到 RGB 数据的转换

写 RGB 文件

程序收尾工作（关闭文件，释放缓冲区）

②代码分析

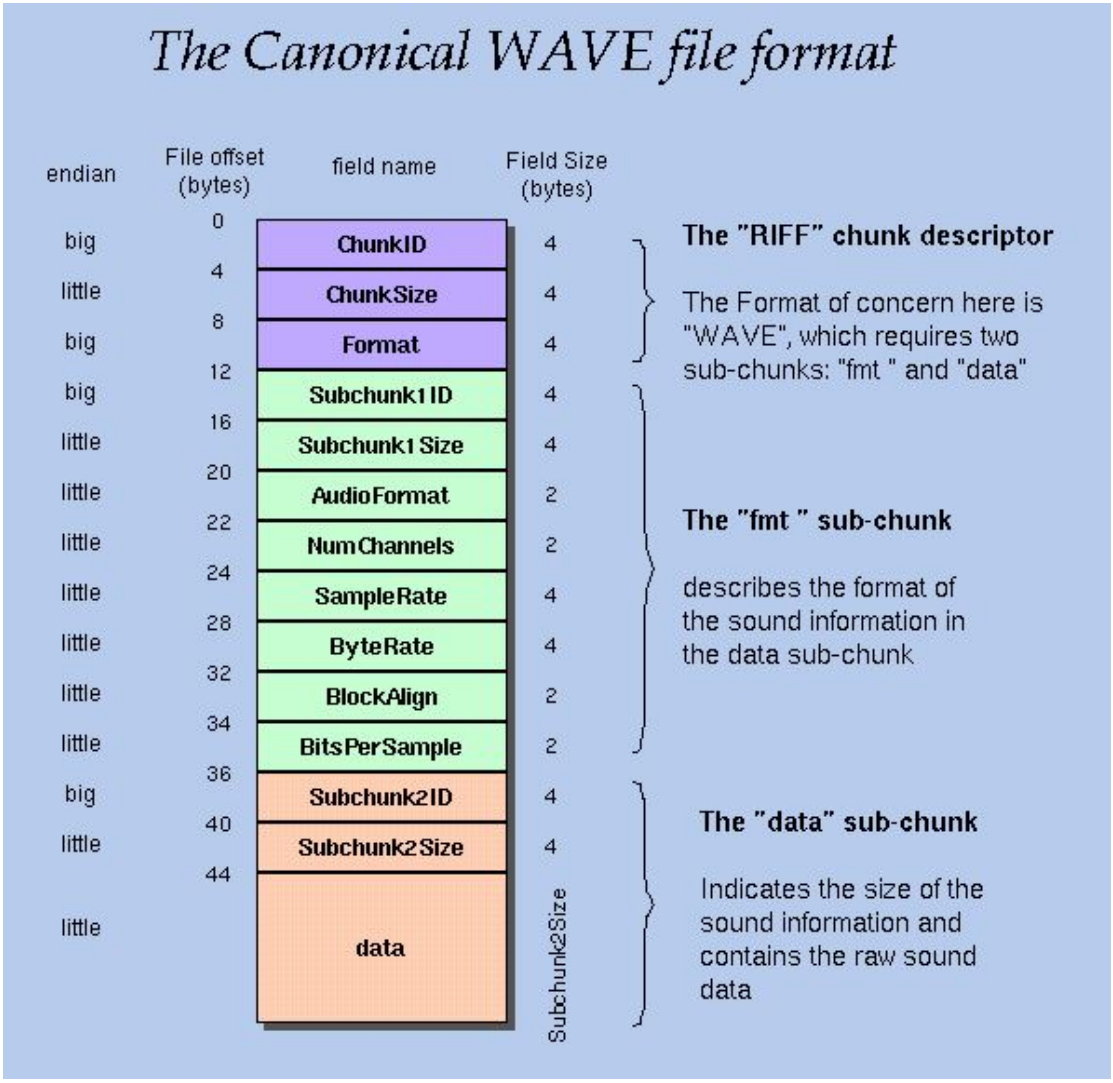
定义变量

调用快速查找表

定义指针、开辟缓存空间

将 YUV 数据转换为 RGB 数据

附查找表

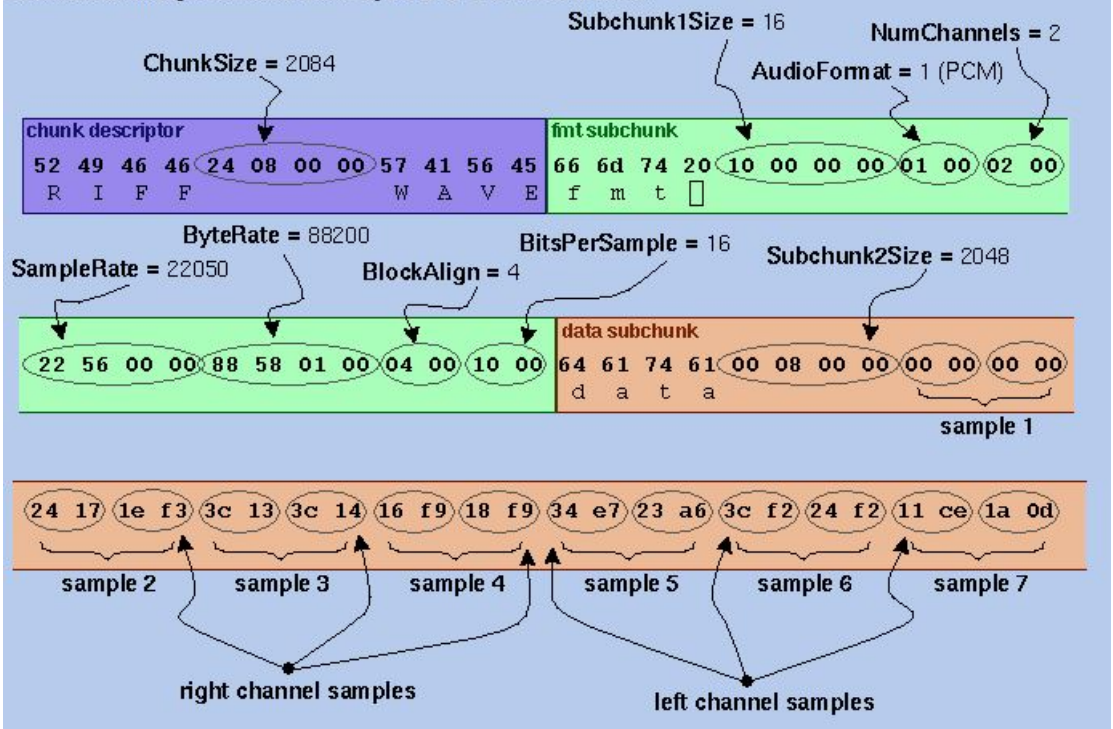


offset	size	name	description
0	4	ChunkID	包含“RIFF”的ASCII码表示(0x52494646)
4	4	ChunkSize	除去ChunkID和ChunkSize的剩余部分大小 36+SubChunk2Size或 4+(8+SubChunk1Size)+(8+SubChunk2Size)
8	4	Format	包含“WAVE”的ASCII码表示(0x57415645)
12	4	SubchunkIID	包含“fmt”的ASCII码表示(0x666d7420)
16	4	Subchunk1Size	如果文件采用PCM编码，则该子块剩余字节数为16
20	2	AudioFormat	如果文件采用PCM编码，则该值为1
22	2	NumChannels	单声道是1，双声道是2
24	4	SampleRate	8000，44100等
28	4	ByteRate	==SampleRate*NumChannels*BitsPerSample/8
32	2	BlockAlign	==NumChannels*BitsPerSample/8
34	2	BitsPerSample	8bit=8，16bit=16等
	2	ExtraParamSize	如果是PCM编码，这项不存在
	X	ExtraParams	给其他编码器的空间
36	4	Subchunk2ID	包含“data”的ASCII码表示(0x64617461)
40	4	Subchunk2Size	==NumSamples*NumChannels*BitsPerSample/8
44	*	Data	实际音频数据

As an example, here are the opening 72 bytes of a WAVE file with bytes shown as hexadecimal numbers:

```
52 49 46 46 24 08 00 00 57 41 56 45 66 6d 74 20 10 00 00 00 01 00 02 00
22 56 00 00 88 58 01 00 04 00 10 00 64 61 74 61 00 08 00 00 00 00 00 00
24 17 1e f3 3c 13 3c 14 16 f9 18 f9 34 e7 23 a6 3c f2 24 f2 11 ce 1a 0d
```

Here is the interpretation of these bytes as a WAVE soundfile:



Code	Description
0(0x0000)	Unknown
1(0x0001)	PCM/uncompressed
2(0x0002)	Microsoft ADPCM
6(0x0006)	ITU G.711 a-law
7(0x0007)	ITU G.711 Åµ-law
17(0x0011)	IMA ADPCM
20(0x0016)	ITU G.723 ADPCM(Yamaha)
49(0x0031)	GSM 6.10
64(0x0040)	ITU G.721 ADPCM
80(0x0050)	MPEG
65536(0xFFFF)	Experimental

第一章 绪论

1、什么是数据压缩

数据压缩，就是以最少的数码表示信源所发的信号，减少容纳给定消息集合或数据采样集合的信号空间。

所谓信源空间即被压缩对象，是指：物理空间、时间区间、电磁频段。

2、为什么要进行数据压缩

数据压缩的好处：

（课本）

①较快地传输各种信源——时间域的压缩

②在现有通信干线上开通更多的并行业务——频率域的压缩

③降低发射功率——能量域的压缩

④紧缩数据存储容量——空间域的压缩

（老师 PPT）

①减少存储空间——文件压缩、图像压缩、语音压缩、视频压缩……

②减少传输时间

③渐进传输——有些编码算法允许先发送重要的比特，使得我们在看到更逼真的内容之前，能先看到更低分辨率的版本

④降低计算——用较少的数据得到近似的结果

3、为什么可以数据压缩

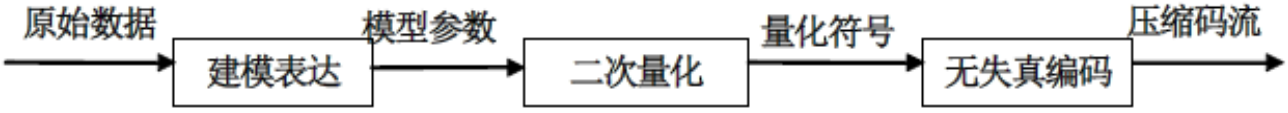
自然界中的大多数数据都是冗余的：任何非随机选择的数据都有一定结构，可利用这种结构得到数据的更小表示。

例如：统计冗余（概率不均等、文本冗余、图像冗余）、数据的物理产生过程（如利用人类的发声系统，设计语音压缩算法）、感知冗余（听觉冗余、视觉冗余）

4、怎样进行数据压缩

利用统计冗余，引入可接受的偏差

数据压缩的一般步骤



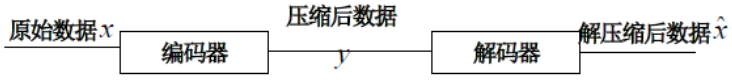
①建立一个数学模型，以便更有效地重新表达原始数据

②对模型参数进行量化（二次量化，对模拟信号进行的量化称为一次量化）

③更紧凑地表示量化后的模型参数，此时的编码要求能“忠实地”再现模型参数的量化符号（熵编码）

5、编码系统的性能比较

效率：压缩比



- 压缩比 $= \dfrac{\text{输入流的大小}}{\text{输出流的大小}} = \dfrac{|x|}{|y|}$
- 压缩增益 $= 100\ln \dfrac{\text{参考大小}}{\text{压缩后的大小}}$，参考大小：输入流大小/某种无失真编码方法所产生的输出流
- 无失真编码：
  - 压缩比通常不超过1:4　$x = \hat{x}$
- 有失真编码：
  - 压缩比通常超过1:10　$x \neq \hat{x}$

质量：对有失真编码而言

客观质量：失真度量

- 最常用：均方误差（Mean Squared Error, MSE）

$$e_k = x_k - \hat{x}_k \qquad \sigma_e^2 = \mathrm{E}\{e_k^2\}$$

- 等价于信噪比（Signal Noise Ratio, SNR）

$$SNR(dB) = 10\lg\frac{\sigma_x^2}{\sigma_e^2}$$

- 优点
  - 易于计算
  - 数学上在优化问题中易于处理
- 缺点
  - 没有考虑人的主观感知特性

- 均方误差：

$$MSE = \frac{1}{M\times N}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\{x(m,n)-\hat{x}(m,n)\}^2$$

- 峰值信噪比（ Peak Signal Noise Ratio, PSNR ）

$$PSNR(dB) = 10\lg\frac{x_{\max}^2}{\sigma_e^2}$$

主观质量

复杂度

算法的运算量和存储量

编码器和解码器的复杂度：对称/非对称

时延

5、本课程的主要内容

第一部分 视音频媒体的特性与表示

图像的属性和常见文件格式，视频信号的数字化及相关标准，AVI 等常见的视频文件格式，音频的特性和常见文件格式

第二部分 无失真压缩的理论基础

信息论基础：信源的建模与分析

变长码的编码基础——无失真信源编码定理

编码的基本途径

第三部分 统计编码方法

Huffman 编码，Golomb 编码和通用变长码，算数编码，游程编码

词典编码（LZ77、LZ78、LZW 算法，词典编码在文件压缩和图像压缩中的应用）

第四部分 有失真压缩的理论基础

失真的度量，率失真理论简介

有失真压缩中常用的模型：概率模型和线性系统模型

第五部分 有失真信源编码（熵压缩）

量化

基于变换的编码、JPEG 标准

预测编码

子带编码、小波编码、JPEG2000

第六部分 信源编码标准及应用

视频编码标准（MPEG-2、MPEG-4、H.264、AVS）

音频编码标准（MP2、MP3、G.721、G.729、……）

应用——码率控制、转码、传输

第二章 媒体的特性与表示

1、图形与图像属性及常用文件格式

1）视觉系统对颜色的感知

从人的主观感觉角度，颜色包含三个要素：色调、饱和度、明亮度

混合色的色调饱和度由三基色的混合比例决定，亮度是三基色亮度之和

2）图像的颜色模型（彩色空间 color space）

颜色模型是用来精确标定和生成各种颜色的一套规则和定义。某种颜色模型所标定的所有颜色就构成了一个颜色空间。颜色空间通常用三维模型表示，空间中的颜色通常使用代表三个参数的三维坐标来指定。

基色颜色空间：RGB、CMYK　色、亮分离颜色空间：HSL、YUV

3）图像的基本属性

①分辨率：

显示分辨率、图像分辨率

②像素深度：

存储每个像素所用的位数。像素深度决定彩色图像每个像素可能有的颜色数，或者确定灰度图像每个像素可能有的灰度级数。

③真彩色、伪彩色、直接色：

真彩色：在组成一幅在色图像的每个像素值中，有 R、G、B 三个基色分量，每个基色分量直接决定显示设备的基色强度，这样产生的彩色称为真彩色

伪彩色：每个像素的颜色不是由每个基色分量的数值直接决定，而是把像素值当做彩色查找表（调色板）的表项入口地址，去查找一个显示图像时使用的 R、G、B 强度值，用查找出的 R、G、B 强度值产生的彩色称为伪彩色

直接色：每个像素值分成 R、G、B 分量，每个分量作为单独的索引值对它做变换。也就是通过相应的彩色变换表找出基色强度，用变换后得到的 R、G、B 强度值产生的彩色称为直接色

4）图像的分类

①矢量图：计算生成，显示精度高，操作灵活性大。图像显示时花费时间比较长，真实世界的彩色图像难以转化为矢量图

②点位图：适合表现细致、色彩和层次比较丰富的图像。存储和传输时数据量比较大，缩放、旋转时算法复杂且容易失真。

③灰度图

④彩色图

5）常见的图像格式

①BMP

位图文件头BITMAPFILEHEADER
位图信息头BITMAPINFOHEADER
调色板Palette
实际的位图数据ImageData

```
typedef struct tagBITMAPFILEHEADER {
    WORD    bfType;        /* 说明文件的类型 */
    DWORD   bfSize;        /* 说明文件的大小，用字节为单位 */
                           /*注意此处的字节序问题
    WORD    bfReserved1;   /* 保留，设置为0 */
    WORD    bfReserved2;   /* 保留，设置为0 */
    DWORD   bfOffBits;     /* 说明从BITMAPFILEHEADER结构
                              开始到实际的图像数据之间的字 节
                              偏移量 */
} BITMAPFILEHEADER;
typedef struct tagBITMAPINFOHEADER {
    DWORD   biSize;     /* 说明结构体所需字节数 */
    LONG    biWidth;    /* 以像素为单位说明图像的宽度 */
    LONG    biHeight;   /* 以像素为单位说明图像的高度*/
    WORD    biPlanes;   /* 说明位面数，必须为1 */
    WORD    biBitCount; /* 说明位数/像素，1、2、4、8、24 */
    DWORD   biCompression; /* 说明图像是否压缩及压缩类型
                BI_RGB，BI_RLE8，BI_RLE4，BI_BITFIELDS */
    DWORD   biSizeImage;   /* 以字节为单位说明图像大小，必须是4的
                              整数倍*/
    LONG    biXPelsPerMeter;  /* 目标设备的水平分辨率，像素/米 */
    LONG    biYPelsPerMeter;  /*目标设备的垂直分辨率，像素/米 */
    DWORD   biClrUsed;   /* 说明图像实际用到的颜色数，如果为0
                            则颜色数为2的biBitCount次方 */
    DWORD   biClrImportant; /*说明对图像显示有重要影响的颜色
                索引的数目，如果是0，表示都重要。*/
} BITMAPINFOHEADER;
typedef struct tagRGBQUAD {
    BYTE    rgbBlue;       /*指定蓝色分量*/
    BYTE    rgbGreen;      /*指定绿色分量*/
    BYTE    rgbRed;        /*指定红色分量*/
    BYTE    rgbReserved;   /*保留，指定为0*/
} RGBQUAD;
```

紧跟在调色板之后的是图像数据字节阵列，图像的每一扫描行由表示图像像素的连续的字节组成，每一行的字节数取决于图像的颜色数目和用像素表示的图像宽度。规定每一扫描行的字节数必须是 4 的整数倍，也就是 DWORD 对齐的。扫描行是由底向上存储的，阵列中的第一个字节表示位图左下角的像素，最后一个字节表示位图右上角的像素。

②GIF

③JPG

2、视频的特性及常用文件格式

1）视频信号的数字化及标准

全信号数字化（直接对复合信号数字化的方式）、分量数字化（分别对 Y、U、V 分量信号数字化的方式）

码率：R=13.5MHz×8bit+（6.75MHz×8bit）×2=216Mb/s

并行数据传输：R=13.5MHz+6.75MHz×2=27Mb/s

①亮度信号的取样频率：13.5MHz

按照奈奎斯特取样定理，取样频率至少为信号上线频率的两倍

为了在取样后保证产生足够小的混叠噪声，要求取样频率是信号带宽的 2.2~2.7 倍。对 PAL 信号，取样频率应大于 13.2MHz

为了获取正交结构，取样频率必须是行频的整数倍

为了使两种扫描制式实现兼容，应采用同一取样频率，625 行频为 12.625Hz，525 行频为 15.734Hz，两者最小公倍数 2.25MHz

②色差信号的取样频率：6.75MHz

③视频信号量化位数：8bit

每行取样点数：每秒的取样点数（取样频率 13.5MHz）×每行扫描时间（64μs）

PAL 制的每行取样点数为 13500kHz×64μs=864

NTSC 制的每行取样点数为 13500kHz×63.55μs=858

有效水平取样点：为了使两种制式兼容，有效部分取 53.3μs，亮度信号 53.3μs÷0.074μs=720（T），色差信号 360（T）

有效垂直取样点：PAL 每帧 576 行，每场 288 行。NTSC 每帧 480 行，每场 240 行

2）AVI 文件解析

3、音频的特性及常用文件格式

1）音频信号及其心理特征

①声压（P）：声波引起某处媒质压强的变化量称为该处的声压。

②声压级（SPL）：将声压的有效值以对数的形式表示声音强弱的数值称为声压级。

③音频：声音信号的频率。人对于声音频率的感觉表现为音调的高低。音高与声音频率的关系大体上呈对数关系。

④音频带宽：20Hz~20kHz 是人类的听觉频带。人耳对不同频率的敏感程度有很大差别，中频段 2kHz~4kHz 最为敏感，幅度很低的信号都能被人耳听到，低频区和高频区较不敏感，能被人耳听到的信号幅度比中频段要高得多。

⑤响度与响度级：响度是指人类所感受到声音大小的程度，而响度级则是以 1kHz 信号的声压级数定义的响度的值，单位是“方”（Phon）

⑥绝对听阈：在安静环境中，能被人耳听到的纯音的最小值。

⑦掩蔽听阈：频域中的一个强音会掩蔽与之同时发声的附近的弱音。

⑧动态范围：是某个声音的最强音与最弱音的强度差，用分贝表示。它是衡量声音强度变化的重要参数。

⑨音频信号在时域和频域中的表现形式：在时域中表现为赋值随时间连续变化的曲线，在频域中则是将音频信号经傅里叶变换后在频率空间的分离或连续的谱线

2）音频信号的数字化

①音频信号的动态范围

②噪声容限

③采样定理及音频采样频率标准

采样频率 f_s 必须高于被采样信号所含最高频率的两倍

④量化

⑤编码

第三章 无失真数据压缩

1、数据压缩的理论极限与基本途径

1）离散无记忆信源及其压缩基本途径

非等概率→统计匹配：统计编码

2）联合信源及其压缩基本途径

不独立的各分量→独立的各分量：彩色空间转换，变换编码

3）随机序列信源及其压缩基本途径

利用条件概率：预测编码

利用联合概率：对多个符号合并编码，如矢量量化

将非平稳信源转换为平稳信源：分析/综合编码，分成多个平稳子带

2、统计编码方法

1）霍夫曼编码

①递归 Huffman 算法

如果只有一个符号，则有一个节点的树是最佳的，否则找到两个概率最小的符号，其概率分别为 p 和 q，用一个概率为 p+q 的新符号代替上述两个符号，用递归法对新的符号集求解，用一个有两个旧符号孩子的中间节点表示新符号。

②霍夫曼编码的实现过程

先从指定文件中读取数据，统计每个符号发生的概率，并建立相应的树叶节点。然后构建霍夫曼树及霍夫曼码，按字符概率由小到大将对应节点排序，得到文件出现的字符种类数，继而构建霍夫曼树，对码树编码。

③霍夫曼解码

比特串行解码、基于查找表的解码

④规范霍夫曼编码

只要符合是及时码、某一字符编码长度和使用二叉树建立的该字符的编码长度相同这两个条件的编码都可以叫做 Huffman 编码

规范霍夫曼编码的提出：

小量数据的压缩，霍夫曼码表的存储是问题

使用某些强制的约定，仅通过很少的数据便能重构处 Huffman 编码树的结构

编码步骤：

统计每个符号的频率并求出该符号所需的位数/编码长度

统计从最大编码长度到 1 的每个长度对应多少个符号，然后为每个符号递增分配编码

编码输出压缩信息

2）算术编码

①基本思路

算术编码是从另一种角度对很长的信源符号序列进行有效编码的方法，是一种非分组码。

它不是将单个的信源符号映射成一个码字，而是将整个输入符号序列映射为实轴[0,1)内的一个小区间，其长度等于该序列的频率；再在该小区间内选择一个代表性的二进制小数作为编码输出，从而达到了高效编码的目的。

②编码方法

将符号序列的累计函数写成二进位的小数，取小数点后 L 位，若后面还有尾数，就进到第 L 位。令 L 满足 $L=\log[1/P(S)]$

③自适应算术编码

3）词典编码

①基本思路

如果用一些简单的代号代替这些字符串，就可以实现压缩。字符串与代号的对应表就是词典。

②第一类词典编码

第一类词典法的想法是企图查找正在压缩的字符序列是否在以前输入的数据中出现过，然后用已经出现过的字符串替代重复的部分，其输出代号定义为指向早期出现过的字符串的“指针”。

LZ77“滑动窗口压缩”

将一个虚拟的，可以跟随压缩进程滑动的窗口作为词典，要压缩的字符串如果在该窗口中出现，则输出其出现位置和长度。

LZSS

如果匹配串的长度比指针本身的长度长就输出指针，否则就输出真实字符。需要输出额外的标志位区分是指针还是字符。

③第二类词典编码

第二类算法的想法是企图从输入的数据中创建一个“短语词典”，这种短语可以是任意字符的组合。编码数据过程中当遇到已经在词典中出现的“短语”时，编码器就输出这个词典中的短语的“索引号”，而不是短语本身。

LZW

只输出代表词典中的字符串的码字。在开始时词典不能是空的，它必须包含可能在字符流出现中的所有单个字符。即在编码匹配时，至少可以在词典中找到长度为 1 的匹配串。

4）游程编码

①游程编码的基本思路

游程长度是指字符（或信号采样值）构成的数据流中各字符重复出现而形成字符串的长度。如果给出了形成串的数据字符 X、串的长度 RL 及串的位置，就能恢复原来的数据流。游程长度编码就是用二进制码字给出上述信息的一类方法。

基本的 RLC 方法最初需要加一个“异字头”前缀，因而低效且不使用。但是，对于二值图像和连续色调图像，该前缀可以省去。因此，改进的 RLC 在图像编码中得到了广泛的应用。

②二值图像的游程编码

二值图像是指仅有黑（用“1”代表)、白（用“0”代表）两个亮度值的图像。

编码思路：

分别对“黑”、“白”的不同游程长度进行 Huffman 编码，形成黑、白两张码表，编译码通过查表进行。

编码规则：

对 0~63 的黑白游程，用结尾码表示

对 64~1728 的黑白游程，用构造码+结尾码表示

规定每行都从白游程开始，若实际扫描行由黑开始，则需在行首加零长度白游程。每行结束要加行同步码 EOL

（构造码、结尾码查表）

5）Golomb 编码

①提出的背景

变长码对差错极其敏感，即时码的特性导致 1 位出错就可能使解码器失去同步

解决思路：

基于某个预先假定的概率模型设计出最佳变长码，码字具有某种固定的结构。癌牺牲一定压缩效率的前提下，提高解码器的可靠性。

②一元码

对非负整数 n，用 n 个 1 和 1 个 0 表示，或者用 n 个 0 和 1 个 1 表示。为即时码

③Golomb 编码

适用对象：服从几何分布的正整数数据流，数据流中整数 n 出现的概率为 $P（n）=（1-p）^{n-1}p$

使用一个可调参数 b 将输出码字分为若干个组，每组内均有码长相近的 m 个码字，每个码字有两个部分组成，前缀码和尾码。前缀码是 q+1 位的一元码字，q=INT[n-1/b]。尾码是 r=n-1-qb 的二进制编码

$b=2^k$ 时刻简化为 G（k），例如

n	k = 0 , m=1	k = 1 ,m=2	k = 2 ,m=4	k = 3,m=8
1	0	0 0	0 00	0 000
2	10	0 1	0 01	0 001
3	110	10 0	0 10	0 010
4	1110	10 1	0 11	0 011
5	11110	110 0	10 00	0 100
6	111110	110 1	10 01	0 101
7	1111110	1110 0	10 10	0 110
8	11111110	1110 1	10 11	0 111
9	111111110	11110 0	110 00	10 000

m 较小，Golomb 码初始很短，增长很快，适合大游程很少的情况

m 较大，Golomb 码初始很长，增长很慢，适合大游程较多的情况

④指数 Golomb 编码（EGC）

尾码增加 1 位，前缀码增加 1 位

可视为 G（0）码加上 q+m 位尾码（m 为阶数，q 为 1 的个数），例如

n	code
0	0
1	100
2	101
3	11000
4	11001
5	11010
6	11011
7	1110000
8	1110001
9	1110010
10	1110011
11	1110100
12	1110101
13	1110110
14	1110111
15	1111000000

第四章 量化

1、标量量化

1）量化器的描述

量化：用一个很小的集合表示一个大集合

量化区间的数目 M，决策边界 b_i，重构水平 y_i，量化误差 x-y

$$D=\int_{-\infty}^{\infty}(x-\hat{x})^2 f(x)dx=\sum_{i=1}^{M}\int_{b_{i-1}}^{b_i}(x-y_i)^2 f(x)dx$$

2）均匀量化

3）Lloyd-Max 算法

$$y_i=\frac{\int_{b_{i-1}}^{b_i}xf(x)dx}{\int_{b_{i-1}}^{b_i}f(x)dx},\ b_i=\frac{y_i+y_{i+1}}{2}$$

①门限（判决）电平应取在相邻量化输出电平的中点

②量化电平（重建电平）应取在量化间隔的质心上

当量化器的输入为均匀分布时，f(x)=c，Lloyd-Max 量化器变为均匀量化器

$$y_i=\frac{\int_{b_{i-1}}^{b_i}xf(x)dx}{\int_{b_{i-1}}^{b_i}f(x)dx}=\frac{c\int_{b_{i-1}}^{b_i}xdx}{c(b_i-b_{i-1})}=\frac{\frac{1}{2}(b_i^2-b_{i-1}^2)}{(b_i-b_{i-1})}=\frac{1}{2}(b_i+b_{i-1})$$

迭代 Lloyd-Mac 算法（已知 f（x））

1. 初始化所有的 $y_i, j=1, D_0=\infty$
2. 更新所有的决策边界： $b_i=\frac{y_i+y_{i+1}}{2}$
3. 更新所有的 y_i：

$$y_i=\frac{\int_{b_{i-1}}^{b_i}xf(x)dx}{\int_{b_{i-1}}^{b_i}f(x)dx}$$

4. 计算*MSE*： $D_j=\sum_{k=1}^{M}\int_{b_{i-1}}^{b_i}(x-y_k)^2 f(x)dx$
5. 如果 $(D_{j-1}-D_j)/D_{j-1}<\varepsilon$ 停止；否则 $j=j+1$，转第2步

4）熵约束量化（ECSQ）

Lloyd-Max 量化器：

对索引用固定码率编码：$\log_2 M$（R）比特

熵约束标量量化器：

对量化索引用变长码编码：

对量化索引用熵编码技术编码

平均码率~重构水平的熵≤$\log_2 M$

$$H(\hat{X})=-\sum_{k=1}^{M}p_k\log p_k\le R$$

比 Llyod-Max 量化器的性能更好

问题的形式化描述

最小化 $D=\mathrm{E}\left((X-\hat{X})^2\right)=\sum_{k=1}^{M}\int_{b_{k-1}}^{b_k}(x-y_k)^2 f(x)dx$

满足 $H(\hat{X})=-\sum_{k=1}^{M}p_k\log p_k\le R$

其中 $p_k=\int_{b_{k-1}}^{b_k}f(x)dx$

用 Lagrange 费用函数

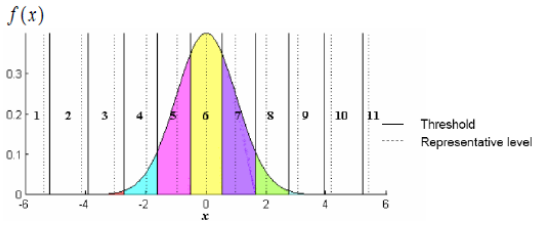
$$J(\lambda)=\mathrm{E}\left(\left(X-\hat{X}\right)^2\right)+\lambda H\left(\hat{X}\right)$$

太复杂，不能直接求解，要用迭代法求解

例： ECSQ算法的应用(I)

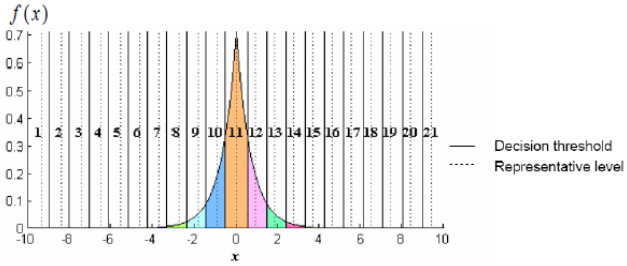
- X为0均值，1方差的高斯分布，即 $X\sim N(0,1)$
- 设计一个 $R\cong 2$ 的ECSQ，使得期望失真 D^*最小
  - 11个区间（[-6，6]内）：几乎是均匀
  - $D^*=0.09=10.53dB,\ R=2.0035$
    - 定长编码：

$$D^*=0.12=9.30\ dB$$



例： ECSQ算法的应用(II)

- X为0均值，1方差的Laplacian分布
- 设计一个 $R\cong 2$ 的ECSQ，使得期望失真 D^*最小
  - 21个区间（[-10，10]内），几乎是均匀的
  - $D^*=0.07=11.38dB$



5）Deadzone Midtread 量化器

0 附近的量化区间大小为其余量化区间的两倍，其他量化区间仍是均匀的

6）嵌入式量化器

动机：可伸缩解码

随着比特流的解码，渐进地精化重构数据

对低带宽连接有用

是 JPEG2000 的一个关键特征

嵌套量化：低码率器的区间被再分割，以产生更高码率的量化器

可以通过截断量化索引获得较粗燥的量化

7）对于已知概率模型及其数字特征的随机过程，比较容易根据概率分布安排量化器的决策边界，以得到最小量化失真的优化量化器。

如果概率分布是均匀的，则采用均匀量化比较理想。

对于分布概率模型未知的随机过程，其优化量化器的设计较为困难，可以采用 Lloyd-Max 算法来解决，但实现时还有一定困难，不宜硬件实现，执行时间也因初始值选取的不同而不同。

2、矢量量化（VQ）

1）矢量量化的基本思想

量化时不是处理单个符号，而是一次处理一组符号（矢量）

假设块大小为 a×b，码书中共有 M 个码字（码字也是长度为 a×b 的矢量），则码率 R=logM/（a×b）bpp

2）LBG 算法

将 Lloyd 算法推广到矢量量化，所以亦称为推广的 Lloyd 算法

给定训练集：$T=\{\boldsymbol{x}_1,\boldsymbol{x}_2,...,\boldsymbol{x}_N\}$

- 1. 初始化所有的 $\boldsymbol{y}_i,\ i=1,...,M$
- 2. 对训练集中所有的训练样本 $\boldsymbol{x}_n,n=1,2,...,N$ ，找到距离最近的码字：

$$Q(\boldsymbol{x}_n)=\boldsymbol{y}_i,\ iff\ d(\boldsymbol{x}-\boldsymbol{y}_i)\le d(\boldsymbol{x}-\boldsymbol{y}_k),\ for\ all\ k\ne i$$

- 3. 计算平均失真
- 4. 如果平均失真足够小，停止；否则转第5步
- 5. 用每个量化区域内所有矢量的平均值替代 y_i，转第2步

优化过程中可能限于局部最小值

依赖于初始码书的选取

初始码书的选取

随机选择：重复多次，取失真最小的结果

分裂：从一个类开始，每次将失真最大/数量最多的类分裂成两个

合并：从 N 个类开始，每次将两个失真最小的类合并

码字中缺少结构

编码复杂性高：需要全搜索

存储要求高，码书指数增长

第五章 预测编码

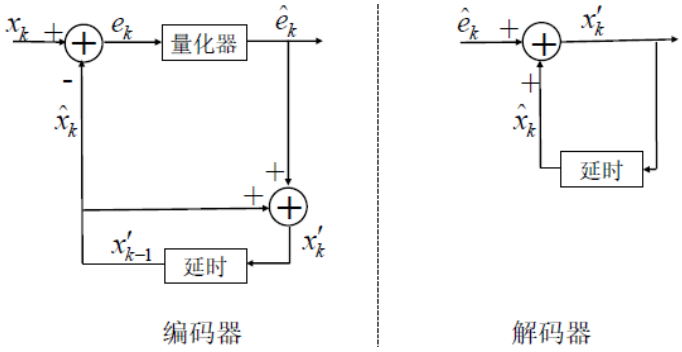
1、预测编码的基本原理

利用信源相邻符号之间的相关性。

根据某一模型利用以往的样本值对新样本进行预测，然后将样本的实际值与其预测值相减得到一个误差值，最后对这一误差值进行编码。

如果模型足够好，且样本序列在时间上相关性较强，则无差信号的幅度将远远小于原始信号，从而得到较大的数据压缩。

1）DPCM（差分脉冲编码调制）的基本原理



预测误差为

$$e_k = x_k - \hat{x}_k = x_k - f(x'_{i-1}, x'_{i-2}, \ldots, x'_{i-N})$$

预测误差的方差为

$$MSE = \sigma_e^2 = \mathrm{E}\left[(x_k - \hat{x}_k)^2\right] = \mathrm{E}\left[(x_k - f(x'_{i-1}, x'_{i-2}, \ldots, x'_{i-N}))^2\right]$$

若用 MSE 作为失真度量，则最小化失真的最佳预测器为

$$\hat{x}_k = \hat{f}(x'_{i-1}, x'_{i-2}, \ldots, x'_{i-N}) = \mathrm{E}\left[x_k \mid x'_{i-1}, x'_{i-2}, \ldots, x'_{i-N}\right]$$

若采用前 N 个样本预测，且每个样本为 B 比特，则上述条件概率需要一个有 2^{BN} 项的表

在一个 DPCM 系统中需要设计两个部分，预测器和量化器。在理想情况下，应同时优化预测器和量化器，但实际应用中，采用一种次优化方法，在这种方法中，量化电平数必须足够大。

线性预测器的设计：

具有最小均方预测误差的预测器——最大的预测增益

即在最小 E[d^2]的条件下，确定一组最佳预测系数（略去量化误差）

例1：DPCM

- 例：令DPCM中的参数 $N=1, w_1=1$，即取前一个取样 x_{k-1} 值作为当前取样值x_k的预测值，则

$$\sigma_e^2 = \mathrm{E}\left[(x_k - \hat{x}_k)^2\right] = \mathrm{E}\left[(x_k - x_{k-1})^2\right] = \mathrm{E}\left(x_k^2 - 2x_k x_{k-1} + x_{k-1}^2\right)$$
$$= 2\left[1 - \frac{R(1)}{R(0)}\right]R(0) = 2(1-\rho)R(0)$$

- 其中 $\rho = R(1)/R(0)$ 为信号的自相关系数，当$0.5 < \rho \le 1$时，$\sigma_e^2 < R(0)$，可达到压缩的效果
  - 对电视图像，通常 $\rho = 0.95 \sim 0.98$ ，此时$\sigma_e^2/R(0) = 1/10 \sim 1/24$ 这意味着误差信号的功率比原始信号降低了$10 \sim 14\ dB$
    - 降低6 dB 约减少1bit

例2：DPCM

- 例：在图像中的例子。$N=3$，像素X用其3个邻居A、B和C来预测，求MSE下的最佳预测系数

$$\sigma_e^2 = \frac{1}{K}\sum_{k=1}^{K}(x_k - \hat{x}_k)^2 = \frac{1}{K}\sum_{k=1}^{K}\left(x_k - \sum_{i=1}^{N} w_i x_{k-i}\right)^2$$

B	C
A	X

A：-1
B：2
C：3

101	128	108	110
100	90	95	100
102	80	90	85
105	75	96	91

输入图像

$$\boldsymbol{x} = \{90, 95, 100, 80, 90, 85, 75, 96, 91\}$$
$$\boldsymbol{x}_{-1} = \{100, 90, 95, 102, 80, 90, 105, 75, 96\}$$
$$\boldsymbol{x}_{-2} = \{101, 128, 108, 100, 90, 95, 102, 80, 90\}$$
$$\boldsymbol{x}_{-3} = \{128, 108, 110, 90, 95, 100, 80, 90, 85\}$$
$$\sigma_e^2 = \left(\boldsymbol{x} - \sum_{i=1}^{3} w_i \boldsymbol{x}_{-i}\right)$$

用最小二乘法求解，得到$w_1 = 0.1691$, $w_2 = 0.1988$, and $w_3 = 0.5382$

例3 预测阶数的选择

- 例：1维N阶高斯马尔科夫过程，自相关系数为

$$r(k)=\frac{R(k)}{R(0)}=\rho^{|k|} \qquad 0<\rho<1$$

- 求解a_i

$$\begin{bmatrix} 1 & \rho & \rho^2 & \cdots \rho^{N-1} \\ \rho & 1 & \rho & \cdots \rho^{N-2} \\ \vdots & & & \\ \rho^{N-1} & \rho^{N-2} & \rho^{N-3} & \cdots 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} \rho \\ \rho^2 \\ \vdots \\ \rho^N \end{bmatrix}$$

若$N=1$，则$a_1=\rho, \hat{x}_k=\rho x_{k-1}$

$$\sigma^2_{emin}=R(0)-a_1R(1)=\left[1-a_1\frac{R(1)}{R(0)}\right]R(0)=(1-\rho^2)R(0)<R(0)$$

$$\begin{aligned} E\{e_k e_{k+j}\} &= E\{[x_k-\rho x_{k-1}][x_{k+j}-\rho x_{k+j-1}]\} \\ &= R(j)-\rho R(j+1)-\rho R(j-1)+\rho^2 R(j) \\ &= R(0)[\rho^j-\rho\cdot\rho^{j+1}-\rho\cdot\rho^{j-1}+\rho^{j+2}]=0 \end{aligned}$$

例3 预测阶数的选择

- 例：1维N阶高斯马尔科夫过程，自相关系数为

$$r(k)=\frac{R(k)}{R(0)}=\rho^{|k|} \qquad 0<\rho<1$$

若$N=2$，即求$\hat{x}_k=a_1x_{k-1}+a_2x_{k-2}$

$$\begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}=\begin{bmatrix} \rho \\ \rho^2 \end{bmatrix}$$

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}=\frac{1}{1-\rho^2}\begin{bmatrix} 1 & -\rho \\ -\rho & 1 \end{bmatrix}\begin{bmatrix} \rho \\ \rho^2 \end{bmatrix}=\begin{bmatrix} \rho \\ 0 \end{bmatrix}$$

可见：N=2时解得的预测方程与N=1时的完全相同

注意：所谓的最佳预测是在MMSE准则下。当考虑到人的主观感觉时，并不一定最佳。

2）ADPCM

DPCM 是以输出数据为平稳随机过程为依据。预测系数和量化器参数一次设计好后即不再改变。但在一些应用中，如在图像平坦区和边缘处要求量化器的输出差别很大。否则会导致图像出现噪声

ADPCM 工作原理：预测器的预测系数和量化器的量化参数能够根据信源符合的局部区域分布特点而自动调整。

2、测编码在语音编码中的应用

1）语音信号的特性

浊音信号有长时间周期性，这是语音压缩的关键。

根据时间域的统计分析，语音信号中存在着多种冗余度：

①幅度非均匀分布

②样本间的相关

③周期之间的相关

④基音之间的相关

⑤静止系数（语音间隔）

⑥长时间自相关函数

如果从频率域考察语音信号的功率谱密度，可以发现：

①非均匀的长时功率谱密度

②语音特有的短时功率谱密度

2）波形编码

尽量保持重建语音与原始语音连信号波形都基本相同。

3）声码编码/参数编码

提取信源信号的特征参数并以数字代码传输；接收端从数字代码中恢复特征参数，由特征参数重建语音信号。

4）混合编码

既保留参数编码的声道模型，又像波形编码那样传递预测误差。

最流行的混合编码为时域合成-分析 AbS 算法，码激励线性预测编码（CELP）是 AbS 编码的一种。

3、预测编码在视频编码中的应用

1）视频信号中的冗余

各成分之间的冗余（彩色图像不同分量之间相关）、空间冗余（同一帧内相邻像素值相似）、时间冗余（相邻帧的内容通常比较相似）

2）帧内预测

去除空间冗余。对视频某一帧中的像素，可用其相邻像素的值来预测，然后对原信号与预测信号的差值进行编码。

3）帧间预测：运动估计/运动补偿

去除时间冗余。

运动估计的分类：

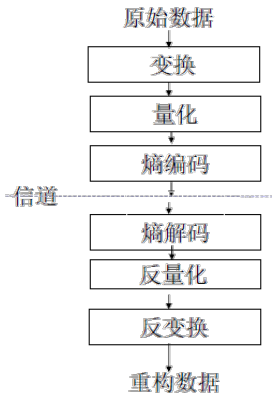
全局运动估计、基于像素点的运动估计、基于块的运动估计、基于区域的运动估计

基于块的运动估计、基于网格的运动估计

块匹配：

视频压缩时，只需保存运动矢量和残差数据就可以完全恢复处当前块

第六章 变换编码



1、变换编码基本原理

如果利用映射变换来实现对数据的建模表达，就称为变换编码。

其中映射变换是把原始信号中的各个样值从一个域变换到另一个域，然后针对变换后的数据再进行量化（二次量化）与编码操作。

接收端首先对收到的信号进行解码和反量化，然后再进行反变换以恢复原来信号（在一定的保真度下）。

2、离散正交变换

1）基本概念

令相邻的 n 个信号样本看作在 n 维线性空间中的一个列向量 $X=(x_1,x_2,\ldots,x_n)^T$

对它进行线性变换，$Y=AX$，称 A 为变换矩阵，Y 是 X 的一个线性变换

如果矩阵 A 具有性质 $AA^T=A^TA=I$，则称 A 为正交矩阵，从 X 到 Y 的变换为正交变换

对于正交变换，反变换可以唯一的得到复原信号 $X=A^TY$

对正交变换的总能量保持 $y=Ax$，$||y||^2=yy^T=x^TA^TAx=||x||^2$，总的能量保持，但是通常能量在各系数上分布并不均匀

X 的协方差矩阵

$$\Phi_X=E\{[X-E(X)][X-E(X)]^T\}=\begin{pmatrix}\Phi_{11} & \Phi_{12} & \ldots & \Phi_{1N}\\ \Phi_{21} & \Phi_{22} & \ldots & \Phi_{2N}\\ \ldots & \ldots & & \ldots\\ \Phi_{N1} & \Phi_{N2} & \ldots & \Phi_{NN}\end{pmatrix}$$

其中 $\Phi_{ij}=E\{[x_i-E(x_i)][x_j-E(x_j)]\}=\Phi_{ji}$

Y 的协方差矩阵 $\Phi_Y=E\{[Y-E(Y)][Y-E(Y)]^T\}=E\{A[X-E(X)][X-E(X)]^TA^T\}=A\Phi_XA^T$

为使得变换后各系数不相关，即要令 Y 的协方差矩阵中只存在对角线元素。

对于一个实对称矩阵Φ，必存在一个正交矩阵 Q，使得 $Q\Phi Q^{-1}=diag[\lambda_1\lambda_2\ldots\lambda_N]=\Lambda$，其中对角阵$\Lambda=diag[\lambda_1\lambda_2\ldots\lambda_N]$的 N 个对角元$\lambda_1\lambda_2\ldots\lambda_N$是Φ的特征根，Q 中的第 i 个行向量是Φ的第 i 个特征根λ_N所对应的满足归一化正交条件的特征向量，即 $\Phi q_i=\lambda_iq_i$，$q_i^Tq_j=\begin{cases}1, & i=j\\ 0, & i\neq j\end{cases}$

只要选择正交矩阵 Q 作为变换核矩阵，其行向量是 X 的协方差矩阵的特征向量，则变换后的矢量 Y 的协方差矩阵为 $\Phi_Y=E\{[Y-E(Y)][Y-E(Y)]^T\}=E\{[QX-E(QX)][QX-E(QX)]^T\}=QE\{[X-E(X)][X-E(X)]^T\}Q^T=Q\Phi_XQ^T=\Lambda$，则变换后各分量之间的相关性被全部去除。

2）KL 变换

以矢量 X 的协方差矩阵的归一化正交特征向量（qi）所构成的正交矩阵 Q，对该矢量所做的正交变换 Y=QX 称作 KL 变换。要实现 KTL，必须要先知道输入信号的协方差矩阵，然后再求出协方差矩阵的特征值和特征向量。用特征向量构成该输入信号矢量的正交变换核矩阵。

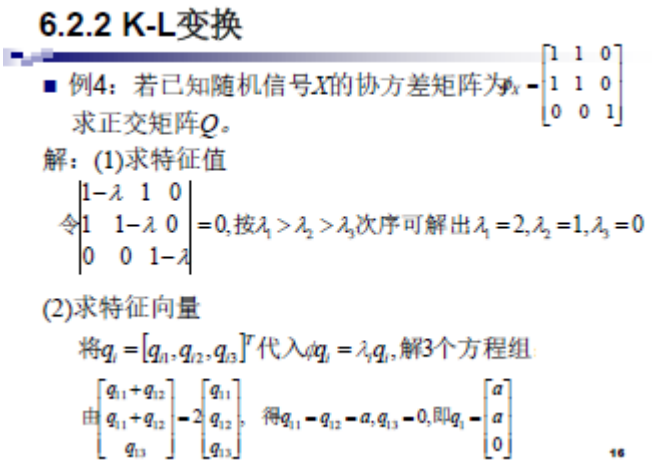
A 为 n 阶方阵，若存在数λ和 n 维非 0 列向量 x 使 $Ax=\lambda x$，则λ为 x 的特征值，x 为 A 对应于特征值λ的特征向量，方程组形式（$A-\lambda I$）$x=0$。

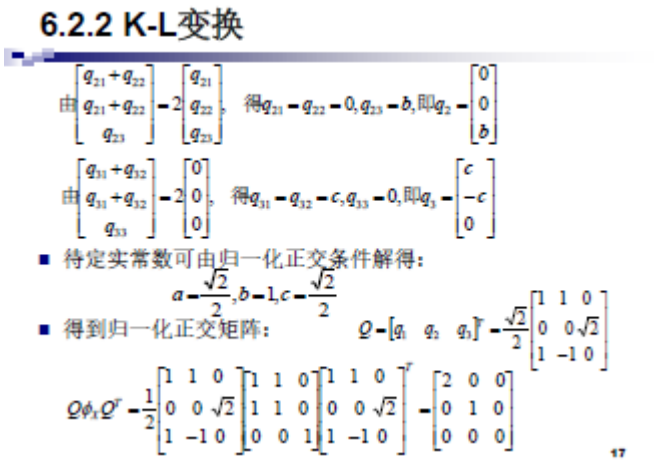
如果特征值和特征向量存在，则上述方程必有非 0 解 x，即要求系数矩阵 $A-\lambda I$ 的秩<n，所以系数行列式$|A-\lambda I|=0$，该式是关于λ的 n 次多项式，即矩阵 A 的特征多项式。

一个特征值可对应无穷个不同的特征向量。某一特征向量只对应于某一特征值。所以方阵 A 的不同特征值所对应的特征向量线性无关。

求法：（1）|A-λI|=λ的多项式，求出特征值

（2）把每个特征值带入（A-λI）x=0，求出齐次线性方程组的一个基础解系的全部解$\xi_1,\xi_2,\ldots,\xi_s$，则$\sum k_i\xi_i$是 A 对应于特征值λ的全部特征向量，其中 $k_1,k_2,\ldots,k_s$ 不全为 0





KLT：基函数为输入信号的协方差矩阵的特征向量。KLT 产生的系数不相关（输出信号协方差矩阵为对角阵），KLT 达到最佳的能量集中，KLT 取决于信号的统计性质，没有快速计算法，寻找结构化的变换，使得其性能接近 KLT

3）DCT 变换

二维 DCT 变换

DCT变换使用下式计算

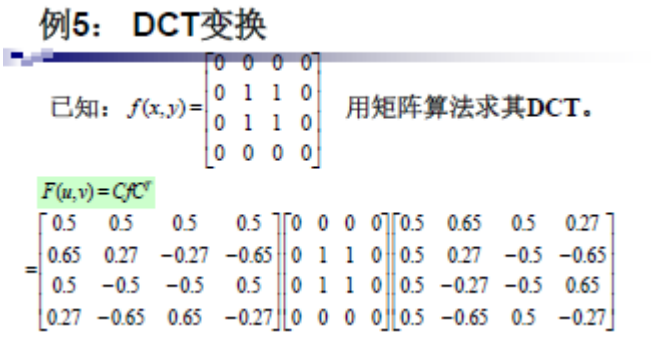
$$F(u,v)=\frac{1}{4}C(u)C(v)\left[\sum_{i=0}^{7}\sum_{j=0}^{7}f(i,j)\cos\frac{(2i+1)u\pi}{16}\cos\frac{(2j+1)v\pi}{16}\right]$$

逆变换使用下式计算

$$f(i,j)=\frac{1}{4}C(u)C(v)\left[\sum_{u=0}^{7}\sum_{v=0}^{7}F(u,v)\cos\frac{(2i+1)u\pi}{16}\cos\frac{(2j+1)v\pi}{16}\right]$$

$$\text{其中}\quad\begin{cases}C(u),C(v)=1/\sqrt{2} & u,v=0\\ C(u),C(v)=1 & otherwise\end{cases}$$

变换核可分离，先进行垂直方向的 DCT 变换，再进行水平方向的 DCT 变换



DFT 在边界不连续，DCT 更连续

3、JPEG 图像压缩标准

1）压缩标准

JPEG 规定了 4 种运行模式，以满足不同需要：

基于 DPCM 的无损编码模式：压缩比可达 2:1

基于 DCT 的有损顺序编码模式：压缩比可达 10:1 以上

基于 DCT 的递增编码模式

基于 DCT 的分层编码模式

2）颜色空间

JPEG 标准本身并没有规定具体的颜色空间，只是对各分量分别进行编码。实现中通常将高度相关 RGB 颜色空间转换到相关性较小的 YUV 颜色空间

RGB to YUV:

```
Y = ( (  66 * R + 129 * G +  25 * B + 128) >> 8) + 16
U = ( ( -38 * R  - 74 * G + 112 * B + 128) >> 8) + 128
V = ( ( 112 * R  - 94 * G  - 18 * B + 128) >> 8) + 128
```

YUV to RGB:

```
C = Y - 16
D = U - 128
E = V - 128

R = clip(( 298 * C + 409 * E + 128) >> 8)
G = clip(( 298 * C - 100 * D - 208 * E + 128) >> 8)
B = clip(( 298 * C + 516 * D + 128) >> 8)
```

clip( ): clipping the data to [0, 255].

3）零偏置

对于灰度级是 2^n 的像素，通过减去 2^{n-1}，将无符号的整数值编程有符号数

对于 n=8，即将 0~255 的值域，通过减去 128，转换为值域在-128~127 之间的值

目的：使像素的绝对值出现 3 位 10 进制的概率大大减少

4）量化

因为人眼对亮度信号比对色差信号更敏感，因此使用了两种量化表：亮度量化值和色差量化值

根据人烟的视觉特性，对低频分量采取较细的量化，对高频分量采取较粗的量化

真正的量化表=缩放因子×基本量化表

质量因子≤50：缩放因子=50/质量因子

质量因子>50：缩放因子=2-质量因子/50

5）DC 系数的差分编码

8×8 图像块经过 DCT 变换之后得到的 DC 直流系数有两个特点：系数的数值比较大，相邻 8×8 图像块的 DC 系数值变化不大：冗余

根据这个特点，JPEG 算法使用了差分脉冲调制编码（DPCM）技术，对相邻图像块之间量化 DC 系数的差值 DIFF 进行编码：$DIFF_k=DC_k-DC_{k-1}$

对 DIFF 用 Huffman 编码：分成类别，类似指数 Golomb 编码

类别 ID：一元码编码　　类内索引：采用定长码

6）AC 系数

Z 字扫描。由于经 DCT 变换后系数大多数集中在左上角，即低频分量区，因此采用 Z 字形按频率的高低顺序读出，可以出现很多连 0 的机会。可以使用游程编码。尤其在最后，如果都是 0，给出 EOB（End Of Block）即可。

在 JPEG 和 MPEG 编码中规定（run，level）为（连续 run 个 0，后面跟值为 level 的系数）。

编码：Run 最多 15 个，用 4 位表示 RRRR；Level 分成 16 个类别，用 4 位表示 SSSS 表示类别符号。对（RRRR，SSSS）联合用 Huffman 编码

解码重构：与编码相反。解码 Huffman 数据，解码 DC 差值，重构量化后的系数，DCT 逆变换，丢弃填充的行/列，反 0 偏置，对丢失的 CbCr 分量插值（下采样的逆过程），YCbCr→RGB

7）JPEG 文件的解析

由若干个必不可少的标记顺序连接构成整个文件

一定以 0xFFD8 开始，即表示图像开始 SOL（Start of Image）标记

一定以 0xFFD9 结束，表示图像结束 EOL（End of Image）标记

8）Huffman 表存储

Huffman 表的长度、类型（AC 或 DC）、索引（Index）、位表（bit table）、值表（value table）

FF C4 00 1D 00 00 03 01 01 01 01 01 01 01 00 00 00 00 00 00 04 05 06 03 02 01 00 09 07 08

红色部分 为哈夫曼表ID和表类型，其值0x00表示此部分数据描述的是DC直流0号表。

蓝色部分（16个字节）为不同位数的码字的数量。这16个数值实际意义为：没有1位的哈夫曼码字；2位的码字有3个；3位-9位的码字各有1个；没有9位或以上的码字。

绿色部分（10个字节）为编码内容。由蓝色部分数据知道，此哈夫曼树有0+3+1+1+1+1+1+1+1=10个叶子结点，即本字段应该有10个字节。这段数据表示10个叶子结点按从小到大排列，其权值依次为04、05、06、03、02、01、00、09、07、08（16进制）

序号	码长	码字	权值
1	2	00	04
2	2	01	05
3	2	10	06
4	3	110	03
5	4	1110	02
6	5	11110	01
7	6	111110	00
8	7	1111110	09
9	8	11111110	07
10	9	111111110	08

FF C4 00 3E 10 00 01 02 05 03 03 03 02 05 03 04 02 02 02 01 05 00 01 03 02 04 05 11 21 22 31 61 06 12 A1 32 41 62 13 51 23 42 71 81 91 15 52 63 07 14 33 53 16 43 08 B1 34 C1 24 D1 09 72 F0 A2

红色部分（1字节）为哈夫曼表ID和表类型，其值0x10表示此部分数据描述的是AC交流0号表。

蓝色部分（16字节）为不同位数的码字的数量。这16个数值实际意义为：没有1位的哈夫曼码字……。

绿色部分（3E-16-2-1=43字节）为编码内容。由蓝色部分数据知道，此哈夫曼树有（绿色数据相加）=43个叶子结点，即本字段应该有43个字节。这段数据表示43个叶子结点按从小到大排列，其权值依次为（16进制）

序号	码长	码字	权值
1	2	00	00
2	3	010	01
3	3	011	03
4	4	1000	02
5	4	1001	04
6	4	1010	05
7	4	1011	11
8	4	1100	21
9	5	11010	22
10	5	11011	31
11	5	11100	61
...	...	...	...
42	16		F0
43	16		A2

建立 Huffman 树/表

在独处 Huffman 表的数据后，就要建立 Huffman 树。具体方法为：

①第一个码字必定为 0

如果第一个码字位数为 1，则码字为 0；如果第一个码字位数为 2，则码字为 00；如此类推。

②从第二个码字开始，如果它和它前面的码字位数相同，则当前码字为它前面的码字加 1；如果它的位数比它前面的码字位数大，则当前码字是前面的码字加 1 后再在后边添若干个 0，直到满足位数长度为止。

直流：第一个例题建立的是 DC（0）Huffman 表，其权值就是解码时再需要读入的 bit 位数。这个再读入的位数通过查表得到真正的码值，表示直流系数

交流：对于交流系数，用交流 Huffman 树/表查的该码字对应的权值。权值的高 4 位表示当前数值前面有多少个连续的 0，低 4 位表示该交流分量数值的二进制位数，也就是接下来需要读入的位数。例如权值为 0X31，克表示为（3,1），表明此交流系数前面有 3 个 0，而此交流系数的具体值还需要再读入 1 个 bit 的码字，才能得到。

第七章 分析-综合编码

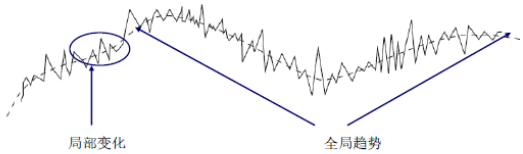
1、引言

基于块变换的编码将信源输出分解为不同频率子带，然后对不同频率的子带编码

子带编码：将原始信号分解为若干个子频带，对其分别进行编码处理后再合成为全频带信号

小波变换编码：使用更强大的非均匀分辨率滤波器对信号进行时间-频率局部分析与综合

2、信号的分解与合成



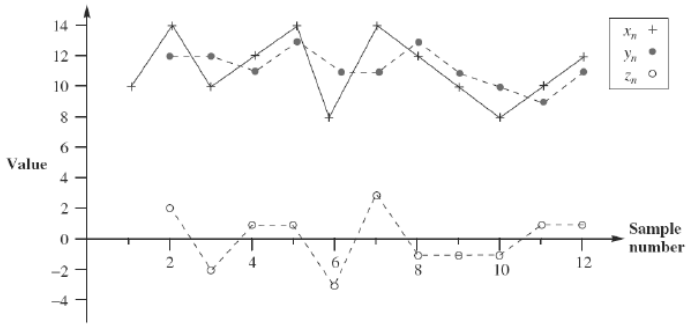
提取全局趋势：在一个滑动窗口上计算样本均值

$y_n = \frac{1}{2}(x_n + x_{n-1})$

- $\{y_n\}$ 比 $\{x_n\}$更适合用差分编码方式
- 为了得到x_n，还需计算z_n:

$z_n = x_n - y_n = x_n - \frac{1}{2}(x_n + x_{n-1}) = \frac{1}{2}(x_n - x_{n-1})$

- 对$\{y_n\}$和$\{z_n\}$可以采用不同的编码方式



$\{y_n - y_{n-1}\}$的动态范围为4

M-水平量化器

$\Delta = 4/M$

MaxQE = 2/M

$\{z_n\}$:

0 2 -2 1 1 -3 3 -1 -1 -1 1 1

动态范围 = 6

$\Delta = 6/M$

MaxQE = 3/M

至此为止

编码了**2倍**的数值，所以比特率也变成了2倍

失真更小：5/M

在两种情况下，最后传送数值的数目相同

两个子序列有不同的特征

因此可以采用不同的编码机制

编码更灵活，从而编码效率更高

还可以递归使用该分解方法，得到子序列

亦称为分析——综合

信号分解可利用数字滤波器实现

例7.1：

$\{x_n\}$: 10 14 10 12 14 8 14 12 10 8 10 12

$\{x_n - x_{n-1}\}$: 10 4 -4 2 2 **-6 6** -2 -2 -2 2 2

Sol #1：*M*-水平量化

$M = 2^m$

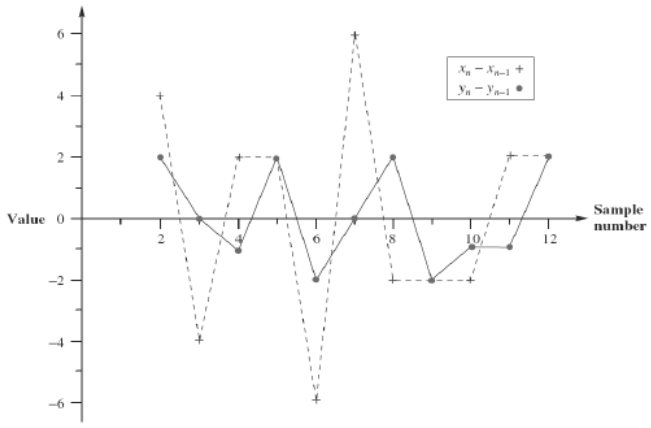
$\Delta = 12/M$

最大量化误差(MaxQE) = $\Delta/2 = 6/M$

Sol #2：利用 $\{y_n\}$ $\{z_n\}$

$\{y_n\}$: 10 12 12 11 13 11 11 13 11 10 9 11

$\{y_n - y_{n-1}\}$: 10 2 0 -1 **2 -2** 0 2 -2 -1 -1 2



降低比特率

将 $\{y_n\}$ 分解成 $\{y_{2n}\}$ & $\{y_{2n-1}\}$

将 $\{z_n\}$分解成 $\{z_{2n}\}$ & $\{z_{2n-1}\}$

只需传送偶数下标的子序列（或奇数下标子序列）

$y_{2n} = \frac{1}{2}(x_{2n} + x_{2n-1})$

$z_{2n} = \frac{1}{2}(x_{2n} - x_{2n-1})$

$y_{2n} + z_{2n} = x_{2n}$

$y_{2n} - z_{2n} = x_{2n-1}$

- 我们又回到了原始比特率，量化误差仍为5/M

3、滤波器组的设计

数字滤波器回顾

- 当前和过去输入（和输出）的加权组合

$$y_n = \sum_{i=0}^{N} a_i x_{n-i} + \sum_{i=1}^{M} b_i y_{n-i}$$

（a_i：滤波器系数）

脉冲：

- $\{x_n\} = 1\ 0\ 0\ 0\ \ldots$

脉冲响应

- 有限 → 有限脉冲响应(Finite Impulse Response, FIR)
- 无限 →无限脉冲响应(Infinite Impulse Response, IIR)
- 注意：$b_i = 0$ ➔ **FIR**

例7.2 $a_0 = 1.25, a_1 = 0.5$

$$x_n = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}$$

$$y_0 = a_0 x_0 + a_1 x_{-1} = 1.25$$
$$y_1 = a_0 x_1 + a_1 x_0 = 0.5$$
$$y_n = 0, \quad n < 0 \text{ or } n > 1$$

脉冲响应 $\{h_n\}$

$$h_n = \begin{cases} 1.25 & n = 0 \\ 0.5 & n = 1 \\ 0 & \text{otherwise} \end{cases}$$

注意 $\{h_i\}$ ➔ $\{a_i\}$

- 脉冲响应函数完全决定滤波器

例7.3 如例7.1中

$$y_n = \frac{1}{2}(x_n + x_{n-1}) \qquad z_n = \frac{1}{2}(x_n - x_{n-1})$$

$\{y_n\}$ $\{z_n\}$ 序列为2抽头(tap) FIR滤波器：

$$h_n = \begin{cases} 0.5 & n = 0 \\ 0.5 & n = 1 \\ 0 & \text{otherwise} \end{cases} \qquad h_n = \begin{cases} 0.5 & n = 0 \\ -0.5 & n = 1 \\ 0 & \text{otherwise} \end{cases}$$

例7.4 令 $a_0 = 1, b_1 = 0.9$

$$y_0 = a_0 x_0 + b_1 y_{-1} = 1(1) + 0.9(0) = 1$$
$$y_1 = a_0 x_1 + b_1 y_0 = 1(0) + 0.9(1) = 0.9$$
$$y_2 = a_0 x_2 + b_1 y_1 = 1(0) + 0.9(0.9) = 0.9^2$$
$$\vdots$$
$$y_n = (0.9)^n$$

$$h_n = \begin{cases} 0 & n < 0 \\ (0.9)^n & n \geq 0 \end{cases}$$

注意：这是一个**IIR**滤波器

脉冲响应函数完全规定了一个FIR & IIR滤波器：

$$y_n = \sum_{k=0}^{M} h_k x_{n-k}, (M < \infty \text{ for } FIR, M = \infty \text{ for IIR})$$

（卷积）

平稳性

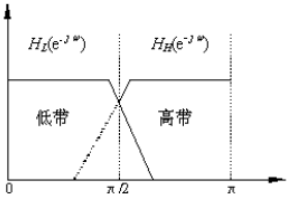
- 如果一个滤波器的输入是有限的意味着其输出也是有限的，则该滤波器是平稳的
- FIR滤波器通常是平稳的：加权平均
- IIR可能对有界的输入产生无界的输出。如：
  - $a_0 = 1, b_1 = 2$ ➔ $y_n = 2^n$
  - 虽然IIR滤波器可能不稳定，但能提供更快的截至和产生更少ripple

最常用的滤波器组

- 正交镜像滤波器(Quadrature mirror filters, QMF)

基本性质

- 如果低通滤波器由$\{h_n\}$给定，则高通滤波器由$\{(-1)^n h_{N-1-n}\}$给定
- QMF幅频特性简图



4、基本的子带编码系统

三大要素：

分析——综合滤波器组，比特分配机制，编码机制：PCM、ADPCM、VQ...

量化&编码

不同子带运载不同数量的感知信息

比特分配对总质量影响极大

感知边界之外的带宽可以被抛弃

不太重要的子带可以用较低码率下采样，且可以采用较粗糙的量化粗量化

运载最重要信息的子带用高质量编码

计算比特分配 R_k，使得给定码率 R 下失真 D 最小 $R = \frac{1}{M}\sum_{k=1}^{M} R_k$ $J_k = D_k + \lambda_k R_k$

5、MPEG 音频压缩

1）MPEG-1 声音的主要性能

输入为 PCM 信号，采样率为 32、44.1 或 48kHz，输出为 32kbps 到 384kbps

三个独立的压缩层次：Layer1、2、3，编码器复杂程度增加

2）编码器说明

输入声音信号经过一个多相滤波器组变换到多个子带，同时经过“心理声学模型”计算以频率为自变量的噪声掩蔽阈值。量化和编码部分用信掩比 SMR 决定分配给子带信号的量化位数，使量化噪声<掩比域值。最后通过数据帧包装将量化的自带样本和其它数据按照规定而帧格式组装成比特数据流。

①多相滤波器组，用来分割子带

划分子带的方法有两种：线性划分和非线性划分

线性划分可能一个子带覆盖好几个临界频带

②量化和编码——比例因子的取值和编码

对各个子带每 12 个样点金鼎一次比例因子计算。先定出 12 个样点中绝对值的最大值。查比例因子表中比这个最大值大的最小值作为比例因子。用 6 比特表示。

③数据帧包装

彩色空间转换实验

基本原理：
RGB 和 YUV 彩色空间的基础知识；数据类型的分析

1）不带参数的主函数定义 void main（）
带参数的主函数定义 void main（int argc，char*argv[ ]）
调用：命令名 参数 1 参数 2 … 参数 n
参数值：argc=n+1 argv[0]=命令名 argv[1]=参数 1 argv[2]=参数 2 … argv[n]=参数 n

2）文件相关指针
FILE *fp;
FILE *fopen（char *filename，char *mode）;
filename 是要打开的文件路径
mode 是要打开的模式
若成功，返回指向被打开文件的指针；若出错，返回空指针 NULL
举例：

打开模式	描　　述
r	只读，打开已有文件，不能写
w	只写，创建或打开，覆盖已有文件
a	追加，创建或打开，在已有文件末尾追加
r+	读写，打开已有文件
w+	读写，创建或打开，覆盖已有文件
a+	读写，创建或打开，在已有文件末尾追加
t	按文本方式打开 (缺省)
b	按二进制方式打开

```
//指针定义
    char *rgbFileName = NULL;
    FILE *rgbFile = NULL;
//参数获取
    rgbFileName = argv[1];
//打开函数
    rgbFile = fopen(rgbFileName, "rb");
    if (rgbFile == NULL)
    {
        printf("cannot find rgb file\n");
        exit(1);
    }
    else
    {
        printf("The input rgb file is %s\n", rgbFileName);
    }
```

函数	功能
fprintf	格式化输出
fscanf	格式化输入
putw	输出一个字
getw	输入一个字
fputs	输出字符串
fgets	输入字符串

fclose（FILE *fp）;
fp 是要关闭的文件指针
若成功，返回 0；若出错，返回 EOF（-1）
不用的文件应
feof（FILE *fp）;
fp 是文件指针
若文件结束，返回非零值；若文件尚未结束，返回 0
fputc（int c, FILE *fp）;
c 是要输出到文件的字符
fp 是文件指针
若成功，返回输出的字符；若失败，返回 EOF
fgetc（FILE *fp）;
fp 是文件指针
若成功，返回输入的字符；若失败或文件结束，返回 EOF

函数	功能
fputc	输出字符
fgetc	输入字符
putc	输出字符
getc	输入字符
fwrite	输出数据块
fread	输入数据块

fwrite/fread（void *buffer, size_t size, size_t count, FILE *fp）;

buffer 是要写/读的数据块地址

size 是要写/读的每个数据项的字节数

count 是要写/读的数据项数量

fp 是文件指针

若成功，返回实际写/读的数据项数量；若失败，一般返回 0

举例：

```
//开辟缓存空间
    rgbBuf = (u_int8_t*)malloc(frameWidth * frameHeight * 3);
    yBuf = (u_int8_t*)malloc(frameWidth * frameHeight);
    uBuf = (u_int8_t*)malloc((frameWidth * frameHeight) / 4);
    vBuf = (u_int8_t*)malloc((frameWidth * frameHeight) / 4);
//读取数据
    fread(rgbBuf, 1, frameWidth * frameHeight * 3, rgbFile);
//写入数据
    fwrite(yBuf, 1, frameWidth * frameHeight, yuvFile);
    fwrite(uBuf, 1, (frameWidth * frameHeight) / 4, yuvFile);
    fwrite(vBuf, 1, (frameWidth * frameHeight) / 4, yuvFile);
```

fprintf/fscanf（FILE *fp, char *format[,address/argument,...]）;

fputs(char *s, FILE *fp);

若成功，返回输出字符个数；若失败，返回 EOF

fgets（char *s, int n, FILE *fp）;

若成功，返回 s 首地址；若失败，返回 NULL

从 fp 输入字符串到 s 中，输入 n-1 个字符，或遇到换行符或 EOF 为止，读完后自动在字符串末尾添加 '\0'

rewind（FILE *fp）;

fp 是文件指针

使文件位置指针重新返回文件开头

fseek（FILE *fp, long offset, int whence）;

fp 是文件指针

offset 是偏移量

whence 是起始位置

随机改变文件的位置指针

SEEK_SET(0)是文件开始

SEEK_CUR(1)是文件当前位置

SEEK_END(2)是文件末尾

ftell（FILE *fp）;

fp 是文件指针

返回 fp 所指向文件中的读写位置

3）动态数组和指针

①申请空间

```
char *name;
name = (char *)malloc(20);

float *pf;
pf = (float *)malloc(sizeof(float)*20);

double *pd;
pd = (double *)malloc(sizeof(double)*50);
```

②赋值

```
int *pi;
pi = (int *)malloc(sizeof(int)*100);
*pi = 0;
*(pi+1) = 1;
*(pi+2) = 2;

int j;
for(j=0; j<100; j++)
    *(pi+j) = 0;
```

举例：

```
for (i = 0; i < size; i++)
{
    g = b + 1;
    r = b + 2;
    *y = (unsigned char)( RGBYUV02990[*r]     + RGBYUV05870[*g]     + RGBYUV01140[*b]);
    *u = (unsigned char)(- RGBYUV01684[*r]    - RGBYUV03316[*g]     + (*b)/2              + 128);
    *v = (unsigned char)(   (*r)/2            - RGBYUV04187[*g]     - RGBYUV00813[*b] + 128);
    b += 3;
    y ++;
    u ++;
    v ++;
}
```

③空间回收

```
free（filename）;
```

4）RGB to YUV 文件转换

①流程分析

程序初始化（打开两个文件、定义变量和缓冲区等）

读取 RGB 文件，抽取 RGB 数据写入缓冲区

调用 RGB2YUV 的函数实现 RGB 到 YUV 数据的转换

写 YUV 文件

程序收尾工作（关闭文件，释放缓冲区）

②代码分析

定义变量

```
static int init_done = 0;
long i, j, size;
unsigned char *r, *g, *b;
unsigned char *y, *u, *v;
unsigned char *pu1, *pu2, *pv1, *pv2, *psu, *psv;
unsigned char *y_buffer, *u_buffer, *v_buffer;
unsigned char *sub_u_buf, *sub_v_buf;
```

调用快速查找表

```
if (init_done == 0)
{
    InitLookupTable();
    init_done = 1;
}
```

定义指针、开辟缓存空间

```
y_buffer  = (unsigned char *)y_out;
sub_u_buf = (unsigned char *)u_out;
sub_v_buf = (unsigned char *)v_out;
u_buffer  = (unsigned char *)malloc(x_dim * y_dim);
v_buffer  = (unsigned char *)malloc(x_dim * y_dim);
```

```
b = (unsigned char *)rgb;
y = y_buffer;
u = u_buffer;
v = v_buffer;
```

将 RGB 数据转换为 YUV 数据

```
for (i = 0; i < size; i++)
{
    g = b + 1;
    r = b + 2;
//定义指针位置，RGB文件的数据存储方式为一个点依次存储B、G、R信息
    *y = (unsigned char)(  RGBYUV02990[*r] + RGBYUV05870[*g] + RGBYUV01140[*b]);
    *u = (unsigned char)(- RGBYUV01684[*r] - RGBYUV03316[*g] + (*b)/2          + 128);
    *v = (unsigned char)(  (*r)/2          - RGBYUV04187[*g] - RGBYUV00813[*b] + 128);
//利用公式将RGB信息转换为YUV信息，其中RGBYUVxxxxx为快速查找表对应值
    b += 3;
    y ++;
    u ++;
    v ++;
//使指针跳到新的位置
```

将转换为的 U、V 数据转换为 4：2：0 格式

```
for (j = 0; j < y_dim/2; j ++)
{
    psu = sub_u_buf + j * x_dim / 2;
    psv = sub_v_buf + j * x_dim / 2;
//用于输出的U、V数据指针
    pu1 = u_buffer + 2 * j * x_dim;          //加2j行
    pu2 = u_buffer + (2 * j + 1) * x_dim;    //加2j+1行
    pv1 = v_buffer + 2 * j * x_dim;          //加2j行
    pv2 = v_buffer + (2 * j + 1) * x_dim;    //加2j+1行
//1为四格U/V值中左上角值，2为左下角值
    for (i = 0; i < x_dim/2; i ++)
    {
        *psu = (*pu1 + *(pu1+1) + *pu2 + *(pu2+1)) / 4;
        *psv = (*pv1 + *(pv1+1) + *pv2 + *(pv2+1)) / 4;
//对取得的四个值做平均
        psu ++;
        psv ++;
        pu1 += 2;
        pu2 += 2;
        pv1 += 2;
        pv2 += 2;
//使指针跳到新位置
    }
```

附查找表

```
void InitLookupTable()
{
    int i;

    for (i = 0; i < 256; i++) RGBYUV02990[i] = (float)0.2990 * i;
    for (i = 0; i < 256; i++) RGBYUV05870[i] = (float)0.5870 * i;
    for (i = 0; i < 256; i++) RGBYUV01140[i] = (float)0.1140 * i;
    for (i = 0; i < 256; i++) RGBYUV01684[i] = (float)0.1684 * i;
    for (i = 0; i < 256; i++) RGBYUV03316[i] = (float)0.3316 * i;
    for (i = 0; i < 256; i++) RGBYUV04187[i] = (float)0.4187 * i;
    for (i = 0; i < 256; i++) RGBYUV00813[i] = (float)0.0813 * i;
}
```

5）YUV to RGB 文件转换

①流程分析

程序初始化（打开两个文件、定义变量和缓冲区）

读取 YUV 文件，抽取 YUV 数据写入缓冲区

调用 YUV2RGB 的函数实现 YUV 到 RGB 数据的转换

写 RGB 文件

程序收尾工作（关闭文件，释放缓冲区）

②代码分析

定义变量

调用快速查找表

定义指针、开辟缓存空间

将 YUV 数据转换为 RGB 数据

附查找表

图像文件的读写和转换

1）BMP 文件格式

位图文件头BITMAPFILEHEADER
位图信息头BITMAPINFOHEADER
调色板Palette
实际的位图数据ImageData

```
Typedef   struct   tagBITMAPFILEHEADER {
WORD    bfType;         /*  说明文件的类型   */
DWORD   bfSize;         /*  说明文件的大小，用字节为单位   */
                        /*注意此处的字节序问题
WORD    bfReserved1;    /*  保留，设置为 0 */
WORD    bfReserved2;    /*  保留，设置为 0 */
DWORD   bfOffBits;      /*  说明从 BITMAPFILEHEADER 结构
                        开始到实际的图像数据之间的字  节
                        偏移量  */
}    BITMAPFILEHEADER;
typedef   struct   tagBITMAPINFOHEADER {
DWORD   biSize;         /*  说明结构体所需字节数  */
LONG    biWidth;        /*  以像素为单位说明图像的宽度  */
LONG    biHeight;       /*  以像素为单位说明图像的高速  */
WORD    biPlanes;       /*  说明位面数，必须为 1 */
WORD    biBitCount;     /*  说明位数/像素，1、2、4、8、24 */
DWORD   biCompression;      /*  说明图像是否压缩及压缩类型
                            BI_RGB，BI_RLE8，BI_RLE4，BI_BITFIELDS */
DWORD   biSizeImage;    /*   以字节为单位说明图像大小 ，必须是 4          的整数倍*/
LONG    biXPelsPerMeter;    /*   目标设备的水平分辨率，像素/米  */
LONG    biYPelsPerMeter;    /*目标设备的垂直分辨率，像素/米  */
DWORD   biClrUsed;      /*  说明图像实际用到的颜色数，如果为 0
                    则颜色数为 2 的 biBitCount 次方  */
DWORD   biClrImportant;     /*说明对图像显示有重要影响的颜色
                            索引的数目，如果是 0，表示都重要。*/
}   BITMAPINFOHEADER;
typedef   struct   tagRGBQUAD {
BYTE    rgbBlue;        /*指定蓝色分量*/
BYTE    rgbGreen;       /*指定绿色分量*/
BYTE    rgbRed;         /*指定红色分量*/
BYTE    rgbReserved;    /*保留，指定为 0*/
}   RGBQUAD;
```

调用文件头信息头

```
#include<windows.h>
BITMAPFILEHEADER File_header;
BITMAPINFOHEADER Info_header;
//    read file & info header
if(fread(&File_header,sizeof(BITMAPFILEHEADER),1,bmpFile) != 1)
{
    printf("read file header error!");
    exit(0);
}
```

```
if (File_header.bfType != 0x4D42)
{
    printf("Not bmp file!");
    exit(0);
}
else
{
    printf("this is a %s\n",File_header.bfType);
}
if(fread(&Info_header,sizeof(BITMAPINFOHEADER),1,bmpFile) != 1)
{
    printf("read info header error!");
    exit(0);
}
//  end read header
```

调用调色板

```
RGBQUAD     *pRGB=(RGBQUAD*)malloc(sizeof(RGBQUAD)*(unsigned char)pow(2,info_h.biBitCount));
if(!MakePalette(pFile,file_h,info_h,pRGB))
    printf("No palette!");
bool MakePalette(FILE * pFile,BITMAPFILEHEADER &file_h,BITMAPINFOHEADER & info_h,RGBQUAD *pRGB_out)
{
    if ((file_h.bfOffBits - sizeof(BITMAPFILEHEADER) - info_h.biSize) == sizeof(RGBQUAD)*pow(2,info_h.biBitCount))
    {
        fseek(pFile,sizeof(BITMAPFILEHEADER)+info_h.biSize,0);
        fread(pRGB_out,sizeof(RGBQUAD),(unsigned int)pow(2,info_h.biBitCount),pFile);
        return true;
    }
    else
        return false;
}
```

2）BMP to YUV 文件转换

①转换流程

程序初始化（打开两个文件、定义变量和缓冲区等）

读取 BMP 文件，抽取或生成 RGB 数据写入缓冲区

读位图文件头（判断是否可读出、判断是否是 BMP 文件）

读位图信息头（判断是否读出）

判断像素的实际点阵数

开辟缓冲区，读数据，倒序存放

根据每像素位数的不同，执行不同的操作

（8bit 以下，构造调色板，位与移位取像素数据查调色板，写 RGB 缓冲区

16bit，位与移位取像素数据转换为 8bit/彩色分量，写 RGB 缓冲区

24/32bit，直接取像素数据，写 RGB 缓冲区）

16bitBMP：

```
for (Loop = 0;Loop < height * width;Loop +=2)
{
    *rgbDataOut = (Data[Loop]&0x1F)<<3;
    *(rgbDataOut + 1) = ((Data[Loop]&0xE0)>>2) + ((Data[Loop+1]&0x03)<<6);
    *(rgbDataOut + 2) = (Data[Loop+1]&0x7C)<<1;
```

```
        rgbDataOut +=3;
    }
    1~8bitBMP：
    int shiftCnt = 1;
    while (mask)
    {
        unsigned char index=mask
        ==0xFF?Data[Loop]:((Data[Loop]&mask)>>(8-shiftCnt* info_h.biBitCount));
        * rgbDataOut = pRGB[index].rgbBlue;
        * (rgbDataOut+1) = pRGB[index].rgbGreen;
        * (rgbDataOut+2) = pRGB[index].rgbRed;
        if(info_h.biBitCount == 8)
            mask = 0;
        Else
            mask >>= info_h.biBitCount;
        rgbDataOut+=3;
        shiftCnt ++;
    }
```

调用 RGB2YUV 的函数实现 RGB 到 YUV 数据的转换

采用部分查找表法，提高运行效率

写 YUV 文件

程序收尾工作（关闭文件，释放缓冲区）

②代码分析

```
/*每一扫描行的字节数必须是4的整数倍，与DWORD对齐
  若扫描行字节数不为4的整数倍，要补零为4的整数倍
  difference用来计算实际字节数和用于图像信息存储的字节数之差*/

if (((info_h.biWidth*info_h.biBitCount/8)%4) == 0)
    Width = info_h.biWidth*info_h.biBitCount/8;
else
    Width = (info_h.biWidth*info_h.biBitCount+31)/32*4;

if ((info_h.biHeight % 2) == 0)
    Height= info_h.biHeight;
else
    Height = info_h.biHeight + 1;

difference = Width-info_h.biWidth*info_h.biBitCount/8;
```

```
if (info_h.biBitCount == 24)
{
    for (k=0; k<Height*Width; k++)
    {
    /***********补零***********/
        if (difference != 0)
        {
            if (difference == 1)
            {
                if ((k+1)%Width == 0)
                    continue;
            }
            else if (difference == 2)
            {
                if ((k+1)%Width == 0 || (k+2)%Width == 0)
                    continue;
            }
            else
            {
                if ((k+1)%Width == 0 || (k+2)%Width == 0 || (k+3)%Width == 0)
                    continue;
            }
        }
        *rgb = *(bmpBuf+k);
        rgb++;
    }
}
```

```
if (!flip)//因为BMP倒序存储数据，所以需要倒序读取
{
    for (j = 0; j < y_dim; j ++)
    {
        y = y_buffer + (y_dim - j - 1) * x_dim;
        u = u_buffer + (y_dim - j - 1) * x_dim;
        v = v_buffer + (y_dim - j - 1) * x_dim;
        for (i = 0; i < x_dim; i ++) {
            g = b + 1;
            r = b + 2;
            *y = (unsigned char)(  RGBYUV02990[*r] + RGBYUV05870[*g] + RGBYUV01140[*b]);
            *u = (unsigned char)(- RGBYUV01684[*r] - RGBYUV03316[*g] + (*b)/2          + 128);
            *v = (unsigned char)(  (*r)/2          - RGBYUV04187[*g] - RGBYUV00813[*b] + 128);
            b += 3;
            y ++;
            u ++;
            v ++;
        }
    }
}
```

Huffman 编解码算法实现与压缩效率分析

1）Huffman 编码的数据结构

```
Typedef    struct    huffman_node_tag        //Huffman 节点标签
{
    unsigned char isLeaf            //是否为树叶
    unsigned long count;            //节点代表的符号加权和
    struct huffman_node_tag *parent;      //父节点指针
    union
    {
        struct
        {
           struct huffman_node_tag *zero, *one;     //子节点指针,分别代表 0,1 子节点指针
        };
        unsigned char symbol;   //节点代表的符号
    };
} huffman_node;

typedef    struct    huffman_code_tag        //Huffman 编码标签
{
    unsigned long numbits;        //该码所用的比特数
    unsigned char *bits;          //指向该码比特串的指针
} huffman_code;
```

2）Huffman 编码的流程

从指定文件中读取数据，统计每个符号发生的概率，并建立相应的树叶节点

注意在此处 pSF 代入的是地址

第一次扫描：统计文件中各个字符出现频率

构建 Huffman 树及生成 Huffman 码

按字符概率由小到大将对应节点排序

得到文件出现的字符种类数

构建 Huffman 树

对码树编码

将码表及其他必要信息写入输出文件

第二次扫描：对源文件进行编码并输出

JPEG 原理分析及 JPEG 解码器的调试

1）JPEG 文件格式介绍

SOI start of image 图像开始

标记代码 2 字节 固定值 0xFFD8

APP0 appllication 应用程序保留标记 0

标记代码 2 字节 固定值 0xFFE0

包含 9 个具体字段

①数据长度 2 字节 ①~⑨9 个字段的总长度

②标识符 5 字节 固定值 0x4A46494600，即字符串“JFIF0”

③版本号 2 字节 一般是 0x0102，表示 JFIF 的版本号 1.2

④X 和 Y 的密度单位 1 字节 只有三个值可选

0：无单位；1：点数/英寸；2：点数/厘米

⑤X 方向像素密度 2 字节 取值范围未知

⑥Y 方向像素密度 2 字节 取值范围未知

⑦缩略图水平像素数目 1 字节 取值范围未知

⑧缩略图垂直像素数目 1 字节 取值范围未知

⑨缩略图 RGB 位图 长度可能是 3 的倍数 缩略图 RGB 位图数据

DQT define quantization table 定义量化表

标记代码 2 字节 固定值 0xFFDB

包含 9 个具体字段：

①数据长度 2 字节 字段①和多个字段②的总长度

②量化表 数据长度-2 字节

a) 精度及量化表 ID 1 字节

高 4 位：精度，只有两个可选值 0：8 位；1：16 位

低 4 位：量化表 ID，取值范围为 0～3

b) 表项 (64×(精度+1))字节

例如 8 位精度的量化表,其表项长度为 64×（0+1）=64 字节

本标记段中，字段②可以重复出现，表示多个量化表，但最多只能出现 4 次

SOF0 start of frame 帧图像开始

标记代码 2 字节 固定值 0xFFC0

包含 9 个具体字段：

①数据长度 2 字节 ①~⑥六个字段的总长度

②精度 1 字节 每个数据样本的位数

通常是 8 位，一般软件都不支持 12 位和 16 位

③图像高度 2 字节 图像高度（单位：像素）

④图像宽度 2 字节 图像宽度（单位：像素）

⑤颜色分量数 1 字节 只有 3 个数值可选

1：灰度图；3：YCrCb 或 YIQ；4：CMYK

而 JFIF 中使用 YCrCb，故这里颜色分量数恒为 3

⑥颜色分量信息 颜色分量数×3 字节（通常为 9 字节）

a)颜色分量 ID 1 字节

b)水平/垂直采样因子 1 字节

高 4 位：水平采样因子

低 4 位：垂直采样因子

c)量化表 1 字节 当前分量使用的量化表的 ID

DHT　define huffman table　定义哈夫曼表

标记代码　2 字节　固定值 0xFFC4

包含 2 个具体字段：

①数据长度　　2 字节

②huffman 表　　数据长度-2 字节

表 ID 和表类型　　1 字节

高 4 位：类型，只有两个值可选

0：DC 直流；1：AC 交流

低 4 位：哈夫曼表 ID，

注意，DC 表和 AC 表分开编码

不同位数的码字数量　16 字节

编码内容　　16 个不同位数的码字数量之和（字节）

本标记段中，字段②可以重复出现（一般 4 次），也可以只出现 1 次。

SOS　start of scan　扫描开始 12 字节

标记代码　2 字节　固定值 0xFFDA

包含 2 个具体字段：

①数据长度　　2 字节　①~④两个字段的总长度

②颜色分量数　　1 字节　应该和 SOF 中的字段⑤的值相同，即：

1：灰度图是；3：　YCrCb 或 YIQ；4：CMYK。

③颜色分量信息

a)颜色分量 ID　　1 字节

b)直流/交流系数表号　　1 字节

高 4 位：直流分量使用的哈夫曼树编号

低 4 位：交流分量使用的哈夫曼树编号

④压缩图像数据

a)谱选择开始　　1 字节　固定值 0x00

b)谱选择结束　　1 字节　固定值 0x3F

c)谱选择　　1 字节　在基本 JPEG 中总为 00

EOI　end of image　图像结束

标记代码　2 字节　固定值 0xFFD9

2）JPEG 文件解码流程

读入文件的相关信息

JPEG 文件解码流程

SOI(0xFFD8)

APP0(0xFFE0)

[APPn(0xFFEn)]可选

DQT(0xFFDB)

SOF0(0xFFC0)

DHT(0xFFC4)

SOS(0xFFDA)

压缩数据

EOI(0xFFD9)

Huffman 表存储方式

在标记段 DHT 内，包含了一个或多个的哈夫曼表。对于单个哈夫曼表，应该包括三部分：

①Huffman 表 ID 和表类型

0x00 表示 DC 直流 0 号表　0x01 表示 DC 直流 1 号表

0x10 表示 AC 交流 0 号表　0x11 表示 AC 交流 1 号表

②不同位数的码字数量

JPEG 的 Huffman 编码只能是 1~16 位。这个字段的 16 个字节分别表示 1~16 位的编码码字在 Huffman 树中的个数。

③编码内容

这个字段记录了 Huffman 树中各个叶子结点的权。所以，上一字段的 16 个数值之和就应该是本字段的长度，也就是 Huffman 树种叶子结点的个数。

初步了解图像数据流的结构

颜色分量单元的内部解码

直流系数的差分编码

反量化&反 Zig-Zag 编码

反离散余弦变化

3）JPEG 文件解码程序实现

对 Huffman 码字解码前不知道码字的长度，解码有两种方法，Huffman 码树遍历和 lookup 查找表

4）对函数 build_huffman_table 的分析

第一步：得到 code_size.（huffsize）

第二步：得到 code.（huffcode）

第三步：得到 code_size 查找表。

Table->code_size[val] = code_size;

第四步：得到 value 查找表（lookup）。

table->lookup[code++] = val

第五步：当码字长度>9，slowtable 处理。

slowtable[0] = code;　slowtable[1] = val;　slowtable[2] = 0;

序号	码长	码字	权值
1	2	00	04
2	2	01	05
3	2	10	06
4	3	110	03
5	4	1110	02
6	5	11110	01
7	6	111110	00
8	7	1111110	09
9	8	11111110	07
10	9	111111110	08

index	lookup	value	Code_size
1	001100	04	2
2	010001	05	2
3	100101	06	2
4	110010	03	3
5	111011	02	4
6	111101	01	5
7	111110	00	6
8			
9			
10			

第一步　第二步　见trace及程序　查找表

图像文件的读写和转换

1）BMP 文件格式


```
Typedef   struct   tagBITMAPFILEHEADER {
WORD    bfType;         /*  说明文件的类型   */
DWORD   bfSize;         /*  说明文件的大小，用字节为单位   */
                        /*注意此处的字节序问题
WORD    bfReserved1;    /*  保留，设置为 0 */
WORD    bfReserved2;    /*  保留，设置为 0 */
DWORD   bfOffBits;      /*  说明从 BITMAPFILEHEADER 结构
                        开始到实际的图像数据之间的字  节
                        偏移量  */
}    BITMAPFILEHEADER;
typedef   struct   tagBITMAPINFOHEADER {
DWORD   biSize;         /*  说明结构体所需字节数  */
LONG    biWidth;        /*  以像素为单位说明图像的宽度  */
LONG    biHeight;       /*  以像素为单位说明图像的高速  */
WORD    biPlanes;       /*  说明位面数，必须为 1 */
WORD    biBitCount;     /*  说明位数/像素，1、2、4、8、24 */
DWORD   biCompression;      /*  说明图像是否压缩及压缩类型
                            BI_RGB，BI_RLE8，BI_RLE4，BI_BITFIELDS */
DWORD   biSizeImage;    /*   以字节为单位说明图像大小 ，必须是 4          的整数倍*/
LONG    biXPelsPerMeter;    /*   目标设备的水平分辨率，像素/米  */
LONG    biYPelsPerMeter;    /*目标设备的垂直分辨率，像素/米  */
DWORD   biClrUsed;      /*  说明图像实际用到的颜色数，如果为 0
                    则颜色数为 2 的 biBitCount 次方  */
DWORD   biClrImportant;     /*说明对图像显示有重要影响的颜色
                            索引的数目，如果是 0，表示都重要。*/
}   BITMAPINFOHEADER;
typedef   struct   tagRGBQUAD {
BYTE    rgbBlue;        /*指定蓝色分量*/
BYTE    rgbGreen;       /*指定绿色分量*/
BYTE    rgbRed;         /*指定红色分量*/
BYTE    rgbReserved;    /*保留，指定为 0*/
}   RGBQUAD;
```

调用文件头信息头

```
#include<windows.h>
BITMAPFILEHEADER File_header;
BITMAPINFOHEADER Info_header;
//    read file & info header
if(fread(&File_header,sizeof(BITMAPFILEHEADER),1,bmpFile) != 1)
{
    printf("read file header error!");
    exit(0);
}
```

```
if (File_header.bfType != 0x4D42)
{
    printf("Not bmp file!");
    exit(0);
}
else
{
    printf("this is a %s\n",File_header.bfType);
}
if(fread(&Info_header,sizeof(BITMAPINFOHEADER),1,bmpFile) != 1)
{
    printf("read info header error!");
    exit(0);
}
//  end read header
```

调用调色板

```
RGBQUAD     *pRGB=(RGBQUAD*)malloc(sizeof(RGBQUAD)*(unsigned char)pow(2,info_h.biBitCount));
if(!MakePalette(pFile,file_h,info_h,pRGB))
    printf("No palette!");
bool MakePalette(FILE * pFile,BITMAPFILEHEADER &file_h,BITMAPINFOHEADER & info_h,RGBQUAD *pRGB_out)
{
    if ((file_h.bfOffBits - sizeof(BITMAPFILEHEADER) - info_h.biSize) == sizeof(RGBQUAD)*pow(2,info_h.biBitCount))
    {
        fseek(pFile,sizeof(BITMAPFILEHEADER)+info_h.biSize,0);
        fread(pRGB_out,sizeof(RGBQUAD),(unsigned int)pow(2,info_h.biBitCount),pFile);
        return true;
    }
    else
        return false;
}
```

2）BMP to YUV 文件转换

①转换流程

程序初始化（打开两个文件、定义变量和缓冲区等）

读取 BMP 文件，抽取或生成 RGB 数据写入缓冲区

读位图文件头（判断是否可读出、判断是否是 BMP 文件）

读位图信息头（判断是否读出）

判断像素的实际点阵数

开辟缓冲区，读数据，倒序存放

根据每像素位数的不同，执行不同的操作

（8bit 以下，构造调色板，位与移位取像素数据查调色板，写 RGB 缓冲区

16bit，位与移位取像素数据转换为 8bit/彩色分量，写 RGB 缓冲区

24/32bit，直接取像素数据，写 RGB 缓冲区）

16bitBMP：

```
for (Loop = 0;Loop < height * width;Loop +=2)
{
    *rgbDataOut = (Data[Loop]&0x1F)<<3;
    *(rgbDataOut + 1) = ((Data[Loop]&0xE0)>>2) + ((Data[Loop+1]&0x03)<<6);
    *(rgbDataOut + 2) = (Data[Loop+1]&0x7C)<<1;
```

```
        rgbDataOut +=3;
    }
    1~8bitBMP：
    int shiftCnt = 1;
    while (mask)
    {
        unsigned char index=mask
        ==0xFF?Data[Loop]:((Data[Loop]&mask)>>(8-shiftCnt* info_h.biBitCount));
        * rgbDataOut = pRGB[index].rgbBlue;
        * (rgbDataOut+1) = pRGB[index].rgbGreen;
        * (rgbDataOut+2) = pRGB[index].rgbRed;
        if(info_h.biBitCount == 8)
            mask = 0;
        Else
            mask >>= info_h.biBitCount;
        rgbDataOut+=3;
        shiftCnt ++;
    }
```

调用 RGB2YUV 的函数实现 RGB 到 YUV 数据的转换

采用部分查找表法，提高运行效率

写 YUV 文件

程序收尾工作（关闭文件，释放缓冲区）

②代码分析

```
/*每一扫描行的字节数必须是4的整数倍，与DWORD对齐
  若扫描行字节数不为4的整数倍，要补零为4的整数倍
  difference用来计算实际字节数和用于图像信息存储的字节数之差*/

if (((info_h.biWidth*info_h.biBitCount/8)%4) == 0)
    Width = info_h.biWidth*info_h.biBitCount/8;
else
    Width = (info_h.biWidth*info_h.biBitCount+31)/32*4;

if ((info_h.biHeight % 2) == 0)
    Height= info_h.biHeight;
else
    Height = info_h.biHeight + 1;

difference = Width-info_h.biWidth*info_h.biBitCount/8;
```

```
if (info_h.biBitCount == 24)
{
    for (k=0; k<Height*Width; k++)
    {
    /***********补零***********/
        if (difference != 0)
        {
            if (difference == 1)
            {
                if ((k+1)%Width == 0)
                    continue;
            }
            else if (difference == 2)
            {
                if ((k+1)%Width == 0 || (k+2)%Width == 0)
                    continue;
            }
            else
            {
                if ((k+1)%Width == 0 || (k+2)%Width == 0 || (k+3)%Width == 0)
                    continue;
            }
        }
        *rgb = *(bmpBuf+k);
        rgb++;
    }
}
```

```
if (!flip)//因为BMP倒序存储数据，所以需要倒序读取
{
    for (j = 0; j < y_dim; j ++)
    {
        y = y_buffer + (y_dim - j - 1) * x_dim;
        u = u_buffer + (y_dim - j - 1) * x_dim;
        v = v_buffer + (y_dim - j - 1) * x_dim;
        for (i = 0; i < x_dim; i ++) {
            g = b + 1;
            r = b + 2;
            *y = (unsigned char)(  RGBYUV02990[*r] + RGBYUV05870[*g] + RGBYUV01140[*b]);
            *u = (unsigned char)(- RGBYUV01684[*r] - RGBYUV03316[*g] + (*b)/2          + 128);
            *v = (unsigned char)(  (*r)/2          - RGBYUV04187[*g] - RGBYUV00813[*b] + 128);
            b += 3;
            y ++;
            u ++;
            v ++;
        }
    }
}
```