哈夫曼编码

姓名:陈扬	任课老师:魏振刚
专业:17 计算机	专业课程:数据结构
学号:17150011001	实验四：哈夫曼（Huffman）编码问题

一、题目描述

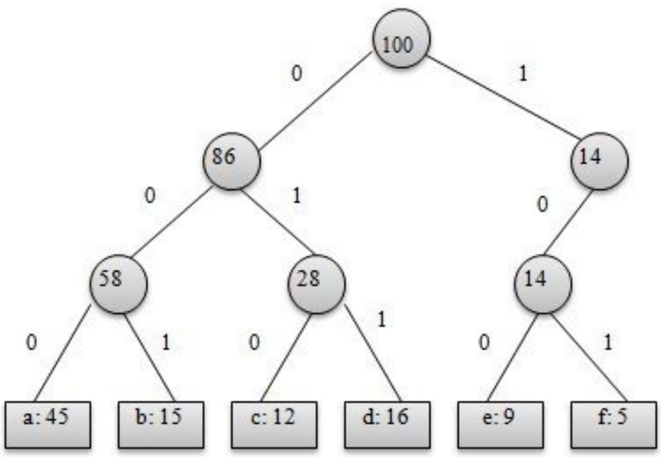
哈夫曼编码是广泛地用于数据文件压缩的十分有效的编码方法。其压缩率通常在20%～90%之间。哈夫曼编码算法用字符在文件中出现的频率表来建立一个用0，1串表示各字符的最优表示方式。一个包含100,000个字符的文件，各字符出现频率不同，如下表所示：

	a	b	c	d	e	f
频率(千次)	45	13	12	16	9	5
定长码	000	001	010	011	100	101
变长码	0	101	100	111	1101	1100

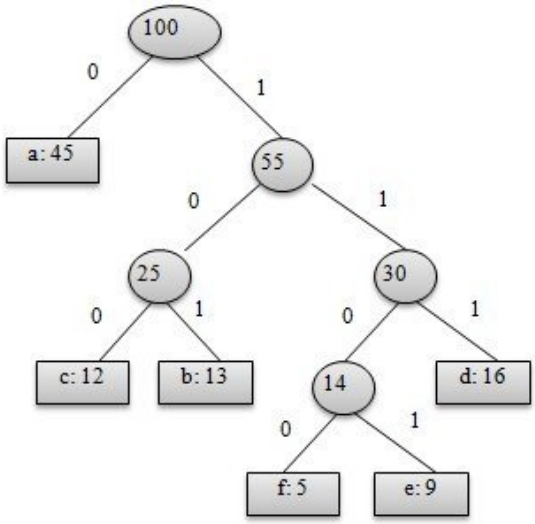
有多种方式表示文件中的信息，若用0,1码表示字符的方法，即每个字符用唯一的一个0,1串表示。若采用定长编码表示，则需要3位表示一个字符，整个文件编码需要300,000位；若采用变长编码表示，给频率高的字符较短的编码；频率低的字符较长的编码，达到整体编码减少的目的，则整个文件编码需要（45×1+13×3+12×3+16×3+9×4+5×4）×1000=224,000位，由此可见，变长码比定长码方案好，总码长减小约25%。

二、算法设计与分析

1、前缀码： 对每一个字符规定一个0,1串作为其代码，并要求任一字符的代码都不是其他字符代码的前缀。这种编码称为前缀码。编码的前缀性质可以使译码方法非常简单；例如001011101可以唯一的分解为0,0,101,1101，因而其译码为aabe。 译码过程需要方便的取出编码的前缀，因此需要表示前缀码的合适的数据结构。为此，可以用二叉树作为前缀码的数据结构：树叶表示给定字符；从树根到树叶的路径当作该字符的前缀码；代码中每一位的0或1分别作为指示某节点到左儿子或右儿子的“路标”。



a



b

图-1a 与固定长度编码对应的树; 图-1b 对应于最优前缀编码的树 从上图可以看出，表示最优前缀码的二叉树总是一棵完全二叉树，即树中任意节点都有2个儿子。图a表示定长编码方案不是最优的，其编码的二叉树不是一棵完全二叉树。在一般情况下，若C是编码字符集，表示其最优前缀码的二叉树中恰有|C|个叶子。每个叶子对应于字符集中的一个字符，该二叉树有|C|-1个内部节点。 给定编码字符集C及频率分布f,即C中任一字符c以频率f(c)在数据文件中出现。C的一个前缀码编码方案对应于一棵二叉树T。字符c在树T中的深度记为dT(c)。dT(c)也是字符c的前缀码长。则平均码长定义为：

$$B(T)=\sum_{c\in C}f(c)d_T(c) \tag{1}$$

使平均码长达到最小的前缀码编码方案称为C的最优前缀码。

2、构造哈夫曼编码：

哈夫曼提出构造最优前缀码的贪心算法，由此产生的编码方案称为哈夫曼编码。其构造步骤如下： (1)哈夫曼算法以自底向上的方式构造表示最优前缀码的二叉树T。 (2)算法以|C|个叶结点开始，执行|C|－1次的“合并”运算后产生最终所要求的树T。 (3)假设编码字符集中每一字符c的频率是f(c)。以f为键值的优先队列Q用在贪心选择时有效地确定算法当前要合并的2棵具有最小频率的树。一旦2棵具有最小频率的树合并后，产生一棵新的树，其频率为合并的2棵树的频率之和，并将新树插入优先队列Q。经过n－1次的合并后，优先队列中只剩下一棵树，即所要求的树T。

构造过程如图-2所示：

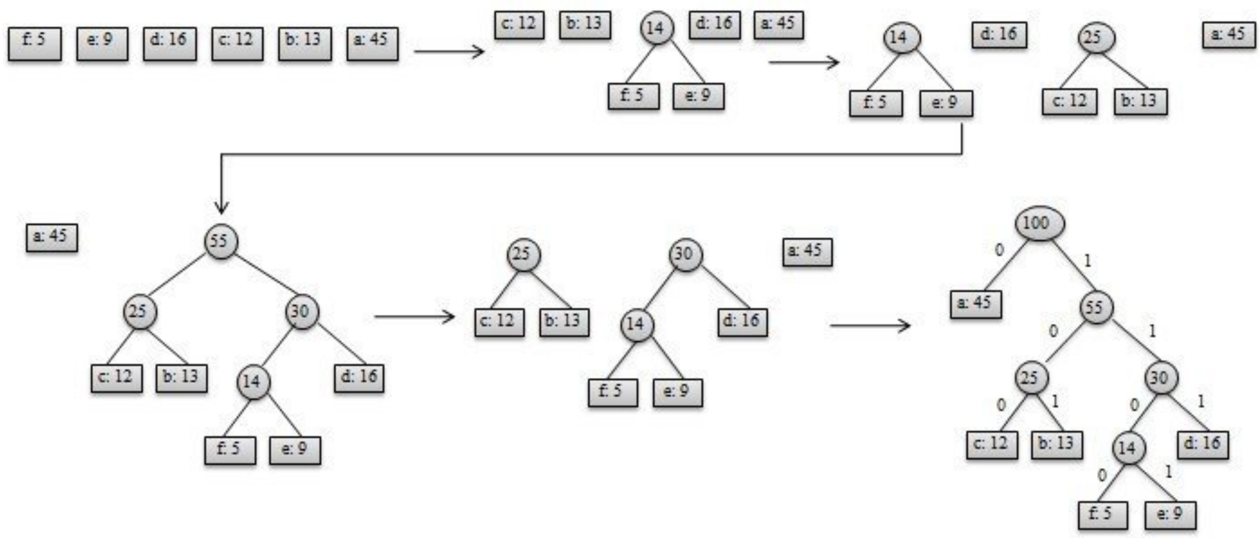


图-2 哈夫曼树构造过程

三、结果与分析

本实验以算法导论书中的例题为测试用例，来验证算法的正确性。即

实验结果截图如下图-7，结果与题目描述中给出的变长代码字一样：

	a	b	c	d	e	f
频率(千次)	45	13	12	16	9	5

四、实验总结

1、实验结果与给出的变长代码字一样，算法正确，且哈夫曼编码问题是一个贪心算法问题。采用哈夫曼编码技术可以最小化总的编码长度，从而实现数据文件的压缩存储。 2、构造好哈夫曼树后，可用排列树回溯法来打印哈夫曼编码，即遇到左子树向左走，vector添加记录0；遇到右子树向右走，vector添加记录1；走到叶子节点并打印出叶节点的编码后回溯，同时往上退一层，则vector弹出一个值。如此不断回溯下去，即可打印所有字符编码。

五、源代码

```
class Node(object):
    def __init__(self,name=None,value=None):
        self._name=name
        self._value=value
        self._left=None
        self._right=None
class HuffmanTree(object):
    def __init__(self,char_weights):
        self.a=[Node(part,char_weights[part]) for part in char_weights]

        while len(self.a)!=1:
```

```
                self.a.sort(key=lambda node:node._value,reverse=True)
                c=Node(value=(self.a[-1]._value+self.a[-2]._value))
                c._left=self.a.pop(-1)
                c._right=self.a.pop(-1)
                self.a.append(c)
            self.root=self.a[0]
            self.b=range(100)
        def pre(self,tree,length):
            node=tree
            if (not node):
                return
            elif node._name:
                print node._name + '\'s encode is :',
                for i in range(length):
                    print self.b[i],
                print '\n'
                return
            self.b[length]=0
            self.pre(node._left,length+1)
            self.b[length]=1
            self.pre(node._right,length+1)

        def get_code(self):
            self.pre(self.root,0)

if __name__=='__main__':
    with open("ACGAN.py","r") as f:
        read_file= f.read()
    print(str(read_file))
    s=str(read_file)
    resoult={}
    for i in set(s):
        resoult[i]=s.count(i)
    print(resoult)
    tree=HuffmanTree(resoult)
    tree.get_code()
```

六、实验结果

```
from __future__ import print_function, division

from keras.datasets import mnist
from keras.layers import Input, Dense, Reshape, Flatten, Dropout,
multiply
from keras.layers import BatchNormalization, Activation, Embedding,
ZeroPadding2D
from keras.layers.advanced_activations import LeakyReLU
```

```
from keras.layers.convolutional import UpSampling2D, Conv2D
from keras.models import Sequential, Model
from keras.optimizers import Adam

import matplotlib.pyplot as plt

import numpy as np

class ACGAN():
    def __init__(self):
        # Input shape
        self.img_rows = 28
        self.img_cols = 28
        self.channels = 1
        self.img_shape = (self.img_rows, self.img_cols, self.channels)
        self.num_classes = 10
        self.latent_dim = 100

        optimizer = Adam(0.0002, 0.5)
        losses = ['binary_crossentropy', 'sparse_categorical_crossentropy']

        # Build and compile the discriminator
        self.discriminator = self.build_discriminator()
        self.discriminator.compile(loss=losses,
            optimizer=optimizer,
            metrics=['accuracy'])

        # Build the generator
        self.generator = self.build_generator()

        # The generator takes noise and the target label as input
        # and generates the corresponding digit of that label
        noise = Input(shape=(self.latent_dim,))
        label = Input(shape=(1,))
        img = self.generator([noise, label])

        # For the combined model we will only train the generator
        self.discriminator.trainable = False

        # The discriminator takes generated image as input and determines validity
        # and the label of that image
        valid, target_label = self.discriminator(img)

        # The combined model  (stacked generator and discriminator)
        # Trains the generator to fool the discriminator
        self.combined = Model([noise, label], [valid, target_label])
        self.combined.compile(loss=losses,
```

```
            optimizer=optimizer)

    def build_generator(self):

        model = Sequential()

        model.add(Dense(128 * 7 * 7, activation="relu", input_dim=self.latent_dim))
        model.add(Reshape((7, 7, 128)))
        model.add(BatchNormalization(momentum=0.8))
        model.add(UpSampling2D())
        model.add(Conv2D(128, kernel_size=3, padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(momentum=0.8))
        model.add(UpSampling2D())
        model.add(Conv2D(64, kernel_size=3, padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Conv2D(self.channels, kernel_size=3, padding='same'))
        model.add(Activation("tanh"))

        model.summary()

        noise = Input(shape=(self.latent_dim,))
        label = Input(shape=(1,), dtype='int32')
        label_embedding = Flatten()(Embedding(self.num_classes, 100)(label))

        model_input = multiply([noise, label_embedding])
        img = model(model_input)

        return Model([noise, label], img)

    def build_discriminator(self):

        model = Sequential()

        model.add(Conv2D(16, kernel_size=3, strides=2, input_shape=self.img_shape, padding="same"))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dropout(0.25))
        model.add(Conv2D(32, kernel_size=3, strides=2, padding="same"))
        model.add(ZeroPadding2D(padding=((0,1),(0,1))))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dropout(0.25))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Conv2D(64, kernel_size=3, strides=2, padding="same"))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dropout(0.25))
```

```
        model.add(BatchNormalization(momentum=0.8))
        model.add(Conv2D(128, kernel_size=3, strides=1, padding="same"))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dropout(0.25))

        model.add(Flatten())
        model.summary()

        img = Input(shape=self.img_shape)

        # Extract feature representation
        features = model(img)

        # Determine validity and label of the image
        validity = Dense(1, activation="sigmoid")(features)
        label = Dense(self.num_classes+1, activation="softmax")(features)

        return Model(img, [validity, label])

    def train(self, epochs, batch_size=128, sample_interval=50):

        # Load the dataset
        (X_train, y_train), (_, _) = mnist.load_data()

        # Configure inputs
        X_train = (X_train.astype(np.float32) - 127.5) / 127.5
        X_train = np.expand_dims(X_train, axis=3)
        y_train = y_train.reshape(-1, 1)

        # Adversarial ground truths
        valid = np.ones((batch_size, 1))
        fake = np.zeros((batch_size, 1))

        for epoch in range(epochs):

            # ---------------------
            #  Train Discriminator
            # ---------------------

            # Select a random batch of images
            idx = np.random.randint(0, X_train.shape[0], batch_size)
            imgs = X_train[idx]

            # Sample noise as generator input
            noise = np.random.normal(0, 1, (batch_size, 100))

            # The labels of the digits that the generator tries to create
an
            # image representation of
```

```
            sampled_labels = np.random.randint(0, 10, (batch_size, 1))

            # Generate a half batch of new images
            gen_imgs = self.generator.predict([noise, sampled_labels])

            # Image labels. 0-9 if image is valid or 10 if it is generated (fake)
            img_labels = y_train[idx]
            fake_labels = 10 * np.ones(img_labels.shape)

            # Train the discriminator
            d_loss_real = self.discriminator.train_on_batch(imgs, [valid, img_labels])
            d_loss_fake = self.discriminator.train_on_batch(gen_imgs, [fake, fake_labels])
            d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

            # ---------------------
            #  Train Generator
            # ---------------------

            # Train the generator
            g_loss = self.combined.train_on_batch([noise, sampled_labels], [valid, sampled_labels])

            # Plot the progress
            print ("%d [D loss: %f, acc.: %.2f%%, op_acc: %.2f%%] [G loss: %f]" % (epoch, d_loss[0], 100*d_loss[3], 100*d_loss[4], g_loss[0]))

            # If at save interval => save generated image samples
            if epoch % sample_interval == 0:
                self.save_model()
                self.sample_images(epoch)

    def sample_images(self, epoch):
        r, c = 10, 10
        noise = np.random.normal(0, 1, (r * c, 100))
        sampled_labels = np.array([num for _ in range(r) for num in range(c)])
        gen_imgs = self.generator.predict([noise, sampled_labels])
        # Rescale images 0 - 1
        gen_imgs = 0.5 * gen_imgs + 0.5

        fig, axs = plt.subplots(r, c)
        cnt = 0
        for i in range(r):
            for j in range(c):
                axs[i,j].imshow(gen_imgs[cnt,:,:,0], cmap='gray')
                axs[i,j].axis('off')
```

```
                cnt += 1
        fig.savefig("images/%d.png" % epoch)
        plt.close()

    def save_model(self):

        def save(model, model_name):
            model_path = "saved_model/%s.json" % model_name
            weights_path = "saved_model/%s_weights.hdf5" % model_name
            options = {"file_arch": model_path,
                        "file_weight": weights_path}
            json_string = model.to_json()
            open(options['file_arch'], 'w').write(json_string)
            model.save_weights(options['file_weight'])

        save(self.generator, "generator")
        save(self.discriminator, "discriminator")


if __name__ == '__main__':
    acgan = ACGAN()
    acgan.train(epochs=14000, batch_size=32, sample_interval=200)

{'\t': 347, '\n': 212, ' ': 484, '#': 32, '"': 40, '%': 17, "'": 22, ')': 155, '(': 155, '+': 3, '*': 8, '-': 88, ',': 117, '/': 4, '.': 141, '1': 38, '0': 59, '3': 13, '2': 41, '5': 12, '4': 4, '7': 6, '6': 3, '9': 1, '8': 12, ':': 21, '=': 113, '>': 1, 'A': 11, 'C': 11, 'B': 8, 'E': 3, 'D': 25, 'G': 5, 'F': 5, 'I': 9, 'M': 4, 'L': 11, 'N': 8, 'P': 3, 'S': 8, 'R': 8, 'U': 8, 'T': 9, 'X': 7, '[': 29, 'Z': 2, ']': 29, '_': 140, 'a': 404, 'c': 105, 'b': 58, 'e': 501, 'd': 258, 'g': 114, 'f': 106, 'i': 333, 'h': 89, 'k': 29, 'j': 7, 'm': 228, 'l': 291, 'o': 283, 'n': 279, 'q': 3, 'p': 141, 's': 316, 'r': 245, 'u': 62, 't': 279, 'w': 14, 'v': 45, 'y': 32, 'x': 11, '{': 1, 'z': 27, '}': 1}
l's encode is : 0 0 0 0

('s encode is : 0 0 0 1 0

)'s encode is : 0 0 0 1 1

s's encode is : 0 0 1 0

i's encode is : 0 0 1 1

1's encode is : 0 1 0 0 0 0 0

"'s encode is : 0 1 0 0 0 0 1

2's encode is : 0 1 0 0 0 1 0

```

```
Z's encode is : 0 1 0 0 0 1 1 0 0 0 0

q's encode is : 0 1 0 0 0 1 1 0 0 0 1

F's encode is : 0 1 0 0 0 1 1 0 0 1

G's encode is : 0 1 0 0 0 1 1 0 1 0

6's encode is : 0 1 0 0 0 1 1 0 1 1 0

+'s encode is : 0 1 0 0 0 1 1 0 1 1 1

:'s encode is : 0 1 0 0 0 1 1 1

-'s encode is : 0 1 0 0 1 0

C's encode is : 0 1 0 0 1 1 0 0 0

A's encode is : 0 1 0 0 1 1 0 0 1

x's encode is : 0 1 0 0 1 1 0 1 0

L's encode is : 0 1 0 0 1 1 0 1 1

v's encode is : 0 1 0 0 1 1 1

    's encode is : 0 1 0 1

h's encode is : 0 1 1 0 0 0

''s encode is : 0 1 1 0 0 1 0 0

P's encode is : 0 1 1 0 0 1 0 1 0 0 0

E's encode is : 0 1 1 0 0 1 0 1 0 0 1

7's encode is : 0 1 1 0 0 1 0 1 0 1

8's encode is : 0 1 1 0 0 1 0 1 1

5's encode is : 0 1 1 0 0 1 1 0 0

3's encode is : 0 1 1 0 0 1 1 0 1

D's encode is : 0 1 1 0 0 1 1 1

c's encode is : 0 1 1 0 1 0

f's encode is : 0 1 1 0 1 1
```

```

a's encode is : 0 1 1 1



's encode is : 1 0 0 0 0

z's encode is : 1 0 0 0 1 0 0 0

j's encode is : 1 0 0 0 1 0 0 1 0 0

X's encode is : 1 0 0 0 1 0 0 1 0 1

w's encode is : 1 0 0 0 1 0 0 1 1

k's encode is : 1 0 0 0 1 0 1 0

]'s encode is : 1 0 0 0 1 0 1 1

='s encode is : 1 0 0 0 1 1

m's encode is : 1 0 0 1 0

g's encode is : 1 0 0 1 1 0

b's encode is : 1 0 0 1 1 1 0

0's encode is : 1 0 0 1 1 1 1

 's encode is : 1 0 1 0

,'s encode is : 1 0 1 1 0 0

['s encode is : 1 0 1 1 0 1 0 0

U's encode is : 1 0 1 1 0 1 0 1 0 0

R's encode is : 1 0 1 1 0 1 0 1 0 1

4's encode is : 1 0 1 1 0 1 0 1 1 0 0

/'s encode is : 1 0 1 1 0 1 0 1 1 0 1

>'s encode is : 1 0 1 1 0 1 0 1 1 1 0 0 0

9's encode is : 1 0 1 1 0 1 0 1 1 1 0 0 1

}'s encode is : 1 0 1 1 0 1 0 1 1 1 0 1 0

{'s encode is : 1 0 1 1 0 1 0 1 1 1 0 1 1
```

```
M's encode is : 1 0 1 1 0 1 0 1 1 1 1
u's encode is : 1 0 1 1 0 1 1
r's encode is : 1 0 1 1 1
e's encode is : 1 1 0 0
d's encode is : 1 1 0 1 0
B's encode is : 1 1 0 1 1 0 0 0 0 0
*'s encode is : 1 1 0 1 1 0 0 0 0 1
S's encode is : 1 1 0 1 1 0 0 0 1 0
N's encode is : 1 1 0 1 1 0 0 0 1 1
y's encode is : 1 1 0 1 1 0 0 1
#'s encode is : 1 1 0 1 1 0 1 0
%'s encode is : 1 1 0 1 1 0 1 1 0
T's encode is : 1 1 0 1 1 0 1 1 1 0
I's encode is : 1 1 0 1 1 0 1 1 1 1
_'s encode is : 1 1 0 1 1 1
t's encode is : 1 1 1 0 0
n's encode is : 1 1 1 0 1
p's encode is : 1 1 1 1 0 0
.'s encode is : 1 1 1 1 0 1
o's encode is : 1 1 1 1 1
```