重 庆 交 通 大 学
学 生 实 验 报 告

实验课程名称 信息论与编码理论

开课实验中心 数学与统计学实验教学中心

开课学院 数学与统计学院

专业年级 信息与计算科学 2018 级 02 班

姓　　名 曾晶晶

学　　号 631810040303

任课老师 王 利 敏

开课时间 2021—2022 学年第 1 学期

实验一　香农编码

一、实验目的

1. 通过本实验进一步加深对香农编码基本原理及特点的理解
2. 熟练掌握香农编码的方法步骤
3. 能够通过上机实验利用编程软件实现香农编码

二、实验要求

输入离散无记忆信源符号以及对应概率分布，对该信源进行香农编码，输出对应信源符号的码字、码长，以及信源编码的平均码长、信源熵和编码效率。

三、基本原理

设有离散无记忆信源$\begin{pmatrix} X \\ P(X) \end{pmatrix} = \{\begin{matrix} a_1, & a_2, & \cdots, & a_i, & \cdots, & a_n \\ p(a_1), & p(a_2), & \cdots, & p(a_i), & \cdots, & p(a_n) \end{matrix}\}, \sum_{i=1}^{n} p(a_i) = 1$。二进制香农码的编码步骤如下：

1. 将信源符号按概率从大到小的顺序排列，为方便起见，令$p(a_1) \geq p(a_2) \geq \cdots \geq p(a_n)$；
2. 令 $\mathrm{p(a_0) = 0}$，用$\mathrm{p_a(a_j)(j = i + 1)}$表示第 i 个码字的累加概率，则

$$\mathrm{p_a(a_j)} = \sum_{\mathrm{i=0}}^{\mathrm{n-1}} \mathrm{p(a_i)}, \quad j = 1,2,\cdots n$$

3. 确定满足下列不等式的整数k_i，并令k_i为第i个码字的长度

$$-log_2 p(a_i) \leq k_i < 1 - log_2 p(a_i)$$

4. 将$p(a_i)$用二进制表示，并取小数点后k_i位作为符号a_i的编码。

四、程序代码

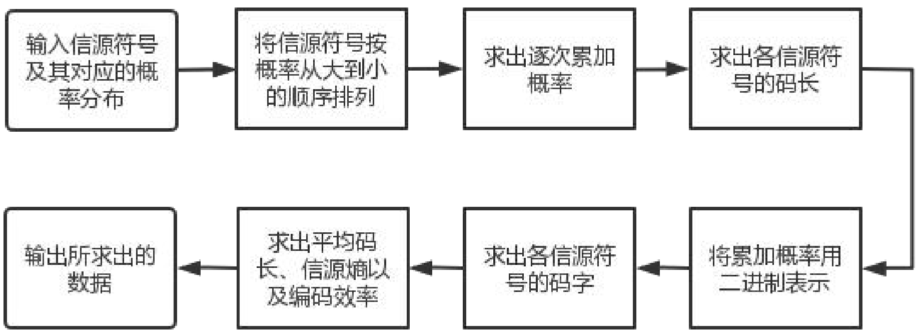
```
# 学号：631810040303
# 姓名：曾晶晶
# 编写时间：2021/11/16
import math

source = {}
# 0.25,0.25,0.20,0.15,0.10,0.05
# a1,a2,a3,a4,a5,a6
pro_sum = []    # 概率累加和
```

```
k = {}    # 对应码长
length_k = 0    # 平均码长
H = 0    # 信源熵


def init():
    source_symbol = input('请输入信源符号（中间以逗号隔开）：')
    source_probability = eval(input('请输入信源符号对应的概率（中间以逗号隔开,不要有空格）：'))
    ss_list = source_symbol.split(",")
    assert len(ss_list) == len(source_probability), '信源模型错误，符号个数和概率对不上'
    source = dict(zip(ss_list, source_probability))
    print('信源模型为:{}'.format(source))
    return source


def probability_summation():
    m = 0.0
    pro_sum.append(m)
    for i in list(source.values()):
        pro_sum.append(round(m + float(i), 4))
        m += float(i)
        if len(pro_sum) == len(source):
            break
    print("累加概率依次为：{}".format(pro_sum))


def code_length(k1):
    for item in source:
        k1[item] = math.ceil(math.log(float(source.get(item)), 2) * (-1))
    print('码长：{}'.format(k))
    return k1

def average_length(lk, s, kd):
    for item in s:
        lk += float(kd[item]) * float(s[item])
    print("平均码长为：{:.3}(bit/sign)".format(lk))
    return lk


def Hx(h, s):
    for item in s:
        h += float(s[item]) * math.log(float(s[item]), 2) * (-1)
    print('信源熵：{:.5}(bit/sign)'.format(h))
```

```
    return h


# 将十进制小数转为二进制小数
def float_to_bin(px, lk):
    b = []
    while True:
        px *= 2
        b.append(1 if px >= 1 else 0)
        px -= int(px)
        if px == 0 and len(b) != lk:
            b.append('0' * (lk - len(b)))
            break
        elif len(b) == lk:
            break
    return b


def codeword():
    code = source.copy()   # 对应码字
    i = 0
    for item in source:
        e = ''
        j = 0
        bin = float_to_bin(float(pro_sum[i]), k[item])
        while len(e) != k[item]:
            e += str(bin[j])
            j += 1
        code[item] = e
        i += 1
    print('码字：{}'.format(code))


def efficiency(hx, lk):
    print('编码效率：{:.2%}'.format(hx / round(lk, 3)))

if __name__ == '__main__':
    # 利用字典的形式存储信源符号及其概率
    print('以下为香农编码过程')
    source = init()   # 输入信源概率模型
    source = sorted(source.items(), key=lambda kv: kv[1], reverse=True)   # 将概率值从大到小
排序
    source = dict(source)
    print(source)
```

```
print("将概率值从大到小排序:{}".format(source))
probability_summation()   # 求概率累加和
k = source.copy()
k = code_length(k)   # 求对应码长
length_k = average_length(length_k, source, k)   # 求平均码长
h = Hx(H, source)   # 求信源熵
codeword()   # 求码字
efficiency(h, length_k)   # 求编码效率
```

五、程序流程图



六、实验内容及结果

实验内容：

有一单符号离散无记忆信源$\begin{pmatrix} X \\ P(X) \end{pmatrix} = \begin{Bmatrix} a_1, & a_2, & a_3, & a_4, & a_5, & a_6 \\ 0.25, & 0.25, & 0.2, & 0.15, & 0.1, & 0.05 \end{Bmatrix}$，对该信源编二进制香农码，并计算其信源熵、平均码长和编码效率。

实验结果：

```
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
以下为香农编码过程
请输入信源符号（中间以逗号隔开）：>? a1,a2,a3,a4,a5,a6
请输入信源符号对应的概率（中间以逗号隔开,不要有空格）：>? 0.25,0.25,0.20,0.15,0.10,0.05
信源模型为:{'a1': 0.25, 'a2': 0.25, 'a3': 0.2, 'a4': 0.15, 'a5': 0.1, 'a6': 0.05}
{'a1': 0.25, 'a2': 0.25, 'a3': 0.2, 'a4': 0.15, 'a5': 0.1, 'a6': 0.05}
将概率值从大到小排序:{'a1': 0.25, 'a2': 0.25, 'a3': 0.2, 'a4': 0.15, 'a5': 0.1, 'a6': 0.05}
累加概率依次为: [0.0, 0.25, 0.5, 0.7, 0.85, 0.95]
码长: {'a1': 2, 'a2': 2, 'a3': 3, 'a4': 3, 'a5': 4, 'a6': 5}
平均码长为: 2.7(bit/sign)
信源熵: 2.4232(bit/sign)
码字: {'a1': '00', 'a2': '01', 'a3': '100', 'a4': '101', 'a5': '1101', 'a6': '11110'}
编码效率: 89.75%
```

七、实验心得

香农码考虑了信源的统计特性，使经常出现的信源符号对应较短的码字，使信源的平均码长缩短，从而实现了对信源的压缩。香农码有系统的、唯一的编码方法，但在很多情况下，编码效率不是很高。

当信源符号出现的概率正好为 2 的负幂次方时，采用香农编码能够达到 100%的编码效率。

香农编码的效率不高，实用性不大，但对其他编码方法有很好的理论指导意义。一般情况下，按照香农编码方法编出来的码，其平均码长不是最短的，即不是紧致码(最佳码)。只有当信源符号的概率分布使不等式左边的等号成立时，编码效率才达到最高。

实验二　费诺编码

一、实验目的

1. 通过本实验进一步加深对费诺编码基本原理及特点的理解

2. 熟练掌握费诺编码的方法步骤

3. 能够通过上机实验利用编程软件实现费诺编码

二、实验要求

输入离散无记忆信源符号以及对应概率分布，对该信源进行费诺编码，输出对应信源符号的码字、码长，以及信源编码的平均码长、信源熵和编码效率。

三、基本原理

1. 将信源符号按概率从大到小的顺序排列，不失一般性，令$p(a_1) \geq p(a_2) \geq \cdots \geq p(a_n)$。

2. 按编码进制数将概率分组，使每组概率和尽可能接近或相等。如编二进制码就分成两组，编 m 进制码就分成 m 组。

3. 给每组分配一位码元。

4. 将每一分组再按同样原则划分，重复步骤 2 和 3，直至概率不再可分为止。

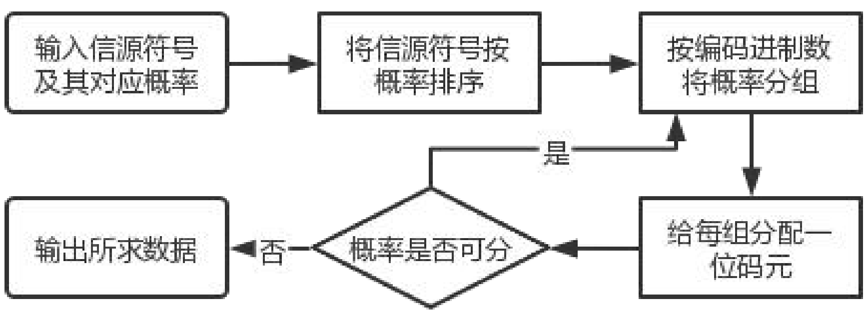
四、程序代码

```
# 学号：631810040303
# 姓名：曾晶晶
# 编写时间：2021/11/17
from Shannon import init, average_length, Hx, efficiency

Fs = {}
# 0.25,0.25,0.20,0.15,0.10,0.05
# a1,a2,a3,a4,a5,a6
k = {}    # 码长
length_k = 0    # 平均码长
code = {}    # 符号及其对应的码字集合
H = 0    # 信源熵
# 编码空间
Fs_code = {}
```

```
def codeword_Fano(p, fcode):
    if len(p) == 1:
        return 1

    # 最佳分组位置
    flag = 1
    find_position = 0
    sum1 = 0
    # 记录差值
    i = 0
    for item in p:
        sum2 = 0
        i += 1
        sum1 += float(p[item])
        for it in reversed(p):
            if item == it:
                break
            sum2 += float(p[it])
        difference = abs(sum2 - sum1)
        if difference < flag:
            flag = difference
            find_position = i
        else:
            break

    # 分组
    j = 0
    leftgroup = p.copy()
    rightgroup = p.copy()
    for item in p:
        if j < find_position:
            fcode[item] += '0'
            del rightgroup[item]
        else:
            fcode[item] += '1'
            del leftgroup[item]
        j += 1

    # 递归编码
    codeword_Fano(leftgroup, fcode)
    codeword_Fano(rightgroup, fcode)

    # 返回编码空间
    return fcode
```

```
def code_length2(k, Fs_code):
    for item in Fs_code:
        k[item] = len(Fs_code[item])
    print('码长： {}'.format(k))
    return k


if __name__ == '__main__':
    # 利用字典的形式存储信源符号及其概率
    # 输入信源概率模型
    print('以下为费诺编码过程')
    Fs = init()
    # 将概率值从大到小排序
    Fs = sorted(Fs.items(), key=lambda kv: kv[1], reverse=True)
    Fs = dict(Fs)
    print(Fs)
    print("将概率值从大到小排序:{}".format(Fs))
    # 信源编码初始化
    Fs_code = Fs.copy()
    for item in Fs_code:
        Fs_code[item] = ''
    # 求码字
    Fs_code = codeword_Fano(Fs, Fs_code)
    print("信源符号及对应码字为:{}".format(Fs_code))
    # 求对应码长
    k = Fs.copy()
    k = code_length2(k, Fs_code)
    length_k = average_length(length_k, Fs, k)
    H = Hx(H, Fs)   # 求信源熵
    efficiency(H, length_k)   # 求编码效率
```

五、程序流程图



六、实验内容及结果

实验内容：

有一单符号离散无记忆信源$\begin{pmatrix}X\\P(X)\end{pmatrix}=\left\{\begin{matrix}a_1, & a_2, & a_3, & a_4, & a_5, & a_6\\ 0.25, & 0.25, & 0.2, & 0.15, & 0.1, & 0.05\end{matrix}\right\}$，对该信源编二进制费诺码，并计算其信源熵、平均码长和编码效率。

实验结果：

```
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
以下为费诺编码过程
请输入信源符号（中间以逗号隔开）：>? a1,a2,a3,a4,a5,a6
请输入信源符号对应的概率（中间以逗号隔开,不要有空格）：>? 0.25,0.25,0.20,0.15,0.10,0.05
>?
信源模型为:{'a1': 0.25, 'a2': 0.25, 'a3': 0.2, 'a4': 0.15, 'a5': 0.1, 'a6': 0.05}
将概率值从大到小排序:{'a1': 0.25, 'a2': 0.25, 'a3': 0.2, 'a4': 0.15, 'a5': 0.1, 'a6': 0.05}
信源符号及对应码字为:{'a1': '00', 'a2': '01', 'a3': '10', 'a4': '110', 'a5': '1110', 'a6': '1111'}
码长：{'a1': 2, 'a2': 2, 'a3': 2, 'a4': 3, 'a5': 4, 'a6': 4}
平均码长为：2.45(bit/sign)
信源熵：2.4232(bit/sign)
编码效率：98.91%
```

七、实验心得

费诺编码后的费诺码要比香农码的平均码长小，消息传输速率大，编码效率高。费诺码比较适合于对分组概率相等或接近的信源编码。

费诺码的编码方法都不唯一，费诺码也可以编 m 进制码，但 m 越大，信源的符号数越多，可能的编码方案就越多，编码过程就越复杂，有时短码未必能得到充分利用。

费诺编码属于概率匹配编码，具有如下特点：

（1）概率大，则分解次数少；概率小则分解次数多，这符合最佳编码原则。

（2）码字集合是唯一的。

（3）分解之后先得码字后得码长。

实验三　哈夫曼编码

一、实验目的

1. 通过本实验进一步加深对哈夫曼编码基本原理及特点的理解

2. 熟练掌握哈夫曼编码的方法步骤

3. 能够通过上机实验利用编程软件实现哈夫曼编码

二、实验要求

输入离散无记忆信源符号以及对应概率分布，对该信源进行哈夫曼编码，输出对应信源符号的码字、码长，以及信源编码的平均码长、信源熵和编码效率。

三、基本原理

1. 将信源符号按概率从大到小的顺序排列，为方便起见，令$p(a_1) \geq p(a_2) \geq \cdots \geq p(a_n)$。

2. 给两个概率最小的信源符号$p(a_{n-1})$和$p(a_n)$各分配一个码位“0”和“1”，将这两个信源符号合并为一个新符号，并用这两个最小的概率之和作为新符号的概率，结果得到一个只包含$(n-1)$个心愿符号的新信源，称为信源的第一次缩减信源，用S_1表示。

3. 将缩减信源S_1的符号仍按概率从大到小的顺序排列，重复步骤（2），得到只含$(n-2)$个符号的缩减信源S_2。

4. 重复上述步骤，直至缩减信源只剩下两个符号为止，此时所剩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，依编码路径向前返回，就得到各信源符号所对应的码字。

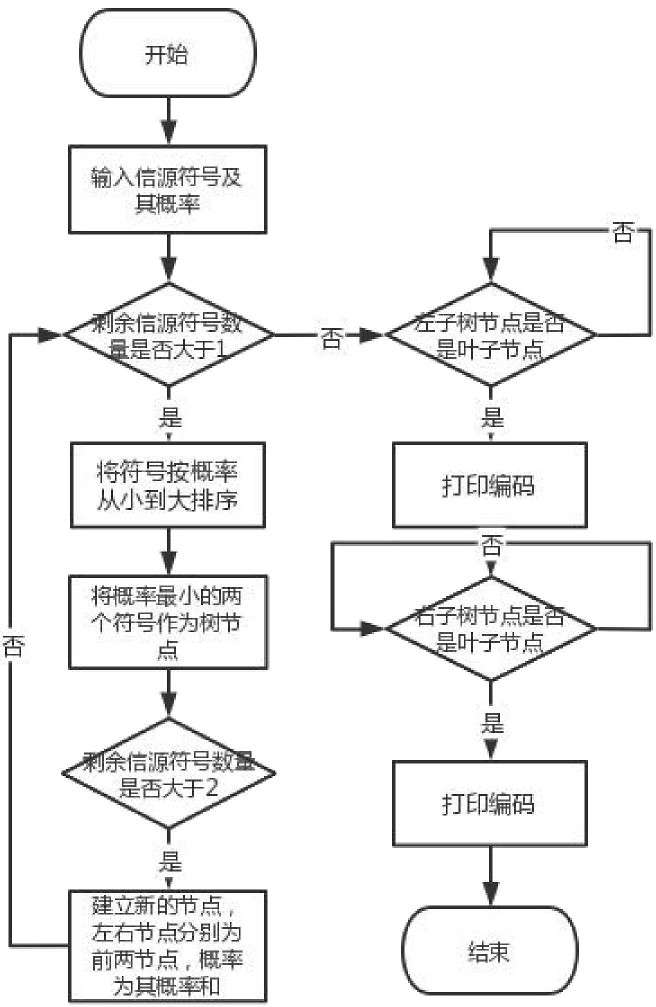
四、程序代码

```
# 学号：631810040303
# 姓名：曾晶晶
# 编写时间：2021/11/17
from Fano import code_length2
from Shannon import init, Hx, efficiency


def huffmanCode(root, tree, rootCode='', codeDict={}, depth=1, res=0):
    # 对左子树进行处理：如果是叶子节点，就打印编码；否则递归
    if len(root['left'][0]) / 2 == 1:
```

```
            codeDict[root['left'][0]] = '0' + rootCode
            res += (len(rootCode) + 1) * root['left'][1]    # 计算平均位数
        else:
            codeDict, res = huffmanCode(tree[root['left'][0]], tree, '0' + rootCode, codeDict, depth + 1, res)

        # 对右子树进行处理：如果是叶子节点，就打印编码；否则递归
        if len(root['right'][0]) / 2 == 1:
            codeDict[root['right'][0]] = '1' + rootCode
            res += (len(rootCode) + 1) * root['right'][1]    # 计算平均位数
        else:
            codeDict, res = huffmanCode(tree[root['right'][0]], tree, '1' + rootCode, codeDict, depth + 1, res)

        return codeDict, res


print('以下为哈夫曼编码过程')
Hfs = init()
Hs = sorted(Hfs.items(), key=lambda kv: kv[1])    # 将概率值从大到小排序

# 建立哈夫曼树
tree = {}
while len(Hs) > 1:
    # 1 根据权重排序
    Hs.sort(key=lambda kv: kv[1])

    # 2 选出最小的两个节点，分别作为左子树，右子树
    l = (Hs[0])
    r = (Hs[1])

    if len(Hs) > 2:
        tree[l[0] + r[0]] = {'left': l, 'right': r}

        # 3 用新节点置换这两个节点
        Hs = Hs[2:]
        Hs.append((l[0] + r[0], l[1] + r[1]))
    else:
        tree['root'] = {'left': l, 'right': r}
        break

# 编码
code, res = huffmanCode(tree['root'], tree)
print('码字：{}'.format(code))
```

```
# 对应码长及平均码长
k = Hfs.copy()
k = code_length2(k, code)
length_k = res / sum(Hfs.values())
print('平均码长：{:.3}'.format(res / sum(Hfs.values())))

# 信息熵 H
H = 0
H = Hx(H, Hfs)

efficiency(H, length_k)    # 编码效率
```

五、程序流程图



六、实验内容及结果

实验内容：

有一单符号离散无记忆信源$\begin{pmatrix} X \\ P(X) \end{pmatrix} = \{\begin{matrix} a_1, & a_2, & a_3, & a_4, & a_5, & a_6 \\ 0.25, & 0.25, & 0.2, & 0.15, & 0.1, & 0.05 \end{matrix}\}$，对该信源编二进制费诺码，并计算其信源熵、平均码长和编码效率。

实验结果：

```
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
以下为哈夫曼编码过程
请输入信源符号（中间以逗号隔开）：>? a1,a2,a3,a4,a5,a6
请输入信源符号对应的概率（中间以逗号隔开,不要有空格）：>? 0.25,0.25,0.20,0.15,0.10,0.05
信源模型为:{'a1': 0.25, 'a2': 0.25, 'a3': 0.2, 'a4': 0.15, 'a5': 0.1, 'a6': 0.05}
码字：{'a3': '00', 'a1': '10', 'a2': '01', 'a4': '011', 'a6': '0111', 'a5': '1111'}
码长：{'a1': 2, 'a2': 2, 'a3': 2, 'a4': 3, 'a5': 4, 'a6': 4}
平均码长：2.45
信源熵：2.4232(bit/sign)
编码效率：98.91%
```

七、实验心得

赫夫曼码的编码方法都不唯一。在赫夫曼编码过程中，对缩减信源符号按概率由大到小的顺序重新排列时，应使合并后的新符号尽可能排在靠前的位置，这样可使合并后的新符号重复编码次数减少，使短码得到充分利用。

赫夫曼码的码字（各符号的代码）是异前置码字，即任一码字不会是另一码字的前面部分，这使各码字可以连在一起传送，中间不需另加隔离符号，只要传送时不出错，收端仍可分离各个码字，不致混淆。