کدینگ منبع

ابتدا از فایل داده شده احتمال وقوع کاراکتر ها را بدست می آوریم و به ازای هر کاراکتر الفبا ، یک نود با alphabet معادل کاراکتر و value معادل احتمال وقوع می سازیم.
این در تابع generate Nodes انجام می شود.

```
def generateNodes():
    alphabets = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p",
                 "q", "r", "s", "t", "u", "v", "w", "x", "y", "z"]

    fileStream = open("freq.txt", "r")
    frequencies = fileStream.read().split(",")

    Nodes = list()

    for i in range(0, len(alphabets)):
        Nodes.append(HuffNode(float(frequencies[i]), alphabets[i]))

    return Nodes
```

سپس برای محاسبه codeBook کافی است هر بار ، دو نودی که کمترین احتمال را دارند در نظر گرفته و آن ها را با هم مرج کنیم و یک نود پدر اضافه کنیم که اشاره گر چپ و راست آن، آن دو نودی است که مینیمم احتمال یعنی Value را داشته اند و آن دو نود را حذف کنیم، مقدار Value نود پدر برابر با جمع value نود های چپ و راست آن می باشد. این کار را آنقدر انجام می دهیم تا یک نود ریشه باقی بماند.
حال یک dictionary در نظر می گیریم و با پیمایش inorder درخت ایجاد شده code معادل هر کاراکتر را به دست می آوریم.

```
def inOrderIterator(root, dictionary, code):
    if root.left is None and root.right is None:
        dictionary[root.alphabet] = code
        return
    inOrderIterator(root.left, dictionary, code + "0")
    inOrderIterator(root.right, dictionary, code + "1")
```

```python
def generateHuffman(Nodes):
    if len(Nodes) == 0:
        return None
    if len(Nodes) == 1:
        return {Nodes[0].alphabet: 0}

    while len(Nodes) != 1:
        left = Nodes[extractMin(Nodes)]
        Nodes.remove(left)
        right = Nodes[extractMin(Nodes)]
        Nodes.remove(right)
        parent = HuffNode(left.value + right.value, left.alphabet + right.alphabet)
        parent.left = left
        parent.right = right
        Nodes.append(parent)

    root = Nodes[0]
    HuffmanCode = dict()
    code = ""
    inOrderIterator(root, HuffmanCode, code)
    return HuffmanCode
```

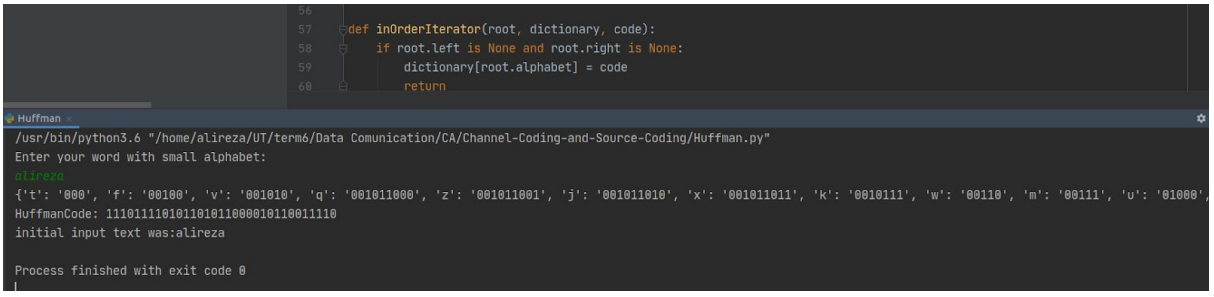
در انتها معادل کد هافمن عبارت ورودی را به دست می آوریم.

برای عمل decoding ، از dictionary خروجی بخش قبل استفاده می کنیم و به کمک آن درخت رو از نو میسازیم. در نهایت خروجی کد شده به دست آمده در بخش قبل را با پیمایش روی درخت ایجاد شده به دست می آوریم.
کد این بخش در زیر آورده شده است.

```python
def destinationDecoding(decoder, cipherText):
    root = decoder
    decodedData = list()
    for binary in cipherText:
        if binary == "0":
            root = root.left
        if binary == "1":
            root = root.right

        if root.left is None and root.right is None:
            decodedData.append(root.alphabet)
            root = decoder

    return "".join(decodedData)
```

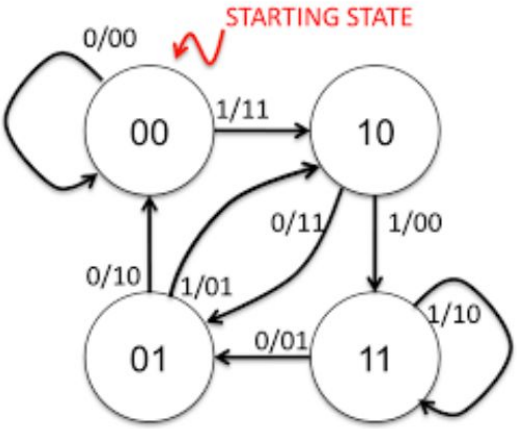
```python
def HuffmanDecoder(HuffmanCode):
    root = HuffNode(-1, None)
    rootPrime = root

    for codeWord in HuffmanCode:
        code = HuffmanCode[codeWord]

        for binary in code:
            if binary == "0":
                if root.left is None:
                    newNode = HuffNode(-1, None)
                    root.left = newNode
                root = root.left
            if binary == "1":
                if root.right is None:
                    newNode = HuffNode(-1, None)
                    root.right = newNode
                root = root.right
        root.alphabet = codeWord
        root = rootPrime

    return root
```

یک نمونه ورودی و خروجی این بخش در زیر آورده شده است.



کدینگ کانال:

در این بخش ابتدا ماشین حالت موجود در فایل رو پیاده سازی می کنیم.




```python
def channelCoding(cipherText):
    convolutionalCoding = dict()
    convolutionalCoding["00"] = dict()
    convolutionalCoding["01"] = dict()
    convolutionalCoding["10"] = dict()
    convolutionalCoding["11"] = dict()
    convolutionalCoding["00"]["0"] = ("00", "00")  # in state 00 if 0 is data ->  go to state 00 and parity is 00
    convolutionalCoding["00"]["1"] = ("10", "11")  # in state 00 if 1 is data ->  go to state 10 and parity is 11
    convolutionalCoding["10"]["0"] = ("01", "11")  # in state 10 if 0 is data ->  go to state 01 and parity is 11
    convolutionalCoding["10"]["1"] = ("11", "00")  # in state 10 if 1 is data ->  go to state 11 and parity is 00
    convolutionalCoding["11"]["0"] = ("01", "01")  # in state 11 if 0 is data ->  go to state 01 and parity is 01
    convolutionalCoding["11"]["1"] = ("11", "10")  # in state 11 if 1 is data ->  go to state 11 and parity is 10
    convolutionalCoding["01"]["0"] = ("00", "10")  # in state 01 if 0 is data ->  go to state 00 and parity is 10
    convolutionalCoding["01"]["1"] = ("10", "01")  # in state 01 if 1 is data ->  go to state 10 and parity is 01
    currentState = "00"
    codeWord = list()
    for i in range(len(cipherText) - 1, -1, -1):
        nextState, code = convolutionalCoding[currentState][cipherText[i]]
        currentState = nextState
        codeWord.insert(0, code)
    return "".join(codeWord)
```

در اینجا فرض کنیم کد هافمن خروجی مرحله قبل را به این تابع می دهیم. در اینجا از سمت راست یعنی least significant بیت شروع می کنیم و در ماشین حالت پیش می رویم و خب چون به ازای هر بیت ، در حقیقت دو بیت parity محاسبه می شود پس در حقیقت اگر N بیت ورودی داشته باشیم n*2 بیت خروجی داریم.

```python
def channelDecoding(codeWord):
    Trellis = dict()
    Trellis["00"] = (["0", "00", "00"], ["0", "10", "01"])  # [incoming bit, parity , prev state]
    Trellis["01"] = (["0", "11", "10"], ["0", "01", "11"])
    Trellis["10"] = (["1", "11", "00"], ["1", "01", "01"])
    Trellis["11"] = (["1", "00", "10"], ["1", "10", "11"])
    states = ["00", "01", "10", "11"]

    splitedCodeWord = list()
    for i in range(0, len(codeWord)):  # split codeword in 2 for example 010011 into ['01', '00', '11']
        if i % 2 == 0:
            splitedCodeWord.append(codeWord[i] + codeWord[i + 1])

    pathMetric = [dict() for i in range(0, len(splitedCodeWord) + 1)]
    # pathMetric[timeSlot][state] = value

    pathMetric[0]["00"] = 0  # initial path metric of state 00 in time 0 is 0 and others is infinite
    pathMetric[0]["01"] = 2 ** 32
    pathMetric[0]["10"] = 2 ** 32
    pathMetric[0]["11"] = 2 ** 32
```

در اینجا چهار آرایه در نظر گرفتیم.برای مثال برای سطر اول

Trelles[00]

نشان می دهد که در دو حالت می توانیم به استیت 00 برسیم ، یکی زمانی که بیت صفر مشاهده شود و parity 00 محاسبه شود و در استیت 00 بوده باشیم و یا زمانی که بیت 0 مشاهده شود و در parity 10 محاسبه شود و در استیت 01 بوده باشیم. بقیه حالت به راحتی از روی نمودار به طریق مشابه حساب می گردد.

path metric استیت اولیه در زمان صفر برابر 0 و بقیه حالات بی نهایت است که این را با دو به توان 32 نشان دادیم.

اگر دو بیت دو بیت داده ها رو جدا کنیم، به تعداد زوج های دو بیتی به علاوه یک در حقیقت بازه ی زمانی داریم.

از دو بیت کم ارزش شروع می کنیم و پیش می رویم. در هر بازه ی زمانی می دانیم که از دو طریق می توانیم به آن برسیم. path metric زمان قبل را برای همه استیت ها داریم و کافی است branch metric رو برابر hamming distance بیت های دریافتی و parity موجود در آن یال کنیم.

طبق قاعده ی گفته شده در پروژه path metric هر بازه زمانی رو آپدیت میکنیم.در نهایت از آخر به اول میایم و مینیمم استیت آخر را در نظر می گیریم و می بینیم که از کدام استیت به آنجا آمده ایم و معادل چه بیتی بوده است.

به این ترتیب می توانیم بیت های ورودی را تشخیص دهیم. توجه می کنیم که ممکن است در حالاتی دو مسیر یک مقدار یکسان داشته باشند که در این مواقع به صورت تصادفی یک حالت را انتخاب می کنیم.

ادامه کد در زیر آورده شده است.

```
    for i in range(0, len(splitedCodeWord)):
        for state in states:
            firstState, secondState = Trellis[state]
            pair_bit = len(splitedCodeWord) - i - 1
            firstPathCost = pathMetric[i][firstState[2]] + hammingDistance(splitedCodeWord[pair_bit], firstState[1])
            secondPathCost = pathMetric[i][secondState[2]] + hammingDistance(splitedCodeWord[pair_bit], secondState[1])
            bestPath = min(firstPathCost, secondPathCost)
            pathMetric[i + 1][state] = bestPath
    print(pathMetric)
    path = list()
    currentState = findMinimumState(pathMetric[len(pathMetric) - 1])
```

```
    for i in range(len(splitedCodeWord) - 1, -1, -1):
        firstState, secondState = Trellis[currentState]
        firstPathCost = pathMetric[i][firstState[2]] + hammingDistance(splitedCodeWord[i], firstState[1])
        secondPathCost = pathMetric[i][secondState[2]] + hammingDistance(splitedCodeWord[i], secondState[1])
        if firstPathCost == secondPathCost:
            print("here")
            if random.randint(0, 1) == 0:
                currentState = firstState[2]
                path.append(firstState[0])
            else:
                currentState = secondState[2]
                path.append(secondState[0])
        elif firstPathCost > secondPathCost:
            currentState = secondState[2]
            path.append(secondState[0])
        else:
            currentState = firstState[2]
            path.append(firstState[0])
    return "".join(path)
```

در اینجا برای خروجی دو حالت نویز و بدون نویز را بررسی میکنیم و داده ی کد شده و دیکود شده را برای کانال حساب می کنیم.

کد را با ورودی زیر تست کردم.

```python
if __name__ == '__main__':
    Nodes = generateNodes()
    HuffmanCode = generateHuffman(Nodes)
    decoder = HuffmanDecoder(HuffmanCode)

    plainText = "alirezazarenejad"
    cipherText = sourceCoding(HuffmanCode, plainText)
    print("encode huffman code is: " + cipherText)

    codeWord = channelCoding(cipherText)
    print("result of channelCoding function encoding with convolutional state diagram: " + codeWord)

    decodedWord = channelDecoding(codeWord)
    print("decoded word:" + decodedWord)

    decodedData = destinationDecoding(decoder, decodedWord)
    print("decodedData : ", decodedData)

    print("\n\nwith noise:")
    noisyCodeWord = noise(codeWord)
    print("result of noisy codeword: " + noisyCodeWord)
    decodedWordNoisy = channelDecoding(noisyCodeWord)
    print("decoded word:" + decodedWordNoisy)
    decodedDataNoisy = destinationDecoding(decoder, decodedWordNoisy)
    print("decodedData : ", decodedDataNoisy)
```

```
/usr/bin/python3.6 "/home/alireza/UT/term6/Data Comunication/CA/Channel-Coding-and-Source-Coding/ChannelCoding.py"
encode huffman code is: 11101111010110101100001011001111000101100111100101101010100001011010111011111
channelCoding: 10000101101000011101010001110101001100001011010100111001101000110010110101010011100110100011101101010011101110111100001011010100011101011000010110101000111
[{'00': 0, '01': 4294967296, '10': 4294967296, '11': 4294967296}, {'00': 2, '01': 4294967296, '10': 0, '11': 4294967297}, {'00': 2, '01': 2, '10': 4, '11': 0}, {'00': 2, '0
decoded word:11101111010110101100001011001111000101100111100101101010100001011010111011111
decodedData :  alirezazarenejad
```

می بینیم که خروجی در حالت بدون نویز به درستی دیکود شده است.

```
with noise:
noisy codeword: 10000101101010011101110001110101001110001011110100111001101000110010110101010010010001110110001001100011111011111000010110101010111010110100101100010000
[{'00': 0, '01': 4294967296, '10': 4294967296, '11': 4294967296}, {'00': 1, '01': 4294967296, '10': 1, '11': 4294967297}, {'00': 1, '01': 3, '10': 3, '11': 1}, {'00': 2, '01
decoded word:11101111010110101100001011001111000101100111100101111010100001011010111010000
decodedData :  alirezazarlntirot
```

در حالت با noise همچنان شاهد خطا در رشته دیکود شده هستیم و در بالا این قابل مشاهده است.