Assignment #3

201624548 이재윤

Huffman Code의 동작방식을 이해한다.
Huffman Code를 구현 해 본다.

- 사용 언어
  Python 3

- 사용 라이브러리
  sys (Command Line input을 위해)
  heapq (priority queue)
  os (파일 사이즈 비교를 위해)

● 파일 분석

Node Class

```
# Huffman Code Node Define
class Node :
    def __init__(self, symbol, frequency) :
        self.symbol = symbol
        self.frequency = frequency
        self.left = None
        self.right = None

    def setleft(self, left) :
        self.left = left

    def setright(self, right) :
        self.right = right

    def setleftright(self,left , right) :
        self.setleft(left)
        self.setright(right)
```

Node Class의 내용이다.

내부 데이터는 책에서 소개한 내용대로 구현하였다.

leftnode, rightnode를 세팅하는 함수만 추가되었다.

mk_huffmanTree 함수이다
인코딩할 data (string)을 허프만 트리로 구성해주는 함수이다.
huffmantree와 frequency, symbol이 저장된 리스트를 반환한다.

```
# Make huffman Tree
def mk_huffmanTree(data) :
    freq_dict = dict()
    # make symbol and frequency
    for d in data :
        if d not in freq_dict.keys() :
            freq_dict[d] = 1
        else :
            freq_dict[d] = freq_dict[d] + 1

    # List (freq, symbol)
    freq_symbol = list()
    for sym,freq in list(freq_dict.items()) :
        freq_symbol.append([freq, sym, Node(sym, freq)])

    # Make Priority Queue
    heapq.heapify(freq_symbol)

    for i in range(len(freq_symbol)-1) :
        # pop
        first = heapq.heappop(freq_symbol)
        second = heapq.heappop(freq_symbol)
        # make fre,sym,node
        lfreq, lsym, lnode = tuple(first)
        rfreq, rsym, rnode = tuple(second)
        # make parent_node
        pfreq, psym = lfreq+rfreq, lsym+rsym
        parent_node = Node(psym ,pfreq)
        parent_node.setleftright(lnode, rnode)
        # enqueue
        heapq.heappush(freq_symbol, [pfreq, psym, parent_node])

    result_huffmantree = heapq.heappop(freq_symbol)[2]

    return result_huffmantree, freq_dict
```

mk_Code 함수

mk_Code는 허프만 트리를 Code로 변환해 반환해주는 함수이다.

dfs를 응용하였다.

```
# huffmantree To Code
# type 0 ('q' : 111)
# For encoding
# type 1 (111 : 'q')
# For decoding
def mk_Code(huffmanTree,mktype) :
    code = dict()
    tmpcode = list()
    tmpcode.append([huffmanTree,""])

    while tmpcode :
        nxtnode, nxtcode = tuple(tmpcode.pop())
        # if leftnode exist
        if nxtnode.left != None :
            tmpcode.append([nxtnode.left,nxtcode+'0'])
        # if rightnode exist
        if nxtnode.right != None :
            tmpcode.append([nxtnode.right,nxtcode+'1'])
        # Enter Code
        if nxtnode.left == None and nxtnode.right == None :
            # type 1 (111 : 'q')
            # For decoding
            if mktype == 1 :
                code[nxtcode] = nxtnode.symbol
            # type 0 ('q' : 111)
            # For encoding
            if mktype == 0 :
                code[nxtnode.symbol] = nxtcode

    return code
```

decoding, encoding 함수
디코딩한 결과, 인코딩한 결과를 반환하는 함수이다.

```
# binary code To original code
def decoding(code, data) :
    decoding = ""
    word = ""
    key = code.keys()
    for c in data :
        # if binary code exist
        # plus original code
        if word in key :
            decoding += code[word]
            word = ""
        word += c
    # last word
    decoding += code[word]
    return decoding

def encoding(huffmanTree, data) :
    code = mk_Code(huffmanTree, 0)
    encoding = ""
    # if word exist
    # plus binary code
    for c in data :
        encoding += code[c]
    return encoding
```

main 함수

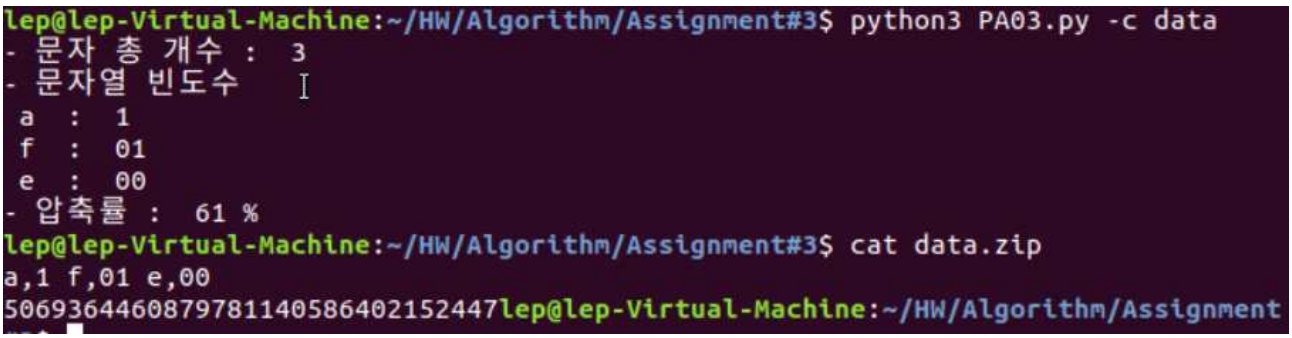
허프만 코드에 대한 모든 일을 이 함수에서 처리한다.

```
def main() :
    sys.argv = ['','-c','data']
    if len(sys.argv) != 3 :
        print('Encoding')
        print("Usage : PA03.py -c [INPUT FILE]")
        print('Decoding')
        print("Usage : PA03.py -d [INPUT FILE]")
        return
    # Encode Type
    code_type = sys.argv[1]
    # Encode
    if code_type == "-c" :
        filename = sys.argv[2].rstrip()
        data = ""
        # delete newline
        with open(filename,'r') as f :
            for l in f.readlines() :
                data += l.replace('\n','')
        huff_tree = mk_huffmanTree(data)
        code = mk_Code(huff_tree,0)
        encode = encoding(huff_tree,data)
        print("- 문자 총 개수 : ", len(code))
        print("- 문자열 빈도수")
        for d in list(code.items()) :
            print(' ' + d[0]," : ",d[1])
        # 바이너리 파일 저장
        with open(filename+'.zip','wb') as f :
            for d in code.items() :
                f.write((d[0] + ',' + str(d[1]) + ' ').encode())
            f.write('\n'.encode())
            f.write(str(int(encode,2)).encode())
        print("- 압축률 : ",round((os.path.getsize(filename+'.zip')/os.path.getsize(filename))
    # Decode
    elif code_type == "-d" :
        filename = sys.argv[2].rstrip()
        data = ""
        with open(filename,'rb') as f :
            for l in f.readlines() :
                data += l.decode()
        ds = data.split()
        tmpcode = ds[:-1]
        data = str(bin(int(ds[-1]))).split('b')[1]
        code = dict()
        for c in tmpcode :
            alpha, alpha_code = tuple(c.split(','))
            code[alpha_code] = alpha
        decode = decoding(code,data)
        print("- 원본 파일 내용")
        print(decode)
        with open(filename.split('.')[0],'w') as f :
            f.write(decode)
    # Else
    else :
        print("Check Your argv plz")
```

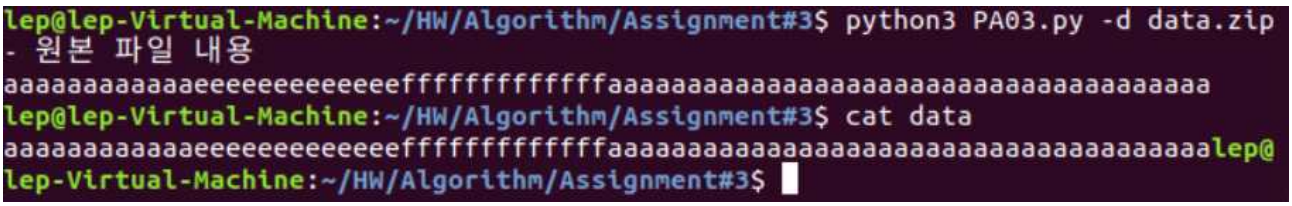
■ 결과

아래는 임의의 파일을 통한 테스트 결과이다.

Encode 결과



Decode 결과



문제없이 실행 잘 되었다.