Report: Text Compression

Xin Sun: 170224545

1. description of the extent of the implementation achieved

(1). count the frequency of the symbols (either characters or words).

(2). create the Node class and build the Huffman tree.

```
text=open('infile.txt','r').read()

# count the frequencs for character base
def freq(text):
    chars = []
    chars_freqs = []
    for character in text:
        if character not in chars:
            char_freq = (character, text.count(character))
            chars_freqs.append(char_freq)
            chars.append(character)
    return chars_freqs

 # count the frequencs for word base
def freq_w(text):
    RE = re.compile(r'[^a-zA-Z]|[a-zA-Z]+')
    content = RE.findall(text)
    chars = []
    chars_freqs = []
    for i in content:
        if i not in chars:
            char_freq = (i, text.count(i))
            chars_freqs.append(char_freq)
            chars.append(i)
    return chars_freqs
```

```
# create the Node class
class Node:
    def __init__(self,freq):
        self.left = None
        self.right = None
        self.freq = freq
        self.father = None
    def isLeft(self):
        return self.father.left == self

def createNodes(freqs):
    return [Node(freq) for freq in freqs]

# create the Huffman tree
def buildHuffmanTree(nodes):
    queue = nodes[:]
    while len(queue) > 1:
        queue.sort(key=lambda item:item.freq)
        temp1 = queue[0]
        temp2 = queue[1]
        queue = queue[2:]
        new_hufftree = Node(temp1.freq + temp2.freq)
        new_hufftree.left = temp1
        new_hufftree.right = temp2
        temp1.father = new_hufftree
        temp2.father = new_hufftree
        queue.append(new_hufftree)
    return queue[0]
```

(3) Huffman encoding and output the model

```
opts, args = getopt.getopt(sys.argv[1:],'s:')
for op,value in opts:
    if op == '-s' and value =='char':
        start1 = time.clock()
        c = freq(text)
        nodes = createNodes([item[1] for item in c])
        root = buildHuffmanTree(nodes)
        codes = huffmanEncoding(nodes,root)
        code_ = {}
        for i in range(len(codes)):
            code_[c[i][0]] = codes[i]
        # output the model file
        output_ = open("infile-symbol-model.pkl", 'wb')
        pickle.dump(code_ , output_)
        elapsed1 = (time.clock() - start1)
        print('it takes',elapsed1,'s to build the symbol model')
```

```
# Huffman encoding
def huffmanEncoding(nodes,root):
    codes = [''] * len(nodes)
    for i in range(len(nodes)):
        node_tmp = nodes[i]
        while node_tmp != root:
            if node_tmp.isLeft():
                codes[i] = '0' + codes[i]
            else:
                codes[i] = '1' + codes[i]
            node_tmp = node_tmp.father
    return codes
```

(4) compress and decompress

```
codearray = array.array('B')
for i in range(0, len(encoded_text), 8):
    c = encoded_text[i:i+8]
    b= int(c, 2)
    codearray.append(b)

#output the compress file
with open('infile.bin', 'wb') as f:
    codearray.tofile(f)
```

```
#decode the txt file
code1 = {value:key for key,value in code.items()}
current_code = ""
for letter in encoded_text:
    current_code += letter
    if code1.get(current_code):
        output.write(code1[current_code])
        current_code = ''

output.close()
```

(5) output three files

2. Evaluation

(1) character-based

(a) The compressed text file is 674 KB.

(b) The symbol model is 1.99 KB.

(c) Time:

```
C:\Users\Administrator\Desktop>python huff-compress.py -s char
it takes 0.6646189080892846 s to build the symbol model
it takes 2.3190698126069984 s to encode the input file given the symbol model

C:\Users\Administrator\Desktop>python huff-decompress.py
it takes 4.8947144460334755 s to decode the compressed file

C:\Users\Administrator\Desktop>
```

(2) word-based

(a) The compressed text file is 464 KB.

(b) The symbol model is 861 KB.

(c) Time

```
C:\Users\Administrator\Desktop>python huff-compress.py -s word
it takes 60.30468523951986 s to build the symbol model
it takes 1.3305834710632851 s to encode the input file given the symbol model

C:\Users\Administrator\Desktop>python huff-decompress.py
it takes 3.113026720861282 s to decode the compressed file

C:\Users\Administrator\Desktop>
```

3.Conclusion

1. As we can see, the word-based method can produce smaller compressed than character-based method, but the symbol model is much bigger. That is because the word-based model has more elements.

2.The word-based method spends more time on building the model, because it takes more time to transfer the text to Regular expression.

3.As a conclusion, if we want to compress the file more effective, we can create the model which contains more elements.