第七周周报

第十九小组 组员：蔡逸文，宋健，李宇星

本周工作

完善了路线规划模块，实现了日记模块

日记模块核心算法为哈夫曼树进行编码和解码，下面是核心代码

哈夫曼树的定义

```
class HuffmanTreeNode:
    def __init__(self, name, weight=0, parent=-1, lchild=-1, rchild=-1):
        self.name = name
        self.weight = weight
        self.parent = parent
        self.lchild = lchild
        self.rchild = rchild
```

哈夫曼树的编码

```python
from HuffmanTree import *

class HuffmanCoder:
    def __init__(self, input_file, output_tree_file, output_code_file):
        self.input_file = input_file
        self.output_tree_file = output_tree_file
        self.output_code_file = output_code_file
        self.char_weights = self.read_file_data()
        self.tree = []
        self.codes = {}
        self.length = 0

    def read_file_data(self):
        with open(self.input_file, "rb") as f:
            bytes = f.read()
        char_weights = {}
        for b in bytes:
            char_weights[b] = char_weights.get(b, 0) + 1
        return char_weights

    def select(self, nodes, n):
        s1 = s2 = -1
        for i in range(n+1):
            if nodes[i].parent == -1:
                if s1 == -1 or nodes[i].weight < nodes[s1].weight:
                    s2 = s1
                    s1 = i
                elif s2 == -1 or nodes[i].weight < nodes[s2].weight:
                    s2 = i
        return s1, s2

    def build_huffman_tree(self):
        num_chars = len(self.char_weights)
        if num_chars == 0:
            return []
        nodes = [HuffmanTreeNode(name) for name in self.char_weights.keys()]

        for i in range(num_chars):
            nodes[i].weight = self.char_weights[nodes[i].name]

        for i in range(num_chars, 2 * num_chars - 1):
            nodes.append(HuffmanTreeNode(name=None))
            s1, s2 = self.select(nodes, i - 1)
            nodes[i].lchild = s1
            nodes[i].rchild = s2
            nodes[s1].parent = nodes[s2].parent = i
            nodes[i].weight = nodes[s1].weight + nodes[s2].weight

        self.tree = nodes

    def huffman_coding(self):
        for i in range(len(self.tree)):
            code = ""
            cur = i
            parent = self.tree[cur].parent
            while parent != -1:
                if self.tree[parent].lchild == cur:
                    code = "0" + code
                else:
                    code = "1" + code
                cur = parent
                parent = self.tree[cur].parent
            if self.tree[i].name is not None:
                self.codes[self.tree[i].name] = code

        with open(self.input_file, "rb") as fr, open(self.output_code_file, "wb+") as fw:
            data = ""
            while True:
                c = fr.read(1)
                if not c:
                    break
                int_value = int.from_bytes(c, byteorder='big')
                data += self.codes.get(int_value, "")
            self.length = len(data)
            if r := self.length % 8:
                data += "0"*(8-r)
            b = bytes([int(data[i:i+8], 2) for i in range(0, len(data), 8)])
            fw.write(b)

    def export_bit_length(self):
            with open(self.output_tree_file, "w+", encoding='utf-8') as f:
                f.write(f"{self.length}\n")
                for node in self.tree:
                    f.write(f"{node.name} {node.weight} {node.parent} {node.lchild} {node.rchil
    def Huffman_coding_main(self):
        self.build_huffman_tree()
        self.huffman_coding()
        self.export_bit_length()

if __name__ == "__main__":
    input_file = "server\\src\\Journal\\Journal.txt"
    output_tree_file = "server\\src\\Journal\\hfmTree.txt"
    output_code_file = "server\\src\\Journal\\hfmzip.zip"

    huffman_coder = HuffmanCoder(input_file, output_tree_file, output_code_file)
    huffman_coder.Huffman_coding_main()
```

哈夫曼树的解码

```python
from HuffmanTree import *

class HuffmanDecoder:
    def __init__(self, encoded_text, zip_path, decoder_path):
        self.encoded_text = encoded_text
        self.zip_path = zip_path
        self.decoder_path = decoder_path
        self.tree , self.i, self.length = self.reshape_huffman_tree()

    def reshape_huffman_tree(self):
        nodes = []
        i = 0
        with open(self.encoded_text, 'r') as f:
            length = int(f.readline())
            while True:
                b = f.readline().split()
                if not b:
                    break
                if b[0] != 'None':
                    nodes.append(HuffmanTreeNode(name=int(b[0])))
                    nodes[i].weight = int(b[1])
                    nodes[i].parent = int(b[2])
                    nodes[i].lchild = int(b[3])
                    nodes[i].rchild = int(b[4])
                else:
                    nodes.append(HuffmanTreeNode(name = None))
                    nodes[i].weight = int(b[1])
                    nodes[i].parent = int(b[2])
                    nodes[i].lchild = int(b[3])
                    nodes[i].rchild = int(b[4])
                del b
                i += 1
        return nodes, i, length

    def Huffman_Decoding(self):
        with open(self.zip_path, 'rb') as fr:
            data = fr.read()
            binstr = "".join(["{:08b}".format(char) for char in data])
            binstr = binstr[:self.length]
            i = self.i-1
            b = []
            for c in binstr:
                if c == '0' or c == '1':
                    if c == '0' and self.tree[i].lchild != -1:
                        i = self.tree[i].lchild
                    if c == '1' and self.tree[i].rchild != -1:
                        i = self.tree[i].rchild
                    if self.tree[i].lchild == -1 and self.tree[i].rchild == -1:
                        b.append(self.tree[i].name)
                        i = self.i-1
            b = bytes(b)
            with open(self.decoder_path, 'wb+') as fw:
                fw.write(b)

    def Huffman_Decoding_main(self):
        self.Huffman_Decoding()

if __name__ == '__main__':
    encoded_text = "server\\src\\Journal\\hfmTree.txt"
    zip_path = "server\\src\\Journal\\hfmzip.zip"
    decoder_path = "server\\src\\Journal\\hfmDecoder.txt"

    decoder = HuffmanDecoder(encoded_text, zip_path, decoder_path)
    decoder.Huffman_Decoding_main()
```

日记模块部分源代码

```python
from ..DataType import Journal,Comment

class JournalManagementSystem:
    def __init__(self):
        self.journals = []

    def add_journal(self, journal: Journal):
        self.journals.append(journal)

    def add_comment_to_journal(self, journal_name: str, comment: Comment):
        for journal in self.journals:
            if journal.journalName == journal_name:
                journal.journalComment.add(comment)
                break

    def get_journals(self):
        return [(journal.journalName, journal.journalDate) for journal in self.journals]

    def get_comments_for_journal(self, journal_name: str):
        for journal in self.journals:
            if journal.journalName == journal_name:
                return [(comment.commentOwner, comment.commentText) for comment in journal.jour
        return []
```

本周总结

本周工作的进度在预料之内，既没有超出也没有减少

下周安排

实现推荐模块和各个模块之间的协作