자료구조

과제 6. 연결리스트

과목명 : 자료구조
담당교수 : 김승태
소속학과 : 응용통계학과
학번 : 20193011
이름 : 유승희

1. 연결 리스트 - 사전 만들기

1.1 사전 만들기

데이터 불러오기

제공된 사전 파일 randdict.TXT 파일을 읽기모드로 한 줄씩 불러와 랜덤 사전 dictionary 리스트에 저장해주었다. 단어들을 불러올 때 편의를 위해 줄바꿈은 제거하고 ':'를 기준으로 나누어 저장했다.

- 코드

```
dictionary = []
with open('randdict.TXT', 'r') as file:
    for text in file:
        dictionary.append(text.strip('\n').split(' : '))
```

연결리스트 클래스

단어 사전을 만들 때 연결리스트 자료형을 이용하기 위해 연결리스트 구조체를 정의해주었다. 수업자료를 이용하여 만들었다.

<연결리스트>
노드 연결리스트 : 각 노드는 데이터(영어단어와 뜻)를 포함한 data와 다음 노드의 주소를 가리키는 next 속성변수를 갖는다.

연결리스트 클래스 : 노드를 감싸는 전체 연결리스트를 클래스로 만든다. 이 클래스를 실행하면 헤드를 포함한 아직 노드가 없는 연결리스트를 생성한다. 따라서 헤드는 None을 가리킨다. 또한 연결리스트의 길이를 알 수 있게 len 속성 변수를 초기값 0을 갖도록 추가해주었다. 연결리스트의 클래스는 다음과 같은 기능(함수)들로 구성되어있다.

- getNode(self, pos) : 노드의 위치를 나타내는 가상 번호인 위치값(pos)를 입력하면 몇 번째 노드인지 찾아 해당하는 노드의 주소값을 반환한다.
- insert(self, pos, elem) : 연결리스트에 새로운 노드를 삽입하는 함수이다. 넣을 위치와 데이터를 입력하면 해당 위치에 노드를 넣어주고 연결리스트의 길이를 1 증가한다.
- delete(self, pos) : 삭제할 노드의 위치를 입력하면 해당 노드를 삭제해주고 연결리스트의 길이를 1 줄인다.
- display() : 해당 연결리스트의 각 데이터가 어떻게 연결되어있는지 화면에 출력한다.

-코드

```
class Node :
    def __init__(self, data, next=None):
        self.data = data
        self.next = next

class LinkedList:
    def __init__(self):
        self.head = None
        self.len = 0

    def getNode(self, pos):
        if pos < 0 : return None
        node = self.head
        while pos > 0 and node != None :
            node = node.next
            pos -= 1
        return node

    def insert(self, pos, elem):
        before = self.getNode(pos-1)
        self.len += 1
        if before == None:
            self.head = Node(elem, self.head)
        else:
            node=Node(elem, before.next)
            before.next = node

    def delete(self, pos):
        before = self.getNode(pos-1)
        removed = before.next
        before.next = before.next.next
        self.len -=1
        del removed

    def display(self):
        node = self.head.next
        while node is not None:
            print(node.data, end="->\n")
            node = node.next
        print()
```

데이터를 정렬하면서 연결리스트에 저장하는 함수

파일의 데이터를 저장한 dictionary 리스트에서 단어 하나를 가져와 정렬된 순서에 단어를 넣기 위한 함수를 정의했다.

함수는 정렬된 단어를 저장할 연결리스트 linkedlist와 삽입할 단어 text를 입력받는다.

연결리스트의 헤드가 가리키는 노드 즉 첫 번째 노드부터 시작하여 노드의 데이터의 영어단어와 text를 문자열 비교한다. (node의 data에 영단어와 영단어 뜻이 모두 포함되어 있다. data[0] : 영단어, data[1] : 영단어 뜻)
비교를 하다가 text가 들어가야할 위치를 나타내기 위해 pos변수를 이용한다.
랜덤 순서로 들어가있는 단어 리스트엔 대문자와 소문자가 들어가있기 때문에 비교시엔 두 영단어를 소문자로 변환한 값으로 비교후 삽입시엔 원래 데이터를 추가한다.

추가할 영단어(text[0])와 연결리스트의 영단어(data[0])를 비교하여 data[0]가 작으면 (예: data[0] : 'abc' < text[0] : 'abd') 그 다음 노드로 넘어가 다시 비교하고 pos값도 1 증가한다.
data[0]이 크면 해당 위치에 text를 삽입한다
text가 마지막에 들어가야 한다면 pos는 마지막 노드를 가리키기 때문에 그 다음 위치에 삽입한다.

-코드

```
def insert_sorted(linkedlist, text):
    node = linkedlist.head
    pos = 0
    while 1:
        data = node.data
        transform_data_key = data[0].lower()
        transform_text_key = text[0].lower()

        if transform_data_key > transform_text_key:
            linkedlist.insert(pos,text)
            break

        if node.next == None:
            linkedlist.insert(pos+1,text)
            break

        node = node.next
        pos += 1

    return linkedlist
```

사전 만들기 (개선 후 방법)

알파벳 별로 (a,b,c, … ,x,y,z) 로 시작하는 별도의 연결리스트를 생성한다. 개별 연결리스트를 딕셔너리 형태로 담아 저장한다.
또한 추가할 영단어의 앞글자에 해당하는 사전 연결리스트에 정렬된 순서로 삽입하여 연결리스트 사전을 생성한다.

```
linkedlists = {}
for alphabet in string.ascii_lowercase:
    linkedlists[alphabet] = LinkedList()
```

위와 같이 생성된 linkedlists는 다음과 같이 개별 연결리스트를 갖는다.

```
linkedlists
```

```
{'a': <__main__.LinkedList at 0x1e1979c7dd8>,
 'b': <__main__.LinkedList at 0x1e1979c79b0>,
 'c': <__main__.LinkedList at 0x1e197a527f0>,
 'd': <__main__.LinkedList at 0x1e197a52f60>,
 'e': <__main__.LinkedList at 0x1e197a52b70>,
 'f': <__main__.LinkedList at 0x1e197a527b8>,
 'g': <__main__.LinkedList at 0x1e197a52358>,
 'h': <__main__.LinkedList at 0x1e197a52cc0>,
 'i': <__main__.LinkedList at 0x1e197a52ac8>,
 'j': <__main__.LinkedList at 0x1e197a52128>,
 'k': <__main__.LinkedList at 0x1e197a52400>,
 'l': <__main__.LinkedList at 0x1e197a526a0>,
 'm': <__main__.LinkedList at 0x1e197a52e48>,
 'n': <__main__.LinkedList at 0x1e197a52550>,
 'o': <__main__.LinkedList at 0x1e197a52208>,
 'p': <__main__.LinkedList at 0x1e197a52470>,
 'q': <__main__.LinkedList at 0x1e197a52dd8>,
 'r': <__main__.LinkedList at 0x1e197a52cf8>,
 's': <__main__.LinkedList at 0x1e197a52940>,
 't': <__main__.LinkedList at 0x1e197a52eb8>,
 'u': <__main__.LinkedList at 0x1e197a52860>,
 'v': <__main__.LinkedList at 0x1e197a52588>,
 'w': <__main__.LinkedList at 0x1e197a52748>,
 'x': <__main__.LinkedList at 0x1e197a52518>,
 'y': <__main__.LinkedList at 0x1e197a52b38>,
 'z': <__main__.LinkedList at 0x1e197a52278>}
```

각 알파벳 별 연결리스트에 다음과 같이 단어를 추가한다. (보고서 작성을 위해 5,000개 단어만 사용했고 결과는 'a','b','c' 연결리스트의 앞뒤 5개씩만 첨부했다.)

```
for text in dictionary[:5000]:
    linkedlist = linkedlists[text[0].lower()[0]]

    if linkedlist.head == None:
        linkedlist.insert(0, text)

    else:
        linkedlist = insert_sorted(linkedlist, text)
```

<결과>

```
linkedlists['a'].display()
```

```
['aback', 'ad.뒤로']->
['abaft', 'ad.(배의)고물에']->
['abashment', 'n.수치']->
['abbacy', 'n.대 수도원장(abbot)의 직(권)']->
['abdominous', 'a.올챙이배의']->
['axle', 'n.굴대']->
['axolotl', 'n.액솔로틀']->
['ayah', 'n.하녀']->
['az', '=azo-']->
['azobenzene', 'n.아조벤젠']->
```

```
linkedlists['b'].display()
```

```
['backbite', 'vt.험담하다']->
['backdate', 'vt.실제보다 날짜를 거슬러 올라가게 하다(흔히 to)']->
['backdoor', 'a.뒷문의']->
['backer', 'n.후원자']->
['backing', 'n.안대기']->
['buttermilk', 'n.버터밀크(버터를 빼낸 뒤의 우유)']->
['butterwort', 'n.벌레잡이제비꽃']->
['buzzer', 'n.윙윙거리는 벌레']->
['byplot', 'n.(소설 희곡 등의)곁줄거리']->
['Byzantine', '비잔티움 의  복잡한']->
```

```
linkedlists['c'].display()
```

```
['caboodle', 'n.무리']->
['caboose', 'n.(화물열차의)승무원차']->
['cachinnate', 'vi.너털웃음을 웃다']->
['cachou', 'n.구중향정']->
['caddish', 'adj.비신사적인']->
['cypripedia', 'n.개불알꽃류']->
['cystiform', 'adj.포 모양의']->
['cytoplasm', 'n.세포질']->
['czarevitch', 'n.제정 러시아의 황태자']->
['Czech', '체코 사람 체코슬로바키아 사람']->
```

1.2 사전 기능 추가하기

사전에 단어 검색하기 기능과 단어 추가하기 기능을 만들어주었다.

검색할 단어를 문자열로 입력받아 word 변수에 저장한다. word의 앞글자에 해당하는 연결리스트에서 word가 있는지 비교한다. 동일한 영단어(data[0])가 존재하면 그 뜻(data[1]) 출력한다. 다음과 같은 탐색을 마지막 노드까지 진행하고 마지막 노드까지 동일한 단어(word)가 없다면 그 word의 의미를 입력받아 meaning 변수에 저장한다.
입력받은 meaning이 공백이면 반복문을 빠져나와 단어를 추가하지 않도록 한다.
입력받은 단어(word)와 뜻(meaning)을 리스트로 묶어 text변수에 저장하여 해당하는 알파벳 연결리스트에 insert_sorted() 함수를 이용하여 추가한다.
추가한 영단어와 뜻 그리고 해당 연결리스트의 길이(len 속성변수 이용)를 출력해준다.

- 코드

```
word = input('단어를 검색하세요. : ')

linkedlist = linkedlists[word.lower()[0]]
node = linkedlist.head
while 1:
    data = node.data

    if data[0] == word :
        print(data[1])
        break

    if node.next != None :
        node = node.next
    else:
        meaning = input("찾을 수 없는 단어입니다. 뜻을 추가하세요. (추가하지 않으려면 공백) : ")
        if meaning == '':
            break
        text = [word, meaning]
        linkedlist = insert_sorted(linkedlist, text)
        print(f'{word} {meaning} 가 추가되었습니다. (총 {linkedlist.len}개 단어)')
        break
```

- 결과

```
단어를 검색하세요. :  cytoplasm
n.세포질

단어를 검색하세요. :  applejuice
찾을 수 없는 단어입니다. 뜻을 추가하세요. (추가하지 않으려면 공백) :  n. 사과주스
applejuice n. 사과주스 가 추가되었습니다. (총 363개 단어)

단어를 검색하세요. :  applejuice
n. 사과주스
```

2. 연결 리스트 - 사전 개선하기

2.1 사전 개선하기 전

연결리스트로 사전 만들기

위의 방법과 달리 알파벳 별로 연결리스트를 생성하지 않고 정렬된 사전 1개의 연결리스트를 생성했다. 개선된 사전과 비교하기 위해 소요 시간을 측정해주었다. 개선된 사전의 경우와 동일하게 dictionary의 5000개 단어만 사용했다.

```
start_time = time.time()
for text in dictionary[:5000]:
    if linkedlist.head == None:
        linkedlist.insert(0, text)

    else:
        linkedlist = insert_sorted(linkedlist, text)
print(round(time.time()-start_time,6), 'sec')
```

```
13.528631 sec
```

단어 검색하기

개선하기 전 사전을 이용해 단어 검색을 할 때 소요되는 시간을 측정했다. 비교하기 위해 5000개 단어중 가장 마지막에 오는 단어인 'zymosis'를 검색했다.

```
word = input('단어를 검색하세요. : ')
start_time = time.time()
node = linkedlist.head
while 1:
    data = node.data

    if data[0] == word :
        print(data[1])
        break

    if node.next != None :
        node = node.next
    else:
        meaning = input("찾을 수 없는 단어입니다. 뜻을 추가하세요. (추가하지 않으려면 공백) : ")
        if meaning == '':
            break
        text = [word, meaning]
        linkedlist = insert_sorted(linkedlist, text)
        print(f'{word} {meaning} 가 추가되었습니다. (총 {linkedlist.len}개 단어)')
        break
print(round(time.time()-start_time,6), 'sec')
```

```
단어를 검색하세요. : zymosis
n.발효
0.007996 sec
```

개선하기 전 연결리스트로 사전을 만드는데 소요되는 시간은 13.527631초 이고 가장 마지막 단어를 검색하는데 0.007996초가 소요되었다.

2.2 사전 개선 후

연결리스트로 사전 만들기

알파벳 별로 연결리스트를 생성했다. 개선 전 사전과 비교하기 위해 소요 시간을 측정해주었다. 동일하게 dictionary의 5000개 단어만 사용했다.

```
start_time = time.time()
for text in dictionary[:5000]:
    linkedlist = linkedlists[text[0].lower()[0]]

    if linkedlist.head == None:
        linkedlist.insert(0, text)

    else:
        linkedlist = insert_sorted(linkedlist, text)
print(round(time.time()-start_time,6), 'sec')
```

```
0.693234 sec
```

단어 검색하기

개선하기 전 사전에 검색한 동일한 단어를 입력한다.

```
word = input('단어를 검색하세요. : ')
start_time = time.time()
linkedlist = linkedlists[word.lower()[0]]
node = linkedlist.head
while 1:
    data = node.data

    if data[0] == word :
        print(data[1])
        break

    if node.next != None :
        node = node.next
    else:
        meaning = input("찾을 수 없는 단어입니다. 뜻을 추가하세요. (추가하지 않으려면 공백) : ")
        if meaning == '':
            break
        text = [word, meaning]
        linkedlist = insert_sorted(linkedlist, text)
        print(f'{word} {meaning} 가 추가되었습니다. (총 {linkedlist.len}개 단어)')
        break
print(round(time.time()-start_time,6), 'sec')
```

```
단어를 검색하세요. :  zymosis
n.발효
0.0 sec
```

개선된 연결리스트로 사전을 만드는데 소요되는 시간은 0.693234초 이고 정렬된 5000개 단어중 가장 마지막 단어를 검색하는데 0.0초가 소요되었다. 개선 전과 성능이 향상 되었음을 볼 수 있고 데이터가 더 클 때 더 효율적일 것이다.
각 알파벳별 연결리스트를 양방향 연결리스트로 구성하면 보다 효율적인 연결리스트 사전을 생성할 수 있다.