Assignment03

List, Stack 연습문제

AI 4조 20241254 이유빈

Chapter 05. List

#1 연결 리스트 구현 시 더미 헤드를 사용할 때의 장점은 예외처리를 하지 않아도 된다. 단점은 더미 헤드를 위한 공간이 필요하기 때문에 메모리 공간이 낭비된다.

```
#2
def contain(self, x) bool:
    if self.index(x)==-12345:
        return False
    else:
        return True
```

```
#4
def print_interval(self, i:int,j:int):
    if i>=0 and i<=self.__num_items-1 and j>=0 and j<=self.__num_items:
        curr=self.get_node(i)
        print(curr.item,end=" ")
        while curr!=self.get_node(j):
            curr=curr.next
            if curr==self.__tail.next:
                continue
            print(curr.item,end=" ")
        print()
    else:
        print("index",i,"or",j,"out of bound in print_interval()")
```

```
#3
def prinInterval(self, i:int,j:int):
    if i>=0 and j<=self.__num_items-1 and j>=0 and j<=self.__num_items:
        if i<=j:
            curr=self.__get_mode(i)
            print(curr.item, end=" ")
            for _in range(j-i):
                curr=curr.next
                print(curr.item, end=" ")
            print()
        else:
            curr=self.__get_mode(i)
            while curr!=None:
                print(curr.item, end=" ")
                curr=curr.next
            curr=self.__head.next
            print(curr.item,end=" ")
            for _ in range(j):
                curr=curr.next
                print(curr.item,end=" ")
            print()
    else:
        print("index",i,"or",j,"out of bound in print_interval()")
```

```
#5-1
def num_items(self):
    cnt = 0
    curr = self.__head.next
    while curr != None:
        cnt += 1
        curr = curr.next
    return cnt

#5-2
def num_items(self, node):
        if node == None:
            return 0
    return 1 + num_items(node.next)
```

```
#6
def pop(self, i: int, k: int):
    if k < 1:
        return None
    if i>= 0 and i <= self.__num_items - 1:
        temp_list = []
        curr = self.__get_node(i)
        for _ in range(num):
            temp_list.append(curr.item)
            curr = curr.next
            if curr == None:
                break
        return tuple(temp_list)
    return None
```

```
#7
def add(self, x) -> None:
    if self.is_empty():
        self.append(x)
    else:
        index = 0
        next = self.__head.next
        while next != self.__head:
            if next.x> x:
                break
            else:
                next = next.next
                index += 1
        self.insert(index, x)
```

```
#8
# node1이 node2보다 리스트의 앞에 있는 경우
same_list = False
curr = node1
while curr != None:
        if curr == node2:
        print("node1과 node2이 같은 연결 리스트에 속합니다.)
        same_list = True
        break
    else:
        curr = curr.next

# node2가 node1보다 리스트의 앞에 있는 경우
if not same_list:
  curr = node2
  while curr != None:
      if curr == node1:
          print("node1과 node2이 같은 연결 리스트에 속합니다.)
          same_list = True
          break
      else:
          curr = curr.next
# node1과 node2이 같은 리스트에 속하지 않는 경우
if not same_list:
        print("node1과 node2이 같은 연결 리스트에 속하지 않습니다.)
```

```
#9
def last_index_of(self, x) -> int:
    curr = self.__head
    last_index = -2
    for index in range(self.__num_items):
        curr = curr.next
        if curr.x == x:
            last_index = index
    return last_index
```

```
#10
def last_index_of(self, x) -> int:
    curr = self.__tail
    if curr.x == x:
        return self.__num_items - 1
    curr = curr.next
    last_index = -2
    for index in range(self.__num_items - 1):
        curr = curr.next
        if curr.x == x:
            last_index = index
    return last_index
```

Chapter 06. Stack

#1 스택에 15를 저장한다. - 스택에 25를 저장한다. - 스택에서 25를 제거한다. - 스택에서 15를 제거한다. - 스택에 15에서 25를 더한 40을 저장한다. - 스택에 10을 저장한다. - 스택에 2를 저장한다. - 스택에서 2를 제거한다. - 스택에서 10을 제거한다. - 스택에 10에서 2를 곱한 20을 저장한다. - 스택에서 20을 제거한다. - 스택에서 40을 제거한다. - 연산 결과 40에서 20을 뺀 20을 얻는다.

```
#2
class ListStack:
    def __init__(self):
        self.__stack = []

    def push(self, x):
        self.__stack.append(x)

    def pop(self):
        if self._is_empty():
            print("No element in stack")
            return None
        else:
            return self.__stack.pop(0)

    def top(self):
        if self._is_empty():
            print("No element in stack")
            return None
        return self.__stack[0]

    def is_empty(self) -> bool:
        return not bool(self.__stack)

    def pop_all(self):
        self.__stack.clear()

    def print_stack(self):
        print("Stack from top:", end = " ")
        for i in range(len(self.__stack)):
            print(self.__stack[i], end = " ")
```

```
#3
def backwards(s: String) -> bool:
    stack = ListStack()
    index = 0
    while s[index] != '$':
        stack.push(s[index])
        index += 1
    index += 1
    while index < len(s):
        if stack.pop() != s[index]:
            return False
        else:
            index += 1
    return True
```

```
#4
def copy(a: 'LinkedStack', b: 'LinkedStack'):
        b.pop_all()
    temp_linked_stack = LinkedStack()
    while not a.is_empty():
        temp_linked_stack.push(a.pop())
    while temp_linked_stack.is_empty():
        a.push(temp_linked_stack.top())
        b.push(temp_linked_stack.pop())
```

```
#5
def paren_balance(s: String) -> bool:
    stack = ListStack()
    for index in range(len(s)):
        if s[index] == '(':
            stack.push('(')
        elif s[index] == ')':
            if stack.is_empty():
                return False
            else:
                stack.pop()
    if stack.is_empty():
        return True
    else:
        return False
```

#6 다소 복잡해질 것이다. string을 탐색할 때 왼쪽 괄호는 stack에 넣고, 오른쪽 괄호가 있으면 stack의 pop을 통해 반환된 왼쪽 괄호가 오른쪽 괄호와 같은 종류의 괄호지만 확인하면 된다.

#7 최초 호출은 factorial(100)이 호출되어 스택에 저장된다. 이후 factorial(100)은 factorial(99)를 호출하고, factorial(99)는 factorial(98)을 호출하는 식으로 진행되어진다.

이러한 과정에서 함수 호출이 끝나지 않고 남아있는 상태로 스택에 쌓이게 되며, 최종적으로 factorial(1)이 호출되었을 때는 factorial(1)부터 factorial (100)까지 총 100개의 호출 정보가 저장되어있다. 따라서 최대 100개의 factorial() 호출이 스택에 저장된다고 할 수 있다.

#8 fib(50) 호풀 시 스택에는 먼저 fib(50)이 저장된다. 그 후 fib(49), fib(48), ...,fib(2)가 차례대로 호출되어 스택에 쌓이게 된다. 이 때 fib(1)과 fib(2)가 기저 조건으로 바로 반환되어지기 때문에 fib(2)까지 호출된다. 따라서 fib(50)의 수행 과정 중 최대 49개의 fib() 호출이 동시에 스택에 저장된다고 할 수 있다.