Lab 8

COMP9021, Session 2, 2016

1 ☞ Linked lists

Entend the module linked_list.py which is part of the material of the 8th lecture into a module extended_linked_list.py to implement the extra method remove_duplicates(), that keeps only the first occurrence of any value. As for the 7th quiz, this should be done without creating new nodes and without using Python lists.

Here is a possible interaction.

```
$ python3
...
>>> from extended_linked_list import *
>>> LL =  ExtendedLinkedList([1, 2, 3])
>>> LL.remove_duplicates()
>>> LL.print()
>>> [1, 2, 3]
>>> LL =  ExtendedLinkedList([1, 1, 1, 2, 1, 2, 1, 2, 3, 3, 2, 1])
>>> LL.remove_duplicates()
>>> LL.print()
>>> [1, 2, 3]
```

2 ☞ Doubly linked lists

Modify the module linked_list.py which is part of the material of the 8th lecture into a module doubly_linked_list.py, to process lists consisting of nodes with a reference to both next and previous nodes, so with the class Node defined as follows.

```
class Node:
    def __init__(self, value = None):
        self.value = value
        self.next_node = None
        self.previous_node = None
```

3 Using linked lists to represent polynomials (optional)

Write a program polynomial.py that implements a class `Polynomial`. An object of this class is built from a string that represents a polynomial, that is, a sum or difference of monomials.

- The leading monomial can be either an integer, or an integer followed by `x`, or an integer followed by `x^` followed by a nonnegative integer.
- The other monomials can be either a nonnegative integer, or a nonnegative integer followed by `x`, or a nonnegative integer followed by `x^` followed by a nonnegative integer.

Spaces can be inserted anywhere in the string.

A monomial is defined by the following class:

```
class Monomial:
    def __init__(self, coefficient = 0, degree = 0):
        self.coefficient = coefficient
        self.degree = degree
        self.next_monomial = None
```

A polynomial is a linked list of monomials, ordered from those of higher degree to those of lower degree. An implementation of the `__str__()` method allows one to print out a polynomial.

Here is a possible interaction.

```
$ python3
...
>>> from polynomial import *
>>> Polynomial('-0')
Incorrect input
>>> Polynomial('+0')
Incorrect input
>>> Polynomial('0x^-1')
Incorrect input
>>> Polynomial('2x + +2')
Incorrect input
>>> Polynomial('2x + -2')
Incorrect input
>>> Polynomial('2x - +2')
Incorrect input
>>> poly_0 = Polynomial('0')
>>> print(poly_0)
0
```

```
>>> poly_0 = Polynomial('0x')
>>> print(poly_0)
0
>>> poly_0 = Polynomial('0x^0')
>>> print(poly_0)
0
>>> poly_0 = Polynomial('0x^5')
>>> print(poly_0)
0
>>> poly_1 = Polynomial('x')
>>> print(poly_1)
x
>>> poly_1 = Polynomial('1x')
>>> print(poly_1)
x
>>> poly_1 = Polynomial('1x^1')
>>> print(poly_1)
x
>>> poly_2 = Polynomial('2')
>>> print(poly_2)
2
>>> poly_2 = Polynomial('2x^0')
>>> print(poly_2)
2
>>> poly_3 = Polynomial('1 + 2-3 +10')
>>> print(poly_3)
10
>>> poly_4 = Polynomial('x + x - 2x -3x^1 + 3x')
>>> print(poly_4)
0
>>> poly_5 = Polynomial('x + 2 + x - x -3x^1 + 3x + 5x^0')
>>> print(poly_5)
x + 7
>>> poly_6 = Polynomial('-2x + 7x^3 +x   - 0  + 2 -x^3 + x^23 - 12x^8 + 45 x ^ 6 -x^47')
>>> print(poly_6)
-x^47 + x^23 - 12x^8 + 45x^6 + 6x^3 - x + 2
```

4 Markov chains (optional)

Write a program markov_chain.py that prompts the user to input two positive integers n and N, and outputs N words generated by a Markov chain where a dictionary file, named dictionary.txt, stored in the working directory, determines the probability that an n-gram (that is, a sequence of n letters) be followed by this or that character (including the "end-of-word" character). More precisely, assume that $n = 3$. Then a word $c_1 \dots c_k$ is generated as follows.

- c_1 is generated following the probability that, according to dictionary.txt, a word starts with c_1.
- c_2 is generated following the probability that, according to dictionary.txt, a word that starts with c_1 starts with c_1c_2; in case c_2 is the end of word marker then $k = 1$.
- c_3 is generated following the probability that, according to dictionary.txt, a word that starts with c_1c_2 starts with $c_1c_2c_3$; in case c_3 is the end of word marker then $k = 2$.
- c_4 is generated following the probability that, according to dictionary.txt, a word that contains $c_1c_2c_3$ contains $c_1c_2c_3c_4$; in case c_4 is the end of word marker then $k = 3$.
- c_5 is generated following the probability that, according to dictionary.txt, a word that contains $c_2c_3c_4$ contains $c_2c_3c_4c_5$; in case c_5 is the end of word marker then $k = 4$.
- c_6 is generated following the probability that, according to dictionary.txt, a word that contains $c_3c_4c_5$ contains $c_3c_4c_5c_6$; in case c_6 is the end of word marker then $k = 5$.
- ...

The program should indicate whether the word that has been generated has been invented (because it does not occur in dictionary.txt), or whether it has been rediscovered (because it does occur in dictionary.txt). Here is a possible interaction.

```
$ python3 markov_chains_for_word_generation.py
What n to use to let an n-gram determine the next character? 2
How many words do you want to generate? 10
Rediscovered ADS
Invented ENTRAMER
Invented LER
Invented EQUILIZED
Invented CIATTLY
Invented GRECOND
Rediscovered ASS
Invented WINCOT
Invented PEENIAR
Rediscovered ANTS
```

```
$ python3 markov_chains_for_word_generation.py
What n to use to let an n-gram determine the next character? 3
How many words do you want to generate? 10
Invented ROYAN
Rediscovered THING
Invented AGGREEABLE
Rediscovered RECEPTION
Invented LISHED
Invented CONTERMING
Invented TUSCUSTIVE
Invented INISM
Invented SWORTHUST
Invented BENTHANGE
$ python3 markov_chains_for_word_generation.py
What n to use to let an n-gram determine the next character? 4
How many words do you want to generate? 10
Invented REFORMEDITOR
Invented DIFFICE
Invented SEMITTERING
Invented INAPPERS
Invented PROPOLDVILLED
Invented KINGBIRDIED
Rediscovered SUBSCRIBED
Invented SCHED
Invented DEGRADIC
Rediscovered MILLION
$ python3 markov_chains_for_word_generation.py
What n to use to let an n-gram determine the next character? 5
How many words do you want to generate? 10
Rediscovered APPEARS
Rediscovered LOWS
Rediscovered SPORTS
Invented CROWDERPUFF
Invented BIRTHRIGHTNESS
Invented BREAKFASTERFUL
Rediscovered DREAMY
Rediscovered JACOB
Rediscovered BRUNHILDE
Invented REORGANISM
```