CSCI-8450

2018 Spring

Yuchen Wang

You will find Chapter 4b posted at:

<http://www.nltk.org/book/ch04.html>

Let *TestText* be the text posted at URL[*https://www.cs.utexas.edu/~vl/notes/dijkstra.html*](https://www.cs.utexas.edu/~vl/notes/dijkstra.html)

To get text out of HTML, use a Python library called BeautifulSoup, available from <http://www.crummy.com/software/BeautifulSoup/> as prescribed in Dealing with HTML subsection in Chapter 3

Complete the following of the exercises:

1. exercise 0 (0 is a dummy name in this case). Write a function to sort a list of WordNet synsets for proximity to a given synset in accordance with their shortest\_path\_distance(). Report what your function produces given the synsets minke\_whale.n.01, orca.n.01, novel.n.01, and tortoise.n.01, sorted with respect to right\_whale.n.01.

code:

**def** synset\_sort(synsets, key):  
 result = [(s.shortest\_path\_distance(key), s) **for** s **in** synsets]  
 result = sorted(result, key=**lambda** x:x[0])  
 **return** [s **for** (\_,s) **in** result]

**def** exercise0():  
 synsets = [**"minke\_whale.n.01"**, **"orca.n.01"**, **"novel.n.01"**, **"tortoise.n.01"**]  
 synsets\_key = **'right\_whale.n.01'** whales = [wn.synset(s) **for** s **in** synsets]  
 print(**"sorted synsets : "**, synset\_sort(whales, wn.synset(synsets\_key)))

Result:

Exercise 0

sorted synsets : [Synset('lesser\_rorqual.n.01'), Synset('killer\_whale.n.01'), Synset('tortoise.n.01'), Synset('novel.n.01')]

1. exercise 26.

◑ The Catalan numbers arise in many applications of combinatorial mathematics, including the counting of parse trees ([6](http://www.nltk.org/book/ch08.html#sec-grammar-development)). The series can be defined as follows: C0 = 1, and Cn+1 = Σ0..n (CiCn-i).

1. Write a recursive function to compute *n*th Catalan number Cn.
2. Now write another function that does this computation using dynamic programming.
3. Use the timeit module to compare the performance of these functions as *n* increases.

Report your findings for (c) for the 10th and 16th Catalan numbers. Template contains the parts utilizing timeit functionality required in (c).

Code:

**def** recursive\_catalan(n):  
  
 **if** n <= 1:  
 **return** 1  
 res = 0  
 **for** i **in** range(n):  
 res += recursive\_catalan(i)\*recursive\_catalan(n-i-1)  
 **return** res  
  
  
**def** dynamic\_catalan(n):  
 result = 0  
 **if** n <= 1:  
 result = 1  
 **for** i **in** range(2, n):  
 **for** j **in** range(i):  
 result += dynamic\_catalan(j) \* dynamic\_catalan(i - j)  
 **return** result  
  
**def** exercise26():  
 *# Example of timer usage:  
 # print(Timer(lambda: recursive\_catalan(n)).timeit(1))* print(**'catalan number results'**)  
 **for** i **in** range(0, 16):  
 print (**"Catalan"**, i ,recursive\_catalan(i))  
  
 print(**"Performance comparison"**)  
 print(**"n = 5"**)  
 n = 5  
 print(**'recursive\_catalan()'**, Timer(**lambda**: recursive\_catalan(n)).timeit(1))  
 print(**'dynamic\_catalan()'**, Timer(**lambda**: dynamic\_catalan(n)).timeit(1))  
 print(**"n = 10"**)  
 n = 10  
 print(**'recursive\_catalan()'**, Timer(**lambda**: recursive\_catalan(n)).timeit(1))  
 print(**'dynamic\_catalan()'**, Timer(**lambda**: dynamic\_catalan(n)).timeit(1))  
 print(**"n = 15"**)  
 n = 15  
 print(**'recursive\_catalan()'**, Timer(**lambda**: recursive\_catalan(n)).timeit(1))  
 print(**'dynamic\_catalan()'**, Timer(**lambda**: dynamic\_catalan(n)).timeit(1))

Result:

Exercise 26

catalan number results

Catalan 0 1

Catalan 1 1

Catalan 2 2

Catalan 3 5

Catalan 4 14

Catalan 5 42

Catalan 6 132

Catalan 7 429

Catalan 8 1430

Catalan 9 4862

Catalan 10 16796

Catalan 11 58786

Catalan 12 208012

Catalan 13 742900

Catalan 14 2674440

Catalan 15 9694845

Performance comparison

n = 5

recursive\_catalan() 5.589338102901848e-05

dynamic\_catalan() 2.8586691060643077e-05

n = 10

recursive\_catalan() 0.00800939350134912

dynamic\_catalan() 0.006456325509397768

n = 15

recursive\_catalan() 2.075885718089146

dynamic\_catalan() 1.4736725108672095

1. exercise 32.

★ Develop a simple extractive summarization tool, that prints the sentences of a document which contain the highest total word frequency. Use FreqDist() to count word frequencies, and use sum to sum the frequencies of the words in each sentence. Rank the sentences according to their score. Finally, print the n highest-scoring sentences in document order. Carefully review the design of your program, especially your approach to this double sorting. Make sure the program is written as clearly as possible.

Test your function on *TestText* with n=5, n=10, n=15, n=20, n=30. Does any of this number produce satisfactory/natural results? Include the output of your function for n=7.

**Code:**

**def** summarization(text, n):  
 tokenized\_words = nltk.word\_tokenize(text)  
 tokenized\_sents = nltk.sent\_tokenize(text)  
  
 word\_list = [word **for** word **in** tokenized\_words]  
 sent\_list = [sent **for** sent **in** tokenized\_sents]  
  
 freq\_words = nltk.FreqDist(tokenized\_words)  
  
  
 res = []  
 **for** sents **in** sent\_list:  
 sum = 0  
 sent\_tokenized\_words = nltk.word\_tokenize(sents)  
 **for** word **in** sent\_tokenized\_words:  
 sum += freq\_words[word]  
 res.append((sents, sum))  
 result = sorted(res, key = **lambda** x: x[1], reverse=**True**)  
  
 **return** result[:n]  
  
  
**def** exercise32():  
  
  
 highest1 = summarization(TestText, 7)  
 print(**"Sentence with the highest total word frequency (n=7):"**)  
 **for** (i, j) **in** highest1:  
 print(i, end=**"\n"**)  
 print()

Result:

Exercise 32

Sentence with the highest total word frequency (n=7):

Well, of what the overhead

projector did to teaching, you are, I'm afraid, a better judge than I.

I remember how, with the advent of terminals, interactive debugging was

supposed to solve all our programming problems, and how, with the advent of

colour screens, "algorithm animation" was supposed to do the same.

In the dozen years that I am now here, I have learnt to appreciate

UT as a place where that link is in full vigour, and my first remark to you,

both as advice and as request, is to remember and cherish your university

as a place dedicated to the practice of intellectual freedom, where the

combination of exercising your brain and opening your mouth is encouraged.

Because the lifting of that siege was a turning point in our struggle for

freedom, William the Silent wanted to reward the city's population for its

bravery and endurance, and gave them the choice between two prizes: either

a full year freedom from tax paying, or the right to a University, and the

citizens chose the latter.

There seems to be one thing that, independently of the height of the fence

that surrounds the campus, any graduating student can take with him into

the world outside, and that is the healthy scepticism that goes with a

well-kept immunity for hype, for slogans, for fads and for fashions.

So far, so good, but a number of decades later, the

need was felt to raise the institute's status, and the Dutch government,

which did not know or did not care what it was doing, agreed to raise that

business school officially to the level of a university.

In the 70s, the creed was diversification, in the 80s, the

gospel said to concentrate on your core business, and in the 90s the

world-wide credo seems to be the intellectual cleansing of the high-tech

industries.

In a sentimental moment, you may long for the lost innocence

and bliss of ignorance, but that ignorance is lost forever: you are now

Bachelors and you will have to carry the burden of your college education

every waking hour of the rest of your lives, nay, even in your sleep you

will be pressed to have dreams full of responsibility.