Lecture 3: Problem Solving

COSC 526: Introduction to Data Mining

THE UNIVERSITY OF TENNESSEE
KNOXVILLE

A problem can have multiple solutions, some better than others

A better solutions?

- When do we define a solution is better than another?
  - (More) accurate
  - (Better) performance

About our assignment today

- Consolidate expertise in Github, Python, and Jupyter Notebook
- Learn to strategize on problems' solutions
  - Write a simple solution first
  - Use git commit to save significant steps
  - Use git branch for new solution efforts
  - **Take performance (executing times) into account as the data analyzed grows**
  - Learn that problems come with constraints and assumptions

Problem 1

(From ProjectEuler Problem 1)

If we list the natural numbers below 10 that are multiples of 3 or 5, we get: 3, 5, 6, and 9. The sum of these multiples is 23.

Write a function that finds the sum of the multiples of p or q below N.

Assumptions and Constraints (I)

- Assume that $1 \leq p < q < N$ and that each of these values are integers. Your code should be able to
  - handle values of N up to at least 100,000,000 (larger and larger data)
  - terminate in less than 1 second (constant execution time!)

Assumptions and Constraints (I)

- Things to keep in mind
  - There are two general approaches to this problem
    - the naïve (slower) approach
    - The more mathematical (faster) approach involving the inclusion-exclusion principle.
  - To meet the performance constraints, you will have to implement the fast approach.
  - Start with the naïve solution to observe how the time increase as N increases

Assumptions and Constraints (II)

- There are different approaching to measure execution times (wall-clock times).
- The approach we propose is one of them. Use what you prefer, as long as you measure **wall-clock time**.

Your Notebook

```
In [ ]: import time

        # Define the function to take three arguments.
        def sumOfMultiples(p, q, n):



            return 0

        # Print the output of your function for p=3, q=5, n=10.
        start_time = time.time()
        print(sumOfMultiples(3, 5, 10))
        print("Duration: %s seconds" % (time.time() - start_time))

        # Print the output of your function for p=3, q=5, n=10000.
        start_time = time.time()
        print(sumOfMultiples(3, 5, 10000))
        print("Duration: %s seconds" % (time.time() - start_time))

        # Print the output of your function for p=3, q=5, n=10000000.
        start_time = time.time()
        print(sumOfMultiples(3, 5, 100000000))
        print("Duration: %s seconds" % (time.time() - start_time))
```

Your Notebook

```
In [ ]: import time

        # Define the function to take three arguments.
        def sumOfMultiples(p, q, n):



            return 0

        # Print the output of your function for p=3, q=5, n=10.
        start_time = time.time()
        print(sumOfMultiples(3, 5, 10))
        print("Duration: %s seconds" % (time.time() - start_time))

        # Print the output of your function for p=3, q=5, n=10000.
        start_time = time.time()
        print(sumOfMultiples(3, 5, 10000))
        print("Duration: %s seconds" % (time.time() - start_time))

        # Print the output of your function for p=3, q=5, n=10000000.
        start_time = time.time()
        print(sumOfMultiples(3, 5, 100000000))
        print("Duration: %s seconds" % (time.time() - start_time))
```

Expected Output: Please note that the execution times r

```
23
Duration: 0.0012371540069580078 seconds
23331668
Duration: 0.00026607513427734375 seconds
2333333316666668
Duration: 0.00013208389282226562 seconds
```

Problem 2

- Manipulate a list of names
  - Use the *csv module* to import a list of names
  - Score names in a list based on name “worth” and alphabetical order

Problem 2

(From ProjectEuler Problem 22.)

- The file p022_names.txt contains one line with over 5000 comma-separated names, each in all-capital letters and enclosed in quotes.
- Import the names and sort them into alphabetical order.
- Working out the alphabetical value for each name, multiply this value by its alphabetical position in the list to obtain a name score.
- Example:
  - When the list is sorted into alphabetical order, COLIN, which is worth 3 + 15 + 12 + 9 + 14 = 53, is the 938th name in the list.
  - COLIN would obtain a score of 938 * 53 = 49714.
- What is the total of the scores for all the names in the file?

Your Notebook

In [ ]:

```
# Rather than manually stripping and slicing the data as we did in the previous assigment,
# let's use the csv module.
import csv

with open('p022_names.txt') as namefile:
    # Complete the line below by specifying the appropriate arguments.
    # HINT: ref [1]
    name_reader = csv.reader('''TODO: add arguments''')
    names = next(name_reader)

# Sort the list of names.
# HINT: ref [2]


# Compute the alphabetical value for each name, with 'A' -> 1, 'B' -> 2, ..., 'Z' -> 26.
# HINT: ref [3]


# Multiply each name value by name's position in the ordered list, where the first name is in position 1.
# HINT: ref [4]


# Print the sum of all the name scores.
```

Your Notebook

In [ ]:

```
# Rather than manually stripping and slicing the data as we did in the previous assigment,
# let's use the csv module.
import csv

with open('p022_names.txt') as namefile:
    # Complete the line below by specifying the appropriate arguments.
    # HINT: ref [1]
    name_reader = csv.reader('''TODO: add arguments''')
    names = next(name_reader)

# Sort the list of names.
# HINT: ref [2]


# Compute the alphabetical value for each name, with 'A' -> 1, 'B' -> 2, ..., 'Z' -> 26.
# HINT: ref [3]


# Multiply each name value by name's position in the ordered list, where the first name is in position 1.
# HINT: ref [4]
```

References:

- 1: csv.reader
- 2: sort
  Note: we can use the function list.sort() without any added arguments, but the sort function has additional capabilities worth exploring. See HowTo/Sorting for more details.
- 3: ord
- 4: enumerate

Problem 3

- Find the smallest TPH number bigger than *n*
  - Triangular (T), Pentagonal (P), and Hexagonal (H) numbers are generated by given formulae
  - Write the code to find the next triangle number that is also pentagonal and hexagonal

Problem 3

- (From ProjectEuler Problem 45.)
- Triangular, Pentagonal, and Hexagonal numbers are generated by the following formulae:

Polygonal	*formula for nth term*	*sequence of terms*
Triangular	$T_n = \frac{n(n+1)}{2}$	1, 3, 6, 10, 15, ...
Pentagonal	$P_n = \frac{n(3n-1)}{2}$	1, 5, 12, 22, 35, ...
Hexagonal	$H_n = (2n-1)\,n$	1, 6, 15, 28, 45, ...

Problem 3

- The number 1 is triangular, pentagonal, and hexagonal (TPH).
- Less obviously, $40755 = T_{285} = P_{165} = H_{143}$ is also TPH
  - 40755 is the smallest TPH number bigger than 1.
- Write a function to find the smallest TPH number bigger than *n*.
- Use your function to find the smallest TPH bigger than 40755.

Your Notebook

```
# Now write a function that returns the least TPH number greater than n.
def nextTPH(n):


    return 0

# Print the output of your function for n=1 and again for n=40754.
print(nextTPH(1))
print(nextTPH(40754))

# Print the output of your function for n=40755.
print(nextTPH(40755))
```

Your Notebook

```
# Now write a function that returns the least TPH number greater than n.
def nextTPH(n):


    return 0

# Print the output of your function for n=1 and again for n=40754.
print(nextTPH(1))
print(nextTPH(40754))

# Print the output of your function for n=40755.
print(nextTPH(40755))
```

Polygonal	*formula for nth term*	*sequence of terms*
Triangular	$T_n = \frac{n(n+1)}{2}$	1, 3, 6, 10, 15, ...
Pentagonal	$P_n = \frac{n(3n-1)}{2}$	1, 5, 12, 22, 35, ...
Hexagonal	$H_n = (2n - 1)$	1, 6, 15, 28, 45, ...

Is there any relationship between the numbers in T and H?

Your Notebook

```python
# You will probably want to define functions that compute the n-th polygonal number
# for various polygons.
def getTri(x):
    return x * (x + 1) // 2
def getPent(x):
    return (x * (3*x - 1)) // 2
def getHex(x):
    return x * (2*x - 1)
```

Polygonal	*formula for nth term*	*sequence of terms*
Triangular	$T_n = \frac{n(n+1)}{2}$	1, 3, 6, 10, 15, ...
Pentagonal	$P_n = \frac{n(3n-1)}{2}$	1, 5, 12, 22, 35, ...
Hexagonal	$H_n = (2n-1)$	1, 6, 15, 28, 45, ...

Your Notebook

```
# (The following is necessary for an efficient solution, but not for a brute-force solution.)
# The quadratic formula is useful for computing a least polygonal number greater than n.
# For example, to find the least Hexagonal number greater than 30, solve the equation
# 30 = x(2x-1), which in general form is 0 = 2x^2 - x - 30. If we write the function below
# to compute the larger of the two solutions to 0=ax^2 + bx + c, then solve_quad(2, -1, -30)
# will return 4.1310435... so the next Hexagonal number is getHex(5) = 45.
# HINT: ref [2]
def solve_quad(a, b, c):
    d = b**2 - 4*a*c
    return (-1*b + d**0.5) / (2*a)
```

find the least Hexagonal number greater than 30:

- solve the equation 30 = x(2x-1)
  - general form is 0 = 2x^2 - x - 30
  - solve_quad(2, -1, -30) → return 4.1310435
    BUT Hexagonal numbers are integer --> 5
  - next Hexagonal number is getHex(5) = 45

Polygonal	*formula for nth term*	*sequence of terms*
Triangular	$T_n = \frac{n(n+1)}{2}$	1, 3, 6, 10, 15, ...
Pentagonal	$P_n = \frac{n(3n-1)}{2}$	1, 5, 12, 22, 35, ...
Hexagonal	$H_n = (2n-1)$	1, 6, 15, 28, 45, ...

Your Notebook

Find the least Hexagonal number greater than 30:

solve the equation 30 = x(2x-1)

general form is 0 = 2x^2 - x - 30

solve_quad(2, -1, -30) → return 4.1310435
BUT Hexagonal numbers are integer --> 5

next Hexagonal number is getHex(5) = 45

Find the least Pentagonal number greater than 30:

solve the equation 30 = x(3x-1) /2

general form is 0 = 1.5 x^2 – 0.5 x - 30

solve_quad(1.5, -0.5, -30) → return 4.641907
BUT Pentagonal numbers are integer --> 5

next Hexagonal number is getPent(5) = 35

Polygonal	formula for nth term	sequence of terms
Triangular	$T_n = \frac{n(n+1)}{2}$	1, 3, 6, 10, 15, ...
Pentagonal	$P_n = \frac{n(3n-1)}{2}$	1, 5, 12, 22, 35, ...
Hexagonal	$H_n = (2n - 1)$	1, 6, 15, 28, 45, ...

Things to consider

- Your choice of data structure can have a significant impact on runtime.
- Think about which operations you are doing the most and choose a data structure which minimizes the average time for this particular operation.
- Python has many built-in data structures
  - The most common data structures are lists, dictionaries, and sets, but Python also has heaps and queues.

Problem 4 (Optional)

From ProjectEuler Problem 87.)

- The smallest number expressible as the sum of a prime square, a prime cube, and a prime fourth power is $2^8=2^2+2^3+2^4$.
- There are exactly four numbers below 50 that can be expressed in such a way:

$$
\begin{aligned}
2^8 &= 2^2 + 2^3 + 2^4 \\
3^3 &= 3^2 + 2^3 + 2^4 \\
4^9 &= 5^2 + 2^3 + 2^4 \\
4^7 &= 2^2 + 3^3 + 2^4
\end{aligned}
$$

Problem 4

- Write code to determine the number of positive integers smaller than N that can be written as the sum of a prime square, a prime cube, and a prime fourth power.
- Your code must accept a single command line parameter
  - this time your Jupyter notebook accepts a user input N
- Your code must print a single integer
  - Example: given the input equal to 50, the output is 4

Assumptions and constraints

- For testing, you may assume that N is a positive integer and that N ≤ 50,000,000 (or larger).
- You should be able to compute the answer when N = 50,000, 000 and terminate in approximately 1 minutes.

Things to consider

- If you are having a hard time getting started, then break down the problem into smaller manageable pieces.
- Almost certainly you’ll need to have a list of primes handy.
  - Can you generate a list of primes?
  - How big do your prime numbers need to be?

Let’s star our breakout sessions ….

- Once you have completed your assignment, push it to your personal repository
- Deadline: Friday Feb 15, 2021 before 8AM ET